ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΙΓΑΙΟΥ
ΤΜΗΜΑ ΜΑΘΗΜΑΤΙΚΩΝ

ΜΙΑ ΕΦΑΡΜΟΓΗ ΓΙΑ ΤΗΝ ΕΠΕΞΕΡΓΑΣΙΑ ΜΑΘΗΜΑΤΙΚΩΝ ΕΚΦΡΑΣΕΩΝ ΣΤΗ ΓΛΩΣΣΑ JAVA

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

ΚΑΤΕΡΙΝΑ ΚΟΥΚΟΥΣΟΥΛΑ

Επιβλέπων: Ανδρέας Παπασαλούρος
Λέκτορας

Καρλόβασι, Ιούλιος 2009

ΣΥΜΒΟΥΛΕΥΤΙΚΗ ΕΠΙΤΡΟΠΗ

Ανδρέας Παπασαλούρος, Λέκτορας, Επιβλέπων
Κωνσταντίνος Χουσιάδας, Επίκουρος καθηγητής
Παναγιώτης Νάστου, Λέκτορας

Η εργασία αυτή εκπονήθηκε το ακαδημαϊκό έτος 2008-2009 στο Τμήμα Μαθηματικών της σχολής Θετικών Επιστημών του Πανεπιστήμιου Αιγαίου.

ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΙΓΑΙΟΥ
ΤΜΗΜΑ ΜΑΘΗΜΑΤΙΚΩΝ
24 Ιουλίου 2009

ΕΥΧΑΡΙΣΤΙΕΣ :

Ευχαριστώ τον επιβλέποντα της πτυχιακής εργασίας κ. **Α. Παπασαλούρο** για την υποστήριξη, την καθοδήγηση και τις συμβουλές του για να φτάσει αυτή η εργασία στο τέλος της.

Επίσης, ευχαριστώ τον κ. **Π. Νάστου** και τον κ. **Κ. Χουσιάδα** οι οποίοι είχαν την καλή θέληση να συμμετάσχουν στην εξεταστική επιτροπή.

Τέλος, ευχαριστώ την *οικογένειά* μου και τους *φίλους* μου για την ηθική υποστήριξή τους.

Πίνακας Περιεχομένων :

ΕΙΣΑΓΩΓΗ

Το αντικείμενο της παρούσας εργασίας είναι η κατασκευή μίας εφαρμογής η οποία θα απεικονίζει μαθηματικές εκφράσεις. Η εφαρμογή υλοποιήθηκε στη γλώσσα προγραμματισμού Java. Για την απεικόνιση των μαθηματικών εκφράσεων χρησιμοποιήθηκε η εργαλειοθήκη Swing της Java και ιδιαίτερα τα συστατικά κειμένου του Swing, η αρχιτεκτονική και η χρήση των οποίων είναι αρκετά πολύπλοκη, όπως θα φανεί παρακάτω. Αναπτύχθηκε κώδικας ο οποίος βασίστηκε στο πρότυπο σχεδίασης Model View Controller (MVC) για το οποίο γίνεται πλήρης αναφορά στο 3ο Κεφάλαιο. Δημιουργήθηκαν *όψεις,* δηλαδή κλάσεις απεικόνισης, για την απεικόνιση των μαθηματικών εκφράσεων. Η τρέχουσα έκδοση της εφαρμογής υποστηρίζει την απεικόνιση ενός υποσυνόλου της γλώσσας των μαθηματικών εκφράσεων. Λόγω της αρχιτεκτονικής που ακολουθήθηκε, είναι εφικτή η επέκταση της εφαρμογής σε ένα πλήρη διορθωτή μαθηματικών εκφράσεων (editor) χωρίς μεγάλες τροποποιήσεις.

Στο συγκεκριμένο έγγραφο περιγράφεται η χρήση των συστατικών δημιουργίας κειμένου της βιβλιοθήκης γραφικών Swing, της γλώσσας Java. Ιδιαίτερα περιγράφονται τα συστατικά κειμένου που χρησιμοποιήθηκαν για την υλοποίηση της εφαρμογής. Παρατίθενται βασικές έννοιες της γλώσσας, χωρίς περαιτέρω εμβάθυνση. Το κείμενο εστιάζει στην αρχιτεκτονική των συστατικών κειμένου. Στο Παράρτημα παρατίθεται ο κώδικας της εφαρμογής ο οποίος αποτελεί εφαρμογή των όσων παρουσιάζονται. Βέβαια και μέσα στα Κεφάλαια δίνονται παραδείγματα, τα οποία όμως είναι συνοπτικά για να αναδεικνύουν κάτι συγκεκριμένο κάθε φορά.

Παρακάτω παρατίθενται τέσσερα κεφάλαια που οργανώνονται με τον εξής τρόπο :

Στο *Κεφάλαιο 1* της παρούσας εργασίας, γίνεται μία εισαγωγή στη Java, στη χρησιμότητά της και στις ποικίλες εφαρμογές της. Αφενός, δίνονται ορισμοί για τα ιδιαίτερα στοιχεία της Java, όπως του αντικειμένου, και αφετέρου, αναπτύσσονται αυτές οι ιδιαιτερότητες.

Στο *Κεφάλαιο 2* παρουσισάζεται η εργαλειοθήκη γραφικών Swing η οποία για τα μέχρι τώρα δεδομένα, είναι ό,τι πιο χρήσιμο υπάρχει για την υλοποίηση μίας γραφικής διασύνδεσης χρήστη και τα συστατικά της. Στο συγκεκριμένο Κεφάλαιο βρίσκονται οι λειτουργίες των περισσότερων συστατικών, μεταξύ άλλων και των συστατικών κειμένου, βάσει των οποίων εργαστήκαμε για την ανάπτυξη του κώδικα της εφαρμογής.

Στο *Κεφάλαιο 3* αναπτύσσονται η αρχιτεκτονική MVC και η αρχιτεκτονική των συστατικών κειμένου. Δίνονται αναλυτικές πληροφορίες για το κάθε ένα και φαίνεται και ο συνδυασμός των δύο, καθότι αναδεικνύεται ότι το έγγραφο ενός συστατικού κειμένου, είναι το μοντέλο του MVC και η εργαλειοθήκη σύνταξης ο ελεγκτής. Επίσης δίνονται αναλυτικές πληροφορίες για την κατασκευή των όψεων των συστατικών κειμένου, εξετάζονται κάποιες κλάσεις και η χρήση τους, η ιεραρχία τους, οι μέθοδοί τους και ούτω καθεξής.

Στο *Κεφάλαιο 4*, παρουσιάζεται η σχεδίαση και υλοποίηση της εφαρμογής. Συνδυάζονται οι πληροφορίες των προηγούμενων κεφαλαίων και περιγράφεται ο τρόπος με τον οποίο χρησιμοποιήθηκαν για την ανάπτυξη του κώδικα της εφαρμογής.

Στο *Παράρτημα* δίνεται ο πλήρης κώδικας της εφαρμογής.

ΚΕΦΑΛΑΙΟ 1.

ΑΝΤΙΚΕΙΜΕΝΟΣΤΡΕΦΗΣ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΣ ΚΑΙ Η ΓΛΩΣΣΑ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΥ JAVA.

Λίγη Ιστορία

Η γλώσσα Java αναπτύχθηκε από την Sun Microsystems το 1991, σαν μέρος του έργου Green, κατά το οποίο μία ερευνητική ομάδα εργαζόταν για να αναπτύξει λογισμικό για έλεγχο ηλεκτρονικών συσκευών ευρείας κατανάλωσης. Οι ερευνητές ήλπιζαν να αναπτύξουν μία γλώσσα προγραμματισμού που θα λειτουργούσε τις έξυπνες συσκευές στο μέλλον – διαλογικές τηλεοράσεις, διαλογικές τοστιέρες κ.λπ. Οι ερευνητές της Sun ήθελαν επίσης αυτές οι συσκευές να επικοινωνούν μεταξύ τους, έτσι ώστε η μία συσκευή να μπορούσε να δώσει πληροφορίες σε μία άλλη.

Για να προχωρήσουν την έρευνά τους, οι ερευνητές της ομάδας Green ανέπτυξαν μία πρωτότυπη συσκευή με όνομα *Star7*, ενός μηχανισμού τύπου *remote control* (έλεγχος εξ αποστάσεως) που μπορούσε να επικοινωνεί με άλλες συσκευές ίδιου τύπου. Η αρχική ιδέα ήταν να αναπτύξουν το λειτουργικό σύστημα Star7 σε C++, την δημοφιλή γλώσσα προγραμματισμού, που αναπτύχθηκε από τον Bjarne Stroustrup. Αλλά όμως, το μέλος της ομάδας James Gosling όντας μη ικανοποιημένος από τα αποτελέσματα που προσέφερε η C++, εργάστηκε με το να κατασκεάσει μία νέα γλώσσα προγραμματισμού που θα μπορούσε να χειριστεί καλύτερα το Star7. Η γλώσσα αυτή ονομάστηκε *Oak*. Η Sun αργότερα ανακάλυψε ότι το όνομα Oak χρησιμοποιόταν ήδη οπότε κατά τη διάρκεια μιας εκ των πολλών συναντήσεών τους σε κάποιο τοπικό καφενείο αποφάσισαν να μετονομάσουν το νέο τους δημιούργημα σε *Java* που εκτός των άλλων ήταν όνομα αγαπητού καφέ για τους δημιουργούς της (Java στην αγγλική γλώσσα είναι το φυτό που βγάζει τον καφέ).

Επειδή σχεδιάστηκε για οικιακές συσκευές, η Java έπρεπε να είναι μικρή, αποδοτική και εύκολα μεταφέρσιμη σε πολλές οικιακές συσκευές. Έπρεπε επίσης να είναι αξιόπιστη. Οι άνθρωποι έχουν μάθει να ζουν με τις περιστασιακές καταρρεύσεις των συστημάτων τους. Δεν είναι όμως εύκολο να έχουν μία τοστιέρα που καίει συνεχώς τα τοστ τους!

Αν και η Java αρχικά χρησιμοποιήθηκε ως εργαλείο ανάπτυξης εφαρμογών για οικιακές συσκευές, αποδείχθηκε ότι είναι εξίσου καλή για τον Παγκόσμιο Ιστό (World Wide Web).

- Η Java είναι μικρή – Κάνει τα προγράμματα να φορτώνονται γρηγορότερα σε μία σελίδα.
- Η Java είναι ασφαλής – Απαγορεύει στους εισβολείς (hackers) να γράφουν προγράμματα που να μπορούν να χαλάσουν τα συστήματα των χρηστών.
- Η Java είναι μεταφέρσιμη – Μπορεί να εκτελείται στα Windows, σε Macintosh, και σε άλλες πλατφόρμες χωρίς τροποποίηση.

Επίσης, η Java μπορεί να χρησιμοποιηθεί σαν μία γλώσσα προγραμματισμού γενικής χρήσης για ανάπτυξη λογισμικού, που μπορεί να εκτελείται σε διαφορετικές πλατφόρμες.

Για να επιδείξει τις δυνατότητες της Java και να διασώσει το ερευνητικό της έργο, δημιουργήθηκε ένας περιηγητής ιστού (web browser) το 1994, ο οποιός μπορούσε να εκτελέσει μικροεφαρμογές Java. Ο περιηγητής επέδειξε δύο πράγματα για την Java τί προσέφερε στον Παγκόσμιο Ιστό και τί είδους προγράμματα μπορούσε να δημιουργήσει. Οι προγραμματιστές Patrik Naughton και Jonathan Payne χρησιμοποίησαν την Java για να δημιουργήσουν τον περιηγητή που αρχικά ονομάστηκε WebRunner, αλλά μετονομάστηκε σε HotJava.

Αν και η Java και ο περιηγητής HotJava έτυχαν ιδιαίτερης προσοχής στην κοινότητα του διαδικτύου, η γλώσσα απογειώθηκε πραγματικά όταν η Netscape έγινε η πρώτη εταιρεία που

έδωσε άδεια χρήσης στην γλώσσα, τον Αύγουστο του 1995. Ο διευθυντής της Netscape και εκατομμυριούχος `Marc Andreesen` ήταν ένας από τους πρώτους, έξω από την Sun, που είδε τις δυνατότητες της Java και την υιοθέτησε στην σύνοδο JavaOne τον Μαίο του 1996. "Η Java δίνει τεράστιες δυνατότητες σε όλους μας", είπε στους συμμετέχοντες. Μετά από την πρώτη δημόσια έκδοση της Java, η Sun πολλαπλασίασε τις προσπάθειες ανάπτυξης της Java και προσέθεσε εκατοντάδες υπαλλήλους για να συνεχίσει να αναπτύσσει τη γλώσσα.

Τί Είναι και τί μας Προσφέρει η Java

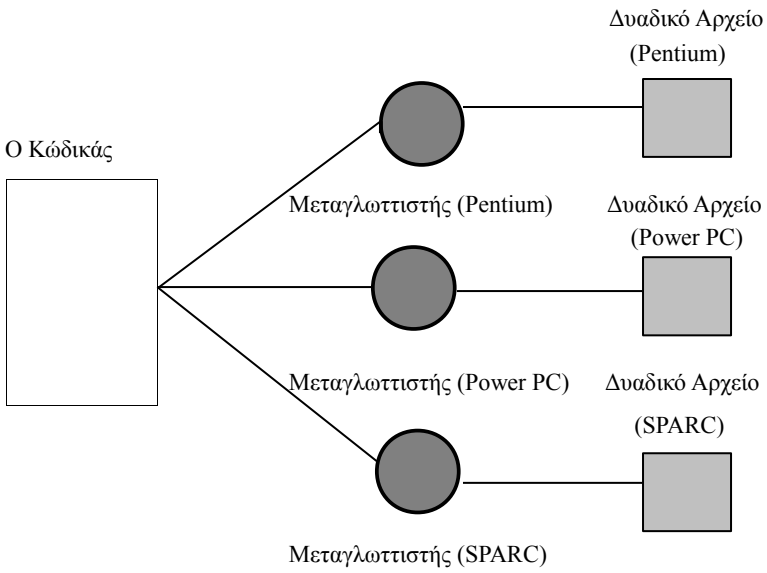
Η Java Είναι Ανεξάρτητη από Πλατφόρμα

Ανεξαρτησία από πλατφόρμα – η δυνατότητα το ίδιο πρόγραμμα να εκτελείται σε διαφορετικές πλατφόρμες και λειτουργικά συστήματα – είναι ένα από τα σημαντικότερα πλεονεκτήματα που έχει η Java σε σχέση με τις άλλες γλώσσες προγραμματισμού.

Όταν μεταγλωττίζουμε ένα πρόγραμμα γραμμένο σε C ή στις περισσότερες άλλες γλώσσες, ο μεταγλωττιστής (compiler) μεταφράζει το αρχείο πηγαίου κώδικα σε *κώδικα μηχανής* – εντολές που είναι συγκεκριμένες για τον επεξεργαστή του υπολογιστή μας. Αν μεταγλωττίσουμε τον κώδικά μας σε ένα σύστημα με επεξεργαστή Intel, το προκύπτον πρόγραμμα θα εκτελείται σε άλλα συστήματα με Intel, αλλά δεν θα εργαστεί σε Macintosh ή σε άλλο μηχάνημα. Αν θελήσουμε να χρησιμοποιήσουμε το ίδιο πρόγραμμα σε μία άλλη πλατφόρμα, πρέπει να μεταφέρουμε τον πηγαίο κώδικα στην νέα πλατφόρμα και να τον μεταγλωττίσουμε εκ νέου για να παράγουμε κώδικα μηχανής για το νέο σύστημα. Σε πολλές περιπτώσεις, θα απαιτηθούν αλλαγές στον πηγαίο κώδικα πριν να μεταγλωττιστεί στο νέο μηχάνημα, λόγω των διαφορών στους επεξεργαστές και σε άλλους παράγοντες.

Η Εικόνα 1 δείχνει το αποτέλεσμα ενός συστήματος ανεξαρτήτου από πλατφόρμα Πολλαπλά εκτελέσιμα προγράμματα πρέπει να παραχθούν για πολλαπλά συστήματα.

ΕΙΚΟΝΑ 1.
Παραδοσιακά μεταγλωττισμένα προγράμματα



Τα προγράμματα Java επιτυγχάνουν αυτή την ανεξαρτησία μέσω της χρήσης μίας εικονικής μηχανής – κάτι σαν ένα υπολογιστή μέσα στον υπολογιστή. Η *εικονική μηχανή* (Java virtual machine) παίρνει μεταγλωττισμένα προγράμματα Java και μετατρέπει τις εντολές τους σε εντολές

που μπορεί να χειριστεί ένα λειτουργικό σύστημα. Το ίδιο μεταγλωττισμένο πρόγραμμα που υπάρχει σε μία μορφή που καλείται `bytecode` μπορεί να εκτελείται σε οποιαδήποτε άλλη πλατφόρμα και λειτουργικό σύστημα που διαθέτει μία εικονική μηχανή της Java.

Bytecode είναι μία μορφή δυαδικού κώδικα μηχανής ο οποίος είναι εκτελέσιμος από την εικονική μηχανή της Java.

Η εικονική μηχανή είναι επίσης γνωστή ως *διερμηνευτής Java (Java interpreter)* ή *Java runtime.* Οι προγραμματιστές της Java δεν χρειάζεται να δημιουργήσουν διαφορετικές εκδόσεις των προγραμμάτων τους για κάθε πλατφόρμα που χρησιμοποιούν, επειδή η εικονική μηχανή χειρίζεται την μετάφραση.

Η Java είναι επίσης ανεξάρτητη πλατφόρμας σε επίπεδο πηγαίου κώδικα. Τα προγράμματα της Java αποθηκεύονται σαν αρχεία κειμένου πριν να μεταγλωττιστούν και τα αρχεία αυτά μπορούν να δημιουργηθούν σε οποιαδήποτε πλατφόρμα που υποστηρίζει την Java. Για παράδειγμα, μπορούμε να γράψουμε ένα πρόγραμμα Java σε ένα Macintosh και να το μεταγλωττίσουμε σε bytecode σε ένα μηχάνημα Windows.

Πηγαίος κώδικας (source code) είναι το σύνολο των προτάσεων προγραμματισμού που εισάγει ένας προγραμματιστής σε έναν κειμενογράφο, όταν δημιουργηθεί ένα πρόγραμμα. Ο πηγαίος κώδικας μεταγλωττίζεται σε bytecode έτσι ώστε να μπορεί να εκτελεστεί από μία εικονική μηχανή Java. Ο bytecode είναι παρόμοιος με τον κώδικα μηχανής που παράγεται από άλλες γλώσσες, αλλά δεν είναι συγκεκριμένος για κανένα επεξεργαστή. Προσθέτει ένα επίπεδο ανάμεσα στον πηγαίο κώδικα και στον κώδικα μηχανής, όπως φαίνεται στην Εικόνα 2.

Η εικονική μηχανή της Java μπορεί να βρεθεί σε αρκετές θέσεις. Για μικροεφαρμογές η εικονική μηχανή είναι είτε ενσωματωμένη μέσα σε έναν περιηγητή ιστού με δυνατότητες Java ή είναι εγκατεστημένη ξεχωριστά για χρήση από τον επεξεργαστή. Οι προγραμματιστές μικροεφαρμογών δεν χρειάζεται να ασχολούνται με το αν υπάρχει σε ένα σύστημα χρήστη ή όχι.

Οι εφαρμογές Java από την άλλη, μπορούν να εκτελεστούν μόνο σε ένα σύστημα στο οποίο έχει εγκατασταθεί η αντίστοιχη εικονική μηχανή της Java.

Γλώσσες σαν την Visual Basic δημιουργούν κώδικα συγκεκριμένο για την πλατφόρμα, μπορούμε να θεωρήσουμε ότι ο διερμηνευτής bytecode προσθέτει ένα άχρηστο επίπεδο ανάμεσα στον πηγαίο κώδικα και στον μεταγλωττισμένο κώδικα μηχανής.

Αυτό το επίπεδο εισάγει ορισμένα προβλήματα απόδοσης – τα προγράμματα Java εκτελούνται βραδύτερα από τις μεταγλωττισμένες γλώσσες που εξαρτώνται από την πλατφόρμα, σαν τη C, και η διαφορά ταχύτητας είναι το κύριο μειονέκτημα της Java. Ορισμένα εργαλεία ανάπτυξης της Java περιλαμβάνουν γρήγορους μεταγλωττιστές (just-in-time-compilers), που μπορούν να εκτελέσουν Java bytecode με μεγαλύτερη ταχύτητα.

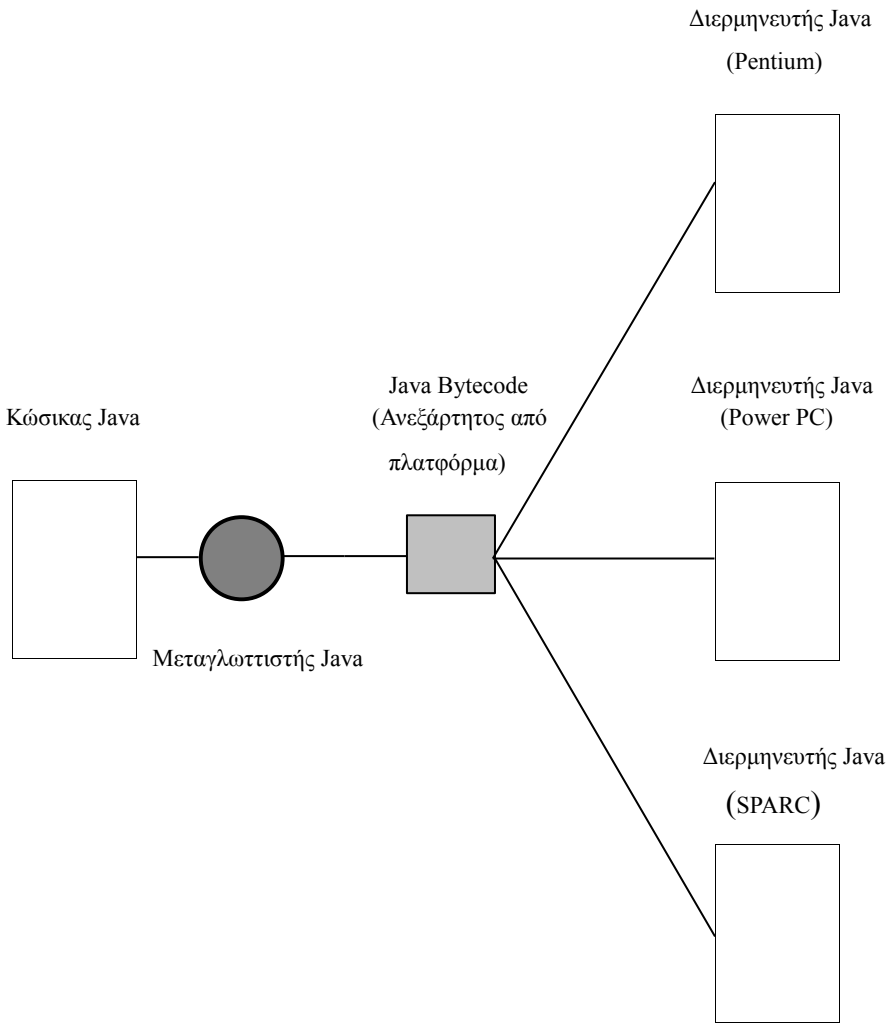
Η δυνατότητα ένα αρχείο bytecode να εκτελείται σε διάφορες πλατφόρμες είναι αυτό που κάνει τη Java να εργάζεται στον Παγκόσμιο Ιστό, επειδή το ίδιο το διαδίκτυο είναι ανεξάρτητο από πλατφόρμα. Όπως τα αρχεία `HTML` μπορούν να διαβαστούν σε οποιαδήποτε πλατφόρμα, οι μικροεφαρμογές μπορούν να εκτελεστούν σε οποιαδήποτε πλατφόρμα που διαθέτει έναν περιηγητή με δυνατότητες Java.

Η Java Είναι Αντικειμενοστρεφής

Αντικειμενοστρεφής προγραμματισμός (object oriented programming, OOP), είναι ένας τρόπος για να μελετήσουμε ένα πρόγραμμα υπολογιστή σαν ένα σύνολο αλληλεπιδρόντων αντικειμένων. Για ορισμένους είναι απλώς ένας τρόπος για να οργανώσουν προγράμματα, και κάθε γλώσσα μπορεί να χρησιμοποιηθεί για δημιουργία αντικειμενοστρεφών προγραμμάτων.

Αλλά όμως, χρησιμοποιώντας μία γλώσσα που είναι σχεδιασμένη για τον αντικειμενοστρεφή προγραμματισμό, σαφώς, έχουμε τα μεγαλύτερα πλεονεκτήματά του. Η Java κληρονομεί πολλές από τις αρχές OOP από την C++, τη γλώσσα στην οποία βασίζεται. Η Java δανείζεται επίσης αρχές από άλλες αντικειμενοστρεφείς γλώσσες. Η κεντρική ιδέα του αντικειμενοστρεφούς προγραμματισμού είναι απλή Οργανώνουμε τα προγράμματά μας κατά ένα τρόπο που να αντανακλά τον τρόπο οργάνωσης αντικειμένων στον πραγματικό κόσμο.

ΕΙΚΟΝΑ 2.
Προγράμματα για πολλαπλές πλατφόρμες της Java.



Η Java Είναι Εύκολη στην Εκμάθηση

Εκτός της μεταφερσιμότητάς της και της αντικειμενοστρεφούς της φύσης, η Java είναι μικρότερη και απλούστερη από τις αντίστοιχές της γλώσσες. Αυτό οφείλεται στον αρχικό στόχο της Java που ήταν να είναι μία γλώσσα που απαιτεί λιγότερη υπολογιστική δύναμη για να εκτελεστεί.

Η Java είχε στόχο να είναι ευκολότερη στη συγγραφή, στη μεταγλώττιση, στην αποσφαλμάτωση και στην εκμάθηση. Η γλώσσα μοντελοποιήθηκε σύμφωνα με την C++, και το μεγαλύτερο μέρος της σύνταξής της και της αντικειμενοστρεφούς της δομής παρέχεται απευθείας από αυτήν τη γλώσσα.

Παρά τις ομοιότητες της Java με την C++, τα πιο περίπλοκα και ευαίσθητα σε σφάλματα μέρη αυτής της γλώσσας έχουν εξαιρεθεί από την Java. Δε θα βρούμε δείκτες ή αριθμητική δεικτών στη Java, επειδή αυτά τα χαρακτηριστικά μπορούν να δημιουργήσουν σφάλματα σε ένα πρόγραμμα και διορθώνονται πιο δύσκολα. Οι συμβολοσειρές (string) και οι πίνακες στη Java, και η διαχείριση μνήμης χειρίζονται αυτόματα και δεν απαιτούν από τον προγραμματιστή να τα παρακολουθεί.

Αν και η Java μαθαίνεται ευκολότερα από τις άλλες γλώσσες προγραμματισμού, ένα άτομο χωρίς καμία προγραμματιστική εμπειρία θα έχει δυσκολίες με την Java. Σίγουρα όμως η κατανόησή της μπορεί να επιτευχθεί και από έναν αρχάριο προγραμματιστή.

Βασικές Έννοιες του Αντικειμενοστρεφή Προγραμματισμού με την Java

Στον αντικειμενοστρεφή προγραμματισμό, το συνολικό πρόγραμμα αποτελείται από διάφορα συστατικά που καλούνται αντικείμενα.

Ένα *αντικείμενο* (object) είναι ένα αυτόνομο στοιχείο ενός προγράμματος υπολογιστή που παριστά μία σχετική ομάδα χαρακτηριστικών και είναι σχεδιασμένο ώστε να κάνει συγκεκριμένες εργασίες. Τα αντικείμενα καλούνται επίσης, *ομότυπα* (instances).

Για να δημιουργήσουμε ένα καινούριο αντικείμενο χρησιμοποιούμε τον τελεστή `new` με το όνομα της κλάσης της οποίας θέλουμε να δημιουργήσουμε ένα ομότυπο, ακολουθούμενο από παρανθέσεις ως εξής :

```
JEditorPane edit = new JEditorPane();
```

Για να πάρουμε την τιμή μιας μεταβλητής ομοτύπου, χρησιμοποιούμε το συμβολισμό της τελείας ως εξής :

```
doc = (ExpressionDocument) edit.getDocument();
```

Ο αντικειμενοστρεφής προγραμματισμός μοντελοποιείται με βάση την παρατήρηση ότι στον πραγματικό κόσμο τα αντικείμενα αποτελούνται από πολλά μικρότερα αντικείμενα. Αλλά όμως η δυνατότητα συνδυασμού αντικειμένων είναι μόνο μία γενική αρχή του αντικειμενοστρεφούς προγραμματισμού. Περιλαμβάνει επίσης αρχές και χαρακτηριστικά που κάνουν τη δημιουργία και τη χρήση αντικειμένων ευκολότερη και πιο ευέλικτη. Το σημαντικότερο από αυτά τα χαρακτηριστικά είναι η κλάση.

Μία *κλάση* (class) είναι ένα πρότυπο που χρησιμοποιείται για τη δημιουργία πολλαπλών αντικειμένων με παρόμοια χαρακτηριστικά. Οι κλάσεις περιέχουν όλα τα χαρακτηριστικά ενός συγκεκριμένου συνόλου αντικειμένων.

Μία *μέθοδος* (method) είναι μία ακολουθία δηλώσεων συγκεντρωμένων σαν να αποτελούν ένα μικρό πρόγραμμα. Χρησιμοποιούνται για να κάνουν συγκεκριμένες εργασίες, με τον ίδιο τρόπο που οι συναρτήσεις χρησιμοποιούνται σε άλλες γλώσσες προγραμματισμού. Τα αντικείμενα επικοινωνούν μεταξύ τους χρησιμοποιώντας μεθόδους. Μία κλάση ή ένα αντικείμενο μπορεί να καλεί μεθόδους μίας άλλης κλάσης ή αντικειμένου.

Τέλος, όπως υπάρχουν μεταβλητές ομοτύπου και κλάσης, υπάρχουν μέθοδοι ομοτύπου και κλάσης, οι οποίες είναι πολύ συνηθισμένες και εφαρμόζονται σε ένα αντικείμενο της κλάσης. Αν η μέθοδος κάνει μία αλλάγη σε ένα μεμονωμένο αντικείμενο, πρέπει να είναι μία μέθοδος ομοτύπου. Οι μέθοδοι κλάσης εφαρμόζονται στην ίδια την κλάση.

Εκτός των κανονικών μεθόδων, μπορούμε επίσης να ορίσουμε μεθόδους κατασκευαστές στον ορισμό της κλάσης μας.

Ένας *κατασκευαστής* (constructor) είναι μία μέθοδος που καλείται σε ένα αντικείμενο όταν αυτό καλείται.

Οι κατασκευαστές μοιάζουν πολύ με τις κανονικές μεθόδους, με δύο βασικές διαφορές :

- Οι κατασκευαστές έχουν πάντα το ίδιο όνομα με την κλάση
- Οι κατασκευαστές δεν έχουν επιστρεφόμενο τύπο

Μία εισαγωγή στον αντικειμενοστρεφή προγραμματισμό δε θα ήταν πλήρης χωρίς
μία αναφορά σε πέντε αρχές κληρονομικότητα, τροποποιητές, αφηρημένες κλάσεις, διασυνδέσεις, πακέτα.

Κληρονομικότητα

Η κληρονομικότητα είναι μία από τις βασικότερες αρχές στον αντικειμενοστρεφή προγραμματισμό και έχει άμεση επίδραση στο πώς σχεδιάζουμε και γράφουμε τις δικές μας κλάσεις στη Java.

Κληρονομικότητα είναι ένας μηχανισμός που επιτρέπει σε μία κλάση να κληρονομεί όλη τη συμπεριφορά και τις ιδιότητες μίας άλλης κλάσης.

Μέσω της κληρονομικότητας, μία κλάση, έχει αμέσως όλη τη λειτουργικότητα μίας υπάρχουσας κλάσης. Λόγω αυτού, η νέα κλάση μπορεί να χρησιμοποιηθεί δηλώνοντας μόνο πώς διαφέρει από μία υπάρχουσα.

Με την κληρονομικότητα όλες οι κλάσεις διατάσσονται σε μία αυστηρή ιεραρχία – αυτές που δημιουργούμε εμείς και αυτές των βιβλιοθηκών.

Μία κλάση που κληρονομεί από μία άλλη κλάση καλείται *υποκλάση* ή *δευτερεύουσα κλάση* (subclass) και η κλάση που δίνει την κληρονομικότητα καλείται *υπερκλάση* (superclass). Στο παρακάτω παράδειγμα η κλάση `ExpLabelView` είναι η υποκλάση και η `LabelView` είναι η υπερκλάση, κάτι το οποίο δηλώνεται με τη λέξη-κλειδί `extends`.

```
public class ExpLabelView extends LabelView
```

Μία κλάση μπορεί να έχει μόνο μία υπερκλάση (μονή κληρονομικότητα), αλλά κάθε κλάση μπορεί να έχει απεριόριστο αριθμό υποκλάσεων. Οι υποκλάσεις κληρονομούν όλες τις ιδιότητες και την συμπεριφορά των υπερκλάσεών τους. Πρακτικά, αυτό σημαίνει ότι αν η υπερκλάση έχει συμπεριφορά και ιδιότητες που χρειάζεται η κλάση που θέλουμε να κατασκευάσουμε, δεν χρειάζεται να την ανακαθορίσουμε ή να αντιγράψουμε τον κώδικα για να έχουμε την ίδια συμπεριφορά και τις ίδιες ιδιότητες. Η κλάση μας δέχεται αυτόματα αυτά τα πράγματα από την υπερκλάση της, η υπερκλάση τα παίρνει από τη δική της υπερκλάση και ούτω καθεξής, σε όλη την ιεραρχία των κλάσεων.

Στην περίπτωση όπου μία υποκλάση ορίζει μία μέθοδο που έχει το ίδιο όνομα, τύπο επιστροφής και ορίσματα με μία μέθοδο ορισμένη σε μία υπερκλάση, τα πράγματα περιπλέκονται. Σε αυτή την περίπτωση, ο ορισμός μεθόδου που βρίσκεται πρώτος (αρχίζοντας από κάτω στην ιεραρχία και εργαζόμενοι προς τα πάνω) είναι αυτός που χρησιμοποιείται. Λόγω αυτού, μπορούμε να δημιουργήσουμε μία μέθοδο σε μία δευτερεύουσα κλάση που απαγορεύει τη χρήση μίας μεθόδου σε μία υπερκλάση. Για να το κάνουμε αυτό, δίνουμε στη μέθοδο το ίδιο όνομα, τύπο επιστροφής και ορίσματα με τη μέθοδο στην υπερκλάση. Η διαδικασία αυτή καλείται *υπέρβαση* (overriding).

Τροποποιητές

Η γλώσσα Java περιέχει αρκετούς *τροποποιητές* (modifiers) περιλαμβανομένων των

- Τροποποιητές για έλεγχο της πρόσβασης σε μία κλάση, μέθοδο ή μεταβλητή `public`, `protected` και `private`.
- Τον τροποποιητή `abstract`, για δημιουργία αφηρημένων κλάσεων και μεθόδων.

1) *Ιδιωτική πρόσβαση* : Για να κρύψουμε πλήρως μία μέθοδο ή μεταβλητή ώστε να μη μπορεί να χρησιμοποιηθεί από άλλες κλάσεις, χρησιμοποιούμε τον τροποποιητή `private`. Το μόνο μέρος που μπορούμε να δούμε αυτές τις μεθόδους ή μεταβλητές είναι μέσα από τις κλάσεις. Μία ιδιωτική μεταβλητή ομοτύπου, για παράδειγμα, μπορεί να χρησιμοποιηθεί από μεθόδους στη δική της κλάση, αλλά όχι από αντικείμενα σε άλλες κλάσεις. Ο περιορισμός αυτός επηρεάζει επίσης την κληρονομικότητα Ούτε οι ιδιωτικές μέθοδοι ούτε οι ιδιωτικές μεταβλητές κληρονομούνται από άλλες κλάσεις. Οι ιδιωτικές μεταβλητές είναι χρήσιμες σε δύο περιπτώσεις :

- Όταν άλλες κλάσεις δεν έχουν λόγο να χρησιμοποιούν αυτή τη μεταβλητή.
- Όταν μία άλλη κλάση μπορεί να προκαλέσει προβλήματα αλλάζοντας τη μεταβλητή με έναν ακατάλληλο τρόπο.

2) *Δημόσια πρόσβαση* : Σε μερικές περιπτώσεις ίσως να θέλουμε μία μέθοδο ή μία μεταβλητή σε μία κλάση να είναι πλήρως διαθέσιμη σε όποιες άλλες κλάσεις θέλουν να τη χρησιμοποιήσουν, τότε θα χρειαστούμε τον τροποποιητή `public`. Η μέθοδος main() μίας εφαρμογής πρέπει απαραιτήτως να είναι δημόσια. Λόγω της κληρονομικότητας της κλάσης, όλες οι δημόσιες μεταβλητές και μέθοδοι μίας κλάσης κληρονομούνται από τις δευτερεύουσες κλάσεις της.

3) *Προστατευμένη πρόσβαση* : Το τρίτο επίπεδο ελέγχου προσπέλασης είναι να περιορίσουμε μία μέθοδο και μεταβλητή σε χρήση από τις δύο παρακάτω ομάδες :

- Υποκλάσεις μίας κλάσης
- Άλλες κλάσεις στο ίδιο πακέτο

Αυτό το κάνουμε χρησιμοποιώντας τον τροποποιητή `protected`. Αυτό το επίπεδο πρόσβασης είναι χρήσιμο αν θέλουμε να διευκολύνουμε μία δευτερεύουσα κλάση να υλοποιηθεί. Η προστατευμένη πρόσβαση δίνει στη δευτερεύουσα κλάση την ευκαιρία να χρησιμοποιήσει μία βοηθητική μέθοδο ή μεταβλητή, ενώ ταυτόχρονα να απαγορεύει σε μία σχετική κλάση να την χρησιμοποιεί.

Αφηρημένες Κλάσεις

Σε μία ιεραρχία κλάσεων όσο υψηλότερα είναι η κλάση, τόσο πιο αφηρημένος είναι ο ορισμός της. Μία κλάση στην κορυφή της ιεραρχίας άλλων κλάσεων μπορεί να ορίσει μόνο τη συμπεριφορά και τις ιδιότητες που είναι κοινές για όλες τις κλάσεις. Πιο συγκεκριμένες συμπεριφορά και ιδιότητες θα βρίσκονται χαμηλότερα μέσα στην ιεραρχία. Όταν συλλέγουμε κοινή συμπεριφορά και τις ιδιότητες κατά τη διάρκεια του ορισμού μίας ιεραρχίας κλάσεων, μερικές φορές μπορεί να βρούμε μία κλάση που δεν χρειάζεται καν να εκκινήσει απευθείας. Αντί αυτού, αυτές οι κλάσεις λειτουργούν σαν μία θέση που θα περιέχει κοινή συμπεριφορά και ιδιότητες που μοιράζονται από τις δευτερεύουσες κλάσεις τους. Αυτές οι κλάσεις καλούνται *αφηρημένες κλάσεις* (abstract classes) και δημιουργούνται με τον τροποποιητή `abstract`.

Οι αφηρημένες κλάσεις μπορούν να περιέχουν οτιδήποτε περιέχει μία κανονική κλάση, περιλαμβανομένων μεθόδων δημιουργών, επειδή οι δευτερεύουσες κλάσεις τους, μπορεί να χρειάζεται να κληρονομούν τις μεθόδους. Οι αφηρημένες κλάσεις μπορούν επίσης να περιέχουν *αφηρημένες μεθόδους*, που είναι υπογραφές μεθόδων χωρίς υλοποίηση και υλοποιούνται στις δευτερεύουσες κλάσεις της αφηρημένης κλάσης.

Διασυνδέσεις

Η μονή κληρονομικότητα, δηλαδή η απαίτηση μία κλάση να έχει μόνο μία υπερκλάση, κάνει τη σχέση ανάμεσα στις κλάσεις και στη λειτουργικότητά τους, ευκολότερη στην κατανόηση και στην σχεδίαση. Αλλά όμως μπορεί να είναι περιοριστική – ειδικά όταν έχουμε παρόμοια συμπεριφορά που θέλουμε να επαναλαμβάνεται σε διάφορους κλάδους μίας ιεραρχίας κλάσης. Η Java επιλύει αυτό το πρόβλημα κοινής συμπεριφοράς χρησιμοποιώντας διασυνδέσεις.

Διασύνδεση (interface) είναι μία συλλογή μεθόδων που δηλώνει ότι μία κλάση έχει κάποια συμπεριφορά επιπρόσθετα σε αυτή που κληρονομεί από υπερκλάσεις της.

Οι διασυνδέσεις όπως και οι αφηρημένες κλάσεις και μέθοδοι παρέχουν πρότυπα συμπεριφοράς που αναμένεται να υλοποιήσουν άλλες κλάσεις. Οι διασυνδέσεις όμως παρέχουν περισσότερη λειτουργικότητα στη Java και στη σχεδίαση κλάσεων και αντικειμένων από αυτή που παρέχουν οι απλές αφηρημένες κλάσεις και μέθοδοι.

Πακέτα

Τα *πακέτα* (packages) είναι ένας τρόπος ομαδοποίησης σχετικών κλάσεων και διασυνδέσεων. Τα πακέτα επιτρέπουν σε ομάδες κλάσεων να χρησιμοποιούνται μόνο αν χρειάζονται, και εξαλείφουν πιθανές διενέξεις ανάμεσα σε ονόματα κλάσεων σε διάφορες ομάδες κλάσεων.

Τα πακέτα είναι χρήσιμα για αρκετούς λόγους :

- Επιτρέπουν την οργάνωση κλάσεων σε μονάδες, έτσι ώστε να μπορούμε να χρησιμοποιήσουμε μόνο ότι χρειάζεται για κάθε πρόγραμμα.
- Μειώνουν τα προβλήματα με διενέξεις ονομάτων. Όπως αυξάνει ο αριθμός των κλάσεων
- Java, έτσι αυξάνεται και η πιθανότητα να χρησιμοποιούμε το ίδιο όνομα κλάσης με κάποιον άλλο και η πιθανότητα να έχουμε διενέξεις ονομάτων και σφάλματα, αν προσπαθήσουμε να ενσωματώσουμε ομάδες κλάσεων σε ένα πρόγραμμα. Τα πακέτα μας επιτρέπουν να

"κρύψουμε" κλάσεις έτσι ώστε να αποφεύγονται οι διενέξεις.

- Μας επιτρέπουν να προστατεύσουμε κλάσεις, μεταβλητές και μεθόδους με περισσότερους τρόπους, εκτός του βασικού τρόπου κλάση προς κλάση.

Αν και ένα πακέτο είναι συνήθως μία συλλογή κλάσεων, τα πακέτα μπορούν επίσης να περιέχουν άλλα πακέτα, που δημιουργούν ένα ακόμα επίπεδο οργάνωσης αντίστοιχο με της κληρονομικότητας.

Για να εισάγουμε κλάσεις από ένα πακέτο, χρησιμοποιούμε την εντολή `import`. Μπορούμε είτε να εισάγουμε μία μεμονωμένη κλάση ως εξής :

```
import javax.swing.text.View;
```

ή μπορούμε να εισάγουμε ένα ολόκληρο πακέτο κλάσεων, χρησιμοποιώντας ένα αστερίσκο για να αντικαταστήσουμε τα ονόματα κλάσεων ως εξής :

```
import javax.swing.text.*;
```

Οι προτάσεις `import` στον ορισμό της κλάσης πηγαίνουν στην αρχή του αρχείου, πριν από κάθε ορισμό κλάσης αλλά μετά τον ορισμό του πακέτου, δηλαδή :

```
package gr.aegean.math.views;
import javax.swing.text.View;
```

Οι Λέξεις Κλειδιά this, super

Στο σώμα του ορισμού μίας μεθόδου, μπορεί να θέλουμε να αναφερθούμε στο αντικείμενο με το οποίο κλήθηκε η μέθοδος. Αυτό μπορεί να γίνει για να χρησιμοποιήσουμε τις μεταβλητές ομοτύπου αυτού του αντικειμένου, ή για να περάσει το τρέχον αντικείμενο σαν όρισμα σε μία άλλη μέθοδο. Σε αυτές τις περιπτώσεις χρησιμοποιούμε τη λέξη κλειδί `this` εκεί όπου κανονικά θα αναφερόμασταν στο όνομα του αντικειμένου.

Η λέξη κλειδί `super`, που είναι κάπως σα τη λέξη κλειδί `this`, είναι μία δέσμευση μνήμης για την υπερκλάση της κλάσης. Μπορούμε να τη χρησιμοποιήσουμε όπου χρησιμοποιούμε και τη λέξη `this`, αλλά η λέξη `super` αναφέρεται στην υπερκλάση της τρέχουσας κλάσης.

ΚΕΦΑΛΑΙΟ 2.

ΤΟ ΣΥΝΟΛΟ ΚΛΑΣΕΩΝ SWING

Το Swing

Η εργαλειοθήκη Java Swing αποτελεί τμήμα της βασικής βιβλιοθήκης της Java με σκοπό τη δημιουργία γραφικής διασύνδεσης χρήστη (Graphical User Interface – GUI) σε εφαρμογές.

Το Swing περιέχει ένα σύνολο από συστατικά για την δημιουργία GUIs και για να προσθέσουμε αλληλεπίδραση με τον χρήστη στις εφαρμογές μας. Γενικότερα, περιέχει όλα τα συστατικά που θα περιμέναμε από μία μοντέρνα εργαλειοθήκη.

Το Swing που είναι μέρος της βιβλιοθήκης `Java Foundation Classes` (JFC) και αποτελέι επέκταση του `Abstract Windowing Toolkit` (AWT), που έχει ενσωματωθεί στη Java 2. Ωστόσο, το Swing απέχει κατά πολύ από μία απλή εργαλειοθήκη συστατικών. Παρέχει βελτιωμένη λειτουργικότητα σε σχέση με τον πρόγονό του – νέα συστατικά, επεκταμένα χαρακτηριστικά συστατικών, καλύτερο χειρισμό συμβάντων και μία καλή εμφάνιση και αίσθηση.

Το Swing επιτρέπει σε ένα πρόγραμμα Java να χρησιμοποιεί μία διαφορετική εμφάνιση και αίσθηση υπό τον έλεγχο του προγράμματος ή ακόμη και του χρήστη ενός προγράμματος. Το χαρακτηριστικό αυτό παρέχει την πλέον δραματική οπτική αλλαγή από το AWT. Το Swing μας επιτρέπει να δημιουργήσουμε ένα πρόγραμμα Java με μία διασύνδεση που χρησιμοποιεί το στυλ του εγγενούς λειτουργικού συστήματος, όπως των Windows ή του Solaris ή ένα νέο στυλ μοναδικό στην Java, το Metal.

Τα συστατικά του Swing, σε αντίθεση με τους πρόγονούς του, υλοποιούνται πλήρως στη Java. Αυτό τα κάνει πιο συμβατά ανάμεσα σε πλατφόρμες, από τα προγράμματα που μπορεί να δημιουργήσουμε χρησιμοποιώντας το AWT.

Όλα τα στοιχεία του Swing είναι μέρος του πακέτου `javax.swing`. Η χρήση ενός συστατικού Swing δεν είναι διαφορετική από τη χρήση συστατικών AWT. Δημιουργούμε το συστατικό καλώντας τον κατασκευαστή του, καλώντας μεθόδους του συστατικού αν χρειάζονται για σωστή εγκατάσταση και προσθέτουμε το συστατικού σε ένα υποδοχέα.

Η Χρήση του Swing

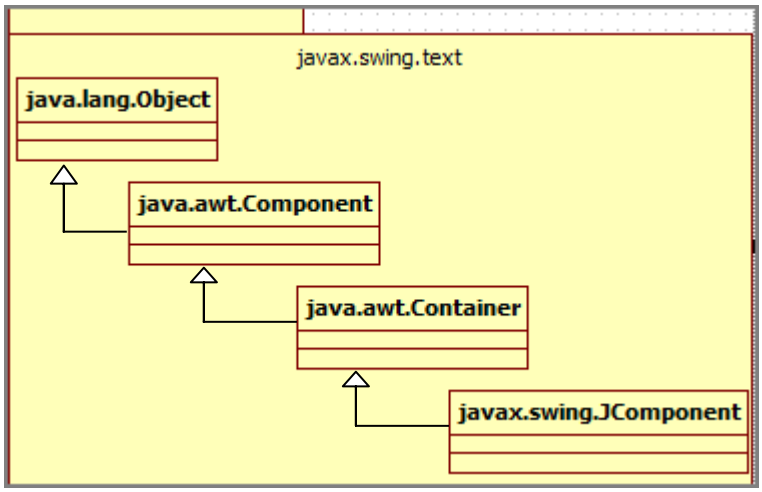
Κάθε γραφική διασύνδεση τύπου Swing πρέπει να έχει τουλάχιστον έναν *κορυφαίο υποδοχέα Swing* (top-level Swing container). Ένας κορυφαίος υποδοχέας Swing παρέχει την απαραίτητη υποστήριξη που χρειάζονται τα *συστατικά Swing* (Swing components) για την εμφάνισή τους και την διαχείριση των γεγονότων που αυτά παράγουν. Υπάρχουν τρείς κορυφαίοι υποδοχείς Swing το `JFrame`, το `JDialog` και (για μικροεφαρμογές) το `JApplet`. Κάθε `JFrame` αντικείμενο δημιουργεί ένα κύριο γραφικό παράθυρο και κάθε `JDialog` δημιουργεί ένα δευτερεύον παράθυρο (δηλαδή παράθυρο που εξαρτάται από κάποιο άλλο παράθυρο). Κάθε `JApplet` αντικείμενο δημιουργεί την περιοχή εμφάνισης μίας μικροεφαρμογής στο παράθυρο του περιηγητή ιστού. Με εξαίρεση τους κορυφαίους υποδοχείς, π.χ το `JFrame`, όλα τα συστατικά Swing είναι υποκλάσεις της κλάσης `JComponent`. Η ιεραρχία της `JComponent` φαίνεται στην Εικόνα 3.

Η εργασία με ένα αντικείμενο `JFrame` είναι πιο πολύπλοκη από την εργασία με το αντίστοιχο του AWT. Αντί να προσθέσουμε υποδοχείς και συστατικά κατευθείαν στο πλαίσιο, πρέπει να τους προσθέσουμε σε ένα ενδιάμεσο υποδοχέα με όνομα *τομέα περιεχομένων* (content pane).

Ένα `JFrame` υποδιαιρείται σε αρκετούς διαφορετικούς τομείς. Ο κύριος τομέας που εργαζόμαστε είναι ο τομέας περιεχομένων, που παριστά την πλήρη περιοχή ενός πλαισίου, στο οποίο μπορούν να

προστεθούν συστατικά.

ΕΙΚΟΝΑ 3.



Συνήθως, όταν υλοποιείται μία GUI εφαρμογή σε Swing δημιουργούμε ένα `JFrame` αντικείμενο και επιλέγουμε συγκεκριμένο πλάνο ή σχέδιο (layout) για αυτό. Κατόπιν τοποθετούμε ένα ή περισσότερα `JPanels` στο `JFrame`. Τα `JPanels` διαθέτουν επίσης επιλογές πλάνων όπως και τα `JFrames`. Στη συνέχεια μπορούμε να προσθέσουμε κι άλλα αντικείμενα τύπου `Component`.

Υπάρχουν συστατικά του Swing για καθένα από τα συστατικά του AWT. Στις περισσότερες περιπτώσεις υπάρχει ένας δημιουργός για το συστατικό Swing που ταιριάζει με το αντίστοιχο συστατικό του AWT. Υπάρχουν επίσης νέοι δημιουργοί για πολλά συστατικά που δέχονται ένα αντικείμενο `Icon` σαν όρισμα. Ένα εικονίδιο είναι ένα μικρό γραφικό, που μπορεί να τοποθετηθεί σε ένα κουμπί, μία ετικέτα, ή ένα άλλο συστατικό της διασύνδεσης, για να το ταυτοποιεί.

Τα Συστατικά του Swing

Ετικέτες

Οι ετικέτες υλοποιούνται στο Swing με την κλάση `JLabel`. Η λειτουργικότητα είναι παρόμοια με τις ετικέτες του AWT, αλλά τώρα μπορούμε να περιλάβουμε εικονίδια. Επίσης, η στοίχιση μίας ετικέτας μπορεί να καθοριστεί με μία από τις τρεις μεταβλητές κλάσης από την κλάση `SwingConstans` LEFT, CENTER ή RIGHT.

ΕΙΚΟΝΑ 4.

Κουμπιά

Τα κουμπιά του Swing βρίσκονται στην κλάση `JButton`. Ένα κουμπί Swing μπορεί να περιέχει μία ετικέτα κειμένου (όπως και τα κουμπιά του AWT), μία ετικέτα εικονιδίου ή ένα συνδυασμό και των δύο.

ΕΙΚΟΝΑ 5.

Πλαίσια Ελέγχου και Πλαίσια Επιλογής

Η κλάση `JCheckBox` είναι η υλοποίηση πλαισίων ελέγχου στο Swing. Η λειτουργικότητα είναι η ίδια με το AWT, με την προσθήκη ετικετών εικονιδίου.

Οι ομάδες πλαισίων ελέγχου υλοποιούνται στο Swing με την κλάση `ButtonGroup`. Σε μία ομάδα πλαισίων ελέγχου μπορεί να επιλεγεί μόνο ένα συστατικό τη φορά. δημιουργούμε ένα αντικείμενο `ButtonGroup` και προσθέτουμε πλαίσια ελέγχου σε αυτό χρησιμοποιώντας τη μέθοδο `add(Component)` για να προσθέσουμε ένα συστατικό στην ομάδα.

Τα κουμπιά επιλογής υλοποιούνται στο Swing με την κλάση `JRadioButton`. Οι κατασκευαστές είναι οι ίδιοι με αυτούς της κλάσης `JCheckBox`.

Η αλλαγή ονόματος από `CheckboxGroup` σε `ButtonGroup` αντανακλά εκτεταμένη λειτουργικότητα – τα κουμπιά και τα κουμπιά επιλογής μπορούν επίσης να ομαδοποιούνται.

Στην Εικόνα 6 φαίνεται ένα παράθυρο στο οποίο ο χρήστης μπορεί να επιλέξει το άθλημα που προτιμά.

ΕΙΚΟΝΑ 6.

Λίστες Επιλογής

Οι λίστες επιλογής που δημιουργούνται στο AWT χρησιμοποιώντας την κλάση `Choice` είναι μία από τις υλοποιήσεις που είναι δυνατές με την κλάση `JComboBox`.

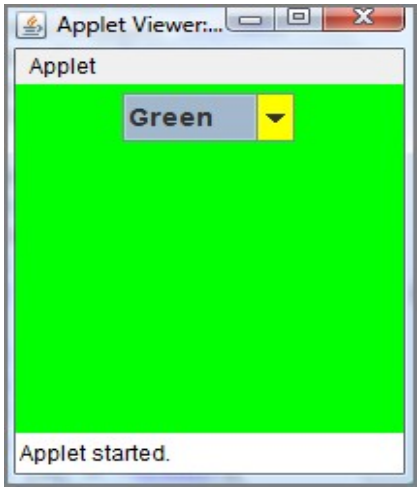
Τα παρακάτω βήματα δείχνουν πώς δημιουργείται μία λίστα επιλογής :

1. Χρησιμοποιείται ο κατασκευαστής `JComboBox()` χωρίς ορίσματα.
2. Η μέθοδος `addItem(Object)` του σύνθετου πλαίσιου προσθέτει στοιχεία στη λίστα.
3. Χρησιμοποιείται η μέθοδος `setEditable(boolean)` του σύνθετου πλαίσιου με ψευδές σαν όρισμά της.

Αυτή η τελευταία μέθοδος μετατρέπει το σύνθετο πλαίσιο σε λίστα επιλογής – οι μόνες επιλογές που μπορεί να κάνει ο χρήστης είναι από τα στοιχεία που έχουν προστεθεί στη λίστα.

Αν το σύνθετο πλαίσιο είναι επεξεργάσιμο, ο χρήστης μπορεί να εισάγει κείμενο στο πεδίο, αντί να χρησιμοποιήσει τη λίστα επιλογής, για να επιλέξει ένα στοιχείο. Αυτή είναι μία συνδυασμένη λειτουργία, που δίνει στα σύνθετα πλαίσια το όνομά τους. Παρακάτω, η Εικόνα 7 δείχνει ένα παράδειγμα με λίστα επιλογής, όπου όταν ο χρήστης κάνει κλικ στο κουμπί με το βελάκι θα εμφανιστούν και άλλα χρώματα και όποιο επιλέξει θα είναι το χρώμα του φόντου του παραθύρου.

ΕΙΚΟΝΑ 7.

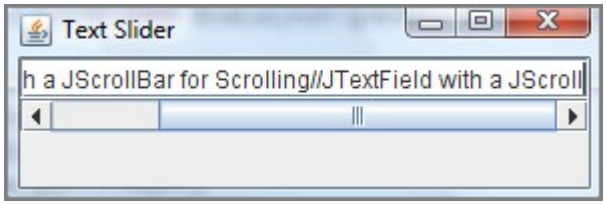


Γραμμές Κύλισης

Οι γραμμές κύλισης υλοποιούνται στο Swing με την κλάση `JScrollBar`. Η λειτουργικότητα είναι παρόμοια με τις γραμμές κύλισεις του AWT.

Ο προσανατολισμός δηλώνεται από τις μεταβλητές HORIZONTAL και VERTICAL της κλάσης `SwingConstants`.

ΕΙΚΟΝΑ 8.



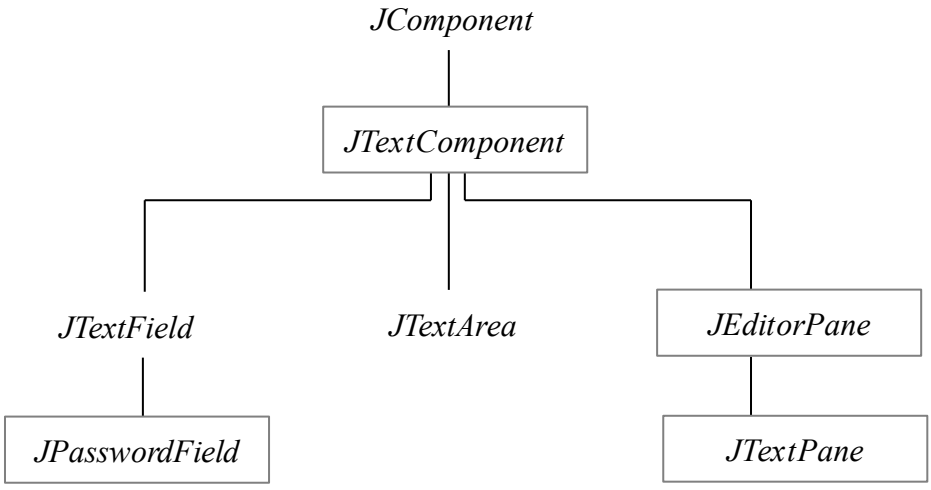
Η Εικόνα 8 είναι μία εφαρμογή που συνδυάζει το συστατικό κειμένου `JTextField`, το οποίο περιγράφεται παρακάτω, με την γραμμή κύλισης.

Το Swing είναι κατά κύριο λόγο γνωστό για τα άφθονα συστατικά GUI που περιέχει, όπως τα συστατικά κειμένου με τα οποία θα ασχοληθούμε.

Η Χρήση των Συστατικών Κειμένου

Όλα τα συστατικά κειμένου (text components) του Swing προέρχονται από την κλάση `JTextComponent`, που είναι μία υποκλάση της κλάσης `JComponent`. Το Swing παρέχει πέντε πλήρως λειτουργικούς μηχανισμούς ελέγχου κειμένου (text controls), τρεις από τους οποίους προσφέρουν λειτουργική αντικατάσταση των συστατικών του AWT `TextField` και `TextArea` ˙και δύο οι οποίοι είναι εντελώς καινούριοι. Η ιεραρχία των κλάσεων για τα συστατικά κειμένου του Swing φαίνενται στην Εικόνα 9. Αν και η Εικόνα 9 δείχνει όλα τα συστατικά κειμένου στο πακέτο του Swing, δε δείχνει το πολύπλοκο σύνολο από τις κλάσεις `model` και `view` που χρησιμοποιούνται για την πλήρη χρήση τους. Η περισσότερη δύναμη αυτών των συστατικών, ειδικά των μηχανισμών ελέγχου `JEditorPane` και `JTextPane`, προέρχεται από τη γενική και ευέλικτη αρχιτεκτονική τους που θα παρουσιαστεί αναλυτικά στη συνέχεια.

ΕΙΚΟΝΑ 9.



Τα `JTextField` και `JPasswordField` είναι τα πιο απλά από όλα τα συστατικά και περιέχουν μία μοναδική γραμμή κειμένου που μπορεί να παρέχεται από το πρόγραμμα ή να πληκτρολογείται από τον χρήστη. Η κύρια διαφορά από αυτούς τους δύο μηχανισμούς ελέγχου είναι ότι το

JPasswordField δεν επιδεικνύει απευθείας το περιεχόμενό του, αντιθέτως, χρησιμοποιεί ένα μοναδικό χαρακτήρα (συνήθως έναν αστερίσκο) για να αναπαριστά τους χαρακτήρες που περιλαμβάνει, έτσι ώστε να είναι δυνατό να δούμε πόσοι χαρακτήρες έχουν πληκτρολογηθεί, αλλά όχι ποιοι είναι. Όπως δηλώνει και το όνομά του το JPasswordField χρησιμοποιείται ως ένας απλός τρόπος αποδοχής για το μυστικό κωδικό του χρήστη. Επειδή προέρχεται από το JTextField, οτιδήποτε ισχύει για το JTextField ισχύει και για το JPasswordField.

```
public  class TextFieldExample {
      public static void main(String[] args) {

      JFrame f = new JFrame("Text Field Examples");
      f.getContentPane().setLayout(new FlowLayout());
      f.getContentPane().add(new JTextField
             ("Text field 1"));
      JTextField t = new JTextField("Text field 2", 8);

      t.setHorizontalAlignment(JTextField.RIGHT);
      f.getContentPane().add(t);

      t.setHorizontalAlignment(JTextField.CENTER);
      f.getContentPane().add(t);


      f.pack();
      f.setVisible(true);
      }
}
```

Το παραπάνω παράδειγμα για το JTextField θα εμφανίσει ένα παράθυρο, όπως φαίνεται στην Εικόνα 10, που θα περιέχει δύο ορθογώνια πλαίσια τα οποία το ένα θα έχει το όνομα "Text Field 1" και θα στοιχίζεται στα αριστερά και το άλλο το όνομα "Text Field 2" και θα στοιχίζεται στο κέντρο.

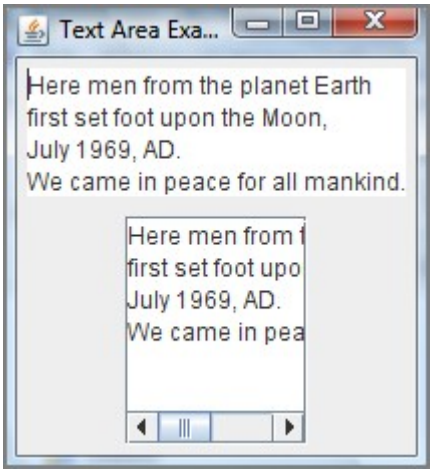
ΕΙΚΟΝΑ 10.

Το JTextArea είναι ένας δισδιάστατος μηχανισμός ελέγχου που επιτρέπει την απεικόνιση παραπάνω από μία γραμμών κειμένου. Με το JTextArea μπορούμε να δημιουργήσουμε μία περιοχή κειμένου αρχίζοντας με μία συμβολοσειρά στην οποία οι διαφορετικές γραμμές θα οριοθετούνται με τους χαρακτήρες αλλαγής γραμμής, έτσι ώστε κάθε γραμμή της συμβολοσειράς να απεικονίζεται σε διαφορετική γραμμή του μηχανισμού ελέγχου. Επίσης μπορούμε να ορίσουμε τον αριθμό των στηλών και των γραμμών που το JTextArea κατέχει. Όσο για το JTextField, φυσικά, προσδιορίζοντας αυτές τις τιμές το μόνο που επηρεάζεται είναι το επιθυμητό μέγεθος της περιοχής κειμένου χωρίς να καθορίζεται απαραίτητα το πραγματικό ύψος και πλάτος.

```
public  class TextAreaExample {
   public  static void main(String[] args) {
   JFrame f = new JFrame("Text Area Examples");
   JPanel upperPanel = new JPanel();
   JPanel lowerPanel = new JPanel();

      f.getContentPane().add(upperPanel, "North");
   f.getContentPane().add(lowerPanel, "South");

   upperPanel.add(new JTextArea(content));

   lowerPanel.add(new JScrollPane(
          new  JTextArea(content, 6, 8)));

   f.pack();
      f.setVisible(true);
   }

 static  String content = "Here men from the planet Earth\n" +
                 "first set foot upon the Moon,\n" +
                 "July 1969, AD.\n" +
                 "We came in peace for all mankind.";
}
```

Η παραπάνω εφαρμογή θα εμφανίσει ένα παράθυρο το οποίο θα περιέχει δύο ορθογώνιες περιοχές, όπως φαίνεται στην Εικόνα 11, τη μία κάτω από την άλλη μέσα στις οποίες θα περιέχεται κείμενο που θα υποστηρίζει περισσότερες από μία γραμμές.

ΕΙΚΟΝΑ 11.



Τα `JEditorPane` και `JTextPane` είναι τα συστατικά κειμένου με τις πιο πολλές δυνατότητες. Αυτοί οι δύο μηχανισμοί μπορούν να χρησιμοποιηθούν για να αποδώσουν ένα απλό κείμενο με τον

ίδιο τρόπο όπως το JTextArea και επίσης μπορούν να χρησιμοποιηθούν για να παραθέσουν πιο πολύπλοκο κείμενο με πληθώρα διάφορων γραμματοσειρών και χρωμάτων. Το JEditorPane είναι στην πραγματικότητα, ένα παράθυρο πάνω σε ένα συντάκτη εγγράφου (document editor). Μπορεί να διαμορφωθεί ώστε να παραθέτει οργανωμένο κείμενο, με το να συνδέει το συγκεκριμένο έγγραφο με μία *εργαλειοθήκη σύνταξης* (*editor kit)* που γνωρίζει πώς να ερμηνεύει το έγγραφο και αποδίδει το αντίστοιχο περιεχόμενο στην οθόνη.

Το Swing έρχεται με δύο ολοκληρωμένες εργαλειοθήκες σύνταξης που μπορούν να χρησιμοποιηθούν για να παραθέσουν έγγραφα κρυπτογραφημένα σε HTML (Hypertext Markup Language) και RTF (Rich Text Format). Μπορούμε να αντιμεταθέσουμε το JEditorPane με μία μέθοδο κατά την οποία μπορούμε να εισάγουμε δεδομένα (edit mode), και σε αυτή την περίπτωση μπορούμε να επιμεληθούμε το περιεχόμενο μίας σελίδας στην οθόνη και μετά να σώσουμε το προκύπτον έγγραφο, το οποίο θα έχει αντίστοιχο αποτέλεσμα με το να το είχαμε γράψει σε HTML. Με άλλα λόγια, μπορούμε να χρησιμοποιήσουμε το JEditorPane για ένα γρήγορα έτοιμο συντάκτη (editor). Αυτή η διευκόλυνση είναι επίσης διαθέσιμη για τα έγγραφα RTF.

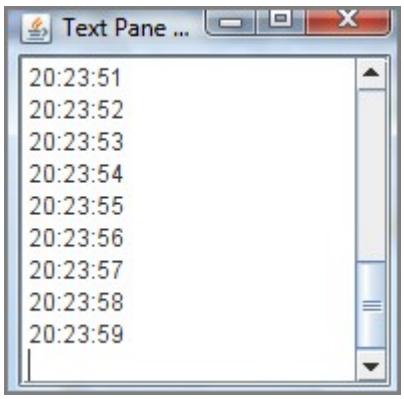
Η JTextPane είναι μία υποκλάση της JEditorPane που μας επιτρέπει να δημιουργήσουμε τα δικά μας έγγραφα με αυθαίρετους συνδυασμούς από γραμματοσειρές, χρώματα και οποιοδήποτε άλλο γνώρισμα ενδιαφερόμαστε να συσχετίσουμε με κείμενο ή με οποιοδήποτε άλλο ενεργό περιεχόμενο που μπορεί να έχουμε εμφυτεύσει στο έγγραφο. Χρησιμοποιώντας τη JTextPane μπορούμε για παράδειγμα, να συνδυάσουμε κείμενο με εικόνες ή με συστατικά Swing που φέρουν κάποιο ενεργό στοιχείο στο έγγραφο. Οι προγραμματιστικες διασυνδέσεις που μας επιτρέπουν να το κάνουμε αυτό είναι σχετικά σαφείς, αν και το *μοντέλο δεδομένων* (*data model*) που υποστηρίζει και τις δύο κλάσεις JEditorPane και JTextPane, είναι αρκετά σύνθετο.

```
public  class AppendingTextPane extends JTextPane {
   public  AppendingTextPane() {
   super ();
   }
   public  AppendingTextPane(StyledDocument doc) {
   super (doc);
   }

   public  void appendText(String text) {
   try  {
         Document doc = getDocument();
    setCaretPosition(doc.getLength());
    replaceSelection(text);
    Rectangle r = modelToView(doc.getLength());
        if  (r != null) {
           scrollRectToVisible(r);
    }
     } catch (BadLocationException e) {
    System.out.println("Failed to append text: " + e) ;
     }
 }

 public  static void main(String[] args) {
  JFrame f = new JFrame("Text Pane with Scrolling Append");
```

```
final  AppendingTextPane atp =
     new  AppendingTextPane();
f.getContentPane().add(new JScrollPane(atp));
      f.setSize(200, 200);
      f.setVisible(true);

   Timer t = new Timer(1000, new ActionListener() {
     public  void actionPerformed(ActionEvent evt) {

String timeString = fmt. format (new Date());
      atp.appendText(timeString + "\n");
         }

     SimpleDateFormat fmt =
      new  SimpleDateFormat("HH:mm:ss");
         });
     t.start();
  }
}
```

Το παραπάνω παράδειγμα θα εμφανίσει ένα παράθυρο, όπως φαίνεται στην Εικόνα 12, στο οποίο θα παρατίθενται η ώρα, τα λεπτά και τα δευτερόλεπτα της συγκεκριμένης χρονικής στιγμής που ο χρήστης θα τρέξει το πρόγραμμα και καθώς θα μεταβάλλεται η ώρα (ανά δευτερόλεπτο) στο παράθυρο θα φαίνεται η καινούρια ώρα μέχρις ώτου ο χρήστης κλείσει το παράθυρο. Λόγω αυτού του γεγονότος, στο παράθυρο αυτόματα θα δημιουργηθεί μία γραμμή κύλισης στα δεξιά, προκειμένου να εμφανίζονται στην οθόνη όλες οι μετατροπές της ώρας.

ΕΙΚΟΝΑ 12.



Μία εφαρμογή του μηχανισμού `JEditorPane` είναι η εφαρμογή η οποία παρουσιάζεται στην παρούσα εργασία.

ΚΕΦΑΛΑΙΟ 3.

Η ΑΡΧΙΤΕΚΤΟΝΙΚΗ ΤΩΝ ΣΥΣΤΑΤΙΚΩΝ ΚΕΙΜΕΝΟΥ

Η Αρχιτεκτονική Model View Controller

Το πρότυπο σχεδίασης Model View Controller (MVC) είναι ένα πάρα πολύ χρήσιμο εργαλείο στον προγραμματισμό. Το MVC μας επιτρέπει να διαχωρίσουμε τη λογική της εργασίας μας από το GUI, πράγμα που μας βοηθάει στο να μετατρέψουμε το ένα χωρίς να επηρεάζεται το άλλο. Αρχικά, το MVC απαιτεί λίγο πιο καλό σχεδιασμό και πολύ κώδικα, αλλά τα μακροπρόθεσμα πλεονεκτήματα μας αποδεικνύουν ότι αξίζει την προσπάθεια.

Όταν το MVC πρωτοπαρουσιάστηκε από τον Trygve Reenskaug το 1979, τα οπτικά εργαλεία είχαν περιορισμένες δυνατότητες και δεν ήταν τόσο διαδεδομένα όσο είναι σήμερα. Τα σημερινά οπτικά εργαλεία, υποστηρίζουν αληθινό οπτικό προγραμματισμό, και μας βοηθούν πρώτον στο να δημιουργήσουμε αρκετές από τις εφαρμογές του GUI και κατόπιν στη λογική της εργασίας.

Η βασική ιδέα του MVC είναι να ξεχωρίσουμε την εφαρμογή μας σε τρία ευδιάκριτα μέρη, κάθε ένα από τα οποία μπορεί να αντικατασταθεί χωρίς να επηρεάζει τα άλλα :

- Μοντέλο : Το μοντέλο παραθέτει τα δεδομένα και τους κανόνες που τα διέπουν και ανανεώνει τα δεδομένα. Το μοντέλο μπορεί να συμμετέχει σε οποιονδήποτε αριθμό αντικειμένων όψης και ελεγκτή.
- Όψη : Η όψη αποδίδει τα δεδομένα του μοντέλου και καθορίζει τον τρόπο με τον οποίο θα έπρεπε να παρουσιαστούν. Αν τα δεδομένα του μοντέλου αλλάξουν, η όψη θα πρέπει να αλλάξει την παρουσίαση όπως απαιτείται.
- Ελεγκτής : Ο ελεγκτής μεταφράζει τις αλληλεπιδράσεις του χρήστη με την όψη σε ενέργειες που θα εκτελέσει το μοντέλο. Ανάλογα με το περιεχόμενο, ένας ελεγκτής μπορεί επίσης να επιλέξει μία νέα όψη, για να παρουσιάσει στο χρήστη.

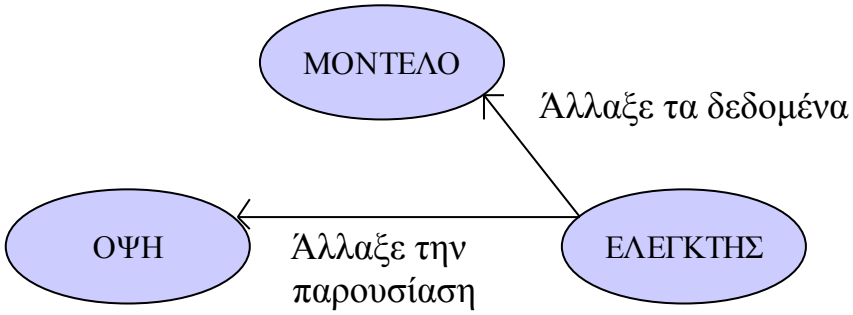
Αλληλεπίδραση Ανάμεσα στα Συστατικά

Αρχικά η όψη και ίσως και ο ελεγκτής, ρωτούν το μοντέλο για την τωρινή του κατάσταση. Η όψη μπορεί να παρουσιάσει δεδομένα στο χρήστη, και ο ελεγκτής μπορεί να ελέγξει τα δεδομένα για να εξετάσει το πώς να διαχειριστεί την αλληλεπίδραση χρήστη.

ΕΙΚΟΝΑ 13.

Όπως φαίνεται στην Εικόνα 13 η όψη και ο ελεγκτής τυπικά, θα ακούσουν για αλλαγές του μοντέλου. Η γλώσσα Java εφαρμόζει αυτή την αίτηση χρησιμοποιώντας συμβάντα (events). Όποτε το μοντέλο λέει "έχω αλλάξει" η όψη και ο ελεγκτής μπορούν να ρωτήσουν για την καινούρια κατάσταση των δεδομένων του μοντέλου και κατόπιν να ανανεώσουν τη δική τους εμφάνιση στο χρήστη.

ΕΙΚΟΝΑ 14.



Η Εικόνα 14 δείχνει την αλληλεπίδραση χρήστη στην εφαρμογή. Αν ο χρήστης αποφασίσει να αλληλεπιδράσει, το καθήκον το αναλαμβάνει ο ελεγκτής. Ο ελεγκτής παρακολουθεί για εισαγωγή δεδομένων από τον χρήστη, όπως το να μετακινήσει ή να κάνει κλικ στο ποντίκι του ή το να πατήσει κουμπιά του πληκτρολογίου. Κατόπιν αποφασίζει τί σημαίνει η αλληλεπίδραση, και ρωτάει το μοντέλο για να ανανεώσει τα δεδομένα του ή/και την όψη για τον τρόπο με τον οποίο θα παραθέσει αυτά τα δεδομένα.

Μόλις απεικονιστούν τα αντικείμενα του μοντέλου, της όψης και του ελεγκτή, συντελούνται τα επόμενα :

1. Η όψη λειτουργεί σαν ακροατής του μοντέλου. Οποιεσδήποτε αλλαγές στα βασικά δεδομένα του μοντέλου, αμέσα, απολήγουν σε μία ειδοποίηση για την αλλαγή, που λαμβάνει η όψη. Να σημειωθεί ότι το μοντέλο δεν έχει επίγνωση της όψης ή του ελεγκτή, απλά εκπέμπει ειδοποιήσεις για τις αλλαγές στους ενδιαφερόμενους ακροατές.

2. Ο ελεγκτής περιλαμβάνει την όψη. Αυτό, τυπικά σημαίνει ότι, οποιεσδήποτε ενέργειες του χρήστη που πραγματοποιούνται στην όψη, θα επικαλεστούν μία καταχωρημένη μέθοδο – ακροατή της κλάσης του ελεγκτή.

3. Ο ελεγκτής δίνει αναφορά στο βασικό μοντέλο.

Μόλις ο χρήστης αλληλεπιδρά με την όψη, συντελούνται τα επόμενα :

1. Η όψη αναγνωρίζει ότι μία GUI ενέργεια – για παράδειγμα το πάτημα ενός κουμπιού – έχει συμβεί, χρησιμοποιώντας μία μέθοδο – ακροατή, που έχει καταχωρηθεί να καλείται όταν συντελείται μία τέτοια ενέργεια.

2. Η όψη καλεί την κατάλληλη μέθοδο του ελεγκτή.

3. Ο ελεγκτής προσεγγίζει το μοντέλο, πιθανώς, με το να το ανανεώνει με ένα

   κατάλληλο τρόπο σύμφωνα με την ενέργεια του χρήστη.

4. Αν το μοντέλο έχει τροποποιηθεί, γνωστοποιεί στους ενδιαφερόμενους ακροατές, όπως είναι η όψη, την αλλαγή. Σε κάποιες αρχιτεκτονικές, ο ελεγκτής μπορεί να είναι επίσης, υπεύθυνος για την ανανέωση της όψης. Αυτό είναι συχνό φαινόμενο σε εγχειρήματα Java που βασίζονται σε αιτήσεις.

Δομή Μαθηματικών Εκφράσεων

Ένα έγγραφο (`Document`) το οποίο περιγράφει τις μαθηματικές εκφράσεις οι οποίες υποστηρίζονται από την εφαρμογή που περιγράφεται εδώ, είναι μια συμβολοσειρά η οποία περιέχει περιεχόμενο, δηλαδή κανονικούς χαρακτήρες, καθώς και *προδιαγραφές στοιχείων* (`ElementSpec`) οι οποίες καθορίζουν τη *δομή* του εγγράφου ως μια ιεραρχία στοιχείων. Το σύνολο των υποστηριζόμενων μαθηματικών εκφράσεων, είναι μια γλώσσα ανεξάρτητη από συμφραζόμενα (Appel και Jens, 2002). Μια τέτοια γλώσσα περιγράφεται από μια γραμματική. Μία τέτοια γραμματική περιέχει ένα σύνολο από κανόνες της μορφής :

```
symbol → symbol symbol symbol … symbol
```

όπου στην δεξιά μεριά βρίσκονται είτε κανένα είτε πιο πολλά σύμβολα. Κάθε σύμβολο είναι είτε *τερματικό* (terminal), δήλαγή είναι ένα δείγμα του αλφάβητου των συμβολοσειρών της γλώσσας, είτε *μη-τερματικό* (nonterminal), κάτι που σημαίνει ότι εμφανίζεται στην αριστερή μεριά ενός αποτελέσματος. Στο τέλος, ένα από τα μη-τερματικά σύμβολα ξεχωρίζει ως το αρχικό σύμβολο της γραμματικής. Κάθε σύμβολο είτε τεματικό είτε μη-τερματικό μπορεί να συσχετίζεται με τους δικούς του τύπους από σημαντικές τιμές. Ο τύπος που συσχετίζεται με ένα δείγμα, πρέπει να ταιριάζει με τον τύπο που το πρόγραμμα θα επιστρέψει για αυτό το δείγμα. Για τον κανόνα $A \rightarrow B\ C\ D$, η σημαντική κίνηση πρέπει να επιστρέψει μία τιμή που ο τύπος της είναι εκείνος που συσχετίζεται με το μη-τερματικό σύμβολο A. Ωστόσο, μπορεί και να χτίσει αυτή την τιμή από τις τιμές που συσχετίζονται με τα τερματικά/μη-τερματικά σύμβολα B, C, D.

Η γραμματική η οποία περιγράφει τη γλώσσα των υποστηριζόμενων μαθηματικών εκφράσεων είναι η ακόλουθη:

```
EXPRESSION → EXPRESSION EXPRESSION
EXPRESSION → FRACTION
EXPRESSION → EXPONENT
EXPRESSION → INDEX
EXPRESSION → INDEXEXPONENT
EXPRESSION → text
FRACTION → <fraction>EXPRESSION EXPRESSION</fraction>
EXPONENT → <exponent>EXPRESSION EXPRESSION</exponent>
FRACTION → <index>EXPRESSION EXPRESSION</index>
INDEXEXPONENT → <indexexponent>EXPRESSION EXPRESSION
</indexexponent>
```

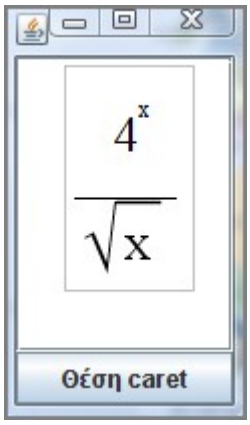
Στην παραπάνω γραμματική, το `EXPRESSION` είναι το αρχικό σύμβολο ενώ τα `INDEX, EXPONENT, INDEXEXPONENT, FRACTION` είναι μη τερματικά σύμβολα τα οποία αντιστοιχούν σε αντίστοιχες υπο-εκφράσεις με προφανή σημασία. Το σύμβολο `text` είναι τερματικό και παριστάνει μια

οποιαδήποτε συμβολοσειρά με περιεχόμενο απλούς χαρακτήρες. Τα σύμβολα <fraction>, </fraction>, κ.λπ., είναι τερματικά και αναφέρονται σε προδιαγραφές χαρακτήρων (`ElementSpecs`) οι οποίες αντιστοιχούν σε ετικέτες αρχής και τέλους με κατάλληλο όνομα.

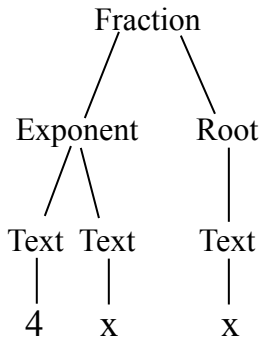
Αφηρημένα Συντακτικά Δέντρα και η Απεικόνισή τους

Η απεικόνιση ενός πηγαίου κώδικα σε δέντρο με κόμβους οι οποίοι αντιπροσωπεύουν σταθερές ή μεταβλητές και τελεστές ή δηλώσεις, είναι ένα *αφηρημένο συντακτικό δέντρο* (Abstract Syntax Tree), το οποίο καλείται επίσης, parse tree, ένα δέντρο δηλαδή, στο οποίο η γλώσσα διαιρείται σε μικρά συστατικά τα οποία μπορούν να αναλυθούν, κάτι το οποίο είναι πολύ σημαντικό στον κλάδο της Πληροφορικής. Να σημειωθεί ότι ένα αφηρημένο συντακτικό δέντρο φανερώνει την συντακτική δομή ενός προγράμματος. Για την εφαρμογή μας ένα συντακτικό δέντρο για την απεικόνιση της μαθηματικής έκφρασης που φαίνεται στην Εικόνα 15,

ΕΙΚΟΝΑ 15.



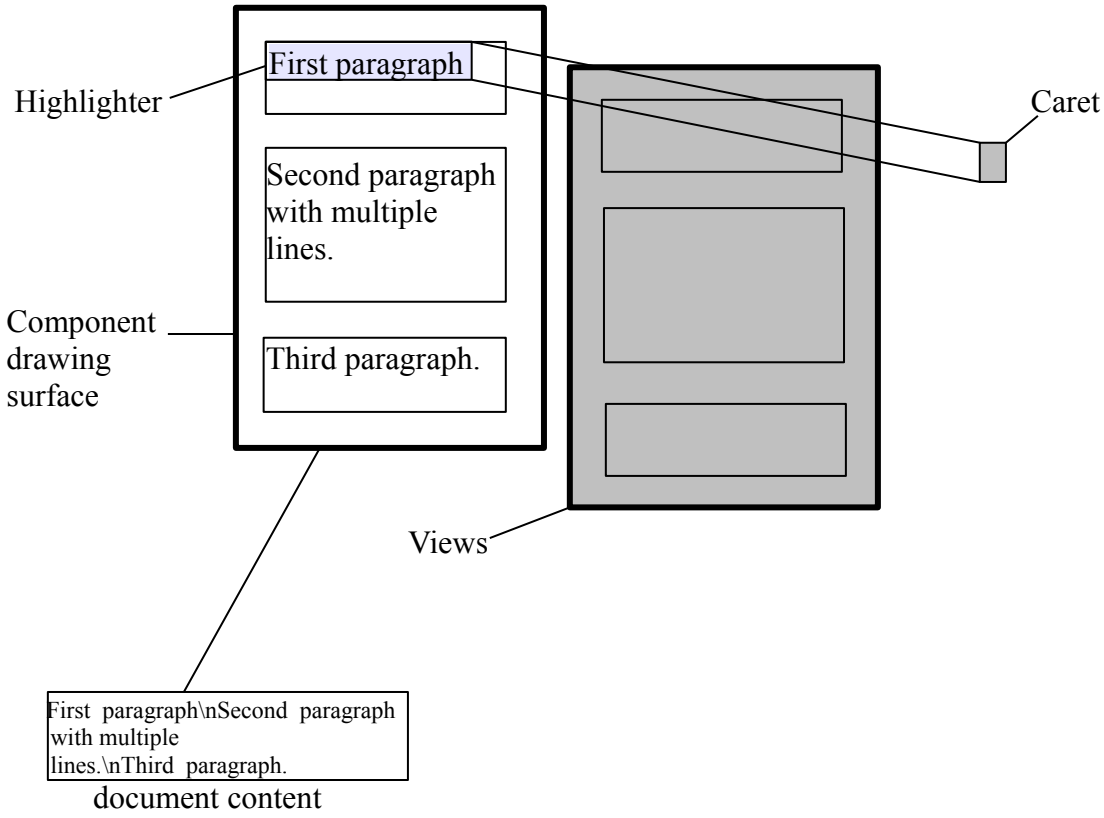
είναι το ακόλουθο :



Η Αρχιτεκτονική των Συστατικών Κειμένου

Τα συστατικά κειμένου του Swing παρέχουν μεγάλη λειτουργικότητα. Για να τα κατανοήσουμε ας εξετάσουμε τον τρόπο με τον οποίο είναι φτιαγμένα. Η Εικόνα 16 δείχνει τα σημαντικά τμήματα που δημιουργούν ένα συστατικό κειμένου. Αυτή είναι μία λογική εικόνα, περισσότερο από μία αντίστοιχη βασισμένη σε μία κλάση. Το διάγραμμα δείχνει ένα φανταστικό έγγραφο που περιέχει τρεις παραγράφους ενωμένες σε ένα αντικείμενο `Document`, απεικονισμένες από κάτω αριστερά. Οι παράγραφοι οριοθετούνται από χαρακτήρες αλλαγής γραμμής. Η πραγματική διάταξη αυτού του κειμένου στην οθόνη φαίνεται πάνω αριστερά. Εδώ, το κείμενο είναι χωρισμένο σε τρεις παραγράφους ευθυγραμμισμένες κατακόρυφα η μία πάνω από την άλλη και το περιεχόμενο από την πρώτη παράγραφο έχει επιλεγεί με το κούνημα του ποντικιού από πάνω του. Ο κέρσορας θα πρέπει να έχει τοποθετηθεί μετά τη λέξη ¨paragraph¨.

ΕΙΚΟΝΑ 16.



Το επιλεγμένο κείμενο είναι τονισμένο χρησιμοποιώντας ένα αντικείμενο που ονομάζεται *highlighter,* το οποίο ζωγραφίζει κατευθείαν πάνω στην επιφάνεια σχεδίασης του συστατικού. Ο highlighter αναπαριστάται από την σκοτεινή περιοχή πάνω από το κείμενο της πρώτης παραγράφου. Μπροστά από την περιοχή του συστατικού είναι μία λογική αναπαράσταση των *οπτικών απεικονίσεων – όψεων (views)* που στην πραγματικότητα αποδίδουν το κείμενο στην οθόνη. Εδώ, υπάρχει μία όψη για κάθε παράγραφο. Σε ένα πραγματικό συστατικό κειμένου υπάρχουν περισσότερα αντικείμενα απεικονίσεων, το σχήμα δείχνει μόνο τις όψεις των παραγράφων για χάριν της απλότητας. Ο κέρσορας, τέλος, είναι σχεδιασμένος χρησιμοποιώντας

ένα αντικείμενο που ονομάζεται δρομέας (caret). Όπως και ο highlighter έτσι και ο δρομέας ζωγραφίζει κατευθείαν πάνω στην επιφάνεια σχεδίασης του συστατικού.

Το Έγγραφο

Το *έγγραφο* αποθηκεύει τα δεδομένα που το συστατικό κειμένου παραθέτει και τις ιδιότητες που συνεργάζονται με αυτό ή με άλλα αντικείμενα μέσα στο έγγραφο. Ενώ τα `JTextField`, `JPasswordField` και `JTextArea` μπορούν να παραθέσουν μόνο κείμενο και περιορίζονται σε μία και μόνο γραμματοσειρά και δύο χρώματα (ένα για το φόντο και ένα για το προσκήνιο) - και τα δύο συστατικά `JEditorPane` και `JTextPane` υποστηρίζουν πολλαπλές γραμματοσειρές και χρώματα και μπορούν να φιλοξενήσουν πιο ενδιαφέροντα περιεχόμενα, όπως εικόνες και συστατικά μαζί με το κείμενο. Το έγγραφο περιέχει όλες τις πληροφορίες που περιγράφουν το περιεχόμενο του συστατικού, μαζί με τις γραμματοσειρές και τα χρώματα. Οι μέθοδοι που αναφέρονται στα περιεχόμενα του εγγράφου είναι καθορισμένες από την διασύνδεση `Document`. Το Swing περιέχει δύο χειροπιαστές υλοποιήσεις αυτής της διασύνδεσης που χρησιμοποιούνται από τα συστατικά κειμένου. Η `PlainDocument` χρησιμοποιείται για να υποστηρίζει μόνο κείμενο, δεν έχει μεθόδους για να υποστηρίζει πιο πολύπλοκα σχήματα που είναι εφικτά με τα συστατικά `JEditorPane` και `JTextPane`, που χρησιμοποιούν μία κλάση για έγγραφα που ονομάζεται `DefaultStyledDocument`. Εκτός από αυτές τις βασικές κλάσεις εγγράφων, το `HTML` παρέχει και μία τρίτη την `HTMLDocument`, η οποία προέρχεται από την `DefaultstyledDocument`.

Στοιχεία και Τύποι Στοιχείων

Αναφορικά με τις κλάσεις και τις διασυνδέσεις, όλα τα στοιχεία είναι εφαρμογές της διασύνδεσης `Element` που καθορίζεται από το πακέτο κειμένου του Swing (`javax.swing.text`). Τα πραγματικά στοιχεία είναι όλα ομότυπα των `LeafElement` ή `BranchElement`, που είναι εσωτερικές κλάσεις του `AbstractDocument` το οποίο αναφέρεται στην διασύνδεση `Element`. Αυτές οι δύο κλάσεις προέρχονται από τη βασική αφηρημένη κλάση `AbstractDocument.AbstractElement`. Παρακάτω βλέπουμε πώς ορίζεται η διασύνδεση `Element`.

```
public interface Element {
      public Document getDocument();
      public Element  getParentElement();
      public String   getName();
      public AtrributeSet  getAtrributes();
      public int  getStartOffset();
      public int  getEndOffset();
      public int  getElementIndex(int offset);
      public int  getElementCount();
      public Element  getElement(int index);
      public boolean  isLeaf();
}
```

Η μέθοδος `getDocument` επιστρέφει μία αναφορά στο `Document` του οποίου το στοιχείο είναι ένα μέρος. Επειδή υπάρχει και στα στοιχεία ιεραρχία, η μέθοδος `getParentElement` μπορεί να βρει το γονιό του στοιχείου στο οποίο απευθύνεται. Η μέθοδος `getName` επιστρέφει ένα `String` που περιγράφει τον τύπο του στοιχείου. Η μέθοδος `getAtrributes` επιστρέφει τα χαρακτηριστικά που συσχετίζονται με το στοιχείο· τα χαρακτηριστικά δε χρησιμοποιούνται πολύ από τα απλά συστατικά κειμένου. Οι μέθοδοι `getStartOffset` και `getEndOffset` επιστρέφουν τα αρχικά και τελικά offsets (θέσεις των στοιχείων) του περιεχομένου του συστατικού κειμένου, με τους οποίους συσχετίζεται το στοιχείο. Επειδή τα στοιχεία ιεραρχούνται, περισσότερα από ένα στοιχεία μπορεί να περιέχουν ένα συγκεκριμένο offset αλλά μόνο ένα στοιχείο περιεχομένου (σε μία δοσμένη

ιεραρχία) θα αποτυπώσει κάποιο καθορισμένο offset. Αυτό το offset μπορεί επίσης να είναι ανάμεσα σε μία σειρά αρχικών και τελικών offsets της παραγράφου που τα περιέχει. Τέλος, οι μέθοδοι `getElementIndex`, `getElementCount`, `getElement` και `isLeaf` χρησιμοποιούνται για να χαρτογραφήσουν ανάμεσα σε μία ιεραρχία στοιχείων και σε ένα περιεχόμενο εγγράφου. Όταν αυτές οι μέθοδοι απαντώνται σε ένα στοιχείο επιστρέφουν πληροφορία που είναι γνωστή από το στοιχείο. Η μέθοδος `getElementIndex` επιστρέφει τον τακτικό αριθμό του παιδιού του στοιχείου, που περιέχει το δοσμένο offset του εγγράφου. Η μέθοδος `getElementCount` επιστρέφει τον αριθμό των παιδιών που έχει ένα δοσμένο στοιχείο. Η μέθοδος `getElement` επιστρέφει το παιδί που ο δείκτης του αντιστοιχεί στο όρισμά της. Αν η μέθοδος απαντιέται σε ένα `LeafElement`, επιστρέφει πάντα `null`. Η μέθοδος `isLeaf` δείχνει αν το στοιχείο στο οποίο απαντιέται *δε* θα μπορούσε να περιέχει παιδιά, επιστρέφει αληθές (true) για ένα `LeafElement` και ψευδές (false) για ένα `BranchElement`. Να σημειωθεί ότι η μέθοδος `isLeaf` *πάντα* επιστρέφει ψευδές για ένα `BranchElement`, ακόμα και αν αυτό δεν έχει παιδιά όταν η μέθοδος καλείται.

Δομημένα Έγγραφα

Τα δομημένα έγγραφα, τα οποία και μας ενδιαφέρουν στην εργασία αυτή, είναι υποκλάσεις της `DefaultStyledDocument`. Ένα δομημένο έγγραφο αποτελείται από μια ιεραρχία στοιχείων (`Elements`). Η κατασκευή των στοιχείων ενός δομημένου εγγράφου γίνεται με χρήση της κλάσης `DefaultStyledDocument.ElementSpec` η οποία περιγράφει *προδιαγραφές στοιχείων*.

Τρεις τύποι *προδιαγραφής στοίχείων* μας ενδιαφέρουν στη συγκεκριμένη εργασία: `ContentType`, `StartTagType`, `EndTagType`. Κάθε ένας από τους παραπάνω τύπους ορίζεται ως μια ακέραια σταθερά της κλάσης `ElementSpec`. Ένα στοιχείο (Element) οριοθετείται από μια ετικέτα αρχής (start tag) και μια ετικέτα τέλους (end tag). Το στοιχείο είναι δυνατόν να περιέχει άλλα στοιχεία, ή απλό περιεχόμενο, με τη μορφή μιας ακολουθίας χαρακτήρων. Μια ετικέτα αρχής ορίζεται από τον τύπο `StartTagType`, μια ετικέτα τέλους από ένα `EndTagType` και το περιεχόμενο από τον τύπο `ContentType`.

Μια προδιαγραφή στοιχείου `ElementSpec` έχει ένα σύνολο από χαρακτηριστικά (attributes). Παραδείγματα χαρακτηριστικών αποτελούν το *όνομα* του στοιχείου, καθώς επίσης και προδιαγραφές μορφοποίησης, π.χ. χρώματος, γραμματοσειράς, παραγράφου, κ.λπ. Ένα σύνολο χαρακτηριστικών προδιαγράφεται από τη διασύνδεση `AttributeSet` του πακέτου `javax.swing.text`. Ποιο συγκεκριμένα, η κλάση `SimpleAttributeSet` υλοποιεί τη διασύνδεση `AttributeSet` επιτρέποντας την εισαγωγή χαρακτηριστικών στο σύνολο με τη μορφή όνομα-τιμή:

```
SimpleAttributeSet attributes = new SimpleAttributeSet();
attributes.addAttribute(AbstractDocument.ElementNameAttribute,
                        "name");
```

Οι παραπάνω εντολές έχουν ως αποτέλεσμα την εισαγωγή του χαρακτηριστικού όνομα (`AbstractDocument.ElementNameAttribute`) με τιμή τη συμβολοσειρά *"name"* στο σύνολο χαρακτηριστικών *attributes*.

Η δημιουργία μιας ετικέτας αρχής με το συγκεκριμένο όνομα γίνεται με χρήση του κατασκευαστή της `ElementSpec`:

```
ElementSpec start = new ElementSpec(attributes,
                        ElementSpec.StartTagType);
```

Αντίστοιχα δημιουργείται η ετικέτα τέλους:

```
ElementSpec end = new ElementSpec(attributes,
                        ElementSpec.EndTagType);
```

Η δημιουργία στοιχείου περιεχομένου γίνεται καλώντας τον παρακάτω κατασκευαστή της `ElementSpec`:

```
public DefaultStyledDocument.ElementSpec(AttributeSet a,  short type,
char[] txt, int offs, int len)
```

όπου `a` είναι ένα σύνολο χαρακτηριστικών, πιθανώς κενό, `type` ο τύπος του στοιχείου, `ContentType` για την εισαγωγή στοιχείου περιεχομένου, `txt` είναι πίνακας των χαρακτήρων που αποτελούν το περιεχόμενο, `offs` είναι η αρχική θέση (offset) μέσα στον πίνακα χαρακτήρων και `len` το μήκος του πίνακα. Για παράδειγμα, το παρακάτω τμήμα κώδικα:

```
String text = "A text";
ElementSpec textSpec = new ElementSpec(new SimpleAttributeSet(),
                        ElementSpec.ContentType, text.toCharArray(), 0,
                        text.length());
```

δημιουργεί ένα στοιχείο με περιεχόμενο το κείμενο `"A text"`.

Η παραπάνω δομή είναι πλήρως ευθυγραμμισμένη με τη δομή γλωσσών επισημείωσης (Markup) όπως η XML (Extensible Markup Language), όπου ένα έγγραφο αποτελείται από εμφωλευμένα (nested) στοιχεία τα οποία οριοθετούνται από ετικέτες αρχής και τέλους.

Η εισαγωγή στοιχείων (elements) σε ένα έγγραφο γίνεται καλώντας τη μέθοδο `insert` της κλάσης `DefaultStyledDocuent`:

```
protected void insert(int offset,
                      DefaultStyledDocument.ElementSpec[] data)
```

όπου `offset` είναι η θέση μέσα στο έγγραφο στην οποία θα εισαχθεί το στοιχείο και `data` είναι ένας πίνακας με προδιαγραφές στοιχείων (ElementSpec). Ως παράδειγμα, δίνεται η εσαγωγή της παρακάτω δομής. Στο παράδειγμα ακολουθείται η σύνταξη της XML για τη δήλωση των ετικετών αρχής και τέλους. Έτσι, η ακολουθία <text>Hello</text> παριστάνει μια προδιαγραφή στοιχείου τύπου `StartTagType` με όνομα `text`, ακολουθούμενη από μια περιγραφή στοιχείου περιεχομένου (`ContentType`) και από μια περιγραφή τύπου `EndTagType`. Η εισαγωγή του παραπάνω στοιχείου σε ένα έγγραφο, γίνεται με τον παρακάτω κώδικα:

```
ArrayList<ElementSpec> es = new ArrayList<ElementSpec>();
//Καθορισμός χαρακτηριστικού ονόματος
SimpleAttributeSet attrs = new SimpleAttributeSet();
attrs.addAttribute(AbstractDocument.ElementNameAttribute, "text");

//Εισαγωγή προδιαγραφής στοιχείου τύπου ετικέτας αρχής
es.add(new ElementSpec(attrs, ElementSpec.StartTagType));

String text = "Hello";
ElementSpec textSpec = new ElementSpec(new SimpleAttributeSet(),
                        ElementSpec.ContentType, text.toCharArray(), 0,
                        text.length());

//Εισαγωγή προδιαγραφής στοιχείου τύπου περιεχομένου
```

```
        es.add(textSpec);

        //Εισαγωγή προδιαγραφής στοιχείου τύπου ετικέτας τέλους
        es.add(new ElementSpec(attrs, ElementSpec.EndTagType));

        ElementSpec[] spec = new ElementSpec[es.size()];
        // Μετατροπή λίστας σε πίνακα
        es.toArray(spec);

        insert(0, spec);
```

Όλη η παραπάνω ακολουθία εισάγει ένα και μοναδικό στοιχείο (`Element`) στο έγγραφο με όνομα `text`. Όπως αναφέρθηκε, τα στοιχεία ενός εγγράφου είναι δυνατόν να είναι εμφωλευμένα. Ως παράδειγμα, ένα έγγραφο είναι δυνατόν να έχει την παρακάτω δομή:

```
<fraction>
        <exponent>
                <text>x</text>
                <text>4</text>
        </exponent>
        <root>x</root>
</fraction>
```

Ο Highlighter

Κάθε συστατικό έχει ένα σχετικό highlighter, που χρησιμοποιείται για να αναδείξει το κομμάτι του κειμένου που είναι ήδη επιλεγμένο. Ένας highlighter είναι μία υλοποίηση της διασύνδεσης `Highlighter`, της οποίας η κλάση `DefaultHighlighter` είναι μία υλοποίηση του πακέτου Swing. Όταν δημιουργηθεί το συστατικό, ένα ομότυπο της κλάσης `DefaultHighlighter` αρχικοποιείται και συνδέεται με αυτήν. Το `DefaultHighlighter` είναι μία απλή υλοποίηση που τονίζει επιλεγμένο κείμενο με το να αλλάζει το χρώμα του φόντου του στο επιλεγμένο χρώμα του συστατικού κειμένου που σχετίζεται με αυτό, το οποίο συνήθως αποσπάται από το αντικείμενο `UIDefaults`.

Ένας highlighter μπορεί στην πραγματικότητα να χρησιμοποιηθεί για να χρωματίσει αυθαίρετα τμήματα ενός εγγράφου. Κάθε τμήμα ελέγχεται από μία κλάση εφαρμόζοντας την διασύνδεση `Highlighter`. `HighlightPainter`. Όταν ένας highlighter χρειάζεται να σχεδιάσει τα σχετικά με αυτόν σημαντικά τμήματα, χρησιμοποιεί το αντίστοιχο `HighlightPainter` για να το κάνει. Η κλάση `DefaultHighlighter` χρησιμοποιεί το `DefaultHighlighter.DefaultHighlightPainter` το οποίο γεμίζει μία ορθογώνια περιοχή με ένα συμπαγές χρώμα, για να ζωγραφίσει.

Ο Δρομέας

Ο δρομέας (ή αλλιώς και κέρσορας) είναι κατά κύριο λόγο ενδιαφέρον επειδή παραθέτει το σημείο όπου το κείμενο θα μπει, σε απάντηση της πληκτρολόγησης. Ο κέρσορας είναι σχεδιασμένος σαν μία λεπτή κάθετη γραμμή.

Ο δρομέας μπορεί να είναι εφοδιασμένος από μία κλάση η οποία υλοποιεί τη διασύνδεση `Caret`. Υπάρχει μία υλοποίηση, που ονομάζεται `DefaultCaret`, η οποία εφοδιάζει όλη την λειτουργικότητα που απαιτείται από τα συστατικά κειμένου.

Το `Caret` περιέχει τα `property change`, `focus`, `mouse` και `mouse motion events` που συντελούνται στα δικά του συστατικά κειμένου και `DocumentEvents` από τα συστατικά της

διασύνδεσης `Document`. Σε απάντηση των `focus` και `mouse events`, ο δρομέας εκπληρώνει μέρος του ελεγκτή από το μοντέλο MVC. Το άλλο μέρος του ελεγκτή, το μέρος το οποίο χειρίζεται την εισαγωγή από το πληκτρολόγιο, εφαρμόζεται από το `JTextComponent`, αν και μεγάλο κομμάτι της εργασίας εκπροσωπεύεται από την εγκατεστημένη εργαλειοθήκη σύνταξης του συστατικού κειμένου.

Τα `property change events` χρησιμοποιούνται για να εντοπίσουν μία αλλαγή στο `Document` του συστατικού κειμένου, έτσι ώστε ο δρομέας να μπορεί να καταγράψει ο ίδιος, σαν ένας `DocumentListener`, για το τρέχον έγγραφο. Τα `DocumentEvents` παράγονται όταν το περιεχόμενο του μοντέλου του συστατικού κειμένου αλλάζει. Μία αλλαγή σε ένα έγγραφο περιεχομένων μπορεί να επηρεάσει την οπτική θέση του δρομέα αν το κείμενο έχει προστεθεί ή έχει μετακινηθεί πριν την τοποθέτηση του δρομέα στο κείμενο.

Τα `FocusEvents` χρησιμοποιούνται για να ανάψουν και να σβήσουν το δρομέα. Στην πράξη, ο δρομέας είναι ορατός μόνο όταν το συστατικό κειμένου είναι στο επίκεντρο. Όταν το συστατικό είναι στο επίκεντρο, ο δρομέας χρωματίζεται μόνος του στην τρέχουσα τοποθεσία, ενώ όταν χάνεται το επίκεντρο ο δρομέας μετακινείται.

Τέλος, τα `mouse events` χρησιμοποιούνται για να ελέγχουν την θέση του δρομέα και για να διαχειρίζονται την επιλογή του κειμένου. Ο δρομέας διατηρεί δύο ξεχωριστές θέσεις μέσα στο έγγραφο που αναφέρονται ως σημάδι (mark) και τελεία (dot). Η τελεία πάντα αντιστοιχεί στην τρέχουσα θέση του δρομέα, ενώ το σημάδι είναι στη θέση στην οποία ο δρομέας ήταν τελικά μετακινημένος με το κλικ του ποντικιού. Σε μία σειρά από έγγραφα ανάμεσα στο σημάδι και την τελεία υπάρχει η επιλογή. Ας υποθέσουμε ότι ο χρήστης κάνει κλικ στο ποντίκι κάπου μέσα στο συστατικό κειμένου. Αυτή η θέση καθίσταται θέση και του σημαδιού και της τελείας. Επειδή τώρα και το σημάδι και η τελεία κατέχουν την ίδια θέση μέσα στο έγγραφο , δε θα υπάρχει επιλογή. Αν τώρα ο χρήστης μετακινήσει το ποντίκι, ο δρομέας ακολουθεί το ποντίκι με το να μετακινεί την τελεία στη θέση που καταγράφηκε με την τελευταία μετακίνηση του ποντικιού. Το σημάδι ωστόσο, έχει μείνει πίσω.

Επειδή η μετακίνηση του ποντικιού επιφέρει τη δημιουργία μίας καινούριας επιλογής, ο δρομέας είναι υπεύθυνος να κάνει αυτή την επιλογή ορατή με το να καλέσει τον highlighter του συστατικού να ξαναζωγραφίσει το φόντο του επιλεγμένου κειμένου. Ένας συνηθισμένος highlighter, μπορεί βέβαια να χρησιμοποιήσει άλλα μέσα για να παραθέσει την επιλογή, όπως το να υπογραμμίσει τα σημεία του κειμένου που έχουν επηρεαστεί. Οι μέθοδοι που επιτρέπουν στον προγραμματιστή να ελέγχει την επιλογή (όπως οι `select`, `selectAll`) επίσης ελέγχουν το δρομέα με το να τοποθετούν απευθείας το σημάδι και την τελεία.

Η Εργαλειοθήκη Σύνταξης

Αν και δε φάνηκε στην Εικόνα 16, η εργαλειοθήκη σύνταξης παρέχει μεγάλο μέρος της λειτουργικότητας των συστατικών κειμένου. Να σημειωθεί ότι, μία εργαλειοθήκη σύνταξης αντιστοιχεί στον ελεγκτή της αρχιτεκτονικής MVC. Τέσσερις συντακτικές εργαλειοθήκες προσφέρονται από τα πακέτα του Swing :

- `DefaultEditorKit`
- `StyledEditorKit`
- `HTMLEditorKit`

- `RTFEditorKit`

Σε γενικές γραμμές, μία εργαλειοθήκη σύνταξης γνωρίζει πώς να χειρίζεται ένα έγγραφο με ένα ιδιαίτερο είδος περιεχομένου. Η `DefaultEditorKit` χρησιμοποιείται για τα απλά συστατικά κειμένου (τα `JTextField`, `JPasswordField` και `JTextArea`) που χρησιμοποιούν μία μοναδική γραμματοσειρά και ένα μοναδικό χρώμα για το προσκήνιο. Η `StyledEditorKit` είναι πιο πολύπλοκη και χειρίζεται αυθαίρετα χαρακτηριστικά που μπορεί να ισχύουν για όλο το έγγραφο, τις παραγράφους, ή για ένα σύνολο από χαρακτήρες. Αυτή η εργαλειοθήκη χρησιμοποιείται από το `JTextPane` και είναι υπερκλάση και των δύο `HTMLEditorKit` και `RTFEditorKit`, οι οποίες χρησιμοποιούν το κείμενο και κληρονομώντας από την `StyledEditorKit` μπορούν να διαβάσουν και να παραθέσουν `HTML` και `RTF` έγγραφα αντίστοιχα. Αυτές οι δύο συντακτικές εργαλειοθήκες τυπικά συνδέονται με το `JEditorPane`.

Μία εργαλειοθήκη σύνταξης είναι υπεύθυνη για τα επόμενα :

- Να δημιουργεί τον κατάλληλο τύπο της `Document` για να υποστηρίζει τις ίδιες διευκολύνσεις με την εργαλειοθήκη σύνταξης. Το `DefaultStyledDocument` δημιουργεί ένα `PlainDocument`· οι `StlyledEditorKit` και `RTFEditorKit` χρησιμοποιούν και οι δύο το `DefaultStyledDocument`· και η `HTMLEditorKit` χρησιμοποιεί ένα `HTMLDocument`, το οποίο προέρχεται από το `DefaultStyledDocument`.

- Να φορτώνει το περιεχόμενο εγγράφου από μία ροή εισόδου (reader) ή να γράφει περιεχόμενο σε μία ροή εξόδου (writer). Οι μέθοδοι `read` και `write` της εργαλειοθήκης σύνταξης συνήθως καλούνται έμμεσα μέσω των ίδιων μεθόδων του `JTextComponent`. Υπάρχει μία σύνδεση ανάμεσα στην εργαλειοθήκη σύνταξης και στη διάταξη της πληροφορίας στην ροή εισόδου. Για παράδειγμα, η `HTMLEditorKit` περιμένει να διαβάσει από μία ροή εισόδου η οποία περιέχει κείμενο και στοιχεία και γράφει στην έξοδο με την ίδια διάταξη. Οι `DefaultEditorKit` και `StyledEditorKit` χειρίζονται ένα σαφές κείμενο, ώστε να μην μπορούμε να αποθηκεύσουμε τη διάταξη που εφαρμόζεται προγραμματιστικά με ένα `JTextPane` σε έναν φάκελο και να το ανακτήσουμε αργότερα, εκτός κι αν υπερβούμε τη μέθοδο `write` της `StyledEditorKit` και συλλέξουμε τα χαρακτηριστικά σε μία νέα διάταξη και εφαρμόσουμε τη μέθοδο `read` έτσι ώστε να μπορούμε να τα ανακτήσουμε.

- Να δημιουργεί τα αντικείμενα `View` που απαιτούνται για να σχεδιάσουμε τα περιεχόμενα ενός συστατικού κειμένου στην οθόνη. Αυτό γίνεται με κατάλληλες κλάσεις δημιουργίας (`View Factories`) και περιγράφεται στη συνέχεια του κεφαλαίου.

- Να παρέχει ένα σύνολο από συντακτικές κινήσεις που μπορούν να συνδεθούν με τις κινήσεις του πληκτρολογίου που πρόκειται να χρησιμοποιηθούν από τον τελευταίο χρήστη.

Οι Όψεις των Συστατικών Κειμένου

Τα συστατικά κειμένου σχεδιάζονται από αντικείμενα τα οποία προέρχονται από την αφηρημένη κλάση `View`. Υπάρχουν αρκετές υποκλάσεις της `View` που χειρίζονται διάφορες πλευρές της διαδικασίας της σχεδίασης. Οι όψεις που θα χρησιμοποιηθούν για να σχεδιάσουν ένα συγκεκριμένο συστατικό εξαρτώνται από τον τύπο του συστατικού και από το ακριβές περιεχόμενό του. Οι κλάσεις που ονομάζονται `PlainView` και `WrappedPlainView` χρησιμοποιούνται από την `JTextArea`, ενώ δύο άλλες που ονομάζονται `FieldView` και `PasswordView` χρησιμοποιούνται από τις `JTextField` και `JPasswordField`, αντίστοιχα. Για τα απλά συστατικά κειμένου, απαιτείται μόνο ένας τύπος `View` για να χειριστεί ολόκληρο το έγγραφο, και το συστατικό από μόνο του γνωρίζει πώς να δημιουργήσει το δικό του ξεχωριστό `View`.

Κάποια συστατικά διαθέτουν μία ιεραρχία από `Views`, καθένα από τα οποία γνωρίζει πώς να παραθέσει ένα μέρος του περιεχομένου του μηχανισμού ελέγχου. Στην Εικόνα 17 φαίνονται οι πιο σημαντικές κλάσεις `View` του Swing.

ΕΙΚΟΝΑ 17.

FieldView και PasswordView	Αυτές οι όψεις χρησιμοποιούνται από τις κλάσεις `JTextField` και `JPasswordField`, αντίστοιχα. Σχεδιάζουν κείμενο σε μία δοσμένη περιοχή μίας και μοναδικής γραμμής. Η κλάση `FieldView` προέρχεται από την `PlainView`. Η `PasswordView` είναι μία υπερκλάση της `FieldView` που διαφέρει μόνο στο ότι αποδίδει κάθε χαρακτήρα στο μοντέλο χρησιμοποιώντας τον διαμορφωμένο απόηχο χαρακτήτα αντί των αντίστοιχων χαρακτήρων από την επιλεγμένη γραμματοσειρά του συστατικού κειμένου.
PlainView και WrappedPlainView	Τα `PlainView` και `WrappedPlainView` χρησιμοποιούνται από την `JTextArea` και σχεδιάζουν δισδιάστατα κείμενα. Το `PlainView` δεν αναδιπλώνει γραμμές που είναι τόσο μεγάλες ώστε να μη χωρούν μέσα στην ορατή περιοχή που προορίζεται για αυτές. Το `WrappedPlainView` είναι μία υποκλάση του `PlainView` που ταξινομεί το κείμενο με το να το διαχωρίζει με άσπρα-κενά όρια όταν η ταξινόμηση της λέξης χρησιμοποιείται ενώ όταν δε χρησιμοποιείται, όταν η γραμμή έχει γεμίσει.
BoxView	Το `BoxView` είναι ένα κιβώτιο εικόνων που λειτουργεί κάπως όπως το συστατικό `Box Swing`. Η λειτουργία του είναι να διαχειρίζεται τις απεικονίσεις των παιδιών του με το να τις τοποθετεί οριζόντια ή κάθετα. Στην πράξη, αυτή η όψη χρησιμοποιείται ως η κύρια όψη είτε του `JTextPane` είτε του `JEditorPane` και διαχειρίζεται ολόκληρο το έγγραφο τακτοποιώντας τα `ParagraphViews` κάθετα,το ένα πάνω στο άλλο.
ParagraphView	Όπως υποδηλώνει το όνομά του, χρησιμοποιείται για να διαχειριστεί το πλάνο των περιεχομένων της παραγράφου είτε από το `JEditorPane` είτε από το `JTextPane`. Ένα `ParagraphView` δημιουργεί παραγράφους με το να κατασκευάζει γραμμές που απαρτίζονται από `LabelViews, IconViews` και `ComponentViews`.
LabelView	Το `LabelView` είναι η βασική όψη που στην πραγματικότητα αποδίδει κείμενο από το έγγραφο. Ένα `LabelView` μπορεί, μπορεί και όχι, να αποτυπώσει πάνω σε μία μοναδική γραμμή της περιοχής της όψης ενός συστατικού κειμένου. Ωστόσο, ένα `LabelView` δεν είναι ποτέ μεγαλύτερο από μία γραμμή.
IconView	Ένα `IconView` φροντίζει να παραθέτει απεικονίσεις τοποθετημένες σαν ομότυπα της διασύνδεσης `Icon` του Swing.
ComponentView	Όμοια, ένα `ComponentView` φιλοξενεί ένα `Component` και το τοποθετεί στη ροή του εγγράφου. Αν και μπορούμε να τοποθετήσουμε ένα AWT `Component` σε ένα έγγραφο μπορούμε πιθανότατα να χρησιμοποιήσουμε ένα όχι και τόσο σημαντικό συστατικό που προέρχεται από την `JComponent` για να αποφύγουμε συνηθισμένα προβλήματα που κληρονομούνται από το συνδυασμό συστατικών του AWT και του Swing.

Στην περίπτωση των JTextPane και JEditorPane, διαφορετικά στοιχεία του εγγράφου διαχειρίζονται από διαφορετικά αντικείμενα View. Για την αποφυγή δύσκολου κώδικα, εξαιτίας αυτής της σχέσης μέσα στα ίδια τα συστατικά, η εργαλειοθήκη σύνταξης χρησιμοποιείται σαν ένα factory που δημιουργεί τα κατάλληλα αντικείμενα View που βασίζονται στη φύση του τμήματος του εγγράφου που έχει αποδοθεί. Για παράδειγμα, στην περίπτωση του JTextPane, για κάθε παράγραφο του εγγράφου, η StyledEditorKit θα δημιουργήσει ένα ParagraphView για να το αναπαραστήσει, και αυτό το ParagraphView θα προστεθεί σε ένα άλλο View, που ονομάζεται BoxView και χειρίζεται το κάθετο πλάνο του εγγράφου. Κάθε κομμάτι της παραγράφου, με τη σειρά, διευθύνεται από ένα LabelView αν αυτό απαρτίζεται από ξεκάθαρο κείμενο. Τα JTextPane και JEditorPane μπορούν επίσης να περιέχουν Icons και αυθαίρετα Components μαζί με το κείμενο. Όταν μία StyledEditorKit εγκαθίσταται σε ένα από αυτά τα συστατικά, αυτά τα αντικείμενα σχεδιάζονται από Views του τύπου IconView ή ComponentView. Μία συνηθισμένη εργαλειοθήκη σύνταξης μπορεί να έχει προσωπικά αντικείμενα View που αποδίδουν κείμενο, προσωπικά Icons ή Components, και επιστρέφει ομότυπα των δικών της προσαρμοσμένων αντικειμένων από τη δική της μέθοδο factory.

Προσαρμοσμένες Όψεις Εγγράφων

Το περιεχόμενο του εγγράφου και τα σχετικά χαρακτηριστικά στο μοντέλο του, καθορίζουν τί ακριβώς θα έπρεπε να παραθέτει ένα συστατικό κειμένου και με ποιο τρόπο θα έπρεπε να το παραθέτει, αλλά η δουλειά της απόδοσης του κειμένου εκπληρώνεται στην πραγματικότητα από πλήθος οπτικών αντικειμένων που γνωρίζουν πώς να ερμηνεύουν το περιεχόμενο του κειμένου. Οι όψεις είναι υπεύθυνες για να διατάσσουν το κείμενο στην οθόνη, να το τοποθετούν σε παραγράφους, να εκπληρώνουν αναδίπλωση λέξεων, να υποστηρίζουν περιθώρια, και να αλλάζουν χρώμα όταν χρειάζεται. Αν οι κλασικές όψεις δεν ανταποκρίνονται στις ανάγκες μας, μπορούμε να δημιουργήσουμε τις δικές μας, που να εκτελούν συνήθη απόδοση. Για να το κάνουμε αυτό χρειάζεται να καταλάβουμε πώς τα χαρακτηριστικά είναι τοποθετημένα στο μοντέλο εγγράφου.

Πώς Τοποθετούνται τα Χαρακτηριστικά

Όπως έχουμε δει, η πληροφορία σε ένα συστατικό κειμένου ελέγχεται από την διασύνδεση Document, αλλά δεν είναι μόνο αυτή που τοποθετεί η Document. Το να ελέγχει την πληροφορία μόνο, δεν είναι επαρκές, ακόμα και για το πιο απλό συστατικό κειμένου. Το περιεχόμενο από ένα JTextArea, για παράδειγμα, είναι χωρισμένο σε ξεχωριστές γραμμές από τους χαρακτήρες αλλαγής γραμμής, επομένως είναι απαραίτητο να θυμόμαστε, όχι μόνο το κείμενο, αλλά επίσης και που είναι τα όρια των γραμμών. Στην περίπτωση των JEditorPane και JTextPane, το κείμενο είναι ταξινομημένο σε παραγράφους και μπορεί να έχει αυθαίρετη διάταξη περιεχομένου που να συσχετίζεται με αυτό. Αντί να προσπαθούμε να εμφυτεύσουμε την κατασκευαστική πληροφορία μέσα στον υποδοχέα που χρησιμοποιείται για να ελέγχει το κείμενο, οι υλοποιήσεις της Document που υποστηρίζονται από το πακέτο του Swing, διατηρούν μία παράλληλη κατασκευή των *στοιχείων* (elements) που οργανώνει το κείμενο σε γραμμές ή παραγράφους και παρέχει την τοποθέτη των σχετικών χαρακτηριστικών. Οι λεπτομέρειες για την κατασκευή των στοιχείων διαφέρουν από συστατικό σε συστατικό.

Όψεις : Αποδίδοντας το Έγγραφο και τα Χαρακτηριστικά του

Το μοντέλο του εγγράφου περιέχει και το ακατέργαστο κείμενο και τα κατασκευαστικά στοιχεία που απαιτούνται για να αναπαράγουν το περιεχόμενο του συστατικού κειμένου με ένα τρόπο που ο προγραμματιστής θα περίμενε. Ωστόσο, το μοντέλο δεν προσδιορίζει κατευθείαν τον τρόπο με τον

οποίο το συστατικό εμφανίζεται στην οθόνη. Αυτή η δουλειά γίνεται από τα οπτικά αντικείμενα, τα οποία εξηγούν την κατασκευή των στοιχείων για να διευθετήσουν το κείμενο σε παραγράφους και γραμμές και χρησιμοποιούν χρώματα, γραμματοσειρές και άλλα επακόλουθα που είναι ορισμένα από τα χαρακτηριστικά μέσα στα στοιχεία. Στη συνέχεια θα δείξουμε πώς τα κλασικά συστατικά αποτυπώνουν αντικείμενα `View` πάνω στο μοντέλο που μας ενδιαφέρει και πώς να δημιουργούμε τις δικές μας όψεις για να βελτιώσουμε την απόδοση των πεδίων κειμένου και του `JTextPane`. Τις τεχνικές που θα χρησιμοποιήσουμε θα τις δείξουμε μέσα σε ένα πλαίσιο για συγκεκριμένα συστατικά κειμένου, αλλά είναι γενικές και γι αυτό μπορούν να χρησιμοποιηθούν από όλους τους ελεγκτές κειμένου.

Η Ιεραρχία των Όψεων

Υπάρχουν δύο βασικοί τύποι για όψεις, εκείνοι που συμπεριφέρονται σαν υποδοχείς και εκείνοι που απλά αποδίδουν κείμενο ή άλλο περιεχόμενο. Αυτό κάνει τις όψεις ανάλογες με τα συστατικά του AWT, μερικά από τα οποία είναι επίσης υποδοχείς. Η ομοιότητα επίσης απευθύνεται στον τρόπο με τον οποίο οι όψεις ενεργούν επειδή, για παράδειγμα, μία όψη μπορεί να διερωτηθεί για το κατάλληλο του μεγέθους της και επακολούθως θα δοθεί μία ορθογώνια περιοχή μέσα στην οποία θα δουλέψει και μέσα στην οποία θα αποδόσει το περιεχόμενό της. Ωστόσο, αντίθετα από το AWT δεν υπάρχουν διαφορετικοί διαχειριστές σχεδίων που μπορούμε να τους προσδιορίσουμε σαν απεικονίσεις "υποδοχείς"· μία απεικόνιση περιέχει τον κώδικα, ο οποίος κάνει τη δουλειά η οποία θα αποτύγχανε αν γινόταν με ένα διαχειριστή πλάνου.

Οι όψεις συνδέονται με τα στοιχεία και όπως ισχύει για την αρχιτεκτονική των στοιχείων, έτσι και στις όψεις υπάρχει ιεραρχία. Στην πιο απλή πιθανή περίπτωση, ένα οπτικό αντικείμενο θα αποτυπώσει ένα μόνο στοιχείο, αλλά αυτό δε συμβαίνει πάντα – κάποιες φορές ένα μόνο στοιχείο κατέχει περισσότερες από μία όψεις που σχετίζονται με αυτό. Ωστόσο, καμία όψη δε σχετίζεται ποτέ με περισσότερα από ένα στοιχεία. Μία επίπτωση αυτού του γεγονότος, είναι ότι η περιοχή του κειμένου που καλύπτεται από μία μοναδική, βασική όψη (μία όψη που δεν είναι και υποδοχέας) έχει, πάντα, μόνο ένα σύνολο χαρακτηριστικών, επειδή ένα στοιχείο έχει μόνο ένα (λογικό) σύνολο από γνωρίσματα. Αναφορικά στην Εικόνα 18, φαίνεται μία λίστα από τις κλασικές όψεις του πακέτου κειμένου του Swing και δείχνει ποιες είναι υποδοχείς και ποιες είναι βασικές όψεις, και τα συστατικά κειμένου με τα οποία συσχετίζονται. Να σημειωθεί ότι ο όρος βασική όψη θα χρησιμοποιείται για τις απεικονίσεις που δεν είναι υποδοχείς.

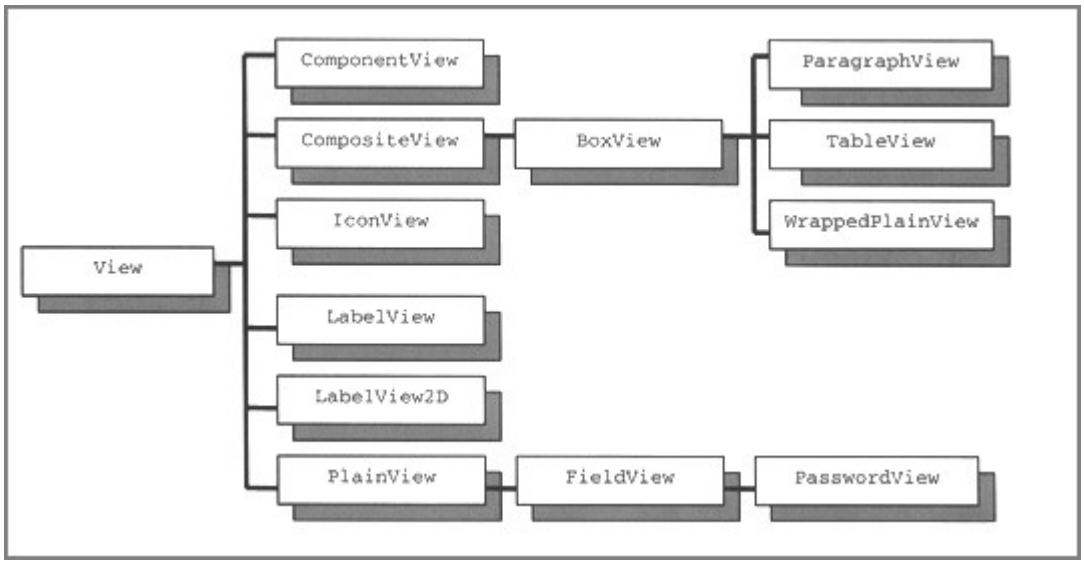
Να σημειώσουμε,επίσης, ότι κάποιες από τις όψεις του πίνακα είναι εσωτερικές κλάσεις άλλων όψεων. Για παράδειγμα η όψη υποδοχέας `ParagraghView` έχει μία εσωτερική κλάση που ονομάζεται `ParagraghView.Row`, η οποία επίσης είναι υποδοχέας. Αυτές οι εσωτερικές κλάσεις των απεικονίσεων χρησιμοποιούνται για τη διαχείριση ενός μικρότερου τμήματος του στοιχείου στο οποίο η κύρια όψή έχει αποτυπωθεί.

ΕΙΚΟΝΑ 18.

View	Type	Text Component
BoxView	Container	Generic
CompositeView	Container	Generic
ComponentView	Basic	JEditorPane/JTextPane
FieldView	Basic	JTextField
IconView	Basic	JEditorPane/JTextPane
LabelView	Basic	JEditorPane/JTextPane
LabelView.LabelFragment	Basic	JEditorPane/JTextPane
ParagraphView	Container	JEditorPane/JTextPane
ParagraphView.Row	Container	JEditorPane/JTextPane
PasswordView	Basic	JPasswordField
PlainView	Basic	JTextArea
TableView	Container	JEditorPane/JTextPane
TableView.TableRow	Container	JEditorPane/JTextPane
TableView.TableCell	Container	JEditorPane/JTextPane
TableView.ProxyCell	Basic	JEditorPane/JTextPane
WrappedPlainView	Container	JTextArea
WrappedPlainView.WrappedLine	Basic	JTextArea

Η ιεραρχία των κλάσεων των συνηθισμένων όψεων φαίνεται στη Εικόνα 19. Για συντομία, αυτό το διάγραμμα δεν απεικονίζει τις εσωτερικές κλάσεις. Ας προσέξουμε ότι όλες οι όψες υποδοχείς προέρχονται από το `BoxView`, που είναι μία υποκλάση του `CompositeView`. Το `CompositeView` εφαρμόζει τις μεθόδους οι οποίες απαιτούνται για την οργάνωση των όψειων των παιδιών, χωρίς να έχουν κάποια συγκεκριμένη σχέση μεταξύ τους (και τις οποίες αναπτύσσουμε αργότερα), ενώ το `BoxView` οργανώνει τα παιδιά του είτε οριζόντια είτε κάθετα. Το `ParagraphView` είναι, με τη σειρά του, μία ειδική μορφή του `BoxView` που διευθετεί τις γραμμές σε μία κάθετη διάταξη για τη διαμόρφωση μίας παραγράφου.

ΕΙΚΟΝΑ 19.



Η Κλάση View

Όπως έχουμε πει, όλες οι απεικονίσεις προέρχονται από την αφηρημένη κλάση `View`. Για να κατασκευάσουμε μία κλάση `View`, πρέπει να έχουμε ένα στοιχείο με το οποίο να συσχετίζεται. Συνήθως, μία κλασική υποκλάση της `View` θα συσχετίζεται με ένα συγκεκριμένο τύπο στοιχείου. Τα `Views` δημιουργούνται από ένα `ViewFactory`, που γνωρίζει πως να δημιουργήσει τα κατάλληλα `Views` για τους τύπους των στοιχείων που περιέχονται μέσα στο έγγραφο. Στη συνέχεια θα μελετήσουμε κάποιες από τις μεθόδους της κλάσης `View`.

Οι μέθοδοι `setParent` και `getParent` εγκαθιστούν και ανακτούν, αντίστοιχα, το `View` που εκτείνεται πάνω από το επιδιωκόμενο `View` στην ιεραρχία των `Views`. Ο γονιός ενός `View` έχει εγκαταστηθεί από την ώρα που η ιεραρχία έχει κατασκευαστεί, κάτι το οποίο συνήθως συμβαίνει όταν το περιεχόμενο προστίθεται στο έγγραφο.

Οι μέθοδοι `getDocument, getStartOffset, getEndOffset, getElement, getContainer` και `getViewFactory`, όλες επιστρέφουν χαρακτηριστικά του `View`. Τα αχικά και τελικά offsets μπορεί κάποιες φορές να είναι τα αντίστοιχα offsets του στοιχείου που το `View` αποτυπώνει, αλλά αυτό δεν είναι απαραίτητο, επειδή κάποιες φορές περισσότερα από ένα `View` μπορεί να χρησιμοποιηθούν για να διοικήσουν ένα μοναδικό στοιχείο. Τα offsets που συσχετίζονται με το `View` είναι πάντα οριοθετημένα από μία σειρά από offsets του στοιχείου και επιπλέον, όχι περισσότεροι από ένα μη-υποδοχέα `View` καλύπτουν μία δοσμένη σειρά από offsets. Η μέθοδος `getElement` επιστρέφει το αποτυπωμένο στοιχείο, ενώ η μέθοδος `getContainer` επιστρέφει το συστατικό κειμένου που διοργανώνει την απεικόνιση. Αυτό μπορεί να χρησιμοποιηθεί, για παράδειγμα, για να επιστρέψουμε στο `JTextPane` ή στο `JTextField` μέσα από ένα `View`.

Η μέθοδος `getViewFactory` επιστρέφει το `ViewFactory` το οποίο δημιούργησε το `View`. Επειδή μόνο ένα `ViewFactory` χρησιμοποιείται για να παραχθούν όλα τα `Views` για ένα συγκεκριμένο συστατικό, η υλοποίηση που υποκαθιστά αυτή τη μέθοδο απλά αναθέτει αρμοδιότητες στο γονιό της, με την προϋπόθεση ότι το `View` που είναι στην καρδιά της ιεραρχίας θα γνωρίζει πώς να τοποθετεί το `ViewFactory` που το δημιούργησε.

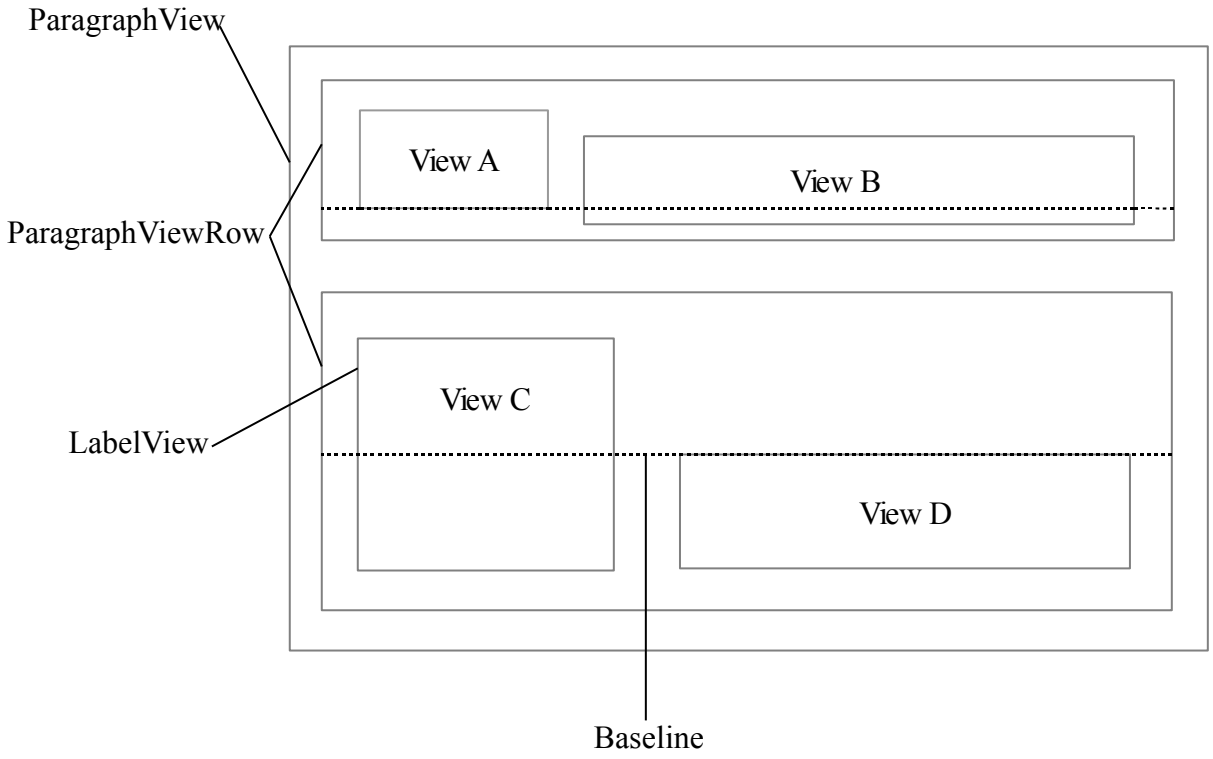
Η μέθοδος `getAtrributes` επιστρέφει τα σχετικό `AtrributeSet` του `View`. Αυτά τα χαρακτηριστικά είναι ουσιαστικά τα χαρακτηριστικά του στοιχείου που το `View` αποτυπώνει και έχουν αποκτηθεί απευθείας από το στοιχείο. Μπορούμε, βέβαια, να δημιουργήσουμε τα δικά μας αντικείμενα `View` που θα επιστρέφουν ένα διαφορετικό `AtrributeSet` που μπορεί είτε να είναι είτε όχι σχετικό με εκείνα του συγκεκριμένου στοιχείου. Αυτή η τεχνική χρησιμοποιείται από τα `Views` που χρησιμοποιούνται για να αποδόσουν αρχεία `HTML`.

Η μέθοδος `isVisible` επιστρέφει αληθές όταν το `View` είναι ορατό και ψευδές όταν δεν είναι. Όλα τα κλασικά `Views` αυτού του πακέτου απλά χρησιμοποιούν την υλοποίηση που υποκαθιστά αυτή τη μέθοδο, η οποία επιστρέφει πάντα αληθές. Κάποια από τα `Views` στο πακέτο `HTML`, ωστόσο, αυτό το προσπερνούν και μπορεί να ισχυριστούν ότι δεν είναι ορατό.

Η μέθοδος `getAlignment` δέχεται ένα όρισμα που προσδιορίζει αν η στοίχιση έχει ζητηθεί να γίνει πάνω στον άξονα των `x` (`View.X_AXIS`) ή στον άξονα των `y` (`View.Y_AXIS`). Για τα συστατικά του AWT και του Swing, η τιμή της στοίχισης κυμαίνεται ανάμεσα στο 0.0 και στο 1.0 και προσδιορίζει τη σχετική τοποθεσία του σημείου της στοίχισης του `View` πάνω στον καθορισμένο άξονα. Η τιμή 0.0 της στοίχισης πάνω στον `y` άξονα τοποθετεί το σημείο της στοίχισης στην κορυφή του `View` ενώ η τιμή 1.0 το τοποθετεί στο τέλος. Όμοια, η τιμή 0.0 της στοίχισης πάνω στον `x` άξονα τοποθετεί το σημείο της στοίχισης στο αριστερό μέρος του `View` ενώ η τιμή 1.0 το τοποθετεί στη δεξιά μεριά. Όταν η μέθοδος επιστρέφει την τιμή 0.5 και για τους δύο άξονες το `View` μπαίνει στο κέντρο του οποιουδήποτε υποδοχέα μέσα στον οποίο έχει τοποθετηθεί. Συχνά, ένα `View` υποδοχεάς μπορεί να ενδιαφέρεται για την στοίχιση μόνο προς μία κατεύθυνση. Για παράδειγμα, όταν ένα `ParagraphView` παρατάσσει μία γραμμή, ενδιαφέρεται μόνο για την στοίχιση κατά τον `y` άξονα, για να καθορίσει τον τρόπο με τον οποίο θα τοποθετήσει τα `Views` των παιδιών του σχετικά με την *βασική γραμμή* (baseline) της κάθε γραμμής μέσα στην παράγραφο. Η Εικόνα 20 δείχνει ένα παράδειγμα κατά το οποίο το `ParagraphView` παρατάσσει δύο γραμμές περιεχομένου, του οποίου διαφορετικά επιμέρους κομμάτια έχουν τη δική τους απαίτηση στοίχισης.

Το σχήμα απεικονίζει το `ParagraphView`, το οποίο τοποθετεί κάθε γραμμή του περιεχομένου μέσα στην υποκλάση του `ParagraphView.Row` και τις διατάσσει τη μία πάνω από την άλλη. Μέσα σε κάθε γραμμή, τα μέρη του περιεχομένου τοποθετούνται οριζόντια. Μπορούμε να φανταστούμε το `ParagraphView` σαν ένα κουτί το οποίο παρατάσσει τα παιδιά κατά τον `y` άξονα, ενώ ένα `ParagraphView.Row` είναι ένα κουτί το οποίο παρατάσσει υποκλάσεις όψεων κατά τον `x` άξονα. Στην πραγματικότητα, αυτός είναι ο τρόπος με τον οποίο δουλεύει, καθώς και το `ParagraphView` και το `ParagraphView.Row` προέρχονται από το `BoxView`, το οποίο μπορεί να κατασκευαστεί για να διαχειριστεί είτε μία κάθετη (`View.Y_AXIS`) είτε μία οριζόντια (`View.X_AXIS`) διάταξη. Μέσα σε κάθε γραμμή, υπάρχουν όψεις που αποδίδουν περιεχόμενο. Στην περίπτωση του `JTextPane`, αυτές οι όψεις θα μπορούσαν, πιθανώς, να αποτελούν ομότυπα του `LabelView`. Αυτή η κάθετη τοποθέτηση αυτών των όψεων περιεχομένου, ελέγχεται από την τιμή την οποία επιστρέφει η μέθοδος `getAlignment` (`View.Y_AXIS`). Η όψη με την ονομασία `View A`, έχει το δικό της σημείο στοίχισης στο χαμηλότερό της όριο και γι αυτό το λόγο αυτό το όριο τοποθετείται πάνω στην βασική γραμμή. Για να το πετύχουμε αυτό, η μέθοδος `getAlignment` θα πρέπει να επιστρέψει 1.0. Η όψη με την ονομασιά `View B`, αντιθέτως, έχει το δικό της σημείο στοίχισης χαραγμένο ανάμεσα στο χαμηλότερο και στο υψηλότερο όριό της ˙πιο κοντά στο κάτω μέρος και όχι στην κορυφή. Η τιμή της στοίχισης αυτής της όψης κυμαίνεται ανάμεσα στο 0.5 και στο 1.0, ανάλογα με το πόσο μέρος της όψης υπάρχει κάτω από την βασική γραμμή. Στη δεύτερη γραμμή η όψη `View C` είναι κεντραρισμένη πάνω στη βασική γραμμή καθώς η στοίχιση κατα τον `y` άξονα είναι 0.5, ενώ η όψη `View D` έχει στοιχιστεί με το υψηλότερο όριό της να βρίσκεται πάνω στη βασική γραμμή, καθώς η δική της τιμή στοίχισης είναι το 0.0.

ΕΙΚΟΝΑ 20.



Στην πράξη, οι κλασικές όψεις δεν επιτρέπουν στις στοιχίσεις τους να είναι εύκολα προσαρμόσιμες. Το `LabelView`, το οποίο αποδίδει κείμενο για το `JTextPane`, χρησιμοποιεί την αναλογία του μεγαλύτερου ύψους της γραμματοσειράς με το μικρότερο για να καθορίσει την στοίχιση κατά τον `y` άξονα. Με αυτό τον τρόπο τοποθετείται η βασική γραμμή του `View` στο ίδιο μέρος με την βασική γραμμή της γραμματοσειράς, έτσι ώστε ο χώρος να κατανέμεται σωστά για να σχεδιαστούν τα κατώτερα μέρη των χαρακτήρων της γραμματοσειράς που τους περιέχει. Για να αλλάξουμε αυτή τη συμπεριφορά θα πρέπει να κατασκευάσουμε τη δική μας υποκλάση του `LabelView` και να υπερβούμε την μέθοδό `getAlignment` της κλάσης `LabelView`.

Οι μέθοδοι `getViewCount` και `getView` χρησιμοποιούνται από όψεις οι οποίες συμπεριφέρονται σαν υποδοχείς και επιτρέπουν στο σύνολο των `Views` των παιδιών, να έχουν πρόσβαση σε αυτές. Επειδή αυτές οι μέθοδοι υπάρχουν σε όλα τα `Views`, καθένα από τα `Views` μπορεί να είναι υποδοχέας, οπότε θεωρητικά μπορούμε μέσω του κώδικά μας να δημιουργήσουμε έναν υποδοχέα `View` χώρις αυτό να είναι υποκλάση του `CompositeView`, που είναι ο κλασικός υποδοχέας στο πακέτο κειμένου. Ωστόσο, αυτό θα μπορούσαμε πιθανώς να το κάνουμε μόνο αν χρειαζόταν να δημιουργήσουμε έναν υποδοχέα `View` ο οποίος έχει ιδιαίτερες απαιτήσεις οι οποίες, με τη σειρά τους, δε μπορούν να επιτευχθούν με το να είναι αυτό το `View` υποκλάση του `CompositeView`. Οι προεπιλεγμένες υλοποιήσεις του `View` για αυτές τις μεθόδους επιστρέφουν 0 και `null` αντίστοιχα.

Οι επόμενες τρείς μέθοδοι, `getPreferredSpan`, `getMinimumSpan` και `getMaximumSpan` είναι ανάλογες με τις αντίστοιχες του AWT `getPreferredSpan`, `getMinimumSpan` και `getMaximumSpan`, εκτός από το γεγονός ότι αυτές ασχολούνται με μόνο μία διάσταση τη φορά. Κάθε μέθοδος δέχεται ένα όρισμα (είτε `View.X_AXIS` είτε `View.Y_AXIS`) που προσδιορίζει την κατεύθυνση για την οποία απαιτείται το μέγεθος. Οι τιμές που επιστρέφονται από αυτές τις μεθόδους χρησιμοποιούνται από

το γονιό υποδοχέα `View` για να αποφασίσει πόσο μεγάλος χρειάζεται να είναι για να στεγάσει τις όψεις των παιδιών του. Η λειτουργικότητα που παρέχεται από τους ελεγκτές σχεδίου όσον αφορά τα συστατικά του AWT και του Swing, κωδικοποιείται κατευθείαν μέσα στους υποδοχείς `Views` και έτσι δεν μπορούν να αλλάξουν με την εφαρμογή μίας διαφορετικής πολιτικής σχεδίου. Αυτό συνήθως, δεν αποτελεί επίμαχο θέμα, καθώς οι απαιτήσεις πλάνου των επιμέρους κομματιών ενός συστατικού κειμένου είναι ευδιακρίτως προσδιορισμένες.

Να σημειωθεί ότι αυτές οι τρείς μέθοδοι, επιστρέφουν τιμές πραγματικών κινητής υποδιαστολής και όχι ακεραίων. Ο σκοπός είναι ότι οι μετρήσεις μεγέθους θα δωθούν σε σημεία και όχι σε στοιχεία οθόνης (pixels), οπότε μπορούμε να αποδόσουμε έγγραφα με έναν τρόπο που δεν εξαρτάται από την ευκρίνεια της οθόνης ή του εκτυπωτή. Στην πράξη, στη Java 2, τα συστατικά κειμένου αποτυπώνουν σημεία κατευθείαν σε στοιχεία οθόνης, οπότε είναι συνηθισμένο να δούμε κώδικα που μετατρέπει την επιστρεφόμενη τιμή αυτών των μεθόδων κατευθείαν σε ακεραίους.

Η μέθοδος `getResizeWeight` χρησιμοποιείται για να υποδείξει αν το `View` θα ήθελε να ξαναλλάξει μέγεθος ώστε να διαθέσει χώρο. Αν αυτή η μέθοδος επιστρέψει 0, το `View` θα προσφέρεται πάντα στο επιθυμητό του μέγεθος και η μέθοδος `getMinimumSpan` θα επιστρέψει την ίδια τιμή με την `getPreferredSpan`.

Η μέθοδος `setSize` καλείται από τον άμεσο γονιό του `View` για να τον ενημερώσει για τις διαστάσεις του χώρου που προορίζεται για αυτόν. Σε απάντηση αυτού, ένας υποδοχέας `View` συνήθως καθορίζει το καινούριο πλάνο των παιδιών του, το οποίο θα έχει σαν αποτέλεσμα την κλήση της δικιάς τους μεθόδου `setSize`. Οι μη- υποδοχείς `Views` συνήθως δεν κάνουν τίποτα όταν καλείται αυτή η μέθοδος καθώς η δική τους κύρια (main) συνάρτηση υπάρχει απλά για να ζωγραφίσει το περιεχόμενό τους, κάτι το οποίο θα συμβεί όταν καλείται η συνάρτηση `paint` από το `View` του παιδιού.

Η μέθοδος `getChildAllocation` εφαρμόζεται μόνο από όψεις υποδοχείς και επιστρέφει το χώρο που προορίζεται για την όψη του παιδιού, που δίνεται από τον τακτικό του αριθμό. Σε ένα μη-υποδοχέα `View`, αυτή η μέθοδος επιστρέφει την τιμή `null`. Η διάταξη των όψεων εξαρτάται από ένα συγκεκριμένο υποδοχέα· για παράδειγμα, το `ParagraphView` τοποθετεί τις γραμμές από την κορυφή προς τα κάτω, έτσι ώστε ο δείκτης 0 να αντιστοιχεί στην πρώτη γραμμή της παραγράφου, ο δείκτης 1 στην επόμενη και ούτω καθεξής. Αντίθετα, το `ParagraphView.Row` κατασκευάζει ένα οριζόντιο πλάνο κατά το οποίο ο δείκτης 0 αντιστοιχεί στο πιο αριστερό συστατικό, ο δείκτης 1 στο αμέσως επόμενο προς τα δεξιά και ούτω καθεξής. Το δεύτερο όρισμα αυτής της μεθόδου και η επιστρεφόμενη τιμή είναι τύπου `Shape`. Το `Shape` είναι μία διασύνδεση που καθορίζεται από το πακέτο AWT και μπορεί να αναπαραστήσει γραφικά με οποιοδήποτε περίγραμμα. Με δεδομένο το `Shape`, μπορούμε να χρησιμοποιήσουμε τη μέθοδο `getBounds` για να πάρουμε το μικρότερο `Rectangle` που περιβάλλει ολοκληρωτικά το σχήμα. Όλα τα κλασικά `Views` κειμένου έχουν να κάνουν με ορθογώνιες περιοχές, έτσι ώστε το όρισμα `Shape` αυτής της μεθόδου (και όλων των άλλων μεθόδων που ορίζουν ένα `Shape`) θα είναι πάντα ένα `Rectangle` όταν χρησιμοποιούνται τα κλασικά `Views`. Το όρισμα `Shape` της `getChildAllocation` είναι στην πραγματικότητα η περιοχή που αναθέτει το `View`. Αυτή η τιμή μπορεί να χρησιμοποιηθεί για να υπολογίσουμε την περιοχή που αντιστοιχεί στο δεδομένο παιδί, αν αυτή η πληροφορία δεν είναι αποθηκευμένη μέσα στον υποδοχέα `View`.

Η μέθοδος `preferenceChanged` ενός υποδοχέα `View` καλείται από ένα `View` παιδιού όταν το πλάτος και το ύψος αυτού του παιδιού μπορεί να έχει αλλάξει λόγω του ότι, για παράδειγμα, μπορεί να γίνει μία αλλαγή στη γραμματοσειρά μέσα στην περιοχή που καλύπτει το `View`. Οι τιμές *πλάτος* (width) και *ύψος* (height) υποδεικνύουν την πλευρά που μπορεί να έχει αλλάξει το επιθυμητό της

μέγεθος, ενώ το όρισμα του παιδιού προσδιορίζει το παιδί που έκανε την κλήση. Δε χρειάζονται όλοι οι υποδοχείς `Views` την πληροφορία που παρέχεται από το όρισμα του παιδιού.

Η μέθοδος `paint` είναι, όπως δηλώνει και το όνομά της, το μέρος όπου το `View` αποδίδει το περιεχόμενό του· παίρνει για ορίσματα ένα αντικείμενο `Graphics` για να σχεδιάσει μέσα του, και ένα `Shape` το οποίο περιγράφει την περιοχή που προορίζεται για το `View`. Συνήθως ένας υποδοχέας `View`, όπως το `ParagraphView` εφάρμοζει τη μέθοδο `paint` με το να την καλεί για κάθε ένα από τα παιδιά του. Αυτή η διαδικασία μπορεί να συνεχίζεται αν υπάρχουν πολλά επίπεδα από φωλιασμένα `Views`. Τέλος, η μέθοδος `paint` ενός `View` που αποδίδει περιεχόμενο, όταν κληθεί κάτι θα σχεδιαστεί πάνω στην οθόνη.

Η μέθοδος `getNextVisualPositionFrom` συνήθως χρησιμοποιείται όταν μετακινείται ο κέρσορας πάνω, κάτω, αριστερά ή δεξιά. Το όρισμα `pos` είναι το offset της τρέχουσας τοποθεσίας του κέρσορα μέσα στο έγγραφο, ενώ το `direction`, που δέχεται μία από τις τιμές `SwingConstants.NORTH, SOUTH, EAST` ή `WEST`, προσδιορίζει προς τα που πηγαίνει ο κέρσορας. Η επιστρεφόμενη τιμή είναι το offset μέσα στο μοντέλο που αντιστοιχεί στη νέα τοποθεσία του κέρσορα. Επειδή αυτή η μέθοδος απαντιέται σε ένα συγκεκριμένο `View` που αποτυπώνει μόνο ένα μέρος του μοντέλου, είναι πιθανό η επιθυμητή τοποθεσία να μην βρίσκεται μέσα στο στοιχείο που το `View` αποτυπώνει. Για παράγειγμα, καθώς το `LabelView` καλύπτει μέρος μίας μοναδικής γραμμής του κειμένου (ή και ολόκληρη) το αποτέλεσμα του να μετακινηθεί ο κέρσορας πάνω (`NORTH`) ή κάτω (`SOUTH`) θα είναι το να τον μετακινήσει έξω από το τρέχον `View`. Αν η νέα τοποθεσία δεν βρίσκεται ανάμεσα στα σύνορα του `View` η τιμή που θα επιστραφεί θα είναι η -1. Όταν κάτι τέτοιο συμβαίνει το `View` του γονιού είναι υπεύθυνο για να ξανααναθέσει την απαίτηση σε πιο κατάλληλα `Views` των παιδιών αυτή τη φορά.

Οι μέθοδοι `modeltoView` και `viewtoModel` αποτυπώνουν κατευθείαν ανάμεσα στις θέσεις του μοντέλου και στις τοποθεσίες του `View`. Επειδή ο τρόπος κατά τον οποίο ένα `View` αποδίδει το περιεχόμενο του μοντέλου είναι συγκεκριμένος για το `View`, μόνο το `View` μπορεί να γνωρίζει πώς να αποτυπώσει από την τοποθεσία της οθόνης στη θέση του μοντέλου που τροφοδοτήθηκε ο χαρακτήρας ή κάποια άλλη πληροφορία που έχει αποδοθεί σε αυτή την τοποθεσία. Για αυτό το λόγο η κλήση της `viewtoModel` γίνεται συνήθως από το `View` που σχεδίασε το περιεχόμενο στη θέση που δόθηκε από τις τιμές `x` και `y` που προμηθεύτηκε. Όμοια, η πρώτη μορφή της μεθόδου `modeltoView` ανατίθεται στο `View` στην τοποθεσία που αντιστοιχεί στο όρισμα `pos`. Όταν χρησιμοποιείται η δεύτερη μορφή της `modeltoView`, όλα τα `Views` με τη σειρά από το p0 στο p1, καλούν τη δική τους `modeltoView (int pos, Shape a, Position.Bias b)` και ένα `Shape` που αντιστοιχεί σε ολόκληρη την περιοχή που καλύπτεται από τη σειρά δημιουργείται, σαν μία ένωση από όλα τα `Shapes` που αντιστοιχούν στα `Views` των παιδιών. Αυτό, για παράδειγμα, μπορεί να χρησιμοποιηθεί για να ληφθεί ένα `Rectangle` που καλύπτει δύο γραμμές τις παραγράφου σαν μία ένωση από `Rectangles` των ξεχωριστικών γραμμών.

Οι μέθοδοι `insertUpdate, removeUpdate` και `changedUpdate` καλούνται όταν λαμβάνει χώρα μία αλλαγή μέσα στο έγγραφο, η οποία αποτυγχάνει μέσα στην περιοχή που έχει αποτυπωθεί από το `View`. Συνήθως, αυτές οι μέθοδοι απαντούν με το να δημιουργούν ή να διαγράφουν `Views` ή με το να αλλάζουν τα χαρακτηριστικά που συσχετίζονται με το υπάρχον `View`. Τα ορίσματα για όλες αυτές τις μεθόδους είναι ένα `DocumentEvent` το οποίο περιγράφει την αλλαγή στο έγγραφο, ένα `Shape` το οποίο προσδιορίζει το χώρο που προορίζεται για το `View` που θα προκύψει, και το `ViewFactory` το οποίο χρησιμοποιείται για να δημιουργεί τα `Views` για το συστατικό κειμένου που φανερώνει το έγγραφο. Αυτό το `factory` χρησιμοποιείται για να δημιουργήσει τα `Views` αν αυτό είναι απαραίτητο.

Οι μέθοδοι `breakView` και `getBreakWeight` χρησιμοποιούνται για να οργανώσουν τη διαδοχή του κειμένου (και άλλο περιεχόμενο αν αυτό απαιτείται) σε μονάδες που μπορούν να χωρέσουν στην υπάρχουσα γραμμή. Οι παράγραφοι χτίζονται από γραμμές, το πλάτος των οποίων, οριοθετείται από το πλάτος του χώρου που προορίζεται για το συστατικό κειμένου. Το επιθυμητό πλάτος του κειμένου, μέσα σε ένα συγκεκριμένο `Element` μπορεί να μην έχει καμία σχέση με το χώρο που είναι διαθέσιμος για να το αποδόσει. Αν το περιεχόμενο του `Element` είναι πιο μεγάλο από το χώρο που διατίθεται για την τρέχουσα γραμμή, πρέπει να διασπαστεί σε ένα κατάλληλο σημείο και κάποιο από το περιεχόμενο να μετακινηθεί στην επόμενη γραμμή του κειμένου. Η μέθοδος `breakView` έχει τη δουλειά του να δημιουργεί ένα `View` που αποτυπώνει τόσο από το `Element` ώστε να χωράει σε ότι απομένει από την τρέχουσα γραμμή. Αν το `Element` είναι μεγάλο, αυτή η διαδικάσία διάσπασης μπορεί να συνεχίζεται για αρκετές γραμμές. Η μέθοδος `getBreakWeight` καθορίζει πόσο ευνοϊκό είναι να σπάσει το `View` που απαντιέται σε ένα offset το οποίο δίνεται από το όρισμα `pos`. Κάποια `Views` (όπως τα `IconView` και `ComponentView`) δε μπορούν να διασπαστούν, οπότε επιστρέφουν την τιμή `BadBreakWeight`. Οι όψεις κειμένου προτιμούν να διασπάζονται σε σύνορα από κενά και επιστρέφουν `ExcellentBreakWeight` αν η προτεινόμενη τοποθεσία συμπίπτει με το κενό, αλλιώς επιστρέφουν `GoodBreakWeight`. Κάποια `Views` χρειάζονται υποχρεωτική διάσπαση αφού έχουν αποδόθει και επιστρέφουν `ForcedBreakWeight` για να το υποδηλώσουν. Συνήθως η διάσπαση λαμβάνει χώρα κατά τον `x` άξονα, αλλά θα μπορούσε (θεωρητικά τουλάχιστον) να συμβεί και κατά τον `y` άξονα, αν ένα συστατικό κειμένου είχε αποδοθεί μέσα σε ένα περιβάλλον που θα είχε την αντίληψη του μεγέθους της σελίδας, στην οποία περίπτωση θα ήταν απαραίτητο να διασπάσει ένα `ParagraphView` έτσι ώστε να περιεχόταν ολόκληρο μέσα στον διαθέσιμο χώρο της προκειμένης σελίδας. Αν και η δομή υπάρχει για να το υποστηρίζει, η προκείμενη υλοποίηση του `ParagraphView` δεν διαχωρίζει τον εαυτό της στον `y` άξονα. Αν χρειαζόμαστε αυτή τη λειτουργικότητα, θα πρέπει να δημιουργήσουμε μία όψη που θα προέρχεται από την όψη `ParagraphView`.

Ενώ το να διασπάμε ένα `View` σε τεμάχια συνήθως χρειάζεται λόγω του περιορισμένου χώρου, ένας άλλος λόγος για να το κάνουμε αυτό είναι η παρουσία του δισκατευθυνόμενου κειμένου. Όταν ένα `Element` περιέχει κείμενο που ρέει σε δύο κατευθύνσεις, αποτυπώνεται από ένα σύνολο από `Views` κάθε ένα από τα οποία αποδίδουν μόνο προς μία κατεύθυνση. Η μέθοδος `creatFragment` χρησιμοποιείται για να δημιουργήσει ένα μέρος του `View` που καλύπτει ένα μεγάλο κομμάτι περιεχομένου χωρίς κατευθύνσεις.

Σε αυτό το σημείο, θα αναπτύξουμε τις μεθόδους τις οποίες πρέπει να υπερβούμε όταν χρησιμοποιούμε την κλάση υποδοχέα `CompositeView` η οποία, είναι μία αφηρημένη υλοποίηση που διαχειρίζεται μία ή και περισσότερες όψεις παιδιών. Να σημειωθεί ότι ο αριθμός των παιδιών που μπορεί να διαχειριστεί είναι σχετικά μικρός. Η κλάση `CompositeView` είναι υπερκλάση των όψεων που περιέχουν όψεις παδιών, όπως το `BoxView`.

Εκτός από τις αφηρημένες μεθόδους της κλάσης `View`, οι υποκλάσεις της `CompositeView` θα πρέπει να υπερβούν τις μεθόδους :

- `isBefore (int x, int y, Rectangle alloc)`

Εξετάζει αν ένα σημείο βρίσκεται πριν την ορθογώνια περιοχή της όψης του γονιού.

  - Παράμετροι :

x - η τετμημένη του σημείου (>=0)
y - η τεταγμένη του σημείου (>=0)
alloc - το ορθογώνιο

- Επιστρέφει :

αληθές, αν το σημείο είναι πριν την δεδομένη περιοχή.

- `isAfter (int x, int y, Rectangle alloc)`

Εξετάζει αν ένα σημείο βρίσκεται μετά την ορθογώνια περιοχή της όψης του γονιού.

- Παράμετροι :

x - η τετμημένη του σημείου (>=0)
y - η τεταγμένη του σημείου (>=0)
alloc - το ορθογώνιο

- Επιστρέφει :

αληθές, αν το σημείο είναι μετά την δεδομένη περιοχή.

- `getViewAtPoint (int x, int y, Rectangle rect)`

  Τοποθετεί τις όψεις των παιδιών στις δεδομένες συντεταγμένες.

- Παράμετροι :

x - η τετμημένη του σημείου (>=0)
y - η τεταγμένη του σημείου (>=0)
rect – το μερίδιο που αντιστοιχεί στον γονιό στην είσοδο, η οποία πρέπει να μετατρέπεται στο μερίδιο που αντιστοιχεί στο παιδί στην έξοδο

- Επιστρέφει :

την όψη του παιδιού

- `childAllocation(int index, Rectangle rect)`

  Επιστρέφει το μερίδιο που αντιστοιχεί σε ένα δεδομένο παιδί.

- Παράμετροι :

index - ο τακτικός αριθμός του παιδιού (>=0 && < getViewCount())
rect - το μερίδιο που αντιστοιχεί στο εσωτερικό του κουτιού στην είσοδο, το μερίδιο που αντιστοιχεί στην όψη του παιδιού με τον συγκεκριμένο τακτικό αριθμό στην έξοδο.

To ViewFactory

Όπως έχουμε δει, υπάρχουν αρκετοί διαφορετικοί τύποι για `Views` που αποδίδουν διαφορετικούς τύπους περιεχομένου. Στην πραγματικότητα, όμοιοι τύποι περιεχομένου μπορούν να διαχειριστούν από ένα διαφορετικό τύπο `View` μέσα σε διαφορετικά συστατικά κειμένου. Για παράδειγμα, το κείμενο σε ένα `JTextField` αποτυπώνεται από ένα `FieldView`, από ένα `PasswordView` σε ένα `JPasswordField`, από ένα `PlainView` ή `WrappedPlainView` σε ένα `JTextArea`, και από ένα `LabelView` σε ένα `JTextPane`. Η ανάθεση της κατάλληλης όψης σε ένα συγκεκριμένο τύπο περιεχομένου εκτελείται από ένα `ViewFactory`. Διαφορετικά συστατικά κειμένου χρησιμοποιούν διαφορετικές υλοποιήσεις του `ViewFactory` για να δημιουργήσουν το σωστό τύπο `View` για τις συγκεκριμένες τους περιπτώσεις. Το `ViewFactory` είναι στην πραγματικότητα μία διασύνδεση με μία μόνο μέθοδο, την `create`.

Η μέθοδος `create` εισάγει ένα νέο ανικείμενο `View` το οποίο μπορεί να αποτυπώσει μέρος, αν όχι όλο, του `Element` που δίνεται από το όρισμά της. Κάποιες υλοποιήσεις του `ViewFactory` (όπως αυτή του `JTextField`) έχουν μόνο ένα τύπο `View` (`FieldView`) και επομένως πάντα επιστρέφουν

ένα αντικείμενο αυτού του τύπου. Πιο πολύπλοκα συστατικά χρησιμοποιούν διάφορους τύπους από Views και συνήθως κοιτούν στο όνομα του Element (που επιστρέφεται από τη μέθοδο getName) για να αποφασίσουν ποια υποκλάση του View θα χρησιμοποιήσουν. Για παράδειγμα, σε ένα JTextPane το περιεχόμενο του Element αποτυπώνεται από ένα LabelView, ενώ ένα συστατικό Element αποτυπώνεται από ένα ComponentView. Η μέθοδος create είναι συνήθως μία if (αν) δήλωση που συγκρίνει το όνομα του στοιχείου με ένα διαμορφωμένο σύνολο από έγκυρα ονόματα για να αποφασίσει ποιο τύπο View να επιστρέψει. Ωστόσο, είναι εξίσου έγκυρο να χρησιμοποιήσουμε κάποιο από τα χαρακτηριστικά που συσχετίζονται με το Element (ή οποιοδήποτε άλλο κριτήριο) για όνομα με το να (ή από το να) χρησιμοποιήσουμε το όνομα που υποδεικνύει τον τύπο View. Αυτή την προσέγγιση συνήθως την παίρνουμε από το JEditorPane όταν εκθέτει HTML έγγραφα.

Υπάρχουν δύο τρόποι με τους οποίους ένα συστατικό κειμένου μπορεί να αντλήσει ένα ViewFactory από τον δικό του Editor Kit ή από το UI delegate. Όταν χτίζεται μία ιεραρχία από Views για το συστατικό, καλείται η μέθοδος modelChanged του UI delegate του συστατικού. Αυτή η μέθοδος παρέχεται από την κλάση javax.swing.plaf.basic.BasicTextUI, η οποία παίρνει το ViewFactory από τη μέθοδο getViewFactory μίας εσωτερικής κλάσης που ονομάζεται RootView και βρίσκεται στην κορυφή της ιεραρχίας των Views για *όλα* τα συστατικά κειμένου. Η μέθοδος getViewFactory της BasicTextUI.RootView πρώτα καλεί τη μέθοδο EditorKit του JTextComponent που υπάρχει για να συνδέει τα Views μεταξύ τους, και μετά καλεί τη μέθοδο getViewFactory της εργαλειοθήκης σύνταξης. Αν αυτή η μέθοδος επιστρέψει ένα factory, αυτό το factory χρησιμοποιείται για να δημιουργήσει όλα τα Views για το συστατικό. Αντιθέτως, αν επιστρέψει null, το UI delegate θα προσφέρει το ViewFactory με το να προμηθεύσει μία κατάλληλη μέθοδο create.

Αναφορικά με τα κλασικά συστατικά κειμένου, τα τρία πιο απλά, JTextField, JPasswordField και JTextArea, όλα χρησιμοποιούν την DefaultEditorKit, η οποίο δεν προμηθεύει ένα ViewFactory (η δικιά του μέθοδος getViewFactory επιστρέφει null) οπότε το ViewFactory για κάθε ένα από αυτά τα συστατικά εφαρμόζεται στη δικιά τους UI delegate κλάση. Αντιθέτως, το JTextPane χρησιμοποιεί την StyledEditorKit (ή μία υποκλάση του) η οποία παρέχει το ViewFactory που γνωρίζει πώς να αποτυπώσει στοιχεία από το DefaultStyledDocument. Το JEditorPane μπορεί να χρησιμοποιήσει οποιαδήποτε εργαλειοθήκη σύνταξης, αλλά αυτή η εργαλειοθήκη σύνταξης πρέπει να προμηθεύει ένα factory όψεων, επειδή το BasicEditorPaneUI δεν υπερβαίνει τη μέθοδο BasicTextUI create, η οποία επιστρέφει null όταν ζητείται για ένα View, οπουδήποτε είδους στοιχείου.

Επειδή ένα ViewFactory δημιουργεί ένα View βασισμένο στον τύπο του κάθε Element που βρίσκεται μέσα στο έγγραφο, είναι φανερό ότι πρέπει να υπάρχει μία στενή σύνδεση ανάμεσα στο ViewFactory και στην κλάση Document που προβάλλεται μέσα στο συστατικό κειμένου. Τα εργοστάσια μέσα στο UI delegate για τα απλά συστατικά κειμένου μπορούν, για παράδειγμα, να επιστρέψουν μόνο αντικείμενα View για τους τύπους Element που δημιουργήθηκαν από το PlainDocument, που είναι το μοντέλο Document που χρησιμοποιείται από αυτά τα συστατικά. Αν δημιουργήσουμε μία υποκλάση του PlainDocument που θέλουμε να χρησιμοποιήσουμε με οποιοδήποτε από αυτά τα συστατικά, θα πρέπει είτε να περιοριστούμε στο να χρησιμοποιήσουμε τους ίδιους τύπους στοιχείων με το PlainDocument είτε να επεκτείνουμε το ViewFactory στην UI delegate κλάση έτσι ώστε να μπορεί να παρέχει Views για τα δικούς μας τύπους στοιχείων.

Το Μέγεθος της Παραγράφου, Πλάνο για τις Γραμμές

Για να κατανοήσουμε ακριβώς πώς και γιατί το `LabelView.LabelFragment` δημιουργήθηκε, ας κοιτάξουμε πιο λεπτομερειακά στο πώς τα `ParagraphView` και `ParagraphView.Row` δημιουργούν και χειρίζονται τα `Views` των δικών τους παιδιών. Όταν πρωτοδημιουργείται ένα `ParagraphView`, περιέχει ένα σχετικό με αυτό, στοιχείο `paragraph`. Σε αυτό το σημείο, δε γνωρίζει πόσος χώρος της οθόνης προορίζεται για αυτό και στην πραγματικότητα, ίσως να ρωτηθεί για το επιθυμητό του μέγεθος πριν του δοθεί ένας προορισμός για να δουλέψει. Επειδή ο υπολογισμός του επιθυμητού του μεγέθους συμπεριλαμβάνει το να μπορεί να μετρήσει το πλάτος του κειμένου (και άλλα αντικείμενα) που περιέχει, χρειάζεται να δημιουργήσει `Views` παιδιών (όπως το `LabelView`) που μπορούν να κάνουν αυτή τη δουλειά με το να εργάζονται απευθείας με το περιεχόμενο των στοιχείων μέσα στο μοντέλο. Το ίδιο το `ParagraphView`, με το να είναι ένα απλό αντικείμενο υποδοχέα, δε γνωρίζει πώς να διεξάγει τη μέτρηση του κειμένου, οπότε πρέπει να αναθέσει αυτό το εγχείρημα στα παιδιά του.

Όπως ήδη γνωρίζουμε από την μέχρι τώρα περιγραφή του `JTextArea`, λίγο μετά αφότου δημιουργηθεί ένα `View`, καλείται η μέθοδός του `setParent`. Για ένα υποδοχέα `View` που προέρχεται από το `CompositeView`, καλείται η μεθόδος που ονομάζεται `loadChildren` η οποία αποτελεί το φυσικό μέρος για να δημιουργηθεί ένα `View` παιδιού, αν απαιτείται κάποιο. Εκεί στην πραγματικότητα, είναι το μέρος όπου το `ParagraphView` δημιουργεί ένα καινούριο σύνολο από `Views` που θα χειριστεί στο τέλος, και ίσως να περιμέναμε από όσα έχουμε δει ως τώρα, αυτά να είναι τα `Views` του `ParagraphView.Row`. Ωστόσο, δε συμβαίνει αυτό. Αντιθέτως, η μέθοδος `loadChildren` σχηματίζει μία θηλιά ανάμεσα σε όλα τα στοιχεία των παιδιών του στοιχείου της παραγράφου που αποτυπώνεται από το `ParagraphView`, και δημιουργεί το κατάλληλο `View` για κάθε στοιχείο, χρησιμοποιώντας το συστατικό κειμένου που συσχετίζεται με το `ViewFactory`.

Έχοντας δημιουργήσει ένα σύνολο από αντικείμενα `LabelView`, το `ParagraphView` τα τοποθετεί όλα μαζί σε ένα `Vector` το οποίο αναφέρεται ως το *σύνολο διάταξής* του (*layout pool*). Αυτό το σύνολο από `Views` διατάσσεται με τον ίδιο τρόπο με τα στοιχεία περιεχομένου που αποτυπώνει, αλλά αυτά τα `Views` δεν εμφανίζονται σαν τα άμεσα παιδιά του `ParagraphView`, επειδή οι μέθοδοι `getViewCount` και `getView` δεν τους επιτρέπουν να φαίνονται έξω από το `ParagraphView`. Λίγο μέτα από αυτό, το συστατικό κειμένου μπορεί να ερωτηθεί για κάποιο μέγεθός του (ή για όλα), για το μικρότερο, το μεγαλύτερο, ή το επιθυμητό του μέγεθος, κάτι που εξαρτάται από τον διαχειριστή πλάνου που χρησιμοποιείται από τον υποδοχέα στον οποίο το συστατικό είναι τοποθετημένο. Τα κατάλληλα μεγέθη για ένα συστατικό κειμένου εξαρτώνται φυσικά, από το πώς τα `Views` του, αποδίδουν το περιεχόμενο του συστατικού, οπότε αυτό το αίτημα μεταβιβάζεται στην ιεραρχία των `Views`. Για παράδειγμα το `BoxView`, υπολογίζει τις απαιτήσεις του με να αναθέτει αυτό το ζήτημα σε κάθε ένα από τα `ParagraphViews` που περιέχει, και χρησιμοποιεί τα αποτελέσματα για να υπολογίσει τη δικιά του απαίτηση. Για παράδειγμα, το κάθετο διάστημα για ένα `BoxView` που χειρίζεται τα παιδιά του πάνω στον `y` άξονα, μπορεί να αποκτηθεί με το να αθροίσουμε τα κάθετα διαστήματα όλων των παιδιών.

Το επιθυμητό οριζόντιο διάστημα για αυτό το `BoxView` θα αποκτηθεί με το να πάρουμε το μεγαλύτερο από όλα τα επιθυμητά μεγέθη των παιδιών του, και υπάρχει ένα αντίστοιχος αλγόριθμος για κάθε κατεύθυνση για τον υπολογισμό των μικρότερων και μεγαλύτερων μεγεθών.

Οπότε, πώς ένα `ParagraphView` υπολογίζει τα δικά του μεγέθη, το μεγαλύτερο, το μικρότερο, το επιθυμητό; Αυτό είναι ένα σχετικά περίπλοκο πρόβλημα καθώς υπάρχουν περισσότεροι από έναν τρόποι για να διατάξουμε ένα κείμενο σε μία παράγραφο όταν τελειώνει η γραμμή. Αν μας είχε δοθεί ένα κείμενο για να το στοιχειοθετήσουμε, θα ξεκινούσαμε από την πάνω αριστερά μεριά του

χώρου που προοριζόταν για εμάς, και θα εργαζόμασταν από την πάνω γραμμή τοποθετώντας τις λέξεις μέχρι να μην είχαμε άλλο χώρο και μετά θα πηγαίναμε στην επόμενη γραμμή και θα συνεχίζαμε με αυτό τον τρόπο μέχρις ώτου να μην υπήρχαν άλλες λέξεις για να γράψουμε. Αυτό βεβαίως, προϋποθέτει ότι γνωρίζουμε το πλάτος της παραγράγου και από αυτό θα καταλήξουμε στην τιμή του ύψους της. Ωστόσο, αν δε γνωρίζουμε πόσο πλατύς είναι ο προορισμένος χώρος μας, υπάρχουν πολλοί τρόποι για να διατάξουμε τις λέξεις που περιέχει, και ποικίλουν από το να τα γράψουμε όλα σε μία γραμμή μέχρι, το να γράψουμε μία μόνο λέξη σε κάθε γραμμή. Αυτό που συμβαίνει ακριβώς, είναι ότι, όταν το `ParagraphView` ρωτάται για το επιθυμητό του διάστημα πάνω στον `x` άξονα, πηγαίνει σε κάθε του παιδί στο σύνολο διάταξής του, και το ρωτάει πόσο οριζόντιο χώρο χρειάζεται (με τη μέθοδο `getPrefferedSpan`) και μετά αθροίζει τα αποτελέσματα. Για το ίδιο, αθροίζει όλα τα ενθέματα της παραγράφου που έχουν τοποθετηθεί και επιστρέφει το αποτέλεσμα. Αυτό αντιστοιχεί στο να διατάξει ολόκληρη την παράγραφο σε μία γραμμή στην οθόνη. Κάτι το οποίο συχνά σημαίνει ότι, το επιθυμητό ύψος της παραγράφου έχει δοθεί ως 0. Γι αυτό το λόγο, το επιθυμητό μέγεθος που επιστρέφεται από το `JTextPane`, δεν έχει πρακτική εφαρμογή.

Όταν επιστρέφεται το επιθυμητό μέγεθος, το συστατικό κειμένου θα προορίσει κάποιο χώρο για το ίδιο στον υποδοχέα του. Στο τέλος, η μέθοδος `setSize` του `RootView` θα κληθεί με τις ακριβείς διαστάσεις του συστατικού κειμένου. Αυτό το έργο γίνεται , βήμα - βήμα, από μία ποικιλία αντικειμένων `View` της ιεραρχίας. Πρώτα το `RootView` καλεί τη μέθοδο `setSize` από το κορυφαίο `View` της ιεραρχίας κάτω από αυτό, το οποίο θα είναι το `BoxView` σε αυτή την περίπτωση. Το `BoxView` χρησιμοποιεί το μέγεθος που προορίζονται για αυτό για να δημιουργήσει την κατανομή μεγέθους για τα `Views` των παιδιών του, τα οποία όλα θα είναι `ParagraphViews`, και κατόπιν καλεί τη μέθοδο `setSize` για κάθε ένα από αυτά με το πραγματικό τους μέγεθος.

Πλέον, κάθε `ParagraphView` γνωρίζει πόσο χώρο στην οθόνη έχει για να εργαστεί και πόσο χώρο πρέπει να προορίσει για κάθε γραμμή του κειμένου που περιέχει. Αυτό το κάνει με το να αναθέτει αυτό το ζήτημα στην κλάση `ParagraphView.Row` για κάθε γραμμή του κειμένου. Σε αυτό το στάδιο, το αντικείμενο `ParagraphView` γνωρίζει το πλήθος των offsets που προορίζονται για το κείμενο (ή άλλο περιεχόμενο) που καλύπτει, αλλά δε γνωρίζει πόσες γραμμές στην οθόνη αυτό θα καλύψει. Στην πραγματικότητα αυτό δε θα μαθευτεί μέχρις ότου όλες οι γραμμές ξεχωριστά τοποθετηθούν. Εκείνο που πραγματικά συμβαίνει είναι ότι το `ParagraphView` δημιουργεί ένα μοναδικό `ParagraphView.Row` και προσπαθεί να χωρέσει μέσα σε αυτό όσο περισσότερο από το περιεχόμενο μπορεί. Όταν γεμίζει το `ParagraphView.Row`, δημιουργείται ένα καινούριο και αυτό συμβαίνει συνεχώς μέχρι να αποτυπωθεί όλο το περιεχόμενο.

Τώρα ας δούμε πώς γεμίζει ένα `ParagraphView.Row`. Στην αρχή, το `ParagraphView.Row` γνωρίζει πόσο πλατιά είναι η περιοχή στην οθόνη μέσα στην οποία θα εργαστεί, και τα αρχικά και τελικά offsets του περιεχομένου που πρέπει να αποτυπώσει. Γενικά, οποιοδήποτε `ParagraphView.Row` θα αποτυπώσει την αρχή αυτού του συνόλου των offsets και θα επιστρέψει το υπόλοιπο για το επόμενο `ParagraphView.Row`. Αρχικά, το `ParagraphView.Row` αφήνει χώρο για το αριστερό ένθεμα της παραγράφου και κατόπιν παίρνει το `View` για το στοιχείο που αποτυπώνει από το σύνολο διάταξής του με το να χρησιμοποιεί το offset του στοιχείου σαν κλειδί. Για να καθορίσει το χώρο που χρειάζεται αυτό το `View`, μία από τις δύο, επόμενες, μεθόδους καλείται. Αν το `View` υποστηρίζει εμφυτευμένους στηλοθέτες, καλείται η μέθοδός του `getTabbedSpan`, αλλιώς χρησιμοποιείται η μέθοδος `getPrefferdSpan`. Αυτές οι μέθοδοι, και οι δύο, έχουν πρόσβαση στο περιεχόμενο που αποδίδουν, επειδή ένα `View` δημιουργείται με ένα σχετικό στοιχείο. Αν το πλάτος που απαιτείται να αποδοθεί από το `View` είναι μικρότερο από το υπόλοιπο της γραμμής, το `View` προστίθεται στο `ParagraphView.Row` και το πλάτος του αφαιρείται από τον διαθέσιμο χώρο. Τότε,

καλείται η μέθοδος `getBreakWeight` της όψης και, αν αυτή επιστρέψει `ForcedBreakView` (ή μία ακόμα μεγαλύτερη τιμή), το `ParagraphView.Row` θεωρείται γεμάτο και θα ξεκινήσει ένα καινούριο `Row` για το `View` που σχετίζεται με το επόμενο στοιχείο.

Ένα Καινούριο ViewFactory

Έχοντας καθορίσει κάποια από τα νέα χαρακτηριστικά, το νέο πρόβλημα είναι κάπως να τα διευθετήσουμε για να επηράσουμε τον τρόπο με τον οποίο η παράγραφος στην οποία απαντώται έχει αποδοθεί. Για να αλλάξουμε τον τρόπο με τον οποίο έχουν σχεδιαστεί οι παράγραφοι, χρειάζεται να χρησιμοποιήσουμε ένα συνηθισμένο `View` που καταλαβαίνει τα νέα χαρακτηριστικά και τα ρυθμίζει με τέτοιο τρόπο ώστε ένα ομότυπό του να δημιουργείται από το έγγραφο για κάθε παράγραφο. Η αποτύπωση ανάμεσα στους τύπους των στοιχείων στα μοντέλα και στις όψεις καθορίζεται από το `ViewFactory` των συστατικών κειμένου. Στην περίπτωση του `JTextPane`, η εργαλειοθήκη σύνταξης προμηθεύει το `ViewFactory`, έτσι ώστε να διευθετήσει ότι ένα νέο `View` θα χρησιμοποιείται για να αποτυπώνει στοιχεία παραγράφου· εμείς πρέπει να δημιουργήσουμε μία νέα εργαλειοθήκη σύνταξης με ένα κατάλληλο `ViewFactory`.

Το `JTextPane` προϋποθέτει ότι η δικιά του εργαλειοθήκη σύνταξης προέρχεται από την `StyledEditorKit`. Εξαιτίας αυτού, η δικιά μας εργαλειοθήκη σύνταξης θα δημιουργηθεί με το να την επεκτείνουμε στην `StyledEditorKit`. Επειδή η μόνη βελτίωση που χρειάζεται να γίνει είναι για το `ViewFactory`, χρειάζεται απλά να υπερβούμε τη μέθοδο `getViewFactory`, που καλείται από το `UI delegate` του `JTextPane`.

Το νέο `ViewFactory` δηλώνεται ως στατική εσωτερική κλάση της δικιάς μας εργαλειοθήκης σύνταξης και ένα μοναδικό ομότυπο δημιουργείται όταν εισάγεται η κλάση πρώτη φορά. Η μέθοδος `getViewFactory` απλά επιστρέφει μία αναφορά για αυτό το μοιρασμένο ομότυπο. Το `ViewFactory` έχει μόνο μία μέθοδο, η οποία δημιουργεί τα `Views` για κάθε τύπο στοιχείου στο σχετικό έγγραφο. Στη μέθοδο `create`, κοιτάμε το όνομα του στοιχείου για να καθορίσουμε ποιο τύπο `View` να επιστρέψουμε. Αν το στοιχείο αναπαριστά μία παράγραφο, το κατάλληλο πράγμα για να επιστρέψουμε είναι μία περίπτωση από τη δικιά μας καινούρια κλάση `View`, ένα `ExtendedParagraphView`. Για όλους τους άλλους τύπους στοιχείων, θα προτιμήσουμε να αναθέσουμε το `ViewFactory` στην `StyledEditorKit` από το να αντιγράψουμε όλο τον κώδικά του μέσα στη δικιά μας μέθοδο `create`. Για να το κάνουμε αυτό παίρνουμε μία αναφορά για το `ViewFactory` από τον δικό μας στατικό ρυθμιστή. Ωστόσο, επειδή η μέθοδος `getViewFactory` δεν είναι στατική, θα χρειαστεί να δημιουργήσουμε ένα αντικείμενο `StyledEditorKit` και να το επικαλεστούμε.

Αξίζει να εξετάσουμε αν είναι πιθανό να γίνει μία εφαρμογή ενός εναλλακτικού `View`. Όπως έχουν τα πράγματα ως τώρα, κάθε παράγραφος του εγγράφου έχει ένα `ExtendedParagraphView` που συσχετίζεται με αυτήν, ακόμα κι αν η παράγραφος χρησιμοποιεί κάποιο από τα δικά μας νέα χαρακτηριστικά. Εναλλακτικά, θα μπορούσαμε ίσως να έχουμε τη μέθοδο `create` για να ψάχνει για αυτά τα χαρακτηριστικά και να επιστρέψει ένα `ExtendedParagraphView` για την παράγραφο στην οποία υπάρχουν, και ένα συνηθισμένο `ParagraphView`, αλλιώς. Το πρόβλημα με αυτό έγγυται στο τί θα έπρεπε να γίνει αν το σύνορο της παραγράφου ή τα χαρακτηριστικά του φόντου της παραγράφου προστέθηκαν μετά από τη δημιουργία του δομήματος του `View`. Όταν συμβαίνει αυτό, είναι απαραίτητο να αντικαταστήσουμε το `ParagraphView` με ένα `ExtendedParagraphView`. Ωστόσο, αν η μόνη αλλαγή που έγινε στο έγγραφο είναι η εφαρμογή αυτών των χαρακτηριστικών (το περιεχόμενο του εγγράφου δεν αλλάζει την ίδια στιγμή) η δομή του `View` δεν ξαναχτίζεται αυτομάτως, και αυτό έχει σαν αποτέλεσμα η επηρεασμένη παράγραφος να συνεχίζει να αποδίδεται από το `ParagraphView`. Περισσότερος κώδικας θα χρειαζόταν να προστεθεί για να αντιδράσει σε

δεδομένες αλλαγές χαρακτηριστικών, με το να εγκατασταθεί ένα `ExtendedParagraphView`, κάτι το οποίο θα περιέπλεκε την εφαρμογή.

ΚΕΦΑΛΑΙΟ 4.

ΣΧΕΔΙΑΣΗ ΚΑΙ ΥΛΟΠΟΙΗΣΗ ΜΙΑΣ ΕΦΑΡΜΟΓΗΣ ΓΙΑ ΤΗΝ ΕΠΕΞΕΡΓΑΣΙΑ ΤΩΝ ΜΑΘΗΜΑΤΙΚΩΝ ΕΚΦΡΑΣΕΩΝ.

Η Αρχιτεκτονική της εφαρμογής

Για την εφαρμογή μας στις μαθηματικές εκφράσεις στη Java, εργαστήκαμε με το πρότυπο σχεδίασης MVC του οποίου η βασική ιδέα είναι να διαχωρίσουμε την εφαρμογή μας σε τρία ευδιάκριτα μέρη, κάθε ένα από τα οποία μπορεί να αντικατασταθεί χωρίς να επηρεάζει τα άλλα · το μοντέλο, που στην εφαρμογή μας είναι η κλάση `ExpressionDocument`, την όψη, εμείς έχουμε κατασκευάσει όψεις για την απεικόνιση διάφορων μαθηματικών εκφράσεων και τον ελεγκτή τον ρόλο του οποίου παίζει η κλάση `ExpressionEditorKit`. Ξεκινώντας την εφαρμογή μας, καθότι εργαστήκαμε με το Swing έπρεπε να δημιουργήσουμε μία κλάση που θα επεκτείνει τον κορυφαίο υποδοχέα `JFrame`, κάτι που κάναμε με την κλάση `Application` η οποία βρίσκεται στο πακέτο `gr.aegean.math` (όπως και οι κλάσεις `ExpressionDocument`, `ExpressionEditorKit`, `ExpressionFactory`). Κατόπιν δημιουργούμε ένα καινούριο `JEditorPane`, που είναι το συστατικό κειμένου της εφαρμογής μέσα στο οποίο θα προστεθούν κι άλλα συστατικά και θα εμφανιστούν οι μαθηματικές εκφράσεις (όπως έχουμε πει το `JEditorPane` είναι ένα παράθυρο πάνω σε ένα συντάκτη εγγράφου) .

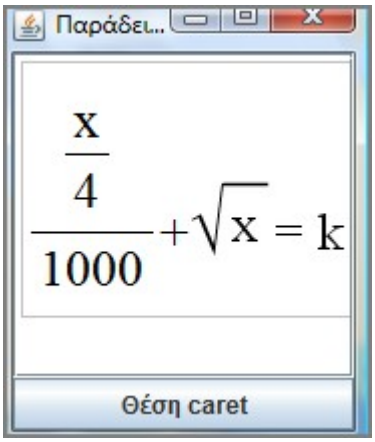
Συνεχίζοντας, κατασκευάσαμε την κλάση `ExpressionDocument` η οποία επεκτείνει την κλάση για έγγραφα `DefaultStyledDocument` και παραθέτει τα δεδομένα που θα εμφανιστούν μέσα στο παράθυρο. Ωστόσο, καθότι τα δεδομένα τα περιέχει το πρόγραμμα και δε μπορούν να αλλάξουν με επέμβαση του χρήστη, το μοντέλο δεν αλλάζει επομένως ούτε και οι όψεις, εκτός κι αν τα μετατρέψει ο προγραμματιστής. Μέσα στην κλάση δηλώνονται τα ονόματα των εκφράσεων και στον κατασκευαστή της θα γράψουμε τις εκφράσεις που θα εμφανιστούν στην οθόνη. Τέλος στην κλάση αυτή βρίσκεται η μέθοδος `init` η οποία γεμίζει τις παραγράφους με τα στοιχεία του εγγράφου.

```
public  class ExpressionDocument extends DefaultStyledDocument {
      public static final String ELEMENT_NAME_TEXT = "text";
      public static final String ELEMENT_NAME_FRACTION = "fraction";
      public static final String ELEMENT_NAME_ROOT = "root";
      public static final String ELEMENT_NAME_ROW= "row";
      // αντίστοιχα και για τις υπόλοιπες εκφράσεις.

      public ExpressionDocument() {

      Fraction e = new Fraction(new Fraction(new Text("x"), new
      Text("4")), new Text("1000"));
      // θα φτιάξει ένα καινούριο κλάσμα του οποίου ο αριθμητής θα είναι
         το κλάσμα x διά 4 και ο παρονομαστής θα είναι το 1000.
      Root a = new Root(new Text ("x"));
      // θα φτιάξει μία ρίζα μέσα στην οποία θα υπάρχει το x.
      Row row = new Row();
      // θα στοιχίσει τις εκφράσεις σε μία γραμμή.

       row.add(e);
      // τοποθετεί πρώτα το κλάσμα e.
      row.add(new Text("+"));
      // τοποθετεί τον τελεστή +.
      row.add(a);
      // τοποθετεί τη ρίζα a.
```

```
 row.add(new Text("= "));
      // τοποθετεί τον τελεστή =.
      row.add(new Text ("k"));
      // τοποθετεί το γράμμα k.

      init(row);
      }
}
```

Παρέχοντας αυτές τις εντολές στο πρόγραμμα αυτό που θα εμφανιστεί στην οθόνη θα είναι το :

ΕΙΚΟΝΑ 21.



Στη συνέχεια, κατασκευάσαμε τη δική μας κλάση εργαλειοθήκης σύνταξης που είναι η `ExpressionEditorKit` και την οποία επεκτείνουμε στην εργαλειοθήκη σύνταξης `StyledEditorKit`. Μέσα στην κλάση αυτή, καθότι παίζει και το ρόλο του ελεγκτή, όπως είναι λογικό, υπάρχουν, μία μέθοδος που παραθέτει το `factory` που είναι κατάλληλο για να παράγει τις όψεις του μοντέλου το οποίο παράγεται από αυτή την εργαλειοθήκη σύνταξης, η `getViewFactory()`, και μία που δημιουργεί ένα μοντέλο για αποθήκευση κειμένου που είναι κατάλληλο για αυτόν τον τύπο συντάκτη, η `createDefaultDocument()`.

Τέλος, σε αυτό το πακέτο υπάρχει η κλάση `ExpressionFactory` η οποία εφαρμόζει την κλάση `ViewFactory` και μέσα στην οποία μέσω της εντολής if εξετάζεται το όνομα του στοιχείου και επιστρέφεται η αντίστοιχη όψη.

Μετά το τέλος του πρώτου πακέτου δημιουργείται το πακέτο `gr.aegean.math.exp` μέσα στο οποίο υπάρχουν οι κλάσεις των εκφράσεων. Πρώτη, μέσα στο πακέτο υπάρχει η αφηρημένη κλάση `Expression` μέσα στην οποία περιέχεται η αφηρημένη μέθοδος `toDocument` η οποία θα υλοποιηθεί σε κάθε μία από τις υπόλοιπες εκφράσεις. Η κλάση `Expression` βρίσκεται στην κορυφή της ιεραρχίας του πακέτου και ορίζει τη συμπεριφορά και τις ιδιότητες που είναι κοινές σε όλες τις κλάσεις του πακέτου.

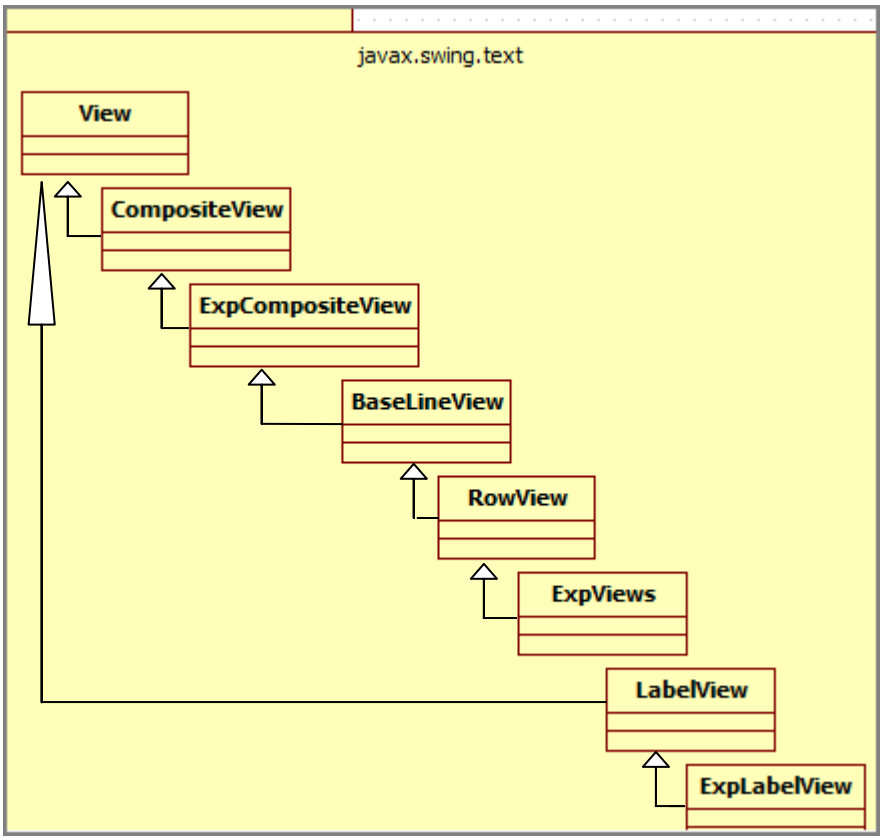
Αμέσως μετά υπάρχουν όλες οι κλάσεις των εκφράσεων που χρησιμοποιούνται για την υλοποιήση του προγράμματος. Ας πάρουμε για παράδειγμα την κλάση `Fraction`.

```
public class Fraction extends Expression {
      Expression nom, denom;
      // για την έκφραση Fraction χρειαζόμαστε δύο υποεκφράσεις τον αριθμητή
      (nom) και τον παρονομαστή (denom).

      public Fraction(Expression nom, Expression denom) {
      this.nom = nom;

      this.denom = denom;
      }

      public void toDocument(ArrayList specs) {

      SimpleAttributeSet attributes = new SimpleAttributeSet();
      // δημιουργούμε μία νέα λίστα που είναι η attributes.
      attributes.addAttribute(AbstractDocument.ElementNameAttribute,
                  ExpressionDocument.ELEMENT_NAME_FRACTION);
      // προσθέτουμε στη λίστα το όνομα του στοιχείου που, στην προκειμένη
         περίπτωση, είναι το Fraction και θα χρησιμοποιηθεί από την κλάση
         ExpressionFactory (όταν στην κλάση ExpressionDocument δώσουμε
         αυτό το στοιχείο) για να παράγει την αντίστοιχη όψη που θα είναι
         το FractionView.

      ElementSpec start = new ElementSpec(attributes,
            ElementSpec.StartTagType);
      // ξεικινάμε να γεμίσουμε τη λίστα με τα στοιχεία.
      specs.add(start);

      nom.toDocument(specs);
      denom.toDocument(specs);

      // οι υποεκφράσεις του κλάσματος nom, denom (αριθμητής,
         παρονομαστής, αντίστοιχα) προστίθενται στη λίστα.
      ElementSpec end = new ElementSpec(attributes,
                  ElementSpec.EndTagType);
      // κλείνει η λίστα που γεμίζει τα στοιχεία Fraction.
      specs.add(end);
      }
}
```

Το τελευταίο πακέτο μας είναι το `gr.aegean.math.views` μέσα στο οποίο περιέχονται οι κλάσεις που δημιουργήσαμε για τις όψεις. Αρχικά ξεκινάμε δημιουργώντας την αφηρημένη κλάση `BaseLineView` η οποία επεκτείνει την, επίσης, αφηρημένη κλάση `ExpCompositeView` η οποία με τη σειρά της επεκτείνει την κλάση `CompositeView`.

Η κλάση `ExpCompositeView` χρησιμοποιείται από την κλάση `ExpLabelView` η οποία επεκτείνει την κλάση `LabelView` και είναι υπεύθυνη για να δώσει την κατάλληλη γραμματοσειρά στις εκφράσεις μέσω της μεθόδου της `getFont()`. Δηλαδή, η δικιά μας ιεραρχία είναι η :

ΕΙΚΟΝΑ 22.



*Με ExpViews εννοούμε τις όψεις των εκφράσεών μας.

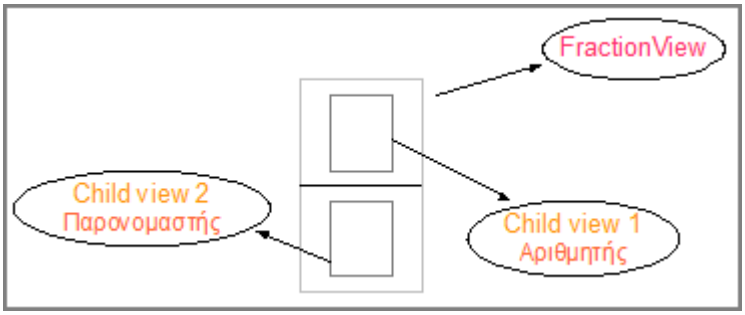
Για τις υπόλοιπες όψεις θα πάρουμε σαν παράδειγμα την όψη για την έκφραση του κλάσματος η οποία είναι η `FractionView` και η οποία επεκτείνει την κλάση `BaseLineView`. Αρχικά εφαρμόζουμε τη μέθοδο `forwardUpdate` η οποία ξαναζωγραφίζει όλο το `EditorPane`, και ύστερα υπερβαίνουμε τις μεθόδους `getAlignment`, `loadChildren`, `calculateSpans`, `getPrefferdSpan`, `paint` και τέλος την αφηρημένη μέθοδο της κλάσης `BaseLineView` `getBaseLine`. Όμως, όπως φαίνεται και στην Εικόνα 22, η κλάση `FractionView` είναι υποκλάση της `CompositeView`, επομένως, χρειάστηκε να υπερβούμε και τις μεθόδους `childAllocation`, `getViewAtPoint`, `isAfter` και `isBefore`.

```
public class FractionView extends BaseLineView {

	public FractionView(Element elem) {
	super(elem);
	}

	protected void forwardUpdate(DocumentEvent.ElementChange ec,
		DocumentEvent e, Shape a, ViewFactory f) {
	super.forwardUpdate(ec, e, a, f);

	calculateSpans();
	Component c = getContainer();
	c.repaint();
	}


	public float getAlignment(int axis)
	{
	if (axis== View.Y_AXIS)
		return .5f;
	return .5f;
	}
	// στοιχίζει στη μέση παίρνοντας σαν παράμετρο τον άξονα.

	protected void loadChildren(ViewFactory f) {
	super.loadChildren(f);
	calculateSpans();
	}
	// εισάγει τα παιδιά.

	float w1, h1, w2, h2;

	protected void calculateSpans()
	{
	w1 = getView(0).getPreferredSpan(X_AXIS);
	h1 = getView(0).getPreferredSpan(Y_AXIS);
	w2 = getView(1).getPreferredSpan(X_AXIS);
	h2 = getView(1).getPreferredSpan(Y_AXIS);
	}
	// μετράει τις τιμές του ύψους και του πλάτους των παιδιών.

	protected void childAllocation(int index, Rectangle rect) {
	if (rect != null) {
		if (index == 0) {
			if (w2 > w1)
				rect.x += .5 * (w2 - w1) + 12.5;
			else
				rect.x += 12.5;
			rect.y += 10;
			rect.width = (int) w1;
			rect.height = (int) h1;
```

```
        } else if (index == 1) {
              if (w2 > w1)
                    rect.x += 12.5;
              else
                    rect.x += .5 * (w1 - w2) + 12.5;
              rect.y += 20 + h1;
              rect.width = (int) w2;
              rect.height = (int) h2;
  } else
        rect = null;
  }
  }
  // ανάλογα με τον τακτικό αριθμό του παιδιού διαμορφώνει τις τιμές του
     ύψους του και του πλάτους του.

  protected View getViewAtPoint(int x, int y, Rectangle rect) {
  if (y < rect.y + h1 + 15) {
        childAllocation(0, rect);
        return getView(0);
  } else {
        childAllocation(1, rect);

        return getView(1);
  }
  }
  // επιστρέφει το παιδί που αντιστοιχεί, ανάλογα με τις αποστάσεις κατά
     τον κατακόρυφο άξονα.

  protected boolean isAfter(int x, int y, Rectangle alloc) {
  return y > alloc.y + alloc.height;
  }
  // εξετάζει αν η τιμή που δόθηκε βρίσκεται μετά το ύψος της όψης
     FractionView.

  protected boolean isBefore(int x, int y, Rectangle alloc) {
  return y < alloc.y;
  }
  // εξετάζει αν η τιμή που δόθηκε βρίσκεται πριν το ύψος της όψης
     FractionView.

  public float getPreferredSpan(int axis) {
  if (axis == X_AXIS)
        return 30 + Math.max(w1, w2);
  return 30 + h1 + h2;
  }
  // επιστρέφει το επιθυμητό μέγεθος της όψης ανάλογα με τον άξονα που
     ζητήθηκε.
```

```
public void paint(Graphics g, Shape shape) {
      g.setColor(Color.lightGray);
      Rectangle rect = shape.getBounds();
      int x = rect.x;
      int y = rect.y;

      Rectangle r1 = getChildAllocation(0, shape).getBounds();

      Rectangle r2 = getChildAllocation(1, shape).getBounds();

      getView(0).paint(g, r1);
      getView(1).paint(g, r2);
      // σχεδιάζει στην οθόνη τα παιδιά σε μία ορθογώνια περιοχή, το κάθε
         ένα, με ύψος και πλάτος τις τιμές που έχουν αποδοθεί για αυτό.


      g.drawLine(x + 5, 15 + (int) h1 + y, 20 + (int)Math.max(w1, w2) + x,
                  15 + (int) h1 + y);
      // σχεδιάζει τη διαχωριστική γραμμή ανάμεσα στον αριθμητή και τον
         παρονομαστή.
      }

      public float getBaseline() {
      return h1 - 2 * inset - 1;
      }
      // εφαρμόζεται η αφηρημένη μέθοδος getBaseLine για να επιστρέψει το
         μέγιστο πάνω ύψος της όψης, το οποίο θα ζητηθεί από την κλάση RowView.
}
```

ΕΙΚΟΝΑ 23.



Στην Εικόνα 23, φαίνεται ένα διάγραμμα για την κλάση FractionView. Η συγκεκριμένη κλάση έχει δύο παιδιά, τον αριθμητή και τον παρονομαστή, τα οποία μέσω της μεθόδου `paint` θα σχεδιαστούν σαν ορθογώνια πλαίσια τα οποία θα τοποθετηθούν στις συντεταγμένες που δίνονται από τη μέθοδο `childAllocation` και από την `getPrefferedSpan` θα δοθούν το επιθυμητό ύψος και πλάτος του πατέρα ο οποίος θα στεγάσει τα παιδιά του. Επίσης, όπως προαναφέρθηκε, στη μέθοδο `paint` σχεδιάζεται και μία γραμμή ανάμεσα στα παιδιά, η οποία είναι η γραμμή του κλάσματος.

Αντίστοιχα, τα ίδια γίνονται για όλες τις όψεις με τη μόνη διαφορά στην όψη `RootView` της ρίζας, στην οποία χρησιμοποιήσαμε έτοιμη απεικόνιση της ρίζας χρησιμοποιώτας τις εντολές :

```
Font f = new Font("Times New Roman", Font.PLAIN, 32);
Font previousFont = g.getFont();
g.setFont(f);

Graphics2D g2 = (Graphics2D) g;
// το αντικείμενο g μετατρέπεται, μέσω αλλαγής τύπου, από την κλάση
   Graphics στην κλάση Graphics2D, για γίνει χρήση, κάποιων, από τις
   μεθόδους που περιέχει.

g2.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
            RenderingHints.VALUE_ANTIALIAS_ON);

// εξομαλύνει τις γραμμές.

float scale = 1.095f * height/32.f;

Rectangle r1 = getChildAllocation(0, shape).getBounds();

g2.setColor(Color.black);

AffineTransform at = new AffineTransform();
// δημιουργείται ένα καινούριο αντικείμενο που είναι αντίγραφο του
   αντικειμένου που θέλουμε να διατηρεί τη γραμμή ίσια όταν η γραμμή
   μεγαλώνει.

at.scale(1, scale);
AffineTransform oldTrans = g2.getTransform();
g2.transform(at);

g2.drawString("\u221A", rect.x + 8, (y  + (int) height)/scale);
// σχεδιάζει τη ρίζα.

g2.setTransform(oldTrans);
// εφαρμόζει αυτή την αλλαγή στο αντικείμενο oldTrans.

Stroke oldStroke = g2.getStroke();
g2.setStroke(new BasicStroke(1.5f));
// θέτει το πάχος της γραμμής που θα προστεθεί δίπλα στη
   γραμματοσειρά της ρίζας.
g2.drawLine(x + (int) w0 + 8, y , x + (int) w0 + r1.width + 12, y);
// σχεδιάζεται η γραμμή.
g2.setStroke(oldStroke);
getView(0).paint(g, r1);
}
```

Για το τέλος, θα παραθέσουμε τον κώδικα της όψης `RowView`, η οποία επεκτείνει την κλάση `BaseLineView`, και διαφέρει από τις υπόλοιπες, κυρίως, στις μεθόδους `calculateSpans`, `childallocation`.

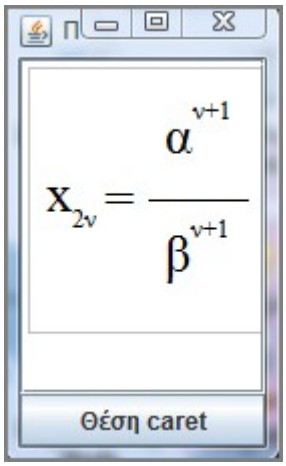
```
public class RowView extends BaseLineView {

      public RowView(Element elem) {
      super(elem);
      }

      protected void loadChildren(ViewFactory f) {
      super.loadChildren(f);
      calculateSpans();
      }

      public float getAlignment(int axis) {
      if (axis == View.Y_AXIS)
            return .5f;
      return .5f;
      }

      float w, w1, h, baseline;
      float x[], y[];

      protected void calculateSpans() {
      w = baseline = 0f;
      float max = -1f;

      int num = getViewCount();
      x = new float[num];
      y = new float[num];
      for (int i = 0; i < this.getViewCount(); i++) {
            View v = getView(i);
            x[i] = v.getPreferredSpan(X_AXIS);
            y[i] = v.getPreferredSpan(Y_AXIS);
            w += x[i];
            if (y[i]>max)
                  max = y[i];
            h = max;
            if (v instanceof BaseLineView) {
                  BaseLineView vb = (BaseLineView) v;
                  float hui = vb.getBaseline();
                  if (hui > baseline)
                        baseline = hui;
            }
      }

      // μέσω αυτού του βρόχου επανάληψης, παίρνουμε το μέγιστο πλάτος
         αθροίζοντας τα πλάτη όλων των όψεων και το μέγιστο ύψος, το οποίο
         θα είναι το ύψος της όψης που θα έχει το μεγαλύτερο ύψος, από
         όλες τις όψεις. Κατόπιν, βρίσκουμε τη βασική γραμμή, η οποία θα

         είναι η βασική γραμμή της όψης με το μεγαλύτερο πάνω ύψος.
      }
```

```
protected  void childAllocation(int index, Rectangle rect) {
      int xi = 0;
      for (int i = 0; i < index; i++) {
            xi += x[i];
      }

      rect.x += 5 + xi;
      rect.width = (int) x[index];
      rect.height = (int) y[index];
      // βρίσκουμε το ύψος και το πλάτος του πλαισίου που θα περιέχει όλες
         τις όψεις.

      if (getView(index) instanceof BaseLineView) {
            BaseLineView bv = (BaseLineView) getView(index);
            rect.y += 5 + baseline  - bv.getBaseline();
      } else
            rect.y += baseline + 19 - rect.height / 2;
      }
      // τοποθετούμε τις όψεις κατά τον y άξονα έχοντας σαν κριτήριο το αν
         είναι υποκλάσεις της κλάσης BaseLineView και τις τοποθετούμε ανάλογα.

      protected View getViewAtPoint(int x, int y, Rectangle rect) {
      for (int i = 0; i < this.getViewCount(); i++){
            if(y < rect.y + View.Y_AXIS) {
                  return getView(i);
            }
            else {
                  return null;
            }
      }
      return null;
      }

      protected boolean isAfter(int x, int y, Rectangle alloc) {
      return y > alloc.y + alloc.height;
      }

      protected boolean isBefore(int x, int y, Rectangle alloc) {
      return y < alloc.y;
      }

      public float getPreferredSpan(int axis) {
      if (axis == X_AXIS)
            return  w + 5;
      return h + 5;
      }
```

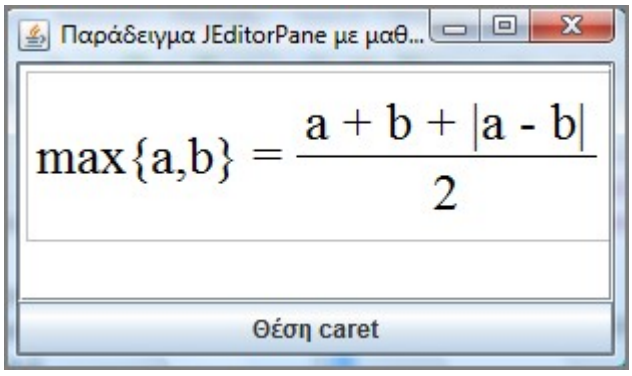
```
public  void paint(Graphics g, Shape s) {

      g.setColor(Color.lightGray);


      Rectangle rect = s.getBounds();
      int x = rect.x;
      int y = rect.y;
      int height = rect.height;
      int width = rect.width;

      g.drawRect(x , y, width + 5, height + 5);

      for (int i = 0; i < this.getViewCount(); i++)
            this.getView(i).paint(g, getChildAllocation(i,s).getBounds());
      // σχεδιάζονται μέσα στο πλαίσιο όλες οι όψεις που θέλουμε να
         απεικονίσουμε.
      }

      public float getBaseline() {
      return baseline;
      }
}
```

Έχοντας εξηγήσει τις λειτουργίες του προγράμματος, ας δούμε κάποια παραδείγματα για το πώς εμφανίζονται οι μαθηματικές εκφράσεις :

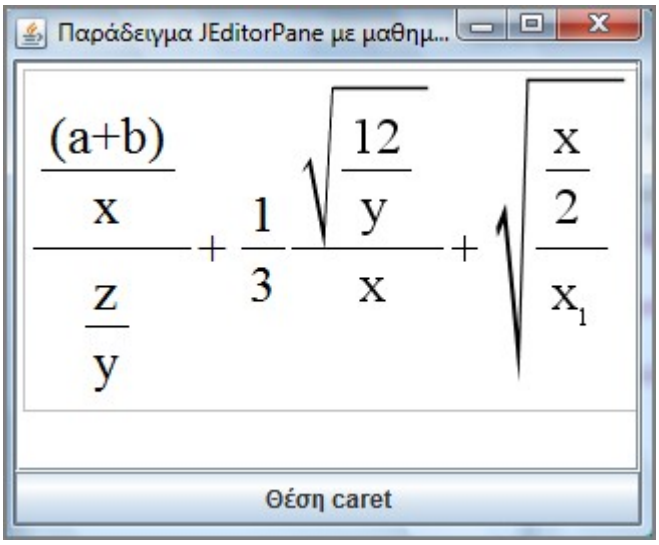
ΕΙΚΟΝΑ 24.



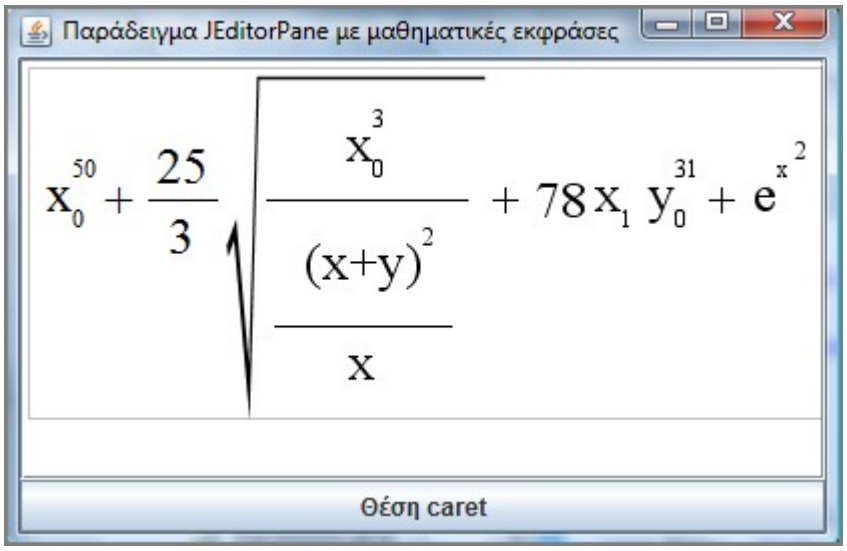
ΕΙΚΟΝΑ 25.



ΕΙΚΟΝΑ 26.



ΕΙΚΟΝΑ 27.



Στις Εικόνες 24 και 25 απεικονίζονται απλές μαθηματικές εκφράσεις. Η πολυπλοκότητα του προγράμματος, αναδεικνύεται περισσότερο στις Εικόνες 26 και 27.

Δυσκολίες και Προβλήματα

Κατά τη διάρκεια της συγγραφής της εργασίας εμφανίστηκαν κάποια προβλήματα τα οποία μας απέτρεψαν από το να επιτύχουμε τον αρχικό μας στόχο, ο οποίος ήταν να κατασκευάσουμε ένα πρόγραμμα το οποίο θα χρησιμοποιούσαν μαθητές Γυμνασίου με το να δίνουν συγκεκριμένες εντολές σε συγκεκριμένα παραδείγματα και το πρόγραμμα θα διαχειριζόταν αυτές τις εντολές. Ωστόσο, η ανάπτυξη εξειδικευμένων εφαρμογών επεξεργασίας (editors) και απεικόνισης (presentation) με χρήση των συστατικών κειμένου του Swing απαιτεί τη γνώση ενός αρκετά εκτεταμένου συνόλου κλάσεων με αποτέλεσμα να είναι δύσκολη και χρονοβόρα.

Η αρχιτεκτονική που υποστηρίζεται από τα συστατικά κειμένου του Swing είναι εύκολα επεκτάσιμη. Ως αποτέλεσμα, η εφαρμογή που παρουσιάστηκε σε αυτή την εργασία είναι εύκολο να επεκταθεί ώστε να υποστηρίζει την απεικόνιση μιας πλήρους γλώσσας μαθηματικών εκφράσεων, πέρα από το υποσύνολο που υποστηρίζεται στην τρέχουσα έκδοση. Έτσι,μπορεί να υποστηρίξει απεικόνιση γλωσσών όπως MathML και LaTeX.

Επίσης, η μετατροπή μιας εφαρμογής απεικόνισης σε εφαρμογή επεξεργασίας είναι εφικτή αλλά και αυτή παρουσιάζει προβλήματα και γι αυτό δεν υλοποιήθηκε στην παρούσα εργασία.

Εντυπώσεις

Από την αρχή το αντικείμενο της εργασίας ήταν πολύ ευχάριστο. Η κατασκευή του προγράμματος, εκτός των προβλεπόμενων δυσκολιών, ήταν μία διαδικασία μάθησης καθόλου κουραστική. Οι εισαγωγικές γνώσεις στη Java δε βοήθησαν τόσο, ωστόσο, ήταν απαραίτητες για την κατανόηση των βασικών λειτουργιών της γλώσσας που αποτελούν προϋπόθεση για την μετάβαση στις πιο πολύπλοκες. Επειδή τα αποτελέσματα των επιμέρους τμημάτων της εργασίας ήταν, άμεσα ορατά, η διαδικασία της συγγραφής του κώδικα γινόταν όλο και πιο ενδιαφέρουσα.

Η Java παρέχει πάρα πολλές δυνατότητες στον προγραμματιστή. Από την αρχή της μάθησης της γλώσσας δεν δυσκολεύεται κάποιος, ειδικά αν έχει ξαναασχοληθεί με τον προγραμματισμό και ακόμα πιο ειδικά με τις γλώσσες C, C++.

Περνώντας στη μάθηση του Swing, τα πράγματα είναι λίγο πιο δύσκολα. Θα πρέπει κάποιος να έχει ασχοληθεί αρκετά με τις αντίστοιχες κλάσεις του AWT ώστε να μπορεί να τις χειρίζεται σε ικανοποιητικό βαθμό, και κατόπιν να μάθει το πώς χρησιμοποιούνται οι υποδοχείς και οι υπόλοιπες ιδιαιτερότητες του Swing.

Όσον αφορά στη μάθηση των συστατικών κειμένου, τα πράγματα είναι ακόμα πιο πολύπλοκα πρώτον γιατί στα περισσότερα συγγράμματα δεν αναφέρονται λεπτομερώς και δεύτερον γιατί για να τα χρησιμοποιήσει κάποιος είναι αρκετά πράγματα που πρέπει πρώτα να μάθει, όπως η ιεραρχία των κλάσεων για τα συστατικά, τί προσφέρει η κάθε κλάση στο πρόγραμμα, τις διαφορές ανάμεσα στις κλάσεις και ούτω καθεξής.

ΒΙΒΛΙΟΓΡΑΦΙΑ

Lemay Laura, Rogers Cadenhead, απόδοση Γιάννης Β. Σαμαράς : *Πλήρες Εγχειρίδιο της Java 2,* Μόσχος Γκιούρδας, 1999

Topley Kim : *Core Swing Advanced Programming,* Prentice Hall PTR, 1999

Andrew W. Appel, Jens Palsberg : *Modern Compiler Implementation in Java*, 2002

Sun Microsystems : *The Java ™ Tutorials,* http://java.sun.com/docs/books/tutorial

Robert Eckstein : *Java SE Application Design with MVC*, http://java.sun.com/developer/technicalArticles/javase/mvc/, 2007

Scott Stanchfield : *Applying MVC in VisualAge for Java, http://www.javadude.com/articles/vaddmvc1/mvc1.htm,* 1996-2009

ΠΑΡΑΡΤΗΜΑ

Κλάση Application

```
package gr.aegean.math;

import java.awt.BorderLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.*;
import javax.swing.text.*;

public class Application extends JFrame {
	JEditorPane edit = new JEditorPane();
	JButton button;
	ExpressionDocument doc;

	public Application() {
		super("Παράδειγμα JEditorPane με μαθηματικές εκφράσεις");
		setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
		edit.setEditable(false);
		edit.setEditorKit(new ExpressionEditorKit());
		doc = (ExpressionDocument) edit.getDocument();

		this.getContentPane().add(new JScrollPane(edit),
					BorderLayout.CENTER);
		button = new JButton("Θέση caret");

		this.getContentPane().add(button, BorderLayout.SOUTH);
		button.addActionListener(new ActionListener() {
			public void actionPerformed(ActionEvent e) {
				int ai;
				button.setLabel(Integer.toString(ai =
						edit.getCaretPosition()));
				Document doc = edit.getDocument();
				Element es = doc.getDefaultRootElement();
				printElement(es, 0);
			}
		});
		pack();
	}

	private void printElement(Element e, int depth) {
		for (int j = 0; j < depth; j++)
			System.out.print("    ");
		System.out.print(e.toString());
		for (int i = 0; i < e.getElementCount(); i++)
			printElement(e.getElement(i), depth + 1);
	}


	public static void main(String[] args) {

		Application m = new Application();
		m.setVisible(true);
	}
}
```

Κλάση ExpressionDocument

```
package gr.aegean.math;

import gr.aegean.math.exp.Expression;
import gr.aegean.math.exp.Fraction;
import gr.aegean.math.exp.Root;
import gr.aegean.math.exp.Row;
import gr.aegean.math.exp.Subscript;
import gr.aegean.math.exp.Subsuperscript;
import gr.aegean.math.exp.Superscript;
import gr.aegean.math.exp.Text;


import java.util.*;

import javax.swing.text.*;

public class ExpressionDocument extends DefaultStyledDocument {
      public static final String ELEMENT_NAME_TEXT = "text";
      public static final String ELEMENT_NAME_FRACTION = "fraction";
      public static final String ELEMENT_NAME_ROOT = "root";
      public static final String ELEMENT_NAME_INDEX_EXPRESSION = "index";
      public static final String ELEMENT_NAME_EXPONENTIAL_EXPRESSION =
                                                            "exponential";
      public static final String ELEMENT_NAME_ROW= "row";
      public static final String ELEMENT_NAME_INDEXEXPONENT_EXPRESSION =
                                                            "indexeponent";


      public ExpressionDocument() {

            Subsuperscript a = new Subsuperscript( new Text ("x"), new Text
                                                      ("50"), new Text("0"));
            Subsuperscript b = new Subsuperscript( new Text ("x"), new Text
                                                       ("3"), new Text("0"));
            Root c = new Root (new Fraction(b, new Fraction (new Superscript
                        (new Text (" (x+y)"), new Text ("2")), new Text ("x"))));
            Subscript d = new Subscript(new Text ("x"), new Text("1"));
            Subsuperscript f = new Subsuperscript( new Text ("y"), new Text
                                                      ("31"), new Text("0"));
            Superscript g = new Superscript (new Superscript (new Text ("e"),
                                          new Text ("x")), new Text ("2"));
            Row row = new Row();

            row.add(a);
            row.add(new Text ("+ "));
            row.add(new Fraction (new Text("25"), new Text("3")));
            row.add(c);
            row.add(new Text("+ "));
            row.add(new Text("78"));
            row.add(d);
            row.add(f);
            row.add(new Text("+"));
            row.add(g);

            init(row);
      }
```

```
	public ExpressionDocument(Expression exp) {
		init(exp);
	}

	public void init(Expression e) {

		try {

			ArrayList<ElementSpec> es = new ArrayList<ElementSpec>();
			SimpleAttributeSet attrs = new SimpleAttributeSet();

			es.add(new ElementSpec(attrs, ElementSpec.EndTagType));

			e.toDocument(es);

			es.add(new ElementSpec(attrs, ElementSpec.StartTagType));

			ElementSpec[] spec = new ElementSpec[es.size()];
			es.toArray(spec);

			insert(0, spec);

		} catch (BadLocationException e1) {

			e1.printStackTrace();
		}
	}
}
```

Κλάση ExpressionEditorKit

```
package gr.aegean.math;

import javax.swing.text.Document;
import javax.swing.text.StyledEditorKit;
import javax.swing.text.ViewFactory;

class ExpressionEditorKit extends StyledEditorKit {

	ViewFactory defaultFactory = new ExpressionFactory();

	public ViewFactory getViewFactory() {
		return defaultFactory;
	}

	public Document createDefaultDocument() {

		return new ExpressionDocument();
	}
}
```

Κλάση ExpressionFactory

```
package gr.aegean.math;

import gr.aegean.math.views.ExpLabelView;
import gr.aegean.math.views.ExponentView;
import gr.aegean.math.views.IndexExponentView;
import gr.aegean.math.views.FractionView;
import gr.aegean.math.views.RootView;
import gr.aegean.math.views.IndexView;
import gr.aegean.math.views.RowView;

import javax.swing.text.AbstractDocument;
import javax.swing.text.BoxView;
import javax.swing.text.ComponentView;
import javax.swing.text.Element;
import javax.swing.text.LabelView;
import javax.swing.text.ParagraphView;
import javax.swing.text.StyleConstants;
import javax.swing.text.View;
import javax.swing.text.ViewFactory;

class ExpressionFactory implements ViewFactory {
      public View create(Element elem) {
            String kind = elem.getName();
            if (kind != null) {
                  if (kind.equals(AbstractDocument.ContentElementName)) {
                        return new ExpLabelView(elem);
                  }else if (kind.equals(AbstractDocument.ParagraphElementName))
{
                        return new ParagraphView(elem);
                  } else if (kind.equals(AbstractDocument.SectionElementName)) {
                        return new BoxView(elem, View.Y_AXIS);
                  } else if (kind.equals(StyleConstants.ComponentElementName)) {
                        return new ComponentView(elem);
                  } else if
                      (kind.equals(ExpressionDocument.ELEMENT_NAME_FRACTION)) {
                        return new FractionView(elem);
                  } else if
          (kind.equals(ExpressionDocument.ELEMENT_NAME_EXPONENTIAL_EXPRESSION)) {
                        return new ExponentView(elem);
                  } else if (kind.equals(ExpressionDocument.ELEMENT_NAME_ROOT)){
                        return new RootView(elem);
                  } else if
                (kind.equals(ExpressionDocument.ELEMENT_NAME_INDEX_EXPRESSION)) {
                        return new IndexView(elem);
                  } else if
        (kind.equals(ExpressionDocument.ELEMENT_NAME_INDEXEXPONENT_EXPRESSION)) {
                        return new IndexExponentView(elem);
                  } else if (kind.equals(ExpressionDocument.ELEMENT_NAME_ROW)) {
                        return new RowView(elem);

                  }
            }

            return new LabelView(elem);
      }
}
```

Κλάση Expression

```
package gr.aegean.math.exp;

import java.util.ArrayList;

public abstract class Expression {

      public abstract void toDocument(ArrayList specs);

}
```

Κλάση Fraction

```
package gr.aegean.math.exp;

import gr.aegean.math.ExpressionDocument;
import java.util.ArrayList;

import javax.swing.text.*;
import javax.swing.text.DefaultStyledDocument.ElementSpec;


public class Fraction extends Expression {
	Expression nom, denom;

	public Fraction(Expression nom, Expression denom) {
		this.nom = nom;
		this.denom = denom;
	}

	public void toDocument(ArrayList specs) {

		SimpleAttributeSet attributes = new SimpleAttributeSet();

		attributes.addAttribute(AbstractDocument.ElementNameAttribute,
					ExpressionDocument.ELEMENT_NAME_FRACTION);

		ElementSpec start = new ElementSpec(attributes,
							ElementSpec.StartTagType);
		specs.add(start);

		nom.toDocument(specs);
		denom.toDocument(specs);

		ElementSpec end = new ElementSpec(attributes,
				ElementSpec.EndTagType);
		specs.add(end);
	}
}
```

Κλάση Root

```
package gr.aegean.math.exp;

import gr.aegean.math.ExpressionDocument;
import java.util.ArrayList;

import javax.swing.text.AbstractDocument;
import javax.swing.text.SimpleAttributeSet;
import javax.swing.text.DefaultStyledDocument.ElementSpec;


public class Root extends Expression{
	Expression sub;

	public Root(Expression exp){
		this.sub = exp;
	}

	public void toDocument(ArrayList specs) {

		SimpleAttributeSet attributes = new SimpleAttributeSet();

		attributes.addAttribute(AbstractDocument.ElementNameAttribute,
					ExpressionDocument.ELEMENT_NAME_ROOT);

		ElementSpec start = new ElementSpec(attributes,
					ElementSpec.StartTagType);
		specs.add(start);

		sub.toDocument(specs);

		ElementSpec end = new ElementSpec(attributes,
					ElementSpec.EndTagType);
		specs.add(end);
	}

}
```

Κλάση Row

```
package gr.aegean.math.exp;

import gr.aegean.math.ExpressionDocument;

import java.util.ArrayList;
import java.util.Iterator;
import java.util.Vector;

import javax.swing.text.AbstractDocument;
import javax.swing.text.SimpleAttributeSet;
import javax.swing.text.DefaultStyledDocument.ElementSpec;


public class Row extends Expression {
	Vector<Expression> expressions;

	public Row() {
		expressions = new Vector<Expression>();
	}

	public void add(Expression exp) {
		expressions.add(exp);
	}

	public Iterator<Expression> getExpressions() {
		return expressions.iterator();
	}

	public void toDocument(ArrayList specs) {

		SimpleAttributeSet attributes = new SimpleAttributeSet();

		attributes.addAttribute(AbstractDocument.ElementNameAttribute,
				ExpressionDocument.ELEMENT_NAME_ROW);

		ElementSpec start = new ElementSpec(attributes,
				ElementSpec.StartTagType);
		specs.add(start);

		for (Expression e : expressions) {
			e.toDocument(specs);
		}

		ElementSpec end = new ElementSpec(attributes,
                                                    ElementSpec.EndTagType);
		specs.add(end);

	}
}
```

Κλάση Subscript

```
package gr.aegean.math.exp;

import gr.aegean.math.ExpressionDocument;

import java.util.ArrayList;

import javax.swing.text.AbstractDocument;
import javax.swing.text.SimpleAttributeSet;
import javax.swing.text.DefaultStyledDocument.ElementSpec;


public class Subscript extends Expression{

	Expression basi, index;

	public Subscript(Expression basi, Expression subscript){

		this.basi = basi;
		this.index = subscript;
	}

	public void toDocument(ArrayList specs) {

		SimpleAttributeSet attributes = new SimpleAttributeSet();

		attributes.addAttribute(AbstractDocument.ElementNameAttribute,
				ExpressionDocument.ELEMENT_NAME_INDEX_EXPRESSION);

		ElementSpec start = new ElementSpec(attributes,
				ElementSpec.StartTagType);
		specs.add(start);

		basi.toDocument(specs);
		index.toDocument(specs);

		ElementSpec end = new ElementSpec(attributes,
				ElementSpec.EndTagType);
		specs.add(end);

	}
}
```

Κλάση Subsuperscript

```
package gr.aegean.math.exp;

import gr.aegean.math.ExpressionDocument;

import java.util.ArrayList;

import javax.swing.text.AbstractDocument;
import javax.swing.text.SimpleAttributeSet;
import javax.swing.text.DefaultStyledDocument.ElementSpec;

public class Subsuperscript extends Expression {
      Expression base, sup,sub;
public Subsuperscript(Expression base, Expression superscript, Expression
subscript){
            this.base = base;
            this.sup = superscript;
            this.sub = subscript;
      }

public void toDocument(ArrayList specs) {

      SimpleAttributeSet attributes = new SimpleAttributeSet();

      attributes.addAttribute(AbstractDocument.ElementNameAttribute,
                  ExpressionDocument.ELEMENT_NAME_INDEXEXPONENT_EXPRESSION);

      ElementSpec start = new ElementSpec(attributes,
                  ElementSpec.StartTagType);
      specs.add(start);

      base.toDocument(specs);
      sup.toDocument(specs);
      sub.toDocument(specs);

      ElementSpec end = new ElementSpec(attributes, ElementSpec.EndTagType);
      specs.add(end);

      }
}
```

Κλάση Superscript

```
package gr.aegean.math.exp;

import gr.aegean.math.ExpressionDocument;

import java.util.ArrayList;

import javax.swing.text.AbstractDocument;
import javax.swing.text.SimpleAttributeSet;
import javax.swing.text.DefaultStyledDocument.ElementSpec;


public class Superscript extends Expression{
	Expression base, exponent;
	public Superscript(Expression base, Expression superscript){
		this.base = base;
		this.exponent = superscript;
	}

	public void toDocument(ArrayList specs) {

		SimpleAttributeSet attributes = new SimpleAttributeSet();

		attributes.addAttribute(AbstractDocument.ElementNameAttribute,
				ExpressionDocument.ELEMENT_NAME_EXPONENTIAL_EXPRESSION);

		ElementSpec start = new ElementSpec(attributes,
				ElementSpec.StartTagType);
		specs.add(start);

		base.toDocument(specs);
		exponent.toDocument(specs);

		ElementSpec end = new ElementSpec(attributes,
				ElementSpec.EndTagType);
		specs.add(end);

	}
}
```

Κλάση Text

```
package gr.aegean.math.exp;

import java.util.ArrayList;

import javax.swing.text.SimpleAttributeSet;
import javax.swing.text.DefaultStyledDocument.ElementSpec;

public class Text extends Expression {
	String text;

	public Text(String text) {
		this.text = text;
	}

	public void toDocument(ArrayList specs) {

		SimpleAttributeSet attributes = new SimpleAttributeSet();

		ElementSpec textSpec = new ElementSpec(attributes,
				ElementSpec.ContentType, text.toCharArray(), 0,
                                                        text.length());
		specs.add(textSpec);

	}

}
```

Κλάση BaseLineView

```
package gr.aegean.math.views;

import javax.swing.text.Element;

public abstract class BaseLineView extends ExpCompositeView {

	public BaseLineView(Element elem) {
		super(elem);
	}

	public abstract float getBaseline();

}
```

Κλάση ExpCompositeView

```
package gr.aegean.math.views;

import javax.swing.text.CompositeView;
import javax.swing.text.Element;
import javax.swing.text.View;

public abstract class ExpCompositeView extends CompositeView {
	int sDepth;

	public ExpCompositeView(Element elem) {
		super(elem);
		sDepth = 1;
	}

	public void setParent(View parent) {
		super.setParent(parent);
		if (!(parent instanceof ExpCompositeView))
			sDepth = 1;
		else {
			ExpCompositeView ev = (ExpCompositeView) parent;
			if (parent instanceof IndexExponentView) {
				sDepth = ev.getDepth() + 1;
			}
			 else

			sDepth = ev.getDepth();
		}
	}

	protected void setDepth(int depth) {
		this.sDepth = depth;
	}

	protected int getDepth() {
		return sDepth;
	}

}
```

Κλάση ExpLabelView

```
package gr.aegean.math.views;

import java.awt.Font;

import javax.swing.text.Element;
import javax.swing.text.LabelView;
import javax.swing.text.View;

public class ExpLabelView extends LabelView {

	public ExpLabelView(Element elem) {
		super(elem);
	}

	public Font getFont() {
		int size = 1;

		View parent = getParent();
		View child = this;
		while (parent != null) {
			if ((parent instanceof ExponentView && child ==
									parent.getView(1)))
				size++;
			else if((parent instanceof IndexView && child ==
									parent.getView(1)))
				size++;
			else if((parent instanceof IndexExponentView && (child ==
					parent.getView(1) || child == parent.getView(2))))
				size++;
			child = parent;
			parent = parent.getParent();
		}

		size = 26 / size;

		return new Font("Times New Roman", Font.PLAIN, size);
	}

	public float getAlignment(int axis) {
		if (axis == View.Y_AXIS)
			return .5f;
		return .5f;
	}

}
```

Κλάση ExponentView

```
package gr.aegean.math.views;

import java.awt.Component;
import java.awt.Graphics;
import java.awt.Rectangle;
import java.awt.Shape;

import javax.swing.event.DocumentEvent;
import javax.swing.text.Element;
import javax.swing.text.View;
import javax.swing.text.ViewFactory;

public class ExponentView extends BaseLineView {
	public ExponentView(Element elem) {
		super(elem);
	}

	protected void loadChildren(ViewFactory f) {
		super.loadChildren(f);
		calculateSpans();
	}

	protected void forwardUpdate(DocumentEvent.ElementChange ec,
			DocumentEvent e, Shape a, ViewFactory f) {
		super.forwardUpdate(ec, e, a, f);
		calculateSpans();
		Component c = getContainer();
		c.repaint();
	}

	float w1, h1, w2, h2;
	int inset = 4;

	protected void calculateSpans() {
		w1 = getView(0).getPreferredSpan(X_AXIS);
		h1 = getView(0).getPreferredSpan(Y_AXIS);
		w2 = getView(1).getPreferredSpan(X_AXIS);
		h2 = getView(1).getPreferredSpan(Y_AXIS);
	}

	protected void childAllocation(int index, Rectangle rect) {
		if (rect != null) {
			if (index == 0){
				rect.x += inset;
				rect.y += .6f * h2;

				rect.width = (int) w1;
				rect.height = (int) h1;
			}
			else if (index == 1){
				rect.x += w1 + inset;
				rect.y += inset;

				rect.width = (int) w2;
				rect.height = (int) h2;
			}
		}
```

```
            else

                  rect = null;
      }

      protected View getViewAtPoint(int x, int y, Rectangle rect) {
            if (x < rect.x + w1 + inset) {
                  childAllocation(0, rect);
                  return getView(0);
            } else {
                  childAllocation(1, rect);
                  return getView(1);
            }
      }

      protected boolean isAfter(int x, int y, Rectangle alloc) {
            return y > alloc.y + alloc.height ;
            }

      protected boolean isBefore(int x, int y, Rectangle alloc) {
            return y < alloc.y;
      }

      public float getPreferredSpan(int axis) {
            if (axis == X_AXIS)
                  return w1 + w2 + 2 * inset;
            return  h2 + h1 + 2 * inset;
      }

      public void paint(Graphics g, Shape shape) {

            Rectangle r1 = getChildAllocation(0, shape).getBounds();

            Rectangle r2 = getChildAllocation(1, shape).getBounds();

            getView(0).paint(g, r1);
            getView(1).paint(g, r2);

      }

      public float getBaseline() {
            View child1 = getView(0);
            View child2 = getView(1);
            if (child1 instanceof BaseLineView ) {
                  BaseLineView bv = (BaseLineView) child1;
                  return bv.getBaseline() + .6f * h2;
            }else if (child2 instanceof BaseLineView) {
                  return  .6f * h2 + h1/8 ;
            }
            return child1.getPreferredSpan(Y_AXIS) / 2 - inset;
      }

}
```

Κλάση FractionView

```
package gr.aegean.math.views;

import java.awt.Color;
import java.awt.Component;
import java.awt.Graphics;
import java.awt.Rectangle;
import java.awt.Shape;

import javax.swing.event.DocumentEvent;
import javax.swing.text.Element;
import javax.swing.text.View;
import javax.swing.text.ViewFactory;

public class FractionView extends BaseLineView {

	public FractionView(Element elem) {
		super(elem);
	}

	protected void forwardUpdate(DocumentEvent.ElementChange ec,
			DocumentEvent e, Shape a, ViewFactory f) {
		super.forwardUpdate(ec, e, a, f);
		calculateSpans();
		Component c = getContainer();
		c.repaint();
	}

	public float getAlignment(int axis)
	{
		if (axis== View.Y_AXIS)
			return .5f;
		return .5f;
	}

	protected void loadChildren(ViewFactory f) {
		super.loadChildren(f);
		calculateSpans();
	}

	float w1, h1, w2, h2;
	int inset = 4;

	protected void calculateSpans()
	{
		w1 = getView(0).getPreferredSpan(X_AXIS);
		h1 = getView(0).getPreferredSpan(Y_AXIS);
		w2 = getView(1).getPreferredSpan(X_AXIS);
		h2 = getView(1).getPreferredSpan(Y_AXIS);
	}

	protected void childAllocation(int index, Rectangle rect) {
		if (rect != null) {
			if (index == 0) {
				if (w2 > w1)
					rect.x += .5 * (w2 - w1) + inset;
				else
					rect.x += inset;
```

```
                        rect.y += inset;

                        rect.width = (int) w1;
                        rect.height = (int) h1;

                  } else if (index == 1) {
                        if (w2 > w1)
                              rect.x +=  inset;
                        else
                              rect.x += .5 * (w1 - w2) + inset;
                        rect.y += 2 * inset + h1;
                        rect.width = (int) w2;
                        rect.height = (int) h2;
                  } else
                        rect = null;
            }
      }

      protected View getViewAtPoint(int x, int y, Rectangle rect) {
            if (y < rect.y + h1 + inset + 2) {
                  childAllocation(0, rect);
                  return getView(0);
            } else {
                  childAllocation(1, rect);
                  return getView(1);
            }
      }

      protected boolean isAfter(int x, int y, Rectangle alloc) {
            return y > alloc.y + alloc.height;
      }

      protected boolean isBefore(int x, int y, Rectangle alloc) {
            return y < alloc.y;
      }
      public float getPreferredSpan(int axis) {
            if (axis == X_AXIS)
                  return 3 * inset + Math.max(w1, w2);
            return 3 * inset + h1 + h2;
      }
      public void paint(Graphics g, Shape shape) {
            g.setColor(Color.lightGray);
            Rectangle rect = shape.getBounds();
            int x = rect.x;
            int y = rect.y;

            Rectangle r1 = getChildAllocation(0, shape).getBounds();

            Rectangle r2 = getChildAllocation(1, shape).getBounds();

            getView(0).paint(g, r1);
            getView(1).paint(g, r2);

            g.drawLine(x, inset + 2 + (int) h1 + y , 2 * inset + (int)
                              Math.max(w1,w2) + x , inset + 2 + (int)h1 +y);
      }
      public float getBaseline() {
            return h1 - 2 * inset - 1;
      }
}
```

Κλάση IndexExponentView

```
package gr.aegean.math.views;

import java.awt.Component;
import java.awt.Graphics;
import java.awt.Rectangle;
import java.awt.Shape;

import javax.swing.event.DocumentEvent;
import javax.swing.text.Element;
import javax.swing.text.View;
import javax.swing.text.ViewFactory;

public class IndexExponentView extends BaseLineView {

	public IndexExponentView(Element elem) {
		super(elem);
	}

	protected void loadChildren(ViewFactory f) {
		super.loadChildren(f);
		calculateSpans();
	}

	protected void forwardUpdate(DocumentEvent.ElementChange ec,
			DocumentEvent e, Shape a, ViewFactory f) {
		super.forwardUpdate(ec, e, a, f);
		calculateSpans();
		Component c = getContainer();
		c.repaint();
	}

	float w1, h1, w2, h2, w3, h3;
	int inset = 4;

	protected void calculateSpans() {
		w1 = getView(0).getPreferredSpan(X_AXIS);
		h1 = getView(0).getPreferredSpan(Y_AXIS);
		w2 = getView(1).getPreferredSpan(X_AXIS);
		h2 = getView(1).getPreferredSpan(Y_AXIS);
		w3 = getView(2).getPreferredSpan(X_AXIS);
		h3 = getView(2).getPreferredSpan(Y_AXIS);
	}

	protected void childAllocation(int index, Rectangle rect) {
		if (rect != null) {
			if (index == 0){
				rect.x += inset;
				rect.y += .6f * h2;

				rect.width = (int) w1;
				rect.height = (int) h1;
			}
			else if (index == 1){
				rect.x += inset + w1;
				rect.y += inset;

				rect.width = (int) w2;
```

```
                rect.height = (int) h2;

            }

            else if (index == 2){
                rect.x += w1 + inset;
                rect.y +=   h1 - inset / 2;//.6f * h2 +

                rect.width = (int) w3;
                rect.height = (int) h3;
            }
        }
        else
            rect = null;
}

protected View getViewAtPoint(int x, int y, Rectangle rect) {
        if (y > .9f * h2 && y < h1/2 + h2) {
            childAllocation(0, rect);
            return getView(0);
        } else if (y > h1 + h2){
            childAllocation(2, rect);
            return getView(2);
        } else {
            childAllocation(1, rect);
            return getView(1);
        }
}

protected boolean isAfter(int x, int y, Rectangle alloc) {
        return x > alloc.x + alloc.width ;
}

protected boolean isBefore(int x, int y, Rectangle alloc) {
        return x < alloc.x;
}

public float getPreferredSpan(int axis) {
        if (axis == X_AXIS)
            return  w1 + Math.max(w2,w3) + 2 * inset;
        return  h1 + h2 + inset;
}

public void paint(Graphics g, Shape shape) {
        Rectangle r1 = getChildAllocation(0, shape).getBounds();

        Rectangle r2 = getChildAllocation(1, shape).getBounds();

        Rectangle r3 = getChildAllocation(2, shape).getBounds();

        getView(0).paint(g, r1);
        getView(1).paint(g, r2);
        getView(2).paint(g, r3);
}

public float getBaseline() {
        View child1 = getView(0);
        View child2 = getView(1);

        if (child1 instanceof BaseLineView) {
```

```
                BaseLineView bv = (BaseLineView) child1;
                return bv.getBaseline() + .6f * h2;

            }else if(child2 instanceof BaseLineView)
                return  .6f * h2 + h1/8;

            return child1.getPreferredSpan(Y_AXIS) / 2 - inset/2 - 1;
        }
    }
```

Κλάση IndexView

```
package gr.aegean.math.views;

import java.awt.Component;
import java.awt.Graphics;
import java.awt.Rectangle;
import java.awt.Shape;

import javax.swing.event.DocumentEvent;
import javax.swing.text.Element;
import javax.swing.text.View;
import javax.swing.text.ViewFactory;

public class IndexView extends BaseLineView {

	public IndexView(Element elem) {
		super(elem);
	}

	public float getAlignment(int axis)
	{
		if (axis== View.Y_AXIS)
			return .5f; //

		return .5f;
	}

	protected void loadChildren(ViewFactory f) {
		super.loadChildren(f);
		calculateSpans();
	}

	protected void forwardUpdate(DocumentEvent.ElementChange ec,
			DocumentEvent e, Shape a, ViewFactory f) {
		super.forwardUpdate(ec, e, a, f);
		calculateSpans();
		Component c = getContainer();
		c.repaint();
	}

	float w1, h1, w2, h2;
	int inset = 4;

	protected void calculateSpans() {
		w1 = getView(0).getPreferredSpan(X_AXIS);
		h1 = getView(0).getPreferredSpan(Y_AXIS);
		w2 = getView(1).getPreferredSpan(X_AXIS);
		h2 = getView(1).getPreferredSpan(Y_AXIS);
	}

	protected void childAllocation(int index, Rectangle rect) {
		if (rect != null) {
			if (index == 0){
				rect.x += inset;
				rect.y += inset;

				rect.width = (int) w1;
				rect.height = (int) h1;
```

```
                }

                else if (index == 1){
                        rect.x += inset + w1;
                        rect.y += h1 - 1.5 * inset - 1;

                        rect.width = (int) w2;
                        rect.height = (int) h2;
                }
        }
        else
                rect = null;
    }

    protected View getViewAtPoint(int x, int y, Rectangle rect) {
        if (x < rect.x + w1) {
                childAllocation(0, rect);
                return getView(0);
        } else {
                childAllocation(1, rect);
                return getView(1);
        }
    }

    protected boolean isAfter(int x, int y, Rectangle alloc) {
        return y > alloc.y + alloc.height ;
        }

    protected boolean isBefore(int x, int y, Rectangle alloc) {
        return y < alloc.y;
    }

    public float getPreferredSpan(int axis) {
        if (axis == X_AXIS)
                return 2 * inset + w1 + w2;
        return 2 * inset + h1 + h2 ;
    }

    public void paint(Graphics g, Shape shape) {

        Rectangle r1 = getChildAllocation(0, shape).getBounds();

        Rectangle r2 = getChildAllocation(1, shape).getBounds();

        getView(0).paint(g, r1);
        getView(1).paint(g, r2);

    }

    public float getBaseline() {
        View child = getView(0);
        if (child instanceof BaseLineView) {
                BaseLineView bv = (BaseLineView) child;
                return bv.getBaseline();
        }
        return child.getPreferredSpan(Y_AXIS) / 2 - 2 *inset;
    }

}
```

Κλάση RootView

```
package gr.aegean.math.views;

import java.awt.BasicStroke;
import java.awt.Color;
import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.Rectangle;
import java.awt.RenderingHints;
import java.awt.Shape;
import java.awt.Stroke;

import javax.swing.text.Element;
import javax.swing.text.View;
import javax.swing.text.ViewFactory;

import java.awt.Font;
import java.awt.geom.AffineTransform;

public class RootView extends BaseLineView {

	public RootView(Element elem) {
		super(elem);
	}

	protected void loadChildren(ViewFactory f) {
		super.loadChildren(f);
		calculateSpans();
	}

	float w1, h1, w0;
	int inset = 4;

	protected void calculateSpans() {
		w1 = getView(0).getPreferredSpan(X_AXIS);
		h1 = getView(0).getPreferredSpan(Y_AXIS);
		w0 = 17f;

	}

	protected void childAllocation(int index, Rectangle rect) {
		if (rect != null) {

			if (index == 0) {

				rect.x += w0 + inset ;
				rect.y += inset;

				rect.width = (int) w1;
				rect.height = (int) h1;
			}
		} else
			rect = null;
	}

	protected View getViewAtPoint(int x, int y, Rectangle rect) {
		if (rect.x < rect.width + inset)
			return getView(0);
```

```
        else

                return null;
}

protected boolean isAfter(int x, int y, Rectangle alloc) {
        return y > alloc.y + alloc.height;
}

protected boolean isBefore(int x, int y, Rectangle alloc) {
        return y < alloc.y;
}

public float getPreferredSpan(int axis) {
        if (axis == X_AXIS) {
                return w1 + 3 * inset + w0;
        }
        return inset + h1;
}

public void paint(Graphics g, Shape shape) {

        Rectangle rect = shape.getBounds();

        int x = rect.x;
        int y = rect.y;
        int height = rect.height;

        Font f = new Font("Times New Roman", Font.PLAIN, 32);
        g.setFont(f);

        Graphics2D g2 = (Graphics2D) g;


        g2.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
                        RenderingHints.VALUE_ANTIALIAS_ON);

        float scale = 1.095f * height/32.f;
        Rectangle r1 = getChildAllocation(0, shape).getBounds();

        g2.setColor(Color.black);
        AffineTransform at = new AffineTransform();
        at.scale(1, scale);
        AffineTransform oldTrans = g2.getTransform();
        g2.transform(at);

        g2.drawString("\u221A", rect.x, (y  + (int) height)/scale);

        g2.setTransform(oldTrans);
        Stroke oldStroke = g2.getStroke();
        g2.setStroke(new BasicStroke(1.5f));

        g2.drawLine(x + (int) w0 , y , x + (int) w0 + r1.width + 2 * inset,
                                                                        y);
        g2.setStroke(oldStroke);

        getView(0).paint(g, r1);

}
```

```
	public float getBaseline() {

		View child = getView(0);
		if (child instanceof BaseLineView) {
			BaseLineView bv = (BaseLineView) child;
			return bv.getBaseline() + inset ;
		}
		return getPreferredSpan(Y_AXIS) / 4;
	}

}
```

Κλάση RowView

```
package gr.aegean.math.views;

import java.awt.Color;
import java.awt.Graphics;
import java.awt.Rectangle;
import java.awt.Shape;

import javax.swing.text.Element;
import javax.swing.text.View;
import javax.swing.text.ViewFactory;

public class RowView extends BaseLineView {

	public RowView(Element elem) {
		super(elem);
	}

	protected void loadChildren(ViewFactory f) {
		super.loadChildren(f);
		calculateSpans();
	}

	public float getAlignment(int axis) {
		if (axis == View.Y_AXIS)
			return .5f;
		return .5f;
	}

	float w, w1, h, baseline;
	float x[], y[];

	protected void calculateSpans() {
		w = baseline = 0f;
		float max = -1f;
		int num = getViewCount();
		x = new float[num];
		y = new float[num];
		for (int i = 0; i < this.getViewCount(); i++) {
			View v = getView(i);
			x[i] = v.getPreferredSpan(X_AXIS);
			y[i] = v.getPreferredSpan(Y_AXIS);
			w += x[i];
			if (y[i]>max)
				max = y[i];
			h = max;
			if (v instanceof BaseLineView) {
				BaseLineView vb = (BaseLineView) v;
				float hui = vb.getBaseline();
				if (hui > baseline)
					baseline = hui;
			}
		}
	}

	protected void childAllocation(int index, Rectangle rect) {
		int xi = 0;
		for (int i = 0; i < index; i++) {
```

```
            xi += x[i];

        }


        rect.x += 5 + xi;
        rect.width = (int) x[index];
        rect.height = (int) y[index];

        if (getView(index) instanceof BaseLineView) {
            BaseLineView bv = (BaseLineView) getView(index);
            rect.y += 5 + baseline  - bv.getBaseline();
        } else
            rect.y += baseline + 19 - rect.height / 2;
    }

    protected View getViewAtPoint(int x, int y, Rectangle rect) {
        for (int i = 0; i < this.getViewCount(); i++){

            if(y < rect.y + View.Y_AXIS) {
                return getView(i);
            }
            else {
                return null;
            }
        }
        return null;
    }

    protected boolean isAfter(int x, int y, Rectangle alloc) {
        return y > alloc.y + alloc.height;
    }

    protected boolean isBefore(int x, int y, Rectangle alloc) {
        return y < alloc.y;
    }

    public float getPreferredSpan(int axis) {
        if (axis == X_AXIS)
            return  w + 5;
        return h + 5;
    }

    public void paint(Graphics g, Shape s) {

        g.setColor(Color.lightGray);

        Rectangle rect = s.getBounds();
        int x = rect.x;
        int y = rect.y;
        int height = rect.height;
        int width = rect.width;

        g.drawRect(x , y, width + 5, height + 5);

        for (int i = 0; i < this.getViewCount(); i++)
            this.getView(i).paint(g, getChildAllocation(i,s).getBounds());

    }
```

```
	public float getBaseline() {
		return baseline;
	}
}
```