

Κεφάλαιο Εννέα
-
**Ο Δρόμος της Sun:
Java και JavaBeans**

9.1 Java

- Σχεδόν καθαρά αντικειμενοστρεφείς γλώσσα, που ορίζει και διαχειρίζεται μεθόδους τάξεων, οι παράμετροι των οποίων καθορίζουν συνθήκες.
- Τάξεις που δεν είναι αντικείμενα κληρονομούν διασύνδεση και εφαρμογή από κάποια άλλη τάξη.
- Τα αντικείμενα αποτελούν περιπτώσεις τάξεων ή πίνακες.
- Μία τάξη μπορεί να εφαρμόσει πολλές διασυνδέσεις και κάθε διασύνδεση μπορεί να εμπλέκεται σε πολλαπλές διαδοχές διασυνδέσεων.
- Στη Java όλες οι τάξεις και διασυνδέσεις περιέχονται σε άλλη τάξη ή μέθοδο. Έτσι έχουμε επίπεδα περιεχομένων – πακέτα. Τάξεις που ανήκουν στο ίδιο σύνολο- πακέτο μπορούν να έχουν πρόσβαση μεταξύ τους.
- Στα πακέτα αυτά των τάξεων υπάρχει ιεραρχία η οποία όμως δεν έχει ισχύ σε θέματα συμπερίληψης και προστασίας τάξεων.
- Η ονομασία των τάξεων είναι σχετική με τον δρόμο διαδοχής στη ιεραρχία του πακέτου που την περιλαμβάνει. Συστήνεται όπως υπάρχει ένα μοναδικό όνομα σε όλο το δίκτυο στο πακέτο του υψηλότερου επιπέδου.

`package.Type.feature`

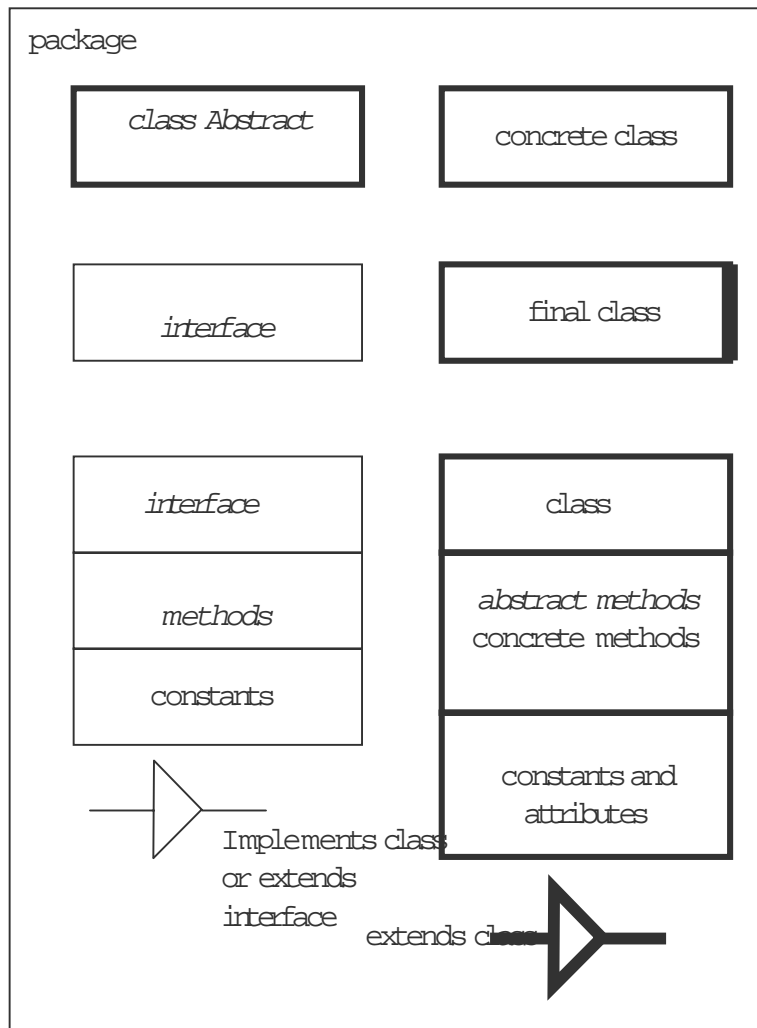
Package = όνομα διαδοχής πακέτων

Type = όνομα τάξης ή διασύνδεσης

Feature = όνομα μεθόδου ή παραμέτρου

- Στο σχήμα 9.1 δίδεται η σημαιογραφία για πακέτα, διασυνδέσεις, τάξεις και μεθόδους. *Τελικές τάξεις* είναι αυτές που δεν έχουν διαδοχή σε άλλες τάξεις.
- Στο σχήμα 9.2 δίδεται το πακέτο `java.awt.image` σε σχέση με το πακέτο `java.lang`.
- Τα τρία βασικά πακέτα χαρακτηριστικών της Java είναι
 - `Java.lang` που εμπλέκεται στα χαρακτηριστικά της γλώσσας. Στη έκδοση JDK1.0 δίδει 21 τάξεις και δύο διασυνδέσεις που περιλαμβάνουν συνολικά 354 μεθόδους. Στη JDK 1.1 οι αριθμοί αυτοί είναι πολύ μεγαλύτεροι.
 - `Java.util`
 - `Java.io`
- Μετάξυ της γλώσσας και των βασικών αυτών πακέτων υπάρχει αρκετό μπλέξιμο, ως αναφορά το αυτόματο πέρασμα τάξεων από το `Java.lang` σε άλλα πακέτα, αλλά και το κτίσιμο τάξεων (`Object`, `String`, `Throwable`), από τη γλώσσα στο `Java.lang`.

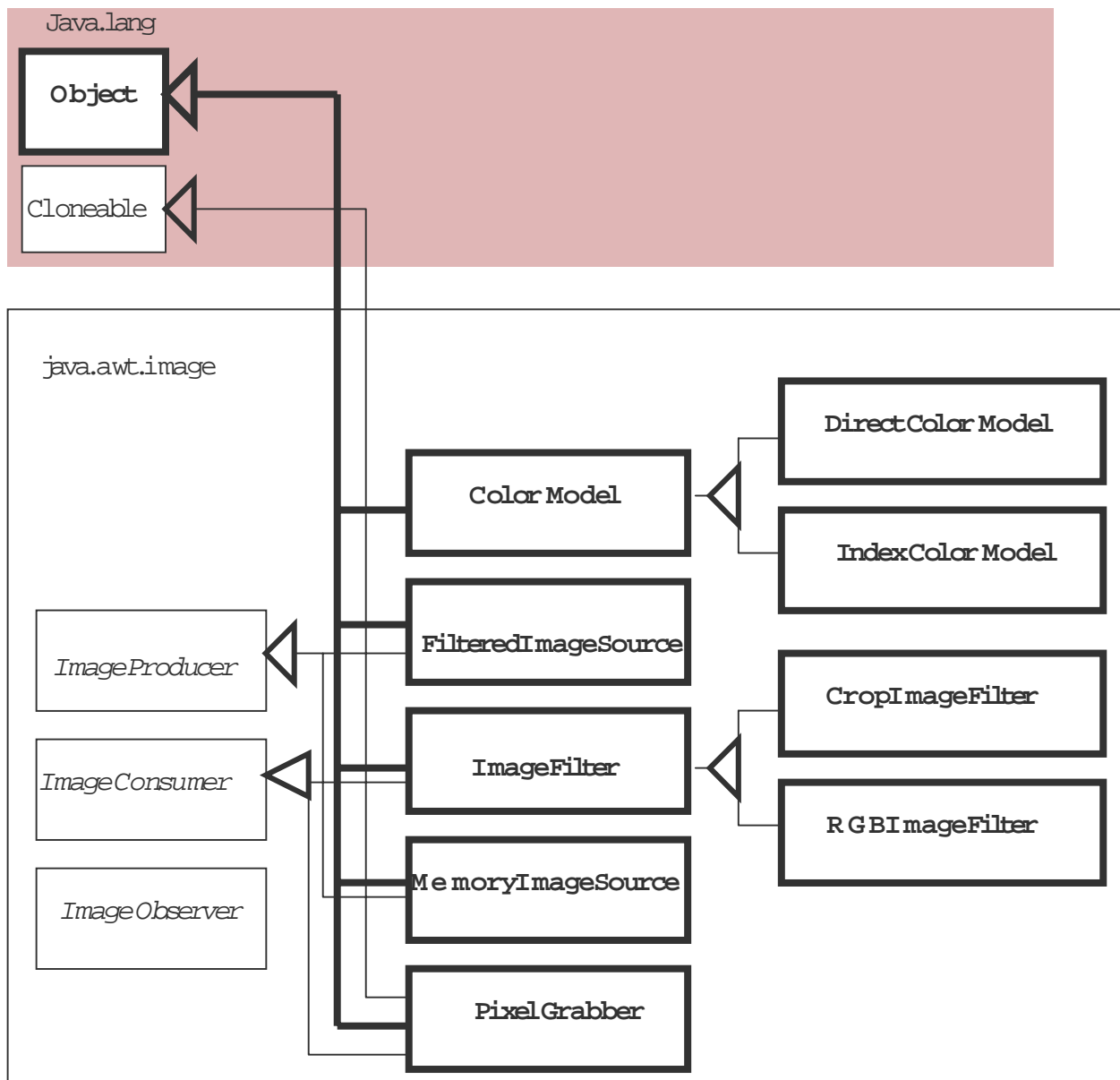
Java



Abstractions from imported package

Σχ. 9.1 Κατασκευές δομής java. Αφηρημένες τάξεις και μέθοδοι και διαδυνδέσεις γράφονται σε *italics*.

Java



Σχ. 9.2 Το `java.awt.image` πακέτο σχημάτων.

Java

- Ο ορισμός δικαιωμάτων πρόσβασης αφήνεται στο περιβάλλον. Υπάρχουν δύο επίπεδα πρόσβασης σε διασυνδέσεις και τάξεις.
 - Το προκαθορισμένο όπου τα δικαιώματα επιπλέον ελέγχονται στο επίπεδο χαρακτηριστικών της τάξεως (μεθόδων και πεδίων).
 - Το δημόσιο.
- Υπάρχουν τέσσερα επίπεδα στατικής πρόσβασης για χαρακτηριστικά τάξεως:
 - Ιδιωτικό, όπου τα χαρακτηριστικά καθορίζονται μόνο από την ορίζουσα τάξη.
 - Προκαθορισμένο, όπου τα χαρακτηριστικά καθορίζονται από τις τάξεις του ορίζοντος πακέτου.
 - Προστατευόμενο, όπου τα χαρακτηριστικά επιπρόσθετα καθορίζονται μέσα από την ορίζουσα τάξη και τις υποτάξεις της.
 - Δημόσιο όπου τα χαρακτηριστικά καθορίζονται από όλους όσους έχουν πρόσβαση στο πακέτο.

9.1.1 Διασυνδέσεις Έναντι Τάξεων

- Υπάρχει διαχωρισμός τάξεων και διασυνδέσεων σε τρόπο ώστε να επιτρέπεται διαδοχή μίας εφαρμογής σε συνδυασμό με πολλαπλές διαδοχές διασυνδέσεων. Έτσι υπάρχει συμβατότητα με διάφορες ανεξάρτητες διασυνδέσεις.
- Αν δύο διασυνδέσεις εισάξουν μεθόδους με το ίδιο όνομα και ενδείκτη, αλλά διαφορετικό τύπο επιστρεπτέας τιμής, τότε καμμιά τάξη δεν μπορεί να εφαρμόσει ταυτόχρονα και τις δύο διασυνδέσεις.
- Στο παρακάτω παράδειγμα το πακέτο `java.util` περιλαμβάνει τάξεις και διασυνδέσεις (`Observable` και `Observer` αντίστοιχα).

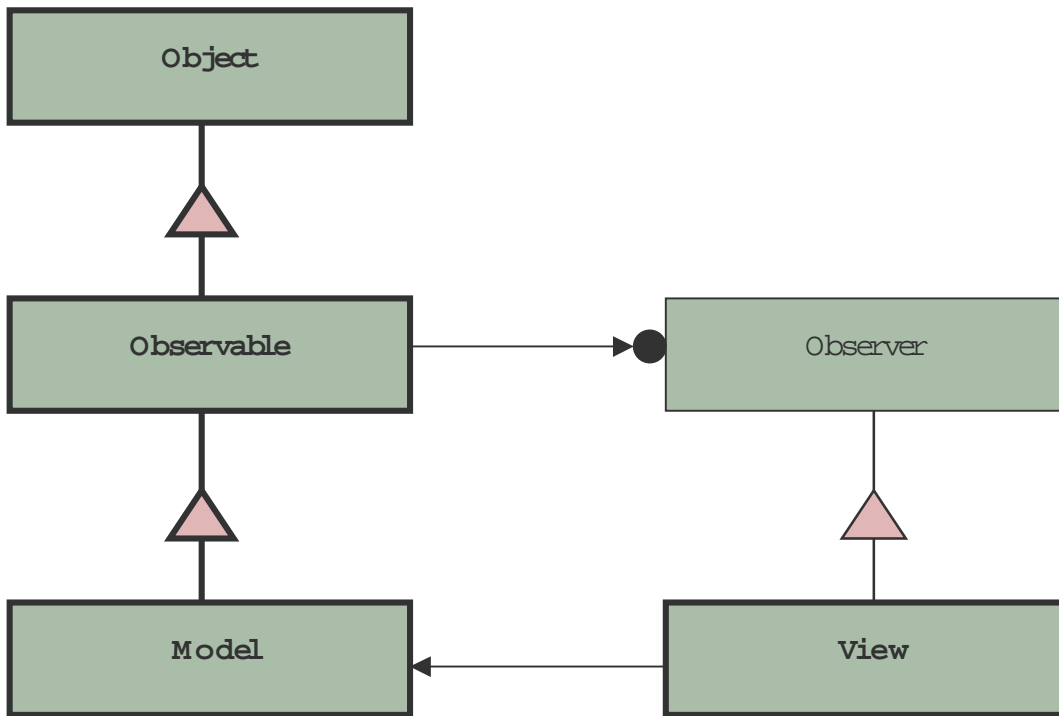
```
Package java.util;
```

```
public class Observable extends Object {
    public void addObserver(Observer o);
    public void deleteObservers(Observer o);
    public void countObservers();
    public void notifyObservers();
    public void notifyObservers(Object arg);
    protected void setChanged();
    protected void clearChanged();
    public boolean hasChanged();
}
```

```
public interface Observer {
    void update (Observable o, Object arg);
}
```

- Τα παρατηρούμενα (`Observable`) αντικείμενα είναι περιπτώσεις τάξεων διαδοχής της `Observable`. Η τάξη `Observable` διατηρεί κατάλογο παρατηρητών και δείκτην που σημειώνει εκσυγχρονισμό του παρατηρούμενου αντικειμένου. Οι παρατηρητές πρέπει να εφαρμόσουν την διασύνδεση `Observer` και ειδοποιούνται από το παρατηρούμενο αντικείμενο μέσω της κλήσεως `notifyObservers`. Στο σχήμα 9.3 εφαρμόζεται η κατασκευή ενός απλού πρότυπου view.
- Όταν η τάξη `Observable` εξαρτάται από υπερτάξη διαφορετική από το `Object` και όταν οι περιπτώσεις αυτής της τάξης πρέπει να παρατηρηθούν, η ιεραρχία τάξεων πρέπει να ξανακατασκευασθεί.

Διασυνδέσεις Έναντι Τάξεων



Σχ. 9.3 Κατασκευή πρότυπου view με Observable και Observer.

Διασυνδέσεις Έναντι Τάξεων

- Αν υποθέσουμε ότι το παρατηρούμενο αντικείμενο διαδέχεται κάποια άλλη τάξη τότε η τάξη `Observable` δεν θα μπορεί να χρησιμοποιηθεί άμεσα. Τότε με τη εφαρμογή μίας υποτάξης του `Observable` αφαιρείται ο προστατευτικός τύπος πρόσβασης από τις μεθόδους `setChange` και `clearChange`. Οι περιπτώσεις `trulyObservable` μπορούν να χρησιμοποιηθούν σαν συνιστώσα αντικείμενα ενός παρατηρούμενου αντικείμενου. Επειδή ο παρατηρητής παίρνει αναφορά στη τάξη `Observable`, η τάξη `ObserverRegistry` δίδει τη μέθοδο `getTrueObservable` για να πάρει το αντικείμενο που πραγματικά έχει αλλάξει.
- Το `fix.it` είναι υπόταξη του `Observable`:

```
package fix.it;
public class ObserverRegistry extends java.util.Observable {
    private Object owner; //reference to the observable object
    public ObserverRegistry(Object trueObservable) {
        //constructor
        owner=trueObservable;
    }
    public Object gettrueObservable() {
        return owner;
    }
    public void setChanged() {
        super.setChanged()
    }
    public void clearChanged() {
        super.setChanged()
    }
}
```

Διασυνδέσεις Έναντι Τάξεων

- Ο κατασκευαστής του καταχωριτή ρυθμίζει αυτή την αναφορά:

```
Package fixed.it;
Import fix.it.*;
Public class StringBufferModel extends stringBuffer {
    private ObserverRegistry obReg=new ObserverRegistry(this);
    public ObserverRegistry getObserverRegistry() {
        return obReg;
    }
    public void setLength(int newLength)
        throws IndexOutOfBoundsException
    {
        super.setLength(int newLength);
        obReg.setChanged();
        obReg.notifyObservers(); //could tell them what has
        changed...
    }
    // Other string methods
}
```

- Η τάξη `StringBufferModel` εκτείνει την καθορισμένη τάξη `java.lang.StringBuffer` που υποστηρίζει την σύνθεση γραμματοσειρών. Καθώς ο μεταφορέας της γραμματοσειράς είναι ένα μεταβλητό αντικείμενο, είναι κατάλληλο για υλοποίηση του προτύπου `view`.
- Η τάξη `StringBufferModel` δεν μπορεί να εκτείνει την `Observable` απ' ευθείας, καθώς ήδη εκτείνει την `StringBuffer`, που δεν είναι σχεδιασμένη να λειτουργεί σαν υποτάξη σε κάτι παρατηρούμενο, και έτσι εκτείνει το `Object` αντί το `Observable`. Γι' αυτό η `StringBufferModel` χρησιμοποιεί το αντικείμενο `ObserverRegistry`. Παρατίθεται το παράδειγμα ενός παρατηρητή- αντικειμένου, να προβάλλει το περιεχόμενο ενός `StringBufferModel`.

Διασυνδέσεις Έναντι Τάξεων

```
package fixed.it;
import java.util.*;
public class StringBufferView implements Observer {
    private StringBufferModel buf;
    public StringBufferView (StringBufferModel b) {
        buf= b;
        b.getObserverRegistry().addObserve(this);
    }
    public void update (Observable o, Object arg) {
        ObserverRegistry reg=(Observer Registry)o; //
        checked cast
        Object observable= reg.getTrueObservable();
        if (observable!=buf) throw new
        IllegalArgumentException();
        // ...get changes from buf and update display
    }
}
```

- Η προηγούμενη σειρά παραδειγμάτων δείχνει τάξεις να εφαρμόζουν μία μόνο διασύνδεση. Όταν συμπίπτουν ονόματα, η Java απορρίπτει την τάξη που προσπαθεί να εφαρμόσει συγκρουόμενες διασυνδέσεις. Στο ερώτημα πώς μπορεί ο χρήστης μίας διασύνδεσης ή τάξης να ξέρει αν το αντικείμενο που χρησιμοποιεί υποστηρίζει άλλες τάξεις και διασυνδέσεις.

- Η Java παρέχει ένα ελεγκτή τύπου, τον `instanceof`, που μπορεί να χρησιμοποιηθεί για έρευνα κατά την εκτέλεση:

```
Interface Blue {...}
```

```
Interface Green {...}
```

```
Interface FunThing implements Blue, Green {...}
```

Κάθε χρήστης που έχει αναφορά στο `Blue` μπορεί να ελέγξει κατά πόσο το αντικείμενο υποστηρίζει και τη διασύνδεση `Green`.

```
Blue thing= new FunThing();
```

```
Green part= null;
```

```
If (thing instanceof Green) //typetest
```

```
Part= (Green) thing; //checked type
```

```
//cast, from Blue to Green
```

9.1.2 Εξαιρέσεις και χειρισμός τους

- Οι εξαιρέσεις και τα εκτελεστικά λάθη στη Java αντανακλώνται σαν αντικείμενα εξαιρέσεων ή λαθών. Αυτά προέρχονται είτε από τάξεις, είτε από το σύστημα όπως η εκτός ορίων κατάταξη σε ένα πίνακα. Γενικά αυτά τα αντικείμενα προέρχονται από τη τάξη `Throwable` και μπορούν να αποκλεισθούν με την συνάρτηση `catch` της οδηγίας `try`. Έτσι η μέθοδος `notifyObservers` της τάξης `Observable`, για παράδειγμα, μπορεί να προστατευθεί από εξαιρέσεις των ειδοποιημένων παρατηρητών:

```
public void notifyObservers() {
    for Observer
    ob=first(),ob!=null,ob=next()){
        try {
            ob.update(this,null);
            //observer may throw an exception
        }
        catch(Exception ex) {
            // ignore exception and continue
            //with next observer
        }
    }
}
```

- Στο πιο πάνω παράδειγμα είναι φανερό πώς η μέθοδος `update`, μπορεί να παράγει εξαιρέσεις. Αυτοί οι τύποι εξαιρέσεων καλούνται *μη ελεγμένες εξαιρέσεις*. Επίσης υπάρχουν τύποι εξαιρέσεων που μπορούν να παραχθούν από τις εφαρμογές μίας μεθόδου αν η δήλωση της το φανερώνει (*ελεγμένες εξαιρέσεις*). Παράδειγμα:

```
class AttemptedFailed extends Exception{}
class Probe {
    public void goForIt (int x) throws
    AttemptedDailed{
        if (x!=42) throw new
            AttemptedFailed();
    }
}
class Agent {
    public void trustMe (Probe p) {
        p.goForIt(6*7); //compile time error!
    }
}
```

Εξαιρέσεις και χειρισμός τους

- Η τάξης `Agent` δεν μπορεί να μεταγλωττισθεί. Η μέθοδος `TrustMe` ούτε δηλώνει ότι μπορεί να παράγει την εξαίρεση `AttemptFailed`, ούτε ότι παίρνει τέτοιες εξαιρέσεις. Στο παράδειγμα η τάξη `Agent` τυγχάνει να ξέρει τι χρειάζεται ο τύπος `Probe` για να αποφύγει την αποτυχία, ούτως ώστε ποτέ δεν δίδει λάθος, όταν κληθεί από το `TrustMe`. Σε άλλη περίπτωση όμως μπορεί να το κάνει.

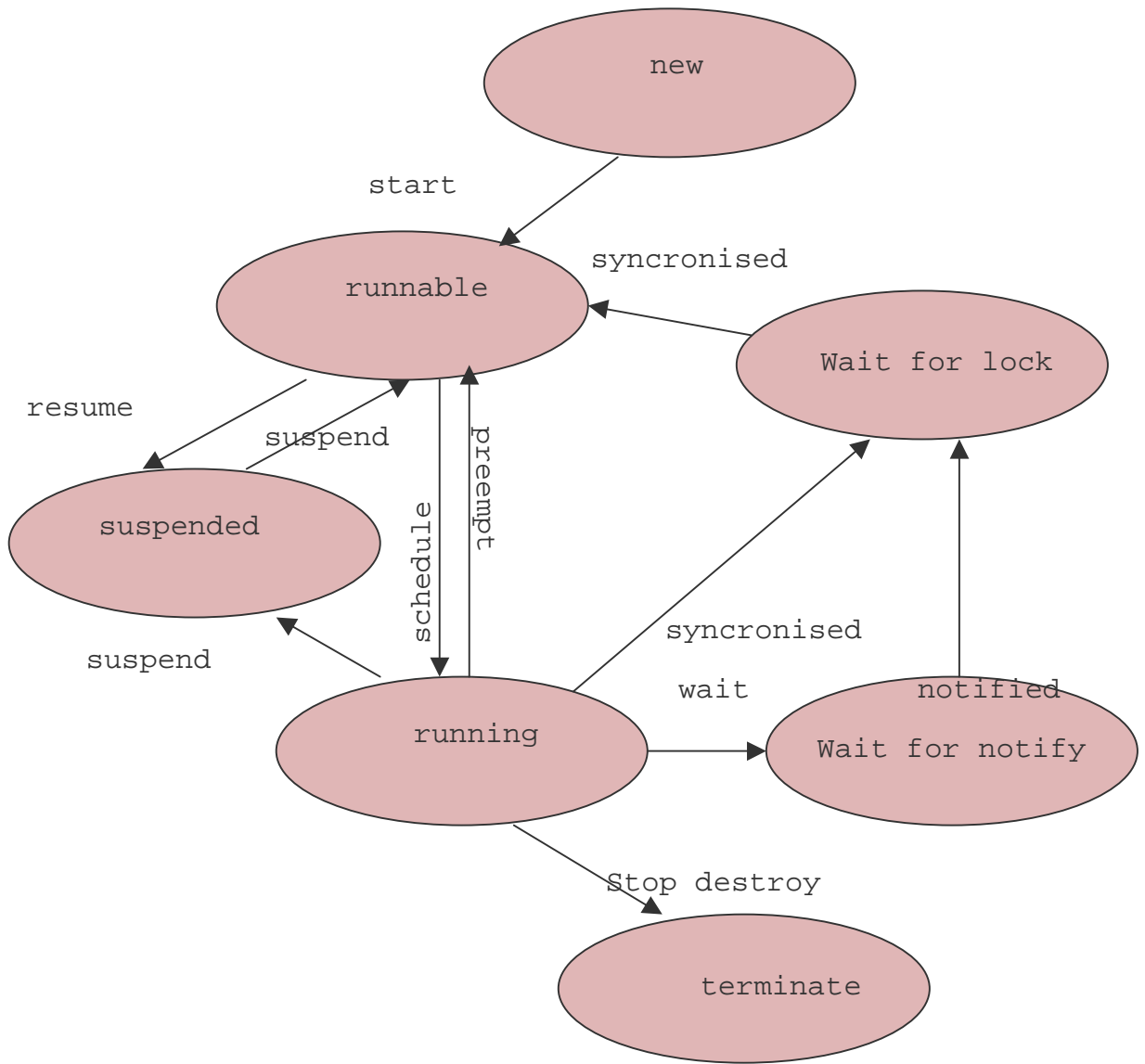
9.1.3 Σύγχρονες Διεργασίες

- Με τον όρο *νήμα* (thread) ορίζουμε τις μονάδες συγχρονισμού σε ένα σύστημα ταυτόχρονων διεργασιών, με κοινή διεύθυνση μνήμης.
 - Τα νήματα μπορούν να επικοινωνούν έμμεσα ή με κοινό συγχρονισμό.
 - Στη σειρά των διαδικασιών του ένα νήμα μεταφέρεται από αντικείμενο σε αντικείμενο καθώς εκτελεί κλήσεις μεθόδων.
 - Τα νήματα μπορούν να δημιουργηθούν δυναμικά, να παίρνουν προτεραιότητα, να αναστέλλονται, να ξαναξεκινούν και να τερματίζονται.
 - Ανήκουν σε σύνολα από την δημιουργία τους.
 - Υπάρχουν δέκα επίπεδα προτεραιότητας σε τύπους χρήστη ή δαίμονα.
 - Κάθε σύνολο θέτει όρια στην προτεραιότητα των νημάτων του.
 - Ο τερματισμός δεν είναι διαδοχικός από γονείς σε παιδιά.
 - Ολόκληρη η εφαρμογή Java τερματίζεται όταν και το τελευταίο νήμα χρήστη τερματισθεί, ανεξαρτήτως νημάτων - δαιμόνων.

Στο σχήμα 9.4 φαινονται οι καταστάσεις νημάτων και οι μεταβολές τους.

- Στη Java τα νήματα έχουν κοινό πεδίο μνήμης και καθώς αυτές παίρνουν τον έλεγχο πρέπει να ρυθμίζεται η ταυτόχρονη πρόσβαση. Τα χαρακτηριστικά της γλώσσας απαιτούν σειριακή πρόσβαση. Για συνεπή πρόσβαση σε μεγαλύτερες μονάδες, όπως πολλαπλά πεδία, χρειάζεται άμεσος συγχρονισμός.
- Η Java υποστηρίζει συγχρονισμό είτε σε είσοδο *συγχρονισμένης μεθόδου*, είτε σε είσοδο *συγχρονισμένης κατάστασης*. Σ' αυτά κάθε νήμα αποκτά κλειδαριά σε ένα αντικείμενο πριν προχωρήσει. Υπάρχει μία τέτοια ασφάλεια για κάθε αντικείμενο.

Σύγχρονες Διεργασίες



Σχ. 9.4 Καταστάσεις και μεταβολές νημάτων.

Σύγχρονες Διεργασίες

```
class SynchStack {
    java.util.Stack stack=new
    java.util.Stack;
    public synchronized void push(Object
    item) {
        stack.push(item)
    }
    public synchronized Object pop() {
        return(stack.pop())
    }
    public synchronized boolean empty() {
        return(stack.empty())
    }
}
```

Το Synchronized παίρνει το αντικείμενο που θα κλειδωθεί σαν παράμετρος:

...//το νήμα μπορεί ήδη να έχει κλειδί στο αντικείμενο.

```
synchronized(obj){
```

...//το νήμα έχει κλειδί για το αντικείμενο

```
} //το κλειδί αφήνεται, εκτός αν το κρατά //κάποιο τρέχων νήμα.
```

- Χωρίς επιπλέον μέτρα, ένα νήμα που περιμένει για ένα συμβάν θα πρέπει να συγκεντρώσει μεταβλητές. Για αποφυγή ανεπάρκειας, η Java επιτρέπει σε νήματα διεργασιών να περιμένουν σε οποιοδήποτε αντικείμενο. Έτσι η τάξη Object ορίζει τις μεθόδους wait, notify και notifyall, που μπορούν να κληθούν σε πλαίσιο που κατέχει κλειδαριά γ'αυτό το αντικείμενο. Καθώς περιμένουν η ασφάλεια απελευθερώνεται και όταν ειδοποιηθεί ένα νήμα ξαναπαίρνει την ασφάλεια πριν να προχωρήσει. Αν το αντικείμενο είναι κλειδωμένο, τότε το ειδοποιημένο νήμα περιμένει. Μπορεί να υπάρξει δηλαδή ανταγωνισμός για την απόκτηση της ασφάλειας σε οποιαδήποτε σειρά, μόλις ο προηγούμενος κάτοχος την απελευθερώσει.

Σύγχρονες Διεργασίες

- Ένα νήμα συνδέεται με ένα κατέχον αντικείμενο τη στιγμή της δημιουργίας του και μέσω αυτού του αντικειμένου μπορεί
 - Να αναστέλλεται, να ξαναξεκινά, ή να τερματίζεται.
 - Να αλλάζει τον τύπο του νήματος μεταξύ χρήστην και δαίμονα.
- Τα αντικείμενα που κατέχουν ένα νήμα είναι περιπτώσεις της τάξης `java.lang.Thread`:

```
public class Thread implements Runnable {
    public Thread(); //default constructor; use //method run
    public Thread(Runnable runObject); //use //method run of
        runObject
    public void run();
    public void start() throws InterruptedException;
    public void stop() throws Security Exception;
    public void suspend() throws Security Exception;
    public void resume() throws Security Exception;
    public void setPriority(int new Priority) throws Security
        Exception, InterruptedException;
    public void setDaemon(boolean on) throws Security
        Exception, InterruptedException;
    public void destroy() throws Security Exception;
    // other methods deleted ..
}
```

- Η μέθοδος `run` της τάξης `Thread` ορίζει την εξωτερική μέθοδο που θα εκτελεσθεί από το νέο νήμα. Διαφορετικά το νήμα ορίζεται με αναφορά σε αντικείμενο που εφαρμόζει την διασύνδεση `Runnable`, όποτε το νήμα αρχίζει εκτελώντας το `run` αυτού του αντικειμένου.
- Μια μέθοδος τερματίζει όταν τερματίζει η εξωτερική μέθοδος και επίσης όταν το `stop` ή το `destroy` του κατέχοντος αντικειμένου καλείται.

9.2 JavaBeans

- Έκδοση τον Οκτώβριο του 1996.
- Τα συστατικά της Java καλούνται 'beans'.
- Δεν υπάρχει καθαρή διαφορά μεταξύ τάξης και αντικειμένου όπως στην Java.
- Οι προγραμματισμένες και συνδεδεμένες περιπτώσεις των συστατικών, επίσης καλούνται 'beans', γι' αυτό είναι καλύτερα να μιλούμε για *bean* και *περίπτωση bean*.
- Έχει σχεδιασθεί για την ολοκλήρωση ενός συστατικού σε περιέχον περιβάλλον έξω από την Java.
- Οι περιπτώσεις των συστατικών πρώτα συναρμολογούνται από τον κατασκευαστή εφαρμογών και μετά εκτελούνται.
- Η περίπτωση ενός *bean* μπορεί να εξετάσει αν υπάρχει απ' ευθείας γραφική διασύνδεση.
- Τα γενικά χαρακτηριστικά του προτύπου είναι:
 - Συμβάντα. Κάθε *bean* είναι πηγή και υποδοχή ειδικών συμβάντων.
 - Ιδιότητες. Αυτές προέρχονται από ζεύγη των μεθόδων *getter* και *setter* για κάθε περίπτωση *bean* ξεχωριστά. Μπορούν να χρησιμοποιηθούν για τυποποίηση ή προγραμματιστικά. Οι αλλαγές τους μπορεί να προκαλέσουν συμβάντα. Στις αλλαγές μπορούν να τεθούν περιορισμοί.
 - Εσωτερική διάγνωση. Αυτό γίνεται από εργαλείο συναρμολόγησης για ιδιότητες, συμβάντα και μεθόδους που υποστηρίζει ένα *bean*.
 - Τυποποίηση. Ρύθμιση ιδιοτήτων από εργαλείο συναρμολόγησης.
 - Επιμονή. Φύλαξη περιπτώσεων για επαναφόρτιση.

9.2.1 Συμβάντα και Συνδέσεις

- Η μετάδοση συμβάντων μπορεί να είναι πολλαπλή σε πολλούς αποδέκτες ενός συγκεκριμένου συμβάντος ή σε ένα αποδέκτη μόνο.
- Τα συμβάντα είναι αμετάβλητα και αδιαφανή. Αν χρειάζεται μεταβολή ενός συμβάντος - αντικειμένου κατά την μετάδοση του, αυτή εμπερικλείεται στις μεθόδους του.
- Οι ακροατές συμβάντων πρέπει να εφαρμόσουν μία διασύνδεση που να εκτείνει την διασύνδεση `java.beans.EventListener` που έχει μία μέθοδο υποδοχής για κάθε συμβάν:

```
Interface UserSleepsListener extends java.beans.EventListener
{
    void userSleeps (UserSleepsEvent e);
}
```

- Η εφαρμογή της διασύνδεσης ακρόασης από ένα ακροατή μπορεί να γίνει μόνο μία φορά. Αν εγγραφεί σε πολλές πηγές συμβάντων πρέπει να αποφασίζει για την προέλευση του καθενός. Αυτό απλουστεύεται με την διαδικασία *προσαρμοστή συμβάντων* (*event adapter*) που μπορεί να είναι και ακροατής και αποστολέας συμβάντων. Επίσης μπορεί να υπάρχουν πολλοί προσαρμοστές με αναφορά για το κάθε συμβάν ξεχωριστά. Τότε κάθε προσαρμοστής καλεί διαφορετική μέθοδο του ακροατή.
- Σημειώστε ότι στη απουσία τύπων και μεταβλητών μεθόδων οι προσαρμοστές συμβάντων οδηγούν σε πληθώρα τάξεων, ή σε αργές και αφύσικες λύσεις που βασίζονται σε αντακλαστικές υπηρεσίες. Στη Java 1.1 αυτό έχει περιορισθεί με την χρησιμοποίηση *ελαφρών προσαρμοστικών εσωτερικών τάξεων* (*light weight adapter classes*).
- Κάθε πηγή συμβάντων παρέχει ζεύγη μεθόδων καταχώρισης και αποβολής ακροατών:

```
public void addUserSleepsListener (UserSleepsListener I);
public void removeUserSleepListener (UserSleepsListener I);
```

Για μετάδοση σε ένα ακροατή μπορεί να παράξει το `TooManyListenersException`:

```
public void addUserSleepsListener (UserSleepsListener I);
throws java.util.TooManyListenersException;
```

Συμβάντα και Συνδέσεις

- Στη μετάδοση συμβάντων με πολλαπλούς αποδέκτες υπάρχει η περίπτωση αυτοί να αλλάξουν, ενώ προωθείται το συμβάν. Είναι ασαφές πώς η JavaBeans το αντιμετωπίζει αυτό, αλλά μπορεί να γίνει αντίγραφο των παραληπτών πριν την αποστολή του συμβάντος.
- Επίσης εξαιρέσεις μπορούν να επέλθουν καθώς ένα συμβάν προωθείται. Και πάλι είναι ασαφές αν θα πρέπει η μετάδοση να συνεχίσει με τους υπόλοιπους ακροατές.
- Τέλος υπάρχει το πρόβλημα της αιτιώδους σειράς μετάδοσης των συμβάντων. Η JavaBeans δεν καθορίζει περιορισμούς στη σειρά παράδοσης συμβάντων.
- Μια πηγή συμβάντων μπορεί να κατέχει ασφάλειες που απαιτούνται από ακροατές για να προωθήσουν άλλα συμβάντα. Το αποτέλεσμα θα είναι αδιέξοδο.
- Συστήνεται όπως οι πηγές συμβάντων να μην κρατούν τις δικές τους εσωτερικές ασφάλειες όταν καλούν μεθόδους ακροατών συμβάντων. Δεν είναι εύκολο να αποφασισθεί αν και ποιές ασφάλειες η πηγή ήδη κατέχει.
- Επιπλέον θα πρέπει να αποφεύγεται να χρησιμοποιείται συγχρονισμένη μέθοδος στην πυροδότηση ενός συμβάντος, αλλά αντί αυτού με μία συγχρονισμένη ενότητα να εντοπίζονται οι στόχοι-ακροατές. Μετά να καλούνται οι ακροατές συμβάντων από μη συγχρονισμένο κώδικα. Και πάλι με αυτό δεν είναι εύκολο να αποφευχθούν πολύπλοκα αδιέξοδα σαν αποτέλεσμα συνδιασμού πολλαπλών νημάτων και επικοινωνίας συμβάντων .

9.2.2 Ιδιότητες

- Ένα `bean` μπορεί να ορίσει αριθμό ιδιοτήτων σε αθαίρετους τύπους. Οι ιδιότητες σαν παράμετροι επηρεάζουν την συμπεριφορά των `beans`.
- Η πρόσβαση σε αυτές γίνεται:
 - με την κλήση των μεθόδων `getter` και `setter`,
 - με οθόνες ιδιοτήτων κατά την συναρμολόγηση, που τυποποιούν τις περιπτώσεις του `bean`
 - κατά την εκτέλεση, που μπορεί να είναι μέρος αλληλεπίδρασης.
- Η χρήση των μεθόδων πρόσβασης για την ιδιότητα `'background color:'`
- Επίσης για βελτισποίηση της συντήρησης πινάκων ιδιοτήτων υποστηρίζεται *ευρετήριο ιδιοτήτων* (`indexed properties`). από το οποίο παίρνουν τιμές οι μέθοδοι.

```
Public java.awt.Color getSpectrum(int index);
Public java.awt.Color[] getSpectrum();
Public void setSpectrum (int index, java.awt.Color color);
Public void setSpectrum (java.awt.Color[] colors);
```

- Μια ιδιότητα μπορεί να περιορισθεί ως προς την αλλαγή της η οποία μπορεί να πυροδοτεί ένα συμβάν. Ο τύπος του συμβάντος αντικειμένου είναι `java.beans.PropertyChangeEvent` που έχει την δομή:

```
Public class PropertyChangeEvent extends
    java.util.EventObject {
    public Object getNewValue();
    public Object getOldValue();
    public String getPropertyName();
}
```

Ιδιότητες

- Οι μέθοδοι για καταχώριση και αποβολή των ακροατών (συμβάντων) για αλλαγές ιδιοτήτων είναι:

```
public void addPropertyChangeListener
    (PropertyChangeListener x);
public void removePropertyChangeListener
    (PropertyChangeListener x);
```

- Η διασύνδεση `java.beans.PropertyChangeListener` εισάγει την μέθοδο για αλλαγές ιδιοτήτων:

```
void propertyChange (PropertyChangeEvent evt);
```

- Η αλλαγή μίας ιδιότητας μπορεί να απαγορευθεί (από άλλους ακροατές), όταν η μέθοδος `setter` ορίζεται να στέλνει `PropertyVetoExceptions`. Όταν πρόκειται να γίνει μία αλλαγή το αντικείμενο `PropertyChangeEvent` περνά σε όλους τους ακροατές που έχουν δικαίωμα απαγόρευσης. Αυτοί μπορούν να στείλουν το `java.beans.PropertyVetoExceptions`. Έστω και μία απαγόρευση επαναφέρει την παλιά τιμή της ιδιότητας και οι ακροατές ενημερώνονται για την επαναφορά με ένα δεύτερο `PropertyChangeEvent`.
- Οι μέθοδοι για καταχώριση και απόρριψη αλλαγών που μπορούν να απαγορευθούν είναι:

```
Public void addVetoableChangeListener
    (VetoableChangeListener x);
Public void removeVetoableChangeListener
    (VetoableChangeListener x);
```

- Η διασύνδεση `java.beans.VetoableChangeListener` εισάγει την μέθοδο για κλήση σε αλλαγές ιδιοτήτων:

```
Public void vetoableChange (PropertyChangeEvent evt) throws
    PropertyVetoException;
```

Ιδιότητες

- Τυπικά οι ιδιότητες εκδίδονται από εκδότες ιδιοτήτων όπως αποφασίζεται από το αντικείμενο `java.beans.PropertyEditorManager`. Αυτό το αντικείμενο κρατά καταχωρίσεις συσχέτισης μεταξύ τύπων Java και τάξεων εκδοτών ιδιοτήτων. Ένα `bean` όμως μπορεί να παραβιάσει αυτή την επιλογή με το να συγκεκριμενοποιήσει μία τάξη εκδότη ιδιοτήτων, που θα χρησιμοποιηθεί για έκδοση μίας συγκεκριμένης ιδιότητας αυτού του `bean`. Επιπλέον, αν η τυποποίηση ενός `bean` είναι περίπλοκη ή περιλαμβάνει πολλές ιδιότητες, μπορεί επίσης να ορίσει μία τάξη τυποποιητή που εφαρμόζει μία συγκεκριμένη τυποποίηση διασύνδεσης. Ένας τυποποιητής παίρνει ξεχωριστό παράθυρο διαλόγου, ενώ οι τυπικοί εκδότες ιδιοτήτων μπορούν να χρησιμοποιηθούν στο διάλογο.

9.2.3 Ενδοσκόπηση

- Τα συμβάντα και οι ιδιότητες υποστηρίζονται από ένα συνδυασμό νέων καθορισμένων διασυνδέσεων και τάξεων όπως `EventListener`, `EventObject`, `EventSetDescriptor`, και `PropertyDescriptor`.
 - `EventListener`. Επιτρέπει την συναρμολόγηση διά μέσου αντακλαστικών υπηρεσιών, για εξασφάλιση υποστήριξης ορισμένων χαρακτηριστικών ενός δεδομένου `bean`. Επιπρόσθετα παρουσιάζει νέες ιδέες για σχηματισμό μεθόδων. Το *σχήμα μεθόδου* (*method pattern*) είναι ένας συνδυασμός κανόνων για σχηματισμό του ενδείκτη της μεθόδου, του τύπου επιστροφής και του ονόματος της. Επίσης επιτρέπει την ελαφριά ταξινόμηση μεθόδων διασύνδεσης ή τάξεων. Παράδειγμα από σχήμα μεθόδου για ένδειξη ενός ζεύγους μεθόδων `getter` και `setter` για συγκεκριμένη ιδιότητα:

```
public <PropertyType> get<PropertyName> ();
public void set<PropertyName> (<PropertyType> a);
```

Έτσι για την ιδιότητα `'Background'` του τύπου `java.awt.Color`, έχουμε τις μεθόδους:

```
public java.awt.Color getBackground();
public void setBackground (java.awt.Color color);
```

- Το ακόλουθο απόσπασμα της τάξης `BeanInfo` και σχετικών τάξεων, δείχνει πως οι πληροφορίες της `BeanInfo` μπορούν να χρησιμοποιηθούν για εύρεση υποστηριζόμενων συμβάντων, ιδιοτήτων και μεθόδων:

```
package java.beans;
public interface BeanInfo {
    public abstract BeanInfo[] getAdditionalBeanInfo();
    // current info takes precedence over // additional bean
    info
    public abstract BeanDescriptor getBeanDescriptor ();
    public abstract EventSetDescriptor[] getEventSetDescriptors
    ();
    public abstract MethodDescriptor[] getMethodDescriptors ();
    public abstract PropertyDescriptor[] getPropertyDescriptors
    ();
}
```

Ενδοσκόπηση

```

public class FeatureDescriptor {
    public Enumeration attributeNames ();
    public Object getValue (String attributeName);
    public void setValue (String attributeName, Object value);
        //setValue and getValue allow to //associate arbitrary
        named attributes //with a feature.
    public boolean is Expert (); // feature //for experts users
        only
    public boolean is Hidden (); // feature //for tool use only
}

public class BeanDescriptor extends FeatureDescriptor {
    public Class getBeanClass (); //the //class representing
        the entire bean.
    public Class getCustomizerClass (); //null, if the bean
        has no customizer
}

public class EventSetDescriptor extends FeatureDescriptor {
    public java.lang.reflect.Method getAddListenerMethod ();
    public java.lang.reflect.Method getRemoveListenerMethod
        ();
    public java.lang.reflect.Method[] getListenerMethods ();
    public Class getListenerType ();
    public boolean isUnicast (); //this is //a unicast event
        source: at most one //listener.
}

public class MethodDescriptor extends FeatureDescriptor {
    public java.lang.reflect.Method getMethod();
}

public class PropertyDescriptor extends FeatureDescriptor {
    public Class getPropertyEditorClass ();
    public Class getPropertyType ();
    public java.lang.reflect.Method getReadMethod ();
    public java.lang.reflect.Method getWriteMethod ();
    public boolean isBound (); //change of //bound property
        fires PropertyChange //event
    public boolean isConstrained (); //attempted change may be
        vetoed
}

```

Ενδοσκόπηση

- Τα σχήματα μεθόδου αποτελούν παρέκκλιση από τις καθιερωμένες αρχές σχεδιασμού της Java. Συνήθως μία τάξη της Java σηματοδοτεί ιδιότητες και λειτουργίες μέσω συγκεκριμένων διασυνδέσεων.
- Υποθετική αντικατάσταση σχήματος μεθόδου για `getters` και `setters` ιδιοτήτων: Η `Property` είναι καθιερωμένη άδεια διασύνδεση, για κάθε ιδιότητα ορίζεται μία υποδιασύνδεση. Κάθε τάξη `μ` που έχει κάποιες απ' αυτές τις ιδιότητες, πρέπει να εφαρμόσει τις αντίστοιχες διασυνδέσεις ιδιοτήτων.

```
interface BackgroundProp extends Property {
//hypothetical!
public Color getBackground ();
public void setBackground (Color color);
}
```

Τα ονόματα των μεθόδων `set` και `get` θα πρέπει να περιέχουν τα ονόματα των ιδιοτήτων, για αποφυγή συγκρούσεων με άλλες διασυνδέσεις ιδιοτήτων. Η επέκταση της διασύνδεσης `Property` μπορεί τώρα να βρει τις διασυνδέσεις ιδιοτήτων σε μία τάξη. Αυτό γίνεται αντί της διαπίστωσης σχημάτων μεθόδων.

9.2.4 Αρχεία Java (JAR)- Συσκευασία Συστατικών (beans)

- Τα αρχεία τάξεων της Java ήταν τα μόνα μέσα για συσκευασία συστατικών πριν την έκδοση 1.1 JDK. Σ' αυτό το είδος αρχείων περιέχεται μόνο μία μεταγλωττισμένη μονάδα (δημόσια τάξη ή διασύνδεση με αριθμό ιδιωτικών τάξεων). Επίσης περιέχει μετα-πληροφορίες για το μεταγλωττισμένο περιεχόμενο.
- Με την χρησιμοποίηση των Java applets:
 - Απαιτούνταν πολλαπλές προσβάσεις για εξασφάλιση περαιτέρω *αρχείων τάξεων* και *πόρων* (class files και resource files).
 - Για την αποστολή συστατικών χρειαζόταν η κατανομή πολλών ξεχωριστών αρχείων.
- Το πρόβλημα λήθηκε με την συσκευασία ενός bean σε αρχείο JAR (Java Archive). Αυτά καταγράφονται σε σχήμα ZIP και περιλαμβάνουν αρχεία πληροφοριών με καταχωρήσεις για
 - Σύνολο αρχείων τάξεων.
 - Σύνολο ακολουθείας αντικειμένων που συχνά χρησιμοποιούνται σε περιπτώσεις πρωτοτύπων bean. Έτσι η αποστολή τους γίνεται σε υποχρεωτική αρχική μορφή. Με αποσειριοποίηση των πρωτοτύπων παράγονται νέες περιπτώσεις bean.
 - Προαιρετικά αρχεία βοήθειας σε HTML.
 - Προαιρετικός εντοπισμός πληροφοριών για αυτοεντοπισμό του bean.
 - Προαιρετικά εικονίδια σε σχήμα GIF.
 - Αρχεία πόρων.

Μπορούν να υπάρχουν πολλά beans σε ένα αρχείο JAR.

9.2.5 Προσθήκη Abstract Windowing Toolkit (AWT)

- Παροχή διασύνδεσης γραφικών.
- Προσθήκες μετά την έκδοση 1.1 JDK:
 - Πρότυπο συμβάντων με αντιπροσώπους. Η σύνθεση και σύνδεση των αντικειμένων χρησιμοποιείται για εφαρμογή διαδοχής.
 - Μεταφορά δεδομένων. Τύποι διαδικτύου Multipurpose Internet Mail Extensions (MIME) για επαφή με μη-Java εφαρμογές. για μεταφορές μεταξύ εφαρμογών της Java χρησιμοποιούνται απ' ευθείας τάξεις Java.
 - Υποδομές ελαφρών διασυνδέσεων. Υπάρχουν δηλ. Συστατικά ανεξάρτητα της τοπικής εφαρμογής.
 - Εκτύπωση. Απ' ευθείας υποστήριξη εκτύπωσης. Δυνατότητα για εκτύπωση γραφικών συστατικών με μεθόδους αναπαράστασης οθόνης. Αδυνατεί όμως στην εκτύπωση ενσωματωμένων αντικειμένων.

9.3 Άλλες Υπηρεσίες Java

- Java Development Kits 1.1 περιλαμβάνει:
 - Σειριοποίηση αντικειμένων.
 - Απόμακρη κλήση μεθόδων RMI (Remote Method Invocation) και αντανάκλαση.
 - Java Native Interface JNI
 - Java Database Connectivity JDBC. Πρόσβαση SQL σε βάσεις δεδομένων, applets με ενδείκτες, υποστήριξη για τοπικά και διεθνή δίκτυα, και νέα πακέτα μαθηματικών.
 - Δέσιμο με την OMG IDL.

9.3.1 Αντανάκλαση

- Η υπηρεσία αντανάκλασης της Java ολοκληρώνει τα χαρακτηριστικά της γλώσσας επιτρέποντας:
 - Επιθεώρηση τάξεων και διασυνδέσεων για τα πεδία και τις μεθόδους τους.
 - Κατασκευή νέων περιπτώσεων τάξεων και νέων πινάκων.
 - Πρόσβαση και τροποποίηση πεδίων αντικειμένων και τάξεων.
 - Πρόσβαση και τροποποίηση στοιχείων των πινάκων.
 - Κλήση μεθόδων σε αντικείμενα και τάξεις.
- για να καθιστούν ικανές οι αντανακλαστικές μέθοδοι, η αντανακλαστική υπηρεσία εισάγει το πακέτο `java.lang.reflect`.
- Οι τάξεις των `Field`, `Method` και `Constructor`: Αυτές δίδουν αντανακλαστικές πληροφορίες για τα πεδία, τις μεθόδους και τους κατασκευαστές που περιγράφουν και επιτρέπουν την ασφαλούς τύπου χρήση τους. Εφαρμόζουν την διασύνδεση `Member` που βρίσκει πώς καλείται το μέλος, τους τροποποιητές του και σε ποιά τάξη ή διασύνδεση ανήκει.

Αντανάκλαση

```

public Interface Member {
    public abstract Class getDeclaringClass();
    public abstract String getName();
    public abstract int getModifiers(); // decode using
                                        // class Modifiers
}
public final class Field implements Member {
    public Class getType ();
    public Object get (Object obj)
    // if static field, obj is ignored
    throws NullPointerException,
           IllegalArgumentException,
           IllegalAccessException;
    public boolean getBoolean (Object obj)
        throws NullPointerException,
           IllegalArgumentException,
           IllegalAccessException;
    // similar for all other // // // // //
    primitivetypes; avoids wrapping in // get
    public void set (Object obj, Object value)
        throws NullPointerException,
           IllegalArgumentException,
           IllegalAccessException;
    public void setBoolean (Object obj, Boolean z)
        throws NullPointerException,
           IllegalArgumentException,
           IllegalAccessException;
    // similar for all other primitive // // types; avoids
    wrapping in set
}
public final class Method implements Member {
    public Class getReturnType();
    public Class[] getParameterTypes();
    public Class[] getExceptionTypes();
    public Object invoke (Object obj, Object[] args) //returns
    null if
        // return type void
    throws NullPointerException,
           IllegalArgumentException,
           InvocationTargetException;
    // wrapper for exception thrown by // invoked method
}

```

Αντανάκλαση

- Τάξη Class (στην java.lang) επιστρέφει περιπτώσεις τάξεων όταν αναζητεί χαρακτηριστικά συγκεκριμένων τάξεων. Οι σημαντικές μέθοδοι αυτών των τάξεων είναι:

```

Public final class Class {
    public static Class forName (String className) throws
    ClassNotFoundException;
    public Object newInstance() throws
    InstantiationException, IllegalAccessException;
    public boolean isInstance(Object obj);
    public boolean isInterface();
    public boolean isArray();
    public boolean isPrimitive();
    public String getName();
    public int getModifiers(); //decode usingh class
                                //Modifiers
    public ClassLoader getClassLoader();
    public Class getSuperclass(); //null if
    primitive,interface or //class Object
    public Class[] getInterfaces();
    public Class getComponentType();
    //type of array components; null //if not array
    public Class getDeclaringClass();
    //declaring class of inner class //or interface
    public Class[] getClasses(); //public inner classes or
    //interfaces, incl. Inherited ones
    public Field[] getFields(); //public accessible foelds,
    incl. //Inherited ones
    public Method[] getMethods();
    //public methods incl. inherited //ones
    public Constructor[] getConstructors(); //public
    //constructors
    public Class[] getDeclaredClasses() throws
    SecurityRception; //all inner //classes or interfaces,
    exc. //Inherited ones
    public Field[] getDeclaredFields() throws
    SecurityException; //all fields exc. Inherited ones
    public Method[] getDeclaredMethods() throws
    SecurityException; //all methods //exc. Inherited ones
    public Constructor[] getDeclaredConstructors() throws
    SecurityExceptions; //all //constructors
    //further methods to get declared //field, method,
    constructor, or //class by name
}

```

Αντανάκλαση

- Σταθερά αντικείμενα για τους αρχικούς τύπους των γλωσσών. Αν δηλαδή υπάρχει ένα αντικείμενο `java.lang.Boolean.Type` αυτό είναι η τάξη (Class) αντικειμένου, για τον αρχικό τύπο `boolean`. Το Class Array υποστηρίζει την δυναμική κατασκευή και χρήση πινάκων. Το Class Modifier απλοποιεί τον έλεγχο των τροποποιημένων πληροφοριών σε τάξεις πεδία και μεθόδους.

```
Public final class Modifier {  
    public static boolean isPublic(int modifiers); //true if  
    modifiers //inc. public similar for private, //static,  
    final, synchronised, //volatile, transient, native,  
    //interface, abstract, - note that //'interface' is vied  
    as a class //modifier!  
}
```

9.3.2 Σειριοποίηση Αντικειμένου

- Μεχρι την έκδοση JDK 1.0.2 η Java δεν υποστήριζε σειριοποίηση αντικειμένων σε `byte streams`.
- Τώρα για σειριοποίηση το αντικείμενο πρέπει να εφαρμόσει ή την διασύνδεση `java.io.Serializable`, ή την διασύνδεση `java.io.Externalizable`.
- Ο μετατροπέας `transient` σημαδεύει τα πεδία που δεν πρέπει να σειριοποιηθούν.
- Στη εφαρμογή `Serializable` αρκετή πληροφορία γράφεται σε *ρεύματα δεδομένων* (`streams`) έτσι ώστε η αποσειριοποίηση συνεχίζει να δουλεύει, έστω και αν χρησιμοποιούνται διαφορετικές (μα συμβατές) εκδόσεις τάξεων.
- Οι μέθοδοι `readObject` και `writeObject` μπορούν να εφαρμοσθούν για περαιτέρω έλεγχο των πληροφοριών που γράφονται, ή να δώσουν περαιτέρω πληροφορίες στο ρεύμα.
- για να είναι η σειριοποίηση ασφαλής και διαμορφωταία οι μέθοδοι `readObject` και `writeObject` είναι ιδιωτικοί και επομένως μία μόνο τέτοια μέθοδος μπορεί να υπάρξει, στο επίπεδο της κάθε υποτάξης (και μπορεί να χειριστεί πεδία αυτής και μόνο). Η ύπαρξη τέτοιων μεθόδων για κάθε τέτοιο επίπεδο διαπιστώνεται με αντανάκλαση. Αν υπάρχουν καλούν την μέθοδο `defaultReadObject` ή `defaultWriteObject`, πριν να παραχωρήσουν επιπρόσθετα ιδιωτικά δεδομένα.
- Στην εφαρμογή `Externalizable` το ίδιο το αντικείμενο πρέπει να χειριστεί και να φυλάξει τα πεδία και τα περιεχόμενα του. Γι' αυτό υπάρχουν οι μέθοδοι `readExternal` και `writeExternal` που είναι δημόσιοι και όσοι τις χρησιμοποιούν παρακάμπτουν την κανονική τους δημόσια διασύνδεση.
- για κάθε τάξη που μπορεί να σειριοποιηθεί υπάρχει ένας μοναδικός ενδείκτης *serial version ID* σαν κωδικοποιημένο αποτύπωμα του ονόματος της τάξης και όλων των χαρακτηριστικών των εφαρμοσμένων διασυνδέσεων που περιλαμβάνει (τύποι και ενδείκτες τους). Δεν περιλαμβάνει τις υπερτάξεις. Ο υπολογισμός του ενδείκτη γίνεται όταν μία περίπτωση τάξης σειριοποιείται χωρίς *serial version ID*. Αν ήδη έχει, τότε η τάξη δηλώνει συμβατή με την τάξη που αρχικά είχε αυτό το ID. Η συμβατότητα με τις σειριοποιημένες εκδόσεις διατηρείται με τη βοήθεια του `readObject`.
- Κάθε τιμή στο ρεύμα σειριοποίησης κατέχει μία ετικέτα που περιγράφει τον τύπο της τιμής. Σαν σχηματισμός μεταφοράς δεδομένων αυτό είναι πολύ ακριβό.

Σειριοποίηση Αντικειμένου

- Ένα κύριο μειονέκτημα της τωρινής υπηρεσίας σειριοποίησης είναι η απουσία υποστήριξης υποβάθμισης σε περίπτωση ελλείποντων ή ασύμβατων τάξεων. Αν για παράδειγμα δεν μπορούν να υποστηριχθούν τοπικά ορισμένα αντικείμενα, απλώς βγαίνει η εξαίρεση `ClassNotFoundException` χωρίς μέσα συγχρονισμού και συνέχισης της αποσειριοποίησης.

9.3.3 Πρότυπο Κατανεμημένων Αντικειμένων

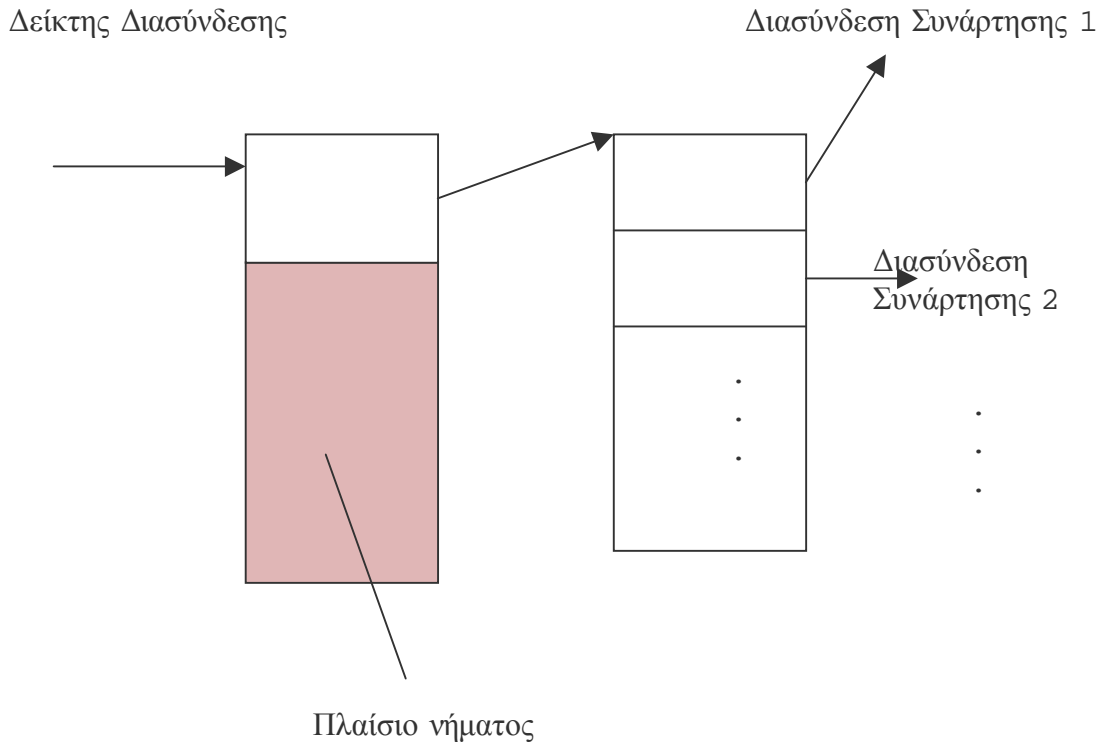
- Αυτό το πρότυπο υποστηρίζεται από την
 - Υπηρεσία σειριοποίησης αντικειμένου.
 - Απόμακρη κλήση μεθόδου (RMI).
 που εισήχθησαν με το JDK 1.1
- Ο χειρισμός κάθε κατανεμημένου αντικειμένου γίνεται με αναφορές σε τύπους διασυνδέσεων. Η απόμακρη πρόσβαση προς αυτές γίνεται με την `java.rmi.Remote`. Σε περίπτωση αποτυχίας βγαίνει η εξαίρεση `java.rmi.RemoteException`.
- για το πέρασμα απόμακρων παραμέτρων (τύποι διασυνδέσεων κ.λ.π.) χρησιμοποιείται η αναφορά τους (*by reference*). Διαφορετικά περνούνται με τιμή (*by value*) με αποσειριοποίηση στην απόμακρη μηχανή πριν την κλήση της απόμακρης διασύνδεσης.
- Δεν σειριοποιούνται όλα τα αντικείμενα της Java και η προσπάθεια να περασθεί μη σειριοποιούμενο αντικείμενο με τιμή προκαλεί εξαίρεση.
- Αυτό το πρότυπο επεκτείνεται και με την συλλογή *αχρήστων*. Αυτό υλοποιείται με προσεκτικό φύλαγμα των αντικειμένων που έχουν απόμακρες αναφορές σε αυτά.
- Ιδιομορφίες του Java RMI:
 - Επεμβαίνει στη ιδέα της ταυτότητας αντικειμένου στη Java (η εφαρμογή απόμακρων αναφορών αντικειμένων δημιουργεί και χειρίζεται αντιπροσώπους αντικειμένων και όχι τα ίδια τα αντικείμενα).
 - Επεμβαίνει στη σημασιολογία του συστήματος κλειδώματος της Java που απορρίπτει το αυτο- κλείδωμα (η ιδέα της ταυτότητας του νήματος δεν εκτείνεται σε πολλές μηχανές και αν μία απόμακρη κλήση εκτελεί μία υπο-κλήση πίσω στην αρχική μηχανή, τότε επέρχεται αδιέξοδος).
 - Μεθόδοι που ορίζονται από την τάξη `java.lang.Object` έχουν υπ' όψη τους το πρότυπο ταυτότητας αντικειμένου της Java. Το αντίστοιχο του `Object` στην Java RMI είναι το `hashCode` και `toString`. Αυτά χρειάζονται διαφορετικό τρόπο εφαρμογής, με επέκταση της τάξεως `java.rmi.RemoteObject`. Το άσχημο είναι ότι αυτό αποκλείει την επέκταση άλλων τάξεων. Επιπλέον η πιο πάνω τάξη δεν μπορεί να προσαρμόσει τις μεθόδους `getClass`, `notify`, `notifyAll` και `wait`. Όταν εργάζονται σε απόμακρες αναφορές, λειτουργούν σε αντιπρόσωπο αντικειμένου.

9.3.4 Τοπική Διασύνδεση της Java (JNI)

- Αυτή καθορίζει για κάθε πλατφόρμα τις τοπικές συνθήκες κλήσης, όταν συνδέεται με τον τοπικό κώδικα έξω από τη νοητή μηχανή της Java. Η JNI δεν καθορίζει δυαδικό πρότυπο αντικειμένου της Java δηλ τρόπους πρόσβασης στα πεδία, ή κλήσεως μεθόδων. Η διαλειτουργικότητα μεταξύ νοητών μηχανών Java στην ίδια πλατφόρμα παραμένει προβλημα. Οι τοπικές μέθοδοι της JNI επιτρέπουν:
 - Δημιουργία, διάβασμα και εκσυγχρονισμό αντικειμένων Java.
 - Κλήσεις μεθόδων Java.
 - Χειρισμό εξαιρέσεων.
 - Φόρτωση τάξεων και απόκτηση πληροφοριών γι'αυτές.
 - Εκτέλεση ελέγχου τύπου.
- Η πραγματική διάταξη των αντικειμένων δεν εκτίθεται στον τοπικό κώδικα. Αντι αυτού η πρόσβαση γίνεται μέσω των *JNI interface pointers*.
- Ο δείκτης (*pointer*) μίας JNI διασύνδεσης μπορεί να αναφερθεί σε ένα συγκεκριμένου νήματος πλαίσιο και δεν αντιστοιχεί σε αντικείμενο Java. Όταν καλείται μία τοπική μέθοδος έχει σαν πρώτη παράμετρο ένα *JNI interface pointer*. Έτσι όλη η πρόσβαση σε αντικείμενα Java γίνεται μέσω συναρτήσεων του *JNI interface pointer* (βλ. σχήμα 9.5).
- Ο συλλέκτης αχρήστων της Java κρατά τα ίχνη όλων των αναφορών που δίδονται (σαν τοπικές αναφορές) στις τοπικές μεθόδους. Ο τοπικός κώδικας μπορεί να μετατρέψει την τοπική αναφορά σε καθολική. Επίσης πληροφορεί την JNI όταν η καθολική αναφορά δεν είναι χρήσιμη πλέον.
- Παραδείγματα των προαναφερθέντων συναρτήσεων της JNI:

```
JNIEnv *env; //the JNI interface pointer
jmethodID mid= env->GetMethodID (ClassPtr, methodName,
    methodSignature); //methodSignature is a string
    representing the mangled signature
jdouble result = env-> CallDoubleMethodode(obj,mid,args);
    //factoring of GetMethodID useful to avoid repeated method
    lookup
```

Τοπική Διασύνδεση της Java (JNI)



Σχ. 9.5 JNI δείκτης διασύνδεσης.

9.4 Διασυνδέσεις έναντι Τάξεων

- Στη Java η διασύνδεση είναι μία πλήρως αφηρημένη τάξη.
- Η JDK 1.1 δεν επιτρέπει τη απόμακρη πρόσβαση σε διασυνδέσεις.