NHMATA

Πολυδιεργαστικά Συστήμτα

- Όταν ξεκινάμε ένα πρόγραμμα λέμε ότι ξεκινάμε μια διεργασία.
- Κατά κανόνα οι μικροί υπολογιστές έχουν ένα επεξεργαστή.
- Σε αυτήν την περίπτωση όταν υποστηρίζουν πολυδιεργασία (πολλές διεργασίες εκτελούνται ταυτόχρονα) θα πρέπει

  => το λειτουργικό σύστημα να έχει την ευθύνη για κατανομή του χρόνου μεταξύ των διεργασιών.

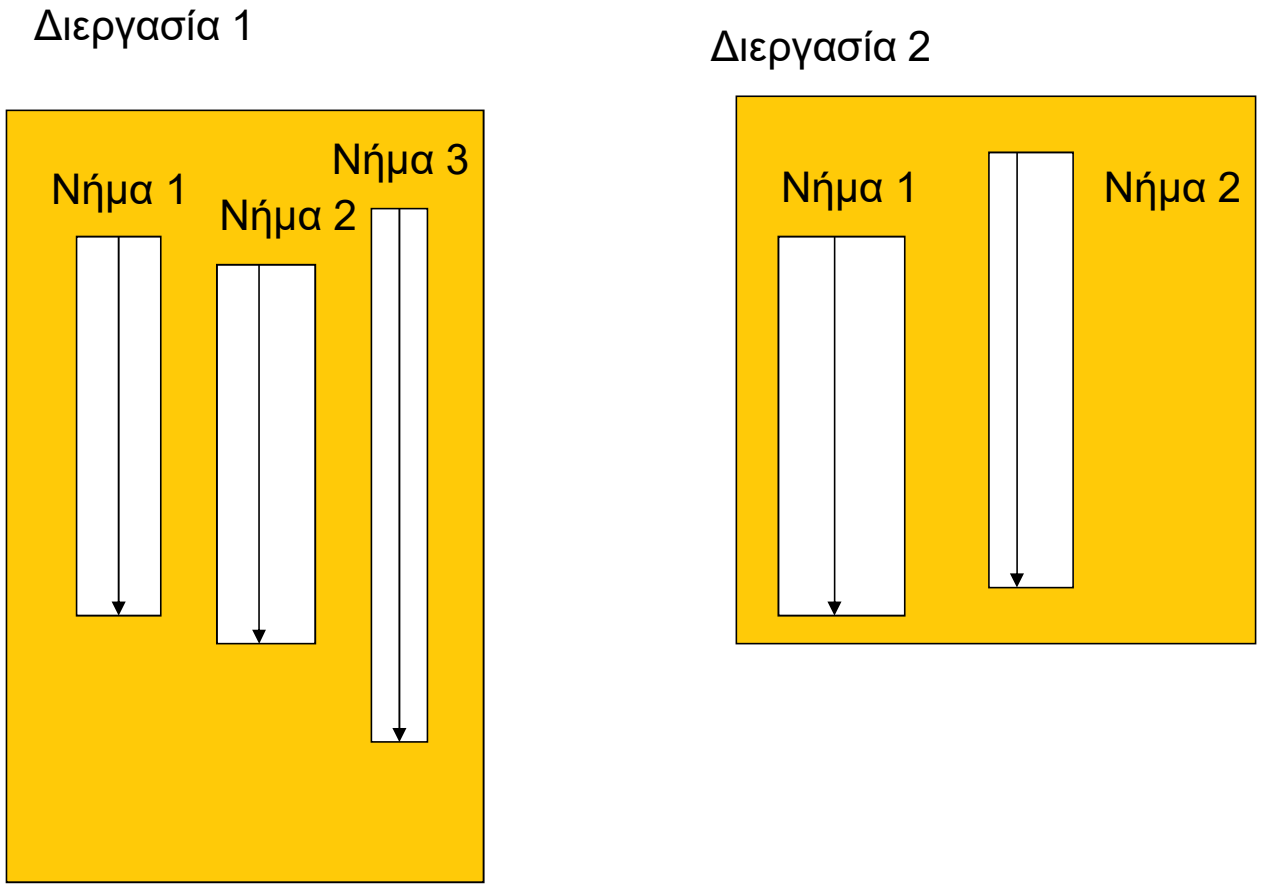
  =>Επίσης θα πρέπει να «θυμάται» που σταμάτησε μια διεργασία για να ξεκινήσει ξανά από το σημείο που σταμάτησε όταν ξαναέρθει η σειρά της. (Θεματική μεταγωγή)

  => επίσης θα πρέπει το λειτουργικό σύστημα να παρέχει λειτουργίες συγχρονισμού μεταξύ των διεργασιών που εκτελούνται ταυτόχρονα. Π.χ μια διεργασία να περιμένει να τελειώσει κατί μια άλλη ή να μην μπορεί να διακοπεί όταν εκτελεί μια συνάρτηση.
- Όταν έχουμε πολυδιεργασία ο χρήστης έχει την αίσθηση ότι εκτελούνται ταυτόχρονα.

Νήματα

- Τα νήματα είναι μίνι διεργασίες, τα οποία αποτελούν τμήμα της διεργασίας.
- Μια διεργασία μπορεί να αποτελείται από πολλά νήματα τα οποία εκτελούνται ταυτόχρονα.
- Τα νήματα ΔΕΝ είναι ξεχωριστή διεργασία άλλα τμήμα κάποια διεργασίας.
- Τα νήματα όμως έχουν δικές τους μεταβλητές και στοίβα εκτέλεσης.

Νήματα



Υλοποίηση Νημάτων στην JAVA

- Κάθα κλάση που θέλει να υλοποιήσει νήματα επεκτείνει την κλάση Thread
- Κάθε νήμα έχει μια μέθοδο run η οποία είναι η μέθοδος που θα εκτελεστεί ταυτόχρονα και η οποία αρχίζει την εκτέλεση της όταν καλεστεί η start.
- Start η μέθοδος η οποία ξεκινάει την εκτέλεση του thread.

Παράδειγμα

```
class NumberPrinter extends Thread {
private int numberToPrint;
public NumberPrinter(int num){
  numberToPrint=number;}
public void run(){
while(true){
system.out.print(numberToPrint);
system.out.print(" ");}}
}
```

```
Class ShowerOfNumbers{
Public static void main(String[] args){
NumberPrinter printer1=new
  NumberPrinter(1);
NumberPrinter printer2=new
  NumberPrinter(2);
NumberPrinter printer3=new
  NumberPrinter(3);
NumberPrinter printer4=new
  NumberPrinter(4);
NumberPrinter printer5=new
  NumberPrinter(5);
Printer1.start();
Printer2.start();
.....
}}
```

Παράδειγμα

3 3 3 1 4 5 2 23 45 3

Η πιο πάνω αποτελεί πιθανή έξοδο του προγράμματος. Όπως βλέπουμε κάποιες φορές δεν υπάρχει κενό. Αυτό σημαίνει ότι εκτέλεση του νήματος σταματάει μεταξύ των δύο print. Κάποιες φορές υπάρχουν δύο κενά. Σημαίνει ότι εκτύπωσε ένα νήμα και μετά πήρε τον έλεγχο ένα άλλο νήμα η εκτέλεση του οποίου είχε διακοπεί μεταξύ των δύο print.

Νήματα και Κληρονιμικότητα

- Αν τα αντικείμενα τα οποία θα υλοποιήσουν τα νήματα κληρονομούν από μια άλλη κλάση τότε έχουμε προβλήματα γιατί δεν μπορούν να κληρονομήσουν από την thread (στην java έχομε μονή κληρονομικότητα).
- Για να αντιμετωπίσομε το πρόβλημα έχουμε την διασύνδεση Runnable την οποία μπορεί να υλοποιήσει η νέα κλάση. Η Runnable μας παρέχει την μέθοδο run άλλα δεν μας παρέχει τις άλλες μεθόδους της κλάσης Thread.
- Για τον λόγο αυτό θα πρέπει να ορίσουμε ένα αντικείμενο της κλάσης Thread για να έχουμε πρόσβαση στην μέθοδο start (δείτε το επόμενο παράδειγμα).

Προχωρημένα ζητήματα στη διαχείριση αντικειμένων

- Η παρούσα διάλεξη αφορά προχωρημένα ζητήματα στη διαχείριση αντικειμένων.
  - Αντιγραφή
  - Σύγκριση
  - Παραγωγή hashCode
- Η Java έχει ενσωματώσει στην κλάση Object μεθόδους που βοηθούν σε αυτές τις λειτουργίες.

Η κλάση java.lang.Object

- Κληρονομείται από όλες τις κλάσεις της Java (άμεσα ή έμμεσα).
- Δεν είναι αφηρημένη (abstract).
- Ορίζει ένα κοινό σύνολο μεθόδων που μπορούν να κληθούν σε κάθε αντικείμενο της Java που δεν βασικού τύπου.
  - Επιτρέπει την επεξεργασία αντικειμένων μέσω γενικού κώδικα
  - Εισάγει κοινή διεπαφή για τα αντικείμενα όλων των τύπων για συνήθεις λειτουργίες όπως οι hashCode(), equals() και clone()

Οι πιο σημαντικές μέθοδοι της κλάσης java.lang.Object

- boolean equals(Object obj)
- int hashCode()
- Object clone()

Η μέθοδος Object.equals()

- Δήλωση:

  public boolean equals(Object obj)

- Η κλάση Object υλοποιεί το πλέον αυστηρό κριτήριο για την ισότητα δύο αντικειμένων:
  - Θεωρούμε τις μεταβλητές αναφοράς x και y. Η κλήση της equals επιστρέφει true αν και μόνο αν οι x και y αναφέρονται στο ίδιο αντικείμενο στη μνήμη. Το ίδιο ισχύει και για τη σύγκριση με τον τελεστή ”==“ (x == y).

Η μέθοδος Object.equals()

- Σε πολλές περιπτώσεις όμως έχει νόημα μια λιγότερο αυστηρή σχέση ισότητας.
  - Παράδειγμα: Δύο αντικείμενα της κλάσης String.

```
String s1 = new String("java");
String s2 = new String("java");
```

- Οι αναφορές s1, s2 δείχνουν σε δύο διαφορετικά αντικείμενα τύπου String στη μνήμη, τα οποία όμως έχουν την ίδια ακριβώς τιμή. Η σύγκριση των δύο αντικειμένων θέλουμε να επιστρέφει true.
- Για να επιτευχθεί αυτό, χρειάζεται η υπερκάλυψη (overriding) της μεθόδου equals της Object στην κλάση String. Αυτό ακριβώς γίνεται στην υλοποίηση της String στη Java.

Γιατί η σωστή υπερκάλυψη της Object.equals() είναι σημαντική

- Η equals χρησιμοποιείται σε πολλές κλάσεις της βιβλιοθήκης της Java που αναπαριστούν συλλογές αντικειμένων, τις λεγόμενες κλάσεις-containers. Οι κυριότεροι containers της Java είναι κλάσεις που υλοποιούν τις παρακάτω διεπαφές (interfaces):
  - List
  - Set
  - Map
- Παραδείγματα:
  - Η κλήση listObj.contains(val) ψάχνει σε μια λίστα για ένα στοιχείο που να είναι ίσο με το var. Η ισότητα ελέγχεται με την equals.
  - Η κλήση setObj.contains(val) ψάχνει σε ένα σύνολο για ένα στοιχείο που να είναι ίσο με το var. Η ισότητα ελέγχεται με την equals.
  - Η κλήση mapObj.get(key) ανακτά από ένα Map το στοιχείο που το κλειδί του είναι ίσο με το key. Η ισότητα ελέγχεται με την equals.

Η σύμβαση για τη μέθοδο equals()

- Σύμφωνα με την προδιαγραφή της Java, η μέθοδος equals υλοποιεί μια σχέση ισοδυναμίας μεταξύ αντικειμένων, για την οποία πρέπει να ισχύουν τα εξής:
  - Αν η x είναι μεταβλητή αναφοράς διάφορη του null, τότε η κλήση x.equals(null) επιστρέφει false
  - Ανακλαστική ιδιότητα: Για κάθε μεταβλητή αναφοράς x, η κλήση x.equals(x) επιστρέφει true.
  - Συμμετρική ιδιότητα: Για κάθε ζεύγος μεταβλητών αναφοράς x και y, η κλήση x.equals(y) επιστρέφει true αν και μόνο αν η κλήση y.equals(x) επιστρέφει true.
  - Μεταβατική ιδιότητα: Για κάθε τριάδα μεταβλητών αναφοράς x, y, και z, αν η κλήση x.equals(y) επιστρέφει true και η κλήση y.equals(z) επιστρέφει true, τότε η κλήση x.equals(z) επιστρέφει true.
  - Συνέπεια: Για κάθε ζεύγος μεταβλητών αναφοράς x και y, πολλαπλές κλήσεις x.equals(y) επιστρέφουν πάντα την ίδια τιμή (είτε true είτε false), όσο οι μεταβλητές που διοχετεύονται ως ορίσματα στις κλήσεις της equals δεν μεταβάλλονται.

Παράδειγμα

- Η υλοποίηση της equals έτσι ώστε να τηρείται η προαναφερθείσα σύμβαση είναι εκ πρώτης όψεως εύκολη υπόθεση. Στην πραγματικότητα όμως είναι κάτι που χρειάζεται ιδιαίτερη προσοχή σε ορισμένες τουλάχιστον περιπτώσεις, όπως για παράδειγμα, όταν χρησιμοποιούμε στις κλάσεις μας κληρονομικότητα.
- Θα εξετάσουμε το σχετικό ζήτημα μέσα από ένα παράδειγμα. Έστω ότι επιθυμούμε να έχουμε μια κλάση (έστω Point) για την αναπαράσταση σημείων ως ζεύγη καρτεσιανών συντεταγμένων, καθώς και μια υποκλάση της (έστω ColorPoint) που αποθηκεύει επιπλέον και πληροφορία για το χρώμα του σημείου.
- Η κλάση Point θα υπερκαλύπτει την equals της Object, ώστε η σύγκριση να γίνεται βάσει των τιμών των καρτεσιανών συντεταγμένων. Η κλάση ColorPoint θα υπερκαλύπτει την equals της Point έτσι ώστε να λαμβάνεται υπόψη στη σύγκριση και η τιμή της μεταβλητής στιγμιοτύπου που αναπαριστά το χρώμα.

Υλοποίηση κλάσης Point

```
public class Point
{
	private int x;
	private int y;

	public Point() {
		x = 0;
		y = 0;
	}

	public Point(int x, int y) {
		this.x = x;
		this.y = y;
	}

	…
}
```

Υλοποίηση κλάσης ColorPoint

```
import java.awt.Color;

public class ColorPoint extends Point
{
	private Color color;

	public ColorPoint() {
		super();
	}

	public ColorPoint(int x, int y, Color color) {
		super(x, y);
		this.color = color;
	}

	…
}
```

Μία πιθανή υλοποίηση της equals για την Point

```
public boolean equals(Object obj) {
  if (this == obj) return true;
  if (obj == null)
        return false;
  if (!(obj instanceof Point)) {
        return false;
  }
  Point p = (Point) obj;
  return this.x == p.x && this.y == p.y;
}
```

Μία πιθανή υλοποίηση της equals για την ColorPoint

```
public boolean equals(Object obj) {
  if (this == obj) return true;
  if (!super.equals(obj))
        return false;
  if (!(obj instanceof ColorPoint)) {
        return false;
  }
  ColorPoint cp = (ColorPoint) obj;
  return this.color == cp.color;
}
```

Παραβίαση συμμετρικής ιδιότητας

```
Point p = new Point(5, 5)
ColorPoint cp = new ColorPoint(5, 5, Color.RED);
System.out.println(“p.equals(cp): "
                                + p.equals(cp));
System.out.println(“cp.equals(p): "
                                + cp.equals(p));
```

Εκτύπωση Προγράμματος:

p.equals(cp): true

cp.equals(p): false

Συνεπώς έχουμε παραβίαση της σύμβασης της μεθόδου equals, που μπορεί να έχει καταστροφικές επιπτώσεις σε κώδικα που χρησιμοποιεί την ColorPoint. Για παράδειγμα στο παρακάτω τμήμα προγράμματος ***δεν*** μπορούμε να είμαστε σίγουροι ποια θα είναι η τιμή της μεταβλητής b μετά την τελευταία εντολή.

```
List list = new ArrayList();
list.add(cp);
Boolean b = list.contains(p);
```

Μία εναλλακτική υλοποίηση της equals για την ColorPoint έτσι ώστε να ισχύει η συμμετρική ιδιότητα

```
public boolean equals(Object obj) {
  if (this == obj) return true;
  if (!(obj instanceof Point)) {
        return false;
  }
  if (!(obj instanceof ColorPoint)) {
        return obj.equals(this);
  }
  ColorPoint cp = (ColorPoint) obj;
  return super.equals(obj) && this.color == cp.color;
}
```

Παραβίαση μεταβατικής ιδιότητας

```
ColorPoint cp1 = new ColorPoint(5, 5, Color.RED);
Point p = new Point(5, 5)
ColorPoint cp2 = new ColorPoint(5, 5, Color.BLUE);
System.out.println(“cp1.equals(p): " + cp1.equals(p));
System.out.println(“p.equals(cp2): " + p.equals(cp2));
System.out.println(“cp1.equals(cp2): " + cp1.equals(cp2));
```

Εκτύπωση Προγράμματος:

```
cp1.equals(p): true
p.equals(cp2): true
cp1.equals(cp2): false
```

Συνεπώς έχουμε παραβίαση της σύμβασης της μεθόδου equals, αφού δεν ισχύει η μεταβατική ιδιότητα. Γενικά είναι πολύ επικίνδυνο να επιτρέπουμε σε αντικείμενα διαφορετικών κλάσεων να θεωρούνται ίσα από την equals. Αυτό ίσως έχει νόημα σε κάποια αφηρημένη βασική κλάση (π.χ. AbstractList, AbstractMap) αλλά κι εκεί με μεγάλη προσοχή!

Προτεινόμενη δομή υλοποίησης/υπερκάλυψης μεθόδου equals

- Έλεγχος αν το όρισμα είναι αναφορά στο this και επιστροφή true, αν αυτό ισχύει.
- Αν η κλάση κληρονομεί απευθείας από την Object, έλεγχος αν το όρισμα είναι null. Αν ναι, επιστροφή false.
- Αν η κλάση δεν κληρονομεί απευθείας από την Object, κλήση της super.equals με όρισμα ίδιο με το τρέχον. Αν η κλήση επιστρέψει false, επιστροφή false.
- Έλεγχος αν ο τύπος (η κλάση) του ορίσματος είναι ο ίδιος με τον τύπο του this (με κλήση της getClass(), ***ΟΧΙ*** με τον τελεστή instanceof). Αν όχι, επιστροφή false.
- Δημιουργία νέου αντικειμένου ίδιου τύπου με το this και εξίσωσή του με το όρισμα (μετά από μετατροπή του τελευταίου από Object στο σωστό τύπο).
- Σύγκριση των αντίστοιχων μεταβλητών στιγμιοτύπου του ορίσματος και του this. Χρήση της μεθόδου equals για τη σύγκριση όπου απαιτείται.

Προτεινόμενη υλοποίηση της equals για την Point

```
public boolean equals(Object obj) {
  if (this == object)
        return true;
  if (obj == null)
        return false;
  if (obj.getClass() != this.getClass()) {
        return false;
  }
  Point p = (Point) obj;
  return this.x == p.x && this.y == p.y;
}
```

Προτεινόμενη υλοποίηση της equals για την ColorPoint

```
public boolean equals(Object obj) {
  if (this == obj)
        return true;
  if (!super.equals(obj))
        return false;
  if (obj.getClass() != this.getClass()) {
        return false;
  }
  ColorPoint cp = (ColorPoint) obj;
  return this.color == cp.color;
}
```

Η κλάση CaseInsensitiveString

- Προβλήματα στην υλοποίηση της equals δεν προκαλούνται μόνο όταν έχουμε κληρονομικότητα. Μια άλλη συνήθης πηγή προβλημάτων εμφανίζεται όταν επιτρέπουμε σε αντικείμενα διαφορετικών κλάσεων να θεωρούνται ίσα από την equals. Θα εξετάσουμε το σχετικό ζήτημα μέσα από ένα παράδειγμα.
- Έστω ότι επιθυμούμε να έχουμε μια κλάση για την αποθήκευση συμβολοσειρών για τις οποίες δεν μας ενδιαφέρει το αν οι χαρακτήρες που την αποτελούν είναι κεφαλαίοι ή πεζοί. Αφού δεν μπορούμε να ορίσουμε υποκλάσεις της String (η οποία είναι final κλάση) θα χρησιμοποιήσουμε σύνθεση της String στη νέα κλάση, την οποία ονομάζουμε CaseInsensitiveString.
- Η νέα κλάση θα υπερκαλύπτει την equals της String, έτσι ώστε να μη λαμβάνεται υπόψη στη σύγκριση το αν οι περιεχόμενοι χαρακτήρες είναι κεφαλαίοι ή πεζοί.

Υλοποίηση κλάσης CaseInsensitiveString

```
public final class CaseInsensitiveString {
        private String s;
        public CaseInsensitiveString(String s) {
                if (s == null)
                        throw new NullPointerException();
                this.s = s;
        }
        // Υλοποίηση της μεθόδου equals που παραβιάζει τη συμμετρικότητα!
        public boolean equals(Object o) {
                if (o instanceof CaseInsensitiveString)
                        return s.equalsIgnoreCase(
                                ((CaseInsensitiveString)o).s);
                // Προσπάθεια για διαλειτουργικότητα με αντικείμενα τύπου String!
                if (o instanceof String)
                        return s.equalsIgnoreCase((String)o);
                return false;
        }
        ... // Υπόλοιπες μέθοδοι της κλάσης
}
```

Παραβίαση συμμετρικής ιδιότητας

```
String str1 = new String(“Dimitris");
CaseInsensitiveString str2 = new String(“dimitris");
System.out.println(“str1.equals(str2): "
                                + str1.equals(str2));
System.out.println(“str2.equals(str1): "
                                + str2.equals(str1));
```

Εκτύπωση Προγράμματος:

str1.equals(str2): false

str2.equals(str1): true

Συνεπώς έχουμε παραβίαση της σύμβασης της μεθόδου equals, που μπορεί να έχει καταστροφικές επιπτώσεις σε κώδικα που χρησιμοποιεί την CaseInsensitiveString. Για παράδειγμα στο παρακάτω τμήμα προγράμματος ***δεν*** μπορούμε να είμαστε σίγουροι ποια θα είναι η τιμή της μεταβλητής b μετά την τελευταία εντολή.

```
List list = new ArrayList();
list.add(str2);
Boolean b = list.contains(str1);
```

Η μέθοδος Object.hashCode()

- Επιστρέφει έναν ακέραιο αριθμό.
- Σύμφωνα με την προδιαγραφή της Java, για τη μέθοδο hashCode πρέπει να ισχύουν τα εξής:
  - Δύο διαφορετικές κλήσεις της hashCode στο ίδιο αντικείμενο και στο πλαίσιο της ίδιας εκτέλεσης μιας εφαρμογής Java πρέπει να επιστρέφουν την ίδια τιμή, αν δεν έχουν μεταβληθεί μεταξύ των κλήσεων οι τιμές μεταβλητών του αντικειμένου από τις οποίες εξαρτάται το αποτέλεσμα της equals.
  - Αν η κλήση x.equals(y) επιστρέφει true, τότε πρέπει x.hashCode() == y.hashCode().
  - Αν η κλήση x.equals(y) επιστρέφει false, τότε ***ΔΕΝ*** είναι απαραίτητο να επιστρέφει false η x.hashCode() == y.hashCode()

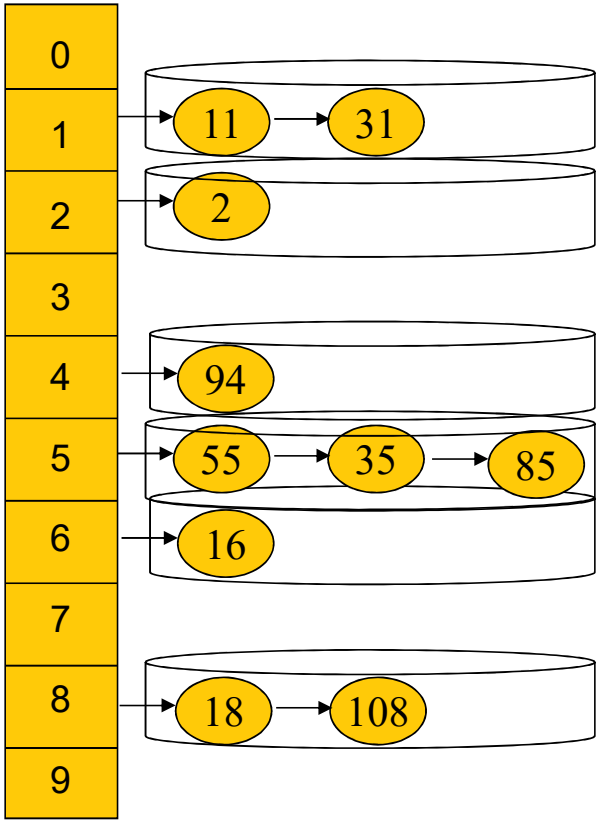
Γιατί η σωστή υπερκάλυψη της Object.hashCode() είναι σημαντική

- Η hashCode χρησιμοποιείται σε κάποιες κλάσεις της βιβλιοθήκης της Java που αναπαριστούν συλλογές αντικειμένων και χρησιμοποιούν την τεχνική hashing (π.χ. HashMap, HashSet, Hashtable).
- Με βάση τις τιμές της hashCode για τις τιμές που χρησιμοποιούνται ως κλειδιά γίνεται αναζήτηση στοιχείων με κριτήριο κλειδί.
- Αν η hashCode είναι τέτοια ώστε να υπάρχει ομοιόμορφη κατανομή κλειδιών, τότε η αναζήτηση είναι πολύ γρήγορη.
- Αν μεταβληθεί ένα από τα αντικείμενα που έχουμε θέσει ως κλειδιά, τότε έχουμε απρόβλεπτα (και ίσως καταστροφικά) αποτελέσματα στην εφαρμογή μας!

Παράδειγμα Hashtable

Συνάρτηση Hash (πολύ απλοϊκή):
Το τελευταίο ψηφίο της τιμής του κλειδιού.

Μαζί με τις τιμές των κλειδιών αποθηκεύονται και οι τιμές των στοιχείων, οι οποίες παραλείπονται στο σχήμα χάριν οικονομίας χώρου.



Κλάσεις συλλογών που λειτουργούν με hashing

- Hashtable, HashMap, LinkedHashMap
- Μοιάζουν με λίστες, αλλά με σημαντικές διαφορές
  - Ουσιαστικά πρόκειται για αντιστοίχιση κλειδιών με τιμές στοιχείων
  - Δεν υπάρχει διάταξη των στοιχείων, η ανάκτηση γίνεται βάσει κλειδιού
  - Το κλειδί μπορεί να είναι οποιουδήποτε τύπου αναφοράς
- Πολύ χρήσιμες και διαδεδομένες στην πράξη, γιατί επιτρέπουν γρήγορη ανάκτηση με βάση κάποιο χαρακτηριστικό του στοιχείου και όχι απλούς δείκτες

Χρήση συλλογών που λειτουργούν με hashing

- Έστω ότι θέλουμε να αποθηκεύσουμε στοιχεία εγγεγραμμένων χρηστών ενός δικτυακού τόπου με κλειδί το email τους
  - Χρησιμοποιούμε μια δική μας κλάση EmailAddress για την αναπαράσταση
  - Αποθηκεύουμε τα στοιχεία των χρηστών σε ένα Hashtable με κλειδί το email τους
- Προσοχή: Πρέπει πάντα υλοποιηθούν σωστά οι μέθοδοι equals και hashCode της κλάσης EmailAddress!

Παράδειγμα χρήσης Hashtable

- Θεωρούμε τα παρακάτω αντικείμενα
  - EmailAddress email1 = new EmailAddress(“user1”, “gmail.com”);
  - User user1 = new User(…);
- Συνήθεις λειτουργίες με hashtable
  - Δημιουργία hashtable συγκεκριμένης αρχικής χωρητικότητας και load factor
    - Hashtable myHashtable = new Hashtable(100, 0.75);
  - Προσθήκη στοιχείου
    - myHashtable.put(email1, user1)
  - Έλεγχος αν υπάρχει ένα συγκεκριμένο κλειδί
    - myHashtable.containsKey(email1) (επιστρέφει true ή false)
  - Ανάκτηση στοιχείου βάσει τιμής κλειδιού
    - User user2 = (User) myHashtable.get(email1);

Οδηγίες για την υλοποίηση/υπερκάλυψη της hashCode()

- Όταν μια κλάση υπερκαλύπτει την equals, πρέπει απαραιτήτως να υπερκαλύπτει και την hashCode.
- Οι μεταβλητές στιγμιοτύπου από τις οποίες εξαρτάται η hashCode πρέπει να είναι οι ίδιες από τις οποίες εξαρτάται η equals.
- Για σύνθετες κλάσεις, υπολογίζουμε το hashCode ως άθροισμα των επιμέρους hashCodes των μεταβλητών στιγμιοτύπου.
- Προσπαθούμε να υλοποιούμε την hashCode, έτσι ώστε οι ακέραιοι που παράγονται να ακολουθούν ομοιόμορφη κατανομή στο διάστημα από 0 έως Integer.MAX_VALUE.

Παράδειγμα απλής υλοποίησης της hashCode()

```
public class CountedString {
  private String s;
  private int id = 0;
  …
  public int hashCode() {
  // Επιλέγουμε πρώτο αριθμό τόσο για αρχική τιμή του
   // αποτελέσματος όσο και για συντελεστή πολλαπλασιασμού.
      int result = 17;
        result = 37*result + s.hashCode();
        result = 37*result + id;
        return result;
   }
  }
```

Η μέθοδος Object.clone()

- Αυτό που κάνει η Object.clone() είναι να παράγει ένα αντίγραφο του αντικειμένου στο οποίο καλείται, όπου:
  - Αντιγράφονται οι τιμές των μεταβλητών βασικού τύπου.
  - Αντιγράφονται οι αναφορές (shallow copy) των μεταβλητών αναφοράς. Αν θέλουμε deep copy για τις μεταβλητές αναφοράς μιας κλάσης, τότε αυτή πρέπει να υπερκαλύπτει την clone της Object.
- Η μέθοδος Object.clone() είναι ***protected***
- Κάθε κλάση που υλοποιεί την clone πρέπει να υλοποιεί τη διεπαφή Cloneable
  - Αν κληθεί η clone σε αντικείμενο κλάσης που δεν υλοποιεί την Cloneable, παράγεται CloneNotSupportedException.
  - Η διεπαφή Cloneable δεν περιέχει καμία δήλωση μεθόδου (ούτε της clone), είναι δηλαδή ένα marker interface, όπως λέγεται στη Java.
- Η κλάση java.lang.Object δεν υλοποιεί την Cloneable, περιέχει όμως υλοποίηση της clone.

Οδηγίες για την υλοποίηση/υπερκάλυψη της clone()

- Για την υλοποίηση της clone σε οποιαδήποτε κλάση συνίστανται να ισχύουν οι παρακάτω κανόνες, οι οποίοι όμως δεν αυστηρά είναι δεσμευτικοί από τη Java:
  - Η παράσταση (x.clone( ) != x) είναι true
  - (x.clone( ).getClass( ) == x.getClass( )) είναι true
  - Η κλήση (x.clone( ).equals(x)) είναι true

Οδηγίες για την υλοποίηση/υπερκάλυψη της clone()

- Επικεφαλίδα:
  - public Object clone( )
    throws CloneNotSupportedException
- Συνιστώμενη δομή:
  - Ποτέ να μην γίνεται η αντιγραφή με κλήση ενός constructor της κλάσης. Αντί για αυτό, το σωστό είναι να δημιουργείται ένα (αρχικό) αντίγραφο με κλήση της super.clone().
  - Να υλοποιείται από την κλάση η διεπαφή Cloneable.
    - Όλα τα arrays θεωρείται ότι υλοποιούν την Cloneable.
  - Να καλείται η clone() σε καθεμία από τις μη στατικές μεταβλητές αναφοράς της κλάσης των οποίων ο τύπος είναι μια μεταλλάξιμη (mutable) κλάση.
    - Σε κάθε κλήση της clone χρειάζεται κατάλληλο casting.

Γιατί δεν είναι σωστή η αντιγραφή μέσω constructor μέσα στην clone

- Στην τεκμηρίωση της Java για την clone() αναφέρεται ότι στο εσωτερικό της δεν πρέπει να καλείται constructor για την παραγωγή του αντιγράφου.
- Η κλήση super.clone() σε υποκλάση επιστρέφει λάθος τύπο (και παράγει ClassCastException) αν έχει χρησιμοποιηθεί constructor στην clone() της βασικής κλάσης.

Παράδειγμα λάθος υλοποίησης της clone()

- Έστω η κλάση Person με υλοποίηση της clone() που καλεί τον constructor της:

```
public class Person implements Cloneable
{
  public Person ( <παράμετροι> )
  {  }

  public Object clone( )
    throws CloneNotSupportedException
  {
   return new Person( <παράμετροι> );
  }
}
```

Παράδειγμα λάθος υλοποίησης της clone()

- Έστω ότι η Student είναι μια υποκλάση της Person. Η υλοποίηση της clone() στην Student δεν μπορεί να καλέσει την super.clone() και να μετατρέψει τον τύπο του αποτελέσματος στη κλάση Student:

```
public class Student extends Person
{
  private Date enrollDate;
  public Object clone( ) throwsCloneNotSupportedException
  {
    // ΛΑΘΟΣ: Παράγεται ClassCastException
     Student stu = (Student)super.clone( );

     if( enrollDate != null ) stu.enrollDate = (Date)enrollDate.clone( );
     return stu;
  }
}
```

Παράδειγμα σωστής υλοποίησης της clone() στη βασική κλάση

- Μια σωστή υλοποίηση της clone() για την κλάση Person είναι η εξής:

```
public class Person implements Cloneable
{
  public Person ( <παράμετροι> ) {   }
  public Object clone( )
  {
     Person p = null;
     try { p = (Person) super.clone(); }
       catch(CloneNotSupportedException e) {
          System.err.println("My Person can't clone"); }
    … // Αντιγραφή μεταλλάξιμων μεταβλητών αναφοράς.
       return p;
 } }
```

Άλλες μέθοδοι της κλάσης java.lang.Object

- String toString()
  - Παράγει μια αναπαράσταση του αντικειμένου με μορφή συμβολοσειράς.
- void finalize( )
  - Καλείται από garbage collector πριν αποδεσμευτεί το αντίστοιχο αντικείμενο.
- Class getClass()
  - Επιστρέφει την κλάση του τρέχοντος αντικειμένου.