Πανεπιστήμιο Κρήτης
Τμήμα Επιστήμης Υπολογιστών

HY-252 – Αντικειμενοστρεφής Προγραμματισμός
Βασίλης Χριστοφίδης

Τελική Εξέταση (3 ώρες)
Ημερομηνία: 20 Ιανουαρίου 2011

Άσκηση 1	Άσκηση 2	Άσκηση 3	Άσκηση 4	Άσκηση 5	Άσκηση 6	Άσκηση 7	Σύνολο
/12	/10	/27	/11	/12	/26	/9	/107

Ονοματεπώνυμο:
Αριθμός Μητρώου:

Άσκηση 1 (12 μονάδες) Χειρισμός Εξαιρέσεων

Η παρακάτω μέθοδος Java παίρνει σαν όρισμα έναν πίνακα ακεραίων και υπολογίζει τον μέσο όρο των ακεραίων του πίνακα. Η μέθοδος μεταγλωττίζεται επιτυχώς αλλά μπορεί να εγείρει τρεις τουλάχιστον διαφορετικές εξαιρέσεις κατά την εκτέλεσή της (run-time exceptions) ανάλογα με τον τρόπο με τον οποίο χρησιμοποιείται.

```
public static int intArrayAverage(int[] intArray) {
      int sum = 0;
      for (int i = 0; i <= intArray.length; i++)  (1) {
            sum = sum + intArray[i]; (2)
      }
      return sum/intArray.length;  (3)
}
```

α) **(6 μονάδες)** Υπογραμμίστε στην ακόλουθη λίστα τρεις πιθανές εξαιρέσεις που μπορεί να εγείρει η μέθοδος `intArrayAverage()` και υποδείξτε σε ποια γραμμή του κώδικά της μπορούν να συμβούν.

```
ArithmeticException
ArrayIndexOutOfBoundsException
ClassNotFoundException
CommentTooLongException
DataFormatException
FontFormatException
IndexOutOfBoundsException
IOException
NegativeArraySizeException
NullPointerException
NegativeZeroException
StringIndexOutOfBoundsException
TimeoutException
UglyNameException
UndefinedVariableException
```

Λύση:
```
ArithmeticException (3)
ArrayIndexOutOfBoundsException (2)
NullPointerException (1)
```
Basically get 2 marks for each answer, 1 for exception and 1 for location

β) **(6 μονάδες)** Διορθώστε την παραπάνω μέθοδο έτσι ώστε να αποφευχθεί η δημιουργία των εξαιρέσεων που έχετε δώσει σαν απάντηση στο προηγούμενο ερώτημα.

Λύση:
```
public static int intArrayAverage(int [] intArray) {
      int sum = 0;
      if ((intArray == null) || (intArray.length == 0))
        return 0;
      for (int i = 0; i < intArray.length; i++) {
            sum = sum + intArray[i];
      }
      return sum/intArray.length;
}
```
Basically get 3 marks for each correct correction made above.

Άσκηση 2 (10 μονάδες) Κληρονομικότητα Κλάσεων & Πολυμορφισμός

Για κάθε ένα από τα παρακάτω ερωτήματα γράψτε τι θα τυπωθεί στην στάνταρ έξοδο του ακόλουθου προγράμματος όταν χρησιμοποιήσετε τους αντίστοιχους ορισμούς των κλάσεων Java που δίνονται:

```
C x = new C();
System.out.println(x.foo());
```

Σε περίπτωση λάθους, σημειώστε το πρόβλημα κατά την μετάφραση ή την εκτέλεση του προγράμματος που πιστεύετε ότι θα δημιουργηθεί.

(α) **(5 μονάδες)**
```
abstract class A {
    abstract int foo();
    abstract int bar(A a);
}
class B extends A {
    int bar(A a) { return 2; }
    int foo() { return bar(this); }
}
class C extends B {
    int bar(A a) { return 3; }
}
```

Λύση:
3

(β) **(5 μονάδες)**
```
abstract class A {
    int bar(A a) { return 0; }
    int bar(B b) { return 1; }
    int bar(C c) { return 2; }
}
class B extends A {
    int bar(A a) { return 3; }
    int bar(B b) { return 4; }
    int bar(C c) { return 5; }
}
class C extends B {
    int foo() { return ((A)this).bar((A)this); }
}
```

Λύση:
3

Άσκηση 3 (27 μονάδες) Χειρισμός Πλαισίου Συλλογών Αντικειμένων

Σε αυτή την άσκηση μας ενδιαφέρει η υλοποίηση μιας κλάσης Java για τον χειρισμό του καταλόγου (catalog) των βιβλίων μιας βιβλιοθήκης (Library). Για λόγους απλότητας, υποθέτουμε ότι ο κατάλογος δεν περιέχει πολλαπλά αντίτυπα του ίδιου βιβλίου. Κάθε βιβλίο χαρακτηρίζεται από ένα σύνολο λέξεων-κλειδιών (index terms) που περιγράφουν το περιεχόμενό του. Σημειώστε ότι ο ίδιος όρος μπορεί να περιγράφει παραπάνω από ένα βιβλία και κατά συνέπεια θα πρέπει να θεωρήσετε στην υλοποίηση της κλάσης αντιστοιχίσεις με πολλαπλές τιμές (Multi-Maps). Η βασική υπηρεσία της βιβλιοθήκης που θέλουμε να υλοποιήσουμε αποδοτικά, είναι η αναζήτηση βιβλίων χρησιμοποιώντας έναν ή περισσότερους όρους: ένα βιβλίο επιστρέφεται στην απάντηση μιας ερώτησης, εάν περιέχει οποιοδήποτε από τους όρους της ερώτησης στην περιγραφή του.

Βοήθεια: Για την υλοποίηση της κλάσης `LibraryCatalog` θα πρέπει να βασιστείτε στις διεπαφές της Java `Set` και `Map`. Αξιοποιήστε τις μαζικές (bulk) μεθόδους που προσφέρουν αυτές οι διεπαφές όπως `addAll()` και `retainAll()`. Μπορείτε να βρείτε τα javadocs όλων των μεθόδων και των δύο διεπαφών στην τελευταία σελίδα του διαγωνίσματος.

α) **(13 μονάδες)** Υλοποιήστε την κλάση `LibraryCatalog` επιλέγοντας την κατάλληλη αναπαράσταση του καταλόγου των βιβλίων μιας βιβλιοθήκης σύμφωνα με το Πλαίσιο Συλλογών Αντικειμένων της Java (Java Collection Framework). Πιο συγκεκριμένα, δηλώστε στην κλάση σας πεδία για α) τη συλλογή των βιβλίων (`collection`) και β) το ευρετήριο των όρων του καταλόγου (`termIndex`) της βιβλιοθήκης, επιλέγοντας τους καταλληλότερους τύπους, σύμφωνα με τις προδιαγραφές που σας δόθηκαν προηγουμένως. Επιπλέον, υλοποιήστε τον κατασκευαστή (constructor) της κλάσης ο οποίος παίρνει σαν μοναδικό όρισμα μια συλλογή από βιβλία (`Collection<Book> books`) και αρχικοποιεί κατάλληλα τα δύο πεδία της κλάσης, με βάση την παράμετρο αυτή. Η χρήση γενικών τύπων (generics) της Java δεν είναι υποχρεωτική.

```
import java.util.*;
public class Book {
      private String title;
      private Set<Author> authors; // ένα Set με αντικείμενα τύπου Author
      private Set<String> terms; // ένα Set με αντικείμενα τύπου String
      public String getTitle() {return title;}
      public Set<Author> getAuthors() {return authors;}
      public Set<String> getTerms(){return terms;}
}

public class LibraryCatalog {
```

Λύση:

```
private Set<Book> collection;
private Map<String,Set<Book>> termIndex;
public LibraryCatalog(Collection<Book> books) {
  collection = new HashSet<Book>(books);
  termIndex = new HashMap<String,Set<Book>>();
     for (Book b: collection) {
         for (String term: b.getTerms()) {
              if (termIndex.containsKey(term))
                  termIndex.get(term).add(b);
              else {
                  Set<Book> idx = new HashSet<Book>();
                  idx.add(b);
                  termIndex.put(term,idx);
              }
         }
     }
}

    private Set collection;
    private Map termIndex;
    public LibraryCatalog(Collection books){
        collection = new HashSet();
        termIndex=new HashMap();
        for(Object o:books){
```

```
            Book b = (Book)o;
            collection.add(b);
            for(Object o2:b.getTerms()){
                String term=(String)o2;
                if(termIndex.containsKey(term))
                    ((Set)termIndex.get(term)).add(b);
                else{
                    Set idx=new HashSet();
                    idx.add(b);
                    termIndex.put(term,idx);
                }
            }
        }
}
```

β) **(5 μονάδες)** Υλοποιήστε την μέθοδο `searchSome(String[] terms)` η οποία επιστρέφει το σύνολο των βιβλίων που περιέχουν στην περιγραφή τους *τουλάχιστον έναν* από τους όρους του πίνακα που δίνεται σαν όρισμα. Εάν ο πίνακας όρων είναι κενός, το κενό σύνολο θα πρέπει να επιστρέφεται.

```
public Set searchSome(String[] terms) {
```

Λύση:

```
  Set<Book> found = new HashSet<Book>();
  for (String term: terms)
    if (termIndex.containsKey(term))
        found.addAll(termIndex.get(term));
  return found;
```

```
        Set found = new HashSet();
        for(String term : terms )
            if(termIndex.containsKey(term))
                found.addAll((Set)termIndex.get(term));
            return found;
    }
}
```

γ) **(9 μονάδες)** Υλοποιήστε την μέθοδο `searchAll(String[] terms)` η οποία επιστρέφει το σύνολο των βιβλίων που περιέχουν στην περιγραφή τους *όλους τους όρους* του πίνακα που δίνεται σαν όρισμα. Εάν ο πίνακας όρων είναι κενός, το κενό σύνολο θα πρέπει να επιστρέφεται.

```
public Set searchAll(String[] terms) {
```

Λύση:

```
if (terms.length == 0)
     return collection;
Set<Book> found = new HashSet<Book>(termIndex.get(terms[0]));
for (int i = 1; i < terms.length && !found.isEmpty(); i++) {
     if (termIndex.containsKey(terms[i]))
         found.retainAll(termIndex.get(terms[i]));
     else
         found.clear();
     }
}
return found;
```

```
        Set found = new HashSet();
        for(String term : terms )
            if(termIndex.containsKey(term))
                found.retainAll((Set)termIndex.get(term));
            return found;
    }
}
```

Άσκηση 4 (11 μονάδες) Βασική Λειτουργικότητα Αντικειμένων

α) **(5 μονάδες)** Σας δίνεται η ακόλουθη κλάση `Photo`. Υλοποιήστε την μέθοδο `compareTo()` έτσι ώστε η φυσική διάταξη (natural order) των φωτογραφιών να είναι αύξουσα ως προς το μέγεθος (`size`) τους. Φωτογραφίες που έχουν το ίδιο μέγεθος διατάσσονται σε αύξουσα σειρά ως προς το όνομά (`filename`) τους.

```
import java.util.Date;
public class Photo implements Comparable<Photo> {
      private String filename;
      private int size;
      private Date date;
      // Constructors, etc.
      public String getFilename() {
            return this.filename;
      }
      public int getSize() {
            return this.size;
      }
      public int compareTo(Photo p) {
```

Λύση:

```
int c = this.size - p.getSize();
      if (c == 0) {
          c = this.filename.compareTo(p.getFilename());
      }
      return c;

      }
}
```

β) **(6 μονάδες)** Σας δίνεται ο παρακάτω κώδικας της κλάσης `SortingClient`. Υλοποιήστε τις κλάσεις `StringAscending` και `StringDescending` που είναι απαραίτητες στις διαδοχικές κλήσεις της μέθοδου `Collections.sort()` ώστε να παραχθεί η ακόλουθη έξοδος..

```
public class SortingClient {
  public static void main(String[] args) {
      ArrayList<String> a = new ArrayList<String>();
      a.add("red");
      a.add("orange");
      a.add("yellow");
      a.add("green");
      a.add("blue");
      Collections.sort(a, new StringAscending());
      print(a);
      Collections.sort(a, new StringDescending());
      print(a);
  }
}
```

Output:

```
blue green orange red yellow
yellow red orange green blue
```

Βοήθεια: Η μέθοδος `Collections.sort()` παίρνει δύο ορίσματα. Το πρώτο είναι η λίστα (`ArrayList`) προς ταξινόμηση και το δεύτερο ένα αντικείμενο τύπου `Comparator` που καθορίζει πως θα συγκριθούν μεταξύ τους τα στοιχεία της λίστας. Στην πρώτη κλήση της μεθόδου `sort()` ένα στιγμιότυπο του `Comparator` (`StringAscending`) χρησιμοποιείται για να συγκρίνει δυο συμβολοσειρές σε αύξουσα λεξικογραφική διάταξη (π.χ. "a" < "b"). Στην δεύτερη της μεθόδου `sort()` ένα στιγμιότυπο του `Comparator` (`StringDescending`) χρησιμοποιείται για να συγκρίνει δυο συμβολοσειρές σε φθίνουσα λεξικογραφική διάταξη (π.χ. "a" > "b").

Λύση:

```
public class StringAscending implements Comparator<String> {
  public int compare(String s1, String s2) {
     return s1.compareTo(s2);
  }
}
```

```
public class StringDescending implements Comparator<String> {
  public int compare(String s1, String s2) {
     return s2.compareTo(s1);
  }
}
```

Άσκηση 5 (12 μονάδες) Σαρωτές Συλλογών Αντικειμένων

Οι επαναλήπτες (Iterators) στη Java χρησιμοποιούνται για να σαρώνουν τα στοιχεία μιας συλλογής αντικειμένων ενθυλακώνοντας τις λεπτομέρειες της υλοποίησης της. Σε αυτή την άσκηση θα πρέπει να υλοποιήσετε την κλάση `IterDouble`, στιγμιότυπα της οποίας είναι σύνθετοι επαναλήπτες (iterator transformers) δηλ. επαναλήπτες που χτίζονται πάνω σε άλλους. Πιο συγκεκριμένα, ο κατασκευαστής της `IterDouble` παίρνει σαν όρισμα έναν επαναλήπτη σε μια ακολουθία συμβολοσειρών, ενώ οι μέθοδοι `hasNext()` and `next()` βασίζονται σε αυτόν τον επαναλήπτη για να επιτελέσουν την λειτουργία τους. Η ακολουθία των συμβολοσειρών που επιστρέφει η `next()` του εξωτερικού επαναλήπτη (δηλ. της `IterDouble`) θα πρέπει να είναι η ίδια με αυτή που επιστρέφει η `next()` του φωλιασμένου επαναλήπτη συμβολοσειρών (`nestedIter`) με την διαφορά ότι κάθε στοιχείο της ακολουθίας θα εμφανίζεται δυο φορές. Για παράδειγμα εάν ο φωλιασμένος επαναλήπτης επιστρέφει `"a" "b" "c"` τότε ο εξωτερικός θα πρέπει να επιστρέφει: `"a" "a" "b" "b" "c" "c"`.

```
class IterDouble implements Iterator<String> {

      // Fields
      Iterator<String> nestedIter;
```

Λύση:
```
String cached = null;
boolean hasCached = false;
```

```
      // Constructor
      IterDouble (Iterator<String> wrapped) {
            nestedIter = wrapped;
      }


      // Iterator methods
      public boolean hasNext () {
```

Λύση:
```
            return (hasCached || nestedIter.hasNext());
```

```
      }
      public String next() {
```

Λύση:
```
            if (hasCached) {
                String s = cached;
                hasCached = false;
                cached = null;
                return s;
            } else {
                cached = nestedIter.next();
                hasCached = true;
                return cached;
            }
```

```
      }
      public void remove() {
            throw new UnsupportedOperationException();
      }
}
```

Άσκηση 6 (26 μονάδες) Γενικοί Τύποι στην Java

Μια μεταφορική εταιρία αντιμετωπίζει το εξής πρόβλημα: ένας πελάτης θέλει να μεταφέρει με πλοίο μερικά φορτία (containers) από την Κίνα στην Ελλάδα. Δυστυχώς το συνολικό βάρος των φορτίων δεν μπορεί να μεταφερθεί με ένα μόνο πλοίο. Πριν ο πελάτης πάρει την τελική του απόφαση σχετικά με ποια εμπορεύματα θα μεταφερθούν και ποια όχι, ζητά από την μεταφορική εταιρία την λίστα όλων των δυνατών συνδυασμών φορτίων. Η μεταφορική εταιρία εκτιμά ότι το πρόβλημα αυτό μπορεί να εμφανιστεί και σε άλλους πελάτες και έτσι ενδιαφέρεται για μια γενική λύση με την μορφή ενός προγράμματος Java που υπολογίζει τους δυνατούς συνδυασμούς ενός συνόλου στοιχείων. Για παράδειγμα, εάν έχουμε τρία φορτία με βάρη {5, 8, 9}, το πρόγραμμα θα πρέπει να υπολογίζει το ακόλουθο σύνολο υποσυνόλων (δηλ. το δυναμοσύνολο): {{}, {5}, {8}, {9}, {5, 8}, {5, 9}, {8, 9}, {5, 8, 9}}. Φυσικά δεν μας ενδιαφέρει η σειρά με την οποία εμφανίζονται τα στοιχεία. Σας δίνεται στην συνέχεια ένας αλγόριθμος που υπολογίζει το δυναμοσύνολο ενός συνόλου:

1. Δημιουργούμε το κενό σύνολο και εισάγουμε το κενό σύνολο σαν πρώτο στοιχείο.
2. Στην συνέχεια σαρώνουμε όλα τα στοιχεία του αρχικού συνόλου και
   (ι) δημιουργούμε ένα αντίγραφο του δυναμοσυνόλου που έχουμε υπολογίσει μέχρι τώρα και εισάγουμε το τρέχον στοιχείο που σαρώνουμε σε όλα τα υποσύνολα που περιέχει το δυναμοσύνολο.
   (ιι) και στο τέλος συγχωνεύουμε το τροποποιημένο δυναμοσύνολο με αυτό που υπολογίσαμε στο προηγούμενο βήμα.

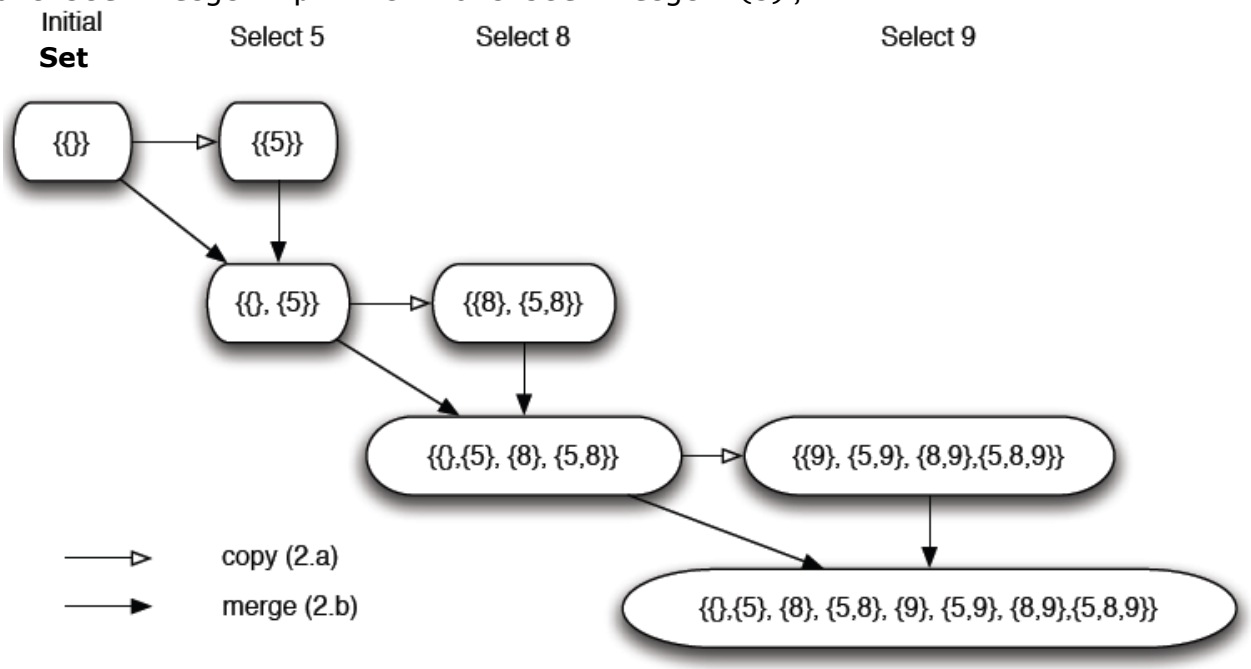
Στο παράδειγμα της επόμενης σελίδας μπορείτε να δείτε τα βήματα του αλγορίθμου που χρειάζονται για να τον υπολογισμό του δυναμοσυνόλου του συνόλου {5, 8, 9}.

Υλοποιήστε τον παραπάνω αλγόριθμο στην κλάση `PowerSet<A>` όπου `A` είναι ο τύπος των αντικειμένων του αρχικού συνόλου για το οποίο θέλουμε να υπολογίσουμε το δυναμοσύνολο του. Στο προηγούμενο παράδειγμα ενδιαφερόμαστε για το `PowerSet<Integer>`. Η κλάση `PowerSet<A>` θα πρέπει να υλοποιεί την ακόλουθη διεπαφή:

```
interface POWERSET<A> {
      public Iterator<Set<A>> iterator ();
}
```

και να υποστηρίζει μια μέθοδο κατασκευής έτσι ώστε να μπορούμε να γράψουμε:

```
Set<Integer> s = new HashSet<Integer>();
s.add (5); s.add (8); s.add (9);
PowerSet<Integer> p = new PowerSet<Integer>(s);
```



α) **(21 μονάδες)** Συμπληρώστε τα κενά στον παρακάτω κώδικα

```
class PowerSet<1> implements POWERSET<A> {
  // PowerSet Field.
  Set<2> p = new HashSet<2>();
  // The constructor.
  PowerSet(Set<A> s) {
    3
  }
```

```
  // The iterator.
  public Iterator <Set<A>> iterator () {
   4
  }
}
```

Βοήθεια: Στην θέση **2** θα πρέπει να ορίσετε τον τύπο των αντικειμένων που αποθηκεύει τα στοιχεία του δυναμοσύνολου (δηλ. τα υποσύνολα του αρχικού συνόλου). Στην θέση **3** θα πρέπει να υλοποιήστε τον κατασκευαστή της κλάσης. Αποτελεί το πιο δύσκολο κομμάτι τις άσκησης και θα πρέπει να προσέξετε ιδιαίτερα τις μορφοποιήσεις τύπων (type casts). Σας συνιστάται να υλοποιήστε πρώτα την μέθοδο χωρίς παραμέτρους τύπων και στην συνέχεια να προσθέστε τους γενικούς τύπους που χρειάζεστε.

Λύση:

```
(3 points)  1 = <A>
(3 points)  2 = HashSet<A> (ή Set<A>)
(12 points) 3 =
p.add(new HashSet<A>());//powerset is initialized with the empty set
for(Iterator<A> i=s.iterator(); i.hasNext();) {//scan initial set
  A x = (A) i.next ();
   Set<Set<A>> q = new HashSet<Set<A>>();//new powerset under
construction
   for(Iterator<Set<A>> j=p.iterator(); j.hasNext();) {
     Set<A> y = j.next();
     Set<A> z = new HashSet<A>();//new powerset element under
construction
     for(Iterator<A> k=y.iterator(); k.hasNext ();) {
       z.add(k.next());
     }
     z.add(x);//insert the current element of the initial set
     q.add(z);//insert the current subset to the powerset under
construction
   }
   p.addAll(q);
}
(3 points) 4 = return p.iterator();
```

β) **(5 μονάδες)** Ποια είναι η πολυπλοκότητα χείριστου χρόνου εκτέλεσης της μεθόδου που έχετε υλοποιήσει; Χρησιμοποιήστε τον ασυμπτωτικό συμβολισμό O() και δικαιολογήστε την απάντησή σας. Για ευκολία αγνοείστε την πολυπλοκότητα των μεθόδων του Πλαισίου Συλλογών Αντικειμένων της Java που χρησιμοποιήστε και εστιάστε στην πολυπλοκότητα του αλγορίθμου που έχετε υλοποιήσει.

Λύση:

O(2^n): There 2^n are elements in the powerset, thus the complexity of the algorithm is O(2^n)

Άσκηση 7 (9 μονάδες) Θεωρία Αντικειμενοστρεφούς Προγραμματισμού

(α) (3 μονάδες) Εξηγήστε σύντομα το αποτέλεσμα κάθε ενός από τους παρακάτω προσδιοριστές (modifiers) στην δήλωση μιας μεταβλητής δεδομένων (field variable):

- `static`

Λύση:

Declares the field as a class variable rather than an instance variable. Only one copy of the field is created and it is shared by all instances.

- `protected`

Λύση:

Modifies the visibility of the field so that only classes of the same package or sub-classes may use the field.

- `final`

Λύση:
Declares the field as a constant value.

(β) **(3 μονάδες)** Ποια είναι η διαφορά μεταξύ της έννοιας του **υπο-τύπου** (sub-type) και της **υπο-κλάσης** (sub-class); Δώστε ένα απλό παράδειγμα.

Λύση:
A sub-type is a design issue while a sub-class is an implementation issue. **A** is a sub-class of **B** if **A** extends **B**. **A** is a sub-type of **B** if any instance of **A** can be substituted for **B**. Not every sub-type is a sub-class, and in java you can implement sub-types using interfaces which are not sub-classes.

(γ) **(3 μονάδες)** Ποια είναι η διαφορά μεταξύ των **στατικών τύπων** (static types) αντικειμένων και της **δυναμικής δέσμευσης** (dynamic binding) των μεθόδων; Δώστε ένα απλό παράδειγμα.

Λύση:
Resolving the types of objects is performed in JAVA at compile-time, thus the static typing. But when attempting to resolve (bind) which method to call, this is determined at run-time and depends on the type of the object the method is being executed upon. Thus we say JAVA supports dynamic binding.

Interface Map<K,V>

`void clear()`	Removes all of the mappings from this map (optional operation).
`boolean containsKey(Object key)`	Returns true if this map contains a mapping for the specified key.
`boolean containsValue(Object value)`	Returns true if this map maps one or more keys to the specified value.
`Set<Map.Entry<K,V>> entrySet()`	Returns a Set view of the mappings contained in this map.
`boolean equals(Object o)`	Compares the specified object with this map for equality.
`V get(Object key)`	Returns the value to which the specified key is mapped, or null if this map contains no mapping for the key.
`int hashCode()`	Returns the hash code value for this map.
`boolean isEmpty()`	Returns true if this map contains no key-value mappings.
`Set<K> keySet()`	Returns a Set view of the keys contained in this map.
`V put(K key, V value)`	Associates the specified value with the specified key in this map (optional operation).
`void putAll(Map<? extends K,? extends V> m)`	Copies all of the mappings from the specified map to this map (optional operation).
`V remove(Object key)`	Removes the mapping for a key from this map if it is present (optional operation).
`int size()`	Returns the number of key-value mappings in this map.
`Collection<V> values()`	Returns a Collection view of the values contained in this map.

Interface Set<E>

`boolean add(E e)`	Adds the specified element to this set if it is not already present (optional operation).
`boolean addAll(Collection<? extends E> c)`	Adds all of the elements in the specified collection to this set if they're not already present (optional operation).
`void clear()`	Removes all of the elements from this set (optional operation).
`boolean contains(Object o)`	Returns true if this set contains the specified element.
`boolean containsAll(Collection<?> c)`	Returns true if this set contains all of the elements of the specified collection.
`boolean equals(Object o)`	Compares the specified object with this set for equality.
`int hashCode()`	Returns the hash code value for this set.
`boolean isEmpty()`	Returns true if this set contains no elements.
`Iterator<E> iterator()`	Returns an iterator over the elements in this set.
`boolean remove(Object o)`	Removes the specified element from this set if it is present (optional operation).
`boolean removeAll(Collection<?> c)`	Removes from this set all of its elements that are contained in the specified collection (optional operation).
`boolean retainAll(Collection<?> c)`	Retains only the elements in this set that are contained in the specified collection (optional operation).
`int size()`	Returns the number of elements in this set (its cardinality).
`Object[] toArray()`	Returns an array containing all of the elements in this set.
`<T> T[] toArray(T[] a)`	Returns an array containing all of the elements in this set; the runtime type of the returned array is that of the specified array.