**Πανεπιστήμιο Κρήτης**
**Τμήμα Επιστήμης Υπολογιστών**

# ΗΥ-252 – Αντικειμενοστρεφής Προγραμματισμός
# Βασίλης Χριστοφίδης

## Επαναληπτική Εξέταση (3 ώρες)
### Ημερομηνία: 29 Αυγούστου 2011

| Θέμα 1 | Θέμα 2 | Θέμα 3 | Θέμα 4 | Θέμα 5 | Θέμα 6 | Θέμα 7 | Σύνολο |
|---|---|---|---|---|---|---|---|
| /15 | /10 | /20 | /20 | /10 | /20 | /10 | /105 |

**Ονοματεπώνυμο:**
**Αριθμός Μητρώου:**

**Άσκηση 1 (15 μονάδες) // Κατανόηση κώδικα Java**
Υποδείξτε την σωστή απάντηση στα παρακάτω υποερωτήματα από την λίστα των απαντήσεων που σας δίνεται. Κάθε σωστή απάντηση βαθμολογείται με πέντε (5) μονάδες, μια ερώτηση χωρίς απάντηση παίρνει μηδέν (0) μονάδες ενώ για κάθε λάθος απάντηση επιβαρύνεστε με δύο (2) μονάδες.

α) Ποια είναι η τιμή της στατικής μεταβλητής x μετά από την δημιουργία ενός δεύτερου αντικειμένου της κλάσης XX;

```
class XX {
     static { x+=2; }
     static int x=-1;
     public XX() { x*=3; }
}
```

A. 3
B. 18
C. -9
D. 9
E. Κανένα από τα παραπάνω

**Λύση:**
D. 9

β) Τι θα τυπωθεί στην στάνταρ έξοδο μετά την εκτέλεση του παρακάτω προγράμματος Java;

```
class X {
     int xpto() {return 5;}
}
class Y extends X {
     int xpto(){return 10;}
     void test(){
          X x = (X) this;
          System.out.print(this.xpto());
          System.out.print(x.xpto());
          System.out.print(((X)this).xpto());
          System.out.print(super.xpto());
     }
     public static void main(String[] args){
```

```
            new Y().test();
      }
}
```
A. 5 5 5 10
B. 10 10 10 5
C. 10 10 5 5
D. 10 5 5 10
E. Κανένα από τα παραπάνω

**Λύση:**
B. 10 10 10 5

γ) Τι θα τυπωθεί στην στάνταρ έξοδο μετά την κλήση της μεθόδου `foo()`;

```
public static void foo(){
      try {
            System.out.print(1);
            return;
      } catch (Throwable e) {
            System.out.print(2);
      } finally {
           System.out.print(3);
      }
}
```
A. 12
B. 123
C. 13
D. 1
E. Κανένα από τα παραπάνω

**Λύση:**
C. 13

**Άσκηση 2 (7+3 μονάδες) //Χειρισμός Εισόδου/Εξόδου& Συμβολοσειρών**

Συμπληρώστε την υλοποίηση της παρακάτω μεθόδου η οποία διαβάζει μια ροή εισόδου `BufferedReader` (ή εναλλακτικά `Scanner`) και εκτυπώνει στην στάνταρ έξοδο όλες τις γραμμές του αρχείου εισόδου που αρχίζουν με μια δοσμένη συμβολοσειρά. Για παράδειγμα, εάν το αρχείο εισόδου περιέχει τις παρακάτω γραμμές:

```
GOOD: Ice cream
BAD: Flat tire
GOOD: Winning the lottery is NOT BAD
BAD Out of gas
Bad is not the same as BAD
```

Η εκτέλεση της μεθόδου `printLinesStartingWith(f,"BAD")` θα εκτυπώσει:

```
BAD: Flat tire
BAD Out of gas
```

Η υλοποίηση σας θα πρέπει να βασιστεί στην αναζήτηση και τον χειρισμό συμβολοσειρών που προσφέρει η κλάση `String`. Μπορείτε να χρησιμοποιήσετε μεθόδους σαν την `startsWith()`, `substring()` και `indexOf()`. Ένας απλός βρόγχος που συγκρίνει ατομικούς χαρακτήρες δεν συνιστά πλήρη απάντηση. Ο σωστός χειρισμός των εξαιρέσεων θα βαθμολογηθεί με τρεις (3) επιπλέον μονάδες.

```
/** Print on System.out all of the lines from file in
 * that begin with str.
 * @param in the input file
 * @param str the string we are looking for at the start
 * of each input line
 */
void printLinesStartingWith(BufferedReader in, String
str) {
```

**Λύση:**

```
try {
     String line = in.readLine();
     while (line != null) {
       if (line.startsWith(str)) {
           System.out.println(line);
       }
       line = in.readLine();
     }
} catch (IOException e) {}
```

Two other ways to check for whether `str` appears at the beginning of the line is

```
1) if (line.indexOf(str) == 0)
2) if (line.length() >= str.length() &&
line.substring(0, str.length()).equals(str)) ...
```

**Note**: in real code it is a bad idea to catch an exception and do nothing, as in the above code. For the test it's enough that you demonstrated that you understood how to use a try-catch block to handle an exception.

```
}
```

**Άσκηση 3 (20 μονάδες) // Χειρισμός Πινάκων**

Υλοποιήστε μια μέθοδο `double getAverageDropFourLowest (double [] grade)` που υπολογίζει τον μέσο όρο ενός πίνακα (array) βαθμών, αφού πρώτα απαλειφούν οι τέσσερεις (4) μικρότερες βαθμολογίες. Για κάτι τέτοιο, θα πρέπει επίσης να υλοποιήσετε μια βοηθητική μέθοδο `double[] dropMin(double [] grade)` η οποία υπολογίζει και διαγράφει την ελάχιστη τιμή του πίνακα που δίνεται σαν όρισμα. Για παράδειγμα, μετά την ακόλουθη κλήση της μεθόδου ο επιστρεφόμενος μέσος όρος θα είναι `90.0`:

```
double[] grade = {100,0,0,0,5,80,90};
grade = getAverageDropFourLowest(grade);
```

**Σημείωση**: Υποθέστε ότι ο πίνακας που δίνουμε σαν όρισμα της μεθόδου έχει τουλάχιστον πέντε βαθμούς (`grades.length >= 5`).

**Λύση:**

```
  /** Returns the average after dropping the lowest
   *  4 grades. */
  public double getAverageDropFourLowest
                                  (double [] grades){
    grades =dropMin(grades);
    grades =dropMin(grades);
    grades =dropMin(grades);
    grades =dropMin(grades);
```

```
    double sum = 0;
    for (double i : grades){
      sum += i;
    }    return sum / grades.length;
    } //  8 points
  /**Returns an array identical to grades but
   * without the lowest number in grades dropped.
   * if there are more than one minimum we only drop
   * the first one.
   * @param grades the array of grades
   * @return a new array of size 1 less than grades.
   */

  public double[] dropMin(double[] grade) {
    double minVal = grade[0];
    int minValIndex = 0;
    for (int i = 1; i < grade.length; i++) {
      if (grade[i] < minVal) {
        minVal = grade[i];
        minValIndex = i;
      }
    }
    double[] result = new double[grade.length - 1]
    // copy values until minValIndex
    for (int i = 0; i < minValIndex; i++) {
      result[i] = grade[i];
    }
    // copy values after minValIndex
    for (int i=minValIndex + 1; i < grade.length
i++) {
      result[i-1] = grade[i];
    }
    return result;
  } // 12 points
```

**Άσκηση 4 (20 μονάδες) Χειρισμός Πλαισίου Συλλογών Αντικειμένων**

Για κάθε μια από τις παρακάτω περιπτώσεις, γράψτε τον ακριβή τύπο της πιο αποδοτικής συλλογής δεδομένων που χρειαζόμαστε για την αποθήκευση των δεδομένων. Μπορείτε να χρησιμοποιήσετε οποιοδήποτε συνδυασμό των γενικών εκδόσεων (generic) των διεπαφών συλλογών `Map`, `Set`, και `List`. Χρησιμοποιήστε τους γενικούς τύπους που ταιριάζουν περισσότερο στα δεδομένα (π.χ. `Room`, `Card`, `Phone`, κλπ.) του κάθε ερωτήματος (π.χ. `List<Artist>`) καθώς και φωλιασμένους (nested) τύπους συλλογών (π.χ. `List<Set<String>>`).
Μπορείτε να υποθέσετε ότι όλες οι κλάσεις που αναπαριστούν τα δεδομένα (π.χ. `Room`, `Card`, `Phone`, κλπ.) κάθε ερωτήματος είναι ήδη ορισμένες και ότι όλες υλοποιούν κατάλληλα τις μεθόδους `equals()` and `hashCode()` ώστε τα αντικείμενά τους να αποθηκεύονται σε σύνολα (`Set`) ή κλειδιά (key) απεικονίσεων (`Map`).

Κάθε σωστή απάντηση βαθμολογείται με πέντε (5) μονάδες, μια ερώτηση χωρίς απάντηση παίρνει μηδέν (0) μονάδες ενώ για κάθε λάθος απάντηση επιβαρύνεστε με δύο (2) μονάδες.

α) Σε ένα ηλεκτρονικό παιχνίδι δράσης θέλουμε να αναπαραστήσουμε τα δωμάτια (Room) στα οποία οδηγεί κάθε δυνατή κατεύθυνση (που έχει τη μορφή συμβολοσειράς). (Διευκρίνιση: Κάθε κατεύθυνση οδηγεί σε ένα το πολύ δωμάτιο)

i. Map<Room, String>
ii. Map<String, Set<Room>>
iii.Map<String, Room>
iv.Set<String>

**Λύση:**
iii

β) Σε ένα ηλεκτρονικό παιχνίδι πασιέντζας χρησιμοποιούμε δεκατρείς (13) στοίβες τραπουλόχαρτων (Card), κάθε μία από τις οποίες έχει τέσσερα (4) τραπουλόχαρτα. Η διάταξη των στοιβών καθώς και των τραπουλόχαρτων σε κάθε στοίβα είναι σημαντική για το παιχνίδι.

i. Set<Set<Card>>
ii. Map<Card, List<Card>>
iii.List<Card>
iv.List<List<Card>>

**Λύση:**
iv

γ) Σε ένα ηλεκτρονικό βιβλίο διευθύνσεων μας ενδιαφέρει να βρίσκουμε γρήγορα τούς αριθμούς τηλεφώνων καθώς και τις αντίστοιχες διευθύνσεις (String) δίνοντας το ονοματεπώνυμο του κατόχου τους (String).

i. Map<Set<String>, Phone>
ii. Map<String, Map<String, Phone>
iii. Map<Phone, Map<String, String>>
iv. Map<String, Map<Phone, String>>

**Λύση:**
iv

δ) Η αναζήτηση του ονόματος ενός μουσικού κομματιού με βάση την διάρκεια της εγγραφής του στον οπτικό δίσκο (CD) είναι μια λειτουργία που χρησιμοποιείται ευρέως από μουσικές εφαρμογές (πχ. iTunes). Μοιάζει παράξενο αλλά η γνώση της διάρκειας εγγραφής μαζί με την σειρά αποθήκευσης (πχ. <147 sec, 2>) όλων των κομματιών είναι αρκετή για την ταυτοποίηση σχεδόν οποιοιδήποτε CD! Δεδομένων μάλιστα των διαφορετικών ονομάτων του ίδιου κομματιού (πχ. για την εγχώρια ή διεθνή αγορά) ορισμένα CD καταγράφονται με πολλαπλά ονόματα. Υποδείξτε την δομή εκείνη που για κάθε CD διατηρεί τον χρόνο και τη σειρά εκτέλεσης των τραγουδιών, καθώς και το σύνολο των διαφορετικών ονομάτων που μπορεί να φέρει ένα τραγούδι, ενώ παράλληλα εξυπηρετεί τις ανάγκες αναζήτησης.

i. Map<Integer, List<List<String>>>
ii. Map<List<Integer>, Set<String>>
iii. List<Map<Integer, String>>
iv. Map<Integer, Map<Integer, String>>

**Λύση:**
ii

**Άσκηση 5 (10 μονάδες) // Βασική λειτουργικότητα Αντικειμένων**

Σας δίνεται η ακόλουθη (μη ολοκληρωμένη) δήλωση της κλάσης `Record`:

```
public class Record implements Comparable<Record> {
      int i;
      String s;
      java.util.Date d;
      long l;
      public int compareTo(Record other) {
      ... }
}
```

Υλοποιήστε την μέθοδο σύγκρισης `int compareTo()` της κλάσης έτσι ώστε αντικείμενα τύπου `Record` να μπορούν να χρησιμοποιηθούν σε διατεταγμένες συλλογές δεδομένων στην Java (πχ. `TreeSet`). Η επιθυμητή διάταξη θα πρέπει να είναι αύξουσα λεξικογραφική με βάση την σειρά ορισμού των πεδίων της κλάσης (δηλ. πρώτα το `i`, μετά το `s`, μετά το `d`, και τέλος το `l`).

**Σημείωση**: Θυμηθείτε ότι η μέθοδος `compareTo(other)` πρέπει να επιστρέφει έναν αρνητικό ακέραιο, μηδέν, ή θετικό ακέραιο εάν το αντικείμενο `this` είναι μικρότερο, ίσο ή μεγαλύτερο από το `other` αντίστοιχα.

**Λύση:**

```
public int compareTo(Record r) {
  if (i < r.i) return -1;
  if (i > r.i) return 1;
  int strCompare = s.compareTo(r.s);
  if (strCompare != 0) return strCompare;
  int dateCompare = d.compareTo(r.d);
  if (dateCompare != 0) return dateCompare;
  if (l < r.l) return -1;
  if (l > r.l) return 1;
  return 0;
}
```

**Άσκηση 6 (20 μονάδες) // Αφαιρετικοί Τύποι Δεδομένων**

Μας ενδιαφέρει ένας Αφαιρετικός Τύπος Δεδομένων (ΑΤΔ) που αναπαριστά Δέντρα των οποίων οι εσωτερικοί κόμβοι έχουν σαν ετικέτα (label) έναν ακέραιο αριθμό ενώ τα παιδιά τους συνιστούν μια λίστα υπο-δέντρων. Ως συνήθως, ένα φύλο του δέντρου είναι ένας κόμβος χωρίς παιδιά (δηλ. μόνο με μια ετικέτα). Στην συνέχεια σας δίνεται η προδιαγραφή του ΑΤΔ Δέντρο (*Tree*) καθώς και του ΑΤΔ Λίστα (*List*) ακεραίων και Δενδρικός Κόμβος (*TreeNode*) στους οποίους βασίζεται.

Syntax:

```
ADT List
createlist: -> List //create an empty list
add : List, Int -> List //add a new integer in the list

ADT Tree
createTree : Int, TreeNode -> Tree //create an elementary
                   // tree with a integer label and a Tree
                   //node (i.e. its list of sub-trees)
getLeaves : Tree -> List  //get the integer labels of
                          //all Tree leaves in a new list
```

```
addLeaves : Tree, List -> List //add the integer labels
                              //of all Tree leaves to the
                              //list given as argument
loop : TreeNode, List -> List //get the integer labels of
                              //the sub-trees of a Tree Node
                              //to the list given as argument
treeNode : Tree, TreeNode -> TreeNode //get the sub-trees
                    //of a Tree to the Tree Node given as
                    //argument (i.e. a list of sub-trees)
ADT TreeNode
emptynode: -> TreeNode//create an empty list of sub-trees
```
Semantics:
```
var T : Tree
var I : Int
var L : List
var N : TreeNode
getLeaves(T) = addLeaves(T, emptylist)
addLeaves(createTree(I, emptynode), L) = add(L, I)
addLeaves(createTree(I, N), L)=loop(N, L) if N!=emptynode
loop(emptynode, L) = L
loop(treeNode(T, N), L) = loop(N, addLeaves(T, L))
```

(α) **(13 μονάδες)** Δώστε τις υλοποιητικές κλάσεις Java των ΑΤΔ Δέντρο (*Tree*) και Δενδρικός Κόμβος (*TreeNode*) χρησιμοποιώντας ένα διάνυσμα ακεραίων (`ArrayList<Integer>`) για την υλοποίηση της λίστας των ετικετών ενός κόμβου και μιας συνδεδεμένης λίστας για την υλοποίηση της λίστας των υπο-δένδρων ενός δενδρικού κόμβου. Εκτός από τις μεθόδους κατασκευής των αντικειμένων των δύο κλάσεων, υλοποιήστε την μέθοδο `getLeaves()` και την βοηθητική της `addLeaves()`. Δεν απαιτείται η υλοποίηση των υπολοίπων μεθόδων της κλάσης *Tree*.

**Λύση:**
```
import java.util.ArrayList;
/**
 * Trees with integer internal labels and lists of
 * subtrees. The {@link TreeNode list of subtrees}
 * is implemented as a linked list.
 *@author <ahref="mailto:christop@csd.uoc.gr">VC</a>
*/
public class Tree {
// attributes: 2 point
private int value; //The internal label of the tree.
private TreeNode subtrees; //A linked list of trees.
/**
  * Creates a new <code>Tree</code> instance.
  * @param i an <code>int</code> value
  * @param n a <code>TreeNode</code> value
  */
  public Tree(int i, TreeNode n) {
     value = i;
     subtrees = n;
  } // 1 point
/**
  * Return a list of all the leaves in the tree.
  * A leaf is a tree with an empty list of subtrees.
  * @return an ArrayList containing all the leaves
```

```
   * of the tree
   */
  public ArrayList<Integer> getLeaves() {
     // add all the leaves to the empty list
     return addLeaves(new ArrayList<Integer>());
  } // 2 points
/**
  * Add all the leaves of the tree to a given list
  * @param v the ArrayList to add the leaves to
  * @return the ArrayList with all the leaves added
  */
private ArrayList<Integer>
                    addLeaves(ArrayList<Integer> v) {
     // add all leaves to the ArrayList
     if (subtrees == null) {
// then this tree is a leaf: add the internal label
        v.addElement(value);
     }
// otherwise, get the leaves from all the subtrees
     else {
     //used to traverse the linked list of subtrees
       TreeNode n = subtrees;
       while(n!=null){//not at the end of the list
       // so add the leaves from the current node
       n.theTree.addLeaves(v);
       // move on to the next node
       n = n.next;
     }
   }
   // all leaves have been added to v
   return v;
  }
} // 5 points
/**
* Minimal linked-list implementation of TreeNode. As
* a linked list, a TreeNode is a node, whose fields
store:
*<ul>
*<li>a Tree (the "value" at the node), and the </li>
*<li> next TreeNode (a pointer to the next node in
the list). </li>
*</ul>
*/
class TreeNode {
  // attributes: 2 point
  private Tree theTree;//The tree at this node.
  private TreeNode next; // The next node.
/**
  * Constructor:
* tree to store at current node, and the remainder
* of the list.
  */
  TreeNode(Tree t, TreeNode n) {
     theTree = t;
     next = n;
  } // 1 points
}
```

(β) **(7 μονάδες)** Σχολιάστε σύντομα γιατί η υλοποίηση της `addLeaves()` που δώσατε στο προηγούμενο ερώτημα είναι σύμφωνη με την σημασιολογία της μεθόδου που προβλέπεται στην προδιαγραφή του ΑΤΔ *Tree*.

**Λύση:**
The if-clause in `addLeaves()` adds the internal label to the list if the Tree is a leaf (the subtrees field is null), which corresponds to the equation:
`addLeaves(createTree(I, emptynode), L) = add(L, I)`
The else-clause implements the 'loop' operation:
```
var N : TreeNode
addLeaves(createTree(I, N), L) = loop(N, L) if
N!=emptynode
```
While the variable `N` is not null, the leaves of `f.theTree` are added to the ArrayList, corresponding to the equation
`loop(treeNode(T, N), L) = loop(N, addLeaves(T, L))`
Finally, when `N` is null, the ArrayList is returned, corresponding to the equation
`loop(emptynode, L) = L`

**Άσκηση 7 (10 μονάδες) // Θεωρία Αντικειμενοστρεφούς Προγραμματισμού**

(α) **(5 μονάδες)** Τι είναι μια αφαιρετική κλάση (abstract class) και που είναι χρήσιμη;

**Λύση:**
An abstract class is a class where some of the methods are declared abstract, with no method bodies. A subclass of an abstract class can fill in some of these abstract methods with concrete method bodies; if it fills in all of the abstract methods, then it is a concrete class and it can be instantiated using new. Abstract classes are used to declare methods that are common to a whole group of concrete classes but implemented differently by each one (this is similar to how interfaces are used) while also providing implementations of some methods that are shared among this whole group of classes.

β) **(5 μονάδες)** Για ποιους λόγους θα χρησιμοποιούσατε μια διεπαφή Java (interface) στην σχεδίαση μιας ιεραρχίας κλάσεων;

**Λύση:**
In general, the base class in a class hierarchy has two main purposes. First, it defines the common interface for using the set of classes. Second, because subclasses inherit methods, the base class is the proper place to implement code that will be common to all subclasses. Although the base class does specify the interface, it also includes implementation information that is of no interest to clients. The usual practice is to use a Java Interface to specify the interface, and then have the base class implement that interface. Client code is typically written to the Java Interface. This not only simplifies the task of writing client code, but the resulting code is more flexible because it will also work with different implementations of the Interface.

# Class ArrayList

| Constructor Summary |
| --- |
| **ArrayList**() Constructs an empty list with an initial capacity of ten. |
| **ArrayList**(Collection<? extends E> c) Constructs a list containing the elements of the specified collection, in the order they are returned by the collection's iterator. |
| **ArrayList**(int initialCapacity) Constructs an empty list with the specified initial capacity. |

| Method Summary | |
| --- | --- |
| boolean | **add**(E e) Appends the specified element to the end of this list. |
| void | **add**(int index, E element) Inserts the specified element at the specified position in this list. |
| boolean | **addAll**(Collection<? extends E> c) Appends all of the elements in the specified collection to the end of this list, in the order that they are returned by the specified collection's Iterator. |
| boolean | **addAll**(int index, Collection<? extends E> c) Inserts all of the elements in the specified collection into this list, starting at the specified position. |
| void | **clear**() Removes all of the elements from this list. |
| Object | **clone**() Returns a shallow copy of this ArrayList instance. |
| boolean | **contains**(Object o) Returns true if this list contains the specified element. |
| void | **ensureCapacity**(int minCapacity) Increases the capacity of this ArrayList instance, if necessary, to ensure that it can hold at least the number of elements specified by the minimum capacity argument. |
| E | **get**(int index) Returns the element at the specified position in this list. |
| int | **indexOf**(Object o) Returns the index of the first occurrence of the specified element in this list, or -1 if this list does not contain the element. |
| boolean | **isEmpty**() Returns true if this list contains no elements. |
| int | **lastIndexOf**(Object o) Returns the index of the last occurrence of the specified element in this list, or -1 if this list does not contain the element. |
| E | **remove**(int index) Removes the element at the specified position in this list. |
| boolean | **remove**(Object o) Removes the first occurrence of the specified element from this list, if it is present. |
| protected void | **removeRange**(int fromIndex, int toIndex) Removes from this list all of the elements whose index is between fromIndex, inclusive, and toIndex, exclusive. |
| E | **set**(int index, E element) Replaces the element at the specified position in this list with the specified element. |
| int | **size**() Returns the number of elements in this list. |
| Object[] | **toArray**() Returns an array containing all of the elements in this list in proper sequence (from first to last element). |
| <T> T[] | **toArray**(T[] a) Returns an array containing all of the elements in this list in proper sequence (from first to last element); the runtime type of the returned array is that of the specified array. |
| void | **trimToSize**() Trims the capacity of this ArrayList instance to be the list's current size. |