Πανεπιστήμιο Κρήτης
Τμήμα Επιστήμης Υπολογιστών

ΗΥ-252 – Οντοκεντρικός Προγραμματισμός
Βασίλης Χριστοφίδης

Επαναληπτική Εξέταση (3 ώρες)
Ημερομηνία: 9 Σεπτεμβρίου 2004

Ονοματεπώνυμο:
Αριθμός Μητρώου:

Άσκηση 1 (17 μονάδες)

Θεωρήστε τις παρακάτω δηλώσεις διεπαφών και κλάσεων Java.

```
public interface ISong {
   void setName(String n);
   String getName();
   boolean equals(ISong s);
}
public class CSong implements ISong {
   private String name;
   public CSong(String n){ name= n; }
   public void setName(String n) { name= n; }
   public String getName() { return name; }
   public boolean equals(ISong s) {
    return ((s instanceof CSong) && name.equals(s.name));
   }
}
public class TimedSong extends CSong implements ISong {
   private double duration;
   public TimedSong(String n, double d){
      super(n); duration= d; }
   public double getTime(){ return duration; }
   public boolean equals(ISong s) {
    return ( (s instanceof TimedSong) &&
             name.equals(s.name) && duration==s.duration);
   }
}
```

1.1 (3 μονάδες) Ποιες από τις παρακάτω εντολές της κυρίας μεθόδου της κλάσης **SongApp** θεωρούνται σαν λάθη από τον μεταφραστή της Java (compile-time errors); Υπογραμμίστε τις λάθος εντολές και δικαιολογήστε σύντομα την απάντησή σας.

```
public class SongApp {
  public static void main(String[] pars) {
    ISong s0, s1; CSong cs0, cs1; TimedSong t0, t1;
    s0= new ISong("Imagine");
    s0= new TimedSong("Voodoo Child",4.0);
    s1= new CSong("A Long Walk",4.0);
    cs0= new TimedSong("Strange Fruit",6.0);
    t0= new TimedSong("Reasons to be Beautiful",5.0);
    t1= (TimedSong) s0;
    s1= t0;
    cs0= t0;
```

```
    t1.setName(t1.getName());
    double d= cs0.getTime();
  }
```

Λύση:
```
// wrong, cannot instantiate an interface
s0= new ISong("Imagine"); // 1 mark
// wrong, CSong has no constructor with two parameters
s1= new CSong("A Long Walk",4.0); // 1 mark
// wrong, cannot assign TimedSong expression (t0) to
// CSong variable (s1)
s1= t0; // 1 mark
```

1.2. (14 μονάδες) Τι θα τυπωθεί μετά την εκτέλεση της παρακάτω κυρίας μεθόδου της κλάσης **SongApp**; Δικαιολογήστε σύντομα την απάντησή σας.

```
public static void main(String[] pars) {
   ISong s0, s1, s2;
   s0= new TimedSong("Somewhat Damaged", 4.0);
   s1= new TimedSong("Somewhat Damaged", 4.0);
   if (s0==s1) { System.out.println("equal"); }
   else { System.out.println("not equal"); }
   if (s0.equals(s1)) { System.out.println("EQUAL"); }
   else { System.out.println("not EQUAL"); }
   s2= s1;
   s2.setName("Felicidade");
   System.out.println( s1.getName() + s2.getName() );
   s0= new CSong("Felicidade");
   if (s2.equals(s0)) { System.out.println("same"); }
   else { System.out.println("not same"); }
   if (s0.equals(s2)) { System.out.println("SAME"); }
   else { System.out.println("not SAME"); }
  }
```

Λύση:
```
public static void main(String[] pars) {
  ISong s0, s1, s2;
  s0= new TimedSong("Somewhat Damaged", 4.0); // 1 mark
  s1= new TimedSong("Somewhat Damaged", 4.0); // 1 mark
  // {s0 and s1 reference distinct TimedSong objects with
  // the same duration and name values}
  if (s0==s1) { System.out.println("equal"); }// 1 mark
  else { System.out.println("not equal"); }   // 1 mark
  // "not equal" has been printed, because the ==
  // condition is false
  if (s0.equals(s1)) { System.out.println("EQUAL"); } //
1 mark
  else { System.out.println("not EQUAL"); } // 1 mark
  // "EQUAL" has been printed, because TimedSong.equals
  // compares fields
  s2= s1;
  // {s2 and s1 reference the same object of type
  //TimedSong}
  s2.setName("Felicidade"); // 1 mark
  // {s2 and s1 reference the same object and name field
  // == "Felicidade"}
  System.out.println( s1.getName() + s2.getName() ); // 1
mark
  // "FelicidadeFelicidade" has been printed
  s0= new CSong("Felicidade"); // 1 mark
  // {s0 references a CSong with name field "Felicidade"}
```

```
  if (s2.equals(s0)) { System.out.println("same"); } // 1
mark
  else { System.out.println("not same"); } // 1 mark
  // "not same" has been printed, because s2 has type
  // TimedSong and method TimedSong.equals requires the
  // argument (s0) to have same type
  if (s0.equals(s2)) { System.out.println("SAME"); } // 1
mark
  else { System.out.println("not SAME"); } // 1 mark
  // "SAME" has been printed, because s0 has type CSong
  // and method CSong.equals requires the argument (s2)
  // to have type Csong (which it does because TimedSong
  // is a subclass of CSong) and to have the same name
  // field
}
```

Άσκηση 2 (30 μονάδες)

Σας δίνεται το παρακάτω πρόγραμμα Java το οποίο διαμερίζει μία λίστα ακεραίων (`list`) σε υπολίστες έτσι ώστε μια από αυτές (`smaller`) να περιέχει πάντα τα k (`curk`) μικρότερα στοιχεία της αρχικής λίστας:

```
List curlist = list;
int curk = k;
while ( !curlist.isEmpty() ) {
  int pivot = curlist.at(1);
  List smaller, larger;
  for ( int topartition = 2; topartition <=
        curlist.getLength();topartition++ ) {
    if ( curlist.at(topartition) <= pivot ) {
         smaller.insert(smaller.getLength()+1,
                        curlist.at(topartition));
    } else if ( curlist.at(topartition) > pivot ) {
                larger.insert(larger.getLength()+1,
                              curlist.at(topartition));
    }
  }
  if ( curk <= smaller.getLength() ) {
    curlist = smaller;
  } else if ( curk == smaller.getLength()+1 ) {
      System.out.println(pivot);
      break;
  } else {
       curlist = larger;
       curk -= smaller.getLength()+1;
  }
}
```

2.1 (10 μονάδες) Ποια είναι η πολυπλοκότητα (complexity) του χρόνου εκτέλεσης (στην καλύτερη περίπτωση) του προγράμματος;

Λύση:
In the best case, the outer loop exits as quickly as possible i.e. it hits the break the first time through. This happens if the first element in the list (the thing chosen as the `pivot`) happens to be the `kth` smallest element. The running time is `O(n)`, because the inner loop repeats n-1 times and the outer loop repeats only once; all of the `List` methods called (at, `getLength()`, `insert()` at end) are O(1) for an array-based List.

2.2 (10 μονάδες) Ποια είναι η πολυπλοκότητα (complexity) του χρόνου εκτέλεσης (στην χειρότερη περίπτωση) του προγράμματος;

Λύση:
In the worst case, the outer loop executes until the current list is empty. Each time through the outer loop, the current list is updated to be one of the sublists smaller or larger; at most one element is not present in either of these lists (the pivot), so in the worst case, the list that replaces `curlist` is one smaller than the old list. This means that the outer loop executes with `curlist` of size `n, n-1, n-2,` ..., down to `1` (when `curlist` has size 0, the test condition is false and the loop exits). The inner loop is executed `O(curlist.size)` times each time the outer loop is executed; the rest of the work in the outer loop is `O(1)` per outer loop iteration. The total work is the sum of the work done for each iteration of the outer loop: $O(n)+ O(n-1)+ O(n-2)+ ...+ O(1) = O(n^2)$.

2.3 (10 μονάδες) Ποια είναι η πολυπλοκότητα (complexity) του χρόνου εκτέλεσης (στην μέση περίπτωση) του προγράμματος;

Λύση:
In the average case, one would expect the `pivot` chosen to be near the middle. This means that on successive iterations, `curlist` will have size `n, n/2, n/4,` ..., `1, 0`. Since `O(curlist.size)` work is done with each iteration, the total work is $n+n/2+n/4+...+1 <= 2n = O(n)$.
If this seems too optimistic, consider that with probability 1/2, the chosen pivot will lead to the larger of the "smaller" or "larger" lists containing at most 3/4 of the elements. If we end up searching the larger of the two, `curlist` will have size `n, 3n/4, 9n/16, ..., 1` on successive iterations - this sum again works out to be `O(n)`.

Άσκηση 3 (53 μονάδες)

3.1 (7 μονάδες) Θεωρήστε τον ακόλουθο ορισμό της κλάσης **`Time`** στηνJava:

```
import java.text.DecimalFormat;
public class Time implements Cloneable
{
 private int hour;
 private int minute;
 // The following method returns true if the receiver is
 // earlier than the parameter
 public bollean earlierThan(Time t)
  { /* implementation */ }
 // other methods
}
```

Υλοποιήστε τις παρακάτω μεθόδους της κλάσης **`Time`**:
(a) **(1 μονάδα)** Έναν κατασκευαστή αντικειμένων (constructor), ο οποίος αρχικοποιεί όλες τις μεταβλητές στιγμιοτύπων του μέσω κατάλληλων παραμέτρων (για ευκολία υποθέτουμε ότι τιμές όλων των παραμέτρων της μεθόδου είναι μέσα στο επιτρεπτό πεδίο τιμών του ΑΤΔ **`Time`**).

Λύση:
```
public Time(int h, int m) { hour = h; minute = m; }
```

(b) **(2 μονάδες)** Μια μέθοδο **`toString()`** η οποία μετατρέπει ένα αντικείμενο `Time` σε μια συμβολοσειρά της μορφής "09:30", "11:05" ή "23:35" (για ευκολία μπορείτε να χρησιμοποιήσετε την κλάση **`java.text.DecimalFormat`**).

Λύση:
```
You may use it, but you don't have to.
public String toString( ) // 0.5 mark
{
 DecimalFormat fmt = new DecimalFormat("00"); // 0.5 mark
 return (fmt.format(hour) + ":" +
         fmt.format(minute)); // 1 mark
}
```

(c) **(4 μονάδες)** Μια μέθοδο **`equals()`** για να ελέγχουμε την ισότητα δύο αντικειμένων `Time`, η οποία υποσκελίζει (overrides) την μέθοδο **`equals()`** που ορίζεται στην κλάση `Object`.

Λύση:
```
public boolean equals(Object ob) // 1 mark
{
 boolean toReturn = false; // 0.5 mark
 if (ob instanceof Time) // 0.5 mark
 {
  Time e = (Time)ob; // 0.5 mark
  toReturn = (hour == e.hour && minute == e.minute); // 1
mark
 }
return toReturn; // 0.5 mark
}
```

3.2 (11 μονάδες) Θεωρήστε τον ακόλουθο ορισμό της κλάσης **`Event`** στηνJava:
```
public class Event implements Cloneable
{
 private String category;
 private Time occurredAt;
 public String getCategory( ) { return category; }
 public Time getOccurredAt( ) { returns occurredAt; }
 // other methods
 // Assume a method equals is also defined in this class.
}
```

(a) **(2 μονάδες)** Υλοποιήστε έναν κατασκευαστή αντικειμένων (constructor) ο οποίος αρχικοποιεί την μεταβλητή `category` με μια παράμετρο τύπου `String` και την μεταβλητή `occurredAt` με δύο παραμέτρους τύπου `int`.

Λύση:
```
public Event(String s, int h, int m) // 0.5 mark
{
 category = s; // 0.5 mark
 occurredAt = new Time(h, m); // 1 mark
}
```

(b) **(2 μονάδες)** Υλοποιήστε έναν κατασκευαστή αντικειμένων (constructor) χωρίς παραμέτρους που αρχικοποιεί την μεταβλητή `category` στην κενή συμβολοσειρά και την μεταβλητή `occurredAt` στις τιμές `hour=minute=0` (σημειώστε ότι αυτός ο κατασκευαστής χρησιμοποιείται εμμέσως από τον κατασκευαστή του ερωτήματος a).

Λύση:
```
public Event( ) // 1 mark
{ this("", 0, 0); } // 1 mark
```

(c) **(2 μονάδες)** Υλοποιήστε μια μέθοδο **`toString()`** η οποία μετατρέπει ένα αντικείμενο `Event` σε μια συμβολοσειρά της μορφής "Διαγώνισμα, 09:30", "Μάθημα, 16:05", or "Γεύμα, 12:10".

Λύση:
```
public String toString( ) // 0.5 mark
{ return category + ", " /* 0.5 mark */ +
occurredAt.toString( ); /* 1 mark */}
```

(d) **(5 μονάδες)** Υλοποιήστε μια μέθοδο **`clone()`** η οποία δημιουργεί ένα βαθύ αντίγραφο (deep copy) ενός αντικειμένου `Event` και η οποία υποσκελίζει (overrides) την μέθοδο **`clone()`** που ορίζεται στην κλάση `Object` (σημειώστε ότι η μέθοδός σας δεν πρέπει να εγείρει καμία εξαίρεση και κατά συνέπεια ο χειρισμός της εξαίρεσης `CloneNotSupported` της μεθόδου **`clone()`** στην κλάση Object πρέπει να γίνεται τοπικά στο σώμα της μεθόδου σας τυπώνοντας ένα κατάλληλο μηνύμα στην στάνταρτ έξοδο).

Λύση:
```
public Object clone( ) // 0.5 mark
{
 Event copy;
 try // 1 mark
  {
   copy = (Event) super.clone( ); // 1 mark
   copy.occurredAt = (Time)occurredAt.clone( ); // 1 mark
  }
 catch (CloneNotSupportedException e) // 1 mark
  {System.out.println("Clone not supported."); }
   return copy; // 0.5 mark
}
```

3.3 (5 μονάδες) Θεωρήστε τον ακόλουθο ορισμό της κλάσης **`ScaledEvent`** η οποία είναι υποκλάση της **`Event`**:

```
public class ScaledEvent extends Event
{
private double importance;
// constructors and methods
}
```

(a) **(2.5 μονάδες)** Υλοποιήστε έναν κατασκευαστή αντικειμένων (constructor) ο οποίος αρχικοποιεί όλες τις μεταβλητές με παραμέτρους.

Λύση:
```
public ScaledEvent(String s, int h, int m, double d) //
0.5 mark
{
  super(s, h, m); // 1 mark
  importance = d; // 1 mark
}
```

(b) **(2.5 μονάδες)** Υλοποιήστε μια μέθοδο **`toString()`** η οποία μετατρέπει ένα αντικείμενο `ScaledEvent` σε μια συμβολοσειρά της μορφής "Διαγώνισμα, 09:30, importance = 1.00" (για ευκολία μπορείτε να χρησιμοποιήσετε την κλάση **`java.text.DecimalFormat`**).

Λύση:
```
public String toString( ) // 0.5 mark
{ return super.toString( ) /* 1 mark */ + ", importance =
" + (new DecimalFormat("0.00")).format(importance); /* 1
mark */}
```

3.4 (17 μονάδες) Θεωρήστε τον ακόλουθο ορισμό της κλάσης **EventList**:

```
import java.util.*;
public class EventList
{// storing references of Events
 private ArrayList events = new ArrayList( );
// public methods
}
```

(a) **(8 μονάδες)** Υλοποιήστε μια μέθοδο **earliestEvent()** η οποία επιστρέφει μία αναφορά σε ένα αντικείμενο Event της μεταβλητής events (τύπου ArrayList) το οποίο θα συμβεί χρονικά πρώτο (στην περίπτωση που η μεταβλητή είναι κενή η μέθοδος πρέπει να δημιουργεί μια εξαίρεση τύπου **EmptyListException**).

Λύση:
```
public Event earliestEvent( ) // 0.5 mark
throws EmptyListException // 1 mark
{
 int len = events.size( );
 Event toReturn = null; // 0.5 mark
 if (len > 0)
 {
  toReturn = (Event) events.get(0); // 1 mark
  for (int i = 1; i < len; ++i)
    {
     if (((Event)events.get(i)).getOccurredAt( ).
          earlierThan(toReturn.getOccurredAt( )))
      { toReturn = (Event)events.get(i); }
    } // 3.5 marks
 }
 else { throw new EmptyListException( ); } // 1 mark
return toReturn; // 0.5 mark
}
```

(b) **(4 μονάδες)** Υλοποιήστε μια μέθοδο **copy()** η οποία δημιουργεί ένα βαθύ αντίγραφο (deep copy) ενός αντικειμένου EventList.

Λύση:
```
public EventList copy( ) // 0.5 mark
{
 ArrayList copy = (ArrayList) events.clone( ); // 1 mark
 for (int i = 0; i < events.size( ); ++i) // 0.5 mark
     { copy.set(i, ((Event)events.get(i)).clone( )); } //
1 mark
 EventList ret = new EventList();
 ret.events = copy; // 0.5 mark
 return ret; // 0.5 mark
}
```

(c) **(5 μονάδες)** Υλοποιήστε μια μέθοδο **remove()** η οποία αφαιρεί όλα τα αντικείμενα τύπου Event της μεταβλητής events (τύπου ArrayList) για μία συγκεκριμένη κατηγορία η οποία δίνεται σαν παράμετρος εισόδου.

Λύση:
```
public void remove(String cat) // 0.5 mark
{
 Iterator it = events.iterator( ); // 1 mark
 while (it.hasNext( )) // 0.5 mark
 {
  Event current = (Event) it.next( ); // 1 mark
  if (current.getCategory( ).equals(cat)) // 1 mark
    { it.remove( ); } // 1 mark
    }
} // A for loop does not work in this method. 2 marks if
a for loop is used.
```

3.5 (13 μονάδες) Θεωρήστε τον ακόλουθο ορισμό της κλάσης `CategoryList` στηνJava η οποία χρησιμοποιείται για να αναπαραστήσει γεγονότα (events) οργανωμένα σε κατηγορίες (categories):

```
import java.util.*;
public class CategoryList
{
private HashMap categories = new HashMap( );
// public methods
}
```

Το κλειδί (key) ενός στοιχείου της μεταβλητής categories είναι το πεδίο *category* με τύπο `String` το οποίο ορίζεται σε μια κλάση Event ενώ η τιμή (value) ενός στοιχείου είναι ένα αντικείμενο τύπου `ArrayList` το οποίο περιέχει όλα τα γεγονότα (Events) για μια συγκεκριμένη κατηγορία.

(a) (**4 μονάδες**) Υλοποιήστε μια μέθοδο **`addEvent()`** στην κλάση `CategoryList` η οποία προσθέτει ένα καινούργιο γεγονός (Event) στην μεταβλητή `categories` (τύπου **`HashMap`**). Θεωρήστε τις ακόλουθες δύο περιπτώσεις: 1) η περιγραφή του γεγονότος βρίσκεται ήδη στο **`HashMap`**, 2) η περιγραφή δεν βρίσκεται στο **`HashMap`**. Η μέθοδος πρέπει να επιστρέφει ένα ενημερωμένο αντικείμενο `CategoryList`.

Λύση:
```
public CategoryList addEvent(Event e) // 0.5 mark
{
 String s = e.getCategory( ); // 0.5 mark
 if (!categories.containsKey(s))
    { categories.put (s, new ArrayList( )); } // 1 mark
 ArrayList v = (ArrayList)categories.get(s); // 1 mark
 v.add(e); // 0.5 mark
 return this; // 0.5 mark
}
```

(b) (**4 μονάδες**) Υλοποιήστε μια μέθοδο **`removeEvent()`** στην κλάση `CategoryList` η οποία αφαιρεί ένα γεγονός (Event) στην μεταβλητή `categories` (τύπου **`HashMap`**). Αν είναι το μοναδικό γεγονός μιας κατηγορίας τότε πρέπει να αφαιρεθεί και η συγκεκριμένη κατηγορία.

Λύση:
```
public void removeEvent(Event e) // 0.5 mark
{
 String cat = e.getCategory( ); // 0.5 mark
```

```
 ArrayList catList = (ArrayList) categories.get(cat); //
1 mark
 catList.remove(e); // 1 mark
 if (catList.isEmpty( ))
   { categories.remove(cat); } // 1 mark
}
```

(c) (**5 μονάδες**) Υλοποιήστε μια μέθοδο **printEvents()** στην κλάση CategoryList η οποία τυπώνει όλα τα γεγονότα (ένα σε κάθε γραμμή της στάνταρτ εξόδου) χωρίς να διατάσει τις κατηγορίες στο **HashMap**.

Λύση:

```
public void printEvents( ) // 0.5 mark
{
  Iterator it = categories.values( ).iterator( ); // 1
mark
  while (it.hasNext( )) // 0.5 mark
  {
  ArrayList current = (ArrayList) it.next( ); // 1 mark
  for (int i = 0; i < current.size( ); ++i)
      { System.out.println((Event)current.get(i)); } // 2
marks
  }
}
```