Πανεπιστήμιο Κρήτης
Τμήμα Επιστήμης Υπολογιστών

ΗΥ-252 – Οντοκεντρικός Προγραμματισμός
Βασίλης Χριστοφίδης

Τελική Εξέταση (3 ώρες)
Ημερομηνία: 8 Ιουνίου 2002

Ονοματεπώνυμο:
Αριθμός Μητρώου:

Ασκηση 1 (10 μονάδες)

Θεωρήστε ένα πρόγραμμα Java για την προσομοίωση αγώνων ποδοσφαίρου. Σας δίνονται οι παρακάτω διεπαφές και κλάσεις Java:

```
interface FootballGame {
          void atBall(Player p) throws OffsideException,
                                       YellowException,
                                       PenaltyException;
}
class Player {}

class FootballException extends Exception {}
class YellowException extends FootballException {}
class RedException extends YellowException {}
class OffsideException extends FootballException {}
class PenaltyException extends FootballException {}
class StormException extends Exception {}
```

α) **(5 μονάδες)** Ποιες απο τις παρακάτω δηλώσεις κλάσεων Java είναι έγκυρες?
Σημείωση: Θυμηθείτε την έννοια του προγραμματισμού βασισμένου σε συμβόλαια (contracts).

```
class NiceGame implements FootballGame {
    public void atBall(Player p) throws OffsideException,
                                        YellowException
    {      }
}
```

Λύση:
Σωστή

```
class HardGame implements FootballGame {
    public void atBall(Player p) throws RedException
    {          }
}
```

Λύση:
Σωστή

```
class StormyGame implements FootballGame {
    public void atBall(Player p) throws YellowException,
                                        OffsideException,
                                        StormException
    {         }
}
```

Λύση:
Λάθος

```
class NormalGame implements FootballGame {
    public void atBall(Player p) throws FootballException
    {          }
}
```

Λύση:
Λάθος

```
class LuckyGame implements FootballGame {
    public void atBall(Player p) throws PenaltyException
    { throw new RedException("Too bad for your team");}
}
```

Λύση:
Λάθος

β) **(2 μονάδες)** Με βάση τα λάθη που βρήκατε στις παραπάνω δηλώσεις προτείνετε μια καινούργια δήλωση της διεπαφής `FootballGame` που απεικονίζει με πιό αφαιρετικό τρόπο τους κανόνες ενός ποδοσφαιρικού παιχνιδιού (και πιο συγκεκριμένα τις παραβιάσεις του συμβολαίου του)

Λύση:
```
interface FootballGame {
     void atBall(Player p) throws FootballException
}
```

γ) **(3 μονάδες)** Με δεδομένο ότι η `FootballException` είναι μια υποκλάση της `Exception` ποιες απο τις παρακάτω εντολές είναι ισοδύναμες με το ακόλουθο `try-catch`

```
try
   { f(); }
catch(Exception e)
   { if(e instanceof FootballException)
       { g(e);}
   }
```

1.
```
try { f(); }
catch(FootballException e) { g(e); }
```
2.
```
try { f(); }
catch(Exception e) {}
catch(FootballException e) { g(e); }
```

3. try { f(); }
   catch(FootballException e) { g(e); }
   catch (Exception e) { }
4. None of the above

Λύση:
1 και 3

Ασκηση 2 (15 μονάδες)

Θεωρήστε τον παρακάτω κώδικα Java:

```
interface Stuff {
     A mangle(int x, int y);
     void print();
}
class A implements Stuff {
     int a;
     int b;
     A (int x, int y) {
        a=x;
        b=y;
     }
     public A mangle(int x, int y) {
        A z;
        z = new A(a+x, b+y);
        return z;
     }
     public void adjustA(int x) {
        a=a*x;
     }
     public void print() {
         System.out.println("a=" + a + ", b=" + b);
     }
}
class B implements Stuff {
     int a;
     int b;
     int c;
     B (int x, int y, int z) {
        a=x;
        b=y;
        c=z;
     }
     public A mangle(int x, int y) {
        A z;
        z = new A(b+x, c+y);
        return z;
     }
     public void fangle(int x) {
        a=a-x;
        b=b-x;
     }
     public void print() {
         System.out.println("a="+ a+", b="+ b+", c="+ c);
     }
```

Τι τα τυπωθεί στην έξοδο για κάθε μια απο τις ακόλουθες περιπτώσεις προγραμμάτων? (σε περίπτωση λάθους σημειώστε **Λάθος** και δώστε την αιτία που το προκάλεσε):

(a)
```
A object1 = new A(1,1);
A object2 = new A(2,2);
B object3 = new B(4,5,6);
Stuff [] array = new Stuff[10];
object3.fangle(1);
array[0]=object1;
array[1]=object2;
array[2]=object3;
array[0].print();
array[1].print();
array[2].print();
```

Λύση:
```
a= 1, b= 1
a= 2, b= 2
a= 3, b= 4, c= 6
```

(b)
```
A object1 = new A(1,1);
A object2 = new A(2,2);
B object3 = new B(4,5,6);
Stuff [] array = new Stuff[10];
array[0]=object1;
array[1]=object2;
array[2]=object3;
array[2].fangle(1);
array[0].print();
array[1].print();
array[2].print();
```

Λύση:
```
error
Method fangle(int) not found in interface Stuff.
array[2].fangle(1);
```

(c)
```
A object1 = new A(1,1);
A object2 = new A(2,2);
B object3 = new B(4,5,6);
Stuff [] array = new Stuff[10];
array[0]=object1;
array[1]=object2;
array[2]=object3;
A tmp = array[2].mangle(1,1);
array[3]=tmp;
tmp.adjustA(2);
array[3].print();
```

Λύση:
```
a=12, b=7
```

Ασκηση 3 (40 μονάδες)

Σας δίνεται ο παρακάτω ορισμός μιας κλάσης Java **Clock** η οποία αναπαριστά την έννοια του χρόνου με την μορφή δύο ακεραίων μεταβλητών: **hour**, και **minute** (**hour** παίρνει τιμές απο 0 έως 23 και **minute** παίρνει τιμές απο 0 έως 59).

```
class Clock {
  // instance variables;
  // note that they are declared as private
  private int hour, minute;
  // the constructor to initialize instance variables
  public Clock(int h, int m) throws
                              IllegalArgumentException

  // add one minute to the current time
  public void tick()

  // check whether the current time is equal to the
  // time which is given as parameter; or not
  public boolean equal(Clock c)

  // convert the time into a printable format: "h:m"
  public String toString()
}
```

α) **(10 μονάδες)** Συμπληρώστε το σώμα των μεθόδων **Clock**, **tick**, **equal**, και **toString**.

Λύση:

```
public Clock(int h, int m) throws
                               IllegalArgumentException
{ if (h < 0 || h > 23 || m < 0 || m > 59)
     throw new IllegalArgumentException(
                                "Invalid Time" + h + m)
  else
     { hour=h; minute=m; }
}
public void tick()
{ minute=minute+1;
  if (minute==60)
   { minute=0; hour=hour+1;
     if (hour==24) hour=0; }
}
public boolean equal(Clock c)
{ return(hour == c.hour && minute == c.minute); }
public String toString()
{ return(hour + ":" + minute); }
```

β) **(5 μονάδες)** Ποια είναι η **invariant** συνθήκη της κλάσης **Clock** και πώς οι κατασκευαστές αντικειμένων (constructors) και οι μέθοδοι που μετασχηματίζουν την κατάσταση των αντικειμένων της κλάσης (transformers) εξασφαλίζουν ότι είναι πάντα αληθής.

Λύση:
Invariant συνθήκη: ∀c∈ Clock c.h>=0 ∧ c.h<=23∧ c.m>=0∧ c.m<=59
Στην περίπτωση του κατασκευαστή αντικειμένων **Clock** ελέγχουμε τις τιμές των παραμέτρων που παίρνει έτσι ώστε να μην παραβιάζεται η **Invariant** συνθήκη της κλάσης (η **Post** συνθήκη του κατασκευαστή είναι ουσιαστικά η **Invariant**). Στην αντίθετη περίπτωση δημιουργούμε μια εξαίρεση για να ειδοποιήσουμε τους πελάτες της κλάσης ότι παραβιάζεται το συμβόλαιό της.
Στην περίπτωση της μεθόδου μετατροπέα **tick** εξασφαλίζουμε ότι οι αλλαγές στην κατάσταση των αντικειμένων δεν παραβιάζουν την **Invariant** συνθήκη της κλάσης (η **Post** συνθήκη του μετατροπέα περιλαμβάνει την **Invariant** συνθήκη). Αν κάτι τέτοιο δεν μπορεί πάντοτε να διασφαλιστεί, τότε δημιουργούμε μια εξαίρεση (η οποία πρέπει να δηλωθεί κατάλληλα στην υπογραφή της μεθόδου) για να ειδοποιήσουμε τους πελάτες της κλάσης ότι παραβιάζεται το συμβόλαιο της.

(γ) **(10 μονάδες)** Η ακόλουθη κλάση **ClockTest** χρησιμοποιεί τις μεθόδους της **Clock**. Συμπληρώστε τα κενά στον παρακάτω κώδικα υποδεικνύοντας πότε χρειάζονται και πότε όχι **try-cath blocks**.

```
public class ClockTest {
  public static void main(String[] args)
                                        throws IOException {
    Clock c, noontime;
    int h1,m1;
    BufferedReader stdio = new
      BufferedReader (new InputStreamReader(System.in));
    System.out.println("Enter Hour: ");
    // Read an Integer value from the standard input
    h1 =
```

Λύση:
```
 h1 = Integer.parseInt(stdio.readLine());
```

```
    System.out.println("Enter Minute: ");
    // Read an Integer value from the standard input
    m1 =
```

Λύση:
```
  m1 = Integer.parseInt(stdio.readLine());
```

```
    // Create a Clock object using the values in
    // variables h1 and  m1; Store the object reference
    //in variable c.What happens if arguments are wrong?
```

Λύση:
```
try { c = new Clock(h1,m1);}
catch (IllegalArgumentException iae) {
     // eventually ask for new input values
}
```

```
//Print in the standard output the time of variable c
```

Λύση:
```
System.out.println("Time is " + c.toString());
```

```
// Add one minute to the time of variable c.
```

Λύση:
```
c.tick();
```

```
// Create a Clock object holding the noontime
// (12:00), and save it in the variable noontime
```

Λύση:
```
try { noontime = new Clock(12,0); }
 catch (IllegalArgumentException iae) {
 // this try-cath block is not useful}
```

```
// Check whether the time in the variable c is equal
// to the noontime (12:00), or not.
```

Λύση:
```
if ( c.equal(noontime) )
    System.out.println("Noon Time");
else
    System.out.println("Not Noon Time");
```

```
  }
}
```

δ) **(15 μονάδες)** Υποθέστε τώρα ότι θέλουμε μια ακριβή αναπαράσταση του χρόνου λαμβάνοντας επιπλέον υπόψιν μας και τα δευτερόλεπτα. Υλοποιείστε μια καινούργια κλάση **PreciseClock** η οποία εξειδικεύει την **Clock** έχοντας μια επιπλέον μεταβλητή στιγμιοτύπων **second** καθώς και μεθόδους συμπεριλαμβανομένου και ενός καινούργιου κατασκευαστή αντικειμένων. Στην **PreciseClock**, πρέπει να (επανα-)ορίσετε τις παρακάτω μεθόδους:

- Ο κατασκευαστής αντικειμένων της **PreciseClock** πρέπει να αρχικοποιεί τρεις μεταβλητές στιγμιοτύπων (**hour**, **minute** και **second**) χρησιμοποιώντας 3 ακεραίους που δίνονται σαν παράμετροι της μεθόδου.
- Η μέθοδος **tick** πρέπει να προσθέτει ένα δευτερόλεπτο στην τρέχουσα τιμή του χρόνου.
- Η μέθοδος **equal** πρέπει να ελέγχει αν η τρέχουσα τιμή του χρόνου είναι ίση με τον χρόνο που δίνεται σαν παράμετρος
- Η μέθοδος **toString** πρέπει να τυπώνει την τρέχουσα τιμή του χρόνου σε μια 24ωρη μορφή που περιέχει **hour**, **minute**, και **second**, π.χ. 23:10:12

Λύση:

```
class PreciseClock extends Clock {
      private int second;
      //constructor
      public PreciseClock(int h, int m, int s)
                  throws IllegalArgumentException {
          super(h,m);
          if (s < 0 || s > 59)
              throw new IllegalArgumentException(
                                  "Invalid Time" + s)
          else
              second=s;
       }
      // tick method
      public void tick() {
          second=second+1;
          if (second==60)
             { second=0; super.tick(); }
      }
      // equal method
      public boolean equal(PreciseClock c) {
          return(super.equal(c) &&
                 second==c.second);
      // toString method
      public String toString() {
          return(super.toString()+":"+second);
      }
}
```

Ασκηση 4 (15 μονάδες)

Θεωρήστε έναν ειδικό τύπο Ακολουθίας που ονομάζεται «**Κόκκινη-Μαύρη Ακολουθία**» ("red-black sequence"). Κάθε στοιχείο που περιέχεται σε μια «κόκκινη-μαύρη ακολουθία» έχει είτε το χρώμα κόκκινο, είτε το χρώμα μαύρο. Μια «**καλά ορισμένη**» **κόκκινη-μαύρη ακολουθία** ικανοποιεί τις παρακάτω 2 ιδιότητες:

1. Το κόκκινο χρώμα εμφανίζεται ακριβώς στον ίδιο αριθμό στοιχείων όσο και το μαύρο χρώμα στην ακολουθία.
2. Το μαύρο χρώμα δεν εμφανίζεται σ'ένα στοιχείο της ακολουθίας που βρίσκεται σε μια (διατεταγμένη) θέση (rank) μικρότερη από την μεγαλύτερη θέση που εμφανίζεται το κόκκινο χρώμα στην ακολουθία.

Αναπαριστώντας το κόκκινο χρώμα με το γράμμα «K» και το μαύρο χρώμα με το γράμμα «M» οι παρακάτω **κόκκινες-μαύρες ακολουθίες**:

1. **K K K K K K M M M M M M** είναι **καλά ορισμένη**
2. **K M M M M M M M M M M M** δεν είναι **καλά ορισμένη** γιατί παραβιάζει την πρώτη ιδιότητα
3. **M K K K K K M M M M M K** δεν είναι **καλά ορισμένη** γιατί παραβιάζει την δεύτερη ιδιότητα

Εξ ορισμού θεωρούμε ότι η **άδεια ακολουθία** είναι μια «**καλά ορισμένη**» **κόκκινη-μαύρη ακολουθία.**

Περιγράψτε το σώμα μιας μεθόδου **isProper(Sequence *s*)** η οποία ελέγχει αν η **κόκκινη-μαύρη ακολουθία** που δίνεται σαν παράμετρος (τύπου **Sequence**) είναι καλά ορισμένη ή όχι επιστρέφοντας την αντίστοιχη λογική τιμή (τύπου **Boolean**). Ο αλγόριθμος της μεθόδου πρέπει να ελέγχει την ακολουθία εισόδου χωρίς να μετρά με άμεσο ή έμμεσο τρόπο τον αριθμό εμφανίσεων των χρωμάτων κόκκινο και μαύρο. Επίσης θα πρέπει να είναι γραμμικός σε σχέση με το μέγεθος (size) **n** της ακολουθίας εισόδου. Για την υλοποίηση της μεθόδου χρησιμοποιείστε την διεπαφή του ΑΤΔ **Ακολουθία** (**Sequence**) μέρος της οποίας σας δίνεται παρακάτω:

- **init()** δημιουργεί μια άδεια ακολουθία.
- **isEmpty()** επιστρέφει **True** αν η ακολουθία δεν περιέχει στοιχεία και **False** στην αντίθετη περίπτωση.
- **size()** επιστρέφει έναν ακέραιο μεγαλύτερο ή ίσο του μηδενός ο οποίος δηλώνει τον αριθμό των στοιχείων που περιέχει η ακολουθία.
- **elementAtRank(r)** επιστρέφει το χρώμα (**Κόκκινο** η **Μαύρο**) του στοιχείου που βρίσκεται στην **r** θέση της ακολουθίας.

Σημειώστε ότι μπορείτε να ελέγξετε το χρώμα ενός στοιχείου μιας ακολουθίας με την συνθήκη **(*s*.elementAtRank(*r*) == Red)** και ότι μπορείτε να τοποθετήσετε τα χρώματα που σας ενδιαφέρουν σε μια ενδιάμεση συλλογή αντικειμένων. Η μέθοδος **isProper** μπορεί να υλοποιηθεί με ή χωρίς την χρήση ενδιάμεσης συλλογής αντικειμένων.

Λύση:

There are several ways to solve this problem that do not involve explicitly or implicitly counting the number of each color in such as sequence. The first technique, which is most appropriate from a data structures context, is to use a stack to match up instances of the color red with instances of the color black. With an additional test, such a solution can also ensure that property 2 above is met by a sequence. An example algorithm using this technique is below:

```
Boolean isProper(Sequence s) {
  Stack st = new Stack;
  r = 0
  // compose a stack containing the first reds
  while (r < s.size()) {
     if (s.elementAtRank(r) == Red)
         st.push(Color.red);
     else // quit loop when a black is found
         break;
     }
  // match stack of reds with blacks in sequence
  while (r < s.size()) {
     if (s.elementAtRank(r) == Black) {
        // sequence is not proper if no red to
        // match this black
        if (s.isEmpty()) return false;
        // otherwise remove the matching red
        else st.pop();
     }
     // return false if a red is found out of order
     else return false;
    }
    // sequence is proper if all reds were matched
```

```
      if (s.isEmpty()) return true;
      // sequence is not proper if any reds remain
      // unmatched
      else return false;
  }
```

Many variations on the above theme are possible, including solutions using a structure other than a stack. It is possible to compress the tests into a single loop. The above algorithm runs in time that is $O(n)$.

A more compact solution is to match the colors directly, rather than using a stack as temporary storage. An example algorithm using this technique is below:

```
   Boolean isProper(Sequence s) {
     start = 0;
     end = s.size() - 1;
     // starting from the front and rear of the
     // sequence
     while (start < end) {
        // match a front red with a rear black
        if ( (s.elementAtRank(start) == Black) ||
              (s.elementAtRank(end) == Color.red)
            // sequence is not proper if colors
            // don't match
            return false;
        // move towards the middle of the sequence
        start = start + 1
        end = end - 1
       }
      // sequence is proper if all colors match
     return true;
    }
```

Many variations on the above theme are possible, including solutions that start matching blacks from the middle rank of the sequence. The above algorithm runs in time that is $O(n)$.

Ασκηση 5 [20 μονάδες]

Υποθέστε ότι η διοίκηση του Πανεπιστημίου Κρήτης σας ανέθεσε την υλοποίηση του Πληροφοριακού Συστήματος Διαχείρισης Φοιτητών (ΠΣΔΦ). Ο Αφαιρετικός Τύπος Δεδομένων (ΑΤΔ) που αναπαριστά τους φοιτητές σας δίνεται με την μορφή μιας διεπαφής Java. Για τις ανάγκες της υλοποίησης του ΠΣΔΦ χρειάζετε να ξέρετε απλώς ότι αυτός ο ΑΤΔ επεκτείνει την παρακάτω βασική διεπαφή Java:

```
public interface CoreStudent {
     int studentID();
     String studentName();
     int studentGraduationYear();
     int studentGradePointAverage();
}
```

α) **[5 μονάδες]** Υστερα από συνεννόηση με τους διοικητικούς υπαλλήλους, αποφασίζετε ότι η πιο σημαντική λειτουργικότητα που το ΠΣΔΦ πρέπει να παρέχει, είναι η παραγωγή λιστών για διαφορετικές κατηγορίες φοιτητών σε αλφαβητική σειρά. Ποιος πιστεύετε ότι είναι ο καταλληλότερος ΑΤΔ (δηλ διεπαφή απο το Java Collection Framework) για να βασίσετε την σχεδίαση του ΠΣΔΦ και γιατί?

Λύση:
The `SortedMap` ADT provides the best support for maintaining alphabetical lists and it available in the `java.util.library` as interface: `java.util.SortedMap`.Note that different students may have the same name and therefore the ADT SortedSet can't be used.

β) **[5 μονάδες]** Ποια συγκεκριμένη δομή δεδομένων θα χρησιμοποιήσετε για την υλοποίηση του ΑΤΔ που επιλέξατε στο προηγούμενο ερώτημα και γιατί?

Λύση:
Various underlying concrete data structures offer a range of performance tradeoffs. Either a binary search tree (BST) or an AVL or even a red-black tree would give reasonable size and update performance while still supporting easy traversal in alphabetical order. The `TreeMap` implementation of `SortedMap` is available in the `java.util.library` as class: `java.util.TreeMap`.

γ) **[5 μονάδες]** Υποθέστε τώρα ότι οι διοικητικοί υπάλληλοι του Πανεπιστημίου Κρήτης έχουν επίσης ανάγκη να ανακτούν γρήγορα πληροφορίες σχετικά με τους φοιτητές δίνοντας τον κωδικό τους (student ID). Πως πρέπει να διορθώσετε ή να επεκτείνετε τον ΑΤΔ και την υποκείμενη δομή δεδομένων του ΠΣΔΦ που επιλέξατε στα προηγούμενα 2 ερωτήματα? Δικαιολογήστε την απάντησή σας.

Λύση:
Adding a `HashTable` mapping IDs to student information in the `SortedMap` would be a good choice. This would allow O(1) access to the information given the student ID. We would need to modify the binary search tree (BST) or an AVL or even a red-black tree to support locator-based methods.

δ) **[5 μονάδες]** Υποθέστε τέλος ότι πρέπει να τυπώνετε τους αποφοιτήσαντες φοιτητές για κάθε ακαδημαϊκό έτος ταξινομημένους σύμφωνα με τον μέσο όρο της βαθμολογίας τους. Ποιον επιπλέον ΑΔΤ, συγκεκριμένη δομή δεδομένων και μεθόδους πρέπει να υλοποιήσετε για το ΠΣΔΦ. Δικαιολογήστε τις επιλογές σας.

Λύση:
A priority queue of items, whose keys are student grade point averages and whose elements are locators for student information would allow for heap sort-based listing of students in order of grade pointer average. We could make one pass through an enumeration of the students, constructing one priority queue for each class year, whenever such a listing was needed. Recall that a *priority queue* is similar to a queue. However, when you remove items from it, rather than removing the front item, you remove the "most important" item. All items are stored with a corresponding priority. The "most important" item is the one with the highest priority