Πανεπιστήμιο Κρήτης
Τμήμα Επιστήμης Υπολογιστών

ΗΥ-252 – Αντικειμενοστραφής Προγραμματισμός
Βασίλης Χριστοφίδης

Τελική Εξέταση (3 ώρες)
Ημερομηνία: 28 Ιανουαρίου 2005

Ονοματεπώνυμο:
Αριθμός Μητρώου:

Άσκηση 1 (10 μονάδες)
Τι τυπώνεται στην έξοδο του παρακάτω προγράμματος Java;

```
public class ExceptionQuestion {
     public static void main(String args[]) {
      m1(15);
      m1(123);
      m1(17);
      m1(110);
     } // end main
     public static void m1(int n) {
      try {
       m2(n);
      }
      catch (NullPointerException e) {
       System.out.println("C" + n);
      }
      catch (NumberFormatException e) {
       System.out.println("D" + n);
      }
     } // end m1
     public static void m2(int x) throws NullPointerException{
      try {
       if (x % 5 == 0)
        throw new NullPointerException();
       if (x > 100)
        throw new NumberFormatException();
       System.out.println("A" + x);
      }
      catch (NumberFormatException e) {
       throw new NullPointerException();
      }
      catch (NullPointerException e) {
       System.out.println("B" + x);
      }
     } // end m2
} // end class ExceptionQuestion
```

Σημείωση: Οι εξαιρέσεις `NullPointerException` και `NumberFormatException` είναι και οι δύο υποκλάσεις της `RuntimeException`. Καμία από τις δύο δεν είναι υποκλάση της άλλης.

Λύση
B15
C123
A17
B110

Άσκηση 2 (15 μονάδες)

Υποθέστε ότι έχετε μια στοίβα aStack και μια βοηθητική κενή στοίβα auxStack. Δείξτε πώς μπορείτε να υλοποιήσετε σε μορφή ψευδοκώδικα κάθε μια από τις παρακάτω λειτουργίες χρησιμοποιώντας μόνο το συμβόλαιο (δηλ. τις μεθόδους) του ΑΤΔ (ADT) στοίβα, που σας έχει δοθεί στις διαλέξεις του μαθήματος (push(), pull()):

1. **(5 μονάδες)** Τυπώστε τα στοιχεία της στοίβας aStack σε αντίστροφη σειρά, δηλ. εμφανίστε τελευταία την οροφή (top) της στοίβας.
2. **(5 μονάδες)** Μετρήστε τον αριθμό των στοιχείων της στοίβας aStack, αφήνοντας την κατάσταση της aStack αμετάβλητη.
3. **(5 μονάδες)** Διαγράψτε όλες τις εμφανίσεις ενός συγκεκριμένου στοιχείου someValue από την aStack, αφήνοντας αμετάβλητη τη σειρά των υπόλοιπων στοιχείων της στοίβας.

Σημείωση: Μπορείτε να θεωρήσετε ότι τα στοιχεία της στοίβας είναι του τύπου StackItemType.

Λύση

1

```
StackItemType val;
while (!aStack.isEmpty()) {
     if (aStack.pop(val))
          auxStack.push(val);
}
while(!auxStack.isEmpty()) {
     if (auxStack.pop(val))
     {
          System.out.println(val);
          aStack.push(val);
     }
}
```

2

```
StackItemType val;
int count = 0;
while (!aStack.isEmpty()) {
     if (aStack.pop(val)) {
          auxStack.push(val);
          count++;
     }
}
while (!auxStack.isEmpty()) {
     if (auxStack.pop(val))
          aStack.push(val);
}
```

3

```
StackItemType val, toDelete = someValue;
while (!aStack.isEmpty()) {
     if (aStack.pop(val))
          if (val != toDelete) auxStack.push(val);
}
while (!auxstack.isEmpty()) {
     if (auxStack.pop(val))
          aStack.push(val);
}
```

Άσκηση 3 (15 μονάδες)

Σας δίνεται η παρακάτω κλάση Date:

```
class Date implements Comparable {
     public int month;
     public int day;
     public Date(int m, int d) {
          month = m;
          day = d;
     } // end toString
     public String toString() {
          return month + "/" + day;
     } // end toString
     public int compareTo(Object obj) {
          Date otherDate = (Date) obj;
          if (month > otherDate.month) return 1;
          else if (month < otherDate.month) return -1;
          else if (day > otherDate.day) return 1;
          else if (day < otherDate.day) return -1;
          else return 0;
     } // end compareTo
} // end class Date
```

Θέλουμε τώρα να χειριστούμε μία συλλογή από ημερομηνίες μέσω της ακόλουθης κλάσης CollectionDate:

```
import java.util.*;
public class CollectionDate {
     public static void main(String args[]) {
          TreeSet dates = new TreeSet();
          dates.add(new Date(12, 25));
          dates.add(new Date(6, 1));
          dates.add(new Date(2, 14));
          dates.add(new Date(6, 6));
          dates.add(new Date(7, 4));
          printSet(dates);
          TreeSet newDates = change(dates);
          printSet(dates);
          printSet(newDates);
     } // end class main
     public static TreeSet change(TreeSet dateSet) {
          TreeSet newSet = new TreeSet();
          Iterator iter = dateSet.iterator();
          while (iter.hasNext()) {
               Date nextDate = (Date)iter.next();
               if (nextDate.month > nextDate.day) {
                    iter.remove();
                    newSet.add(nextDate);
               } // end if
          } // end while
          return newSet;
          } // change
     public static void printSet(Set s) {
          // To be completed ...
     } // end printSet

} // end class CollectionDate
```

1. **(5 μονάδες)** Υλοποιείστε την μέθοδο printSet() της κλάσης CollectionDate ώστε να τυπώνονται οι ημερομηνίες σε μία γραμμή αφήνοντας ένα κενό μεταξύ τους.

Λύση

```
public static void printSet(Set s) {
     Iterator iter = s.iterator();
     while (iter.hasNext())
     System.out.print(iter.next() + " ");
     System.out.println();
     } // end printSet
```

Common Errors:

- Not proper use of iterators
- Decaplutation of `Date` object
- Explicit use of the `toString()` method

2. **(10 μονάδες)** Τι τυπώνεται στην έξοδο του προγράμματος όταν εκτελεστούν οι τρεις εντολές `printSet()` στην μέθοδο `main()` της κλάσης `CollectionDate;`

Λύση

Η μέθοδος τυπώνει τα αποτελέσματα σε φθίνουσα διάταξη. Την πρώτη φορά τυπώνονται οι αρχικές ημερομηνίες, την δεύτερη οι ημερομηνίες που δεν έχουν αφαιρεθεί και την τρίτη οι ημερομηνίες που έχουν αφαιρεθεί.
Output from first `printSet`:
2/14 6/1 6/6 7/4 12/25
Output from second `printSet`:
2/14 6/6 12/25
Output from third `printSet`:
6/1 7/4

Common Errors:

• elements not properly sorted (remember a `TreeSet` is a `SortedSet` and the method `compareto()`of `Date` class will be used to sort the objects)
• dates not formatted according to `toString()` in `Date` class
• many students had the 2nd or the 3rd printing equal to the 1st
• many students left out one of the dates consistently (usually 6/1 or 6/6)

Άσκηση 4 (15 μονάδες)

Ένα παλίνδρομο είναι μία συμβολοσειρά η οποία μπορεί να διαβαστεί ακριβώς με τον ίδιο τρόπο είτε από αριστερά προς τα δεξιά είτε αντιστρόφως. Για παράδειγμα, η συμβολοσειρά “radar” είναι ένα παλίνδρομο. Τυπικά ένα παλίνδρομο ορίζεται ως εξής:

- η κενή συμβολοσειρά είναι ένα παλίνδρομο
- μια συμβολοσειρά με το λιγότερο ένα χαρακτήρα, είναι ένα παλίνδρομο όταν ο πρώτος και ο τελευταίος της χαρακτήρας είναι ίσοι και η υπο-συμβολοσειρά που παράγεται αφαιρώντας τους δύο αυτούς χαρακτήρες είναι επίσης ένα παλίνδρομο.

Υλοποιήστε μια στατική αναδρομική μέθοδο `isPalindrome()` η οποία ελέγχει εάν μία συμβολοσειρά είναι παλίνδρομη σύμφωνα με τον παραπάνω τυπικό ορισμό. Η υπογραφή της μεθόδου σας δίνεται παρακάτω, και για να ελέγξετε εάν η συμβολοσειρά `s` είναι παλίνδρομη θα πρέπει να εκτελεστεί η μέθοδος `isPalindrome(s, 0, s.length()-1)` με τις κατάλληλες παραμέτρους:

```
public static boolean isPalindrome(String s,  int first, int last){
     // ... }
```

Δώστε επίσης τις εκ των προτέρων (pre) και εκ των υστέρων (post) συνθήκες που διέπουν το συμβόλαιο χρήσης της.
Σημείωση: Για την υλοποίηση της μεθόδου χρησιμοποιήστε την διεπαφή της κλάσης `String`.

Λύση

```
// Determines whether a string is a palindrome.
// PRE: s != null, first >= 0, last < s.length()
// POST: Returns true if the substring of s starting from
// index first through to index last is a palindrome,
// false otherwise

public static boolean isPalindrome(String s, int first, int last) {
// Base case: empty substring and single-char substring
     if (first >= last)
          return true;
     // Recursive step: compare first and last chars
     else
          return (s.charAt(first) == s.charAt(last)
                  && isPalindrome(s, first+1, last-1));
} // end isPalindrome
```

Άσκηση 5 (10 μονάδες)

Ποια είναι η πολυπλοκότητα του χρόνου εκτέλεσης των τριών μεθόδων που σας δίνονται ακολούθως; Δικαιολογήστε σύντομα την απάντηση σας.

1. (3 μονάδες)

```
public static void method1(int n){
     double x = Math.random(); // you may assume this method call
                               // takes constant time
     if (x < 0.1)
          for (int i = 1; i <= n; i++)
               System.out.println("*");
     else
          for (int i = 1; i <= 10; i++)
               System.out.println("*");
     }
```

Λύση

O(n)

2. (4 μονάδες)

```
public static void method2(int n){
     for (int i = n; i > 0; i = i/2)
          System.out.println("*");
     }
```

Λύση

O(log n)

3. (3 μονάδες)

```
public static void method3(int n){
     int m = 2*n;
     for (int i = 1; i <= m; i++)
          for (int j = i; j <= m; j++)
               System.out.println(i*j);
     }
```

Λύση

$O(n^2)$

Common Errors:

Unnecessary constants (such as answering O(21*n) for part a)) resulted in 1 mark being subtracted for each instance. For part b), an answer of O(n) was given 2 marks, since this was the most difficult of the three parts. For part c), an answer of O(n) was given 1 mark, and an answer of $O(n^3)$ was given 2 marks.

Άσκηση 6 (25 μονάδες)

Πολλά παιχνίδια υπολογιστών κρατούν το ιστορικό των αποτελεσμάτων που οι παίκτες έχουν επιτύχει σε προηγούμενα παιχνίδια. Μια ταξινόμηση των αποτελεσμάτων βασίζεται συνήθως σε μια διάταξη όπου τα "καλύτερα" αποτελέσματα εμφανίζονται πριν από τα "λιγότερο καλά" αποτελέσματα. Όταν τα αποτελέσματα εμφανίζονται σε μια οθόνη (π.χ.με το `System.out`), αυτό σημαίνει ότι τα καλύτερα αποτελέσματα τυπώνονται πρώτα ενώ τα λιγότερο καλά αποτελέσματα τυπώνονται αργότερα. Υποθέστε ότι έχουμε την κλάση `Score` και τη διεπαφή `HighScoreRanking`:

```
public class Score {
     private String name ;
     private int points ;
     private String date ;
     public Score (String name, int points, String dateOfGame) {
          this.name = name ;
          this.points = points ;
          date = dateOfGame ;
     }
     public String toString() {
          return (points + " " + name + " ( " + date + " ) " ) ;
     }
}

import java.util.*;
public interface HighScoreRanking {
     boolean add(Comparable c) ;
     Iterator iterator() ;
}
```

Ένα αντικείμενο `HighScoreRanking` προορίζεται να αποθηκεύσει όλες τις ταξινομήσεις που προστέθηκαν χρησιμοποιώντας την μέθοδο `add()`. Το αντικείμενο `Iterator` που επιστρέφεται από την μέθοδο `iterator()` θα πρέπει να επιστρέφει αυτά τα αποτελέσματα σε μια διάταξη τέτοια ώστε το καλύτερο αποτέλεσμα να βγαίνει πρώτο, έπειτα να βγαίνει το δεύτερο καλύτερο αποτέλεσμα, κλπ. Φυσικά, ένα αποτέλεσμα με περισσότερους βαθμούς είναι καλύτερο από ένα αποτέλεσμα με λιγότερους. Εάν οι βαθμοί είναι οι ίδιοι, το αποτέλεσμα του πιο πρόσφατου παιχνιδιού θεωρείται το καλύτερο. Εάν οι βαθμοί και οι ημερομηνίες είναι οι ίδιες, διατάζουμε τα αποτελέσματα αλφαβητικά μετά από τα ονόματα των παιχτών. Μια υλοποίηση της `HighScoreRanking` θα πρέπει λοιπόν να αποθηκευτεί σε μια ταξινομημένη συλλογή. Θυμηθείτε ότι τα τυποποιημένα `iterators` της Java σε ταξινομημένες συλλογές επιστρέφουν αρχικά το μικρότερο στοιχείο, έπειτα το δεύτερο μικρότερο στοιχείο, κλπ. Επομένως, για να εμφανίσουμε πρώτα το καλύτερο αποτέλεσμα, θα πρέπει να υλοποιήσουμε μια κατάλληλη μέθοδο διάταξης στα αποτελέσματα των παιχνιδιών έτσι ώστε το καλύτερο αποτέλεσμα να βγαίνει και μικρότερο σύμφωνα με αυτήν την διάταξη.

1 (10 μονάδες) Η μέθοδος `add()` της `HighScoreRanking` απαιτεί ορίσματα του τύπου `Comparable`. Αλλά τα αντικείμενα της κλάσης `Score` δεν είναι (ακόμα) τύπου `Comparable`. Τροποποιήστε την κλάση `Score` έτσι ώστε τα αντικείμενα της να μπορούν να δοθούν σαν ορίσματα της μεθόδου `add()`. Δεν επιτρέπεται φυσικά να αλλάξετε τη διεπαφή `HighScoreRanking`.

Σημείωση: Δεν απαιτείται η υλοποίηση της μεθόδου `equals()`. Μπορείτε να θεωρήσετε ημερομηνίες της μορφής: "YYMMDD", όπως "051207" ή "051130" έτσι ώστε να χρησιμοποιήσετε την αλφαβητική διάταξη των συμβολοσειρών και για την ταξινόμηση των ημερομηνιών.

Λύση

```
public class Score implements Comparable {
     . . .
     public int compareTo (Object o) {
       Score other = (Score) o ;
       if (points < other.points) return -1;
       else if (points > other.points) return 1;
            else {
              if (date.compareTo(other.date) !=0)
                 return –(date.compareTo(other.date));
              else
                 return name.copareTo(other.name);
             }
     }
     . . .
}
```

2 (10 μονάδες) Δώστε την κλάση `MyScoreRanking` η οποία υλοποιεί την διεπαφή `HighScoreRanking` σύμφωνα με τις παρατηρήσεις που σας δόθηκαν προηγουμένως.

Λύση

```
import java . util.*;
public class MyScoreRanking implements HighScoreRanking {
     private Set theScores = new TreeSet() ;
     public boolean add(Comparable c) {
          return theScores.add (c);
     }
     public Iterator iterator() {
          return theScores.iterator() ;
     }
}
```

3 (5 μονάδες) Συμπληρώστε την παρακάτω κλάση (πελάτης) `HighScoreTest` σύμφωνα με τις παρατηρήσεις που σας δόθηκαν προηγουμένως και γράψτε τι θα τυπωθεί στην έξοδο του προγράμματος μετά την εκτέλεση της μεθόδου `main()`:

```
import java.util.*;
public class HighScoreTest {
     public static void main(String [ ] args) {
     HighScoreRanking ranking = new MyScoreRanking();
     ranking.add (new Score ("Vassilis", 700, "041207"));
     ranking.add (new Score ("Dimitris", 900, "041208"));
     ranking.add (new Score ("Nikos", 900, "041209"));
     ranking.add (new Score ("Giorgos", 800, "041206"));
     ranking.add (new Score ("Nikos", 700, "041207"));
     System.out.println("Here comes the current ranking: ");
     // print all scores of ranking to System.out
     }
}
```

Λύση

```
     System.out.println( "Here comes the current ranking: ");
     Iterator it = ranking.iterator();
     while (it.hasNext())
          System.out.println(it.next());
```

```
900 Nikos 041209
900 Dimitris 041208
800 Giorgos 041206
700 Nikos 041207
700 Vassilis 041207
```

Άσκηση 7 (10 μονάδες)

Σας δίνεται ο παρακάτω κώδικας Java:

```
public interface Mapper {
     public int f (int i);
}
public class AMapper implements Mapper {
     public int f (int i) {return 2+i;}
}
public class MyObj {
     int state;
     MyObj (int i) {state = i;}
}
public class Test {
     static void map (Mapper m, int i) {
          i = m.f(i);}
     public static void main(String [ ] args) {
          MyObj o = new MyObj(7);
          Mapper mm = new AMapper();
          map (mm, o.state);
          System.out.println(o.state);
     }
}
```

1. **(3 μονάδες)** Τι τυπώνεται μετά την εκτέλεση της μεθόδου `main()` της κλάσης `Test`; Δικαιολογήστε σύντομα την απάντησή σας.

Λύση

It prints 7. Since in Java parameters are call-by-value, the statement `i = m.f(i)` does not change the value stored in `o.state`.

2. **(7 μονάδες)** Διορθώστε τον παραπάνω κώδικα της κλάσης `Test` έτσι ώστε η μεταβλητή στιγμιοτύπων `state` του αντικειμένου o να αλλάζει πραγματικά.

Σημείωση: Μην αλλάξετε τον κώδικα (προμηθευτή) της κλάσης `MyObj`.

Λύση

```
public class Test2 {
     static void map(Mapper m, MyObj o) {
          o.state = m.f(o.state);
     }
     public static void main(String[] argv) {
          MyObj o = new MyObj(7);
          Mapper mm = new AMapper();
          map(mm, o);
          System.out.println(o.state);
     }
}
```

Εναλλακτικά

```
public class Test3 {
     static int map(Mapper m, int i) {
          return m.f(i);
     }
     public static void main(String[] argv) {
          Mapper mm = new AMapper();
          MyObj o = new MyObj(map(mm, 7));
          System.out.println(o.state);
     }
}
```