Πανεπιστήμιο Κρήτης
Τμήμα Επιστήμης Υπολογιστών

ΗΥ-252 – Αντικειμενοστρεφής Προγραμματισμός
Βασίλης Χριστοφίδης

Πρόοδος (3 ώρες)
Ημερομηνία: 3 Δεκεμβρίου 2011

Άσκηση 1	Άσκηση 2	Άσκηση 3	Άσκηση 4	Άσκηση 5	Άσκηση 6	Σύνολο
/12	/20	/10	/32	/12	/20	/106

Ονοματεπώνυμο:
Αριθμός Μητρώου:

Άσκηση 1 (12 μονάδες) // Στατικοί και Δυναμικοί Τύποι Αντικειμένων

Σας δίνονται οι ακόλουθοι ορισμοί κλάσεων:
```
class S { }
class T extends S { }
class U extends S { }
class V extends U { }
```

Για κάθε μία από τις παρακάτω ερωτήσεις κυκλώστε την σωστή απάντηση: μπορεί να υπάρχει καμία, μία ή περισσότερες σωστές απαντήσεις.

(α) **(4 μονάδες)** Ποιος είναι ο στατικός τύπος της μεταβλητής x στον παρακάτω κώδικα;
```
S x = new V();
x = new U();
```

i) `Object` ii) `S` iii) `T` iv) `U` v) `V`

> **Λύση:**
> ii

(β) **(4 μονάδες)** Ποιος είναι ο δυναμικός τύπος της τιμής της μεταβλητής x μετά την εκτέλεση του παρακάτω κώδικα;
```
S x = new V();
x = new U();
```

i) `Object` ii)`S` iii) `T` iv) `U` v) `V`

> **Λύση:**
> iv

(γ) **(4 μονάδες)** Ποιοι τύποι είναι υπερτύποι του δυναμικού τύπου μεταβλητής o;
```
Object o = new U();
```

i) `Object` ii) `S` iii) `T` iv) `U` v) `V`

> **Λύση:**
> i, ii and iv. We would also accept only i and ii, as the lecture notes don't specify that classes can be considered supertypes of themselves.

Άσκηση 2 (20 μονάδες) // Κατανόηση κώδικα Java

Τι θα τυπωθεί στην στανταρ έξοδο μετά την εκτέλεση των παρακάτω προγραμμάτων Java; Αναφέρετε τυχόν λάθη.

(α) **(4 μονάδες)**

```
class A { int foo() { return 1; }}
class B extends A { int foo() { return 2; }}
class C extends B { int bar(A a) { return a.foo(); }}
C x = new C();
System.out.println(x.bar(x));
```

Λύση:
2

(β) **(4 μονάδες)**

```
class A { int e = 1; }
class B extends A { int e = 2; }
class C extends B { int bar(A a) { return a.e; }}
C x = new C();
System.out.println(x.bar(x));
```

Λύση:
1

(γ) **(12 μονάδες)**

```
class C {
     public void foo() {
          try {
               bar();
               System.out.println("here in foo");
          } catch (RuntimeException e) {
               System.out.println("caught in foo");
          }
     }
     public void bar() {
          try {
               baz();
          } catch (RuntimeException e) {
               System.out.println("caught in bar");
               throw e;
          } finally {
               System.out.println("here in bar");
          }
          System.out.println("there in bar");
     }
     public void baz() {
          try {
               if (true) throw new RuntimeException();
          } catch(RunTimeException e) {

          }finally {
               System.out.println("here in baz");
          }
          System.out.println("there in baz");
     }
}

public class Exn {
```

```
    public static void main (String args[]) {
        (new C()).foo();
    }
}
```

Λύση:

```
here in baz
there in baz
here in bar
there in bar
here in foo
```

Άσκηση 3 (10 μονάδες) //Χειρισμός Συμβολοσειρών

Οταν γράφουμε μια μαθηματική έκφραση, ένα πρόγραμμα, ή ένα τεχνικό εγχειρίδιο, χρειαζόμαστε συχνά να ελέγχουμε εάν οι παρενθέσεις που χρησιμοποιούμε ταιριάζουν. Για παράδειγμα, θα θέλαμε να ελέγξουμε εάν στις παρακάτω συμβολοσειρές όλες οι παρενθέσεις που ανοίγουν επίσης κλείνουν (φυσικά και αντιστρόφως):

- `((3+5)-6)/2`
- `int i = (temp + 2) / (j – 6);`
- `γράφω μια πρόταση (η οποία περιέχει παρενθέσεις).`
- `(lambda (arg) (+ arg 1))`

Κάτι τέτοιο δεν συμβαίνει στις παρακάτω συμβολοσειρές:

- `(3 + 5) / (2 + 7`
- `int i = (3 * (54 – 88);`
- `γράφω μια πρόταση με μια μόνο παρένθεση (η οποία δεν κλείνει!`
- `(+ 3 (/ 5 8) (* 3 4)`
- `(3 + 5) / 4 )`
- `) 76 / 6 (`

Υλοποιήστε μια μέθοδο Java η οποία παίρνει μια συμβολσειρά (`String`) σαν είσοδο και επιστρέφει αληθές (`true`) εάν όλες οι παρενθέσεις ταιριάζουν (δηλ για κάθε παρένθεση που ανοίγει υπάρχει μια παρένθεση που κλείνει), διαφορετικά ψευδές (`false`).

```
public boolean parenthesesBalanced(String str) {
```

Λύση:

```
int openParentheses = 0;
for (int i = 0; i < str.length(); i++) { //for each character
  if (str.charAt(i) == '(') {
    openParentheses++; //we just opened something new
  }
  else if (str.charAt(i) == ')') {
    if (openParentheses == 0)
      return false; //we closed without having anything open
    else
      openParentheses--; //we just closed an open parenthesis
  }
}
return openParentheses==0; //don't want parentheses not closed
}
```

Άσκηση 4 (32 μονάδες) // Χειρισμός Πινάκων και Αναδρομικές Μέθοδοι

Υλοποιήστε μια αναδρομική μέθοδο για την επίλυση του παρακάτω προβλήματος διαμέρισης. Δεδομένου ενός πολυ-συνόλου ακεραίων μας ενδιαφέρει η διαμέριση των στοιχείων του σε δύο πολυ-σύνολα έτσι ώστε το άθροισμα των ακεραίων σε κάθε μέρος να

είναι το ίδιο. Για παράδειγμα, το πολυ-σύνολο {3, 3, 5, 9, 2} μπορεί να διαμεριστεί στα {3, 3, 5} και {9, 2} (με συνολικό άθροισμα 11 και για τα δύο μέρη), αλλά το πολυ-σύνολο {1, 5, 16} δεν μπορεί να διαμεριστεί. Σημειώστε ότι τα δύο μέρη δεν χρειάζεται να έχουν το ίδιο πλήθος στοιχείων και ότι τόσο στο αρχικό όσο και στα παραγόμενα πολυ-σύνολα ακεραίων, ένα στοιχείο μπορεί να εμφανίζεται παραπάνω από μια φορές.

α) **(17 μονάδες)** Υλοποιήστε την μέθοδο `CanBePartitioned()` η οποία παίρνει σαν παράμετρο έναν πίνακα ακεραίων προς διαμέριση και επιστρέφει αληθές ή ψευδές ανάλογα με το εάν οι τιμές του πίνακα μπορούν να διαμεριστούν σύμφωνα με την παραπάνω λογική. Θα χρειαστείτε για αυτό τον σκοπό να υλοποιήσετε και μια βοηθητική αναδρομική μέθοδο `partition()` η οποία διασχίζει τα στοιχεία του πίνακα ακεραίων που δίνεται σαν όρισμα και ελέγχει εάν η διαμέριση τους είναι δυνατή λαμβάνοντας υπόψη το τρέχον άθροισμα των ακεραίων της διαμέρισης σε κάθε βήμα της αναδρομής.

```
boolean CanBePartitioned(int [] values) {
```

Λύση:

```
 return partition(values, 0, 0); //2 points
}

boolean partition(int[] values, int pos, int sum) {
```

Λύση:

```
if (pos >= values.length)
    return sum == 0;
if (partition(values, pos+1, sum+values[pos]))
   return true;
return partition(values, pos+1, sum-values[pos]);
}
```

β) **(5 μονάδες)** Ποια είναι η ασυμπτωτική πολυπλοκότητα χείριστης περίπτωσης της μεθόδου `partition()` που υλοποιήσατε στο προηγούμενο ερώτημα;

Λύση:

$O(2^n)$

γ) **(10 μονάδες)** Σας δίνεται το παρακάτω πρόγραμμα:

```
public static void main(String[] args) {
    int[] array = new int[]{1, 2, 3, 4};
    int[] cloneArray = array.clone();

    System.out.println(array == cloneArray);
    System.out.println(array.equals(cloneArray));
    System.out.println(myEquals(array, cloneArray));
}
public static boolean myEquals(int[] array1, int[] array2) {
    for (int i = 0; i < array1.length; i++) {
        if (array1[i] != array2[i]) {
            return false;
        }
    }
    return true;
}
```

1) **(5 μονάδες)** Τι θα εκτυπωθεί μετά την εκτέλεση του προγράμματος;

```
i)   true true true,
ii)  false true true
```

iii) false false true
iv) false false ArrayIndexOutOfBounds

Λύση:
iii

2) **(5 μονάδες)** Σκοπός της μεθόδου `myEquals()` είναι να συγκρίνει τα στοιχεία δύο πινάκων και να επιστρέφει αληθές (true) αν οι δύο πίνακες περιέχουν ακριβώς τα ίδια στοιχεία στις ίδιες θέσεις. Η μέθοδος αυτή δουλεύει σωστά για οποιουσδήποτε πίνακες ακεραίων; Αν όχι σε ποιες περιπτώσεις δε θα δούλευε σωστά; Πώς μπορεί να διορθωθεί το πρόβλημα αυτό;

Λύση:
This method assumes that the two arrays have the same number of elements. A check about the length of the arrays should be added at the beginning of the method.

```
if (array1.length != array2.length) {
    return false;
}
```

Άσκηση 5 (12 μονάδες) // Υλοποίηση Βασισμένη σε Συμβόλαια

Σας δίνεται η ακόλουθη κλάση Java η οποία ικανοποιεί την αμετάβλητη συνθήκη (invariant condition): «*Εάν ο πίνακας της μεταβλητής data είναι null ή ο πίνακας είναι κενός, η τιμή της μεταβλητής x θα πρέπει να είναι μηδέν. Στην αντίθετη περίπτωση η τιμή της μεταβλητής x θα πρέπει να ισούται με την μεγαλύτερη τιμή του πίνακα.*»

```
1 public class A {
2    private int x = 0;
3    private int[] data;
4
5    // update x to satisfy the invariant
6    private void updateX() {
7         if (data == null || data.length == 0) {
8              x = 0;
9         } else {
10             x = data[0];
11             for (int i=1; i< data.length; i++) {
12                  if (data[i] > x) {
13                       x = data[i];
14                  }
15             }
16        }
17   }
18
19   public A(int[] d0) {
20        data = d0;
21        updateX();
22   }
23
24   // increment all values in the array
25   public void incr() {
26        for (int i = 0; i < data.length; i++) {
27             data[i] = data[i] + 1;
28        }
29        updateX();
30   }
31
32 }
```

(α) **(4 μονάδες)** Είναι δυνατό ένας κώδικας πελάτης να προκαλέσει την δημιουργία μιας εξαίρεσης κατά την εκτέλεση του κώδικα αυτής της κλάσης; Σε ποιά γραμμή του κώδικα της κλάσης θα εγερθεί (throw) η εξαίρεση; Ποιες εντολές θα δημιουργήσουν αυτή την εξαιρετική κατάσταση λειτουργίας του προγράμματος;

Λύση:
The problem is at line 26, at the use of `data.length`. A `NullPointerException` could be thrown with the call `new A(null).incr();`

(β) **(8 μονάδες)** Είναι δυνατό ένας κώδικας πελάτης να παραβιάσει την αμετάβλητη συνθήκη χωρίς να δημιουργεί μια εξαίρεση; Ποια ακολουθία εντολών θα μπορούσε να προκαλέσει αυτό το πρόβλημα;

Λύση:
The invariant could be violated by changing the data array after the object was created through an alias.

```
int data[] = { 1 };
A a = new A(data);
data[0] = 500;
```

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

Σχεδιάστε και υλοποιήστε τον Αφαιρετικό Τύπο Δεδομένων (**ΑΤΔ**) μιας αυτο-διοργανωμένης συνδεδεμένης λίστας (**`Self-Organizing List`**). Μία τέτοια λίστα έχει την ιδιότητα να αναδιοργανώνει τα στοιχεία της ανάλογα με την συχνότητα αναζήτησης τους μεταθέτοντάς τα στην αρχή της λίστας. Δώστε τις υπογραφές (signatures), το είδος ανάλογα με την επίδραση που έχουν στην κατάσταση των αντικειμένων (constructors, accessors, transformers), και τις εκ των προτέρων, τις εκ των υστέρων και τις αμετάβλητες συνθήκες (preconditions, postconditions, invariants) που διέπουν τις παρακάτω λειτουργίες στο συμβόλαιο του **ΑΤΔ `Self-Organizing List`**:

- `SelfOrganizedList()`:Δημιουργία μιας κενής αυτό-διοργανωμένης συνδεδεμένης λίστας
- `void add(Object o)`: Εισαγωγή ενός αντικειμένου στην λίστα
- `boolean remove(Object o)`: Διαγραφή ενός αντικειμένου από την λίστα
- `int size()`: Επιστροφή του μεγέθους της λίστας
- `Object find(Object o)`: Αναζήτηση και επιστροφή ενός στοιχείου της λίστας

Σημασιολογία

- Η εισαγωγή ενός στοιχείου στην λίστα γίνεται με το σύνηθες τρόπο, δηλαδή εισαγωγή στο τέλος της λίστας. Με το σύνηθες τρόπο γίνεται και η διαγραφή, δηλαδή αφαιρείται το στοιχείο και γίνονται οι απαραίτητες ανανεώσεις των δεικτών της λίστας.
- Η αναζήτηση ενός στοιχείου γίνεται ξεκινώντας από την κεφαλή της λίστας και ακολουθώντας τους δείκτες στα επόμενα στοιχεία μέχρι ωσότου βρεθεί αυτό που αναζητούμε.
- Σε μια αυτο-διοργανωμένη λίστα, κάθε φορά που γίνεται μια επιτυχής αναζήτηση ενός στοιχείου, το ανευρεθέν στοιχείο μεταφέρεται στην αρχή της λίστας. Αυτή η στρατηγική, μετά από πολλές επαναλήψεις της μεθόδου της αναζήτησης τείνει στο να φέρνει τα πιο δημοφιλή στοιχεία στην κεφαλή της λίστας, ενώ τα μη-δημοφιλή καταλήγουν στην ουρά της.

Βοήθεια: Η υλοποίηση του **ΑΤΔ Self-Organizing List** θα πρέπει να βασιστεί στην διεπαφή της κλάσης Vector που περιλαμβάνει μεθόδους όπως η add(), remove(),indexOf() και size().

Λύση:

```
public class SelfOrganizedList {
     private java.util.Vector vector;
     /**
      * Constructs a new SelfOrganizedList.
      * Method_signature: SelfOrganizedList<br>
      * Preconditions:<br>
      * -<br>
      * Postconditions:<br>
      * creates a new valid instance of SelfOrganizedList.
      */ // 1 point
     public SelfOrganizedList() {
          vector = new java.util.Vector();
     } //2 points
     /* transformers - mutative */
     /**
      *    Inserts    the    specified    object    in    this
SelfOrganizedList.
      * Method_signature: void_add_Object<br>
      * Preconditions:<br>
      * -<br>
      * Postconditions:<br>
      *   the   specified   object   is   inserted   to   the
SelfOrganizedList.
      * @param o object to be inserted.
      */  // 1 point
     public void add(Object o) {
          vector.add(o);
     } // 2 points
     /**
      *    Removes    the    specified    object    of    this
SelfOrganizedList.
      * Method_signature: boolean_remove_Object<br>
      * Preconditions:<br>
      * Object o is a valid instance<br>
      * Postconditions:<br>
      *   the   specified   object   is   removed   from   the
SelfOrganizedList.
      * @param o object to be removed.
      * @return true if the SelfOrganizedList contained the
object
      */ // 1 point
     public boolean remove(Object o) {
          return vector.remove(o);
     } // 2 points
     /* accessors */
     /**
      * Returns the size of this SelfOrganizedList.
      * Method_signature: int_size<br>
      * Preconditions:<br>
      * this is a valid instance of SelfOrganizedList<br>
      * Postconditions:<br>
      * returns the size of this SelfOrganizedList
      * @return the size of this SelfOrganizedList
      */  // 1 point
     public int size() {
```

```
            return vector.size();
        } // 2 points
        /**
         * Finds the specified object in this SelfOrganizedList.
         * If the specified object is found, it is moved to the
         * beginning of the SelfOrganizedList and returned.<br>
         * Method_signature: Object_find_Object<br>
         * Preconditions:<br>
         * this is a valid instance of SelfOrganizedList<br>
         * Postconditions:<br>
         * returns the specified object if found and moved to the
         * beggining of this SelfOrganizedList.
         * @param o the object to be founded.
         * @return the object if found, null otherwise
         */
        public Object find(Object o) {
            int index = vector.indexOf(o);
            if (index == -1) {
                return null;
            }
            Object foo = vector.remove(index);
            vector.add(0, foo);
            return foo;
        } // 6 points
}
```