Πανεπιστήμιο Κρήτης
Τμήμα Επιστήμης Υπολογιστών

ΗΥ-252 – Αντικειμενοστρεφής Προγραμματισμός
Βασίλης Χριστοφίδης

Επαναληπτική Εξέταση (3 ώρες)
Ημερομηνία: 28 Αυγούστου 2006

Ονοματεπώνυμο:
Αριθμός Μητρώου:

Άσκηση 1 (10 μονάδες)

Θεωρήστε τις παρακάτω δηλώσεις κλάσεων Java:

```
public class A {
  A() { System.out.println("A.A()"); }
  public void f(A) { System.out.println("A.f(A)"); }
  public void f(B) { System.out.println("A.f(B)"); }
  public void g(A) { System.out.println("A.g(A)"); }
};
public class B extends A {
  B() { System.out.println("B.B()"); }
  public void f(A) { System.out.println("B.f(A)"); }
  public void f(B) { System.out.println("B.f(B)"); }
  public void g(B) { System.out.println("B.g(B)"); }
  public static void h(A) { System.out.println("B.h(A)"); }
};
```

Ποιο είναι το αποτέλεσμα κάθε μιας από τις ακόλουθες εντολές Java που εκτελούνται στη σειρά: π.χ., εκτυπώνει (prints) xxx, εγείρει (throws) κατά την διάρκεια της εκτέλεσης μια εξαίρεση τύπου E, επιστρέφει κατά την διάρκεια της μετάφρασης ένα συντακτικό λάθος, κλπ.;

i) `A a = new A();`

Λύση:
```
prints A.A()
```

ii) `B b = new B();`

Λύση:
```
prints A.A()
prints B.B()
```

iii) `A ab = new B();`

Λύση:
```
prints A.A()
prints B.B()
```

iv) a.f(ab);

Λύση:
prints A.f(A)

v) a.g(b);

Λύση:
prints A.g(A)

vi) b.f(ab);

Λύση:
prints B.f(A)

Άσκηση 2 (10 μονάδες)

Σας δίνεται το παρακάτω πρόγραμμα Java:

```
class L1Exception extends Exception {
  public L1Exception() {}
  public L1Exception(String msg) { super(msg); }
}
class L2Exception extends L1Exception {
  public L2Exception() {}
  public L2Exception(String msg) { super(msg); }
}
public class ExceptionTest{
  public static void f(int x) throws
                                    L1Exception,L2Exception {
      if (x > 0) { throw new L1Exception(); }
      else { throw new L2Exception(); }
  }
  public static void g(int y) throws
                                    L2Exception,L1Exception {
      if (y <= 0) { throw new L1Exception(); }
      else { throw new L2Exception(); }
  }
  public static void main(String[] args) throws
                                    L1Exception,L2Exception {
      int a = Integer.parseInt(args[0]),
          b = Integer.parseInt(args[1]);
      try { f(a); g(b); }
      catch (L1Exception e) { System.out.println(e); }
      try { g(b); }
      catch(L2Exception e) { System.out.println(e); }
      finally {
         if (a <= 0) f(a);
         System.out.println("quitting");
      }
   }
}
```

Ποιο είναι το αποτέλεσμα της εκτέλεσης του κώδικα `ExceptionTest` για κάθε ένα από τα ακόλουθα ορίσματα: π.χ., εκτυπώνει (prints) xxx, εγείρει (throws) κατά την διάρκεια της εκτέλεσης μια εξαίρεση τύπου E, επιστρέφει κατά την διάρκεια της μετάφρασης ένα συντακτικό λάθος, κλπ.;

i) `1 1`

Λύση:
```
prints L1Exception
prints L2Exception
prints "quitting"
```

ii) `1 -1`

Λύση:
```
prints L1Exception
prints quitting
throws L1Exception
```

iii) `-1 1`

Λύση:
```
prints L2Exception
prints L2Exception
throws L2Exception
```

iv) `-1 -1`

Λύση:
```
prints L2Exception
throws L2Exception
```

Άσκηση 3 (10 μονάδες)

Ποια από τις δυο ακόλουθες λειτουργίες είναι αποδοτικότερη; Εξηγήστε γιατί.

```
static String a (String args []) {
   StringBuffer b = new StringBuffer();
   for (int a = 0; a < args.length; ++a)
        b.append(args[a]);
   return b.toString();
}

static String b (String args []) {
   String result = "";
   for (int a = 0; a < args.length; ++a)
        result += args[a];
   return result;
}
```

Λύση:
`a()` should be more efficient because `b()` keeps creating a new object for each concatenation.

Άσκηση 4 (20 μονάδες)

Ένας πίνακας συμβολοσειρών Java, `vehicle`, περιέχει αριθμούς μητρώου οχημάτων, και ένας δεύτερος πίνακας συμβολοσειρών, `owner`, περιέχει τις λεπτομέρειες των εγγεγραμμένων ιδιοκτητών οχημάτων. Μπορείτε να υποθέσετε ότι ο αριθμός οχημάτων είναι ο ίδιος με το μέγεθος κάθε πίνακα, ότι δεν υπάρχει κανένας διπλός αριθμός μητρώου, και ότι το στοιχείο `owner[i]`περιέχει πληροφορίες σχετικές με τον ιδιοκτήτη του `vehicle[i]`.

α) **(3 μονάδες)** Γράψτε μια μέθοδο Java `addValues(String[] vehicle, String[] owner, java.util.Map map)` για να εισάγετε όλους τους αριθμούς μητρώου και τις λεπτομέρειες των ιδιοκτητών στην απεικόνιση `map`, χρησιμοποιώντας τους αριθμούς μητρώου ως κλειδιά. Μπορείτε να υποθέσετε ότι ο `map` υπάρχει ήδη.

Λύση:
```
public void addValues(String[] vehicle,
                      String[] owner,
                      Java.util.Map map) {
   for (int i = 0; i < vehicle.length; i++)
   map.put(vehicle[i],owner[i]);
}
```

β) **(7 μονάδες)** Γράψτε σε μορφή ψευτο-κώδικα (χρησιμοποιώντας τις μεθόδους των διεπαφών Map και Set) μια μέθοδο η οποία χρησιμοποιεί έναν `Iterator` στο `map` που δημιουργήσατε στο προηγούμενο ερώτημα, για να μετρήσει πόσα (άλλα) οχήματα ανήκουν στον ιδιοκτήτη ενός οχήματος με έναν δεδομένο αριθμό μητρώου.

Λύση:
```
String owner = (String)map.get(v);
Iterator i = map.values().iterator();
int count = -1; /*(no marks deducted if count = 0)*/
while (i.hasNext())
  if (owner.equals(i.next()))
     count++;
return count;
```

γ) **(5 μονάδες)** Αναφέρετε έναν τρόπο με τον οποίο μπορεί να χρησιμοποιηθεί ένα `HashMap` χωρίς να απαιτείται η επαναϋλοποίηση της μεθόδου `hashCode()`.

Λύση:
Use only Strings as keys.
Or, use keys for which == is an appropriate equality test.

δ) **(5 μονάδες)** Ποια από τις παρακάτω δηλώσεις είναι αληθής για την κλάση `java.util.HashSet`;

1. Τα στοιχεία της συλλογής είναι διατεταγμένα.
2. Η συλλογή εγγυάται ότι τα στοιχεία της είναι αμετάβλητα (immutable).

3. Τα στοιχεία της συλλογής είναι μοναδικά.
4. Τα στοιχεία της συλλογής προσπελάζονται χρησιμοποιώντας ένα μοναδικό κλειδί.
5. Η συλλογή εγγυάται ότι τα στοιχεία της είναι συγχρονισμένα (synchronized).

Λύση:
3

Άσκηση 5 (20 μονάδες)

α) **(10 μονάδες)** Δώστε τον ορισμό μιας κλάσης `IntContainer` που υλοποιεί τη διεπαφή `Comparable`. Ο κατασκευαστής αντικειμένων του `IntContainer` πρέπει να δέχεται έναν ακέραιο `int`, βάση του οποίου θα γίνεται η ταξινόμηση των αντικείμενων του (μια εξαίρεση `RuntimeException` θα δημιουργείται όταν το αντικείμενο που δίνεται σαν είσοδο στην μέθοδο `compareTo()` δεν είναι ένα στιγμιότυπο του `IntContainer`). Η αναπαράσταση με μια συμβολοσειρά ενός αντικειμένου `IntContainer` πρέπει να είναι `"IntContainer:"` ακολουθούμενη από την τιμή του ακέραιου. Η κλάση `IntContainer` θα πρέπει επίσης να παρέχει μια μέθοδο `IntContainer difference(IntContainer i)`, η οποία επιστρέφει ένα νέο αντικείμενο τύπου `IntContainer` του οποίου η τιμή είναι η διαφορά των τιμών του τρέχοντος αντικειμένου `IntContainer` και του `i` που δίνεται στην είσοδο. Ένα παράδειγμα εκτέλεσης των μεθόδων της κλάσης σας δίνεται στην συνέχεια.

```
> a = new IntContainer(5);
> b = new IntContainer(10);
> System.out.println(b);
IntContainer: 10
> System.out.println(b.difference(a));
IntContainer: 5
> a.compareTo(b)
-5
```

Λύση:

```
class IntContainer implements Comparable {
  public int val;
  public IntContainer(int val) {
    this.val = val;
  }
  public int compareTo(Object obj) {
    return this.val - ((IntContainer)obj).val;
  }
  public IntContainer difference(IntContainer obj) {
    return new IntContainer(compareTo(obj));
  }
  public String toString() {
    return "IntContainer: " + val;
  }
}
```

β) **(10 μονάδες)** Υποθέστε ότι θέλουμε να έχουμε έναν εξειδικευμένο τύπο αντικειμένων `IntContainer` που απαγορεύει στο χρήστη την παροχή ενός αρνητικού ακέραιου αριθμού σαν όρισμά του.

- Ορίστε μια κλάση `NegativeIntException` που επεκτείνει την `RuntimeException`. Ο κατασκευαστής της θα πρέπει να δέχεται έναν ακέραιο σαν είσοδο. Θα πρέπει επίσης να υποσκελίζει την μέθοδο `String getMessage()`που κληρονομεί από την RuntimeException με μια καινούργια υλοποίηση που επιστρέφει `"Illegal Argument: "`, ακολουθούμενη από από την τιμή του ακέραιου.
- Ορίστε μια κλάση `PositiveIntContainer`, η οποία επεκτείνει την `IntContainer`. Ο κατασκευαστής της θα πρέπει να εγείρει μια εξαίρεση τύπου `NegativeIntException` εάν δέχεται σαν όρισμα έναν αρνητικό ακέραιο αριθμό. Η μέθοδος `difference(IntContainer i)`θα πρέπει τώρα να κατασκευάσει ένα αντικείμενο τύπου `PositiveIntContainer`.

Ένα παράδειγμα εκτέλεσης των μεθόδων της κλάσης σας δίνεται στην συνέχεια.

```
> a = new IntContainer(5);
> b = new PositiveIntContainer(10);
> c = new PositiveIntContainer(-1);
NegativeIntException: Illegal Argument: -1
  at PositiveIntContainer.<init>(Comparator.java:39) …
> System.out.println(a.difference(b)); //-5 is fine, since a
constructs IntContainers
IntContainer: -5
> System.out.println(b.difference(a));
IntContainer: 5
> System.out.println(b.difference(a).difference(a));
IntContainer: 0
> System.out.println(b.difference(a).difference(a).difference(a));
NegativeIntException: Illegal Argument: -5
  at PositiveIntContainer.<init>(Comparator.java:39)
```

Λύση:

```
class NegativeIntException extends RuntimeException {
  int i;
  public NegativeIntException(int i) {
    this.i = i;
  }
  public String getMessage() {
    return "Illegal Argument: " + i;
  }
}

class PositiveIntContainer extends IntContainer {
  public PositiveIntContainer(int val) {
    super(val);
    if (val < 0) throw new NegativeIntException(val);
  }
  public IntContainer difference(IntContainer obj) {
    return new PositiveIntContainer(compareTo(obj));
  }
}
```

Άσκηση 6 (20 μονάδες)

Ένας τύπος δεδομένων Συνδεμένη Λίστα υλοποιείται χρησιμοποιώντας την ακόλουθη κλάση Java `Slist` η οποία προσφέρει τις παρακάτω μεθόδους:

- `Slist(Object head, Slist tail)` κατασκευάζει μια καινούργια λίστα της οποίας το πρώτο στοιχείο είναι το `head` και τα υπόλοιπα στοιχεία της περιλαμβάνονται στην λίστα `tail`.
- `boolean isEmpty()` ελέγχει εάν η λίστα είναι κενή.
- `Object head()` επιστρέφει το πρώτο στοιχείο της λίστας και εγείρει μια εξαίρεση εάν η λίστα είναι κενή.
- `Slist tail()`επιστρέφει μια καινούργια λίστα που περιλαμβάνει όλα τα στοιχεία της λίστας εκτός από το πρώτο και εγείρει μια εξαίρεση εάν η λίστα είναι κενή.

Καμία από τις παραπάνω μεθόδους στιγμιοτύπων της κλάσης `Slist` δεν αλλοιώνει (mutative) τα αντικείμενα.

α) **(4 μονάδες)** Στα πλαίσια της υλοποίησης τύπων δεδομένων τι εννοούμε όταν λεμε ότι μία μέθοδος αλλοιώνει τα αντικείμενα (mutative) ή ότι τα αφήνει αναλλοίωτα (applicative);

Λύση:
A transformer method is:
mutative if it overwrites the old value with the new value
applicative if it returns the new value, without overwriting the old value.

β) **(6 μονάδες)** Χρησιμοποιώντας τις παραπάνω μεθόδους, γράψτε μια μέθοδο Java για να μετρήσετε πόσες φορές ένα αντικείμενο X εμφανίζεται σε μια δεδομένη λίστα.

Λύση:
```
private static int countOccurences(Object x, Slist
list) {
  int count=0;
  for (Slist s=list; !s.isEmpty(); s=s.tail())
  if (x.equals(s.head())) count++;
      return count;
}
```

γ) **(10 μονάδες)** Χρησιμοποιώντας τις παραπάνω μεθόδους, γράψτε μια αναδρομική μέθοδο Java για να ενώσετε μια λίστα στο τέλος μιας άλλης, και για να επιστρέψετε την παραγόμενη λίστα.

Λύση:
```
private static Slist appendtotail(Slist s1, Slist s2)
{
  if (s1.isEmpty()) return s2;
  if (s2.isEmpty()) return s1;
  return new
          Slist(s1.head(),appendtotail(s1.tail(),s2));
}
```

Άσκηση 7 (20 μονάδες)

Ποια είναι η πολυπλοκότητα (complexity) του χρόνου εκτέλεσης (στην χειρότερη περίπτωση) των παρακάτω μεθόδων; Δώστε μια σύντομη εξήγηση.

α)
```
int function1 (int n) {
  int sum = 0;
  for (int i=n; i<100*n; i++) {
      for( int j=0; j<i; j=j+2 ) {
          sum = sum + i*j*n;
      }
  }
  return sum;
}
```

Λύση:
function1(n) is $O(n^2)$.

β)
```
int function2 (int n)  {
      int sum = 0;
      while(n>=1) {
              sum = n*n*n;
         n = n/2;
      }
      return sum;
}
```

Λύση:
Function2(n) is $O(\log n)$.

γ)
```
int function3 (int n) {
      int val = 0;
      for (int i=1; i<=n; i++) {
          int j=i;
          while( j>=1 ) {
               val = val + j;
               j = j/2;
          }
      }
      return val;
}
```

Λύση:
Function3(n) is $O(n \log n)$.

δ)
```
int function4 (int n) {
      int sum = 0;
      for (int i=n*n; i<n*n + n; i++) {
          for( int j=0; j<9999; j=j+2 ) {
              sum = sum + j*n;
          }
          for( int k=n; k<n+1000; k=k+2 ) {
              sum = sum + k*n;
          }
      }
      return sum;
}
```

Λύση:
Function4(n) is $O(n)$.