Πανεπιστήμιο Κρήτης
Τμήμα Επιστήμης Υπολογιστών

ΗΥ-252 – Αντικειμενοστρεφής Προγραμματισμός
Βασίλης Χριστοφίδης

Τελική Εξέταση (3 ώρες)
Ημερομηνία: 1 Φεβρουαρίου 2010

Ονοματεπώνυμο:
Αριθμός Μητρώου:

Άσκηση 1(10 μονάδες) Κληρονομικότητα Κλάσεων&Τύποι Αντικειμένων
Υποθέστε μια μέθοδο `m()` η οποία κληρονομείται σε μια κλάση `B` από μια κλάση `A`. Μας ενδιαφέρει να γνωρίζουμε πόσες φορές η μέθοδος `m()` εκτελείται από ένα αντικείμενο του οποίου ο δυναμικός τύπος είναι `A` και πόσες φορές εκτελείται από ένα αντικείμενο του οποίου ο δυναμικός τύπος είναι `B`. Υλοποιήστε την μέθοδο `m()` της κλάσης `A` έτσι ώστε να κρατάμε την εν λόγω πληροφορία. Ποιά πεδία χρειάζεται να ορίσετε στην κλάση A;

```
public class B extends A { ... }
public class A {
```

Λύση:
```
  static int aCount = 0;
  static int bCount = 0;

  public void m() {
```

Λύση:
```
    if (this instanceof B) bCount++;
    else aCount++;
  }
}
```

Άσκηση 2 (22 μονάδες) Βασική Λειτουργικότητα Αντικειμένων

Η μέθοδος `compareTo()` της κλάσης `String` συγκρίνει δύο συμβολοσειρές σύμφωνα με την λεξικογραφική διάταξη. Για παράδειγμα, η συμβολοσειρά "`log10.txt`" προηγείται της συμβολοσειράς "`log2.txt`", γιατί το '1' λεξικογραφικά προηγείται του '2'. Ωστόσο, συχνά είναι χρήσιμο να διατάσσουμε την συμβολοσειρά "`log2.txt`" πριν από την "`log10.txt`". Ονομάζουμε αυτή την διάταξη συμβολοσειρών `FileNameOrder`. Απαντήστε στα παρακάτω ανεξάρτητα ερωτήματα της άσκησης.

α) **(14 μονάδες)** Μας ενδιαφέρει μια κλάση `FileNameOrder` που υλοποιεί την διεπαφή `Comparator`, στιγμιότυπα της οποίας μας επιτρέπουν την σύγκριση δύο αντικειμένων `File` με βάση το όνομά τους σύμφωνα με την διάταξη συμβολοσειρών που περιγράφηκε παραπάνω.

```
interface Comparator{
  // Compares its two arguments for order.
  // the integer returned by compare should be 0 if o1 is
  // equal to o2, -1 if o1 is before o2, and 1 if o1 is
  // after o2.
  int compare(Object o1, Object o2)
}
public class FileNameOrder implements Comparator {
...
}
```

Υλοποιήστε την μέθοδο `compare()` της κλάσης `FileNameOrder`. Στην υλοποίηση σας μπορείτε να χρησιμοποιήσετε την μέθοδο `String[] split(String s)` η οποία παίρνει σαν όρισμα εισόδου το όνομα ενός αρχείου και επιστρέφει ένα πίνακα συμβολοσειρών με τα 3 συστατικά μέρη του ονόματος. Για παράδειγμα, στην κλήση
`String[] result = split("log2.txt");`
το `result[0]` περιέχει το μη αριθμητικό κομμάτι του ονόματος (δηλ. το "`log`"), το `result[1]` περιέχει το αριθμητικό κομμάτι του ονόματος (δηλ. το "2") ή η κενή συμβολοσειρά όταν δεν υπάρχει αριθμός και το `result[2]` περιέχει την κατάληξη του ονόματος (δηλ. το "`.txt`").

```
public int compare(Object o1, Object o2){
```

Λύση:

```
String[] s1 = split(((File) o1).getName());
String[] s2 = split(((File) o2).getName());
int c=s1[0].compareTo(s2[0]); //compare the strings
if (c!=0) //if they are equal, they are equal
  return c;
c = s1[2].compareTo(s2[2]);
if (c!=0) //if they have different last name, they
          //are different
  return c;
if (s1[1].equals(""))
    s1[1] = "0";//if one of them did not have a
                //number, add it
if (s2[1].equals(""))
    s2[1] = "0";
c = Integer.parseInt(s1[1]) -
    Integer.parseInt(s2[1]); //compare the numbers.
return c;
}
```

β) **(8 μονάδες)** Χρησιμοποιήστε ένα αντικείμενο σύγκρισης `FileNameOrder` για να υλοποιήστε την παρακάτω μέθοδο ταξινόμησης:
`String[] sortFilesIn(File directory)`
η οποία παίρνει σαν όρισμα εισόδου έναν κατάλογο (directory) αρχείων και επιστρέφει έναν πίνακα με τα ονόματα των αρχείων του καταλόγου ταξινομημένα σύμφωνα με την διάταξη που υλοποιεί η `FileNameOrder`. Όταν το όρισμα εισόδου δεν είναι κατάλογος ο επιστρεφόμενος πίνακας ονομάτων είναι `null`.

Μπορείτε να θεωρήσετε σαν εκ των προτέρων συνθήκη της μεθόδου ότι ο κατάλογος δεν περιέχει υπο-καταλόγους.. Στην υλοποίηση σας μπορείτε να χρησιμοποιήσετε την μέθοδο `void Arrays.sort(Object[] a, Comparator c)` η οποία ταξινομεί τα αντικείμενα του πίνακα a χρησιμοποιώντας το αντικείμενο τύπου `Comparator` που δίνεται σαν όρισμα.

```
static String[] sortFilesIn(File directory){
```

Λύση:

```
if (!directory.isDirectory()) return null;
File[] files = directory.listFiles();
Arrays.sort(files, new FileNameOrder());
String[] res = new String[files.length];
for(int i=0;i<files.length;i++)
  res[i]=files[i].getName();
return res;
// There is a small trick, in that the method should
// return a list of sorted filenames, not Files.
```

```
}
```

Βοήθεια: Στην απάντησή σας μπορεί να χρειαστείτε κάποιες από τις παρακάτω μεθόδους της κλάσης `File`:

- `boolean isDirectory()`: Επιστρέφει true αν το συγκεκριμένο `File` είναι κατάλογος (φάκελος). Αλλιώς false.
- `boolean isFile()`: Επιστρέφει true αν το συγκεκριμένο `File` είναι απλό αρχείο, αλλιώς false.
- `File[] listFiles()`: Επιστρέφει τον πίνακα από τα `Files` που περιέχει ο συγκεκριμένος κατάλογος.
- `String[] list()`: Επιστρέφει τον πίνακα από τα ονόματα των `Files` που περιέχει ο συγκεκριμένος κατάλογος.
- `String getName()`: Επιστρέφει το όνομα του συγκεκριμένου `File`.

Άσκηση 3 (20 μονάδες) Χειρισμός Πλαισίου Συλλογών Αντικειμένων

Για κάθε μια από τις παρακάτω περιπτώσεις, γράψτε τον ακριβή τύπο της πιο αποδοτικής συλλογής δεδομένων που χρειαζόμαστε για την αποθήκευση των δεδομένων. Μπορείτε να χρησιμοποιήσετε οποιοδήποτε συνδυασμό των γενικών εκδόσεων (generic) των διεπαφών συλλογών `Map`, `Set`, και `List`. Χρησιμοποιήστε τους γενικούς τύπους που ταιριάζουν περισσότερο στα δεδομένα (π.χ. `Title`, `Artist`, `Price`, κλπ.) του κάθε ερωτήματος (π.χ. `List<Artist>`, και όχι απλά `List`) καθώς και φωλιασμένους (nested) τύπους συλλογών (π.χ. `List<Set<String>>`).
Σε περίπτωση που αντιμετωπίζετε δυσκολίες στην ακριβή σύνταξη των γενικών τύπων απαντήστε περιγραφικά.
Μπορείτε να υποθέσετε ότι όλες οι κλάσεις που αναπαριστούν τα δεδομένα (π.χ. `Title`, `Artist`, `Price`, κλπ.) κάθε ερωτήματος είναι ήδη ορισμένες και ότι όλες υλοποιούν κατάλληλα τις μεθόδους `equals()` and `hashCode()` ώστε τα αντικείμενά τους να αποθηκεύονται σε σύνολα (`Set`) ή κλειδιά (key) απεικονίσεων (`Map`).

Παράδειγμα: Υποθέστε ότι μας ενδιαφέρει η οργάνωση των παιχτών ποδοσφαιρικών ομάδων της Premier League έτσι ώστε να μπορούμε να βρούμε τους ποδοσφαιριστές δεδομένης της ομάδας τους. Ο γενικός τύπος της πιο αποδοτικής συλλογής για αυτή την λειτουργικότητα είναι η `Map<Team, Set<Player>>` (σε αυτή την λύση υποθέσαμε ότι ένα αντικείμενο `Team` δεν περιέχει το σύνολο των παιχτών της ομάδας).: Εναλλακτικά, μπορείτε να γράψετε ότι χρειαζόμαστε μια συλλογή `Map` με *κλειδί* τύπου `Team` και *τιμή* τύπου Set (συνόλου) με αντικείμενα τύπου `Player`.

(α) **(4 μονάδες)** Ένα δισκοπωλείο στον Παγκόσμιο Ιστό θέλει να κρατήσει για κάθε καλλιτέχνη (`Artist`) τους τίτλους (`Title`) όλων των δίσκων ή CD του καθώς και την τιμή (`Price`) τους.

Λύση:
```
Map<Artist, Map<Title, Price>>
```

(β) **(4 μονάδες)** Το ηλεκτρονικό δισκοπωλείο θέλει να κρατήσει τον κατάλογο ενδεχόμενων αγορών ενός πελάτη (`Customer`) με το σύνολο των (`Title`) τίτλων δίσκων ή CD που τον ενδιαφέρουν.

Λύση:
```
Map<Customer, Set<Title>> ή
Map<Customer, List<Title>>
```

(γ) **(4 μονάδες)** Το ηλεκτρονικό δισκοπωλείο θέλει να κρατήσει ένα πίνακα με τους πιο δημοφιλείς τίτλους δίσκων ή CD: ποιοι τίτλοι βρίσκονται στο 0-10% των πωλήσεων, ποιοι στο 10-20%, ποιοι στο 20-30% κλπ. Με άλλα λόγια μας ενδιαφέρει μια δομή που επιτρέπει την ταξινόμηση των τίτλων (`Title`) στο αντίστοιχο μερίδιο των πωλήσεων.

Λύση:
```
List<Set<Title>> ή
Map<Integer, Set<Title>>
```

(δ) **(4 μονάδες)** Το ηλεκτρονικό δισκοπωλείο θέλει επίσης να κρατήσει για κάθε πελάτη (`Customer`) άλλους πελάτες που έχουν τα ίδια μουσικά ενδιαφέροντα με αυτόν.

Λύση:
```
Map<Customer, Set<Customer>>
```

(ε) **(4 μονάδες)** Ο αλγόριθμος συστάσεων (recommendation) που χρησιμοποιεί το ηλεκτρονικό δισκοπωλείο παράγει αυτόματα μουσικά προγράμματα (Playlists), δηλαδή τίτλους (`Title`) ορισμένων μουσικών κομματιών σε μια σειρά εκτέλεσης, με βάση τα ενδιαφέροντα ενός πελάτη. Το ηλεκτρονικό δισκοπωλείο θέλει να κρατήσει αυτές τις συστάσεις έτσι ώστε να εξασφαλίζεται μια γρήγορη

ανάκτηση των μουσικών προγραμμάτων (Playlist) και του περιεχομένου τους (Title) για κάθε πελάτη (Customer).

> **Λύση:**
> `Map<Customer, Map<Playlist, List <Title>>>` ή
> `Map<Customer, Map<String, List <Title>>>`

Άσκηση 4 (18 μονάδες) Σαρωτές Συλλογών Αντικειμένων

Οι επαναλήπτες (Iterators) στην Java χρησιμοποιούνται για να σαρώνουν τα στοιχεία μιας συλλογής αντικειμένων ενθυλακώνοντας τις λεπτομέρειες της υλοποίησης της. Σε αυτή την άσκηση θα πρέπει να υλοποιήσετε έναν επαναλήπτη (MergeIterators) ο οποίος συγχωνεύει τα αντικείμενα που επιστρέφουν δύο άλλοι επαναλήπτες συλλογών. Ο επαναλήπτης MergeIterators υλοποιεί την διεπαφή Iterator από την οποία έχουμε παραλείψει την μέθοδο remove():

```
public interface Iterator {
     /* pre: none
      * post: return true if the iterator is not empty */
     boolean hasNext();

     /* pre: hasNext()
      * post: return the next element of the iterator */
     Object next();
}
class MergeIterators implements Iterator {
     Iterator a,b;
     MergeIterators (a Iterator, b Iterator) {
     ...}
     boolean hasNext() { ... }
     Object next(){ ... }
}
```

Η συγχώνευση δύο επαναληπτών a και b στον c θα γίνει με τον παρακάτω τρόπο:

- Ο συγχωνευμένος επαναλήπτης c έχει στοιχεία μόνο όταν τουλάχιστον ένας από τους επαναλήπτες a ,b έχει και άλλα στοιχεία.
- Το επόμενο στοιχείο στον συγχωνευμένο επαναλήπτη c είναι:
    - ένα από τα επόμενα στοιχεία των a και b όταν οι επαναλήπτες έχουν και άλλα στοιχεία. Η επιλογή γίνεται με τυχαίο τρόπο χρησιμοποιώντας την στατική μέθοδο random() του πακέτου java.lang.Math που επιστρέφει ένα double από 0.0 μέχρι 1.0.
    - το επόμενο στοιχείο του a όταν ο επαναλήπτης b δεν έχει άλλα στοιχεία.
    - το επόμενο στοιχείο του b όταν ο επαναλήπτης a δεν έχει άλλα στοιχεία.

α) **(4 μονάδες)** Δώστε την αμετάβλητη συνθήκη της κλάσης `MergeIterators` την οποία θα πρέπει να ικανοποιεί η υλοποίηση που θα κάνετε στο ερώτημα β.

Λύση:
The invariant is already given, it is the two bullet points with the merging conditions of iterators a and b into c.

β) **(14 μονάδες)** Υλοποιήστε τον κατασκευαστή και τις μεθόδους `hasNext()` και `next()` της κλάσης `MergeIterators`.

Λύση:
```
MergeIterators(Iterator a, Iterator b){
     this.a = a;
     this.b = b;
} //  2 points

public boolean hasNext(){
     return a.hasNext() || b.hasNext();
} //  2 points
public Object next(){
     if (a.hasNext() )
          if (b.hasNext())
               if (Math.random() < 0.5 )
                    return a.next();
               else
                    return b.next();
          else
               return a.next();
     else
          return b.next();
} // 10 points
```

Άσκηση 5 (23 μονάδες) Γενικοί Τύποι στην Java

Σας δίνονται οι παρακάτω δυο κλάσεις Java, οι οποίες χρησιμοποιούνται στην υλοποίηση ενός τηλεφωνικού καταλόγου. Η κλάση `PhoneNumber` αναπαριστά διεθνείς τηλεφωνικούς αριθμούς ενώ η `Address` αναπαριστά διευθύνσεις.

```
public class PhoneNumber {
     int country; // Invariant country > 0
     int city; // Invariant city > 0
     int local; // Invariant local > 0
     public PhoneNumber(int country, int city, int local) {
          this.country = country;
          this.city = city;
          this.local = local;
     }
}
public class Address {
     String address;
     public Address(String address) {
```

```
            this.address = address;
        }
        public String toString () {
            return address;
        }
}
```

Για την υλοποίηση της λειτουργίας αναζήτησης του τηλεφωνικού καταλόγου βασιζόμαστε την γενική εκδοχή της δενδρικής υλοποίησης απεικονίσεων `TreeMap<K,V>`. Σύμφωνα με το συμβόλαιο της δομής αυτής στα Java APIs μπορούμε να χρησιμοποιήσουμε έναν κατασκευαστή με την ακόλουθη υπογραφή: `TreeMap(Comparator<? super K> c)` ο οποίος δημιουργεί μια καινούργια κενή απεικόνιση (empty map) ταξινομημένη σύμφωνα με την διάταξη που υλοποιεί το αντικείμενο σύγκρισης `c`. Σημειώστε ότι και για το όρισμα εισόδου του κατασκευαστή χρησιμοποιείται μια έκφραση γενικών τύπων. Πιο συγκεκριμένα, η `Comparator<T>` αποτελεί την γενική εκδοχή της απλής διεπαφής `Comparator` η οποία προσφέρει την μεθόδο: `int compare(T o1, T o2)`. Ως συνήθως, το συμβόλαιο της μεθόδου `compare()` δηλώνει ότι επιστρέφεται ένας αρνητικός αριθμός, μηδέν ή ένας θετικός αριθμός ανάλογα με το αν το `o1` είναι μικρότερο, ίσο ή μεγαλύτερο του `o2`.

α) **(5 μονάδες)** Εξηγήστε την χρησιμότητα της έκφρασης γενικών τύπων `<? super K>` σαν όρισμα της διεπαφής `Comparator`, η οποία με την σειρά της είναι όρισμα στον κατασκευαστή της `TreeMap<K,V>`.

Λύση:
If a tree map contains objects of a subclass of K, the comparator still has to derive information about their equality, or inequality. This is why the Comparator has to work on any subclass of K.

β) **(13 μονάδες)** Στην ακόλουθη κλάση `CompForPhoneNumbers`, υλοποιήστε τις μεθόδους `compare(T o1, T o2)` και `isequal(T o1, T o2)` για αντικείμενα τύπου `PhoneNumbers`. Στην υλοποίησή σας μπορείτε να χρησιμοποιήσετε βοηθητικές μεθόδους σύγκρισης ακέραιων τιμών που αναπαριστούν τα συστατικά μέρη ενός τηλεφωνικού αριθμού (`country,city,local`).

```
class CompForPhoneNumbers implements Comparator<PhoneNumber>{
```

Λύση:

```
 private int cmp(int x, int y) {
   //return x - y; or:
   if (x < y) {return -1;}
   else {if (x > y) {return +1;}
         else {return 0;}
        }
 } // 4 points
 public int compare(PhoneNumber p1, PhoneNumber p2) {
   int country = cmp(p1.country, p2.country);
   if (country != 0) {return country;}
```

```
    else {
        int city = cmp(p1.city, p2.city);
        if (city != 0) {return city;}
        else return (cmp(p1.local, p2.local));
    }
  } // 7 points
  boolean isequal(PhoneNumber p1, PhoneNumber p2) {
    return (compare(p1, p2) == 0);
  } // 2 points
}
```

γ) **(5 μονάδες)** Δηλώστε και αρχικοποιήστε ένα καινούργιο στιγμιότυπο του γενικού τύπου TreeMap, το οποίο είναι κατάλληλο για την υλοποίηση της λειτουργίας αναζήτησης του τηλεφωνικού καταλόγου δηλ. της ανεύρεσης διευθύνσεων με βάση έναν αριθμό τηλεφώνου.

Λύση:
```
TreeMap<PhoneNumber,Address> dictionary =
                    new TreeMap<PhoneNumber,Address>(c);
```

Άσκηση 6 (20 μονάδες) Θεωρητική Ανάλυση Κώδικα & Πολυπλοκότητα

Ποια είναι η πολυπλοκότητα (complexity) του χρόνου εκτέλεσης (στην χειρότερη περίπτωση) των παραπάνω στατικών μεθόδων; Δώστε μια σύντομη εξήγηση της τάξης αύξησης του χρόνου εκτέλεσης καθώς το N αυξάνεται.

α)
```
public static double f1(double[][] array) {
     int N = array[0].length;
     double b = Math.random();
     for(int i = 0; i < 3; i++)
          for(int j = N - 1; j > 0; j--)
               b *= N * N * array[i][j];
     return b ;
}
```

Λύση:
O(N)

β)
```
public static int f2(int[] arr) {
     // N is arr.length
     int count = arr.length -1;
     while(count >0) {
          if(arr[count] < 0)
               count -= 7;
          count = count - arr.length / 50;
     }
     return count;
}
```

Λύση:
O(1) (For large N the loop iterates ~50 times)

γ)
```
public static boolean f3(double[][] array) {
     //N is array.length
     for(int i=0;i<array.length;i++)
          for(int j=0;j<i;j++)
               if(array[i][j] == 0) return true;
     return false;
}
```

Λύση:
O(N^2)

δ)
```
public boolean f4(int[] arr, int lo, int hi) {
     //N=initial value of (hi-lo+1) of 1st activation of f4
     if(lo == hi) return false;
     if(arr[hi] * arr[lo]>10) return true;
     return f4(arr,lo+1 , hi);
}
```

Λύση:
O(N)

ε)
```
public int f5(int[] array, int x) {
     // N = array.length
     int m = array.length-1;
     while(array[m] > x && m>0)
          m=m/2;
     return m;
}
```

Λύση:
O(lg N)