Πανεπιστήμιο Κρήτης
Τμήμα Επιστήμης Υπολογιστών

ΗΥ-252 – Αντικειμενοστρεφής Προγραμματισμός
Βασίλης Χριστοφίδης

Επαναληπτική Εξέταση (3 ώρες)
Ημερομηνία: 30 Αυγούστου 2010

Θέμα 1	Θέμα 2	Θέμα 3	Θέμα 4	Θέμα 5	Σύνολο
/10	/16	/10	/12	/55	/103

Ονοματεπώνυμο:
Αριθμός Μητρώου:

Άσκηση 1 (10 μονάδες) Κατανόηση Αντικειμενοστρεφούς Κώδικα

Τι θα τυπωθεί στην στανταρ έξοδο του παρακάτω προγράμματος;

```
public class Test {
     public static void main(String[] args) {
          A a = new A();
          System.out.println(a.x);
          System.out.println(a.m());
          B b = new B();
          System.out.println(b.x);
          System.out.println(b.m());
          a = new B();
          System.out.println(a.x);
          System.out.println(a.m());
          b = (B)a;
          System.out.println(b.x);
          System.out.println(b.m());
     }
}
class A {
     int x;
     A() { this(1); }
     A(int x) { this.x = x; }
     int m() { return x; }
}
class B extends A {
     int x;
     B() { this(2); }
     B(int x) {
          super(x+1);
          this.x = super.x + 1;
     }
     int m() { return x; }
}
```

Λύση:
11443444

Άσκηση 2 (16 μονάδες) Βασική λειτουργικότητα Αντικειμένων

Υλοποιήστε τις ακόλουθες μεθόδους της κλάσης `Presents`:

- (**2 μονάδες**) Τον κατασκευαστή (constructor) που παίρνει σε μια ακέραιη παράμετρο την μέγιστη χωρητικότητα του πίνακα συμβολοσειρών (Strings) με τα δεδομένα της κλάσης. Δημιουργήστε ένα καινούργιο πίνακα αυτής της χωρητικότητας για την μεταβλητή στιγμιοτύπων `data`.
- (**4 μονάδες**) Την μέθοδο στιγμιοτύπων `add()` που παίρνει σε μια παράμετρο συμβολοσειράς το όνομα που θα προστεθεί στα δεδομένα της κλάσης και επιστρέφει μια boolean τιμή. Υποθέτοντας μη μηδενικές (non-null) συμβολοσειρές, εάν υπάρχει διαθέσιμος χώρος στον πίνακα η μέθοδος προσθέτει το όνομα στον πίνακα και επιστρέφει αληθές, αφού πρώτα ανανεώσει τη μεταβλητή μεγέθους του πίνακα (size). Διαφορετικά επιστρέφει ψευδές.
- (**10 μονάδες**) Την μέθοδο στιγμιοτύπων `equals()` που παίρνει σε μια παράμετρο τύπου `Object` μια αναφορά σε ένα αντικείμενο και επιστρέφει μια boolean τιμή. Επιστρέφει αληθές αν και μόνο αν το αντικείμενο εισόδου είναι επίσης στιγμιότυπο της κλάσης `Presents` και περιέχει την ίδια λίστα ονομάτων και με την ίδια σειρά. Διαφορετικά επιστρέφει ψευδές.

```
public class Presents {
     // valid strings stored in 0... size-1
     private String[] data;
     // initially 0, data.length once array is full
     private int size;
     public int getSize() {return size;}
     public String get(int i) {return data[i];}
     public Presents(int capacity) {
```

Λύση:

```
          data = new String[capacity];
     }
     public boolean add(String s) {
```

Λύση:

```
          if(size == data.length) return false;
          data[size++] = s;
          return true;
     }
     public boolean equals(Object other) {
```

Λύση:

```
          if(other instanceof Presents) {
            Presents p = (Presents) other;
            if(p.size != this.size) return false;
            for(int i=0;i < size; i++) {
               if(!get(i).equals(p.get(i))) return false;
            }
          return true;
          }
          return false;
     }
}
```

Άσκηση 3 (10 μονάδες) // Χειρισμός Πινάκων και Αντικειμένων

Σας δίνονται οι παρακάτω κλάσεις `Movie` και `Actor` των οποίων τα αντικείμενα αναπαριστούν ηθοποιούς και τις ταινίες στις οποίες συμμετέχουν. Για κάθε ταινία αποθηκεύουμε τον τίτλο (`title`) και την λίστα των ηθοποιών (`actors`) της. Ένας ηθοποιός μπορεί να εμφανίζεται σε πολλές ταινίες και κατά συνέπεια να αναφέρεται από πολλούς πίνακες `actors` διαφορετικών αντικειμένων της κλάσης `Movie`. Αρχικά η τιμή της boolean μεταβλητής στιγμιοτύπων `printed` είναι ψευδής. Υλοποιήστε μια αναδρομική μέθοδο `visit()` στην κλάση `Actor` η οποία τυπώνει τα ονόματα των ηθοποιών σε ξεχωριστές γραμμές και επιστρέφει το πλήθος των διακριτών ηθοποιών σε μια ακέραια τιμή. Σημειώστε ότι κάθε ηθοποιός τυπώνεται μόνο μία φορά ενώ η μεταβλητή `printed` γίνεται αληθής. Για ευκολία υποθέστε ότι όλες οι μεταβλητές στιγμιοτύπων έχουν αρχικοποιηθεί σωστά, οι αναφορές αντικειμένων είναι μη μηδενικές (non-null) και ο γράφος των αναφορών των αντικειμένων `Movie` και `Actor` είναι συνδεδεμένος.

```
public class Movie {
     public String title;
     public Actor[] actors;
}

public class Actor {
     private String name;
     private Movie[] movies;
     private boolean printed;
     // constructor not shown.

     int visit() {
```

Λύση:

```
          if(printed) return 0;
          printed = true;
          System.out.println(name);
          int count =1;
          for(int i =0;i < movies.length; i++)
              for(int j=0;j<movies[i].actors.length;j++)
                  count += movies[i].actors[j].visit();
          return count;
     }
}
```

Άσκηση 4 (12 μονάδες) Εξαιρέσεις και Διόρθωση Κώδικα

Διορθώστε την παρακάτω μέθοδο `cat()`έτσι ώστε να μεταγλωττίζεται επιτυχώς:

```
public static void cat(File file) {
    RandomAccessFile input = null;
    String line = null;

    try {
        input = new RandomAccessFile(file, "r");
        while ((line = input.readLine()) != null) {
```

```
            System.out.println(line);
        }
        return;
    } finally {
        if (input != null) {
            input.close();
        }
    }
}
```

Βοήθεια: Στην εκτέλεση του παραπάνω κώδικα μπορούν να εμφανιστούν εξαιρέσεις τύπου **`FileNotFoundException`** και **`IOException`**.

Λύση:
The code to catch exceptions is shown in bold:

```
  public static void cat(File file) {
    RandomAccessFile input = null;
    String line = null;
    try {
         input = new RandomAccessFile(file, "r");
         while ((line = input.readLine()) != null) {
                System.out.println(line);
         }
         return;
     } catch(FileNotFoundException fnf) {
         System.err.format("File:%s not found%n",
file);
     } catch(IOException e) {
              System.err.println(e.toString());
    } finally {
       if (input != null) {
           try {
                input.close();
             } catch(IOException io) {
             }
       }
    }
```

Άσκηση 5 (55 μονάδες) Σχεδίαση και Υλοποίηση ΑΤΔ

Ένας πωλητής οπωροκηπευτικών ενδιαφέρεται να αγοράσει ένα πρόγραμμα διαχείρισης των προϊόντων του: όνομα εμπορεύματος, όνομα προμηθευτή κάθε προϊόντος, τιμή αγοράς (purchase) και πώλησης (sales) ανά κιλό, κλπ.

α) **(7 μονάδες)** Δώστε την προδιαγραφή του ΑΤΔ Vegetable δηλ. τις υπογραφές όλων των βασικών μεθόδων (παραμέτρους εισόδου και επιστρεφόμενες τιμές) καθώς και το είδος τους (constructors, assessors, transformers). Οι βασικές μέθοδοι είναι: δημιουργία ενός αντικειμένου παίρνοντας σαν είσοδο το όνομα του οπωροκηπευτικού, ανάγνωση και αλλαγή του ονόματος του προμηθευτή,

ανάγνωση και αλλαγή της τιμής αγοράς (purchase) και πώλησης (sales) ανά κιλό.

Λύση:

```
//createvegetable with given name
 createVegetable(inname: String)
//retrieves the supplier
getSupplier():string {accessor}
//change supplier to name
setSupplier(inname:string) {transformer}
//retrieves the purchase price
getPurchasePrice():integer {accessor}
//change purchase price to pp
setPurchasePrice(inpp:integer) {transformer}
//retrieves the sales price
getSalesPrice():integer {accessor}
//change purchase price to sp
setSalesPrice(insp:integer) {transformer}
```

β) **(7 μονάδες)** Υλοποιήστε τον ΑΤΔ Vegetable σε μια κλάση Java. Βεβαιωθείτε ότι η υλοποίησή σας δεν παραβιάζει την προδιαγραφή που δώσατε στο προηγούμενο ερώτημα (π.χ. σεβόμενοι τις αρχές της ενθυλάκωσης). Αναπαραστήστε τα ονόματα (name) με συμβολοσειρές (String) και τις τιμές (price) με ακεραίους (int) που δηλώνουν cents.

```
class Vegetable {
```

Λύση:

```
    private String name, supplier;
    private int salesPrice, purchasePrice;
    public Vegetable (String n) {
      name = n; }
    public String getSupplier( ) {
      return supplier; }
    void setSupplier(String s) {
      supplier = s; }
    public int getPurchasePrice( ) {
      return purchasePrice; }
    public void setPurchasePrice (int pp) {
      purchasePrice = pp; }
    public int getSalesPrice( ) {
      return salesPrice; }
    public void setSalesPrice (int sp) {
      salesPrice = sp ; }
}
```

(γ) **(4 μονάδες)** Σας δίνεται η παρακάτω διεπαφή Java:

```
interface ProductWithProfit {
    public int profit( );
}
```

Υλοποιήστε μια κλάση VegetableWithProfit η οποία επαναχρησιμοποιεί την υλοποίηση των μεθόδων της κλάσης Vegetable που δώσατε στο προηγούμενο ερώτημα και προσφέρει επίσης μια υλοποίηση της διεπαφής ProductWithProfit. Σημειώστε ότι το κέρδος που επιστρέφει η μέθοδος profit() υπολογίζεται σαν την διαφορά μεταξύ της τιμής πώλησης και αγοράς ανά κιλό ενός προϊόντος.

Λύση:

```
class VegetableWithProfit extends Vegetable implements
                                        ProductWithProfit{
public int profit() {
     return getSalesPrice() – getPurchasePrice() ;
 }
}
```

(δ) **(13 μονάδες)** Μας ενδιαφέρει η ταξινόμηση των στιγμιότυπων της κλάσης VegetableWithProfit με βάση την τιμή κέρδους που επιστρέφει η μέθοδος profit(). Για αυτό τον σκοπό μπορούμε να χρησιμοποιήσουμε την μέθοδο sort της κλάσης Collections από το πλαίσιο διαχείρισης συλλογών της Java (Java Collections Framework - JCF):
static <T> void sort(List<T> list, Comparator<? super T> c)
η οποία ταξινομεί την λίστα εισόδου List<T> σε αύξουσα διάταξη (mutable transformer). Θυμηθείτε ότι η διεπαφή Comparator<T> απαιτεί μια μέθοδο
public int compare (T o1, T o2) που συγκρίνει την διάταξη των παραμέτρων εισόδου. Η μέθοδος επιστρέφει έναν αρνητικό ακέραιο, μηδέν, ή έναν θετικό ακέραιο εάν η πρώτη παράμετρος είναι μικρότερη, ίση ή μεγαλύτερη από την δεύτερη. Υλοποιήστε τη στατική μέθοδο sortProductWithProfitList() η οποία ταξινομεί σε αύξουσα διάταξη κέρδους μια λίστα με στιγμιότυπα της κλάσης ProductWithProfit.
Δώστε προσοχή στην υπογραφή της μεθόδου και σε τυχόν βοηθητικές κλάσεις που πρέπει να ορίσετε ώστε να μπορέσετε να χρησιμοποιήστε για την ταξινόμηση την μέθοδο sort() της JCF κλάσης Collections.

```
static void sortProductWithProfitList
                        (List<ProductWithProfit> list){
```

Λύση:

```
Collections.sort(list,newProductWithProfitComparator())
;
}
class ProductWithProfitComparator
              implements Comparator<ProductWithProfit>{
public compare(ProductWithProfitp1,ProductWithProfitp2)
{
 if(p1.getProfit()<p2.getProfit())
    return –1;
 else if(p1.getProfit()==p2.getProfit())
    return 0;
else return 1;
```

```
}
}
```

(ε) **(12 μονάδες)** Υποθέστε τώρα ότι μας ενδιαφέρει η αποθήκευση αντικειμένων της κλάσης Product σε μια κατάλληλη δομή. Για την εύκολη αναγνώριση των προϊόντων η κλάση Product υλοποιεί την διεπαφή KeyedItem<Integer>: η κλήση p.getKey() επιστρέφει έναν ακέραιο αριθμό με τον κωδικό ενός προϊόντος p:

```
class Product implements KeyedItem<Integer> {
 . . .
}
interface KeyedItem<KT extends Comparable<?super KT>> {
 public KT getKey() ;
}
```

Έχουμε 10000 προϊόντα για αποθήκευση με κωδικούς που κυμαίνονται από 0 έως 9999. Επιπλέον οι κωδικοί των προϊόντων είναι μοναδικοί. Οι βασικές λειτουργίες που μας ενδιαφέρουν είναι:

- Δημιουργία άδειας αποθηκευτικής δομής
- Εισαγωγή ενός αντικειμένου Product στην αποθηκευτική δομή δεδομένου του κωδικού του pn. Σημειώστε ότι μια εξαίρεση Java δημιουργείται μετά την εισαγωγή προϊόντος του οποίου ο κωδικός έχει ήδη αποθηκευτεί.
- Αναζήτηση ενός αντικειμένου Product στην αποθηκευτική δομή δεδομένου του κωδικού του pn. Σημειώστε ότι επιστρέφεται null εάν ο κωδικός του προϊόντος δεν είναι ήδη αποθηκεμένος.
- Διαγραφή ενός αντικειμένου Product από την αποθηκευτική δομή δεδομένου του κωδικού του pn. Σημειώστε ότι επιστρέφεται αληθές εάν ο κωδικός του προϊόντος δεν είναι ήδη αποθηκεμένος, διαφορετικά ψευδές.

Σας δίνεται η ακόλουθη υλοποίηση της κλάσης UnsortedProductArray:

```
class UnsortedProductArray {
     static private MaxSize = 10000;
     private int currentSize;
     private Product [ ] products;
     public UnsortedProductArray() {
          currentSize = 0 ;
          products = new Product [MaxSize ];
     }
     public void insert (Product p) throws Exception {
          if (retrieve (p.getKey()) != null )
               throw new Exception (”duplicate item”);
          else products [currentSize++] = p ;
     }
     public Product retrieve (Integer pn) {
          Product foundProduct = null ;
          for (int i =0; i < current Size ; i++)
               if (products[i].getKey().equals(pn)) {
                    foundProduct = products [i] ;
                    break ;
               }
```

```
        return foundProduct ;
    }
    public boolean delete(Integer pn) {
        boolean foundFlag=false;
        for (int i=0; i<currentSize; i++)
            if (products[i].getKey().equals(pn)){
                foundFlag=true ;
                products[i]=products[currentSize-1];
                currentSize = c currentSize – 1 ;
                break ;
            }
        return foundFlag ;
    }
}
```

Ποιό είναι το κόστος χείριστης περίπτωσης των μεθόδων `insert()`, `retrieve()`, και `delete()` σε σχέση με τον τρέχοντα αριθμό n των προϊόντων που είναι αποθηκευμένα, χρησιμοποιώντας τον συμβολισμό O(. . .);

Λύση:
- insert: O(n)
- retrieve: O(n)
- delete: O(n)

(ζ) **(12 μονάδες)** Δώστε μια πιο αποδοτική υλοποίηση των μεθόδων `insert()`, `retrieve()`, και `delete()` καθώς και το βελτιωμένο κόστος χείριστης περίπτωσης τους σε σχέση με το προηγούμενο ερώτημα.

Λύση:
Idea: Use an array of size 10000 and use the given product number ranging from 0 to 9999) as index.

```
classProductTable{
    private intMaxSize=10001;
    private Product[] products;
    public ProductTable(){
      products = newProduct[MaxSize];
    }
    public void insert(Product p) throwsException{
      if(products[p.getKey()]==null)
         products[p.getKey()]=p;
      else throwException(...);
    }
    public Product retrieve(Integerpn){
      return products[pn];
    }
    public boolean delete(Integerpn){
      boolean foundFlag=(products[pn]!=null);
      products[pn]=null;
      Return foundFlag;
    }
```

```
}
```
All 3 operations have O(1) complexity. Also ermissible is a binary search tree ith the the usual logarithmic times, but with less marks.