Πανεπιστήμιο Κρήτης
Τμήμα Επιστήμης Υπολογιστών

HY-252 – Αντικειμενοστρεφής Προγραμματισμός
Βασίλης Χριστοφίδης

Τελική Εξέταση (3 ώρες)
Ημερομηνία: 9 Φεβρουαρίου 2012

Άσκηση 1	Άσκηση 2	Άσκηση 3	Άσκηση 4	Άσκηση 5	Άσκηση 6	Άσκηση 7	Άσκηση 8	Σύνολο
/11	/09	/11	10/	/23	/8	/18	/15	/105

Ονοματεπώνυμο:
Αριθμός Μητρώου:

Άσκηση 1 (11 μονάδες) Στατικοί Τύποι και Πίνακες Αντικειμένων

Εάν μια κλάση `B` είναι υπο-κλάση της `A`, ο ελεγκτής στατικών τύπων (static type checker) της Java θεωρεί ότι ο πίνακας αντικειμένων `B[]` είναι υπο-τύπος του `A[]`. Γι' αυτό τον λόγο ο παρακάτω κώδικας Java μεταφράζεται χωρίς προβλήματα σε bytecode προς εκτέλεση.

```
1. class A { ... };
2. class B extends A { ... };
3. B[] bArray = new B[10];
4. A[] aArray = bArray;
5. A x = new A();
6. if ( ... ) x = new B();
7. aArray[5] = x;
```

(α) (**3 μονάδες**) Εξηγήστε γιατί η ανάθεση στην γραμμή 4 (`A[] aArray = bArray`) είναι ορθώς τυποποιημένη (well-typed) για τον ελεγκτή στατικών τύπων (static type checker) της Java;

> **Λύση:**
> As stated in Java, if `B <: A` then the type checker considers `B[] <: A[]`. Then an object reference like `aArray` can refer to an object instance of any of its subtypes.

(β) (**5 μονάδες**) Εξηγήστε κάτω από ποιες συνθήκες η ανάθεση στην γραμμή 7 (`aArray[5] = x`) θα οδηγήσει σε ένα λάθος εκτέλεσης;

> **Λύση:**
> If `x` points to an `A` object, there is no problem when an `A` object can be reached through `aArray`. The problem is that the static type of `bArray` is `B[]` and `bArray` now contains an object that is not of type `B`.

(γ) (**3 μονάδες**) Πως ο bytecode compiler της Java διαχειρίζεται το πρόβλημα που περιγράψατε στο προηγούμενο ερώτημα;

> **Λύση:**
> The Java compiler inserts a run-time type check at the assignment `aArray[5]`

Άσκηση 2 (9 μονάδες) Διαχείριση Σκουπιδιών Μνήμης

Σας δίνεται η ακόλουθη κλάση Java:

```
class Avatar {
  static Navi Tom, Norm, Jake;
  private void blockBuster( ) {
     Tom = new Navi(1); // object 1
     Norm = new Navi(2); // object 2
     Jake = new Navi(3); // object 3
     Jake = Tom;
  }
}
```

α) **(4 μονάδες)** Ποιο(α) αντικείμενα θα αποδεσμευτούν από την μνήμη (garbage collected) μετά την εκτέλεση της μεθόδου blockBuster ( ); Δώστε μια σύντομη εξήγηση.

> **Λύση:**
> Object 3 is garbage since it is no longer reachable.

β) **(5 μονάδες)** Εξηγήστε σύντομα την διαφορά μεταξύ της έννοιας της προσβασιμότητας (reachability) και της βιωσιμότητας (liveness) των αντικειμένων στα πλαίσια της διαχείρισης σκουπιδιών μνήμης (garbage collection) της Java.

> **Λύση:**
> An object is reachable if there is some way for the program to access it, but is live only if it is actually accessed in the future. Garbage collection uses reachability to approximate liveness since the latter is much harder to determine.

Άσκηση 3 (11 μονάδες) Αφαιρετικές Κλάσεις και Διεπαφές

Σας δίνονται η ακόλουθη διεπαφή (Specifications) και κλάσεις (BaseClass, NextClass) Java:

```
public interface Specifications {
     int CONSTANT = 1000;
     int methodThree (int numOne, int numTwo);
} // end Specifications interface

public abstract class BaseClass {
     private int baseNum;
     public BaseClass (int baseNum) {
          this.baseNum = baseNum;
     }
     public int getBaseNum () {
          return baseNum;
     }
     public int methodOne (int numOne) {
          return numOne + baseNum;
     }
     public abstract int methodTwo (int numOne);
} // end BaseClass

public abstract class NextClass extends BaseClass {
     private int nextNum;
     public NextClass (int numOne, int numTwo) {
          super(numOne);
          nextNum = numTwo;
```

```
    }
    public int methodOne () {
        return getBaseNum() + nextNum;
    }
} // end NextClass
```

Στην συνέχεια μπορείτε να βρείτε ένα κώδικα πελάτη που επιδεικνύει πώς η κλάση `MissingClass` περιμένουμε να λειτουργεί:

```
public class TestMissingClass {
    public static void main(String[] args) {
        MissingClass mc = new MissingClass (1, 20, 300);
        System.out.println(mc.methodOne(500));
        System.out.println(mc.methodTwo(100));
        System.out.println(mc.methodThree(200, 400));
    } // end main
} // end TestMissingClass
/* OUTPUT:
501
1301
621
*/
```

Υλοποιήστε την κλάση `MissingClass` έτσι ώστε η εκτέλεση της μεθόδου `main()` της `TestMissingClass` να τυπώνει την παραπάνω έξοδο. Η `MissingClass` θα πρέπει να επεκτείνει (extends) την αφαιρετική κλάση `NextClass` και να υλοποιεί (implements) την διεπαφή `Specifications`.
Σημείωση: Η μόνη μαθηματική πράξη που θα χρειαστείτε είναι η πρόσθεση μεταξύ λεκτικών τιμών (literal values).

Λύση:

```
public class MissingClass extends NextClass implements
Specifications { // 2 points
  private int missingNum; // 2 points
  public MissingClass (int numOne, int numTwo, int numThree) {
     super (numOne, numTwo);
     missingNum = numThree;
  } // 3 points
  public int methodTwo (int numOne) {
     return CONSTANT + getBaseNum() + missingNum;
  } // 2 points
  public int methodThree (int numOne, int numTwo) {
     return methodOne() + numOne + numTwo;
  } // 2 points
} // end MissingClass
```

Άσκηση 4 (10 μονάδες) Χειρισμός Πλαισίου Συλλογών Αντικειμένων
Μας ενδιαφέρει η σάρωση μιας λίστας λέξεων ώστε να βρούμε εκείνες τις λέξεις (words) που δεν περιέχονται σε ένα δεδομένο λεξικό (dictionary) και να καταγράψουμε πόσες φορές εμφανίζονται στην λίστα. Υλοποιήστε την παρακάτω μέθοδο `countUnknownWords()` που παίρνει σαν παραμέτρους μία λίστα συμβολοσειρών (`List<String>`) για τις λέξεις και ένα σύνολο συμβολοσειρών (`Set<String>`) που αναπαριστά το λεξικό και επιστρέφει έναν πίνακα αντιστοιχήσεων (`Map<String,Integer>`) με τις λέξεις της λίστας εκτός λεξικού και την συχνότητα εμφάνισής τους. Οι καταχωρήσεις (entries) του παραγόμενου `Map` δεν απαιτείται να ταξινομηθούν ως προς κάποια διάταξη. Για αυτό τον λόγο, οποιαδήποτε κατάλληλη υλοποίηση του Map (π.χ. `HashMap` `TreeMap`) μπορεί να χρησιμοποιηθεί για την συλλογή του αποτελέσματος της μεθόδου. Οι συμβολοσειρές θα πρέπει να συγκρίνονται ως

προς την ταυτότητά τους χωρίς καμία τροποποίηση: π.χ. όλες οι ακόλουθες λέξεις "foo", "Foo", "FOO" και " FOO " είναι διαφορετικές. Αποφύγετε την δημιουργία επιπλέον δομών δεδομένων πέρα από αυτές που δίνονται στην εκφώνηση καθώς και την χρήση μεθόδων της `List` που εν δυνάμει έχουν μεγάλο κόστος εκτέλεσης όπως η `get(i)`.

```
// Return a Map containing a list of <String,Integer> pairs where
// each String is a String that appears in the List words but not
// in the Set dictionary, and the associated Integer is the number
// of times the word appears in the List words
Map<String,Integer> countUnknownWords(List<String> words,
                                      Set<String> dictionary) {
```

Λύση:

```
// allocate map to hold result frequencies
Map<String,Integer> freq = new HashMap<String,Integer>();
// process words in the dictionary
for (String word: words) {
`if (!dictionary.contains(word)) {
     // input word is not in the dictionary. count it
     if (freq.containsKey(word)) {// or (freq.get(word)!=null)
       // increase count of existing word
       freq.put(word, freq.get(word)+1);
     } else {
       // add new word
       freq.put(word,1);
     }
  }
}
// return result map
return freq;
}
```

Άσκηση 5 (23 μονάδες) Βασική Λειτουργικότητα Αντικειμένων

Για την ακόλουθη κλάση `Card` που υλοποιεί την διεπαφή Comparator υλοποιήστε τα παρακάτω:

α) **(3 μονάδες)** Ορίστε έναν απαριθμητό τύπο (enumerated type) με τα χρώματα (suits) των φύλλων σε μια τράπουλα: μπαστούνια (spades), κούπες (hearts), καρό (diamonds) και σπαθιά (clubs).

β) **(20 μονάδες)** Υλοποιήστε την μέθοδο `compareTo()` της διεπαφής `Comparator` για αντικείμενα τύπου `Card` έτσι ώστε τα χρώματα των φύλων να διατάσσονται σε μία λίστα ως εξής: μπαστούνια (spades) > κούπες (hearts)> καρό (diamonds) > σπαθιά (clubs).

```
public class Card implements Comparable {
```

Λύση:

```
public enum Suit {Spades, Hearts, Diamonds, Clubs}

private Suit mySuit;
Card (Suit s)  { mySuit = s; }
Card (int i) {
     if (i == 1) mySuit = Suit.Spades;
     else if (i == 2) mySuit = Suit.Hearts;
     else if (i == 3) mySuit = Suit.Diamonds;
     else mySuit = Suit.Clubs;
}
public Suit getSuit() { return mySuit; }
public String toString(){ return mySuit.toString(); }
public int compareTo(Object o) {
```

Λύση:

```
// return 1 if greater, 0 if equal, -1 if less than
```

```
if (o instanceof Card) {
  Card card = (Card) o;
  Suit c = card.getSuit();
  if (mySuit == Suit.Spades) {
     if (c == Suit.Spades) return 0;
     else return 1;
  } else if (mySuit == Suit.Hearts) {
     if (c == Suit.Spades) return -1;
     else if (c == Suit.Hearts) return 0;
     else return 1;
  } else if (mySuit == Suit.Diamonds) {
     if ((c == Suit.Spades) || (c == Suit.Hearts)) return -1;
     else if (c == Suit.Diamonds) return 0;
     else return 1;
  } else // (mySuit == Suit.Clubs) {
     if (c == Suit.Clubs) return 0;
     else return -1;
}
else
     throw new IllegalArgumentException();
}

Ή

if (o instanceof Card) {
    return ((Card) o).getSuit().compareTo(mySuit);
} else {
    throw new IllegalArgumentException();
}
```

}

Σημείωση: Η μέθοδος `compareTo()` θα πρέπει να υλοποιηθεί με τέτοιο τρόπο ώστε η εκτέλεση του ακόλουθου κώδικα να τυπώνει στην στάνταρ έξοδο: `1 0 -1`

```
public static void main(String[] args) {
       Card s = new Card(Suit.Spades);
       Card h = new Card(Suit.Hearts);
       System.out.println("S vs H = "+s.compareTo(h));
       System.out.println("S vs S = "+s.compareTo(s));
       System.out.println("H vs S = "+h.compareTo(s));
}
```

Άσκηση 6 (8 μονάδες) Επαναλήπτες Συλλογών Αντικειμένων

Σας δίνονται οι δύο ακόλουθες κλάσεις Java, `Student` και `Course`:

```
public class Student {
     public String name;
     public double average;
} // end Student
public class Course {
     // Private instance variables, storing data about a course,
     // including a collection of the students in the course
     // public methods for initializing a course, adding and
     // removing students, etc.
     // This method returns an iterator which will return each of
     // the students in the course, as Student objects.
     public Iterator iterator() {
          // body omitted
     }
}
```

Σε κάποιο άλλο σημείο του προγράμματος μας ενδιαφέρει να βρούμε τον υψηλότερο μέσο όρο βαθμολογίας των φοιτητών (`Student`) που παρακολουθούν ένα μάθημα (`Course`). Για να ενισχύουμε την γενικότητα της μεθόδου θέλουμε να αγνοήσουμε τις λεπτομέρειες αποθήκευσης των φοιτητών ενός μαθήματος εάν πχ. χρησιμοποιείται ένας πίνακας (`array`), ένα διάνυσμα (`Vector`), ένα σύνολο με κατακερματισμό (`HashSet`), κλπ. Αυτή την δυνατότητα μας την προσφέρουν οι επαναλήπτες (Iterators) στη Java που επιτρέπουν να σαρώνουμε τα στοιχεία μιας συλλογής αντικειμένων ενθυλακώνοντας τις λεπτομέρειες της υλοποίησης της. Υλοποιήστε την παρακάτω μέθοδο `maxAverage(Course c)` χρησιμοποιώντας την μέθοδο `iterator()` της κλάσης `Course`.

```
// This method must find the maximum average of any student in
// the course. It may assume that there is at least one student
// in the course and that all averages are between 0 and 100.
public static double maxAverage(Course c) {
// fill in the body of this method
```

Λύση:

```
Iterator iter = c.iterator();
double max = 0;
while (iter.hasNext()) {
  Student s = (Student) iter.next();
  if (s.average > max)
      max = s.average;
} // end while
return max;
```

Some common mistakes in the use of Iterators was:

Not knowing how to create the iterator. Some people tried `new Iterator()`, or `Course.iterator`

Forgetting to cast the result of `iter.next()` into `Student`

Assuming some kind of `getAverage()` method

Trying to compare entire Student objects with ">" or "<", instead of comparing the averages

Expecting `iter.next()` to return an `int` or an `Integer`

Calling `iter.next()` more than once inside the loop

```
}
```

Άσκηση 7 (18 μονάδες) Γενικοί Τύποι στην Java

Ορισμένες φορές είναι χρήσιμο να έχουμε λειτουργίες (operations) που επιστρέφουν ζεύγη τιμών. Για παράδειγμα, μια ακέραια διαίρεση μπορεί να θεωρηθεί σαν μια λειτουργία η οποία επιστρέφει ένα ζεύγος ακεραίων: ο πρώτος είναι το πηλίκο (factor) και ο δεύτερος το υπόλοιπο (remainder) της διαίρεσης. Δίνοντας π.χ. σαν είσοδο το 13, η διαίρεση με το 2 δίνει πηλίκο 6 και υπόλοιπο 1, η διαίρεση του 6 με το 2 δίνει πηλίκο 3 και υπόλοιπο 0, η διαίρεση του 3 με το 2 δίνει πηλίκο 1 και υπόλοιπο 1, ενώ η διαίρεση του 1 με το 2 δίνει πηλίκο 0 και υπόλοιπο 1. Όπως μπορείτε να δείτε από αυτό το παράδειγμα, εάν διαιρέσουμε διαδοχικά με το 2 τα παραγόμενα πηλίκα και βάλουμε τα υπόλοιπα σε μια λίστα 1 1 0 1, παίρνουμε την δυαδική αναπαράσταση του αρχικού αριθμού 13.

(α) **(9 μονάδες)** Υλοποιήστε μια γενική κλάση (generic class) `Pair<A,B>`, όπου A είναι ο τύπος της πρώτης συνιστώσας και B της δεύτερης συνιστώσας του ζεύγους. Υλοποιήστε την μέθοδο κατασκευής και τις μεθόδους πρόσβασης στις δύο συνιστώσες ενός ζεύγους `getFirst()` και `getSecond()`.

```
class Pair<A,B> {
```

Λύση:
```
    private A first;
    private B second;
    Pair(A a, B b) {
      first = a;
      second = b;
    }
    A getFirst() {
      return first;
    }
    B getSecond() {
      return second;
    }
}
```

(β) **(2 μονάδες)** Ορίστε την γενική διεπαφή (generic interface) `PairFunction<A,B>` η οποία προβλέπει μια μέθοδο `compute()` με όρισμα εισόδου κάποιου τύπου A και επιστρεφόμενη τιμή ένα ζεύγος του οποίου η πρώτη συνιστώσα έχει τύπο A και η δεύτερη τύπο B.

```
interface PairFunction<A,B> {
```

Λύση:
```
public Pair<A,B> compute(A a);
```

```
}
```

(γ) **(3 μονάδες)** Στην συνέχεια ορίστε μια κλάση `DivByTwo` η οποία υλοποιεί την γενική διεπαφή `PairFunction<A,B>` για ακέραιους τύπους παραμέτρων (δηλ `A=B=Integer`) και προσφέρει μια συγκεκριμένη υλοποίηση για την μέθοδο `compute()`.

Λύση:
```
class DivByTwo implements PairFunction<Integer,Integer> {
 public Pair<Integer,Integer> compute(Integer i) {
        return new Pair<Integer,Integer>(i/2, i%2);
 }
}
```

(δ) **(4 μονάδες)** Οι γενικοί τύποι στην Java υλοποιούνται μέσω ενός μηχανισμού απαλοιφής (erasure) των τύπων που χρησιμοποιούνται σαν παράμετροι. Ποιο είναι το αποτέλεσμα της απαλοιφής των τύπων παραμέτρων (type parameters) για την γενική διεπαφή `PairFunction<A,B>` και την μέθοδο `Pair<Integer,Integer> compute(Integer i)` των προηγούμενων ερωτημάτων;

Λύση:
```
a interface PairFunction {
     public Pair compute(Object a); }
b public Pair compute(Integer i) {
        return new Pair(i/2, i%2);}
```

Άσκηση 8 (15 μονάδες) Θεωρία Αντικειμενοστρεφούς Προγραμματισμού

(α) **(10 μονάδες)** Η αφαίρεση (abstraction) και η ενθυλάκωση (encapsulation) αποτελούν δύο από τις αρχές του αντικειμενοστρεφούς προγραμματισμού (object-oriented programming).

i) **(5 μονάδες)** Πως υποστηρίζεται η αφαίρεση δεδομένων στην Java

Λύση:
Abstraction provides a simple high-level view of an entity or activity. Data abstraction means that internal data of objects (defined by the fields of a class) cannot be directly accessed from outside of the object. This is achieved in Java by class or interface based programming.

ii) **(5 μονάδες)** Πως υποστηρίζεται η ενθυλάκωση στην Java;

Λύση:
Confining / hiding information so it can only be accessed through a defined interface. Data is hidden inside objects and can only be accessed through its public methods. This is achieved in Java by declaring the fields to be private (or default, to allow package-wide access, or protected to allow subclass access). Getter and setter methods may be provided to allow indirect access to the data.

(β) **(5 μονάδες)** Γιατί η δυναμική δέσμευση ονομάτων (dynamic binding) είναι σημαντική στον αντικειμενοστρεφή προγραμματισμό;

Λύση:
Dynamic binding is the selection of the code implementing a method call at run-time, based on the actual run-time (i.e. dynamic type) of the target object of the call. This is important because it allows the same code to be used for different sub-types of object: it makes code more reusable.