Πανεπιστήμιο Κρήτης
Τμήμα Επιστήμης Υπολογιστών

ΗΥ-252 – Οντοκεντρικός Προγραμματισμός
Βασίλης Χριστοφίδης

Πρόοδος (3 ώρες)
Ημερομηνία: 12 Απριλίου 2003

Ονοματεπώνυμο:
Αριθμός Μητρώου:

Άσκηση 1 (10 μονάδες)

Γράψτε ένα πρόγραμμα Java (`FindChar`) το οποίο να τυπώνει τον αριθμό της γραμμής όπου εμφανίζεται για πρώτη φορά ένας συγκεκριμένος χαρακτήρας μέσα σε ένα αρχείο. Ο χαρακτήρας δίνεται σαν πρώτο όρισμα εισόδου (command line argument) του προγράμματος, ενώ το όνομα του αρχείου σαν δεύτερο όρισμα. Για ευκολία μπορείτε να θεωρήσετε ότι το πρόγραμμα εκτελείται στο ίδιο directory με το αρχείο εισόδου. Η έξοδος του προγράμματος πρέπει να τυπώνει τα παρακάτω (η αρίθμηση των γραμμών αρχίζει από το 0):

- Αν ο χαρακτήρας «I» εμφανίζεται για πρώτη φορά στην γραμμή **7** του αρχείου **file1.txt** τότε `java FindChar I file1.txt` εκτυπώνει:

```
The character I appears first in line 7 of file file1.txt
```

- Αν ο χαρακτήρας «y» δεν εμφανίζεται στο αρχείο **file2.txt** τότε `java FindChar y file2.txt` εκτυπώνει:

```
The character y does not appears in file file2.txt
```

Τι θα τυπώσει το πρόγραμμά σας όταν το εκτελέσετε με τα παρακάτω ορίσματα εισόδου; `java FindChar x FindChar.java`

Λύση:

```
import java.io*
class FindChar {
 public static void main (String[] args)
                                    throws IOException {
  char match = args[0].charAt(0);
  FileReader filein =  new FileReader(args[1]);
  LineNumberReader in=new LineNumberReader(filerin);
  int i;
  while ((i=in.read() != -1) {
    char ch = (char)I;
    if (ch == match) {
       System.out.println("The character " + match +
      " appears first in line " + in.getLineNumber()
                          + " of file " + args[1] );
       return;
      }
   System.out.println("The character " + match +
              " does not appear in file "+ args[1] );
  }
}
```

Άσκηση 2 (10 μονάδες)

Ποια είναι η έξοδος του παρακάτω προγράμματος **Test**;

```
class A {
  public int x=1;
  public int y=5;
  public A() { y=6; }
  public A(int a) { x=a; }
  public A(int a, int b) { x=a; y=b; }
}

class B extends A {
  public B() {}
  public B(int a) { super(a); }
  public B(int a, int b) { super(a,b); }
  public B(int a, int b, int c) { x=a+b+c; }
  public B(int a, int b, int c, int d) {
         super(a,b); y=c+d; }
}

public class Test {
  public static void main(String[] args) {
    B b1 = new B();
    System.out.println("b1.x: "+b1.x+"  b1.y: "+b1.y);
    B b2 = new B(2);
    System.out.println("b2.x: "+b2.x+"  b2.y: "+b2.y);
    B b3 = new B(3,4);
    System.out.println("b3.x: "+b3.x+"  b3.y: "+b3.y);
    B b4 = new B(1,1,1);
    System.out.println("b4.x: "+b4.x+"  b4.y: "+b4.y);
    B b5 = new B(2,2,2,2);
    System.out.println("b5.x: "+b5.x+"  b5.y: "+b5.y);
  }
}
```

Λύση:

```
b1.x: 1      b1.y: 6
b2.x: 2      b2.y: 5
b3.x: 3      b3.y: 4
b4.x: 3      b4.y: 6
b5.x: 2      b5.y: 4
```

Άσκηση 3 (14 μονάδες)

Θεωρήστε το παρακάτω πρόγραμμα Java. Απαντήστε στις παρακάτω ερωτήσεις με στόχο να συμπληρώσετε τα κενά στις δηλώσεις των μελών (δεδομένων και μεθόδων) της κλάσης **Professor**.

```
public class Professor {
  String name;
  boolean nice;
  boolean exciting;
  private ________ int instance_count;
```

```
  public Professor() {
         instance_count++; }
  public __________ printname() {
         System.out.println("Prof." + name); }
  __________ shouldTakeHisCourse() {
         return (nice && exciting); }
  protected void finalize(){
         __________ }
}
```

(α) Ποιο είναι το αναγνωριστικό δικαιώματος πρόσβασης (visibility modifier) που εμμέσως δηλώνεται στις μεταβλητές δεδομένων **name, nice** και **exciting** ;

Λύση:
package

(β) Η μεταβλητή **instance_count** πρέπει να δηλωθεί σαν στατική (static), ή σαν μη-στατική (δηλ. dynamic);

Λύση:
static

(γ) Η μέθοδος **printname** πρέπει να δηλωθεί σαν στατική (static), σαν μη-στατική (δηλ. dynamic), ή και τα δύο;

Λύση:
nonstatic

(δ) Ποιος πρέπει να είναι ο επιστρεφόμενος τύπος της μεθόδου **printname** ;

Λύση:
void

(ε) Ποιο είναι το καταλληλότερο αναγνωριστικό δικαιώματος πρόσβασης (visibility modifier) για την μέθοδο **shouldTakeHisCourse**;

Λύση:
public

(ζ) Ποιος πρέπει να είναι ο επιστρεφόμενος τύπος της μεθόδου **shouldTakeHisCourse**;

Λύση:
Boolean

(η) Ποιο είναι το σώμα της μεθόδου **finalize**;

Λύση:
--instance_count;

Άσκηση 4 (26 μονάδες)

Θεωρήστε την παρακάτω κλάση Java για την αναπαράσταση κλασματικών αριθμών (`Fraction`):

```
class Fraction {
/*Rational numbers are represented by fractions of the
  form (a,b), where a and b are integers and b is not 0.
  a is called the "numerator" and b is the "denominator".
  Fractions are stored in a reduced canonical form such
  that the greatest common divisor of a and b is 1, and b
  is always positive. Each fraction will be reduced every
  time it gets set or changed.
*/
  private int numerator;
  private int denominator;
  public Fraction (int a, int b) {
       if (b != 0) {
          // Make sure denominator is positive.
          if (b < 0) {
             denominator = Math.abs(b);
              numerator = -a; }
         else {
             numerator = a;
             denominator = b; }
         reduce(); }
       }
  // Returns this fraction's numerator.
  public int getNumerator() {
         return numerator;
       }
  // Returns this fraction's denominator.
  public int getDenominator() {
         return denominator;
       }
  // Returns this fraction's value.
  public double getValue() {
         return (double)numerator/denominator;
        }
  // Divides this fraction in half.
  public void halve() {
     denominator = denominator * 2;
     reduce();
    }
  // Reduces this Fraction to its simplest form:
  // denominator is > 0 and gcd of numerator and
  // denominator is 1.
  // For example, 6/8 would be reduced to 3/4.
  private void reduce() {
      // Divide both numerator and denominator by gcd.
      final int g = gcd(numerator, denominator);
      numerator = numerator/g;
      denominator = denominator/g;
      }
  // Greatest common divisor of two integers
  private int gcd(int i,int j) {
```

```
      // At least one argument of the gcd must be nonzero
      if (i != 0 || j != 0) {
       int big = Math.abs(i);
       int small = Math.abs(j);
       int remainder = big % small;
       while (remainder !=0) {
           big=small;
           small=remainder;
           remainder=big % small;
         }
       return small;}
    }
  // Returns true if this Fraction equals otherFrac, and
  // false otherwise.
  public boolean equals(Fraction otherFrac) {
     if (numerator == otherFrac.getNumerator() &&
         denominator == otherFrac.getDenominator())
         return true;
     else
         return false;
    }
 // Return String version of a fraction. We distinguish
 // three cases: (a) if Fraction is zero then "0"
 // (b) if Fraction is between -1..1 then “num” “/” “den”
 // (c) if Fraction is >= 1 or <= -1 then “integralpart”
                                          +“fractionpart”
 public String toString() {
     // To be completed by the students
     } // end of toString
} // end of Rational class
```

(α) Ποιες από τις παραπάνω μεθόδους δεν ανήκουν στο συμβόλαιο της κλάσης `Fraction` ;

> **Λύση:**
> The private methods `gcd(i,j)` and `reduce()`

(β) Ποιες είναι οι αμετάβλητες συνθήκες (invariant conditions) που πρέπει να ικανοποιούν όλα τα έγκυρα στιγμιότυπα της κλάσης `Fraction` ; Ποια είναι η εκ των υστέρων συνθήκη (post condition) του κατασκευαστή αντικειμένων της;

> **Λύση:**
> **Invariant** condition: Once created and until destroyed, this instance contains a valid fraction `getDenominator() > 0` stored in a normalized form `gcd(getNumerator(), getDenominator()) == 1`
> The **Post** condition of the constructor is essentially the **Invariant** and that the value has been set correctly: `getValue() == (double) getnumerator()/getdenominator()`

(γ) Ποια είναι η εκ των προτέρων συνθήκη (pre condition) της μεθόδου `gcd(i,j)`; Γιατί η μέθοδος `reduce()` δεν χρειάζεται να ελέγχει τις εκ των προτέρων συνθήκες της πριν καλέσει στον κώδικά της την μέθοδο `gcd(i,j)`;

Λύση:
Pre condition: At least one argument of the `gcd (i, j)` must be nonzero
`i != 0 || j != 0`
From the invariant conditions of class `Fraction` we guarantee that at least the `denominator` is nonzero.

(δ) Πως η μέθοδος `equals` μπορεί να συγκρίνει δύο `Fraction`s αν είναι ίσα με δεδομένο ότι παίρνει μόνο ένα `Fraction` για παράμετρο εισόδου; Πως εξασφαλίζεται ότι διαφορετικοί κλασματικοί αριθμοί 1/2, 2/4, 3/6 κλπ. είναι όλοι ίσοι; Γιατί είναι λάθος η υλοποίηση της μεθόδου χρησιμοποιώντας την εντολή: `return(this->getvalue() == otherFrac->getvalue())`

Λύση:
The two Fractions compared are (1) the parameter and (2) the Fraction whose `equals` method is being called. Constructors and Mutators (e.g. `halve()`) ensure that fractions are stored in a normalized form by calling the method `reduce()`. Finally value equality is wrong due to double comparison on non-significant bits.

(ε) Υλοποιήστε την μέθοδο `toString()` σύμφωνα με τις προδιαγραφές που σας δίνονται σαν σχόλια της μεθόδου;

Λύση:
```
// Fraction is zero.
if (numerator==0)
    return "0";
else {
   // Fraction is between -1..1.
   if (Math.abs(numerator)<denominator)
            return Integer.toString(numerator)+
                   "/"+Integer.toString(denominator);
   else {
    // Fraction is >= 1 or <= -1.
    // Represent as a mixed number.
    String intPart= Integer.toString(numerator/
                                     denominator);
    String fractPart="";
    // Fractional part is non-zero.
    if (numerator % denominator != 0) {
        fractPart=" "+
             Integer.toString(Math.abs(numerator) %
                                      denominator)
                +"/"+Integer.toString(denominator);
       }
    return intPart+fractPart;
}
```

(ζ) Για κάθε κομμάτι κώδικα Java που σας δίνετε παρακάτω υποδείξτε αν εκτελείται σωστά ή αν υπάρχει κάποιο λάθος. Στην πρώτη περίπτωση γράψτε τα αποτελέσματα που τυπώνονται έξοδο του προγράμματος. Στην δεύτερη περίπτωση εξηγείστε τους λόγους στους οποίους οφείλεται το λάθος.

```
Fraction f1 = new Fraction(3, 5);
Fraction f2 = new Fraction(3, 5);
Fraction f3 = f1;
if (f1 == f2)
        System.out.println("alpha");
if (f1 == f3)
        System.out.println("beta");
if (f1.equals(f2))
        System.out.println("gamma");
if (f1.equals(f3))
        System.out.println("delta");
```

Λύση:
Runs successfully.
Output:
beta
gamma
delta

```
Fraction[] list;
list = new Fraction[15];
System.out.print("The numerators are: ");
for (int i=0; i<15; i++)
   System.out.print(list[i].getNumerator());
```

Λύση:
Produces an error message because the elements of the list have not been initialized

```
Fraction apple = new Fraction (1, 2);
Fraction peach = new Fraction (4, 5);
Fraction pear = apple;
peach.halve();
pear.halve();
System.out.println(apple);
System.out.println(peach);
System.out.println(pear);
```

Λύση:
Runs successfully.
Output:
1/4
2/5
1/4

Άσκηση 5 (20 μονάδες)

Θέλουμε να υλοποιήσουμε ένα πρόγραμμα Java για την αποτίμηση αριθμητικών εκφράσεων ακέραιων αριθμών. Χρησιμοποιήσετε τον μηχανισμό της κληρονομικότητας αφαιρετικών ή συγκεκριμένων κλάσεων και μεθόδων για να αναπαραστήσετε και να υλοποιήσετε την αποτίμηση (evaluation) αριθμητικών εκφράσεων (`Expression`), ακέραιων τιμών (`Value`) καθώς και των τελεστών (`Operator`) + (`AddOp`), - (`SubOp`), * (`MulOp`) και / (`DivOp`).

Για ευκολία μπορείτε να παραλείψετε τις μεθόδους ανάγνωσης και κατασκευής του συντακτικού δέντρου μιας αριθμητικής έκφρασης. Για την ιεραρχία των κλάσεων της άσκησης συνιστάτε η χρήση και ενδιάμεσων βοηθητικών κλάσεων όπως δυαδική έκφραση (**Binary**) και αριθμητικός τελεστής (**ArithmeticOp**). Για την αποτίμηση των αριθμητικών εκφράσεων των ακέραιων τιμών (**Value**) καθώς και των τελεστών θα χρειαστείτε την υλοποίηση μιας μεθόδου **evaluate()**.

Λύση:

```
abstract class Expression {
  public abstract int evaluate();
}

class Value extends Expression {
  int Val;
  public Value(int i) { Val = i; }
  public int evaluate() { return Val; }
}

class Binary extends Expression {
  Operator op;
  Expression term1, term2;
  public Binary(Operator o, Expression t1,
                                    Expression t2) {
       op = o; term1 = t1; term2 = t2;
    }
  public int evaluate () {
       return op.evaluate(term1.evaluate(),
                          term2.evaluate());
    }
}

abstract class Operator {
  public abstract int evaluate(int v1, int v2);
}

abstract class ArithmeticOp extends Operator {}

class AddOp extends ArithmeticOp {
  public int evaluate(int v1, int v2) {
       return v1 + v2;}
}

class SubOp extends ArithmeticOp {
  public int evaluate(int v1, int v2) {
       return v1 - v2;}
}

class MulOp extends ArithmeticOp {
  public int evaluate(int v1, int v2) {
       return v1 * v2;}
}

class DivOp extends ArithmeticOp {
  public int evaluate(int v1, int v2) {
       return v1 / v2;}
}
```

Άσκηση 6 (20 μονάδες)

(α) Εξηγείστε την έννοια του εμβέλειας (scope) και της ορατότητας (visibility) των αναγνωριστικών (identifiers) όπως οι μεταβλητές σε ένα πρόγραμμα Java.

Λύση:
The *scope* of a variable declaration is the *collection of statements* that can access that variable. Program blocks (e.g., using {}) determine the collection of statements where a variable declaration is visible. The scope of *global* variables includes all statements in the program. The scope of *local* variables includes only those statements inside the function where it is declared. Program blocks can be also nested. If an identifier in an outer scope is redeclared in an inner scope, the new declaration is said to *hide* the outer declaration. An identifier is *visible* in a statement if its scope includes that statement and the identifier is not hidden. Note that in Java programs, class (or interfaces) definitions introduce blocks, and therefore determine the scope of variables. Local variables are called instance variables. Do not confuse, the scope of variables within a program block with the Java visibility modifiers determining access rights of variables (or methods) across blocks as class (or interfaces) definitions.

(β) Τι εννοούμε σαν αφαίρεση δεδομένων (data abstraction) και πως υποστηρίζεται από την Java;

Λύση:
Data abstraction or abstract data type provides an *encapsulation* mechanism to limit the visibility of both the *data values* and the *functions* defined for the values. The key notion of data abstraction is to bundle code and data together in a single module so that the *functions* provide a *public interface* to a logical type. They allow for the separation of *interface* from *implementation* and restrict access to the *underlying representation*.
In an object-oriented language like Java, the data type is bound together with the initialization and other operations on that type. A data type is referred to as a class (concrete or abstract) or an interface. Operations are implemented by methods. The initialization is accomplished by special methods called constructors. Each method or instance variable in a class can be declared to have a particular level of visibility—*private*, *protected*, or *public*—with regard to its client classes. A *public* variable or method is visible to *any client* of the class; for most classes, the instance variables are not public. The public methods define the interface of the class. A *private* variable or method is visible only to the current class and not to either its subclasses or to the outside world. A *protected* variable or method is visible only to subclasses of the class.

(γ) Τι είναι μια αφαιρετική κλάση (abstract class); Ποία είναι η κύρια χρησιμότητα των αφαιρετικών μεθόδων (abstract method); Εξηγήστε τον πολυμορφισμό (polymorphism) που οφείλεται στις αφαιρετικές κλάσεις.

Λύση:
A class is declared abstract if one or more of its methods are declared to be abstract. A method is declared abstract if the code of the method is not provided by the class but must instead be provided by its subclasses. Abstract classes are used to define a public interface to be shared by a set of classes. It is illegal to create (initialize) an object of an abstract class. This restriction ensures that a call cannot be made to an abstract (unimplemented) method. Abstract methods specify an interface that concrete subclasses have to satisfy by overriding them with a concrete method. One motivation for this feature is that it allows separating interface from implementation and thus facilitates programming in the large. Polymorphism refers to the late binding of a call to a specific method in an object. Consider the following method call obj.m();
The declared type T of the object obj guarantees that all subtypes of T implement an appropriate method m; Java verifies this at compile time. The actual instance of the method m that is called depends on the dynamic type of the object obj at runtime.

(δ) Τι είναι μια στατική μέθοδος (static method) ; Δώστε ένα παράδειγμα για την χρησιμότητα τους.

Λύση:
A static method is a method that "belongs" not to an individual object, but to an entire class. It cannot access instance variables, and it can be called without having an instance to call it on. All information required by a static method is passed as arguments. One motivation for this feature is that for example trigonometric functions are not logically associated to any object. Hence, it would feel wrong to have to provide an object in order to call them. In other terms they correspond to traditional C functions.

(ε) Τι είναι μια τελική μέθοδος (final method) ; Ποία είναι η κύρια χρησιμότητα τους ;

Λύση:
A final method is a method that cannot be overridden in a subclass. One motivation for this feature is that it may make code easier to read, because the client can rely on calling one particular implementation of a method instead of any of a set of overriding versions (i.e., turn off the polymorphism).