Πανεπιστήμιο Κρήτης
Τμήμα Επιστήμης Υπολογιστών

ΗΥ-252 – Οντοκεντρικός Προγραμματισμός
Βασίλης Χριστοφίδης

Πρόοδος (3 ώρες)
Ημερομηνία: 30 Απριλίου 2004

Ονοματεπώνυμο:
Αριθμός Μητρώου:

Άσκηση 1 (9 μονάδες)

Το API της Java δεν περιέχει κάποια κλάση για τον άμεσο χειρισμό της εισόδου από το πληκτρολόγιο καθώς και την μετατροπή αυτής της εισόδου στον αντίστοιχο τύπο δεδομένων, όπως `Integer`, `Float` κλπ. Μπορούμε όμως, για το σκοπό αυτό, να ορίσουμε μια βοηθητική κλάση **`Keyboard`** η οποία θα υλοποιεί τις παρακάτω στατικές μεθόδους:

`String readString();`
Διαβάζει την είσοδο που δίνει ο χρήστης από το πληκτρολόγιο και επιστρέφει μια `String` αναπαράσταση αυτής χωρίς το τερματικό *`enter key (\n)`*.

`Integer readInteger();`
Διαβάζει την είσοδο όπως και η `readString` και προσπαθεί να δημιουργήσει ένα `Integer` αντικείμενο με τιμή την τιμή της εισόδου.

`Float readFloat();`
Διαβάζει την είσοδο όπως και η `readString` και προσπαθεί να δημιουργήσει ένα `Float` αντικείμενο με τιμή την τιμή της εισόδου.

`void pause ();`
Παγώνει την εκτέλεση του προγράμματος ωσότου ο χρήστης πατήσει ένα πλήκτρο.

1.1. (2 μονάδες) Χρησιμοποιήστε την κλάση **`Keyboard`** και γράψτε ένα πρόγραμμα Java το οποίο διαβάζει μία συμβολοσειρά από το πληκτρολόγιο και την εμφανίζει στην οθόνη δύο φορές σε ξεχωριστές γραμμές.

Λύση:
```
public class thema1 {
   /* Reads a string and prints it twice */
   public static void main (String args[]) {
       String aString;
       System.out.println("Please enter a string:");
       aString=Keyboard.readString();
       System.out.println(aString);
       System.out.println(aString);
   }
}
```

1.2 (3 μονάδες) Χρησιμοποιήστε την κλάση **`Keyboard`** και γράψτε ένα πρόγραμμα Java το οποίο διαβάζει δύο ακεραίους από το πληκτρολόγιο και εμφανίζει στην οθόνη το άθροισμά τους (Αγνοείστε την επαλήθευση της εισόδου).

Λύση:

```
public class thema2 {
    /* Reads two integers and prints the sum */
    public static void main (String args[]) {
      Integer firstInteger, secondInteger;
      System.out.println("Please enter first integer:");
      FirstInteger=Keyboard.readInteger();
      System.out.println("Please enter second integer:");
      secondInteger=Keyboard.readInteger();
      System.out.println("sum=" +
                          firstInteger.intValue() +
                          secondInteger.intValue());
    }
}
```

1.3. (4 μονάδες) Χρησιμοποιήστε την κλάση **Keyboard** και γράψτε ένα πρόγραμμα Java το οποίο διαβάζει έναν ακέραιο (μεταξύ 0 και 9) από το πληκτρολόγιο αφού εμφανίσει στην οθόνη την συμβολοσειρά: «Εισαγάγετε έναν αριθμό (0-9):». Το πρόγραμμα πρέπει να επαληθεύει την είσοδο ώστε να επιτρέπει μόνο ένα ψηφίο μεταξύ 0 και 9.

Λύση:

```
public class thema3 {
  /* Validates the input of a digit between 0 and 9 */
  public static void main (String args[]) {
   Integer number;
   do {
     System.out.print("Please enter your choice (0-9):");
     number = Keyboard.readInteger();
   } while (number==null ||
     number.intValue() < 0 || number.intValue() > 9);
  }
}
```

Άσκηση 2 (26 μονάδες)

Σας δίνονται οι ορισμοί τριών κλάσεων Java που ορίζονται στο ίδιο πακέτο (package). Για κάθε μία από τις ακόλουθες περιπτώσεις, γράψτε τον απλούστερο τρόπο αναφοράς στην μεταβλητή ή μέθοδο των κλάσεων που σας δίνονται στην εκφώνηση. Αν αυτή δεν μπορεί να προσπελαστεί μέσα στο εύρος του προγράμματος που σας δίνεται επίσης στην εκφώνηση, σημειώστε "ΟΧΙ" και δικαιολογήστε σύντομα την απάντησή σας.
Σημείωση: Στην περίπτωση που μπορείτε να χρησιμοποιήσετε στις αναφορές είτε το όνομα μιας κλάσης είτε το όνομα μιας μεταβλητής αντικειμένων, προτιμείστε την χρήση ονομάτων κλάσεων.

Λύση:

In each case, the first answer is the expected answer. Where more than one answer is given, the additional answers refer to different variables having the same name but belonging to the locally created objects. Knowing how to refer to variables is **essential** to programming in Java; if you missed more than a very few, this is the most important thing for you to study.

```
public class A {
    static int staticA;
    int instanceA;
    private int privateA;
    void methodA() {
        int localA = 1;
        A a = new A();
        B b = new B();
        C c = new C();
      // Inside methodA
    }
}

public class B extends A {
    static int staticB;
    int instanceB;
    private int privateB;
    void methodB() {
        int localB = 2;
        A a = new A();
        B b = new B();
        C c = new C();
      // Inside methodB
    }
}

public class C {
    static int staticC;
    int instanceC;
    private int privateC;
    static void methodSC(){
        int localC = 3;
        A a = new A();
        B b = new B();
        C c = new C();
    // Inside methodSC
    }
}
```

Variable Method	Inside methodA	Inside methodB	Inside methodSC
staticA	staticA	staticA	A.staticA
instanceA	instanceA, a.instanceA, b.instanceA	instanceA, a.instanceA, b.instanceA	a.instanceA, b.instanceA
privateA	privateA, a.privateA, ((A)b).instanceA	NO	NO
localA	localA	NO	NO
methodA	methodA(), a.methodA(), b.methodA()	methodA(), a.methodA(), b.methodA()	a.methodA(), b.methodA()
staticB	B.staticB	staticB	B.staticB
instanceB	b.instanceB	instanceB	b.instanceB
privateB	NO	privateB	NO
localB	NO	localB	NO
methodB	b.methodB()	methodB(), b.methodB()	b.methodB()
staticC	C.staticC	C.staticC	staticC
instanceC	c.instanceC	c.instanceC	c.instanceC
privateC	NO	NO	NO, c.privateC
localC	NO	NO	localC
methodSC	C.methodSC()	C.methodSC()	methodSC()

Άσκηση 3 (20 μονάδες)

Απαντήστε στις παρακάτω ερωτήσεις με στόχο να συμπληρώσετε τα κενά στις δηλώσεις των μεθόδων των κλάσεων **Employee**, **Faculty**, **Staff** και **Persons** που δίνονται στην συνέχεια:

```
// Person.java: abstract base class
public abstract class Person {
   // Get job payment
   // abstract method, so omit body
   public abstract double getWages();
}
// Employee.java: the class Employee inherits from Person
public class Employee extends Person {
   protected String name; // in the form "Last, First"
   protected String address;
   // constructor: first name, then address
   public Employee(String n, String a) {
      // Fill in for Part a.
   }
   // convert Person objects into a String representation
```

```
    public String toString() {
       return name + ", " + address; }
    // must be either faculty or staff for wages
    public double getWages() {
       return 0.0; }
}
// Faculty.java: the class Faculty inherits from Employee
public class Faculty extends Employee {
    protected double salary;
    // Constructor: salary, name, address
    public Faculty(double s, String n, String a) {
       // Fill in for Part b.
    }
    // convert Faculty objects to a String representation
    public String toString() {
       // Fill in for Part c.
    }
    public double getWages() {
       return salary; }
}
// Staff.java: the class Staff inherits from Employee
public class Staff  ...
    // Fill in for Parts d., e., f.
}
// Persons.java: test the Person-Employee-Faculty-Staff
// hierarchy
public class Persons {
    public static void main (String[] args) {
       Person[] persons = new Person[3];
       persons[0] = // Fill in for Part g.
       persons[1] = // Fill in for Part g.
       persons[2] = // Fill in for Part g.
       printPersons(persons);
       System.out.println(); }
    public static void printPersons(Person[] s) {
       // Fill in for Part h.
    }
}
```

a. **(2 μονάδες)** Συμπληρώστε το σώμα της μεθόδου κατασκευής της κλάσης **Employee**.

Λύση:
```
// constructor
public Employee(String n, String a) {
   name = n; address = a;
}
```

b. **(2 μονάδες)** Συμπληρώστε το σώμα της μεθόδου κατασκευής της κλάσης **Faculty** χρησιμοποιώντας την μέθοδο κατασκευής της **Employee**.

Λύση:
```
// constructor
public Faculty(double s, String n, String a) {
   super(n, a);  // call the superclass constructor
   salary = s;
}
```

c. **(2 μονάδες)** Συμπληρώστε το σώμα της μεθόδου **toString** της κλάσης **Faculty** χρησιμοποιώντας την μέθοδο **toString** της **Employee**.

Λύση:
```
// convert the Faculty class to a String
public String toString() {
  return "Name and address:" + super.toString() +
         "; Wages = " + getWages();
}
```

d. **(2 μονάδες)** Ολοκληρώστε την δήλωση της κλάσης **Staff** η οποία κληρονομεί την **Employee** περιέχοντας σαν επιπλέον μεταβλητές στιγμιοτύπων **double hourlyRate** και **int hoursWorked**.

Λύση:
```
// Staff.java: the class Staff
public class Staff extends Employee {
   protected double hourlyRate;
   protected int hoursWorked; ...
}
```

e. **(2 μονάδες)** Συμπληρώστε το σώμα της μεθόδου κατασκευής της κλάσης **Staff** η οποία παίρνει σαν παραμέτρους τις μεταβλητές στιγμιοτύπων του προηγούμενου ερωτήματος καθώς και τις **name** και **address**.

Λύση:
```
// Constructor
public Staff(double hr, int hw, String n, String a) {
 super(n, a);  // call the superclass constructor
 hourlyRate = hr; hoursWorked = hw;
}
```

f. **(4 μονάδες)** Συμπληρώστε το σώμα της μεθόδου **toString** της κλάσης **Staff** χρησιμοποιώντας την μέθοδο **toString** της **Employee**.

Λύση:
```
// convert the Staff class to a String
public String toString() {
  return "Name and address:" + super.toString() +
         "; Wages = " + getWages();
}
// Get job payment
public double getWages() {
      return hourlyRate * hoursWorked;
}
```

g. **(2 μονάδες)** Στην κύρια μέθοδο της κλάσης **Persons** δημιουργήστε τρία καινούργια αντικείμενα για τις παρακάτω περιπτώσεις:
   a. **Employee** (με name "Boithos Java", και address "Katw apo ta asteria")
   b. **Faculty** (με salary 1700, name "Vassilis Christophides", και address "Kapou sto Hrakleio")

c. `Staff` (με `hourly rate` 7.5, `hours worked` 40, `name` "Mamalaki Maria", και `address` "Panepisthmioy Kritis 100").

Λύση:
```
persons[0] = new Employee("Boithos Java",  "Katw apo
ta asteria");
persons[1] = new Faculty(1700, "Vassilis
Christophides",  "Kapou sto Hrakleio");
persons[2] = new Staff( 7.5, 40, "Mamalaki Maria",
"Panepisthmioy Kritis 100");
```

h. **(4 μονάδες)** Συμπληρώστε το σώμα της μεθόδου **printPersons** της κλάσης **Persons**.

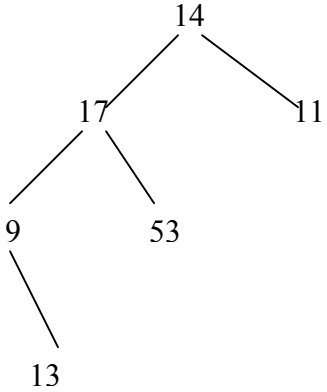
Λύση:
```
public static void printPersons(Person[] s) {
  for (int i = 0; i < s.length; i++)
       System.out.println(s[i]);
}
```

Άσκηση 4 (10 μονάδες)

Σας δίνεται ο παρακάτω ορισμός της κλάσης **BTNode** που αναπαριστά έναν κόμβο ενός δυαδικού δέντρου:

```
public Class BTNode {
      private Object data;
      private BTNode left;
      private BTNode right;
……..
}
```

Υλοποιήστε μια στατική μέθοδο **print** η οποία θα τυπώνει ένα δυαδικό δέντρο (σαν αυτό του παραδείγματος που ακολουθεί) στην παρακάτω μορφή κειμένου. Για παράδειγμα, η μέθοδός σας θα πρέπει να τυπώνει το δέντρο της αριστερής εικόνας στην κειμενική μορφή που απεικονίζεται στην δεξιά εικόνα.
Σημειώστε τις εκ των προτέρων (pre) και εκ των υστέρων (post) συνθήκες της μεθόδου **print** με την μορφή σχολίων Javadoc.



Η μέθοδός σας θα πρέπει να τυπώνει το δέντρο που σας δίνεται στην αριστερή φιγούρα με τον παρακάτω τρόπο:

```
14
        17
                9
                        --
                        13
                53
        11
```

Οι διπλές παύλες "--" τυπώνονται στα σημεία όπου ένας κόμβος-παιδί δεν έχει αδέλφια.

Λύση:

```
// Uses an inorder traversal to print the data from
// each node at or below this node of the binary
// tree, with indentations to indicate the depth
// of each node.
// @param:  depth: the depth of this node (with 0
// for root, 1 for the root's children, and so on)
// Precondition: depth is the depth of this node.
// Postcondition: The data of this node and all its
// descendants have been written by
// System.out.println( ) using an inorder traversal.
public void print(int depth) {
   int i;
   // Print the indentation and the data from the
   // current node:
   for (i = 1; i <= depth; i++)
        System.out.print("    ");
   System.out.println(data);
   // Print the left subtree (or a dash if there is
   // a right child and no left child)
   if (left != null)
       left.print(depth+1);
   else if (right != null) {
       for (i = 1; i <= depth+1; i++)
            System.out.print("    ");
       System.out.println("--"); }
   // Print the right subtree (or a dash if there is
   // a left child and no right child)
   if (right != null)
       right.print(depth+1);
   else if (left != null) {
       for (i = 1; i <= depth+1; i++)
            System.out.print("    ");
       System.out.println("--"); }
}
```

Άσκηση 5 (40 μονάδες)

Θέλουμε να σχεδιάσουμε και να υλοποιήσουμε έναν αφαιρετικό τύπο δεδομένων ΑΤΔ **Polynomial** για πολυώνυμα μιας μεταβλητής με ακέραιους συντελεστές και θετικούς εκθέτες (πχ. $1+2x+5x^2$) του οποίου το συμβόλαιο υποστηρίζει τις παρακάτω λειτουργίες:

- *κατασκευή* ενός πολυωνύμου δίνοντας σαν παράμετρο έναν πίνακα ακεραίων που αναπαριστά όλους τους συντελεστές και εκθέτες των όρων του πολυωνύμου.
  Παράδειγμα: `init(1, 98, -3, 2, 8, 7)` = $x^{98} + 8x^7 - 3x^2$ και `init(3, 4, 1, 2)` = $3x^4 + x^2$
- επιστροφή του *βαθμού* του πολυωνύμου
  Παράδειγμα: Αν p1 = $x^2 + 3x$ τότε **degree**(p1) = 2 και αν p1=0 τότε **degree**(p1) = 0
- επιστροφή του *συντελεστή* του όρου που έχει εκθέτη την τιμή που δίνεται σαν παράμετρος
  Παράδειγμα: Αν p1 = $x^2 + 3x$ τότε **coefficient**(2) = 1, **coefficient**(1) = 3 και **coefficient**(5) = 0
- *πρόσθεση* δύο πολυωνύμων

Παράδειγμα: Αν p1=$x^2 + 3x$ και p2=$2x^2 + 5$ τότε **add**(p1,p2)= $(x^2 + 3x) + (2x^2 + 5) = 3x^2 + 3x + 5$

- *πολλαπλασιασμό* δύο πολυωνύμων
  Παράδειγμα: Αν p1=$x^2 + 3x$ και p2=$2x^2 + 5$ τότε **multiply**(p1,p2)= $(x^2 + 3x)(2x^2 + 5) = 2x^4 + 6x^3 + 5x^2 + 15x$
- *αφαίρεση* δύο πολυωνύμων
  Παράδειγμα: Αν p1=$x^2 + 3x$ και p2=$2x^2 + 5$ τότε **subtract**(p1,p2)= $(x^2 + 3x) - (2x^2 + 5) = -x^2 + 3x - 5$
- *αποτίμηση* ενός πολυωνύμου δίνοντας σαν παράμετρο την τιμή της μεταβλητής
  Παράδειγμα: Αν p1=$x^2 + 3x$ τότε **eval**(p1,2) = $2^2 + 3(2) = 4 + 6 = 10$
- έλεγχος *ισότητας* δύο πολυωνύμων
  Παράδειγμα: Αν p1=$x^2 + 3x$ και p2=$x^2 + 3x$ τότε **equals**(p1,p2)= true
- *δημιουργία ενός αντιγράφου* ενός πολυωνύμου
  Παράδειγμα: Αν p1= $x^2 + 3x$ τότε **clone**(p1)= $x^2 + 3x$
- *εκτύπωση* ενός πολυωνύμου
  Παράδειγμα: Αν p1= $x^2 + 3x$ τότε **toString**(p1) τυπώνει "x^2+3x"

5.1 (16 μονάδες) Δημιουργείστε μια διεπαφή αντικειμένων (interface) Java για την αναπαράσταση του ΑΤΔ **Poly**. Ταξινομείστε σε κατηγορίες τις παραπάνω λειτουργίες του ΑΤΔ (Accessors, Transformers, κλπ.) και δηλώστε με την μορφή σχολίων Java (κατά προτίμηση Javadoc) τις εκ των προτέρων (pre), εκ των υστέρων (post) και αμετάβλητες (invariant) συνθήκες που πρέπει να ικανοποιούνται από κάθε λειτουργία. Δώστε για κάθε μία από αυτές την υπογραφή (signature), και το αναγνωριστικό δικαιώματος πρόσβασης (visibility modifier).

Σημειώστε ότι οι κατασκευαστές αντικειμένων (constructors) του ADT δεν εμφανίζονται στην αντίστοιχη διεπαφή Java που αναπαριστά την αφαιρετική τους συμπεριφορά.

Λύση:

```
package SoftwareInterfaces;
/**
 * Represents the ADT Polynomial.
 * Interface Invariant: Once created and until destroyed,
 * this instance contains a valid Polynomial with integer
 * coefficients and non-negative exponents.
 * @author XXXXXXXXXX
 * @version 30 April 2004
 */
public interface Polynomial {

 public int degree();
 //pre: none
 //post:  returns the value (>= 0) of the largest
 //exponent in this, or 0 if this is the 0 polynomial.

 public int coefficient(int exp);
 //pre:  exp >= 0
 //post: returns the coefficient of the term whose
 //degree is exp

 public Polynomial add(Polynomial other);
```

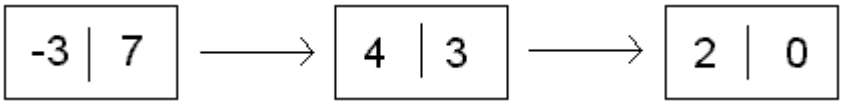
```
  //pre: other is non-null and an instance of the same
  //class as this
  //post: returns a new Polynomial that represents the sum
  //of this and other
  //this and other are unmodified

  public Polynomial multiply(Polynomial other);
  //pre: other is non-null and an instance of the same
  //class as this
  //post: returns a new Polynomial, which is the product
  //of this and other
  //this and other are unmodified

  public Polynomial subtract(Polynomial other);
  //pre: other is non-null and an instance of the same
  //class as this
  //post: returns a new Polynomial, which is the
  //subtraction of this and other
  //this and other are unmodified

  public int eval(int value);
  //pre: value is non-null
  //post: returns the result of f(value) where f(x)=this
  //this is unmodified

  public boolean equals(Polynomial other);
  //pre:  other is non-null and an instance of the same
  //class as this
  //post: returns true iff the coefficients are equal for
  //each (non-zero and zero) term in this and other

  public Object clone( )
  //pre: none
  //post: returns an identical object to this

  public String toString();
  //pre: none
  //post: returns a string representation of this
  //polynomial, of the form, (c1)x^0 + (c2)x^1 + ... +
  //(cN)x^N where cj != 0, 0<=j<= N and N = this.degree()
}
```

5.2 (24 μονάδες) Προτείνετε μια υλοποίηση για την διεπαφή αντικειμένων (interface) Java **Polynomial** που δημιουργήσατε στο ερώτημα **5.1**. Για την υλοποίηση του ΑΤΔ, μπορείτε να χρησιμοποιήσετε είτε πίνακες είτε λίστες (απλές, διπλές, κυκλικές, κλπ.). Εξηγήστε τα πλεονεκτήματα της κάθε υλοποίησης σε σχέση με τα χαρακτηριστικά ενός πολυωνύμου δηλ. με το αν το πολυώνυμο είναι πυκνό (έχει πολλούς μη-μηδενικούς συντελεστές) ή αραιό (στην αντίθετη περίπτωση) σε όρους.
Σημείωση: Για την υλοποίηση της μεθόδου **eval** μπορείτε να χρησιμοποιήσετε την στατική μέθοδο **Math.pow(**value, exponent**)** με ορίσματα τύπου double.

```
// from the Math class
public static double pow(double a, double b)
// Returns of value of the first argument (a - the base)
// raised to the power (b - the exponent) of the second //
argument (i.e., the value a^b).
```

Λύση:

If most of the coefficients $coeff_i$ are nonzero i.e., most of the terms are present (dense), we could use a simple array to store the coefficients and write the methods to perform addition, subtraction, multiplication, as well as other operations on these polynomials. However, if the polynomials are sparse, e.g., $p1 = 3x^{1000} + 5x^{14}$ or $p2 = 10x^{1990} - 20x^{1492} + 11x + 5$, then the running time is likely to be unacceptable. It becomes obvious that if arrays were used, then most of the time is spend in multiplying zeros and stepping through what amounts to nonexistent parts of the input polynomials. An alternative solution is to use a singly linked list. A list entry corresponds to a monomial $coeff_i \cdot x^{expi}$ of the polynomial and must hold its coefficient $coeff_i$ and its exponent exp_i. All entries in the list must be sorted by the exponent. Do not store a value when the coefficient is zero, and there should not be multiple coefficients with the same exponents. For example, if your linked list looked like this:



it would represent the polynomial $-3x^7 + 4x^3 + 2$.

```
/*******Solution with table: */
// This is the implementation class for the interface
// Polynomial.
// Implementation Invariants:
// A table of size equal to the degree of the polynomial is
// created and every coefficient is stored in the corresponding
// position of the table. For example 3*x^189 is stored in a
// table of size 190 and each cell has the (default) value 0
// except of the last one that has the value 3. The trivial
// polynomial is the one that has 0 in all cells.

public class MyPolynomial implements Polynomial {
	private int[] coeffs;

	/**
	 * Create a new instance of a polynomial.
	 * The parameter is a table of a sequence of integers
	 * where the first integer is the coefficient and the
	 * second is the corresponding exponent. For example,
	 * [1,2,4,1] is the 1*x^2 + 4*x^1 polynomial.
	 * @param params[] the table of the coefficients and
	 * exponents
	 */
	public MyPolynomial(int params[]) {
		int degree = 0;
		if (params.length < 2) {
			coeffs = new int[0]; // the trivial polynomial
			return;
		}
		//find the degree of the polynomial;
		for (int i = 1; i < params.length; i+=2) {
			if (params[i] > degree) {
				degree = params[i];
			}
		}
		coeffs = new int[degree+1];
		for (int i = 0; i < params.length; i+=2) {
			coeffs[params[i+1]] = coeffs[params[i+1]] +
params[i];
		}
	}
```

```
	/**
	 * Returns the degree of this polynomial.
	 *
	 * @return the degree of this polynomial
	 */
	public int degree() {
		return coeffs.length-1;
	}

	/**
	 * Returns the corresponding coefficient of the given
	 * exponent.
	 * @param exponent the exponent whose coefficient is to
	 * be returned, exponent must be &gt 0.
	 * @return the coefficient of the given exponent
	 */
	public int coefficient(int exponent) {
		return coeffs[exponent];
	}

	/**
	 * Adds to this polynomial an other one and returns the
	 * sum.
	 * @param other the polynomial to be added to this
	 * @return a new polynomial that is the sum of this and
	 * other polynomials
	 */
	public MyPolynomial add(MyPolynomial other) {
		int maxdegree;
		if (coeffs.length > other.coeffs.length)
			maxdegree = coeffs.length;
		else
			maxdegree = other.coeffs.length;
		int[] addcoeffs = new int [maxdegree];
		for (int i = 0; i < maxdegree; i++) {
			addcoeffs[i] = coeffs[i] + other.coeffs[i];
		}

		return ((new MyPolynomial()).coeffs = addcoeffs);
	}

	/**
	 * Myltiplies this Polynomial to other.
	 *
	 * @param other the polynomial to be multiplied to this
	 * @return a new polynomial that is the product of this
	 * and other polynomial
	 */
	public MyPolynomial multiply(MyPolynomial other) {
	     int degree = coeffs.length + other.coeffs.length - 1;
	     int[] multicoeffs = new int[degree];
		 for (int i = 0; i < coeffs.length; i++) {
		   for (int j = 0; j < other.coeffs.length; j++) {
		      multicoeffs[i+j] =
		          multicoeffs[i+j]+coeffs[i]*other.coeffs[j];
		    }
		 }
		 return ((new MyPolynomial()).coeffs = multicoeffs);
	}

	/**
	 * same as add
	 */
	public MyPolynomial subtract(MyPolynomial other) {
		return null; //same as add
	}

	/**
	 * Evaluates the polynomial for a given value
```

```
	 *
	 * @param value the value of x
	 * @return the result of the evaluation
	 */
	public int eval(int value) {
		double res = 0.0;
		for (int i = 0; i < coeffs.length; i++) {
			res = res + coeffs[i]*Math.pow((double)value,
(double)i);
		}
		return (int) res;
	}

	/**
	 * Checks if this polynomial is equal to the given one.
	 * In order for two polynomials to be equal, they must
	 * be of the same degree and the coefficients must be
	 * equal for each (non-zero and zero) exponent.
	 *
	 * @param other the polynomial to be checked for
	 * equality
	 * @return true if this polynomial is equal to other,
	 * false else
	 */
	public boolean equals(MyPolynomial other) {
		if (degree() != other.degree()) {
			return false;
		}

		for (int i = 0; i < coeffs.length; i++) {
			if (coeffs[i] != other.coeffs[i]) {
				return false;
			}
		}
		return true;
	}

	/**
	 * Creates a string representation of this polynomial.
	 * The string is of the form, (c1)x^0 + (c2)x^1 + ... +
	 * (cN)x^N where cj != 0, 0&lt;=j&lt;=N and N =
	 * this.degree().
	 *
	 * @return the string representation of this polynomial
	 */
	public String toString() {
		String str = "";

		if (coeffs.length = 0) {
			str = "0";
		} else {
		 for (int i = 0; i < coeffs.length; i++) {
			int coeff = ((Integer)it.next()).intValue();
			int exp = ((Integer)it.next()).intValue();
			str = str + "(" + coeffs[i] + ")" + "x^" + i;
			if (i != (coeffs.length - 1)) {
				str = str + " + ";
			}
		}
		}
		return str;
	}
}
```

```
/*******Solution with list: */

import java.util.*;

// This is the implementation class for the interface
//Polynomial.
// Implementation Invariants:
// All monomial terms are sorted by the exponent
// Every coefficients and exponent pair is stored sorted
// by the exponent in a linked list. If the coefficient
// is zero nothing is stored, thus an empty  list is
// the trivial polynomial (p = 0).

public class MyPolynomial implements Polynomial {
      private java.util.LinkedList list;
      private int degree;

      /**
       * Create a new instance of a polynomial.
       * The parameter is a table of a sequence of integers
       * where the first integer is the coefficient and the
       * second is the corresponding exponent. For example,
       * [1,2,4,1] is the 1*x^2 + 4*x^1 polynomial.
       * @param params[] the table of the coefficients and
       * exponents
       */
      public MyPolynomial(int params[]) {
            degree = -1;
            list = new java.util.LinkedList();
            if ((params.length == 0) ||
                (params.length % 2 != 0)) {
                  return ;
            }

            for (int i = 0; i < params.length; i += 2) {
                  int coeff = params[i];
                  int exp = params[i+1];

                  if ((coeff == 0) || (exp < 0)) {
                        continue;
                  }

                  boolean inserted = false;
                  for (int j = 0; j < list.size(); j += 2) {
                        int list_exp =
                          ((Integer)list.get(j+1)).intValue();
                        if (exp > list_exp) {
                              list.add(j, new Integer(coeff));
                              list.add(j+1, new Integer(exp));
                              inserted = true;
                              break;
                        } else if (exp == list_exp) {
                              Integer list_coeff
                                    =(Integer)list.remove(j);
                              list.add(j, new
                                 Integer(list_coeff.intValue()
                                          + coeff));
                              inserted = true;
                              break;
                        }
                  }

                  if (!inserted) {
                        list.add(new Integer(coeff));
                        list.add(new Integer(exp));
                  }
            }

            if (!list.isEmpty()) {
```

```
                degree = ((Integer)list.get(1)).intValue();
        }
    }

    /**
     * Returns the degree of this polynomial.
     *
     * @return the degree of this polynomial
     */
    public int degree() {
        return degree;
    }

    /**
     * Returns the corresponding coefficient of the given
     * exponent.
     * @param exponent the exponent whose coefficient is to
     * be returned
     * @return the coefficient of the given exponent
     */
    public int coefficient(int exponent) {
        if (degree < exponent)
            return 0;
        for (Iterator it = list.iterator(); it.hasNext();)
        {
            int coeff = ((Integer)it.next()).intValue();
            int exp = ((Integer)it.next()).intValue();
            if (exp == exponent)
                return coeff;
            if (exp < exponent)
                return 0;
        }
        return 0;
    }

    /**
     * Adds to this polynomial an other one and returns the
     * sum.
     * @param other the polynomial to be added to this
     * @return a new polynomial that is the sum of this and
     * other polynomials
     */
    public MyPolynomial add(MyPolynomial other) {
        if (other.list.isEmpty())
            return this.clone();
        if (this.list.isEmpty())
            return other.clone();
        LinkedList newList = new LinkedList();
        int i = 0, j = 0;
        while ( j < other.list.size() && i < list.size() ) {
            int exp =
                    ((Integer)list.get(i+1)).intValue();
            int other_exp =
              ((Integer)other.list.get(j+1)).intValue();
            if (exp == other_exp) {
                newList.add(new Integer(
                   ((Integer)list.get(i)).intValue()
                 ((Integer)other.list.get(j)).intValue())
                                          );
                newList.add(list.get(i+1));
                i += 2; j += 2;
            } else if (exp < other_exp) {
                newList.add(other.list.get(j));
                newList.add(other.list.get(j+1));
                j += 2;
            } else {
                newList.add(list.get(i));
                newList.add(list.get(i+1));
                i += 2;
```

```
                }
            }
            if (i < list.size()) {
                while (i < list.size()) {
                    newList.add(list.get(i));
                    newList.add(list.get(i+1));
                    i += 2;
                }
            } else if (j < other.list.size()) {
                while (j < other.list.size()) {
                    newList.add(other.list.get(j));
                    newList.add(other.list.get(j+1));
                    j += 2;
                }
            }

            MyPolynomial sum = new MyPolynomial(new int[0]);
            sum.list = newList;
            return sum;
        }

        /**
         * Myltiplies this Polynomial to other.
         *
         * @param other the polynomial to be multiplied to this
         * @return a new polynomial that is the product of this
         * and other polynomial
         */
        public MyPolynomial multiply(MyPolynomial other) {
            int params[] = new int[(this.degree +
                                    other.degree)*2];
            int k = 0;
            for (int i = 0; i < list.size(); i += 2) {
              for (int j = 0; j < other.list.size(); j += 2) {
                params[k]=((Integer)list.get(i)).intValue() *
                    ((Integer)other.list.get(j)).intValue();
                params[k+1] =
                    ((Integer)list.get(i+1)).intValue() +
                   ((Integer)other.list.get(j+1)).intValue();
                      k += 2;
                }
            }
            return new MyPolynomial(params);
        }

        /**
         */
        public MyPolynomial subtract(MyPolynomial other) {
            return null; } //same as add
        /**
         * Evaluates the polynomial for a given value
         *
         * @param value the value of x
         * @return the result of the evaluation
         */
        public int eval(int value) {
            double res = 0.0;
            for (Iterator it=list.iterator(); it.hasNext();) {
                double coeff =
                            ((Integer)it.next()).doubleValue();
                double exp =
                            ((Integer)it.next()).doubleValue();
                res = res + coeff*Math.pow((double)value, exp);
            }
            return (int)res;
        }
```

```
	/**
	 * Checks if this polynomial is equal to the given one.
	 * In order for two polynomials to be equal, they must
	 * be of the same degree and the coefficients must be
	 * equal for each (non-zero and zero) exponent.
	 *
	 * @param other the polynomial to be checked for
	 * equality
	 * @return true if this polynomial is equal to other,
	 * false else
	 */
	public boolean equals(MyPolynomial other) {
		if (this.degree != other.degree) {
			return false;
		}

		for (int i = 0; i < list.size(); i += 2) {
		  if ( (((Integer)list.get(i)).intValue() !=
		     ((Integer)other.list.get(i)).intValue()) ||
		     (((Integer)list.get(i+1)).intValue() !=
		      ((Integer)other.list.get(i+1)).intValue()) ) {
				return false;
			}
		}
		return true;
	}

	/**
	 * Creates a string representation of this polynomial.
	 * The string is of the form, (c1)x^0 + (c2)x^1 + ... +
	 * (cN)x^N where cj != 0, 0&lt;=j&lt;=N and N =
	 * this.degree().
	 *
	 * @return the string representation of this polynomial
	 */
	public String toString() {
		String str = "";

		if (list.isEmpty()) {
			str = "0";
		} else {
		 for (Iterator it=list.iterator(); it.hasNext();) {
			int coeff = ((Integer)it.next()).intValue();
			int exp = ((Integer)it.next()).intValue();
			str = str + "(" + coeff + ")" + "x^" + exp;
			if (it.hasNext()) {
				str = str + " + ";
			}
			}
		}
		return str;
	}
}
```