Πανεπιστήμιο Κρήτης
Τμήμα Επιστήμης Υπολογιστών

ΗΥ-252 – Οντοκεντρικός Προγραμματισμός
Βασίλης Χριστοφίδης

Πρόοδος (3 ώρες)
Ημερομηνία: 06 Απριλίου 2002

Ονοματεπώνυμο:
Αριθμός Μητρώου:

Άσκηση 1 (6 μονάδες)

Θεωρήστε τις παρακάτω κλάσεις Java

```
public class Wide {
  public int a;
  public Wide() { a = 1; }
  public int getA() { return a; }
}

public class Narrow extends Wide {
  public int a;
  public Narrow() { a = 3; }
  public int getA() { return 2 * a; }
}
```

Ποια είναι η έξοδος του παρακάτω κώδικα και γιατί?

```
Wide wide = new Wide();
Wide wideNarrow = new Narrow();
Narrow narrow = new Narrow();
System.out.println(wide.getA());
System.out.println(wideNarrow.getA());
System.out.println(narrow.getA());
System.out.println(wide.a);
System.out.println(wideNarrow.a);
System.out.println(narrow.a);
```

Λύση: The `println()` statements print the following numbers:
1
6 (uses Narrow's `getA()` that overrides Wide's)
6
1
1 (uses Wide's `a`)
3 (uses Narrow's `a` that showdows Wide's)

Άσκηση 2 (10 μονάδες)

Θεωρείστε τη δήλωση: `Object o = “abc”;`

Η κλάση **`Object`** βρίσκεται στην κορυφή της ιεραρχίας όλων των κλάσεων. Κάθε άλλη κλάση, συμπεριλαμβανομένων και αυτών που ορίζονται από τον προγραμματιστή, είναι απόγονός της. Στην κλάση **`Object`** ορίζεται και υλοποιείται η βασική συμπεριφορά που πρέπει να έχει κάθε κλάση στο Java σύστημα. Μερικές από αυτές τις βασικές μεθόδους είναι η **`equals(Object),`** που συγκρίνει αν δύο αντικείμενα είναι ίδια και η **`toString()`** που επιστρέφει την αναπαράσταση του αντικειμένου σε μορφή συμβολοσειράς.

1. Για κάθε μια από τις εντολές του παρακάτω προγράμματος, αναφέρετε αν θα συμβεί λάθος κατά την ώρα της μεταγλώττισης (compile-time error) και γιατί.
2. Για όσες από τις παρακάτω εντολές περάσουν επιτυχώς το στάδιο της μεταγλώττισης, αναφέρετε αν παράγεται exception και γιατί.
3. Για όσες από τις εντολές του ερωτήματος 2 δεν παράγεται exception βρείτε το αποτέλεσμά τους.

```
a. boolean b = o.equals(“a, b, c”);
b. char c = o.charAt(1);
c. Object o2 = b;
d. String s = o;
e. String t = (String) o;
f. c = t.charAt(1);
g. c = t.charAt(3);
```

Λύση:

Ερώτημα α

a. Valid

b. Method **`charAt(int)`** not found in class **`java.lang.Object`**

c. Incompatible type for declaration. Can't convert **`boolean`** to **`java.lang.Object`**

d. Explicit cast needed to convert **`java.lang.Object`** to **`java.lang.String`**

e. Valid

f. If **`c`** is considered not defined invalid else valid

g. If **`c`** is considered not defined invalid else valid

Ερώτημα β

`c=t.charAt(3);`⇒ **`java.lang.StringIndexOutOfBoundsException`**

Ερώτημα γ

`t ⇒ abc`

`c ⇒ a`

Άσκηση 3 (40 μονάδες)

Ο Java Πρωτάρης, είναι ενας αρχάριος προγραμματιστής που έχει δημιουργήσει τις παρακάτω κλάσεις και μεθόδους για την αναπαράσταση γεωμετρικών σχημάτων (θεωρείστε ότι ήδη υπάρχει η κλάση **`Point2D`** εφοδιασμένη με τον κατασκευαστή αντικειμένων **`Point2D(x:int,y:int)`**).

3.1 (16 μονάδες) Συμπληρώστε τα κενά στον παρακάτω ατελή κώδικα χρησιμοποιώντας μόνο τον διαθέσιμο χώρο που σας δίνεται στην εκφώνηση. Πιο συγκεκριμένα:

a. Γράψτε τον κώδικα του κατασκευαστή αντικειμένων (constructor) της κλάσης **`Rectangle`**
b. Γράψτε τον κώδικα όλων των μεθόδων της διεπαφής (interface) **`Shape`** που υλοποιεί η κλάση **`Rectangle`**
c. Γράψτε τον αντίστοιχο κώδικα για τις μεθόδους της κλάσης **`Circle`**

```
// Shape models a two-dimensional geometric shape
public interface Shape
{
// getLocationOf() returns the shape's location
     public Point2D getLocationOf();

// area() returns the shape's area
    public double area();

// equals method returns true if the shapes are equal
    public boolean equals(Object obj)
}
// Rectangle class models a rectangle shape
public class Rectangle implements Shape
{ private int length, width;
  private Point2D topLeft;
  // constructor: given point is top left corner
  public Rectangle(Point2D p, int long, int wide){
```

Λύση:
```
topLeft = p;
length = long;
width = wide;
```

```
  }
  public Point2D getLocationOf(){
```

Λύση:
```
return topLeft;
```

```
  }
  public int getLength(){
```

Λύση:
```
return length;
```

```
  }
```

```
  public int getWidth(){
```

Λύση:
```
 return width;
```

```
 }
  public double area(){
```

Λύση:
```
return (length * width);
```

```
 }
 // equals method returns true if the two rectangles are
 // congruent i.e., one can be picket up and placed
 // exactly on top of the other. Think carefully!
  public boolean equals(Object obj){
```

Λύση:
```
 if (! obj instanceof Rectangle) return false;
 else return((length = (Rectangle)obj.getLength()
              && width = (Rectangle)obj.getWidth())
         ||  (length = (Rectangle)obj.getWidth()
              && width = (Rectangle)obj.getLength())
```

```
  }
}// end of class Rectangle
// Circle class models a circular shape
public class Circle implements Shape
{   private int radius;
    private Point2D center;
    // constructor: given point is center of the circle
    public Circle(Point2D p, int rad){
```

Λύση:
```
 center = p;
 radius = rad;
```

```
    }
    public Point2D getLocationOf(){
```

Λύση:
```
return center;
```

```
    }
    public int getRadius(){
```

Λύση:
```
 return radius;
```

```
    }
    public double area(){
```

Λύση:
```
 return Math.PI * radius * radius;
```

```
    }
```

```
 // equals method return true if the two circles are
 // congruent i.e., one can be picket up and placed
 // exactly on top of the other.
    public boolean equals(Object obj){
```

Λύση:
```
 if (!obj instanceof Circle) return false;
 else return (radius = (Circle)obj.radius);
```

```
    }
}// end of class Circle
```

Ο αρχάριος προγραμματιστής μας, Java Πρωτάρης, έχει επίσης δημιουργήσει μια κλάση **ClientShapes** που χρησιμοποιεί τις κλάσεις του υποερωτήματος 3.1.

3.2 (24 μονάδες) Υποθέστε τώρα ότι είστε Βοηθοί στο μάθημα HY252 και προσπαθείτε να βοηθήσετε τον Java Πρωτάρη απαντώντας στα παρακάτω ερωτήματα και χρησιμοποιώντας μόνο τον διαθέσιμο χώρο που σας δίνεται στην εκφώνηση.

```
public class ClientA {

   public static void main(String args[]) {
```

a. Δηλώστε μια μεταβλητή αναφορών, με όνομα **shapes**, σε αντικείμενα τύπου πίνακας από στοιχεία τύπου **Shape** και δημιουργήστε ένα στιγμιότυπο πίνακα με 5 στοιχεία για την μεταβλητή shapes.

Λύση:
```
  Shape [] shapes = new Shape[5];
```

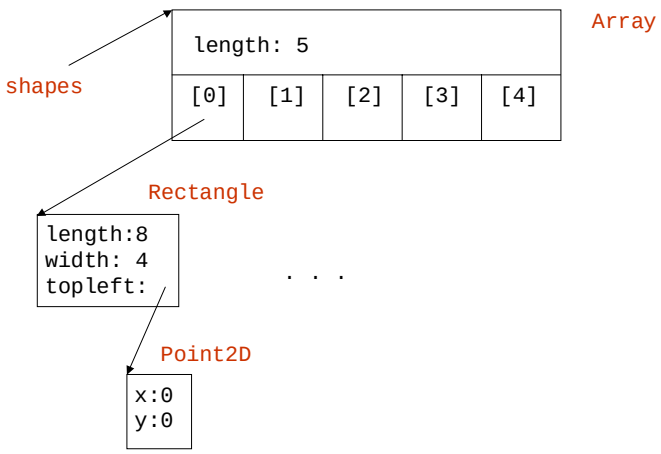
b. Αρχικοποιείστε τον πίνακα shapes που δημιουργήσατε με αντικείμενα που έχουν τους παρακάτω τύπους και τιμές: (1) a **Rectangle** of length 8 and width 4 with top left at the origin (0,0), (2) a **Rectangle** of length 8 and width 4 with top left at the point (1,1), (3) a **Rectangle** of length 4 and width 8 with top left at (1,2), (4) a **Rectangle** of length 4 and width 4 with top left at (1,2), and (5) a **Cycle** of radius 5 centred at (1,1).

Λύση:
```
shapes[0] = new Rectangle(new Point2D(0,0),8,4);
shapes[1] = new Rectangle(new Point2D(1,1),8,4);
shapes[2] = new Rectangle(new Point2D(1,2),4,8);
shapes[3] = new Rectangle(new Point2D(1,2),4,4);
shapes[4] = new Circle(new Point2D(1,1),5);
```

```
  }
```

c. Δώστε ένα διάγραμμα που απεικονίζει την κατάσταση (state) του αντικειμένου πίνακα που αναφέρει η μεταβλητή **shapes** μετά από την παραπάνω αρχικοποίηση

Λύση:



d. Γράψτε έναν βρόγχο (loop) ο οποίος τυπώνει στην στάνταρ έξοδο τις τιμές του εμβαδού (areas) κάθε γεωμετρικής μορφής που περιέχει ο πίνακας **shapes**.

Λύση:
```
for(int i =0; i < shapes.length; i++)
          {System.out.println(shapes[i].area());}
```

e. Δώστε την υλοποίηση της μεθόδου **checkItOut(Shape myshape, Shapes[] shapes)** στην κλάση **ClientShapes** η οποία καλούμενη ως ακολούθως
checkItOut(new Rectangle(new Point2D(0,0),8,4),shapes);
τυπώνει:
```
true;
true;
true;
false;
false;
```

Λύση:
```
public static void checkItOut(Shape myshape,
                              Shapes [] shapes){
  for (int i=0; i < shapes.length; i++)
  System.out.println(myshape.equals(shapes[i]));
}
```

f. Τι θα απαντούσατε σαν Βοηθοί του μαθήματος HY252 στις παρακάτω ερωτήσεις του Java Πρωτάρη:

I. Γιατί χρειαζόμαστε την διεπαφή (interface) **Shape** αφού όλες οι μέθοδοι υλοποιούνται στις κλάσεις **Rectangle** και **Circle**?

Λύση: Προγραμματισμός βασισμένος σε συμβόλαια
Μπορούμε να έχουμε πολλαπλές υλοποιήσεις του ADT **`Shape`** από διαφορετικές κλάσεις Java, όλοι όμως οι πελάτες του ADT χρησιμοποιούν την κοινή αφαιρετική του συμπεριφορά όπως αυτή ορίζεται απο την αντίστοιχη διεπαφή Java. Αυτή η κοινή συμπεριφορά θα μπορούσε να οριστεί και σε μια αφαιρετική κλάση. Σε αυτή την περίπτωση οι κλάσεις **`Rectangle`** και **`Circle`** αντί να υλοποιούν την διεπαφή θα εξειδίκευαν την αφαιρετική κλάση. Κάτι τέτοιο δεν είναι πάντα δυνατό γιατί οι συγκεκριμένες κλάσεις μπορούν να συνδυάζουν πολλές αφαιρετικές συμπεριφορές (απο πολλούς ADT) και η πολλαπλή κληρονομικότητα δεν υποστηρίζεται απο την Java

II. Γιατί οι μεταβλητές στιγμιοτύπων (instance variable) των 2 κλάσεών μας σε αντίθεση με τις μεθόδους τους δεν πρέπει (και δεν είναι) δηλωμένες σαν **`public`**?

Λύση:Προγραμματισμός που σέβεται την αρχή της ενθυλάκωσης
Θέλουμε να προστατεύσουμε την κατάσταση των αντικειμένων μιας κλάσης απο μή εξουσιοδοτημένες προσβάσεις/αλλαγές. Μόνο μέσα απο τις μεθόδους μιας κλάσης μπορούμε να ελέγχουμε με συνέπεια τις **`invariant`** και **`post`** συνθήκες που πρέπει να ικανοποιεί η κατάσταση των αντικειμένων της κλάσης.

III. Η μέθοδος **`checkItOut`** πρέπει να δηλωθεί σαν στατική (**`static`**) ή όχι? Σε κάθε περίπτωση εξηγείστε γιατί.

Λύση: Πρέπει να δηλωθεί σαν στατική (**`static`**) γιατί παίρνει σαν ορίσματα όλα τα δεδομένα που χειρίζεται και επιπλέον η κλάση **`ClientShapes`** δεν έχει στιγμιότυπα. Μπορεί όμως να χρησιμοποιηθεί απο την μέθοδο **`main`** που ορίσαμε στην κλάση **`ClientShapes`**

IV. Ποιο είναι το καταλληλότερο αναγνωριστικό δικαιώματος πρόσβασης (visibility modifier) για την μέθοδο **`checkItOut`**

Λύση: Πρέπει να δηλωθεί σαν δημόσια (**`public`**) γιατί η μέθοδος πρέπει να δυνατόν να χρησιμοποιεί και απο οποιαδήποτε άλλη μέθοδο του προγράμματός μας.

Άσκηση 4 (60 μονάδες)

Κάθε Αφαιρετικός Τύπος Δεδομένων (ADT) χαρακτηρίζεται από το συμβόλαιο του, που προσδιορίζει: (α) Το σύνολο των επιτρεπτών τιμών του ADT και (β) Το σύνολο των επιτρεπτών λειτουργιών του ADT. Θεωρείστε τον Αφαιρετικό Τύπο Δεδομένων **`Date`** του οποίου το συμβόλαιο δίνεται στην συνέχεια:

a) Κάθε ημερολογιακή ημέρα (calendar date) προσδιορίζεται από (α) την αριθμητική ένδειξη του χρόνου (year), (β) την αριθμητική ένδειξη του μήνα (month), και (γ) την αριθμητική ένδειξη της ημέρας (day) για κάθε μήνα. Σαν βοηθητική πληροφορία μπορούμε να θεωρήσουμε την ονομασία των ημερών (π.χ. Σάββατο).

b) Η διεπαφή χρήσης του ADT **Date** περιλαμβάνει τις παρακάτω λειτουργίες (operations):

a. **init(y, m, d): Int x Int x Int-> Date**
b. **setDate(y, m, d): Date x Int x Int x Int-> Date**
c. **advance(n): Date x Int -> Date**
d. **getDay(): Date -> Int**
e. **getMonth(): Date -> Int**
f. **getYear(): Date -> Int**
g. **getWeekday(): Date -> String**
h. **equals(D'): Date x Date -> Boolean**
i. **daysBetween(D'): Date x Date -> Int**
j. **toString(): Date -> String**

4.1 (20 μονάδες) Δημιουργείστε μια διεπαφή αντικειμένων (interface) Java για την αναπαράσταση του ADT **Date**. Ταξινομείστε σε κατηγορίες τις παραπάνω λειτουργίες του ADT (Accessors, Transformers, κλπ.) και δηλώστε με την μορφή σχολίων Java (κατά προτίμηση σχολίων Javadoc) τις **pre, post** και **invariant** συνθήκες που πρέπει να ικανοποιούνται απο κάθε λειτουργία. Δώστε για κάθε μία από αυτές την υπογραφή (signature), το αναγνωριστικό δικαιώματος πρόσβασης (visibility modifier) καθώς και τις εξαιρέσεις (exceptions) που μπορούν να δημιουργούν οι παραβιάσεις των προηγούμενων συνθηκών.

```
package SoftwareInterfaces;
/**
 * This is the interface for the ADT Date.
 *
 * Interface Invariant: Once created and until destroyed, this
 * instance contains a valid date: getYear() != 0 &&
 * 1 <= getMonth() <= 12 && 1 <= getDay() <= #days in
 * getMonth().
 * Also calendar date getMonth()/getDay()/getYear() does not
 * fall in the gap formed by the change to the modern
 * (Gregorian) calendar.
 * @author XXXXXXXXXX
 * @version 06 April 2002
 */

public interface Date
{
/**
   * Set the year, month and day of a date D (Transformer).
   * @param y the year
   * @param m the month
   * @param d the day
   * @throws IllegalArgumentException (e.g., if precondition
   * not satisfied)
   * <dt><b>Preconditon:</b>
   * <dd><code>y != 0 && 1 <= m <= 12 && 1 <= d <= #days</code>in
   * month and ><code>(y,m,d)></code> does not fall in the gap
   * formed by the change to the modern (Gregorian) calendar
   * <dt><b>Postcondition:</b>
   * <dd><code>D</code> is a valid instance of Date.
   */
   public void setDate(int y, int m, int d)
                                    throws IllegalArgumentException;
/**
   * Advance a date D with a number of days (Transformer).
   * @param n the number of days
   * <dt><b>Preconditon:</b>
   * <dd><code>D</code> is a valid instance of Date
   * <dt><b>Postcondition:</b>
```

```
  * <dd><code>D</code> is a valid instance of Date.
  */
  public void advance(int n);

/**
 * Get the day of a date D (Accessor).
 * @return the day d of the month from D
 * <dt><b>Preconditon:</b>
 * <dd><code>D</code> is a valid instance of Date
 * <dt><b>Postcondition:</b>
 * <dd><code> 1 <= d <= #days</code> in month getMonth(D)
 */
  public int getDay();

   /**
 * Get the month of a date D (Accessor).
 * @return the the month m from D
 * <dt><b>Preconditon:</b>
 * <dd><code>D</code> is a valid instance of Date
 * <dt><b>Postcondition:</b>
 * <dd><code>1 <= m <= 12</code>
 */
  public int getMonth();

/**
 * Get the year of a date D (Accessor).
 * @return the year y from D
 * <dt><b>Preconditon:</b>
 * <dd><code>D</code> is a valid instance of Date
 * <dt><b>Postcondition:</b>
 * <dd><code> y != 0 </code>
 */
  public int getYear();

/**
 * Get the month day of a date D (Accessor).
 * @return the day wd of the week upon which D falls
 * <dt><b>Preconditon:</b>
 * <dd><code>D</code> is a valid instance of Date
 * <dt><b>Postcondition:</b>
 * <dd><code> 0=Sunday, 1=Monday, ..., 6=Saturday </code>
 */
  public String getMonthDay();

/**
 * Check if date D and D' are equal (Accessor).
 * @param the date D' to compare
 * @return the eq Boolean value of the check
 * <dt><b>Preconditon:</b>
 * <dd>None
 * <dt><b>Postcondition:</b>
 * <dd><code>eq is true</code> if and only if D and D' denote the
 * same calendar date.
 */
  public boolean equals(Date D');

/**
 * Get the days between a date D and D' (Accessor).
 * @return the number of calendar days d from D to D'
 * <dt><b>Preconditon:</b>
 * <dd><code>D, D'</code> are valid instances of Date
 * <dt><b>Postcondition:</b>
 * <dd><code>equals(D', D.advance(d))</code> would be true
 */
  public int daysBetween (Date D');
```

```
   /**
    * Get a string representation of a date D (Accessor).
    * @return a string sd from a date D
    * <dt><b>Preconditon:</b>
    * <dd><code>D, D'</code> are valid instances of Date
    * <dt><b>Postcondition:</b>
    * <dd> sd is the date D expressed in the format
    * <code> Date[getYear(D),getMonth(D),getDay(D)]</code>
    */
     public String toString();
  }// End of Date Interface
```

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

4.2 (40 μονάδες) Προτείνετε μια υλοποιήση για την διεπαφή αντικειμένων (interface) Java Date που δημιουργήσατε στο ερώτημα **4.1**. Μπορείτε να χρησιμοποιήσετε την κλάση **`GregorianCalendar`** από το Java API για να πάρετε την σημερινή ημερομηνία καθώς και ιδιωτικές (**`private`**) μεθόδους για να μετατρέψετε ημερομηνίες απο το Ιουλιανό στο Γρεγοριανό ημερολόγιο και αντιστρόφως.

```
import java.util.*;
import java.io.*;

public class MyDate implements Date
{
    // This is the implementation class for the interface Date.
    // This class implementation is adapted from the Day class in
    // Horstmann and Cornell, Core Java 1.2: Volume I - Fundamentals
    // (Fourth Edition), Prentice Hall, 1999.

    // Implementation Invariants:
    // year != 0 && 1 <= month <= 12 && 1 <= day <= #days in month
    // (year,month,day) not in gap formed by the change to the
    // modern (Gregorian) calendar

    private int year;
    private int month;
    private int day;

  //  Constants for days of the week

    public static final int SUNDAY    = 1;
    public static final int MONDAY    = 2;
    public static final int TUESDAY   = 3;
    public static final int WEDNESDAY = 4;
    public static final int THURSDAY  = 5;
    public static final int FRIDAY    = 6;
    public static final int SATURDAY  = 7;

  //Public Methods

  // Constructors

    public MyDate()
    // Pre:   true
    // Post:  the new instance's day, month, and year set to today's
    //        date (i.e., the date of creation of the instance)

    {   GregorianCalendar todaysDate = new GregorianCalendar();
        year  = todaysDate.get(Calendar.YEAR);
        month = todaysDate.get(Calendar.MONTH) + 1;
```

```
      day   = todaysDate.get(Calendar.DAY_OF_MONTH);
  }

  public MyDate(int y, int m, int d)
                                  throws IllegalArgumentException
  // Pre:   y != 0 && 1 <= m <= 12 && 1 <= d <= #days in month m
  //        (y,m,d) does not fall in the gap formed by the
  //        change to the modern (Gregorian) calendar.
  // Post:  the new instance's day, month, and year set to y, m,
  //         and d, respectively
  //Exception: IllegalArgumentException if y m d not a valid date
  {   year  = y;
      month = m;
      day   = d;
      if (!isValid())
          throw new IllegalArgumentException();
  }

// Transformers

  public void setDate(int y, int m, int d)
                                   throws IllegalArgumentException
  // Pre:   y != 0 && 1 <= m <= 12 && 1 <= d <= #days in month m
  //        (y,m,d) does not fall in the gap formed by the
  //        change to the modern (Gregorian) calendar.
  // Post:  this instance's day, month, and year set to y, m,
  //         and d, respectively
  // Exception: IllegalArgumentException if y m d not a valid date
  {   year  = y;
      month = m;
      day   = d;
      if (!isValid())
          throw new IllegalArgumentException();
  }

  public void advance(int n)
  // Pre:   true
  // Post:  this instance's date moved n days later.  (Negative n
  //         moves to an earlier date.)
  {   fromJulian(toJulian() + n);
  }

// Accessors

  public int getDay()
  // Pre:   true
  // Post:  returns the day from this instance, where
  //            1 <= getDay() <= #days in this instance's month
  {   return day;
  }

  public int getMonth()
  //  Pre:   true
  //  Post:  returns the month from this instance's date, where
  //             1 <= getMonth() <= 12
  {   return month;
  }

  public int getYear()
  //  Pre:   true
  //  Post:  returns the year from this instance's date, where
  //             getYear() != 0
  {   return year;
  }

  public int getWeekday()
  // Pre:   true
  // Post:  returns the day of the week upon which this instance
  //        falls, where 1 <= getWeekday() <= 7;
```

```
  //        1 == Sunday, 2 == Monday, ..., 7 == Saturday
  {   //  calculate day of week
      return (toJulian() + 1) % 7 + 1;
  }

  public boolean equals(MyDate D')
  // Pre:   D' is a valid instance of MyDate
  // Post:  returns true if and only if this instance and instance
  //             D' denote the same calendar date
  {   return (year == D'.getYear() && month == D'.getMonth()
              && day == D'.getDay());
  }

  public int daysBetween(MyDate D')
  // Pre:   D' is a valid instance of MyDate
  // Post:  returns the number of calendar days from the dd
  //        instance's date to this instance's date, where
  //        equals(D'.advance(n)) would hold
  {   //  implementation code
      return toJulian() - D'.toJulian();
  }

  public String toString()
  //  Pre:   true
  //  Post:  returns this instance's date expressed in the format
  //             "Day[year,month,day]"
  {
      return "Date[" + year + "," + month + "," + day + "]";
  }

//  Private Methods

  private boolean isValid()
  //  Pre:   true
  //  Post:  returns true iff this is a valid date
  {   MyDate t = new MyDate();
      t.fromJulian(this.toJulian());
      return t.day == day && t.month == month
             && t.year == year;
  }

  private int toJulian()
  //  Pre:   true
  //  Post:  returns Julian day number that begins at noon of this
  //  day
  //  A positive year signifies A.D., negative year B.C.
  //  Remember that the year after 1 B.C. was 1 A.D. (i.e., no
  //  year 0).
  //  A convenient reference point is that May 23, 1968, at noon
  //  is Julian day 2440000.
  //
  //  Julian day 0 is a Monday.
  //
  //  This algorithm is from Press et al., Numerical Recipes
  //  in C, 2nd ed., Cambridge University Press 1992.
  //
  {   int jy = year;
      if (year < 0)
          jy++;
      int jm = month;
      if (month > 2)
          jm++;
      else
      {   jy--;
          jm += 13;
      }
      int jul = (int) (java.lang.Math.floor(365.25 * jy)
            + java.lang.Math.floor(30.6001*jm) + day + 1720995.0);
```

```
        int IGREG = 15 + 31*(10+12*1582);
            // Gregorian Calendar adopted Oct. 15, 1582

        if (day + 31 * (month + 12 * year) >= IGREG)
           // change over to Gregorian calendar
        {   int ja = (int)(0.01 * jy);
            jul += 2 - ja + (int)(0.25 * ja);
        }
        return jul;
    }

    private void fromJulian(int j)
    //  Pre:   true
    //  Post:  this calendar Day is set to Julian date j
    //
    //  This algorithm is from Press et al., Numerical Recipes
    //  in C, 2nd ed., Cambridge University Press 1992
    //
    {   int ja = j;

        int JGREG = 2299161;
            /* the Julian date of the adoption of the Gregorian
               calendar
            */

        if (j >= JGREG)
        /* correct for crossover to Gregorian Calendar */
        {   int jalpha = (int)(((float)(j - 1867216) - 0.25)
                / 36524.25);
            ja += 1 + jalpha - (int)(0.25 * jalpha);
        }
        int jb = ja + 1524;
        int jc = (int)(6680.0 + ((float)(jb-2439870) - 122.1)
                 /365.25);
        int jd = (int)(365 * jc + (0.25 * jc));
        int je = (int)((jb - jd)/30.6001);
        day = jb - jd - (int)(30.6001 * je);
        month = je - 1;
        if (month > 12)
            month -= 12;
        year = jc - 4715;
        if (month > 2)
            --year;
        if (year <= 0)
            --year;
    }

}
```