CSC 175 – Intermediate Progamming

Lecture 3 – A Review of Programming With Classes

What is Object-Oriented Programming?

- ***Object-oriented programming*** (or ***OOP***) attempts to allow the programmer to use data objects in ways that represent how they are viewed in the real world with less attention to the implementation method.
- An ***object*** is characterized by its name, its properties (values that it contains) and its methods (procedures and operations performed on it).

Principles of OOP

There are four main principles of OOP:

- ***Data Abstraction*** - our main concern is what data represents and not how it is implemented.
- ***Encapsulation*** - Private information about an object ought not be available the outside world and used only in prespecified ways.
- ***Polymorphism*** - There may be more than version of a given function, where the different functions share a name but have different parameter lists
- ***Inheritance*** - New classes of objects can be built from other classes of objects.

Objects and Classes

- An object in Java will have ***methods*** (its own procedures) and ***properties*** (its own variables).
- There can be more than one method with a given name if their parameters lists are different.

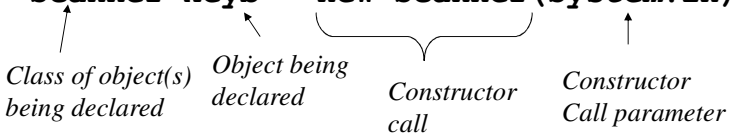
Using Standard Classes

- Java has standard classes, classes that are prewritten and supplied as part of the Java Development Kit (JDK).
- These standard classes allow you to read and write text from the keyboard and to the screen as well as to and from files, create graphic interfaces and so on.

Scanner Class

- Scanner is an example of a standard class. It allows us to use text files, the keyboard or even strings of text as sources of input.
- Like any other class, using it involves creating an instance of that class (an ***object***), and initializing it with a constructor call:

```
Scanner keyb = new Scanner(System.in);
```



Scanner Methods

Like other classes (standard and user-defined), Scanner has many methods, a few of which we have already seen:

- **`next()`** – returns a string of characters up to the next white space character in the input.
- **`nextInt()`** – returns an integer value that is next in the input.
- **`nextDouble()`** – returns an integer value that is next in the input.
- **`nextLine()`** – returns a String with the remainder of the input line.

What Is A Class?

- A ***class*** is similar to a data type, except that the data items that it contains, the data types of which they are declared and the operations performed by them are all declared by the programmer.
- An instance of a class (similar to a variable of a given type) is called an ***object***.
- The data contained within an object is called a ***property***.
- The actions (or operations) performed by an object is called a ***method***.

Declaring Objects

- To declare objects of a given class we write:

```
MyClass object1 = new MyClass();
```

- If I write

```
MyClass object2;
```

- Object2 is null (without value) until I write

```
Object2 = new myObject();
```

A Simple Class - `DateFirstTry.java`

```
public class DateFirstTry  {
  public String month;
  public int day;
  public int year; // A four-digit number

  // writeOutput() - Write the date in a
  //                 proper format
  public void writeOutput()  {
    System.out.println(month + " " + day
                       + ", " + year);
  }
}
```

TestFirstDate.java

```
public class TestFirstDate  {
  public static void main(String[] args) {
    DateFirstTry  date1, date2;

    // Call the constructors – Now we can use
    // them
    date1 = new DateFirstTry();
    date2 = new DateFirstTry();

    //Initialize date1
    date1.month = "December";
    date1.day = 31;
    date1.year = 2006;
    System.out.println("date1:");
    date1.writeOutput();
```

One instance of an object

```
    //Initialize date2
    date2.month = "July";
    date2.day = 4;
    date2.year = 1776;
    System.out.println("date2:");
    date2.writeOutput();

  }
}
```

Another instance of an object

Output

```
date1:
December 31, 2006
date2:
July 4, 1776
```

Invoking A Method

- Assume that we have these methods declared:

```
public double myMethod();
public String yourMethod();
public void ourMethod()
```

  `myMethod` and **`yourMethod`** will always return a **`double`** and a **`String`** respectively and `ourMethod` will not return any value but all three will perform some action.

Instances Of A Method

- When we invoke a method, a set of instructions are performed using the properties belonging to the object.
- **`object1.myMethod()`** uses the property values belonging to **`object1`** and **`object2.myMethod()`** belonging to **`object2.`**

DateSecondTry.java

```
import java.util.Scanner;

public class DateSecondTry  {
  private String month;
  private int day;
  private int year;  // A four-digit number

  // writeOutput() - Write the date in a proper
  //                 format
  public void writeOutput()  {
    System.out.println(month + " " + day + ", "
                       + year);
  }
```

```
  // readInput() - Read a date from the keyboard
  public void readInput()  {
    Scanner keyb = new Scanner(System.in);

    // Prompt the user for the month,
    // day and year and read them
    System.out.println("Month?");
    month = keyb.next();
    System.out.println("Day?");
    day = keyb.nextInt();
    System.out.println("Year?");
    year = keyb.nextInt();
  }
```

```
// getDay() - Allows you to get the data
//            property
public int getDay()  {
  return day;
}

// getYear() - Allows you to get the data
//             property
public int getYear()  {
  return year;
}
```

```
// getMonth() - Returns the month as a number
public int getMonth()  {
  if (month.equals("January"))
        return 1;
  if (month.equals("February"))
        return 2;
  if (month.equals("March"))
        return 3;
  if (month.equals("April"))
        return 4;
  if (month.equals("May"))
        return 5;
  if (month.equals("June"))
        return 6;
  if (month.equals("July"))
        return 7;
```

```
    if (month.equals("August"))
          return 8;
    if (month.equals("September"))
          return 9;
    if (month.equals("October"))
          return 10;
    if (month.equals("November"))
          return 11;
    if (month.equals("December"))
          return 12;
```

```
    else {
      System.out.println("Fatal error - "
                  + "not a valid month");
      // Since it isn't valid, terminate the
      // program
      System.exit(0);
      // The compilers insists on this return
      return 0;
    }
  }
}
```

Local Variables

- Just the main method can have variables that are available for its exclusive use, so can other methods within a class.

TestSecondDate.java

```
public class TestSecondDate  {
  public static void main(String[] args) {
    DateSecondTry  date = new DateSecondTry();

    //Initialize date1
    date.readInput();
```

Invoking a void method

Invoking an int method

```
    int dayNumber = date.getDay();
    System.out.println("That is the " + dayNumber
                       + "th day of the month.");
  }
}
```

DateThirdTry.java

```
import java.util.Scanner;

public class DateThirdTry  {
  private String month;
  private int day;
  private int year;  // A four-digit number

  // setDate() - Sets all three properties
  public void setDate(int newMonth, int newDay,
                                    int newYear)  {
    month = monthString(newMonth);
    day = newDay;
    year = newYear;
  }
```

Local variables (that are also formal parameters)

```
  // monthString() - Returns the string for the
  //                 month specified in numeric
  //                 format
  public String monthString(int monthNumber)  {
    switch(monthNumber)  {
      case 1: return "January";
      case 2: return "February";
      ……
      case 12: return "December";
      default:
        System.out.println("Fatal error - not a
                                  + " valid month");
        System.exit(0);
        return "Error";
    }
  }

  public void writeOutput()  {
    … … …
  }
```

```
    public void readInput()  {
… … …
  }

  public int getDay()  {
  … …
  }

  public int getYear()  {
    … …
  }

  public int getMonth()  {
    …  …
  }
}
```

```
  // getYear() - Allows you to get the data
  property
  public int getYear()  {
    return year;
  }

  // getMonth() - Returns the month as a number
  public int getMonth()  {
    …  …
  }
}
```

TestThirdData.java

```
public class TestThirdDate  {
  public static void main(String[] args) {
    Date  date = new Date();
    int year = 1882;

    date.setDate(6, 17, year);
    date.writeOutput();
  }
}
```

The actual parameters whose values are copied over into setDate's formal parameters

Output

```
June 17, 1882
```

Ownership of Methods

- Every method, whether or not there is a parameter list, has some parameters passed to it subtly – the properties belonging to that object.
- Writing `today.writeOutput()` is equivalent to writing

```
System.out.println(today.month + " "
     + today.day + ", " + today.year);
```

- Because the references to month, day and year all refer to the properties of the object that s doing the calling.

`this`

- If you wish to make it clear which that the property or method being used belongs to the calling obect, use the reserved word **`this`**.
- **`this`** serves as a hidden parameter because it is always implied when no object's name is used before a property or method.

`this` – An Example

```
// writeOutput() - Write the date in a proper
//                 format
public void writeOutput()  {
  System.out.println(month + " " + day + ", "
                                       + year);
}
```

is equivalent to:

```
// writeOutput() - Write the date in a proper
//                 format
public void writeOutput()  {
  System.out.println(this.month + " " +
this.day
         + ", " + this.year);
}
```

Methods That May Be Expected

- There are certain methods that are found quite frequently and therefore are usually expected: **`toString`** and **`equals`** are among them.

DateFourthTry.java

```
import java.util.Scanner;

public class DateFourthTry  {
  private String month;
  private int day;
  private int year;  // A four-digit number

  // toString() - Converts the date into a string
  public String toString()  {
    return (month + " " + day + ", " + year);
  }

  public void writeOutput()  {
    … …
  }
```

```
// equals() - Returns true if the parameter and
//            the object are the same data
//            Returns false otherwise
public boolean equals(DateFourthTry otherDate)
{
  return( (month.equals(otherDate.month))
           && (day == otherDate.day)
           && (year == otherDate.year));
}
```

```
// precedes() - Returns true if the object is
//              an earlier date than the
//              parameter
//              Returns false otherwise
public boolean
         precedes(DateFourthTry otherDate)  {
  return( (year < otherDate.year) ||
    (year == otherDate.year
      && getMonth() < otherDate.getMonth())
   ||(year == otherDate.year
   && month.equals(otherDate.month)
     && day < otherDate.day));
}
```

```
  public void setDate(int newMonth,
                  int newDay, int newYear)  {
    … …
  }
  public String monthString(int monthNumber)  {
    … … …
  }
  public void readInput()  {
    … …
  }
  public int getDay()  {
    … …
  }
  public int getYear()  {
    … …
  }
  public int getMonth()  {
    … …
  }
}
```

EqualsAndToStringDemo.java

```
public class EqualsAndToStringDemo  {
  public static void main(String[] args)  {
    DateFourthTry date1 = new DateFourthTry(),
                  date2 = new DateFourthTry();
    date1.setDate(6, 17, 1882);
    date2.setDate(6, 17, 1882);

    if (date1.equals(date2))
      System.out.println(date1 + " equals "
                                    + date2);
    else
      System.out.println(date1
              + " does not equal " + date2);
```

```
    date1.setDate(7, 28, 1750);

    if (date1.precedes(date2))
      System.out.println(date1 + " comes before "
                              + date2);
    else
      System.out.println(date2
           + " comes before or is equal to "
                                       + date1);
  }
}
```

Output

```
June 17, 1882 equals June 17, 1882
July 28, 1750 comes before June 17, 1882
```

Encapsulation

- **Encapsulation** – means that data belonging to a class is hidden from the classes that use is methods.
- ***Public*** properties and methods may be used by objects of other classes.
- ***Private*** properties and methods may not be used by objects of other classes.

Accessors and Mutators

- Accessor methods make private data items available to other classes.
- Mutators methods allow private data values to be changed.
- Typically, accessor methods will have names with the prefix **`get`** and mutator methods will have names with the prefix **`set`**.

DateFifthTry.java

```
import java.util.Scanner;

public class DateFifthTry  {
  private String month;
  private int day;
  private int year;  // A four-digit number

  public void writeOutput()  {
    … …
  }
```

```
// readInput() - Read a date from the keyboard
public void readInput() {
  boolean tryAgain = true;
  Scanner keyb = new Scanner(System.in);

  // Keep asking until the user enters a
  // valid date
  while (tryAgain)  {
    System.out.println
            ("Enter month, day and year");
    System.out.println("as three integers:");
    System.out.println("do not use commas " +
                  + " or other punctuations.");
    int monthInput = keyb.nextInt();
    int dayInput = keyb.nextInt();
    int yearInput = keyb.nextInt();
```

New version of readInput ensures that the data is reasonable.

```
    if (dateOK(monthInput, dayInput,
                                  yearInput)) {
      setDate(monthInput, dayInput, yearInput);
      tryAgain = false;
    }
    else
      System.out.println
             ("Illegal date.  Reenter input.");
  }
}
```

```
// setDate() - A mutator that expects the month
//             in numeric format
public void setDate(int month, int day,
                                int year)  {
  if (dateOK(month, day, year)) {
    this.month = monthString(month);
    this.day = day;
    this.year = year;
  }
  else {
    System.out.println
             ("Fatal error - invalid date");
    System.exit(0);
  }
}
```

```
// setMonth() - A mutator for month that checks
//              whether it is a valid value for
//              month
public void setMonth(int monthNumber)  {
  if ((monthNumber <= 0)&&(monthNumber > 12))  {
    System.out.println
             ("Fatal error - Invalid month");
    System.exit(0);
  }
  else
    month = monthString(monthNumber);
}
```

```
// setDay() - A mutator for month that checks
//            whether it is a valid value for day
public void setDay(int day)  {
  if ((day <= 0) && (day > 31))  {
    System.out.println
              ("Fatal error - Invalid day");
    System.exit(0);
  }
  else
    this.day = day;
}
```

```
// setYear() - A mutator for month that checks
//             whether it is a valid value for
//             year
public void setYear(int year)  {
  if ((year < 1000) && (year > 9999))  {
    System.out.println
              ("Fatal error - Invalid year");
    System.exit(0);
  }
  else
    this.year = year;
}
```

```
// equals() - Returns true if the parameter and
//            the object are the same data
//            Returns false otherwise
public boolean equals(DateFifthTry otherDate)  {
  return
    ((month.equalsIgnoreCase(otherDate.month))
           && (day == otherDate.day)
           && (year == otherDate.year));
}
```

```
// precedes() - Returns true if the object is an
//              earlier date than the parameter
//              Returns false otherwise
public boolean precedes(DateFifthTry otherDate)
{
  return( (year < otherDate.year)
   || (year == otherDate.year
        && getMonth() < otherDate.getMonth())
   || (year == otherDate.year
        && month.equals(otherDate.month)
        && day < otherDate.day));
}
```

```
// getMonth() - Returns the month as a number
public int getMonth()  {
  if (month.equalsIgnoreCase("January"))
        return 1;
  if (month.equalsIgnoreCase("February"))
        return 2;
  … …
  if (month.equalsIgnoreCase("December"))
        return 12;
  else {
    System.out.println
        ("Fatal error - not a valid month");
    //Since it isn't valid, terminate the program
    System.exit(0);
    // The compilers insists on this return
    return 0;
}
```

```
// getDay() - Accessor for the day
public int getDay()  {
  return day;
}

// getYear() - Accessor for the year
public int getYear()  {
  return year
}

// toString() - Converts the date into a string
public String toString()  {
  return (month + " " + day + ", " + year);
}
```

```
  // dateOK() - Returns true if the date is valid
  //            Returns false if the date is
  //            invalid
  //            Assumes that the month is in numeric
  //            format
  public boolean dateOK(int monthInt, int dayInt,
                                      int yearInt)  {
    return( (monthInt >= 1) && (monthInt <= 12) &&
           (dayInt >= 1) && (dayInt <= 31) &&
           (yearInt >= 1000) && (yearInt <= 9999) );
  }

  // monthString() - Returns the string for the
  //                 month specified in numeric
  //                 format
  public String monthString(int monthNumber)  {
    … …
  }
}
```

Preconditions and Postconditions

- A precondition states what is assumed to be true when the method is first called.
- A postcondition states what is assumed to be true when control is returned to the calling method.

Overloading Methods

- Frequently there can be more than one way of doing a task depending on what is available.
- Similarly there may be more than one method that shares a common name that do essentially the same job, but the details of the methods may differ; they ***must*** differ in parameter lists.This is called ***overloading*** a method.

Rules For Overloading

- Although overloaded methods can have different return types, it is essential that they have different parameter lists.
- They can differ in the number of parameters or in the type of parameters.
- Their exact algorithms can even be different, although the assumption is usually made that they are different ways of doing the same jobs with different types of data.

DataSixthTry.java

```
import java.util.Scanner;

public class DateSixthTry  {
  private String month;
  private int day;
  private int year;  // A four-digit number
```

```
  // setDate() - A mutator that expects the month
  //             in numeric format
  public void setDate(int month, int day,
                                    int year)  {
    if (dateOK(month, day, year)) {
      this.month = monthString(month);
      this.day = day;
      this.year = year;
    }
    else {
      System.out.println
                  ("Fatal error - invalid date");
      System.exit(0);
    }
  }
```

```
// setDate() - A mutator that expects the month
//             in string format
public void setDate(String monthString,
                        int day, int year)  {
  if (dateOK(monthString, day, year))  {
    this.month = monthString;
    this.day = day;
    this.year = year;
  }
  else  {
    System.out.println("Fatal error");
    System.exit(0);
  }
}
```

```
//setDate() - A mutator that assumes that the
//            day is the first day of the
//            specified year
public void setDate(int year)  {
  setDate(1, 1, year);
}

// dateOK() - Returns true if the date is valid
//            Returns false if the date is
//            invalid
//            Assumes that the month is in
//            numeric format
public boolean dateOK(int monthInt,
                  int dayInt, int yearInt)  {
  return( (monthInt >= 1) && (monthInt <= 12)
      && (dayInt >= 1) && (dayInt <= 31) &&
      (yearInt >= 1000) && (yearInt <= 9999) );
}
```

```
    // dateOK() - Returns true if the date is valid
    //            Returns false if the date is
    //            invalid
    //            Assumes that the month is in
    //            string format
    public boolean dateOK(String monthString,
                      int dayInt, int yearInt)  {
      return( monthOK(monthString)
         &&  (dayInt >= 1) && (dayInt <= 31)
       && (yearInt >= 1000) && (yearInt <= 9999) );
    }

    //monthOK() - Returns true if the month is one
    //            of the twelve valid strings
    //            Returns false other if not
    public boolean monthOK(String month)  {
      return (month.equals("January") || … …
            || month.equals("December"));
    }
…
}
```

Constructors

- A constructor is a special type of method that is called when the object is created.
- When you write

  `MyClass myObject = new MyClass();`

  you are calling the constructor.

Default Constructor

- A default constructor does not have any parameters; it assumes that some initial value must be supplied:

```
//  Date() - A default constructor for
//           the class
 public Date()  {
   month = "January";
   day = 1;
   year = 1000;
 }
```

Multiple Initializing Constructors

- An initialization constructor provides initial values for the object.
- Sometimes, there may be multiple versions, depending on the type of data supplied:

```
// Date() - A constructor that accepts the
//          month in numeric format
public Date(int monthInt, int day, int year)  {
  setDate(monthInt, day, year);
}
```

```
  // Date() - A constructor that accepts the
  //          month in string format
  public Date(String monthString, int day,
                                  int year)  {
    setDate(monthString, day, year);
  }

  //Date() - A constructor that assumes that the
  //         day is the first day of the
  //         specified  year
  public Date(int year)  {
    setDate(1, 1, year);
  }
```

ConstructorsDemo.java

```
public class ConstructorsDemo  {
  public static void main(String[] arg)  {
    Date date1 = new Date("December", 16, 1770),
         date2 = new Date(1, 27, 1756),
         date3 = new Date(1882),
         date4 = new Date();

    System.out.println("Whose birthday is "
                                  + date1 + "?");
    System.out.println("Whose birthday is "
                                  + date2 + "?");
    System.out.println("Whose birthday is "
                                  + date3 + "?");
    System.out.println("Whose birthday is "
                                  + date4 + "?");
  }
}
```

- Output

```
Whose birthday is December 16, 1770?
Whose birthday is January 27, 1756?
Whose birthday is January 1, 1882?
Whose birthday is January 1, 1000?
```

The Complete **`Date.java`**

```
import java.util.Scanner;

public class Date  {
  private String month;
  private int day;
  private int year;  // A four-digit number

  //  Date() - A default constructor for the
  class
  public Date()  {
    month = "January";
    day = 1;
    year = 1000;
  }
```

```
// Date() - A constructor that accepts the
//          month in numeric format
public Date(int monthInt, int day, int year)  {
  setDate(monthInt, day, year);
}

// Date() - A constructor that accepts the
//          month in string format
public Date(String monthString, int day, int
year)  {
  setDate(monthString, day, year);
}

//Date() - A constructor that assumes that the
//         day is the first day of the
//         specified  year
public Date(int year)  {
  setDate(1, 1, year);
}
```

```
// Date() - An copy constructor
public Date (Date aDate)  {
  if (aDate == null)  {
    // Not a real date
    System.out.println("Fatal error.");
    System.exit(0);
  }
  month = aDate.month;
  day = aDate.day;
  year = aDate.year;
}
```

```
// dateOK() - Returns true if the date is valid
//            Returns false if the date is
//            invalid
//            Assumes that the month is in
//            numeric format
public boolean dateOK(int monthInt,
                      int dayInt, int yearInt)  {
  return( (monthInt >= 1) && (monthInt <= 12)
     &&  (dayInt >= 1) && (dayInt <= 31)
   && (yearInt >= 1000) && (yearInt <= 9999));
}
```

```
// dateOK() - Returns true if the date is valid
//            Returns false if the date is
//            invalid
//            Assumes that the month is in
//            string format
public boolean dateOK(String monthString,
                      int dayInt, int yearInt)  {
  return( monthOK(monthString)
    &&  (dayInt >= 1) && (dayInt <= 31)
    && (yearInt >= 1000) && (yearInt <= 9999));
}
```

```
//monthOK() - Returns true if the month is one
//            of the twelve valid strings
//            Returns false other if not
public boolean monthOK(String month)  {
  return (month.equals("January")
    || month.equals("February")
    || month.equals("March")
    || month.equals("April")
    || month.equals("May")
    || month.equals("June")
    || month.equals("July")
    || month.equals("August")
    || month.equals("September")
    || month.equals("October")
    || month.equals("November")
    || month.equals("December"));
}
```

```
// monthString() - Returns the string for the month
//                 specified in numeric format
public String monthString(int monthNumber)  {
  switch(monthNumber)  {
    case 1: return "January";
    case 2: return "February";
    case 3: return "March";
    case 4: return "April";
    case 5: return "May";
    case 6: return "June";
    case 7: return "July";
    case 8: return "August";
    case 9: return "September";
    case 10: return "October";
    case 11: return "November";
    case 12: return "December";
    default: System.out.println
                ("Fatal error - not a valid month");
      System.exit(0);
      return "Error";
  }
}
```

```
// toString() - Converts the date into a string
public String toString()  {
  return (month + " " + day + ", " + year);
}
```

```
// equals() - Returns true if the parameter and
//            the object are the same data
//            Returns false otherwise
public boolean equals(Date otherDate)  {
  return( (month.equals(otherDate.month)) &&
(day == otherDate.day)
    && (year == otherDate.year));
}
```

```
// precedes() - Returns true if the object is
//              an earlier date than the
//              parameter
//              Returns false otherwise
public boolean precedes(Date otherDate)  {
  return((year < otherDate.year) ||
      (year == otherDate.year
         && getMonth() < otherDate.getMonth())
    || (year == otherDate.year
         && month.equals(otherDate.month)
         && day < otherDate.day));
}

// writeOutput() - Write the date in a proper
//                 format
public void writeOutput()  {
  System.out.println(month + " " + day
                          + ", "  + year);
}
```

```
// readInput() - Read a date from the keyboard
public void readInput() {
  boolean tryAgain = true;
  Scanner keyb = new Scanner(System.in);

  // Keep asking until the user enters a valid
  // date
  while (tryAgain)  {
    System.out.println
               ("Enter month, day and year.");
    System.out.println("Do not use a comma.");
    String monthInput = keyb.next();
    int dayInput = keyb.nextInt();
    int yearInput = keyb.nextInt();
```

```
    if (dateOK
          (monthInput, dayInput, yearInput))  {
      setDate(monthInput, dayInput, yearInput);
      tryAgain = false;
    }
    else
      System.out.println
            ("Illegal date.  Reenter input.");
  }
}

// getDay() - Accessor for day
public int getDay()  {
  return day;
}
// getYear() - Accessor for year
public int getYear()  {
  return year;
}
```

```
// getMonth() - Returns the month as a number
public int getMonth()  {
  if (month.equalsIgnoreCase("January"))  return 1;
  if (month.equalsIgnoreCase("February")) return 2;
  if (month.equalsIgnoreCase("March"))  return 3;
  if (month.equalsIgnoreCase("April")) return 4;
  if (month.equalsIgnoreCase("May")) return 5;
  if (month.equalsIgnoreCase("June"))  return 6;
  if (month.equalsIgnoreCase("July"))  return 7;
  if (month.equalsIgnoreCase("August")) return 8;
  if (month.equalsIgnoreCase("September")) return 9;
  if (month.equalsIgnoreCase("October")) return 10;
  if (month.equalsIgnoreCase("November")) return 11;
  if (month.equalsIgnoreCase("December")) return 12;
  else {
    System.out.println
            ("Fatal error - not a valid month");
    //Since it isn't valid, terminate the program
    System.exit(0);
    // The compilers insists on this return
    return 0;
  }
}
```

```
// setDate() - A mutator that expects the month
//             in numeric format
public void setDate(int month,
                    int day, int year)  {
  if (dateOK(month, day, year)) {
    this.month = monthString(month);
    this.day = day;
    this.year = year;
  }
  else {
    System.out.println
               ("Fatal error - invalid date");
    System.exit(0);
  }
}
```

```
// setDate() - A mutator that expects the month
//             in string format
public void setDate(String monthString,
                    int day, int year)  {
  if (dateOK(monthString, day, year))  {
    this.month = monthString;
    this.day = day;
    this.year = year;
  }
  else  {
    System.out.println("Fatal error");
    System.exit(0);
  }
}
```

```
//setDate() - A mutator that assumes that the
//            day is the first day of the
//            specified year
public void setDate(int year)  {
  setDate(1, 1, year);
}

// setMonth() - A mutator for month that checks
//              whether it is a valid value for
//              month
public void setMonth(int monthNumber)  {
  if (monthNumber <= 0 && monthNumber > 12)  {
    System.out.println
            ("Fatal error - Invalid month");
    System.exit(0);
  }
  else
    month = monthString(monthNumber);
}
```

```
  // setDay() - A mutator for day that checks
  //            whether it is a valid value for
  //            day
  public void setDay(int day)  {
    if ((day <= 0) && (day > 31))  {
      System.out.println
            ("Fatal error - Invalid day");
      System.exit(0);
    }
    else
      this.day = day;
  }
```

```
  // setYear() - A mutator for year that checks
  //             whether it is a valid value for
  year
  public void setYear(int year)  {
    if ((year < 1000) && (year > 9999))  {
      System.out.println("Fatal error - Invalid
  year");
      System.exit(0);
    }
    else
      this.year = year;
  }
}
```