Objects and Classes: a little bit of the big picture

At the simplest level, a Java program consists of a collection of classes ... and, classes describe objects. So, even before attempting the traditional "Hello, World" program, let's take a quick look at both objects and classes.

Objects

An object is a "thing," a "gizmo," a "gadget," ... an object.

For example, a car, a soda machine, a dog, a person, a house, a bank account, a pair of dice, a deck of cards, a point in the plane, a TV, a VCR, an ATM machine, an elevator, a square, a circle, a flea, an elephant, a camera, a movie star, a computer mouse, a live mouse, a phone, an airplane, a song, ... just about anything is an object. In computing, a window is an object, so is a mouse, a menu, a textbox, and a button.

Objects come in all shapes, sizes, and colors. An object may be physical, like a radio, or intangible, like a song. For our purposes, however, objects are entities that have

1. attributes, (characteristics or properties), and
2. methods, (actions or behaviors of an object).

Example 1: An elevator is an object.

The attributes? Perhaps:
1. the current floor
2. whether or not the door is open or closed

The methods (actions/behaviors) might be:
1. give (display) the current floor
2. open the door
3. close the door
4. change the current floor
5. ring the alarm

Three elevator objects

- Elevator 1: floor: 3, open
- Elevator 2: floor: 5, open
- Elevator 3: floor: 2, closed
Notice that the three elevator objects have different attribute values.

The attribute values determine the state of an object.

Thus the state of elevator 1 is that the floor is 3 and the door is open. The state of elevator 3 is that the current floor is 2 and the door is closed.

Each elevator object has a unique state. On the other hand, all elevator objects have the same behavior. However, all three objects can do the same things (open the door, close the door etc.). All three have the same methods.

Example 2: A rectangle is an object
Some possible properties or attributes of a rectangle are:
1. length and
2. width
Some possible methods are:
1. get the length
2. get the width
3. change the length
4. change the width
5. get the area
6. get the perimeter

Here are two rectangle objects:

Two rectangle objects

The state of the first rectangle object is {length = 7, width = 5} ; the state of the second rectangle is {length = 2, width = 8}. 
Example 3: A list of numbers is an object
   Possible attributes include:
   1. the actual numbers on the list
   2. the size (count) of the list
   Possible methods/actions are:
   1. get the largest number
   2. get the smallest number
   3. add a number to the list
   4. remove a number from the list
   5. sort the list
   6. get the average of the numbers on the list etc.

Two list of numbers objects

Example 4: A computer window is an object
   Some of the (many) attributes include:
   1. length
   2. width
   3. background color
   4. font style
   5. font color
   6. state – maximized, minimized or downscaled
   Some of the (many) methods include:
   1. resize the window (change length and width)
   2. maximize
   3. minimize
   4. change background color
   5. change font etc.

In the context of a computer program, you might think of an object as a representation, model or abstraction of some entity consisting of
   1. data (attributes) and
   2. functions (methods) which use or manipulate the data

An object’s data determines its state. For example, the data for the first elevator (above) specifies that the “floor” attribute has value 3 and the door is open. The current state of the first rectangle indicates that the length of the rectangle is 7 and the width is 5.

The methods/functions specify what an object does, i.e., the behavior of an object.
The attributes and methods of an object depend on our specific use and view of an object. For example, in one application, a rectangle object might be a simple geometrical figure with just two attributes, length and width. Another, perhaps graphical, view of a rectangle might include color and location (x and y coordinates) among the attributes. Similarly, an elevator has many potential attributes (carpet color, number of passengers, maximum weight, Muzak playing, date of last inspection) but only a few attributes are of interest in any application.

Some attributes themselves might be other objects. For example, a light object may have:
- **attributes:**
  - number of watts
  - current state (on or off)
- **methods:**
  - turn light on
  - turn light off

Now a light object may be part of (an attribute of) an elevator object.

In each of the following examples, the attributes and methods have been chosen arbitrarily. Indeed, choosing the “right” attributes and methods for an object is a skill and an art that comes with practice and patience.

**Examples:**

1. **A circle** is an object.
   - The sole attribute (data) of a circle might be its radius, a real number.
   - The methods (functions) might be
     - a. give (return) its area
     - b. give (return) its circumference.

2. **A Bank account** is an object.
   - The data or attributes of a bank account might be
     - a. an ID number
     - b. customer’s name, address, and balance.
   - The methods/functions/operations might be
     - a. give (return) the balance
     - b. give personal information about the account owner
     - c. make a deposit
     - d. make a withdrawal

3. **A deck of cards** is an object.
   - The data consists of the fifty-two cards of the deck.
   - The functions or operations
     - a. shuffle the deck
     - b. deal a card

4. **A pair of dice** is an object.
   - The data/properties are the values on the two dice.
   - The single operation/function/method is “roll the dice.”

**Classes**

A class is a template, blueprint, or description of a group of objects. Every object is described by some class. For example, an elevator class specifies the characteristics and behaviors of all elevator objects. The elevator class is a general *description* of an elevator. An elevator class is *not* an elevator. A rectangle class describes the attributes and methods of all rectangle objects. A rectangle class may specify that every rectangle object has both a length and a width. However, a rectangle class is not a rectangle.
An architect’s blueprint is analogous to a class. A blueprint is not a house but a description or specification of a potential house. When a builder constructs two real houses from a blueprint, well, now we have two “house objects.” The skill of the programmer in defining classes is akin to the skill of the architect, and the labor of the compiler in building objects is like the work of the construction company.

Just as a builder creates houses from a blueprint, a program creates objects from a class. From one blueprint, a builder can build many houses. From one class, a program can create many objects. Every object is “manufactured” according to some class specification. Every object belongs to some class. An object is an instance of a class.

Classes in Java

Below is an example of a Rectangle class in Java. The class is a description (in Java) of the attributes and behaviors of a rectangle object. For now, don’t be concerned with the syntax or any of the Java particulars.

Example:

```java
public class Rectangle {
    //Every Rectangle object has both length and width attributes (int)
    private int length;
    private int width;

    //default values for a rectangle object are length = 1 and width = 1
    public Rectangle() // default constructor
    {
        length = 1;
        width = 1;
    }

    //can create a Rectangle object with any dimensions
    public Rectangle(int x,int y) //constructor
    {
        length = x;
        width = y;
    }

    //can change the dimensions of any rectangle object
    public void changeDimensions(int x,int y) // mutator
    {
        length = x;
        width = y;
    }

    //gives the area of a rectangle object // accessor
    public int getArea()
    {
        return length*width;
    }

    //gives the perimeter of a rectangle object
    public int getPerimeter()
    {
        return 2*(length+width);
    }
}
```
Java and Object Oriented Programming
Day 1 - Java Fundamentals

The preceding code (a Rectangle class) is a template for a rectangle. According to the specifications, any potential rectangle object has both a length and a width of type int. Moreover any rectangle object can

- change its dimensions,
- give its area, and
- give its perimeter.

OK, we know what a rectangle object has and what it can do. So, we can now manufacture, create, instantiate as many rectangles as we like. We have the blueprint, so let's start production:

```java
Rectangle r1 = new Rectangle(5,7); // makes a 5 X 7 rectangle named r1
Rectangle r2 = new Rectangle(7,5); // makes a 7 X 5 rectangle named r2
Rectangle r3 = new Rectangle(); // makes a default 1 X 1 rectangle named r3
```

Voilà. With three magic statements, we have created three rectangle objects --- built according to the specifications of our class/blueprint. Each rectangle object has a length property and a width property with appropriate values:
Messages

In a Java program, objects interact with other objects by sending messages. Messages are similar to function calls in procedural style programming. The following three statements send messages to r1, r2 and r3, respectively:

```java
System.out.println(r1.getPerimeter());
area = r2.getArea();
r3.changeDimensions(7,3);
```

The purpose of the messages should be pretty obvious:
- "r1, get your perimeter!"
- "r2, get your area!"
- "r3, change your dimensions!"

Example:

The following class describes Dog objects. (Some syntax has been simplified.) From our perspective a Dog object is a very simple creature:

```java
public class Dog {
    private String bark; //A Dog object has but a single attribute—its bark
    public void setBark(String s) //set the sound: "woof-woof," "bow-wow" etc.
    {
        bark = s;
    }
    public void speak() //Every dog can bark
    {
        print(bark);
    }
}
```

The Dog class has but a single attribute:
- bark.
A Dog object can do only two things:
- set its bark and
- speak, i.e., bark

The dog class provides a description of a dog but, nonetheless, no dogs exist yet.
We now create (instantiate) a few Dog objects:

```java
Dog fido = new Dog(); // create a Dog named fido
Dog brutus = new Dog(); // create a Dog named brutus
```

... and, send a few messages to the critters:

```java
fido.setBark("Bow-Wow"); // a message to fido
brutus.setBark("Woof-Woof"); // a message to brutus
fido.speak(); // fido, speak, boy!!
brutus.speak(); // you too, brutus, speak
```

You might wonder, who or what sends these mysterious messages?

In a Java program, objects send other objects messages.

For example:

- a customer object might send an ATM object a message: “deposit this money.”
- a gambler object might send a slot machine object a message: “take my money, please.”
- the slot machine object might send back a message to the gambler object (“You lose.”).

But, now we are getting ahead of ourselves. For the present, you should have a very general, albeit sketchy, understanding of objects and classes. During the next two weeks, you will learn how to design your own classes. You will learn how to write programs by building objects that interact and send messages to each other. What does a simple but non-trivial program look like using the object-oriented paradigm? It’s coming, but first …
Simple Basic Java

We begin with a quick tour of some very basic Java constructions – I/O, selection, iteration, and arrays. What follows is a selection of simple, though boring, programs which will illustrate many of the nitty-gritty details that crop up when you design your own classes. Rather than make numerous side trips in the next chapters, we discuss these constructions first.

These simple examples in no way illustrate the object-oriented paradigm. Because the following sample programs are so simple, you may get the impression that Java is just a dialect of C++. Don’t be fooled. The similarities are superficial. Although each of the following programs consists of a single class with only one method, virtually all Java programs consist of many classes.

As tradition dictates, we begin with a plain vanilla “hello world” style program.

Example 1:

1. // our first program!!!!
2. public class Prog1
3. {
4.     public static void main(String args[])
5.     {
6.         System.out.println("Dopey");
7.         System.out.println("Grumpy");
8.     }
9. }

Let’s look at the program line by line:

- Line 1 is a comment. As in C++, comments begin with the symbol // and continue to the end of the line. Multiple line comments are enclosed by the symbols /* and */.

- The word class (Line 2) indicates that we are defining a new class. The name of the class is Prog1. (A class is a blueprint for objects, so, in theory, we could create a Prog1 object from class Prog1).

- The keyword public is an access modifier. A public class is accessible anywhere in a program. We will discuss this in greater detail later. C++ does not have access modifiers for classes.

- Line 4 is the heading of the method called main. Every Java application must have one method called main which is defined exactly as it is on line 4. The main method is always the first method that is executed. When the program executes, main goes first. This particular class is very simple and has but a single method, main(). Of course most classes will have many methods.

- The word public on line 4 is an access modifier for the main method. If a method is declared public then that method is accessible anywhere within the program i.e. anywhere outside the class where it is defined. The private access modifier specifies that the method is accessible only within the class. If main were not public, then main could not be called from “outside” the class (i.e. by the system) and the program could not be executed.

- The keyword static signifies that a method is a class method. A static method is one that can be invoked whether or not an instance of the class is created. Outside of the defining class, non-static methods are invoked via objects (r1.getArea()); and static
methods are invoked using the class name, (Math.sqrt(64.0));, where Math is a class. Static methods are available whether or not any objects are created. A static method belongs to the class, not to any particular instance of the class (object). In a way, static methods really go against the grain of object-oriented programming, and they are very much like the functions of C, and procedures of Pascal. Now, suppose that main were not a static method, what would be the consequences?

- The keyword `void` is used as in C++, to specify that method main returns no value.

- `String args[]` is a parameter to main. One of the next examples illustrates how to use such a parameter. The idea is simple and most easily explained via an illustration.

- The curly braces denote the start and end of a block and are used as in C++.

- The method main has two similar lines

  ```java
  System.out.println("Dopey");
  System.out.println("Grumpy");
  ```

  System.out is a pre-defined system object associated with the standard output stream (usually the screen). So, in each of these statements, the System.out object invokes its println method. The println method accepts a string, s, as a parameter (strings are enclosed in double quotes) and sends s, followed by a carriage return, to the standard output device (the screen). “Dopey” and “Grumpy” are string literals.

Notice the class Prog1 is sending a message to System.out. The message is println("Dopey"). “

Output:

```
Dopey
Grumpy
```

Here are a few other points to keep in mind:

- Java, like C++, is case sensitive

- The class Prog1 must be saved in a file called Prog1.java. This is not the case with C++. C++ is not so picky about file names.

- Java programs often contain many classes. Each class is usually saved in a separate file (although this is not necessary). The file name must be the same as the name of the class and have a .java extension. You may save only one public class per file.

- Java convention dictates that class names begin with an uppercase letter. All other names begin with a lowercase letter. Uppercase letters are used to separate “words” within a name: getArea(), getCircleRadius() etc.
Example 2:

```java
public class Prog2 //saved as Prog2.java
{
    public static void main (String args[])
    {
        System.out.print("Dopey"); // print not println
        System.out.print("Grumpy");
    }
}
```

Notice that the `print` method is used instead of `println`. In this case, no carriage return is appended to the output string.

Output:

```
DopeyGrumpy
```

Example 3:

```java
public class Prog3
{
    public static void main (String args[])
    {
        System.out.print("Dopey"+ "Grumpy");
    }
}
```

Here, the `+` symbol functions as the string concatenation operator.

Output:

```
Dopey  Grumpy
```
Java and Object Oriented Programming  
Day 1 - Java Fundamentals

Example 4:
public class Prog4  //saved as prog4.java
{
    public static void main (String args[])
    {
        System.out.println(args[0]);
        System.out.println(args[1]);
    }
}

If you run this program with the command

> java Prog4 Dopey Grumpy

the two strings entered at the command line, “Dopey” and “Grumpy,” are stored in the array args. Consequently, args[0] holds the string “Dopey” and args[1] holds “Grumpy.” Notice that arrays are indexed from 0, as they are in C++.

Output:
Dopey
Grumpy

Example 5:
public class Prog5  //saved as prog5.java
{
    public static void main (String args[])
    {
        System.out.println("Dopey" + 2+3+4);  
        System.out.println(2+3+4+"Dopey");  
        System.out.println("Dopey" + ( 2+3+4 ) );  
        System.out.println(2+3+4);  
        System.out.println( ( 2+3+4) );
    }
}

Output:
Dopey234
9Dopey
Dopey9
9
9

Here, you should notice that:

• Addition, as usual, is performed left to right.
• The argument to println is always a String.
• If the argument to println is x+y and either x or y is a string, then + effects string concatenation. For example, consider the method call println("Dopey" + 2). The integer 2 is converted to a string and the concatenated string “Dopey2” is passed to println. However, in the method call println(2+3+"Dopey"), the first plus represents addition, the second effects concatenation. So, the string “5Dopey” is passed to the println method.
Primitive data types in Java:

<table>
<thead>
<tr>
<th>Type</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>byte</td>
<td>-128 ... 127</td>
</tr>
<tr>
<td>short</td>
<td>-32,768 ... 32,767 (2 bytes)</td>
</tr>
<tr>
<td>int</td>
<td>-2,147,483,648 ... 2,147,483,647 (4 bytes)</td>
</tr>
<tr>
<td>long</td>
<td>-9,223,372,036,854,775,808 ... 9,223,372,036,854,775,807 (8 bytes)</td>
</tr>
<tr>
<td>float</td>
<td>4 bytes, seven significant digits</td>
</tr>
<tr>
<td>double</td>
<td>8 bytes, 15 significant digits</td>
</tr>
<tr>
<td>boolean</td>
<td>false...true (not treated as integers as in C++)</td>
</tr>
<tr>
<td>char</td>
<td>any unicode character - two bytes (ASCII uses one byte per character)</td>
</tr>
</tbody>
</table>

(Uncode is an extension of ASCII, so that ASCII and Unencode values coincide)

Notes:

- In C++ the range of a type is machine dependent. This is not the case in Java.
- Primitive variables are declared and initialized as in C++.
  
  ```
  int x = 5, y = 7;
  boolean z = true;
  ```
- Type casting is permitted. Note the syntax.
  
  ```
  double x = 12.23;
  int i = (int) x;
  ```
- In C++ you can assign a double value to an int variable:
  
  ```
  int pi = 3.14159;
  ```

  Of course, such an assignment will result in a truncation and loss of information. So the C++ variable `pi` has the value 3. In Java, if a type coercion results in a loss of data, you must always cast the data element to the new type.

**Example:**

```cpp
C++:    int x;
double y = 3.14;
x = y; // x is now 3 and the decimal has been lost
```

```java
Java:  int x;
double y = 3.14;
x = (int)y; // use a typecast, x is 3
```

Also, notice that following Java assignment:

```java
float y = 3.14;
```

will result in an error because the constant 3.14 is considered of type `double` and `double` cannot be assigned to a `float` without an explicit cast. Such an assignment might result in a loss of accuracy. It is probably a good practice to stick with type `double` whenever you use real numbers.
Operators and Operator Precedence

Java operators are much the same as in C++. Operator precedence is according to the following chart.

Java Operator Precedence (highest to lowest)

<table>
<thead>
<tr>
<th>Operator</th>
<th>Associativity</th>
</tr>
</thead>
<tbody>
<tr>
<td>[ ] ( ) . (dot)</td>
<td>left</td>
</tr>
<tr>
<td>++ -- +(unary) -(unary) ! (type) new</td>
<td>right</td>
</tr>
<tr>
<td>* / %</td>
<td>left</td>
</tr>
<tr>
<td>+ -</td>
<td>left</td>
</tr>
<tr>
<td>&lt; &lt;= &gt; &gt;= instanceof</td>
<td>left</td>
</tr>
<tr>
<td>== !=</td>
<td>left</td>
</tr>
<tr>
<td>&amp;&amp;</td>
<td></td>
</tr>
<tr>
<td>? : (conditional)</td>
<td>right</td>
</tr>
<tr>
<td>= (assignment) *= /= += -= %=</td>
<td>right</td>
</tr>
</tbody>
</table>

One or two operators, like `instanceof`, which are not available in C++ will be explained in later chapters.

Data Input

Because Java insists that all methods belong to classes, data input is not simple in Java. For the present we will use a class `MyInput.java`, which you can find in the course disk.

The `MyInput` class contains static (class) methods:

1. `int readInt()`
2. `String readString()`  // We will cover strings a little later
3. `double readDouble()`
4. `float readFloat()`

The purpose of each of these methods should be self-explanatory.

Because the methods of `MyInput` are static, you do not need to create a `MyInput` object to invoke these methods. These methods are class methods. You can invoke them with the class name, e.g., `x = MyInput.readInt();`

Example:

```java
public class IOStuff   //save as IOStuff.java
{
    public static void main(String args[])
    {
        int x, y;
        System.out.println("Enter two numbers");
        x = MyInput.readInt();
        y = MyInput.readInt();
        System.out.println("The sum is "+ (x+y));
    }
}
```

1 The class `MyInput` was adapted from *Introduction to Java Programming* by Y.D.Liang.
When using the methods of MyInput, be aware that each input value must appear on a separate line. Thus, input to the preceding program must be of the form:

```
3
4
```

do not

```
3  4
```

We conclude with a selection of program-ettes that illustrate some Simple Java constructions – selection, iteration, constants etc.

**Example 1:**

```java
class Circle {
    public static void main(String args[]) {
        final double pi = 3.14159;  // final denotes a constant not const as in C++
        double radius;
        double area;
        System.out.print("Enter the radius of a circle ");
        radius = MyInput.readDouble(); // note use of MyInput
        area = pi*radius*radius;
        System.out.println("The area is "+area);
    } // end main()
} // end class circle
```
Example 2:

//computes the average of 4 grades and assigns a letter grade
public class Grades
{
    public static void main( String args[])
    {
        int answer;
        int grade, sum;
        double average;

        do
        {
            sum = 0;
            System.out.println("Enter 4 grades ");
            for (int i = 1; i <= 4; i++)
            {
                grade = MyInput.readInt();
                sum += grade;
            }
            average = sum/4.0;
            System.out.print("Average is "+ average + " ");
            if (average >= 90)
                System.out.println('A');
            else if (average >= 80)
                System.out.println('B');
            else if (average >= 70)
                System.out.println('C');
            else if (average > 60)
                System.out.println('D');
            else
                System.out.println('F');
            System.out.println("Again? type 1 for yes ");
            answer = MyInput.readInt();
        }while (answer ==1); //end do..while
    } //end main()
} //end class Grades
Example 3:

//Reads 4 grades from the command line. Thus, the grades are stored as strings and must be converted to a numeric type before any processing can occur.

public class Grades1
{
    public static void main( String args[])
    {
        char answer;
        int grade, sum = 0;
        double average;
        for (int i = 0; i < 4; i++)
            sum += Integer.parseInt(args[i]) ; // converts from string to int
        average = sum/4.0; // note that an explicit cast is not necessary
        System.out.println("Average is "+ average + " ");
        if (average >= 90)
            System.out.println('A');
        else if (average >= 80)
            System.out.println('B');
        else if (average >= 70)
            System.out.println('C');
        else if (average > 60)
            System.out.println('D');
        else
            System.out.println('F');
    } //end main()
} //end class Grades1

To run the above program:
    > java Grades1 80 90 70 60

Notice that
args[0] is “80”
args[1] is “90”
args[2] is “70”
args[3] is “60”

In order to calculate an average, the strings stored in args must be converted to integers. Java provides a class Integer with a static method, parseInt(String s), that does exactly that -- accepts a string of digits and converts that string to an integer. If the string contains any non-numeric characters, the program will crash. Again, notice that parseInt is a static method so it can be invoked without instantiating an Integer object. We will be looking at the Integer class in more detail later.

Finally, while loops, do-while loops and the switch statement work exactly as in C++.
Arrays

In Java, arrays are objects. This is not the case in C++. Consequently, some array operations differ somewhat from their C++ counterparts:

1. Array declaration:
   ```java
   int[] x;  -- or int x[]; if you like.
   ``
   An array declaration does not allocate memory for the array. This is different than C++. Here, x is a reference (think pointer) and x has an initial value of `null`.

2. To allocate memory for an array use the `new` function.
   ```java
   int[] x;   // x is a reference
   x = new int[25];
   
   or
   int []x = new int[25];
   
   or
   int[] x = {2,4,6,8,1,3,5};
   ``
   Memory for an array is allocated at run time from the heap – the collection of free memory dynamically managed by the operating system. This differs from C++, where memory for arrays is allocated at compile time.

3. The syntax for array assignment is the same as in C++ i.e. `x[i]= 324`;

4. If x is an array, then x.length holds the size of the array. x.length is not a method call. Here x is an array object and length is a `public` variable i.e attribute.

5. Consider the following declarations:
   ```java
   int[] x = {2,3,4,5};
   int [] y;
   y = x;  // not allowed in C++
   y[0] = 100;
   System.out.print(x[0]);
   ``
   What is the output?

6. Java provides a static method in the System class that allows you to copy one array to another:
   ```java
   System.arraycopy(sourcearray , start, targetarray ,  start, count);
   ``
   ```java
   System.arraycopy(x,3,y,7,2);
   Copies x[3],x[4] to y[7],y[8]
   ``

The following examples illustrate some array operations as well as a few array peculiarities.
Example 1:

```java
public class Array1 //Fills an array and outputs its data in reverse order
{
    public static void main(String args[])
    {
        final int max = 20; // max is a constant

        int[] x;
        x = new int[max]; // allocate memory for 20 integers
        int size = 0;
        int data;

        System.out.println("Enter up to \" + max +\" integers. End with -999; \"");

        data = MyInput.readInt();
        while(data != -999)
        {
            x[size] = data;
            size++;
            data = MyInput.readInt();
        }

        System.out.println("The list in reverse is ");
        for (int i = size-1; i >= 0; i- -)
            System.out.println(x[i]);
    }
}
```

Output:

```
Enter up to 20 integers. End with -999;
4
2
8
6
1
-999
The list in reverse is
1
6
8
2
4
Press any key to continue . . .
```
Example 2:

```java
public class Array2   //Demonstrates assignment and manipulation of arrays.
{
    public static void main(String args[])
    {

        int[] x = {1,2,3,4,5}; // array initialization, like C++
        int[] y = new int[5];

        System.out.println("The values stored in x are :");
        for(int i = 0; i < 5; i++)
            System.out.println(x[i]);

        y = x;  //what will happen here?
        //set everything in y to 0
        for (int i = 0; i < 5; i++)
            y[i] = 0;

        //now print the values stored in x
        System.out.println("The values stored in x are now :");
        for(int i = 0; i < 5; i++)
            System.out.println(x[i]);
    }
}
```

Output:

```
The values stored in x are :
1
2
3
4
5
The values stored in x are now :
0
0
0
0
0
```
Example 3:

```java
public class Array3   //Demonstrates use of System.arraycopy()
{
    public static void main(String args[])
    {
        int[] x= {1,2,3,4,5};
        int[] y = new int[5];

        System.out.println("The values stored in x are :");
        for(int i = 0; i <5; i++)
            System.out.println(x[i]);

        System.arraycopy(x,0,y,0,5);
        //set everything in y to 0
        for (int i = 0; i <5; i++)
            y[i] = 0;

        System.out.println("The values stored in x are now :");
        for(int i = 0; i <5; i++)
            System.out.println(x[i]);

        System.out.println("The values stored in y are :");
        for(int i = 0; i <5; i++)
            System.out.println(y[i]);
    }
}
```

The values stored in x are:
1
2
3
4
5
The values stored in x are now:
1
2
3
4
5
The values stored in y are:
0
0
0
0
0
Example 4:

```java
public class Array4    //Demonstrates use of System.arraycopy()
{
    public static void main(String args[])
    {
        int[] x = {1,2,3,4,5,6,7,8,9};
        int[] y = new int[10];

        System.out.println("Here is y initially");
        for(int i = 0; i < y.length; i++)
            System.out.println(y[i]);

        //copy x[3], x[4], x[5], x[6] to y[5],y[6],y[7],y[8]
        System.arraycopy(x,3,y,5,4);

        System.out.println("Here is y after the copy");
        for(int i = 0; i < y.length; i++)
            System.out.println(y[i]);
    }
}
```

Here is y initially

```
0
0
0
0
0
4
5
6
7
0
```

Here is y after the copy

```
0
0
0
0
4
5
6
7
0
```
Methods

Methods in Java are a little like functions in C++, however every Java method is defined within a class. In this respect, Java differs radically from C++ which allows independent or “stand-alone” methods. In C++, methods may be designated as private or public. A public method may be used by any other class, while a private method may be used only within its defining class.

As you have seen, a method may also be qualified as static.

- A static method is a class method (as opposed to an instance (object) method).
- A static method is associated with a class and not with any particular object.
- You can use a static method even when no objects of the class exist.
- To invoke a public static method outside its defining class, use dot notation with the class name (Classname.myMethod()).
Example:
The syntax and semantics of the following methods (max and min) should cause no difficulty.

```java
public class Compare
{
    public static int max(int a, int b)
    {
        if (a > b)
            return a;
        else
            return b;
    }

    public static double max(double a, double b)
    {
        if (a > b)
            return a;
        else
            return b;
    }

    public static int min(int a, int b)
    {
        if (a < b)
            return a;
        else
            return b;
    }

    public static double min(double a, double b)
    {
        if (a < b)
            return a;
        else
            return b;
    }
}
```

Here, we have a class Compare with four public static methods, two (overloaded) `max` methods and two (overloaded) `min` methods. (Java’s rules for method overloading are the same as in C++.) Since these four methods are public, any class can use these methods. Because the methods are static, no Compare object need be created.

```java
public class Demo
{
    public static void main(String args[])
    {
        int first, second;
        int big, small;
        System.out.println("Enter two numbers");
        first = MyInput.readInt();
        second = MyInput.readInt();
        big = Compare.max(first, second);
        small = Compare.min(first, second);
        System.out.println("Big: "+ big + " Small: "+ small);
    }
}
```
Parameter Passing

Finally, it is important to note that,

**in Java, parameters are passed by value.**

In this respect, Java differs from C++ which supports both value and reference parameters. Consequently, no Java method can permanently change the value of any of its parameters. Of course, if an address is passed then the value stored at the referenced location can be changed.

For example, like C++, an array parameter is a reference i.e. an address or a pointer. Passing an array to a method means passing an address. Although a method cannot alter the address of an array, a method can change the **elements** of the array, as the next example illustrates.

**Example:**

```java
public class ArrayDemo
{
    public static void readArray(int [] x, int n)
    {
        for(int i = 0; i < n; i++)
            x[i] = MyInput.readInt();
    }
    public static void reverse(int [] x, int n)
    {
        for(int i = n-1; i >=0; i--)
            System.out.println(x[i]);
    }

    public static void main(String args[])
    {
        int list[];
        int size;
        System.out.println("How many elements in the list? ");
        size = MyInput.readInt();
        list = new int[size];
        System.out.println("Enter "+ size+ " integers");
        readArray(list, size);
        System.out.println("The list in reverse: ");
        reverse(list, size);
    }
}
```
Output:
How many elements in the list? 4
Enter 4 integers
4
3
2
1
The list in reverse:
1
2
3
4

Java Libraries and Packages

Java provides a rather large library of hundreds of pre-defined classes that can be used in any Java program. Moreover, related classes are organized or grouped into packages. For example:

- The Abstract Window Toolkit package (java.awt) contains dozens of classes that are useful for graphics programming.
- The java.text package contains classes that simplify text formatting.
- The java.util package contains, among other classes, a random number class, a binary tree class and a linked list class.

Example:
The Random class is a member of the java.util package. Among class Random's methods are several different random number generators. The fully qualified name of this class, Random, is

```
java.util.Random            //   packageName.className
```

To utilize the Random class (and its methods) in a program, use an import statement in one of two forms:

1. `import java.util.Random;` // imports class Random from java.util
   
   ```java
   public class MyClass
   {
       // can use the methods of Random
   }
   ```

2. `import java.util.*;` // import the entire java.util package:

   ```java
   public class MyClass
   {
       // can use the methods of Random or any other classes in java.util
   }
   ```

That's all there is to it. Now everything Random has to offer is available to MyClass. We will see Random again a little later in the course.
The java.lang package

The java.lang package is automatically imported into every program so an import statement is both redundant and unnecessary.

The java.lang package includes many useful classes including the Math class. The class, Math, provides an array of very handy (static) methods:

- Absolute Value:
  - int abs(int x)
  - double abs (double x)

- Trig functions (Angles are measured in radians)
  - double cos(double a)
  - double sin(double a)
  - double tan(double a)
  - double acos(double a)
  - double asin(double a)
  - double atan(double a)

- A random number generator.
  - double random() // returns a double, x, such that 0.0 < x < 1.0:

- Power and square root functions:
  - double pow(double base, double power)
  - double sqrt(double x)

- Ceiling and floor functions:
  - double ceil (double x)
    // returns the smallest whole number greater than or equal to x
  - double floor (double x)
    // returns the greatest whole number less than or equal to x

- Logarithmic and exponential functions
  - double log(double x) // returns the natural logarithm
  - double exp(double x) // returns e^x

Since the methods of the Math class are static, they are called using the class name Math. For example:

Math.sqrt(56.54);
Math.pow(1.5, 10);
Math.random();

Conclusion

The examples of this section are simple and illustrate some of the syntactical features of Java. None could be classified as object-oriented. All the sample programs use static methods and created no objects. In the next two days, we will look more closely at object-oriented programming and the foundational concepts of encapsulation, inheritance, and polymorphism.