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.
Averages.java

import java.util.Scanner;

public class Averages {
    public static void main(String[] args) {
        int a = readInt(),
        b = readInt(),
        c = readInt(),
        avgInt;

        double d = readDouble(),
        e = readDouble(),
        f = readDouble(),
        avgDouble;

        avgInt = average3(a, b, c);
        avgDouble = average3(d, e, f);
        System.out.println("The average is " + avgInt);
        System.out.println("The average is " + avgDouble);
    }

    public static int readInt() {
        Scanner keyb = new Scanner(System.in);
        System.out.println("Enter an integer value?");
        int inputValue = keyb.nextInt();
        return inputValue;
    }

    public static double readDouble() {
        Scanner keyb = new Scanner(System.in);
        System.out.println("Enter a double value?");
        double inputValue = keyb.nextDouble();
        return inputValue;
    }
}
public static int average3(int i, int j, int k) {
    int sum = i + j + k;
    return sum/3;
}

public static double average3(double x, double y, double z) {
    double sum = x + y + z;
    return sum/3.0;
}

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:

  ```java
  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.
### TestFile.java – An Example

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

public class TestFile {
    public static void main(String[] args)
        throws FileNotFoundException {
        Scanner myFile = new Scanner(new File("C:\Program Files\Java\jdk1.5.0_03\MyFile.dat")
        );
        String myString = new String();
        myString = myFile.nextLine();
        System.out.println(myString);
    }
}
```

### The Basic Data Types

Java have eight data types:

- Four integer types, which are **byte**, **short**, **int**, and **long**
- Two floating point types, which **float** and **double**
- **char**, which are individual characters.
- **boolean**, which are **true** or **false**
Integers

<table>
<thead>
<tr>
<th>Type</th>
<th>Storage</th>
<th>Min. Value</th>
<th>Max. Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>byte</td>
<td>8 bits</td>
<td>-128</td>
<td>127</td>
</tr>
<tr>
<td>short</td>
<td>16 bits</td>
<td>-32,768</td>
<td>32,767</td>
</tr>
<tr>
<td>int</td>
<td>32 bits</td>
<td>-2,147,483,648</td>
<td>2,147,483,647</td>
</tr>
<tr>
<td>long</td>
<td>64 bits</td>
<td>-9,223,372,036,854,775,808</td>
<td>9,223,372,036,854,775,807</td>
</tr>
</tbody>
</table>

Floats and Doubles

- Floating point values can have fractional parts and exponents.
- Floating point values contains of a mantissa and exponent.
  - If the number is $6.5 \times 10^3$, we rewrite it as $0.65 \times 10^4$; $0.65$ is the mantissa and $4$ is the exponent
- float and double differ in how many places of precision they involve
Characters

- Characters are individual characters.
- Characters are represented in Java by Unicode, a 16-bit character representation code that has become the international standard for representing characters.

Wrapper Classes

- Every data type in Java has a wrapper class, i.e., a class that contains (or wraps around) the data items together with methods for manipulating data of this type.
- We have seen some examples of this with the class Character, which is the wrapper class for char and contains the methods, `isLetter()`, `isDigit()`, `toUpperCase()` and so on.
Examples of Wrapper Classes

- The wrapper classes include:
  - `Character` - a wrapper class for `char`
  - `Integer` - a wrapper class for `int`
  - `Double` - a wrapper class for `double`
  - `Boolean` - a wrapper class for `boolean`

The `Object` Class

- All classes in Java are derived from the superclass `Object`.
- Any method that expect an object of class `Object` as a parameter will accept an object of `any` class.
- This allows users to pass any type of variable or object to a particular method.
Example: Using The **Object** Class

```java
public class ObjectTest {
    // ObjectTest - a basic class to demonstrate
    // using objects to pass
    // parameters
    public static void main(String args[]) {
        int i = 9;
        char c = 'a';

        // oi and oc are wrapper classes for i and c
        Integer oi = new Integer(i);
        Character oc = new Character(c);
        myFunc(oi);
        myFunc(oc);
    }
}
```

```java
// myFunc() - can accepts either Integers
// or Characters
public static void myFunc(Object x) {
    System.out.println("I\'m in!!");
}
```

The Comparable Class

- There are a group of classes in Java that are comparable, i.e., there is a set order for any and all objects of this class.
- Comparable classes are expected to have methods called `compareTo()` and `equals()`,
- All the wrapper classes are comparable.

Using `compareTo()`

```java
public static void main(String args[]) {
    int i = 9, j = 7, k, l, m;
    Integer oi = new Integer(i),
        oj = new Integer(j);

    k = oi.compareTo(oj);
    l = oj.compareTo(oi);
    m = oi.compareTo(oi);
    System.out.println(k + " " + l + " " + m);
}

Output
1 -1 0```
**toString()**

- **toString()** is a method common to the wrapper classes that returns a character string representing the value in printable form:

- The wrapper classes have two such methods: one belonging to the object that requires no parameter and a static method that requires a parameter.

```java
public class ObjectTest {
    public static void main(String args[]) {
        int i = 9;
        double x = 4.5;
        char c = 'a';
        boolean p = true;

        System.out.println(Integer.toString(i));
        System.out.println(Double.toString(x));
        System.out.println(Character.toString(c));
        System.out.println(Boolean.toString(p));
    }
}
```
```java
public class ObjectTest {
    public static void main(String args[]) {
        Integer i = new Integer(9);
        Double x = new Double(4.5);
        Character c = new Character('a');
        Boolean p = new Boolean(true);

        System.out.println(i.toString());
        System.out.println(x.toString());
        System.out.println(c.toString());
        System.out.println(p.toString());
    }
}
```

**parseNumericType()**

- There are methods within the numeric data type’s wrapper classes that allow the programmer to parse a numeric value from the beginning of a string.
- This allows us to read string input and to extract a numeric value from our input.
- We will concern ourselves only with parseInt and parseDouble at this time.
An Example of `parseInt()` and `parseDouble()`

```java
import java.util.Scanner;
public class PrintString {
    public static void main(String[] args) {
        Scanner keyb = new Scanner(System.in);
        String inString = new String();
        int inputInteger;
        double inputDouble;

        System.out.print("Enter an integer\t?");
        inString = keyb.nextLine();
        inputInteger = Integer.parseInt(inString);
        System.out.print("Enter a double value\t?");
        inString = keyb.nextLine();
        inputDouble = Double.parseDouble(inString);

        System.out.println("The values are: ");
        System.out.println(inputInteger);
        System.out.println(inputDouble);
    }
}
```
Converting From String

• Several wrapper classes include a static method called `valueOf(s)` which converts `s`, a string, into a wrapper class.

• Examples
  - `Integer.valueOf(s)` returns an Integer.
  - `Boolean.valueOf(s)` returns a Boolean.
  - `Double.valueOf(s)` returns a Double.

Converting From String: An Example

```java
public static void main(String[] args) {
    Integer i = new Integer(0);
    Boolean p = new Boolean(false);
    Double x = new Double(5.50);

    i = Integer.valueOf("45");
    p = Boolean.valueOf("true");
    x = Double.valueOf("3.14159");

    System.out.println("The values are:");
    System.out.println(i);
    System.out.println(p);
    System.out.println(x);
}
```
Converting From Integer

• Integer objects can be converted to other types using the following methods:
  – `doubleValue()` – converts the Integer object into a double value
  – `intValue()` – converts the Integer object into an integer value

Converting From Double

• Double objects can be converted to other types using the following methods:
  – `doubleValue()` – converts the Integer object into a double value
  – `intValue()` – converts the Integer object into an integer value
Converting From Double: An Example

```java
public static void main(String[] args) {
    Double dObject = new Double(5.5);
    int i;
    double x;

    i = dObject.intValue();
    x = dObject.doubleValue();
    System.out.println("The values are:");
    System.out.println(i);
    System.out.println(x);
}
```

Math Class

- The Math class contains a lot of methods for performing basic numeric operations, such as exponentiation, logarithms, square roots, and trigonometric functions.
- The methods of this class are all static, i.e., they all belong to the class as a whole, not to any one method.
Math Class’s Properties

• Math has two properties:
  – E – the root of natural logarithms = 2.718281828…
  – PI – the ratio of the circumference of a circle to its diameter = 3.141592653589…

Math Class’s Numeric Methods

• These methods do basic arithmetic operations:
  – abs(x) – returns absolute value of x.
  – exp(x) – returns e^x
  – min(x, y) – returns minimum of x and y
  – max(x, y) – returns maximum of x and y
  – pow(a, b) – returns a^b
  – sqrt(x) – returns square root of x
• abs, min, max return the same type as their arguments.
• exp, pow and sqrt expect a double value and return a double value
MathTest.java: An Example

```java
import java.lang.Math;

public class MathTest {
    public static void main(String[] args) {
        System.out.println("Abs(-9) = " + Math.abs(-9));
        System.out.println("Exp(2) = " + Math.exp(2));
        System.out.println("Min(3, -4) = 
                + Math.min(3, -4));
        System.out.println("Max(3, -4) = 
                + Math.max(3, -4));
        System.out.println("Pow(2,5) = 
                + Math.pow(2, 5));
        System.out.println("Sqrt(2) = 
                + Math.sqrt(2));
    }
}
```

Output of MathTest.java

Abs(-9) = 9
Exp(2) = 7.38905609893065
Min(3, -4) = -4
Max(3, -4) = 3
Pow(2,5) = 32.0
Sqrt(2) = 1.4142135623730951
Trigonometric Functions

\( \sin(x) \) – returns the sine of \( x \) radians

\( \cos(x) \) – returns the cosine of \( x \) radians

\( \tan(x) \) – returns the tangent of \( x \) radians

\( \text{asin}(y) \) – returns the arc sin(y) in radians in the range \(-\pi/2\) to \(+\pi/2\)

\( \text{acos}(y) \) – returns the arc cos(y) in radians in the range 0 to \(+\pi\)

\( \text{atan}(y) \) – returns the arc tangent in radians in the range \(-\pi/2\) to \(+\pi/2\)

---

**TrigTest.java: An Example**

```java
import java.lang.Math;

public class TrigTest {
    public static void main(String[] args) {
        double pi = Math.PI;
        System.out.println("Sin(pi/4) = " + Math.sin(pi/4));
        System.out.println("Cos(pi/4) = " + Math.cos(pi/4));
        System.out.println("Tan(pi/4) = " + Math.tan(pi/4));
        System.out.println("Arc Sin(.7) = " + Math.asin(0.7));
        System.out.println("Arc Cos(.7) = " + Math.acos(0.7));
        System.out.println("Arc Tan(.7) = " + Math.atan(0.7));
    }
}
```
Output of \texttt{TrigTest.java}

\begin{align*}
\sin(\pi/4) &= 0.7071067811865475 \\
\cos(\pi/4) &= 0.7071067811865476 \\
\tan(\pi/4) &= 0.9999999999999999 \\
\arcsin(0.7) &= 0.775397496610753 \\
\arccos(0.7) &= 0.7953988301841436 \\
\arctan(0.7) &= 0.6107259643892086
\end{align*}