Java Programming: Guided Learning with Early Objects

Chapter 3

Introduction to Objects and Classes

Using Predefined Classes and Methods in a Program

- **Method**: collection of programming statements to accomplish a task
- Java includes many **predefined classes (Math class)**
  - Predefined class includes **predefined methods**
  - Organized as **class libraries**
- To use a predefined method, you must know:
  - Name of the class and package
  - Method name and parameters

Example

Math.pow : class name is Math and the method name is pow (power) separted by a dot

Math.pow(2,3) is 2 to the power 3 which is 8.

A Precaution

- **Dot operator (.) is called a Member access operator** in Java.
- Dot separates reference variable from method
- Methods distinguished from variables by parentheses
- Methods with no parameters must have empty parentheses (such as nextInt()).

Example :

```
console.nextInt()  :  console is the name of the
                      reference variable,
                      nextInt() is the name of the
                      method.
```


The `class String`

- Index of first character in string is 0
- Length of string: number of characters in it
- String variables are reference variables
  - String object an instance of `class String`
- `class String` contains methods to process strings
- `String` variable invokes `String` method using the dot operator, method names, arguments

Sample Program

```
    /*   program illustrates few usages of the predefined String
         methods that are available in Java
     */

public class VariousStringMethods
{
    public static void main(String[] args)
    {
        String sentence;
        String str1;
        String str2;
        String str3;
        int index;

        sentence = "Now is the time for the birthday party";

        System.out.println("sentence = " + sentence );
        System.out.println("The length of sentence = "
                        + sentence.length());

        System.out.println("The character at index 16 in "
                          + "sentence = " + sentence.charAt(16));

        System.out.println("The index of first t in sentence = "
                        + sentence.indexOf('t'));

        System.out.println("The index of for in sentence = "
                        + sentence.indexOf("for"));

        System.out.println("sentence.substring(0, 6) = "
                         + sentence.substring(0, 6) );
```


```
        System.out.println("sentence.substring(7, 12) = "
                           + sentence.substring(7, 12) );

        str1 = sentence.substring(0, 8);

        System.out.println("str1 = " + str1 );

        str2 = sentence.substring(2, 12);
        System.out.println("str2 = " + str2 );

        System.out.println("sentence in uppercase = "
                           + sentence.toUpperCase() );

        index = sentence.indexOf("birthday");

        str1 = sentence.substring(index, index + 14);

        System.out.println("str1 = " + str1 );

        System.out.println("sentence.replace('t', 'T') = "
                           + sentence.replace('t', 'T') );  }   }
```

The `class Math`

- Method type is data type returned by the method
- Contained in package `java.lang`

Using Methods of the `class Math` in a Program

- Java does not require package `java.lang` to be imported
  - Classes automatically imported by default
- `static` method: method heading contains the word `static`
  - **Non static** method does not contain `static`
- **Actual parameters**: parameters used in a method call

- `static import` statements

```
import static packagename.ClassName.*
```


Example

```
import static java.lang.Math.*;

public class Main {

    /* This program Written Husain Gholoom
       program illustrates few usages of the predefined Math
       methods that are available in Java
     */
    public static void main(String[] args) {

    //  used when you do not import the class Math.

  System.out.println("The square root of val is:"+Math.sqrt(30));
  System.out.println(" the sin of val is:"+Math.sin(100));
  System.out.println(" the cosine of val is:"+Math.cos(100));
  System.out.println("The floor value is:"+Math.floor(44.7));
  System.out.println("The ceiling value is:"+Math.ceil(44.7));
  System.out.println("The round value is:"+Math.round(44.7));
  System.out.println(" the max valueis:"+Math.max(75,70));
  System.out.println("the min value"+Math.min(75,70));

  System.out.println(Math.round(1.49));
  System.out.println(Math.round(1.50));
  System.out.println(Math.round(-1.49));
  System.out.println(Math.round(-1.50));

  System.out.println(Math.floor(1.49));
  System.out.println(Math.floor(1.50));
  System.out.println(Math.floor(-1.49));
  System.out.println(Math.floor(-1.50));

  System.out.println(Math.ceil(1.49));
  System.out.println(Math.ceil(1.50));
  System.out.println(Math.ceil(-1.49));
  System.out.println(Math.ceil(-1.50));

//  used when you do not import the class Math.

  System.out.println("10 to the power 3 = " + pow(10, 3));
  System.out.println("The maximum of 23.67 and 14.28 = " +
                      max(23.67, 14.28));
 System.out.println("The absolute value of -67 = "
                     + abs(-67));
  System.out.println("floor(34.23) = " + floor(34.23));
  System.out.printf("sin(PI / 2) = %.2f %n", sin(PI / 2));
  System.out.println("the min value = "+ PI);
   System.out.println("random nuber  =  "+ random());
  System.out.println("random nuber  =  "+ (round((random()*1000)))); }   }
```


Primitive Data Types and Wrapper Classes

- Wrapper classes for primitive types:
  - `class Integer` for `int`
  - `class Long` for `long`
  - `class Double` for `double`
  - `class Float` for `float`
  - `class Character` for `char`

- Wrapper classes are immutable : in object-oriented and functional programming, an **immutable object** is an object whose state cannot be modified after it is created (unchangeable). This is in contrast to a **mutable object**, which can be modified after it is created.

  A classic example of an immutable object is an instance of the Java String class.

```
String s = "ABC";
s.toLowerCase();
```

```
The method toLowerCase() will not change the data
"ABC" that s contains. Instead, a new String object
is instantiated and given the data "abc" during its
construction. A reference to this String object is
returned by the toLowerCase() method.

To make the String s contain the data "abc", a
different approach is needed.
s = s.toLowerCase();
Now the String s references a new String object
that contains "abc". The String class's methods
never affect the data that a String object
contains.
```


The `class Character` (Optional)

- `Class Character` manipulates characters
- Contained in package `java.lang`
- `public static` methods of `class Character` used like methods of `class Math`

```
public class PredefinedMethods3
{
    public static void main(String[] args)
    {
    System.out.println("Character.isDigit('*') = "
                + Character.isDigit('*'));
    System.out.println("Character.isLetter('a') = "
                + Character.isLetter('a'));
    System.out.println("Character.isLetter('*') = "
                + Character.isLetter('*'));
    System.out.println("Character.isLowerCase('a') = "
                + Character.isLowerCase('a'));
    System.out.println("Character.isLowerCase('A') = "
                + Character.isLowerCase('A'));
    System.out.println("Character.isUpperCase('B') = "
                + Character.isUpperCase('B'));
    System.out.println("Character.isUpperCase('k') = "
                + Character.isUpperCase('k'));
    System.out.println("Character.isSpaceChar(' ') = "
                + Character.isSpaceChar(' '));
    System.out.println("Character.isSpaceChar('*') = "
                + Character.isSpaceChar('*'));
    System.out.println("Character.isWhitespace('\\n') = "
                + Character.isWhitespace('\n'));
    System.out.println("Character.isWhitespace('*') = "
                + Character.isWhitespace('*'));
    System.out.println("Character.toLowerCase('D') = "
                + Character.toLowerCase('D'));
    System.out.println("Character.toLowerCase('*') = "
                + Character.toLowerCase('*'));
    System.out.println("Character.toUpperCase('j') = "
                + Character.toUpperCase('j'));
    System.out.println("Character.toUpperCase('8') = "
                + Character.toUpperCase('8'));   }   }
```


Introduction to user defined Methods

A Method is :-

- A series of Java statements that carry out some task.
- Any class can contain an unlimited number of methods.
- Methods in Java are similar to procedures, functions and subroutines in other programming languages

Why to Create Methods

First, the main() method will remain short and easy to follow because main() will contain just one statement to call a method, rather than three separate println() statements to perform the work of the method. A method must include the following:

- A declaration
- An opening curly bracket
- A body of statements
- A closing curly bracket

The declaration will contain:

- Optional access modifiers
- The return type for the method
- The method name
- An opening parenthesis
- An optional list of method arguments, separated by commas
- A closing parenthesis

- Two categories of user-defined methods:
  - **Void methods (none Value-returning methods)**
  - **Value-returning methods**

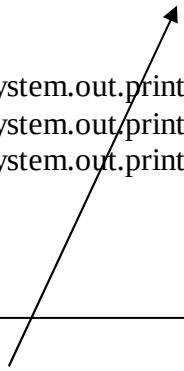
- To use a method in a program, you must know:
  - Name of method
  - Number, type, order of parameters
  - Data type of value returned
  - Code required to accomplish the task

Example

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

       System.out.println("The number of the event site is  25 ");
       System.out.println("Usage fee " + 10);
       System.out.println("Manager is " + "Husain Gholoom");
  }

}
```



The Method main does not contain any other methods.

Another way of writing this is rewrite the code by using separate methods that do not return any values :

```
public class First {
   public static void main(String args[])
        {
        methodPrint();
        System.out.print("My Fist Java Program ");

         }
   public static void   methodPrint()
        {
        System.out.println("The number of the event site is  25 ");
        System.out.println("Usage fee " + 10);
        System.out.println("Manager is " + "Husain Gholoom");
        }
```

In the above example , Main Method calls the methodPrint with no arguments . The control of execution goes back to Main method after last statement in the methodPrint() is executed.

Creating Methods that Require a Single or multiple Arguments

Some methods require additional information. If a method could not receive your communications, called **arguments,** then you would have to write an infinite number of methods to cover every possible situation. An ***access modifier*** for a method may be public, private, friendly, protected, private protected, or static.

Methods are usually given public access, meaning that any class can use the method. To refer to a method in a different class, you use the class name followed by the dot operator followed by the method name. An example is the syntax for System.out.println.

An important principle of object-oriented programming is the notion of **information hiding**. It is not necessary to know the details of how the method is executed, only the name of the method and what type of information the method requires. A programmer can write a new implementation for a method and as long as the interface for the method is not changed, the calling programs will require no modification. The method declaration for a method that can receive an argument must include the type of argument and the local name for the argument within the declaration parenthesis.

- Formal parameter: declared in method header
- Actual parameter: listed in call to a method
- **`methodName`**: Java identifier
- Syntax of a formal parameter list:
  `dataType identifier, dataType identifier, …`
- Method call syntax:
  `methodName (actual parameter list);`

- Actual parameter list syntax:
  `expression or variable, expression or variable,…`

- Call a method, use name together with parameters in parentheses

- Formal parameter list may be empty
  – Parentheses still needed

- Actual parameter list is empty if formal parameter list is also empty

Example :-

```
public class Second {

     // Illustrate a Method that Requires a Single Argument
    // This class calculates a waitperson's tip

        public static void main(String args[])
            {
                double myCheck = 50.00;
                double yourCheck = 19.95;
                System.out.println("Tips are");
                calcTip(myCheck);
                calcTip(yourCheck);
            }
        public static void calcTip(double  bill)
           {
                double tip;
                tip = bill * .15;
                System.out.println("The tip should be at least " + tip);
           }    }
```


Creating Methods that Require Multiple Arguments

A method can require more than one argument. You can pass multiple arguments to a method by listing the arguments within the call to the method and separating them with commas.

Example :-

```
public class Third {

      // This program is Written By Husain Gholoom
      // This program calculates tuition bills

       public static void main(String args [])
            {
                 int myCredits = 15;
                 double yourCredits = 16.5;
                 double rate = 75.84;
                 tuitionBill(myCredits,rate);
                 tuitionBill(yourCredits,rate);
                 System.out.println("myCredits  =   " + myCredits);
                 System.out.println("yourCredits  =   " + yourCredits);

            }
       public static void tuitionBill(int c , double r)
           {
            System.out.println("Total due " + (r*c));
           }
      public static void tuitionBill(double c, double r)
           {
            System.out.println("Total due " + (r*c));
           }
}
```

How many methods are in this class?
Does the value of myCredits and yourCredits changes after tuitionbill method is executed. ?

Value-returning methods

- Like functions in other programming languages
- Formal parameter: declared in method header
- Actual parameter: listed in call to a method
- **`modifier(s)`**: visibility of method (public , private ..)
- ***`returnType`**: type of value method returns*
- **`methodName`**: Java identifier
- Syntax of a formal parameter list:
  `dataType identifier, dataType identifier, …`
- Method call syntax:
  `methodName (actual parameter list);`
- Actual parameter list syntax:
  `expression or variable, expression or variable,…`
- Call a method, use name together with parameters in parentheses
- Formal parameter list may be empty
  – Parentheses still needed
- Actual parameter list is empty if formal parameter list is also empty

return Statement

- Method syntax:

```
modifier(s) returnType methodName
            (formal parameter list)
{
   statement(s)
}
```

- Value-returning methods use `return` statement to return a value
  - Pass value back when method completes
- Syntax: `return expr;`
- `expr` is a variable, constant, or expression
- `return` is a reserved word
- Method immediately terminates and control goes back to caller after `return` statement

Example :-

```
public class Class1V2 {

    //  This Program is Written By Husain Gholoom
    //  This program Demonstrates A Calculator
    //  This Program uses 4  methods within  One  class
    //  It uses methods that return values
    //  ( Like Functions in Other Programming Languages )

       public static void main(String args [])
           {
               int No1 = 15;
               int No2 = 10;
               int Add       = addOperation(No1,No2);
               int Subtract  = subtractOperation(No1,No2);
               int Multiply  = multiplyOperation(No1,No2);
               double  Divide    = divideOperation(No1,No2);
               System.out.println("No1 + No2  =   " + Add   );
               System.out.println("No1 - No2  =   " + Subtract);
               System.out.println("No1  *  No2  =   " + Multiply);
               System.out.println("No1  /  No2  =   " + Divide);

           }
       public static int addOperation(int X , int Y)
          {
           return  (X + Y);
          }
       public static int  subtractOperation(int X , int Y)
          {
            return ( X - Y );
           }

       public static int multiplyOperation(int X , int Y)
          {
            return ( Y * Y );
          }
      public static double  divideOperation(double  X , double Y)
          {
            return ( X / Y );
          }  }
```


Nested Methods that returns values

```
public class Assignment2Convert {
   //  This  Program is Written By Husain Gholoom
   //  The Function of this Program is to convert
   //  measurement given in feet to its equivalent number
   //   in inches, yards, centimeters, and meters
   //  It uses calls within methods to other methods


     public static void main(String[] args) {
      int Feet = 100000 , Yard , Centimeter , Meter;
      double Inches;

      feetToInche (100000);  // Convert feet to inche
      yardToFeet (100000);  //  Convert Feets to Yard
      inchesToCentimeter (100000);  // Convert Feets to centimeters
      meterToCentimeter (100000); // Convert feets to Meters
   }

   public static void feetToInche ( int infeet )       {
     //    1  feet   =  12  inche

     System.out.println( infeet  +"   feet   =    "  + infeet*12 + " inche ");     }

   public static void yardToFeet ( int infeet )  {
    //   1   yard  =  3  feet, thus     1  feet  =  1/3  yard

      System.out.println( infeet  +"   feet   =    "  + (infeet / 3.0) + " yard ");    }

 public static void inchesToCentimeter ( int infeet )      {
       // 1  inche  =  2.54 centimeter,  thus  we need to convert
       //  feet  to inches , then iches to centimeters

       int inches = feetToInchefunction ( infeet );
       System.out.println( infeet  +"   feet   =    "  + (inches* 2.54) + " Centimeters ");   }

   public static int feetToInchefunction ( int infeet )   {
   //    Function to return the conversion value from feety to inches
   //    1  feet   =  12  inche

         return  infeet*12 ;     }
```


```
public static double inchesToCentimeterFunction ( int infeet )    {
        // 1 inche = 2.54 centimeter, thus we need to convert
        // feet to inches , then iches to centimeters

        int inches = feetToInchefunction ( infeet );
        return inches * 2.54 ;                                    }

   public static void meterToCentimeter ( int infeet )   {
     int inches = feetToInchefunction ( infeet );
     double centimeter = inchesToCentimeterFunction ( inches);
     System.out.println( infeet +"   feet   =    " + (centimeter / 100.0) + " Meter ");   }

}
```

Void Method

- Value-returning method returns one value
- May want to return no value or many values
- Void method returns no value:

```
public static void printNum(double x)
{
   System.out.print(x);
}
```


Flow of (Method) Execution

- Methods can appear in any order
  - Method `main` executes first
  - Other methods execute when called
- Method call transfers control to first statement in method body
- After method finishes, control passed back to caller
- After value-returning method executes, return value replaces method call statement in caller

Global and Local Variables Declarations

Global Variables : variables that are defined out side the entire class and are visible to all methods within a class.

Local Variables : variables that are defined inside a method and are visible only in that method.

Example:

```
public static double square(double num)
{
   return num * num;
}
```


Example Of Global & local variables

```
public class Class1V2 {

      //  This Program is Written By Husain Gholoom
      //  This program Demonstrates A Calculator
      //  This Program uses 4  methods within  One  class
      //  It uses methods that return values
      //  ( Like Functions in Other Programming Languages )

         static          int No3 ;

         public static void main(String args [])
             {
                 int No1 = 15;
                 int No2 = 10;
                 int Add       = addOperation(No1,No2);
                 int Subtract  = subtractOperation(No1,No2);
                 int Multiply  = multiplyOperation(No1,No2);
                 double  Divide    = divideOperation(No1,No2);
                 System.out.println("No1 + No2  =  " + Add  );
                 System.out.println("No1 - No2  =  " + Subtract);
                 System.out.println("No1  *  No2  =  " + Multiply);
                 System.out.println("No1  /  No2  =  " + Divide);

             }
         public static int addOperation(int X , int Y)
            {
              int Z;
              No3 = X + Y;
              Z = X + Y;
              return  (X + Y);            }
        public static int  subtractOperation(int X , int Y)
           {
              int Z;
              No3 = X -  Y;
              Z = X - Y;
              return  (X - Y);            }

        public static int multiplyOperation(int X , int Y)
           {
              int Z;
              No3 = X * Y;
              Z = X * Y;
              return Z;            }
       public static double  divideOperation(double  X , double Y)
           {              return ( X / Y );    }

}
```

Global Variable

Local Variables

Z is Local To AddOperation method Only

No3 is Global

What is the value of No3 after the program is terminated. ??????

Another way is to have each operation in separate class.

```
public class MainClass {

      //  This Program is Written By Husain Gholoom
      //  This program Demonstrates A Calculator. it uses
      //  multiple classes to perform this operation
     //    Main Class is calling AddClass, SubtractClass, MultiplyClass
    //   and Divide Class

            public static void main(String args [])
                {
                    int No1 = 15;
                    int No2 = 10;
                    AddClass.addOperation(No1,No2);
                    SubtractClass.subOperation(No1,No2);
                    MultiplyClass.multiplyOperation(No1,No2);
                    DivideClass.divideOperation(No1,No2);

                }  }
```

```
public class AddClass {

   public static void addOperation(int No1 , int No2)
           {
            System.out.println( No1  + "  +  " + No2 + "  =   " + (No1 + No2));
           }
}
```

```
public class SubtractClass {

   public static void subOperation(int No1 , int No2)
           {
            System.out.println( No1  + "  -  " + No2 + "  =   " + (No1 - No2));
           }
}
```


```
public class MultiplyClass {

   public static void multiplyOperation(int No1 , int No2)
          {
           System.out.println( No1  + "  *  " + No2 + "  =   " + (No1 * No2));
          }
}
```

```
public class DivideClass {

   public static void divideOperation(int No1 , int No2)
         {
          System.out.println( No1  + "  /  " + No2 + "  =   " + (No1 / No2));
         }
}
```

Classes

- **Class**: a collection of components called **members**
- Syntax for defining a class:

```
modifier(s) class ClassName modifier(s)
{
   classMembers
}
```

- `private`: cannot be accessed directly outside the class
- `public`: can be accessed directly outside class
- Members accessed directly from outside the class should be `public`
- Members that should not be accessed directly should be `private`
- User should never be given direct access to data members
- **Instance variables**: variables declared without using the modifier `static`
- Non static methods of a class are **instance methods**

Constructors

- Constructor has same name as the class
- Executes automatically when object is created
- Used to guarantee instance variables are initialized
- Two types of constructors:
  - With parameters
  - Without parameters (**default constructor**)
- Has no return type
- Class can have more than one, but must have same name, different parameters

Variable Declaration and Object Instantiation

- Declaring a reference variable of `class` type does not allocate memory
- Syntax for `new` operator:

```
new className();
new className(arg1, arg2, …,argN);
```

- Number of arguments and types match formal parameters
- If type of arguments does not match, Java tries to find the best match

Defining Methods and Constructors

- Method definition:

```
public void setCurrentTemp(int cTemp) {
   currentTemp = cTemp;
}
```

- Constructor definitions:

```
public Thermometer() {
   currentTemp = 0;
}
public Thermometer (int cTemp) {
   currentTemp = cTemp;
}
```


The Method `toString`

- Definition of method `toString`:

```
public String toString() {
   return "Current temp: " + currentTemp;
}
```

- Suppose `myTemp` is a `Thermostat` object
  - `System.out.println(myTemp)` is:

```
Current temperature: 40
```

- When object reference parameter to `print`, `println`, `printf`, method `toString` called

Summary

- Reference variables store a memory location
- Objects are instances of a class
  - Use the `new` operator to instantiate an object
- Unused memory collected and freed using Java garbage collector
- Predefined classes organized as collections of packages called class libraries
- Package must be imported to be used
- All primitive data types have corresponding wrapper class
- Wrapping and unwrapping facilitated by autoboxing and auto-unboxing
- Two types of methods:
  - Value-returning methods
  - Void methods
- Two types of parameters:
  - Actual parameters
  - Formal parameters
- Encapsulation: combining data and operations