Introduction to Programming in Java

Presented by

Dr. Rubeena Doomun

Programming paradigms

1. **Declarative language:**
   In a declarative programming style you describe the results that you want, but not how to get there.
   Example: SQL language
2. **Imperative:**
   The word "imperative" comes from the Latin "impero" meaning "I command". You are the emperor. You give the computer little orders to do and it does them one at a time and reports back.
3. **Procedural :**
   Procedural programming is imperative programming where you are allowed to write reusable subs/functions.
4. **Functional:**
   In functional programming you compose your program of short functions. All codes are within a function. All variables are scoped to the function and you cannot modify any variables.
5. **Object Oriented Programming(OOP):**
   In Object Oriented Programming(OOP) you create objects in order to be able to group themselves in different categories. Therefore, each particular group has their own characteristics, also known as properties. In this way, we avoid redundancy of codes as objects will inherit the properties of the groups they belong.

Value and Type

- A *value* is simply a piece of information, such as a number, like the value 43, or a string, like the value "Hello world!" A "string" is: a string of characters.
- Each value belongs to a *type*, often known as data type, where a type is simply a set of possible values that are related by the fact that they can be processed by similar operations.
- In Java, there exist two types:
  1. Primitive Data Types
  2. Reference/Object Data Types

Primitive v/s Reference type

- A primitive type is predefined by the language and is named by a reserved keyword. There are 8 primitive data types.
- All types which are not among the 8 primitive data types are referred as reference data type. For instance, the Java programming language also provides special support for character strings via the java.lang.String class. Therefore, a string literal is a reference to an instance of class String.

Primitive Data types

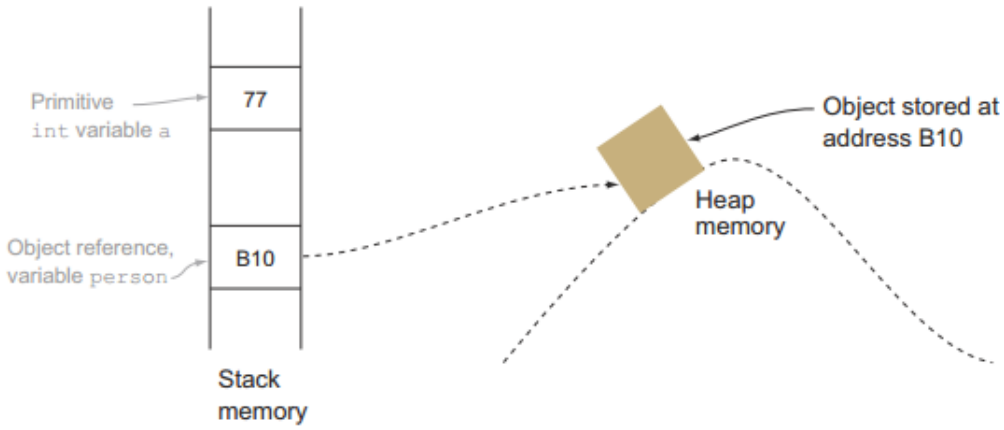
- Int
- Boolean
- Long
- Short
- Double
- Float
- Byte
- Char

Variable

- A *variable* is a name that refers to a value and which is not a pre-defined name in the particular language code you are programming. As variable needs to refer to a value, a variable needs to be associated with a particular data type.
- As there are exist two different data types, there exist two different variables types: primitive variable and reference variable.
- Primitive variables store primitive values.
- Reference/class variables store addresses.

Difference between primitive and reference variable

```
int a = 77;
Person person = new Person();
```



Primitive variables store the actual values, whereas object reference variables store the addresses of the objects they refer to.

Variable

- Variables can be accessed only to a particular location. They are known as local variables.
- Some variables can be accessed from anywhere, they are known as global variables. Global variables can be accessed by using the instance of a particular class in OOP. In OOP, these global variables are referred to instance variables.
- Another type of variables which exist in OOP, is the static variable. This will be covered more in detail when learning OOP.

Variable

- Local variable

  Local variables are declared in methods, constructors, or blocks.
- Instance variable

  Instance variables are declared in a class, but outside a method, constructor or any block.
- Class/static variable

  Class variables also known as static variables are declared with the static keyword in a class, but outside a method, constructor or a block.

Exercise 1

- What do each of the following print?
  a. System.out.println(2 + "bc");
  b. System.out.println(2 + 3 + "bc");
  c. System.out.println((2+3) + "bc");
  d. System.out.println("bc" + (2+3));
  e. System.out.println("bc" + 2 + 3);
  f. System.out.println(“bc”+(2+3)+”a”+5);
  g. System.out.println(5+“bc”+(2*3));

Operator

- Arithmetic Operators
- Relational Operators
- Logical Operators
- Assignment Operators

Arithmetic Operators

Operator	Description
+	Addition - Adds values on either side of the operator
-	Subtraction - Subtracts right hand operand from left hand operand
*	Multiplication - Multiplies values on either side of the operator
/	Division - Divides left hand operand by right hand operand
%	Modulus - Divides left hand operand by right hand operand and returns remainder
++	Increment - Increases the value of operand by 1
--	Decrement - Decreases the value of operand by 1

Relational Operators

Operator	Description
==	Checks if the values of two operands are equal or not, if yes then condition becomes true.
!=	Checks if the values of two operands are equal or not, if values are not equal then condition becomes true.
>	Checks if the value of left operand is greater than the value of right operand, if yes then condition becomes true.
<	Checks if the value of left operand is less than the value of right operand, if yes then condition becomes true.
>=	Checks if the value of left operand is greater than or equal to the value of right operand, if yes then condition becomes true.

Logical Operators

Operator	Description
&&	Called Logical AND operator. If both the operands are non-zero, then the condition becomes true.
||	Called Logical OR Operator. If any of the two operands are non-zero, then the condition becomes true.
!	Called Logical NOT Operator. Use to reverses the logical state of its operand. If a condition is true then Logical NOT operator will make false.

Assignment Operators

=	Simple assignment operator, Assigns values from right side operands to left side operand
+=	Add AND assignment operator, It adds right operand to the left operand and assign the result to left operand
-=	Subtract AND assignment operator, It subtracts right operand from the left operand and assign the result to left operand
*=	Multiply AND assignment operator, It multiplies right operand with the left operand and assign the result to left operand
/=	Divide AND assignment operator, It divides left operand with the right operand and assign the result to left operand
%=	Modulus AND assignment operator, It takes modulus using two operands and assign the result to left operand

Exercise 2

- Write down the operators used to represent the following:
- AND
- OR
- NOT
- To check if the value of two operands are equal or not
- Increment a value by 1
- Decrement a value by 1
- Returns the remainder
- Simple assignment operator

Class and Object

- A class is a prototype that defines variables and methods common to all objects of a given type.
- An objet is an instance of a class.

Method

- Method definition consists of a method header and a method body.

```
public static int funcName(int a, int b) {
   // body
}
```

- public static : modifier.
- int: return type
- funcName: function name
- int a, int b: list of parameters

Method

- Methods are also known as Procedures or Functions:

  Procedures: They don't return any value.

  Functions: They return value.

```
modifier returnType nameOfMethod (Parameter List) {
 // method body
}
```

Syntax of a method

- modifier:

  It defines the access type of the method and it is optional to use.
- returnType:

  Method may return a value.
- nameOfMethod:

  This is the method name. The method signature consists of the method name and the parameter list.
- Parameter List:

  The list of parameters, it is the type, order, and number of parameters of a method. These are optional, method may contain zero parameters.
- method body:

  The method body defines what the method does with statements.

Parameter Passing

- Passing Parameters by Value means calling a method with a parameter.

```
public class ExampleMinNumber{

   public static void main(String[] args) {
      int a = 11;
      int b = 6;
      int c = minFunction(a, b);
      System.out.println("Minimum Value = " + c);
   }

   /** returns the minimum of two numbers */
   public static int minFunction(int n1, int n2) {
      int min;
      if (n1 > n2)

         min = n2;
      else
         min = n1;

      return min;
   }
}
```

Void keyword

- The void keyword allows us to create methods which do not return a value.

```
Public void setName(String a){
        //implementation
}
```

Constructor

- A constructor is a method that can be used to initialise a new object.
- Example:

```
public class MyClass{
    //This is the constructor
    MyClass(){
    }
    ..
}
```

Example - constructor

```
public class Hello {
   String name;
   //Constructor
   Hello(){
      this.name = "BeginnersBook.com";
   }
   public static void main(String[] args) {
      Hello obj = new Hello();
      System.out.println(obj.name);
   }
}
```

Selection

- If else statement

```
if(Boolean_expression){
//Executes when the Boolean expression is true
}else{
//Executes when the Boolean expression is false
}
```

If ..else if ..else statement

```
if(Boolean_expression1){
//Executes when the Boolean expression 1 is true
}elseif(Boolean_expression2){
//Executes when the Boolean expression 2 is true
}elseif(Boolean_expression3){
//Executes when the Boolean expression 3 is true
}else{
//Executes when the none of the above condition is true.
}
```

Nested if...else Statement

```
if(Boolean_expression1){
//Executes when the Boolean expression 1 is true
if(Boolean_expression2){
//Executes when the Boolean expression 2 is true
}
}
```

switch Statement

```
switch(expression){
case value :
//Statements
break;//optional
case value :
//Statements
break;//optional
//You can have any number of case statements.
default://Optional
//Statements
}
```

Iteration

- while Loop

  A while loop is a control structure that allows you to repeat a task a certain number of times.

```
while(Boolean_expression)
{
//Statements
}
```

do...while Loop

- A do...while loop is similar to a while loop, except that a do...while loop is guaranteed to execute at least one time.

```
do
{
//Statements
}while(Boolean_expression);
```

for Loop

- A for loop is a repetition control structure that allows you to efficiently write a loop that needs to execute a specific number of times.

```
for(initialization;Boolean_expression; update)
{
//Statements
}
```

References

- https://www.tutorialspoint.com/java/index.htm
- https://docs.oracle.com/javase/tutorial/java/index.html