CONTROL-FLOW STATEMENTS

Java Control statements control the order of execution in a javaprogram, based on data values and conditional logic.

There are three main categories of control flow statements;

- **Selection statements:** if, if-else and switch.
- **Loop statements:** while, do-while and for.
- **Transfer statements:** break, continue, return, try-catch-finally and assert.

We use control statements when we want to change the default sequential order of execution

1. Selection statements (Decision Making Statement)

There are two types of decision making statements in Java. They are:

- if statements
- if-else statements
- nested if statements
- if-else if-else statements
- switch statements

if Statement:

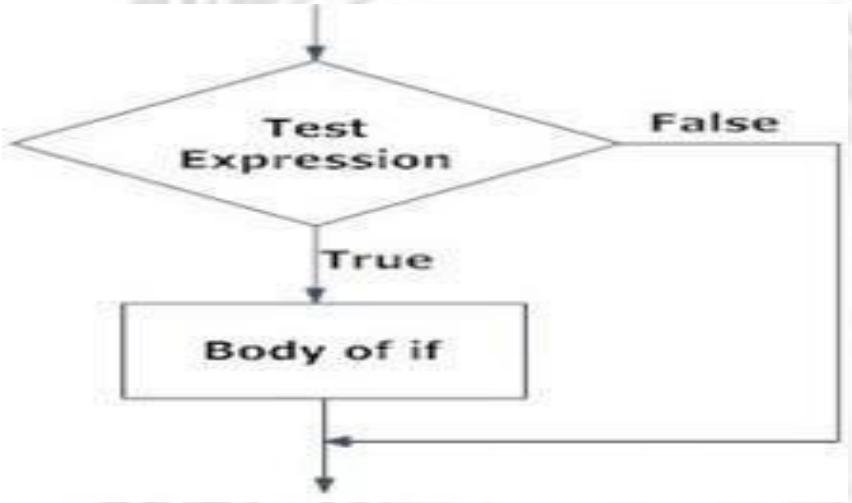
- An if statement consists of a Boolean expression followed by one or more statements.
- Block of statement is executed when the condition is true otherwise no statement will be executed.

Syntax:

```
if(<conditional expression>)
{
  < Statement Action>
}
```

If the Boolean expression evaluates to true then the block of code inside the if statement will be executed.
If not the first set of code after the end of the if statement (after the closingcurly brace) will be executed.

Flowchart:



Example:

```
public class IfStatementDemo {

        public static void main(String[] args)
        {
        int a = 10, b = 20;
        if (a > b)
              System.out.println("a > b");
         if (a < b)
              System.out.println("b > a");
        }
      }
```

Output:

```
$java IfStatementDemo
b > a
```

if-else Statement:

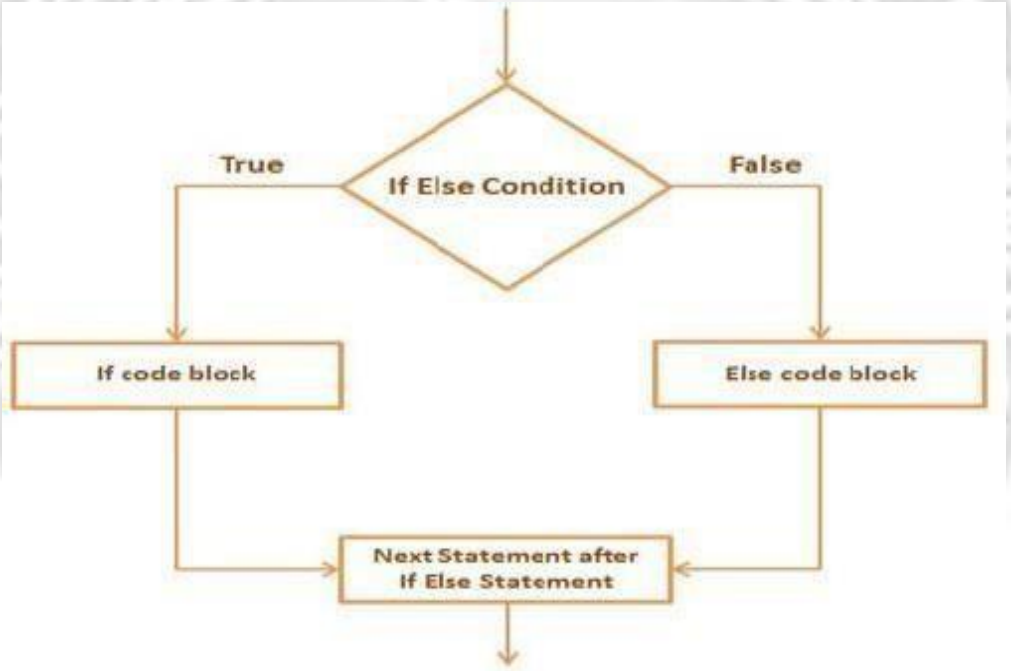
The if/else statement is an extension of the if statement. If the statements in the

ifstatement fails, the statements in the else block are executed.

Syntax:

The if-else statement has the following syntax:

```
if(<conditional expression>)
{
  < Statement Action1>
}
else
{
  < Statement Action2>
}
```



Example:

```
public class IfElseStatementDemo {

        public static void main(String[] args)
        {
                int a = 10, b = 20;
                if (a > b) {
                        System.out.println("a > b");
                }
                else {
                        System.out.println("b > a");
                }
        }
}
```

Output:

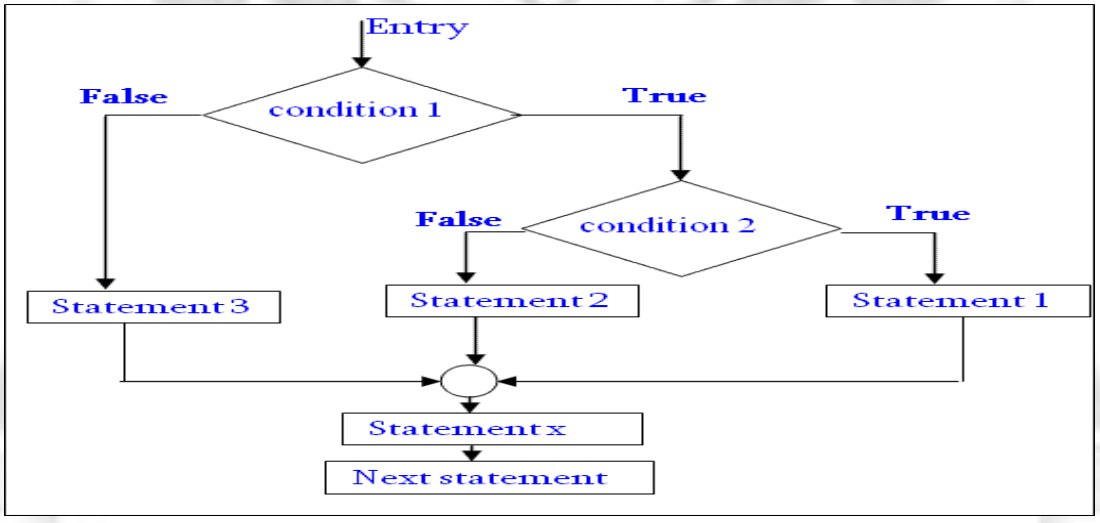
```
$java IfElseStatementDemo
    b > a
```

Nested if Statement:

Nested if-else statements, is that using one if or else if statement inside another if or else ifstatement(s).

Syntax:

```
if(condition1)
{
    if(condition2)
    {
    //Executes this block if  condition is True
    }
   else
   {
    //Executes this block if condition is false
   }
}
else
{
 //Executes this block if condition is false
}
```



Example-nested-if statement:

```
class NestedIfDemo
{
```

```
public static void main(String args[])
{
int i = 10;
if (i ==10)
{
if (i < 15)
{
System.out.println("i is smaller than 15");
}
else
{
System.out.println("i is greater than 15");
}
}
else
{
System.out.println("i is greater than 15");
}
}
}
```

Output:

i is smaller than 15

if...else if...else Statement:

An if statement can be followed by an optional *else if...else* statement, which is veryuseful to test various conditions using single if...else if statement.

Syntax:

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

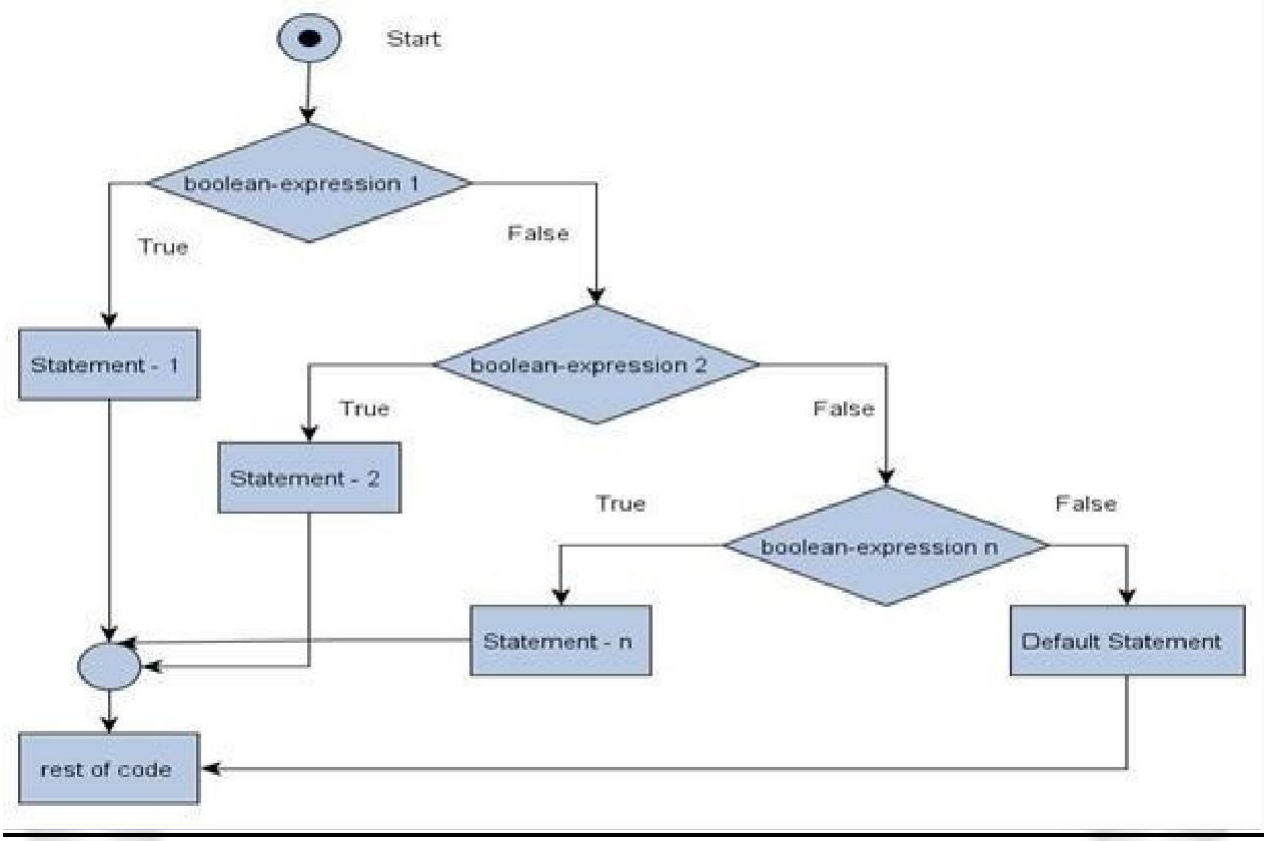
Example:

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

```
    int x = 30;

    if( x == 10 ){

      System.out.print("Value of X is 10");
    }else if( x == 20 ){
      System.out.print("Value of X is 20");
    }else if( x == 30 ){
      System.out.print("Value of X is 30");
    }else{
      System.out.print("This is else statement");
    }
  }
}
```

Output:

Value of X is 30



switch Statement:

- **The switch case statement, also called a case statement is a multi-way branch with several choices. A switch is easier to implement than a series of if/else statements.**
- A *switch* statement allows a variable to be tested for equality against a list of

values. Eachvalue is called a case, and the variable being switched on is checked for each case.

- The switch statement begins with a keyword, followed by an expression that equates to a no long integral value. Following the controlling expression is a code block that contains zero or more labeled cases. Each label must equate to an integer constant and each must be unique.
- When the switch statement executes, it compares the value of the controlling expression to the values of each case label.
- The program will select the value of the case label that equals the value of the controllingexpression and branch down that path to the end of the code block.
- If none of the case label values match, then none of the codes within the switch statementcode block will be executed.
- Java includes a default label to use in cases where there are no matches. We can have a nested switch within a case block of an outer switch.

Syntax:

switch (<expression>)
{
case label1:
<statement1>
case label2:
<statement2>
…
case labeln:
<statementn>
default:
<statement>
}

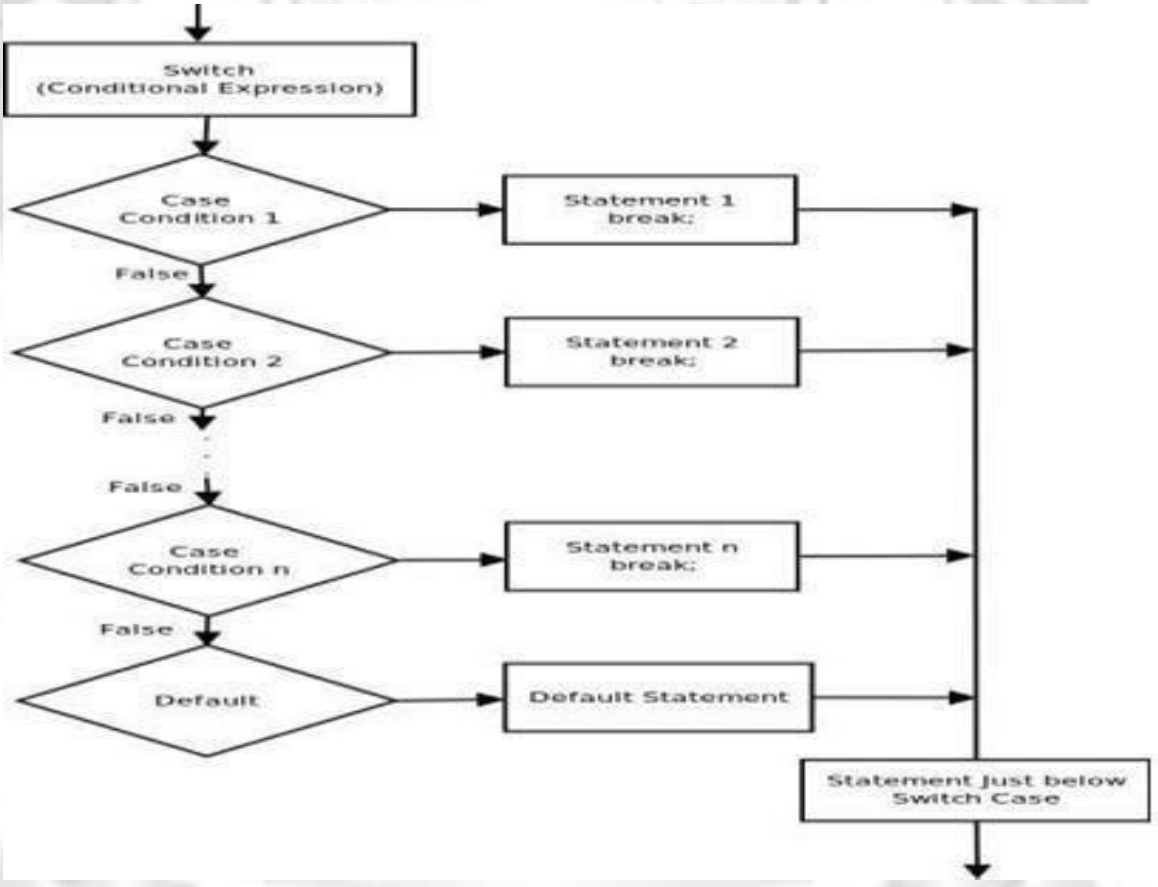
Example:

```
public class SwitchCaseStatementDemo {

    public static void main(String[] args) {int a =
            10, b = 20, c = 30;
            int status = -1;
            if (a > b && a > c) {
                    status = 1;
            } else if (b > c) {
                    status = 2;
            } else {
```

```
                status = 3;
            }
            switch (status) {case 1:
                System.out.println("a is the greatest");break;
            case 2:
                System.out.println("b is the greatest");break;
            case 3:  System.out.println("c is the greatest");break;
            default: System.out.println("Cannot be determined");
            }
        }
}
```

Output:

c is the greatest



2. Looping Statements (Iteration Statements)

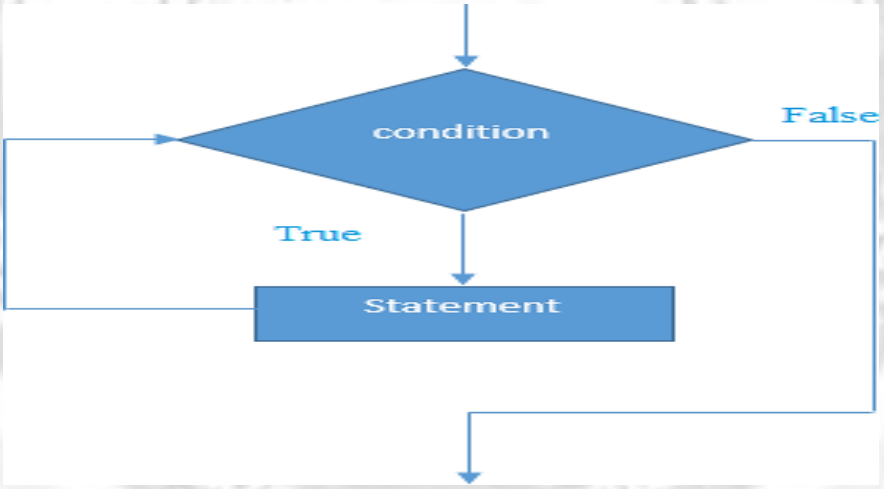
While Statement

- The while statement is a looping control statement that executes a blockof code while a condition is true. It is entry controlled loop.
- You can either have a single statement or a block of code within the while loop.The loop will never be executed if the testing expression evaluates to false.
- The loop condition must be a boolean expression.

Syntax:

The syntax of the while loop is

```
while (<loop condition>)
{
<statements>
}
```



Example:

```
public class WhileLoopDemo {
        public static void main(String[] args) {int
                count = 1;
                System.out.println("Printing Numbers from 1 to 10");
                while (count <= 10) {
                        System.out.println(count++);
                }
        }
}
```

Output

```
Printing Numbers from 1 to 10
        1
        2
        3
```

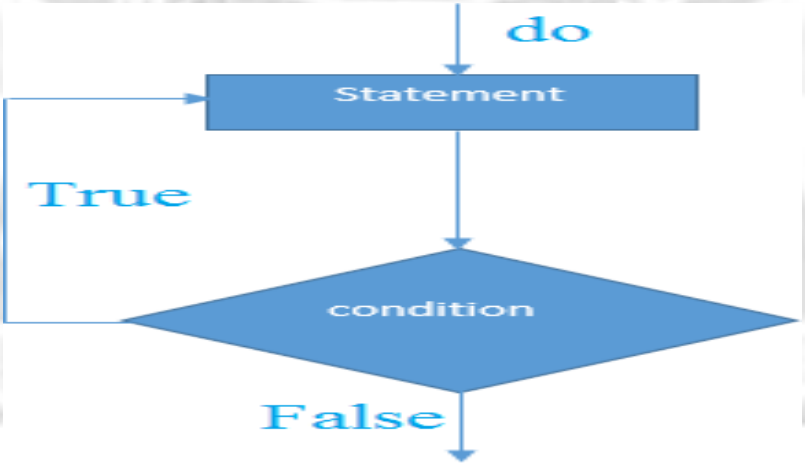
4
5
6
7
8
9
10

do-while Loop Statement

- do while loop checks the condition after executing the statements atleast once.
- Therefore it is called as Exit Controlled Loop.
- The do-while loop is similar to the while loop, except that the test is performed at the endof the loop instead of at the beginning.
- This ensures that the loop will be executed at least once. A do-while loop begins with the keyword do, followed by the statements that make up the body of the loop.

Syntax:

```
do
{
<loop body>
}while (<loop condition>);
```



Example:

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

```
        int count = 1;
        System.out.println("Printing Numbers from 1 to 10");
        do {
                System.out.println(count++);
        } while (count <= 10);
    }
}
```

Output:

```
Printing Numbers from 1 to 10
1
2
3
4
5
6
7
8
9
10
```

For Loops

The for loop is a looping construct which can execute a set of instructions a specified number of times. It's a counter controlled loop. A for statement consumes the initialization, condition and increment/decrement in one line. It is the entry controlled loop.

Syntax:

```
for (<initialization>; <loop condition>; <increment expression>)
{
        <loop body>
}
```

- ✓ The first part of a for statement is a starting initialization, which executes once before the loop begins. The <initialization> section can also be a comma-separatedlist of expression statements.
- ✓ The second part of a for statement is a test expression. As long as the expression istrue, the loop will continue. If this expression is evaluated as false the first time, theloop will never be executed.

- ✓ The third part of the for statement is the body of the loop. These are the instructionsthat are repeated each time the program executes the loop.
- ✓ The final part of the for statement is an increment expression that automatically executes after each repetition of the loop body. Typically, this statement changes the value of the counter, which is then tested to see if the loop should continue.

Exmple:

```
public class ForLoopDemo {
        public static void main(String[] args)
        {
        System.out.println("Printing Numbers from 1 to
                10");
        for (int count = 1; count <= 10; count++)
        {
                        System.out.println(count);
       }
}
}
```

Output:

```
Printing Numbers from 1 to 10
1
2
3
4
5
6
7
8
9
10
```

Enhanced for loop or for- each loop:

As of Java 5, the enhanced for loop was introduced. This is mainly used for Arrays.

- ✓ The for-each loop is used to traverse array or collection in java.
- ✓ It is easier to use than simple for loop because we don't need to increment value and use subscript notation.
- ✓ It works on elements basis not index.
- ✓ It returns element one by one in the defined variable.

Syntax:

```
for(declaration : expression)
{
   //Statements
}
```

- **Declaration:** The newly declared block variable, which is of a type compatible with the elements of the array you are accessing. The variable will be available within the for block and its value would be the same as the current array element.
- **Expression:** This evaluates to the array you need to loop through. The expression can be an array variable or method call that returns an array.

Example:

```
public class Test {


public static void main(String args[])
{
   int [] numbers = {10, 20, 30, 40, 50};

   for(int x : numbers )
   {
      System.out.print( x );
      System.out.print(",");
   }
   System.out.print("\n\n");
   String [] names ={"B", "C", "C++", "JAVA"};
   for( String name : names )
   {
      System.out.print( name );
      System.out.print(",");
   }
   }
   }
```

Output:

```
10,20,30,40,50,
B,C,C++,JAVA
```

1. Transfer Statements / Loop Control Statements/Jump Statements)

1. break statement
2. continue statement

1. Using break Statement:

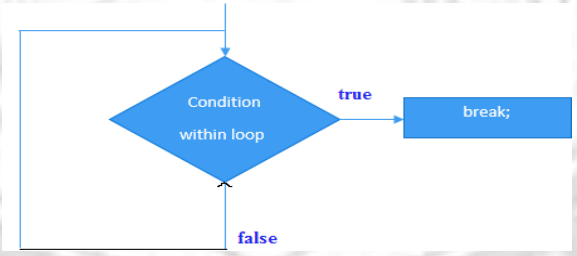
- ✓ The *break* keyword is used to stop the entire loop. The break keyword must beused inside any loop or a switch statement.
- ✓ The break keyword will stop the execution of the innermost loop and start executingthenext line of code after the block.

Syntax:

The syntax of a break is a single statement inside any loop:

```
break;
```

Flowchart:



Example:

```
public class Test {
public static void main(String args[]) { int [] numbers = {10, 20, 30, 40, 50};
for(int x : numbers ) { if( x == 30 ) { break;
}
System.out.print( x ); System.out.print("\n");
}
}
}
```

Output:

10
20

2. Using continue Statement:

- The *continue* keyword can be used in any of the loop control structures. It causes theloop to immediately jump to the next iteration of the loop.
- **The Java *continue statement* is used to continue loop. It continues the current flow of the program and skips the remaining code at specified condition. In case of inner loop, it continues only inner loop.**

Syntax:

The syntax of a continue is a single statement inside any loop:

continue;

Example:

```
public class Test {

  public static void main(String args[]) {int
    [] numbers = {10, 20, 30, 40, 50};

    for(int x : numbers )
    {
      if( x == 30 )
      {
        continue;
      }
      System.out.print( x );
      System.out.print("\n");
    }
  }
}
```

Output:

10
20
40
50

DEFINING CLASSES and OBJECTS

A class is a collection of similar objects and it contains data and methods that operate on that data. In other words – **Class is a blueprint or template for a set of objects that share a common structure and a common behavior**.

DEFINING A CLASS:

The keyword **class** is used to define a class.

Rules to be followed:

1. Classes must be enclosed in parentheses.
2. The class name, superclass name, instance variables and method names may be any validJava identifiers.
3. The instance variable declaration and the statements of the methods must end with ;(semicolon).
4. The keyword **extends** means derived from i.e. the class to the left of the **extends** (subclass) is derived from the class to the right of the **extends** (superclass).

Syntax to declare a class:

```
[public|abstract|final] class class_name [extends superclass_name implements interface_name]
{
        data_type instance_variable1;
        data_type instance_variable2;
        .
        .
        data_type instance_variableN;

        return_type method_name1(parameter list)
        {
        Body of the method
        }
        .
        .
        return_type method_nameN(parameter list)
        {
        Body of the method
        }
}
```

- ✓ The data, or variables, defined within a **class** are called *instance variables*.
- ✓ The code to do operations is contained within *methods*.
- ✓ Collectively, the methods and variables defined within a class are called *members* of

the class.

- ✓ Variables defined within a class are called instance variables because each instance of the class (that is, each object of the class) contains its own copy of these variables.
- ✓ Thus, the data for one object is separate and unique from the data for another.

✓ **Example:**

```
class box {
  double width;
  double height;
  double depth;
  void volume()
 {
          System.out.println(–\n Volume is : –);
          Systme.out.println(width*height*depth);
  }
}
```

Program Explanation:

Class : keyword that initiates a class definition
Box : class name
Double : primitive data type
Height, depth, width: Instance variables
Void : return type of the method
Volume() : method name that has no parameters

DEFINING OBJECTS

An **Object** is an instance of a class. It is a blending of methods and data.

Object = Data + Methods

- It is a structured set of data with a set of operations for manipulating that data.
- The methods are the only gateway to access the data. In other words, the methods and dataare grouped together and placed in a container called Object.

Characteristics of an object:

An object has three characteristics:

1) **State:** represents data (value) of an object.
2) **Behavior:** represents the behavior (functionality) of an object such as deposit, withdraw etc.
3) **Identity:** Object identity is an unique ID used internally by the JVM to identify each object uniquely.

For Example: Pen is an object. Its name is Reynolds, color is white etc. known

as its state. It is used to write, so writing is its behavior.

CREATING OBJECTS:

Obtaining objects of a class is a two-step process:

1. Declare a variable of the class type – this variable does not define an object. Instead,it is simply a variable that can refer to an object.
2. Use **new** operator to create the physical copy of the object and assign the referenceto the declared variable.

NOTE: The **new** operator dynamically allocates memory for an object and returns a referenceto it. This reference is the address in memory of the object allocated by **new.**

Advantage of using new operator: A program can create as many as objects it needs duringthe execution of the program.

Syntax:

```
class_name object_name = new class_name();
    (or)
class_name object_name;
object_name = new class_name();
```

Example:

```
box b1=new box();(or)
box b2; b2=new box();
```

ACCESSING CLASS MEMBERS:

- ✓ Accessing the class members means accessing instance variable and instance methods in a class.
- ✓ To access these members, a dot (.) operator is used along with the objects.

Syntax for accessing the instance members and methods:

```
object_name.variable_name;
object_name.method_name(parameter_list);
```

Example:

```
class box
{
    double width;
    double height;
```

```
        double depth;
void volume()
{
        System.out.print("\n Box Volume is : ");
        System.out.println(width*height*depth+" cu.cms");
}
}
public class BoxVolume
{
public static void main(String[] args)
{
     box b1=new box(); // creating object of type box
     b1.width=10.00;      // Accessing instance variables through object
     b1.height=10.00;
     b1.depth=10.00;
     b1.volume();          // Accessing method through object
}
}
```

Output:

Box Volume is: 1000.0 cu.cms

METHODS

DEFINITION :

A Java method is a collection of statements that are grouped together to perform an operation.

Syntax: Method:

```
modifier Return –type method_name(parameter_list) throws exception_list
   {
// method body
    }
```

The syntax shown above includes:

- **modifier:** It defines the access type of the method and it is optional to use.
- **returnType:** Method may return a value.
- **Method_name:** This is the method name. The method signature consists of the methodname 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.

Example:

This method takes two parameters num1 and num2 and returns the maximum between the two:

```
        /** the snippet returns the minimum between two numbers */
        public static int minFunction(int n1, int n2)
        {
int min;
if (n1 > n2)
    min = n2;
else
    min = n1;
return min;
        }
```

- **METHOD CALLING (Example for Method that takes parameters and returning value):**
  - ✓ For using a method, it should be called.

- ✓ A method may take any no. of arguments.
- ✓ A *parameter* is a variable defined by a method that receives a value when the method is called. For example, in **square( )**, **i** is a parameter.
- ✓ An *argument* is a value that is passed to a method when it is invoked. For example, **square(100)** passes 100 as an argument. Inside **square( )**, the parameter **i** receives that value.
- ✓ There are two ways in which a method is called.
- calling a method that returns a value or
- calling a method returning nothing (no return value).
- ✓ The process of method calling is simple. When a program invokes a method, the programcontrol gets transferred to the called method.
- ✓ This called method then returns control to the caller in two conditions, when:

1. return statement is executed.
2. reaches the method ending closing brace.

- ✓ **Example:**

Following is the example to demonstrate how to define a method and how to call it:

```
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;
}
}
```

This would produce the following result:

```
Minimum value = 6
```

ACCESS SPECIFIERS

Definition:

Access specifiers are used to specify the visibility and accessibility of a class constructors, member variables and methods.

Java classes, fields, constructors and methods can have one of four different accessmodifiers:

1. Public
2. Private
3. Protected
4. Default (package)

1. **Public (anything declared as public can be accessed from anywhere):**
   A variable or method declared/defined with the public modifier can be accessed anywhere in the program through its class objects, through its subclass objects andthrough the objects of classes of other packages also.

2. **Private (anything declared as private can't be seen outside of the class):**
   The instance variable or instance methods declared/initialized as private can be accessedonly by its class. Even its subclass is not able to access the private members.

3. **Protected (anything declared as protected can be accessed by classes in the samepackage and subclasses in the other packages):**
   The protected access specifier makes the instance variables and instance methods visibleto all the classes, subclasses of that package and subclasses of other packages.

4. **Default (can be accessed only by the classes in the same package):**
   The default access modifier is friendly. This is similar to public modifier except only theclasses belonging to a particular package knows the variables and methods.

	PRIVATE	DEFAULT	PROTECTED	PUBLIC
Same class	Yes	Yes	Yes	Yes
Same package Subclass	No	Yes	Yes	Yes
Same package Non-subclass	No	Yes	Yes	Yes
Different package Subclass	No	No	Yes	Yes
Different package Non-subclass	No	No	No	Yes

Example: Illustrating the visibility of access specifiers:

Z:\MyPack\FirstClass.java

```
package MyPack;

public class FirstClass
{
   public String i="I am public variable";
   protected String j="I am protected variable";
```

```
    private String k="I am private variable";
    String r="I dont have any modifier";
}
```

Z:\MyPack2\SecondClass.java

```
package MyPack2;
import MyPack.FirstClass;
class SecondClass extends FirstClass {
    void method()
    {
        System.out.println(i);    // No Error: Will print "I am public variable".
       System.out.println(j);    // No Error: Will print "I am protected variable".
       System.out.println(k); // Error: k has private access in FirstClass
       System.out.println(r);  // Error: r is not public in FirstClass; cannot be accessed
                                 //          from outside package
    }

    public static void main(String arg[])
    {
       SecondClass obj=new SecondClass();
       obj.method();
    }
}
```

Output:

```
I am public variable
I am protected variable

Exception in thread "main" java.lang.RuntimeException: Uncompilable source code - k has private access in MyPack.FirstClass
```

"static" MEMBERS

Static Members are data members (variables) or methods that belong to a static or non-static class rather than to the objects of the class. Hence it is not necessary to create object of that class to invoke static members.

- ✓ The static can be:
    1. variable (also known as class variable)

2. method (also known as class method)
3. block
4. nested class

❖ **Static Variable:**

- ✓ **When a member variable is declared with the static keyword, then it is called static variable and it can be accessed before any objects of its class are created, and without reference to any object.**
- ✓ **Syntax** to declare a static variable:

[access_spefier] static data_type instance_variable;

- ✓ When a static variable is loaded in memory (static pool) it creates only a single copy of static variable and shared among all the objects of the class.
- ✓ A static variable can be accessed outside of its class directly by the class name anddoesn‘t need any object.

Syntax : <class-name>.<variable-name>

Advantages of static variable

- ✓ It makes your program **memory efficient** (i.e., it saves memory).

❖ **Static Method:**

If a method is declared with the static keyword , then it is known as static method.

- ✓ A static method belongs to the class rather than the object of a class.
- ✓ A static method can be invoked without the need for creating an instance of a class.
- ✓ A static method can access static data member and can change the value of it.

o **Syntax: (defining static method)**

```
[access_specifier] static Return_type method_name(parameter_list)
{
  // method body
}
```

o Syntax to access static method:

```
<class-name>.<method-name>
```

✓ The most common example of a **static** member is **main( )**. **main( )** is declared as **static** because it must be called before any objects exist.

▪ Methods declared as **static** have several restrictions:

- ✓ They can only directly call other **static** methods.
- ✓ They can only directly access **static** data.
- ✓ They cannot refer to **this** or **super** in any way.

❖ **Static Block:**

Static block is used to initialize the static data member like constructors helps toinitialize instance members and it gets executed exactly once, when the class is first loaded.

- It is executed before main method at the time of class loading in JVM.
- **Syntax:**

```
class classname
{
   static
   {
      // block of statements
   }
}
```

The following example shows a class that has a **static** method, some **static** variables, and a **static**
initialization block:

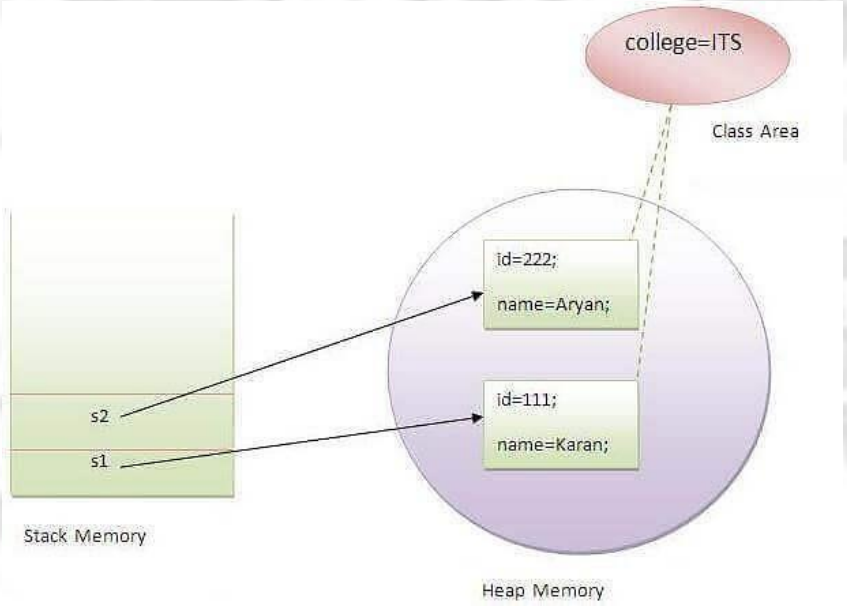
// Demonstrate static variables, methods, and blocks.

```
1.  class Student
2.  {
3.  int rollno;
4.  String name;
5.  static String college = "ITS";
6.  //static method to change the value of static variable
7.  static void change(){
8.  college = "BBDIT";
9.  }
10. //constructor to initialize the variable
11. Student(int r, String n){
12. rollno = r;
13. name = n;
14. }
15. //method to display values
16. void display()
17. {
18. System.out.println(rollno+" "+name+" "+college);
19. }
20. }
21. //Test class to create and display the values of object
22. public class TestStaticMembers
```

```
23.         {
24.         static
25.         {
26.         System.out.println(-*** STATIC MEMBERS – DEMO ***||);
27.         }
28.
29.         public static void main(String args[])
30.         {
31.         Student.change(); //calling change method
32.         //creating objects
33.         Student s1 = new Student(111,"Karan");
34.         Student s2 = new Student(222,"Aryan");
35.         Student s3 = new Student(333,"Sonoo");
36.         //calling display method
37.         s1.display();
38.         s2.display();
39.         s3.display();
40.         }
41.         }
```

Here is the output of this program:

```
*** STATIC MEMBERS – DEMO ***
111         Karan           BBDIT
222         Aryan           BBDIT
333         Sonoo           BBDIT
```



JavaDoc Comments

Definition:

Javadoc is a tool which comes with JDK and it is used for generating Java code documentation in HTML format from Java source code. Java documentation can be created as part of the source code.

Input: Java source files (.java)

- Individual source files
- Root directory of the source files

Output: HTML files documenting specification of java code

- One file for each class defined
- Package and overview files

HOW TO INSERT COMMENTS?

The javadoc utility extracts information for the following items:

- Packages
- Public classes and interfaces
- Public and protected methods
- Public and protected fields

Each comment is placed immediately above the feature it describes.

Format:

- ✓ A Javadoc comment precedes similar to a multi-line comment except that it beginswith a forward slash followed by two asterisks (/**) and ends with a */
- ✓ Each /** . . . */ documentation comment contains free-form text followed by tags.
- ✓ A tag starts with an @, such as @author or @param.
- ✓ The first sentence of the free-form text should be a summary statement.
- ✓ The javadoc utility automatically generates summary pages that extract thesesentences.
- ✓ In the free-form text, you can use HTML modifiers such as ... foremphasis, <code>...</code> for a monospaced —typewriter font, ... for strong emphasis, and even <img ...> to include an image.
- ✓ Example:

```
/**
This is a <b>doc</b> comment.
*/
```

TYPES OF COMMENTS:

1. ***Class Comments***

The class comment must be placed *after* any import statements, directly before the classdefinition.

Example:

```
import java.io.*;
/** class comments should be written here */Public class sample
{
....
}
```

2. ***Method Comments***

The method comments must be placed immediately before the method that it describes.

Tags used:

Tag	Description	Syntax
@param	It describes the method parameter	**@param name description**
@return	This tag describes the return value from a method with the exception void methods and constructors.	**@return description**
@throws	This tag describes the method that throwsan exception.	**@throws class description**

Example:

```
/** adding two numbers
@param a & b are two numbers to be added
@return the result of addition
**/
public double add(int a,int b)
{
int c=a+b;
return c;
}
```

3. ***Field Comments***

Field comments are used to document public fields—generally that means static constants.

For example:

```
/**
* Account number
*/
public static final int acc_no = 101;
```

4. ***General Comments***

Tag	Description	Syntax
The following tags can be used in class documentation comments		
@author	This tag makes an —author entry. You can have multiple @author tags, one for each author.	**@author name**
@version	This tag makes a —version‖ entry. The text can be any description of the current version.	**@version text**
The following tags can be used in all documentation comments		
@since	This tag makes a —since‖ entry. The text can be any description of the version that introduced this feature. For example, @since version 1.7.1	**@since text**
@deprecated	This tag adds a comment that the class, method, or variable should no longer be used. The text should suggest a replacement. For example: @deprecated Use <code>setVisible(true)</code>instead	**@deprecated text**
Hyperlinks to other relevant parts of the javadoc documentation, or to external documents,with the @see and @link tags.		
@link	This tag place hyperlinks to other classes or methods anywhere in any of your documentationcomments.	**{@link package.class#feature label}**

@see	This tag adds a hyperlink in the —see also section. It can be used with both classes and methods. Here, reference can be one of the following: package.class#feature label label "text" Example: @see -Core java 2 @see Core Java	**@see reference**

COMMENT EXTRACTION

Here, *docDirectory* is the name of the directory where you want the HTML files to go. Follow these steps:

1. Change to the directory that contains the source files you want to document.
2. To create the document API, you need to use the javadoc tool followed by java file name.There is no need to compile the javafile.

Here, docDirectory is the name of the directory where you want the HTML files to go.

Follow these steps:

1. Change to the directory that contains the source files you want to document.
2. Run the command

javadoc -d docDirectory nameOfPackage

for a single package. Or run

javadoc -d docDirectory nameOfPackage1 nameOfPackage2...

to document multiple packages.

If your files are in the default package, then instead run

javadoc -d docDirectory *.java

If you omit the -d docDirectory option, then the HTML files are extracted to the currentdirectory.

Example:

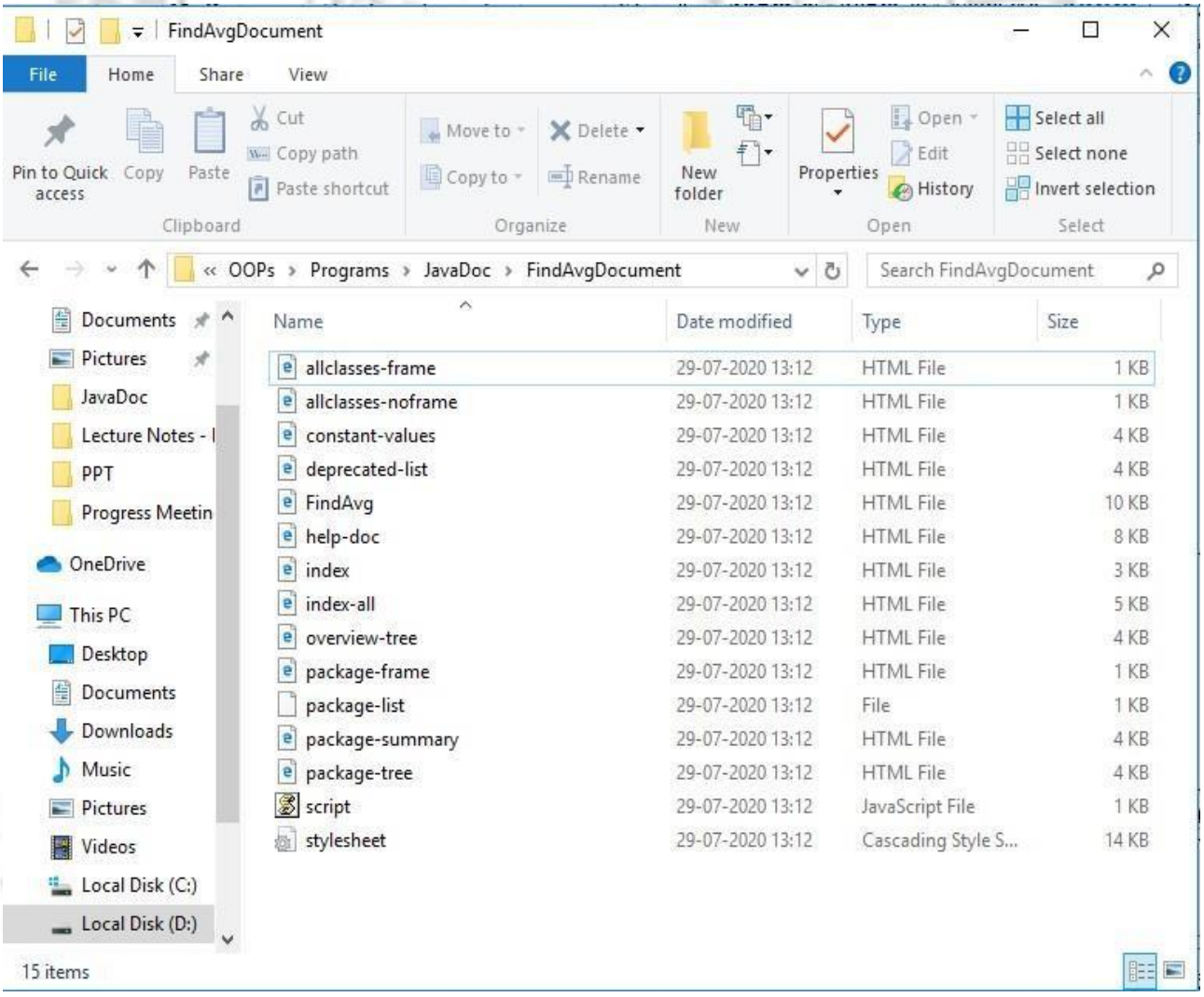
OUTPUT:

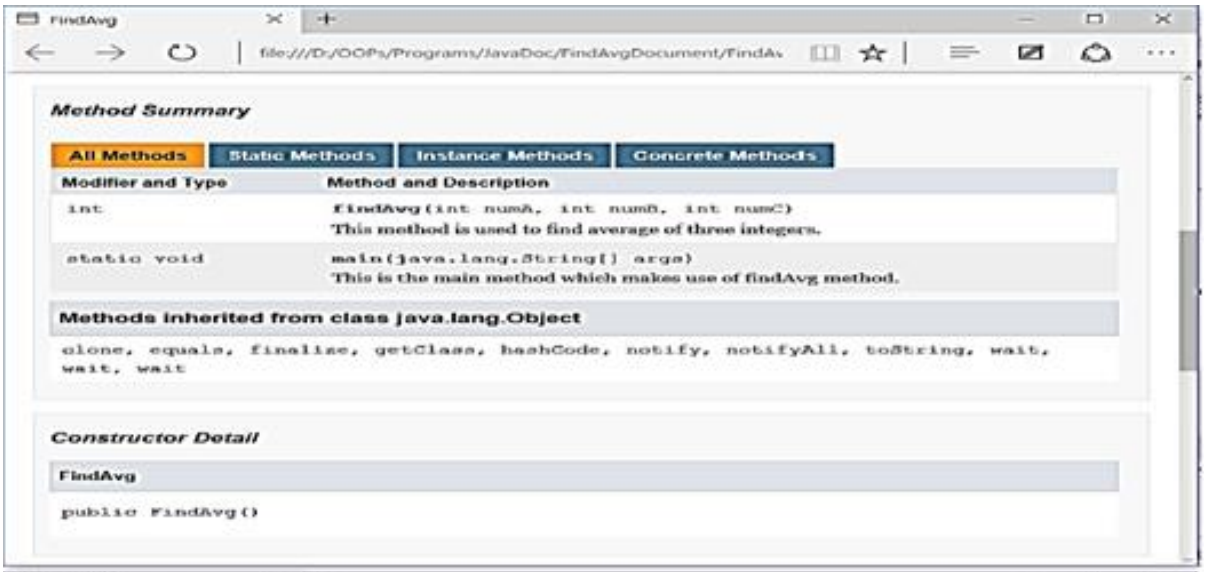
```
//Java program to illustrate frequently used
// Comment tags

/**
* <h1>Find average of three numbers!</h1>
* The FindAvg program implements an application that
* simply calculates average of three integers and Prints
* the output on the screen.
*
* @author  Pratik Agarwal
* @version 1.0
* @since  2017-02-18
*/
public class FindAvg
{
   /**
   * This method is used to find average of three integers.
   * @param numA This is the first parameter to findAvg method
   * @param numB  This is the second parameter to findAvg method
   * @param numC  This is the third parameter to findAvg method
   * @return int This returns average of numA, numB and numC.
   */
   public int findAvg(int numA, int numB, int numC)
   {
      return (numA + numB + numC)/3;
   }

   /**
   * This is the main method which makes use of findAvg method.
   * @param args Unused.
   * @return Nothing.
   */

   public static void main(String args[])
   {
      FindAvg obj = new FindAvg();
      int avg = obj.findAvg(10, 20, 30);

      System.out.println("Average of 10, 20 and 30 is :" + avg);
   }
}
```

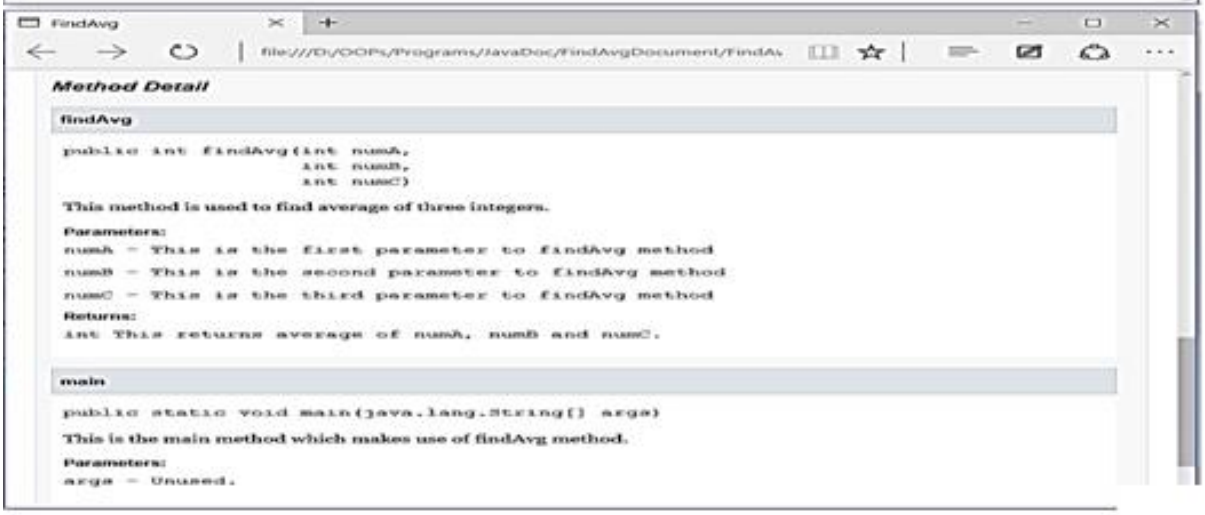
D:\OOPs\Programs\JavaDoc>javadoc -d FindAvgDocument FindAvg.java

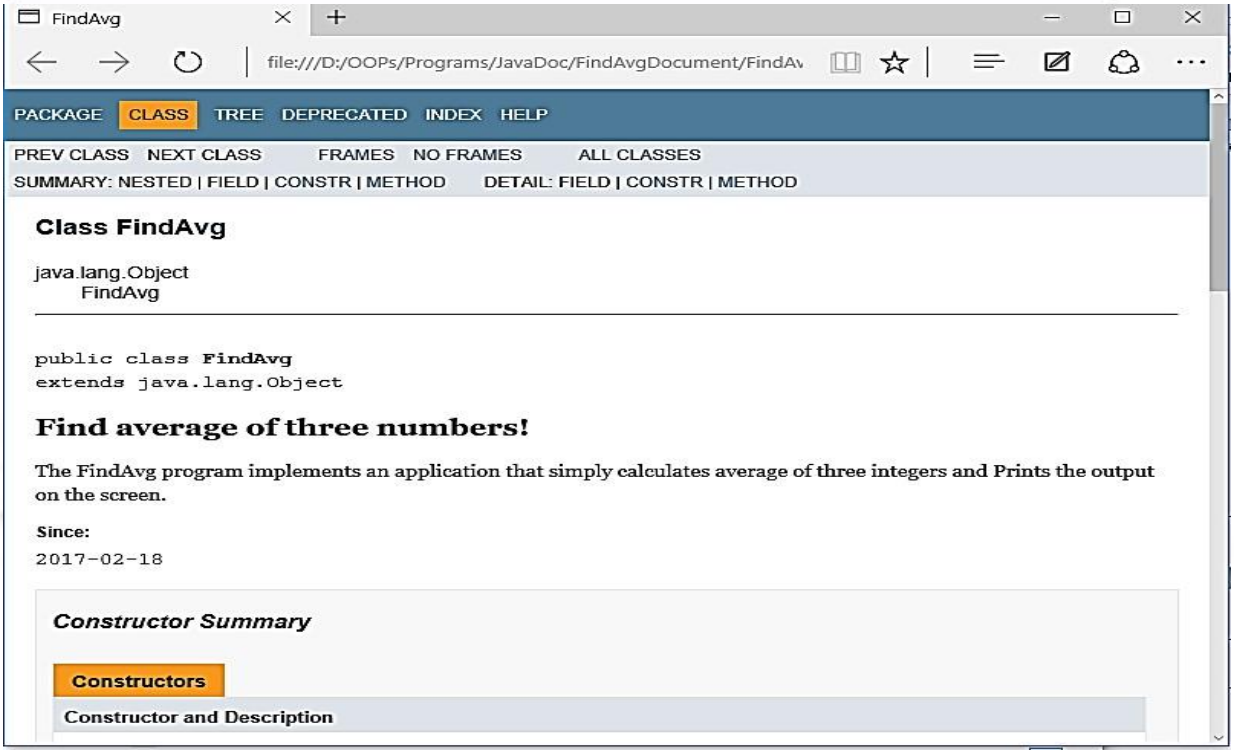
Loading source file FindAvg.java...
Constructing Javadoc information...
Creating destination directory: "FindAvgDocument\"
Standard Doclet version 1.8.0_251

```
Building tree for all the packages and classes...
Generating     FindAvgDocument\FindAvg.html...
FindAvg.java:32: error: invalid use of @return
    * @return Nothing.
      ^
Generating FindAvgDocument\package-frame.html...
Generating FindAvgDocument\package-summary.html...
Generating FindAvgDocument\package-tree.html...
Generating FindAvgDocument\constant-values.html...
Building index for all the packages and classes...
Generating FindAvgDocument\overview-tree.html...
Generating FindAvgDocument\index-all.html...
Generating FindAvgDocument\deprecated-list.html...
Building index for all classes...
Generating FindAvgDocument\allclasses-frame.html...
Generating       FindAvgDocument\allclasses-noframe.html...
Generating FindAvgDocument\index.html...
Generating     FindAvgDocument\help-doc.html...
1 error
```




FindAvg
file:///D:/OOPs/Programs/JavaDoc/FindAvgDocument/FindA
Method Summary
All Methods
Static Methods
Instance Methods
Concrete Methods
Modifier and Type
Method and Description
int
findAvg(int numA, int numB, int numC)
This method is used to find average of three integers.
static void
main(java.lang.String[] args)
This is the main method which makes use of findAvg method.
Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait
Constructor Detail
FindAvg
public FindAvg()


FindAvg
file:///D:/OOPs/Programs/JavaDoc/FindAvgDocument/FindA
Method Detail
findAvg
public int findAvg(int numA,
int numB,
int numC)
This method is used to find average of three integers.
Parameters:
numA - This is the first parameter to findAvg method
numB - This is the second parameter to findAvg method
numC - This is the third parameter to findAvg method
Returns:
int This returns average of numA, numB and numC.
main
public static void main(java.lang.String[] args)
This is the main method which makes use of findAvg method.
Parameters:
args - Unused.


FindAvg
file:///D:/OOPs/Programs/JavaDoc/FindAvgDocument/FindA
PACKAGE
CLASS
TREE
DEPRECATED
INDEX
HELP
PREV CLASS
NEXT CLASS
FRAMES
NO FRAMES
ALL CLASSES
SUMMARY: NESTED | FIELD | CONSTR | METHOD
DETAIL: FIELD | CONSTR | METHOD
Class FindAvg
java.lang.Object
FindAvg
public class FindAvg
extends java.lang.Object
Find average of three numbers!
The FindAvg program implements an application that simply calculates average of three integers and Prints the output on the screen.
Since:
2017-02-18
Constructor Summary
Constructors
Constructor and Description

Additional Topics
Comments, Literals, Keywords, Type Conversion, Garbage Collection, Command Line Arguments

JAVA – COMMENTS

- Java comments are either explanations of the source code or descriptions of classes, interfaces, methods, and fields.
- They are usually a couple of lines written above or beside Java code to clarify what it does.
- Comments in Java do not show up in the executable program.
- The Java language supports three kinds of comments:

1. Line comment:

- ✓ When you want to make a one line comment type "//" and follow the two forward slasheswith your comment.
- ✓ **Syntax:** // text
- ✓ **Example:** // this is a single line comment
- ✓ The compiler ignores everything from // to the end of the line.

2. Block Comment:

- ✓ To start a block comment type "/*". Everything between the forward slash and asterisk, even if it's on a different line, will be treated as comment until the characters "*/" end the comment.
- ✓ **Syntax:** /* *text* */
- ✓ **Example:** /* it is a comment */ (or)

```
/* thisis a block
comment
*/
```

- ✓ The compiler ignores everything from /* to */.

3. Documentation Comment:

- ✓ This type of comment helps is generating the documentation automatically.
- ✓ **Syntax:** /** *documentation* */

The JDK javadoc tool uses doc comments when preparing automatically generated documentation. For more information on javadoc, see the Java tool documentation.

- ✓ **Example:**

```
/*
* Title: Conversion of Degrees
* Aim: To convert Celsius to Fahrenheit and vice versa
* Date: 31/08/2000
* Author: Tim
*/
```

: JAVA - CONSTANTS

- ✓ A constant is an identifier written in uppercase (convention and not a rule) that prevents
  its contents form being modified by the program during the execution.
- ✓ If an attempt is made to change the value, the compiler will give an error message.
- ✓ In Java, the keyword **final** is used to declare constants.
- ✓ The value of a final variable cannot change after it has been initialized.

- ✓ **Syntax:**

```
final datatype variablename=value;
```

- ✓ **Example:**final float PI=3.14f;

: JAVA - IDENTIFIERS

- ✓ Identifiers are names given to the variables, classes, methods, objects, labels, package and interface in our program.
- ✓ The name we are giving must be meaningful and it may have random length.
- ✓ The following rule must be followed while giving a name:
  1. The first character must not begin with a number.
  2. The identifier is formed with alphabets, number, dollar sign ($) and underscore (_).
  3. It should not be a reserved word.
  4. Space is not allowed in between the identifier name.

- ✓ **Example:**

```
String name = "Homer Jay Simpson";
int weight = 300;
double height = 6;
```

: JAVA – RESERVED WORDS (KEYWORDS)

- ✓ There are some words that you cannot use as object or variable names in a Java program. These words are known as reserved words; they are keywords that are already used by the syntax of the Java programming language.
- ✓ For example, if you try and create a new class and name it using a reserved word:

```
// you can't use finally as it's a reserved word!
        class finally {
        public static void main(String[] args)
        {
```

```
//class code..
}
}
```

- ✓ It will not compile, instead you will get the following error: <identifier> expected
- ✓ The table below lists all the words that are reserved:

abstract	**Assert**	**boolean**	**Break**	**byte**	**case**
catch	**Char**	**class**	const*	continue	default
double	Do	else	Enum	extends	false
final	**Finally**	**float**	**For**	goto*	**if**
implements	**Import**	instanceof	**Int**	**interface**	long
native	**New**	null	**package**	private	protected
public	Return	short	**Static**	strictfp	**super**
switch	Synchronized	**this**	**Throw**	**throws**	transient
True	**Try**	**void**	volatile	while	

TYPE CONVERSIONS AND CASTING

Type Conversion is the task of converting one data type into another data type.

Two types of type conversion:

1. Implicit Type Conversion (or) Automatic Conversion
2. Explicit Type Conversion (or) Casitng

1. Implicit Type Conversion (or) Automatic Conversion:

If the two types are compatible, then Java will perform the conversion automatically. When one type of data is assigned to another type of variable, an ***Automatic type conversion (or) WideningConversion*** will take place if the following two conditions are met:

- Two types are compatible
- The destination type is larger than the source type

.

Example:

```
byte a=100;
int b=a;        // b is larger than a

long d=b;       // d is large than b
float e=b;      // e is larger than b

float sum=10;
int s=sum;      // s is smaller than sum, So we need to go for explicit conversion.
```

1. Explicit Type Conversion (or) Casting:

If the two types are compatible, a forced conversion of one type into another type is performed This forced conversion is called as Explicit Type Conversion. **Casting (or) narrowing conversion** is an operation which performs an explicit conversion between incompatible types.

Example: converting int to byte.

Syntax to perform "Cast":

(target-type) value;

Here,

Target-type = specifies the desired type to convert the specified value.

Example:

```
class conversion {
public static void main(String arg[])
{
byte  b;
int i=257;
double d=323.142;

System.out.println("\nConversion of int to byte: ");
b=(byte) i;
System.out.println("i and b : "+i+" , "+b);

System.out.println("\nConversion of double to int: ");
i=(int) d;
System.out.println("d and i : "+d+" , "+i);

System.out.println("\nConversion of double to byte: ");
b=(byte) d;
```

```
System.out.println("d and b : "+d+" , "+b);

        // Automatic Type promotions in expressions
byte   r=40;
byte   s=50;
byte t=100;
int p=r * s / t;          // r*s exceeds the range of byte, so automatic type promotion take place.
System.out.println("Value of P = "+p);
s=s*2;          //Error! cannot assign int to a byte.
s=(byte)(s*2);          // Possible.
}
}
```

Output:

```
Conversion of int to byte:
i and b : 257 , 1
Conversion of int to byte:
d and i : 323.142 , 323
Conversion of int to byte:
d and b : 323.142 , 67
Value of P = 20
```

Type Promotions rules:

1. All **byte, short** and **char** values are promoted to int.
2. If one operand is **long,** the whole expression is promoted to **long.**
3. If one operand is **float,** the whole expression is promoted to **float.**
4. If any of the operand is **double,** the result is **double.**

GARBAGE COLLECTION

- ✓ Since objects are dynamically allocated by using the **new** operator, you might be wondering how such objects are destroyed and their memory released for later reallocation.
- ✓ In some languages, such as C++, dynamically allocated objects must be manually released by use of a **delete** operator.
- ✓ Java takes a different approach;

Automatic Garbage Collection: The technique that accomplishes automatic deallocation of memory occupied by an unused object is called *garbage collection*.

It works like this:

- When no references to an object exist, that object is assumed to be no longer needed, and the memory occupied by the object can be reclaimed. There is no explicit need to destroy objects as in C++.
- Garbage collection only occurs sporadically (if at all) during the execution of your program.

➢ **Finalization:**

- ✓ Sometimes an object will need to perform some action when it is destroyed. For example, if an object is holding some non-Java resource such as a file handle or character font, then you might want to make sure these resources are freed before an object is destroyed.
- ✓ To handle such situations, Java provides a mechanism called *finalization*. By using finalization, you can define specific actions that will occur when an object is just about tobe reclaimed by the garbage collector.

➢ **Finalize() method:**

A **finalize()** method is a method that will be called by the garbage collector on an object when garbage collection determines that there are no more references to the object.

Inside the **finalize( )** method, we will specify those actions that must be performed before anobject is destroyed.

The finalize( ) method has this general form:

```
protected void finalize( )
{
// finalization code here
}
```

Here, the keyword protected is a specifier that prevents access to finalize( ) by code definedoutside its class.

Example:

```
public class TestGarbage1
{
public void finalize()
{
```

```
 System.out.println("object is garbage collected");
 }
public static void main(String args[])
{
TestGarbage1 s1=new TestGarbage1();
TestGarbage1 s2=new TestGarbage1();
s1=null;
 s2=null;
 System.gc();
 }
 }
```

Output:

object is garbage collectedobject is garbage collected

USING COMMAND LINE ARGUMENTS:

- ✓ Sometimes you will want to pass information into a program when you run it. This is accomplished by passing *command-line arguments* to **main( )**.

- ✓ **A command-line argument is the information passed to the main() method that directly follows the program's name on the command line when it is executed.**

- ✓ To access the command-line arguments inside a Java program is quite easy—they are stored as strings in a **String** array passed to the **args** parameter of **main( )**.
- ✓ The first command-line argument is stored at **args[0]**, the second at **args[1]**, and so on.
- ✓ For example, the following program displays all of the command-line arguments that it iscalled with:

```
// Display all command-line arguments.

class CommandLine
{
public static void main(String args[])
{
for(int i=0; i<args.length; i++)
  System.out.println("args[" + i + "]: " + args[i]);
}
}
```

Try executing this program, as shown here:

>java CommandLine this is a test 100 -1

When you do, you will see the following output:

```
args[0]: this
args[1]: is
args[2]: a
args[3]: test
args[4]: 100
args[5]: -1
```


KOHINI COLLEGE OF ENGINEERING AND TECHNOLOGY