SRI AKILANDESWARI WOMEN’S COLLEGE, WANDIWASH

INTRODUCTION TO JAVA PROGRAMMING

Class: II. B. Sc Computer Science

Prepared by
C. BALASUBRAMANIAN,
Assistant Professor, Dept of Computer Science

SWAMY ABEDHANADHA EDUCATIONAL TRUST, WANDIWASH

Course Objectives

- On completing the course, you will understand
  - Create, compile, and run Java programs
  - Primitive data types
  - Java control flow
  - Methods
  - Arrays (for teaching Java in two semesters, this could be the end)
  - Object-oriented programming
  - Core Java classes (Swing, exception, internationalization, multithreading, multimedia, I/O, networking, Java Collections Framework)

Course Objectives

- Write simple programs using primitive data types, control statements, methods, and arrays.
- Create and use methods
- Develop a GUI interface and Java applets
- Write interesting programs
- Establish a firm foundation on Java concepts

Java programming

Fundamentals of Programming

- Introduction to Java
- Primitive Data Types and Operations
- Control Statements
- Methods
- Arrays

Java programming

- Object-Oriented Programming
  - Objects and Classes
  - Strings
  - Class Inheritance and Interfaces

Java programming

- GUI Programming
  - Getting Started with GUI Programming
  - Creating User Interfaces
  - Applets and Advanced GUI

Java programming

Developing Comprehensive Projects

- Exception Handling
- Internationalization
- Multithreading
- Multimedia
- Input and Output
- Networking
- Java Data Structures

Introduction to Java

- What Is Java?
- Getting Started With Java Programming
  - Create, Compile and Running a Java Application

What Is Java?

- History
- Characteristics of Java

History

- James Gosling and Sun Microsystems
- Oak
- Java, May 20, 1995, Sun World
- HotJava
  - The first Java-enabled Web browser
- JDK Evolutions
- J2SE, J2ME, and J2EE

Characteristics of Java

- Java is simple
- Java is object-oriented
- Java is distributed
- Java is interpreted
- Java is robust
- Java is secure
- Java is architecture-neutral
- Java is portable
- Java's performance
- Java is multithreaded
- Java is dynamic

JDK Versions

- JDK 1.02 (1995)
- JDK 1.1 (1996)
- Java 2 SDK v 1.2 (a.k.a JDK 1.2, 1998)
- Java 2 SDK v 1.3 (a.k.a JDK 1.3, 2000)
- Java 2 SDK v 1.4 (a.k.a JDK 1.4, 2002)

JDK Editions

- Java Standard Edition (J2SE)
  - J2SE can be used to develop client-side standalone applications or applets.
- Java Enterprise Edition (J2EE)
  - J2EE can be used to develop server-side applications such as Java servlets and Java ServerPages.
- Java Micro Edition (J2ME).
  - J2ME can be used to develop applications for mobile devices such as cell phones.

Getting Started with Java Programming

- A Simple Java Application
- Compiling Programs
- Executing Applications

Keying a java program

- Always type the java programs in notepad or any other editors
- Since java is case sensitive, type in upper / lower case letters as prescribed by jdk
- After completion of keying the java program, save your program with extension .java
- Without the .java as extension, the java program will not be compiled.

A Simple java Program

Example 1.1

```
//This program prints "A TRIAL JAVA PROGRAM"


public class trial{
  public static void main(String[] args) {
    System.out.println(" A TRIAL JAVA
  PROGRAM");
  }
}
```

Compiling and Executing Programs

- On command line
- To compile a java program, type

```
javac filename.java
```

- To execute a java program, type

```
java filename
```

Example

```
javac trial.java


java trial


output:...
 A TRIAL JAVA PROGRAM
```

contents of a Java Program

- Comments
- Package
- Reserved words
- Modifiers
- Statements
- Blocks
- Classes
- Methods
- The main method

Comments

In Java, comments are preceded by two slashes (//) in a line, or enclosed between /* and */ in one or multiple lines. When the compiler sees //, it ignores all text after // in the same line. When it sees /*, it scans for the next */ and ignores any text between /* and */

Reserved Words

Reserved words or *keywords* are words that have a specific meaning to the compiler and cannot be used for other purposes in the program. For example, when the compiler sees the word class, it understands that the word after class is the name for the class. Other reserved words are public, static and void.

Modifiers

Java uses certain reserved words called *modifiers* that specify the properties of the data, methods, and classes and how they can be used. Examples of modifiers are <u>public</u> and <u>static</u>. Other modifiers are <u>private</u>, <u>final</u>, <u>abstract</u>, and <u>protected</u>. A <u>public</u> datum, method, or class can be accessed by other programs. A <u>private</u> datum or method cannot be accessed by other programs.

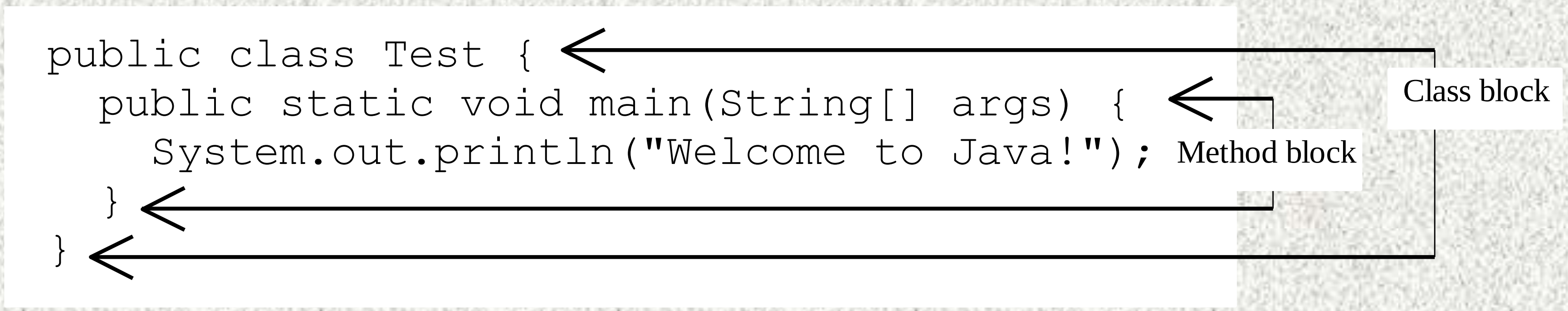
Statements

A *statement* represents an action or a sequence of actions. The statement System.out.println(" A TRIAL JAVA PROGRAM") in the program is a statement to display the greeting " A TRIAL JAVA PROGRAM"

Every statement in Java ends with a semicolon (;).

Blocks

A pair of braces in a program forms a block that groups components of a program.



```
public class Test {
  public static void main(String[] args) {
    System.out.println("Welcome to Java!");
  }
}
```

Class

The *class* is the essential Java construct. A class is a template or blueprint for objects.

To program in Java, you must understand classes and be able to write and use them.

Methods

What is System.out.println? It is a *method*: a collection of statements that performs a sequence of operations to display a message on the console. It can be used even without fully understanding the details of how it works. It is used by invoking a statement with a string argument. The string argument is enclosed within parentheses. In this case, the argument is "`A TRIAL JAVA PROGRAM`" You can call the same println method with a different argument to print a different message.

main Method

The main method provides the control of program flow. The Java interpreter executes the application by invoking the main method.

The main method looks like this:

```
public static void main(String[]
args) {
   // Statements;
}
```

The exit Method

Use Exit to terminate the program and stop all threads.

NOTE: When your program starts, a thread is spawned to run the program. When the showMessageDialog is invoked, a separate thread is spawned to run this method. The thread is not terminated even if you close the dialog box. To terminate the thread, you have to invoke the exit method.

Data types in java

primitive and non primitive data types:

Primitive data types [predefined in compiler]

int

long

float

double

char

boolean

Int – an example

- **int**: Used to declare a numeric quantity without a decimal point.

  Default size: 4 byte,   Default value: 0

  Example:

```
 class JavaExample
      { public static void main(String[] args)
         {
                 int num; num = 150;
                 System.out.println(num);
          }
       }
```

Output:

- 150

Floating point Literals

- Floating-point Literals are also called as real constants. The Floating Point contains decimal points and can contain exponents. They are used to represent values that will have a fractional part and can be represented in two forms – fractional form and exponent form.

Float - example

- **float**: Sufficient for holding 6 to 7 decimal digits   size: 4 bytes

```
 class JavaExample
{
     public static void main(String[] args)
            {
               float num = 19.98;
               System.out.println(num);
             }
 }
```

Output:

- 19.98

Fractional and exponent form

- in the fractional form, the number contains integer and fractional part. A dot (.) is used to separate integer part and fractional part.
- **Example:** float x = 2.7;
- In the exponential form, the fractional number contains constants a mantissa and exponent.
- **Example:** 23.46e3 = 23.46 x 10^3

Double - example

- **double**: Sufficient for holding 15 decimal digits size: 8 bytes

  Example:

```
class JavaExample
 {
     public static void main(String[] args)
        {     double num = -42937737.9 ;
                  System.out.println(num);
         }
  }
```

Output:

```
-4.29377379E7
```

Character Literals

- Character Literals are specified as single character enclosed in pair of single quotation marks.
- Example
- char a = ‘good morning’;

Char - example

- **char**: holds characters.
  size: 2 bytes

```
class JavaExample
    {
        public static void main(String[] args)
       {
          char ch = 'Z';
          System.out.println(ch);
        }
    }
```

Output:

```
Z
```

String Literals

- String Literals are treated as an array of char. By default, the compiler adds a special character called the ‘null character’ (‘\0’) at the end of the string to mark the end of the string.
- **Example:**
- String str = “good morning”;

Boolean literals

- There are two Boolean literals
- true
- false

Boolean - example

- **boolean**: holds value either true of false.

```
class JavaExample
{
     public static void main(String[] args)
       {
            boolean b = false;
            System.out.println(b);
       }
   }
```

Output:

```
false
```

Non-primitive data types

- There are four types of non-primitive data:
- **array**: It can store any type of data
- **string**: used to store consecutive characters
- **class**: Class is used to create objects. It may have different pieces of data into a single object.
- **interface**: An interface is like a dashboard or control panel for a class.

Operator in Java

1) Basic Arithmetic Operators
2) Assignment Operators
3) Increment and decrement Operators
4) Logical Operators
5) Comparison (relational) operators
6) Bitwise Operators
7) Ternary Operator

Arithmetic Operators

Basic arithmetic operators are: +, -, *, /, %

+	is for addition.
–	is for subtraction.
*****	is for multiplication.
/	is for division.
%	is for modulo.

```
public class ArithmeticOperatorDemo
{ public static void main(String args[])
   { int num1 = 100;      int num2 = 20;
     System.out.println("num1 + num2: " + (num1 + num2) );
    System.out.println("num1 - num2: " + (num1 - num2) );
    System.out.println("num1 * num2: " + (num1 * num2) );
    System.out.println("num1 / num2: " + (num1 / num2) );
    System.out.println("num1 % num2: " + (num1 % num2) );
  }
 }
```

Output:

```
num1 + num2: 120
num1 - num2: 80
num1 * num2: 2000
num1 / num2: 5
num1 % num2: 0
```

Assignment Operators

- Assignments operators in java are:

  =, +=, -=, *=, /=, %=

num2 = num1

num2+=num1 is equal to num2 = num2+num1

num2-=num1 is equal to num2 = num2-num1

num2*=num1 is equal to num2 = num2*num1

num2/=num1 is equal to num2 = num2/num1

num2%=num1 is equal to num2 = num2%num1

```
public class AssignmentOperatorDemo
{
  public static void main(String args[])
{ int num1  =  10; int num2 = 20;
num2 = num1;
     System.out.println("= Output: "+num2); num2 += num1;
   System.out.println("+= Output: "+num2); num2 -= num1;
   System.out.println("-= Output: "+num2); num2 *= num1;
   System.out.println("*= Output: "+num2); num2 /= num1;
   System.out.println("/= Output: "+num2); num2 %= num1;
   System.out.println("%= Output: "+num2);
 }
}
```

Output:

```
   = Output: 10          += Output: 20                    -= Output: 10
              *= Output: 100    /= Output: 10             %= Output: 0
```

increment and decrement Operators

- ++ and —

  num++ is equivalent to num=num+1;

  num-- is equivalent to num=num-1;

```
public class AutoOperatorDemo
 { public static void main(String args[])
   {     int num1=100;
         int num2=200;
         num1++; num2--;
         System.out.println("num1++ is: "+num1);
         System.out.println("num2-- is: "+num2);
    }
 }
```

Output:

```
num1++ is: 101                         num2-- is: 199
```

Logical Operators

- Logical Operators are used with binary variables. They are mainly used in conditional statements and loops for evaluating a condition.
- Logical operators in java are:
- && -> and
- || -> or
- ! -> not

```
public class LogicalOperatorDemo
 {
   public static void main(String args[])
   {
          boolean b1 = true;
          boolean b2 = false;
          System.out.println("b1 && b2: " + (b1&&b2));
          System.out.println("b1 ||b2: " + (b1||b2));
          System.out.println("!(b1 && b2): " + !(b1&&b2));
   }
 }
```

- **Output:**

```
b1 && b2: false
b1 || b2: true
!(b1 && b2): true
```

Relational operators

- We have six relational operators in Java:

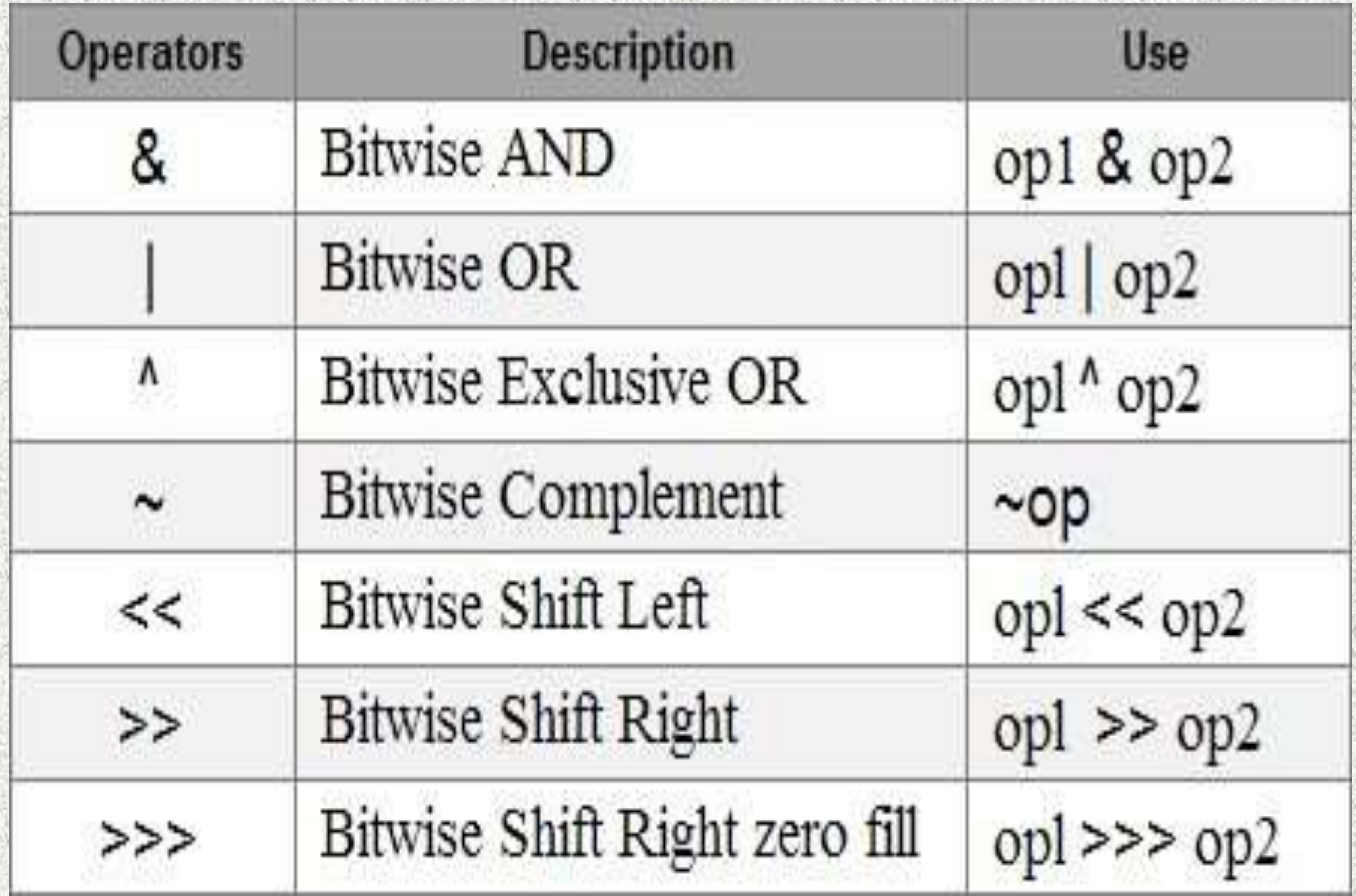
==	Equal to
!=	Not equal to
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to

```
public class RelationalOperatorDemo
{ public static void main(String args[])
{ int num1 = 10;
int num2 = 50;
if (num1==num2)
{ System.out.println("num1 and num2 are equal");
} else
{ System.out.println("num1 and num2 are not equal");}
}
}
```

- **Output:**

```
num1 and num2 are not equal
```

Bitwise operators



Operators	Description	Use
&	Bitwise AND	op1 & op2
|	Bitwise OR	opl | op2
^	Bitwise Exclusive OR	opl ^ op2
~	Bitwise Complement	~op
<<	Bitwise Shift Left	opl << op2
>>	Bitwise Shift Right	opl >> op2
>>>	Bitwise Shift Right zero fill	opl >>> op2

Ternary Operator

- This operator evaluates a boolean expression and assign the value based on the result.
  Syntax:
- variable num1 = (expression) ? value if true : value if false
- If the expression results true then the first value before the colon (:) is assigned to the variable num1 else the second value is assigned to the num1.

```
public class TernaryOperatorDemo
{
   public static void main(String args[])
   {
        int num1, num2; num1 = 25;
        num2 = (num1 == 10) ? 100: 200;
        System.out.println( "num2: "+num2);
        num2 = (num1 == 25) ? 100: 200;
        System.out.println( "num2: "+num2);
    }
}
```

Output:

```
num2: 200
num2: 100
```

Java Control Statements

a) if statement
b) nested if statement
c) if-else statement
d) if-else-if statement

If statement

- If statement consists a condition, followed by statement or a set of statements as shown below:

- if(condition){ Statement(s); }

```
public class IfStatementExample
{ public static void main(String args[])
{ int num=70;
if( num < 100 )
System.out.println("number is less than 100");
 }
}
```

Output:

```
number is less than 100
```

Nested if statement in Java

- When there is an if statement inside another if statement then it is called the **nested if statement**.
  The structure of nested if looks like this:

```
if(condition_1)
{ Statement1(s);
 if(condition_2)
{ Statement2(s);
}
}
```

```
public class NestedIfExample
 { public static void main(String args[])
   { int num=70;
   if( num < 100 )
         { System.out.println("number is less than 100");
         if(num > 50)
            { System.out.println("number is greater than 50");
            }
          }
     }
}
```

Output:

```
number is less than 100
number is greater than 50
```

If else statement in Java

- This is how an if-else statement looks:

```
if(condition)
{ Statement(s); }
 else
{ Statement(s); }
```

```
public class IfElseExample
 { public static void main(String args[])
   { int num=120;
   if( num < 50 )
        { System.out.println("num is less than 50"); }
   else
        { System.out.println("num is greater than or equal 50");
        }
   }
}
```

Output:

```
num is greater than or equal 50
```

if-else-if Statement

- if-else-if statement is used when we need to check multiple conditions. In this statement we have only one "if" and one "else", however we can have multiple "else if". It is also known as **if else if ladder**.

if(condition_1)

{ /*if condition_1 is true execute this*/ statement(s); }

else if(condition_2)

{ /* execute this if condition_1 is not met and * condition_2 is met */ statement(s); }

else if(condition_3) { /* execute this if condition_1 & condition_2 are * not met and condition_3 is met */ statement(s); } . . .

else { /* if none of the condition is true * then these statements gets executed */ statement(s); }

```
public class IfElseIfExample
{ public static void main(String args[])
{ int num=1234;
if(num <100 && num>=1)
{ System.out.println("Its a two digit number"); }
else if(num <1000 && num>=100)
 { System.out.println("Its a three digit number"); }
else if(num <10000 && num>=1000)
{ System.out.println("Its a four digit number"); }
 else if(num <100000 && num>=10000)
{ System.out.println("Its a five digit number"); }
else { System.out.println("number is not between 1 & 99999"); } } }
```

Output:

```
Its a four digit number
```

Switch case statement

- it is used when we have number of choices and we may need to perform a different task for each choice.
- The syntax of Switch case statement

```
switch (variable or an integer expression)
 { case constant: //Java code ;
   case constant: //Java code ;
   default: //Java code ;
  }
```

```
public class SwitchCaseExample1
 { public static void main(String args[])
   { int num=2;
   switch(num+2)
      { case 1: System.out.println("Case1: Value is: "+num);
         case 2: System.out.println("Case2: Value is: "+num);
         case 3: System.out.println("Case3: Value is: "+num);
         default: System.out.println("Default: Value is: "+num);
       }
    }
}
```

Output:

```
Default: Value is: 2
```

Example with break statement

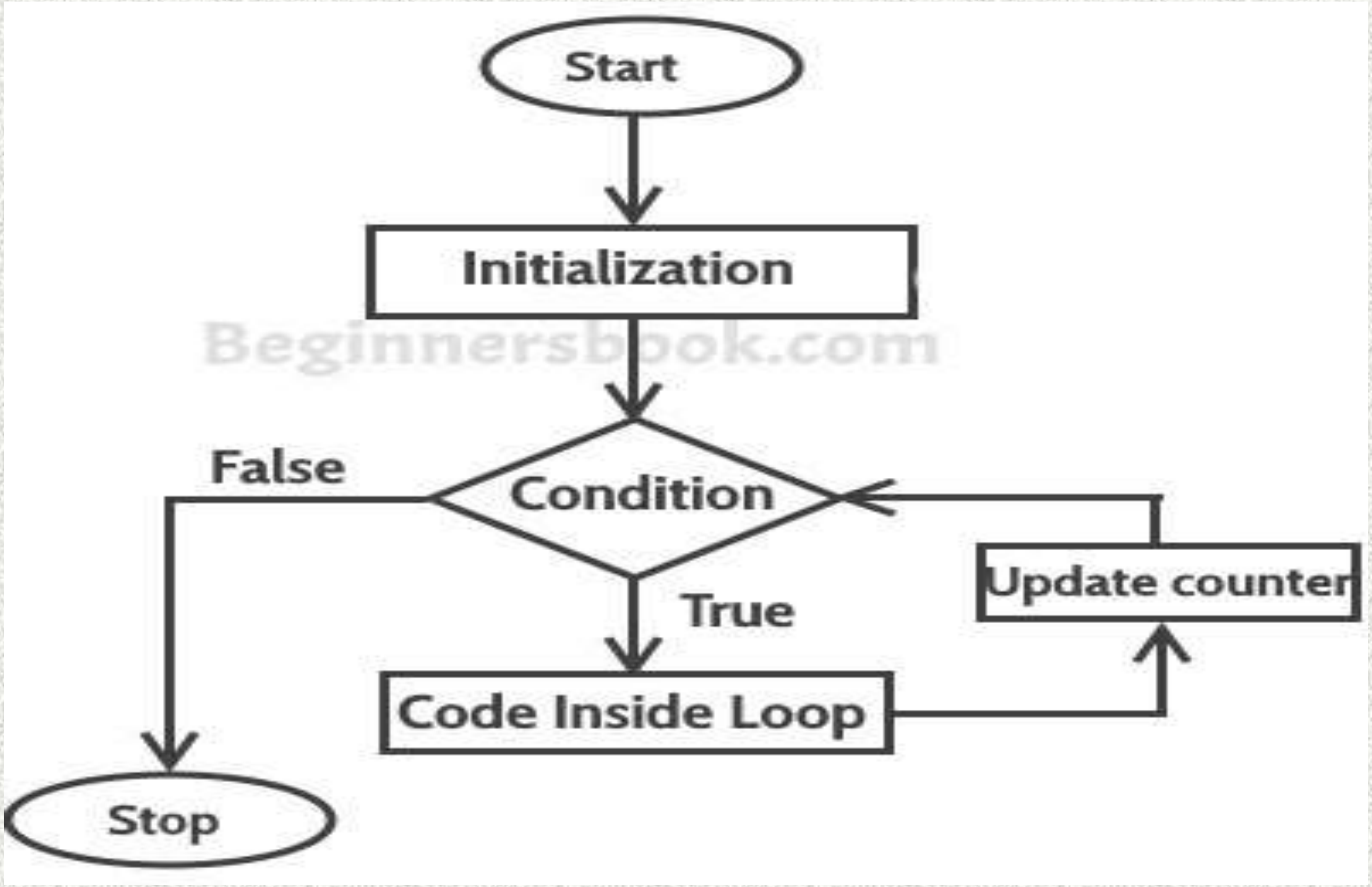
```
public class SwitchCaseExample2
{ public static void main(String args[])
   { int i=2;
   switch(i)
         { case 1: System.out.println("JAVA "); break;
         case 2: System.out.println("C++"); break;
         case 3: System.out.println("DBMS"); break;
         case 4: System.out.println("DAA"); break;
         default: System.out.println("Default ");
          }
      }
 }
```

For loop in Java

Syntax of for loop:

```
for(initial;condition ; incr/decr)
{
        statement(s);
}
```

Flow of Execution of the for Loop



Example of Simple For loop

```
class ForLoopExample
 {
    public static void main(String args[])
     {
         for(int i=10; i>1; i--)
        {
              System.out.println("The value of i is: "+i);   }
    }
}
```

OUTPUT

```
The value of i is: 10
The value of i is: 9
The value of i is: 8
The value of i is: 7
The value of i is: 6
The value of i is: 5
The value of i is: 4
The value of i is: 3
The value of i is: 2
```

Infinite for loop

```
class ForLoopExample2 {
    public static void main(String args[])
{
        for(int i=1; i>=1; i++)
        {
            System.out.println("The value of i is: "+i);   }
    }
}
```

For loop example to iterate an array:

```
class ForLoopExample3
 {
    public static void main(String args[])
{        int arr[]={2,11,45,9};
          for(int i=0; i<arr.length; i++){
          System.out.println(arr[i]);
          }
    }
 }
```

Enhanced For loop

```
class ForLoopExample3 {
   public static void main(String args[]){
      int arr[]={2,11,45,9};
      for (int num : arr) {
         System.out.println(num);
      }
   }
}
```

Output:

```
2
11
45
9
```

NESTED FOR LOOP

- A for loop within another for loop is called nested for loop.
- The inner most for loop will be executed first and the outer loop next.
- Any number of for loops can be nested wherein none of the loops get crossed

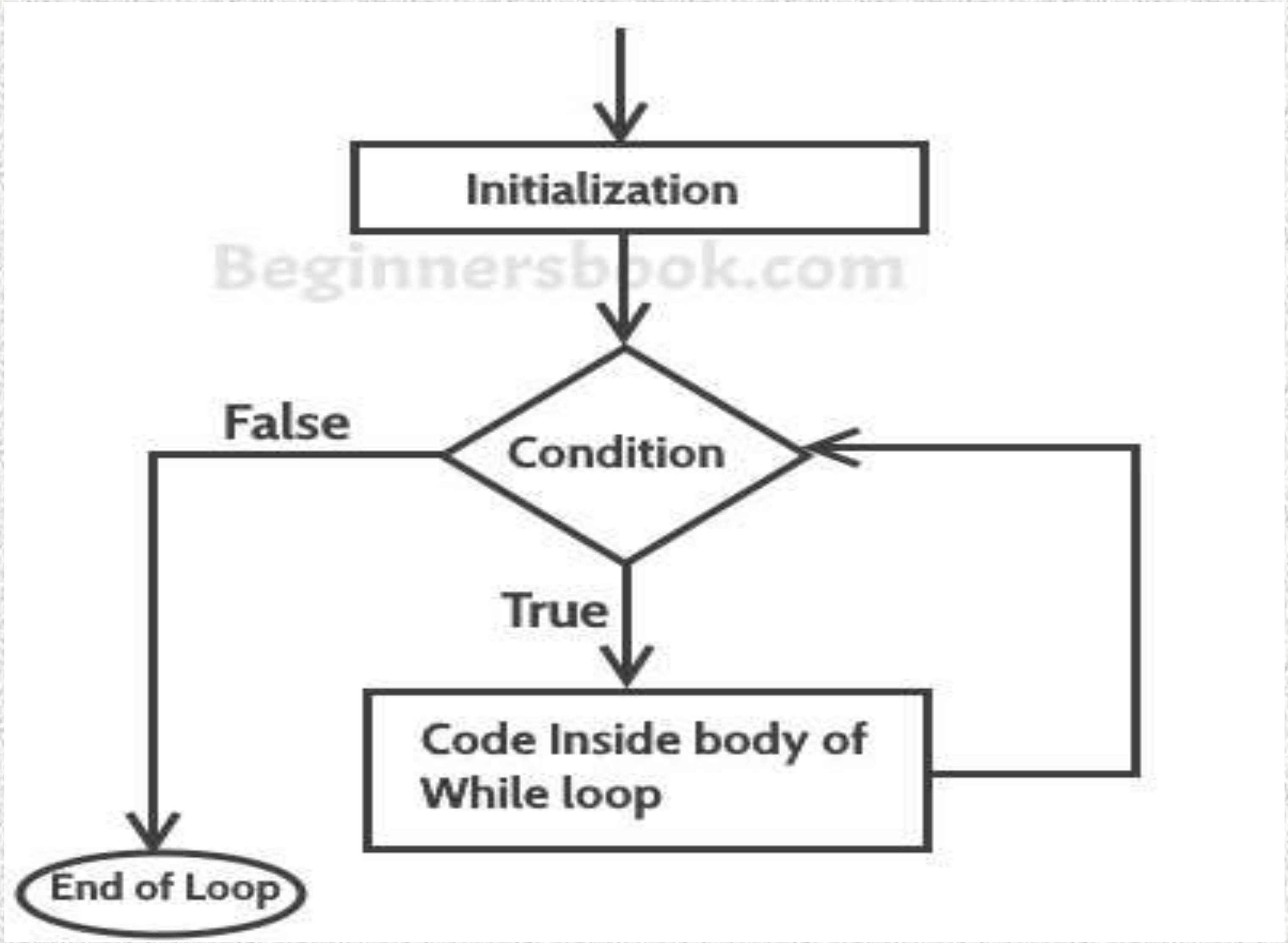
Sample nested for loop

```
class ForLoopExample
 {
    public static void main(String args[])
     {
         for(int i=10; i>1; i--)
             for (int j-5;j<=1;j--)
        {
              System.out.println("The value of  I and J is: "+i +j);
         }
    }
}
```

While loop in Java

- In while loop, condition is evaluated first and if it returns true then the statements inside while loop execute. When condition returns false, the control comes out of loop and jumps to the next statement after while loop.

Flow of Execution of WHILE Loop



Simple while loop example

```
class WhileLoopExample
{
    public static void main(String args[])
   {
        int i=10;
        while(i>1)
         {
             System.out.println(i);
             i--;
          }
    }
}
```

Infinite while loop

```
class WhileLoopExample2
{
    public static void main(String args[])
{
         int i=10;
         while(i>1)
         {
             System.out.println(i);
              i++;
         }
    }
 }
```

Iterating array using while loop

```
class WhileLoopExample3 {
    public static void main(String args[]){
         int arr[]={2,11,45,9};
         int i=0;
        while(i<4){
             System.out.println(arr[i]);
              i++;
         }     }     }
```

Output:

```
2
11
45
9
```

do-while loop

- In while loop, condition is evaluated before the execution of loop's body but in do-while loop condition is evaluated after the execution of loop's body

Syntax of do-while loop

```
do
{
   statement(s);
} while(condition);
```

Output:

```
10 9 8 7 6 5 4 3 2
```

do-while loop example

```
class DoWhileLoopExample {
    public static void main(String args[]){
        int i=10;
        do{
            System.out.println(i);
            i--;
        }while(i>1);
    }
}
```

Output:

```
10   9   8   7   6   5   4   3   2
```

Array using do while loop

```
class DoWhileLoopExample2 {
    public static void main(String args[]){
        int arr[]={2,11,45,9};
        //i starts with 0 as array index starts with 0
        int i=0;
        do{
            System.out.println(arr[i]);
            i++;
        }while(i<4);
    }}
```

Output:

```
2    11    45    9
```

Continue Statement

- **Continue statement** is mostly used inside loops. Whenever it is encountered inside a loop, control directly jumps to the beginning of the loop for next iteration, skipping the execution of statements inside loop's body for the current iteration. This is particularly useful when you want to continue the loop but do not want the rest of the statements(after continue statement) in loop body to execute for that particular iteration.

Syntax

continue word followed by semi colon

.

```
continue;
```

Output:

```
0 1 2 3 5 6
```

continue statement inside for loop

```
public class ContinueExample {
   public static void main(String args[]){
              for (int j=0; j<=6; j++)
              {
           if (j==4)
           {
                   continue;
               }
           System.out.print(j+" ");
              }
   }
}
```

Output:

```
0        1         2        3        5        6
```

Use of continue in While loop

```
public class ContinueExample2 {
  public static void main(String args[]){
            int counter=10;
            while (counter >=0)
            {
        if (counter==7)
        {
                counter--;
                continue;
        }
        System.out.print(counter+" ");
        counter--;
            }
  }
}
```

```
Output:   10   9   8   6   5   4   3   2   1   0
```

Break statement

- a) Whenever a break statement is encountered inside a loop, the control directly comes out of loop and the loop gets terminated for rest of the iterations. It is used along with if statement,
- It is also used in **switch case** control. Generally all cases in switch case are followed by a break statement so that whenever the program control jumps to a case, it doesn't execute subsequent cases.

Class

- A class can be defined as a template/blueprint that describes the behavior/state that the object of its type support

Class sample program

```
public class Dog
{
        String breed;
          int age;
          String color;
  void barking() {      }
  void hungry() {     }
  void sleeping() {    }
}
```

Object

- Objects have states and behaviors.
- Example: A dog has states - color, name, breed as well as behaviors – wagging the tail, barking, eating.
- An object is an instance of a class.

Contents of a class

- Local variables − Variables defined inside methods, constructors or blocks are called local variables. The variable will be declared and initialized within the method and the variable will be destroyed when the method has completed.
- Instance variables − Instance variables are variables within a class but outside any method. These variables are initialized when the class is instantiated. Instance variables can be accessed from inside any method, constructor or blocks of that particular class.
- Class variables − Class variables are variables declared within a class, outside any method, with the static keyword.
- A class can have any number of methods to access the value of various kinds of methods. In the above example, barking(), hungry() and sleeping() are methods.

Three steps for creating an object from a class

- **Declaration** − A variable declaration with a variable name with an object type.
- **Instantiation** − The 'new' keyword is used to create the object.
- **Initialization** − The 'new' keyw ord is followed by a call to a constructor. This call initializes the new object.

Constructors

- A **constructor in Java** is a special method that is used to initialize objects. The **constructor** is called when an object of a class is created.
- Constructors have the same name as the class or struct, and they usually initialize the data members of the new **object**. In the following example, a class named Taxi is defined by using a simple constructor

Characteristics of constructor

- They should be declared in the public section.
- They are invoked automatically when the objects are created.
- They do not have return (data type) type not even void and there for they cannot return any values.
- They can not be inherited, the a derived **class** can call the base **class** constructor.

creating an object

```
public class Puppy {
   public Puppy(String name) {
      // This constructor has one parameter, name.
      System.out.println("Passed Name is :" + name );
   }
   public static void main(String []args) {
      // Following statement would create an object myPuppy
      Puppy myPuppy = new Puppy( "tommy" );
   }
}
```

Source File Declaration Rules

- There can be only one public class per source file.
- A source file can have multiple non-public classes.
- The public class name should be the name of the source file as well which should be appended by **.java** at the end. For example: the class name is *public class Employee{}* then the source file should be as Employee.java.
- If the class is defined inside a package, then the package statement should be the first statement in the source file.
- If import statements are present, then they must be written between the package statement and the class declaration. If there are no package statements, then the import statement should be the first line in the source file.
- Import and package statements will imply to all the classes present in the source file. It is not possible to declare different import and/or package statements to different classes in the source file.

Packages and import

- A package in Java is used to group related classes. Think of it as **a folder in a file directory**. We use packages to avoid name conflicts, and to write a better maintainable code.
- Packages are divided into two categories:
- Built-in Packages (packages from the Java API)
- User-defined Packages (create your own packages)
- Syntax
- import *package.name.Class*; // Import a single class
import *package.name.**; // Import the whole package

Import a Class

- If you find a class you want to use, for example, the Scanner class, **which is used to get user input**, write the following code:
- Example
- import java.util.Scanner;
- In the example above, java.util is a package, while Scanner is a class of the java.util package.
- To use the Scanner class, create an object of the class and use any of the available methods found in the Scanner class documentation. In our example, we will use the nextLine() method, which is used to read a complete line:

- Example

Using the Scanner class to get user input:

```
import java.util.Scanner;
class MyClass
  {   public static void main(String[] args)
      { Scanner myObj = new Scanner(System.in);
        System.out.println("Enter username");
        String userName = myObj.nextLine();
        System.out.println("Username is: " + userName);
      } }
```

User-defined Packages

- To create your own package, you need to understand that Java uses a file system directory to store them. Just like folders on your computer

```
package mypack;
class MyPackageClass
{ public static void main(String[] args)
    { System.out.println("This is my package!");
    }
 }
```

Static class

- static members are those which belongs to the class and you can access these members without instantiating the class.
- The static keyword can be used with methods, fields, classes (inner/nested), blocks.
- **Static Methods** − You can create a static method by using the keyword *static*. Static methods can access only static fields, methods.

Example

```
public class MyClass
{ public static void sample()
{ System.out.println("Hello"); }
 public static void main(String args[])
{ MyClass.sample();
 } }
```

Output

```
Hello
```

- **Static Fields** – You can create a static field by using the keyword static. The static fields have the same value in all the instances of the class. These are created and initialized when the class is loaded for the first time. Just like static methods you can access static fields using the class name
- **Example**

```
public class MyClass
{ public static int data = 20;
public static void main(String args[])
{ System.out.println(MyClass.data); }
}
```

Output **20**

Overloading in java

- **overload** means that there are multiple versions of a constructor or method. They will each have a different number of **arguments**, or values, that they take in to work with.

Example of an overload

```
public Conversion
 {
     public double conversionRate;
     public double modifier;
     // constructor here:
     public Conversion(double c) {
      conversionRate = c;
     }
     //another constructor: the overload
     public Conversion(double c, double m) {
      conversionRate = c;
      modifier = .00587;
     }
   }
```

Different ways to overload the method

- There are two ways to overload the method

- By changing number of arguments
- By changing the data type

Overloading: changing no. of arguments

```
class Adder{
static int add(int a,int b){return a+b;}
static int add(int a,int b,int c){return a+b+c;}
}
class TestOverloading1{
public static void main(String[] args){
System.out.println(Adder.add(11,11));
System.out.println(Adder.add(11,11,11));
}}
```

Output: 22 33

Overloading: changing data type of arguments

```
class Adder{
static int add(int a, int b){return a+b;}
static double add(double a, double b){return a+b;}}
class TestOverloading2{
public static void main(String[] args){
System.out.println(Adder.add(11,11));
System.out.println(Adder.add(12.3,12.6));
}}
```

Test it Now

Output 22 24.9

this keyword in java

- In java, this is a **reference variable** that refers to the current object.
- Usages of java this keyword
- This is used to refer current class instance variable.
- this can be used to invoke current class method
- this can be used to invoke current class constructor.
- this can be passed as argument in method call.
- this can be passed as argument in constructor call.
- this can be used to return the current class instance from the method.

problem without this keyword

```
class Student{
int rollno;    String name;     float fee;
Student(int rollno,String name,float fee){
rollno=rollno;    name=name;   fee=fee;  }
void display(){System.out.println(rollno+" "+name+" "+fee);}  }
class TestThis1{
public static void main(String args[]){
Student s1=new Student(111,"ankit",5000f);
Student s2=new Student(112,"sumit",6000f);
s1.display();
s2.display();
}}
```

- Output: null 0.0 0   null 0.0

Solution of the above problem by this keyword

```
class Student{
int rollno;      String name;      float fee;
Student(int rollno,String name,float fee){
this.rollno=rollno;      this.name=name;     this.fee=fee;  }
void display(){System.out.println(rollno+" "+name+" "+fee);}    }
class TestThis2{
public static void main(String args[]){
Student s1=new Student(111,"ankit",5000f);
Student s2=new Student(112,"sumit",6000f);
s1.display();
s2.display();
}}
```

Output:

```
111 ankit 5000 112 sumit 6000
```

Java Enumerator (enum)

- The **Enum in Java** is a data type which contains a fixed set of constants.
- It can be used for days of the week (SUNDAY, MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY, and SATURDAY) , directions (NORTH, SOUTH, EAST, and WEST), season (SPRING, SUMMER, WINTER, and AUTUMN or FALL), colors (RED, YELLOW, BLUE, GREEN, WHITE, and BLACK) etc. According to the Java naming conventions, we should have all constants in capital letters. So, we have enum constants in capital letters.
- Java Enums can be thought of as classes which have a fixed set of constants (a variable that does not change). The Java enum constants are static and final implicitly.

Points to remember for Java Enum

- Enum improves type safety
- Enum can be easily used in switch
- Enum can be traversed
- Enum can have fields, constructors and methods
- Enum may implement many interfaces but cannot extend any class because it internally extends Enum class

Simple Example of Java Enum

```
class EnumExample1{
//defining the enum inside the class
public enum Season { WINTER, SPRING, SUMMER, FALL }
//main method
public static void main(String[] args) {
//traversing the enum
for (Season s : Season.values())
System.out.println(s);
}}
```

Output

```
WINTER SPRING SUMMER FALL
```

What is the purpose of the values() method in the enum?

The Java compiler internally adds the values() method when it creates an enum. The values() method returns an array containing all the values of the enum.

What is the purpose of the valueOf() method in the enum?

The Java compiler internally adds the valueOf() method when it creates an enum. The valueOf() method returns the value of given constant enum.

What is the purpose of the ordinal() method in the enum?

The Java compiler internally adds the ordinal() method when it creates an enum. The ordinal() method returns the index of the enum value.

Defining Java Enum

- The enum can be defined within or outside the class because it is similar to a class. The semicolon (;) at the end of the enum constants are optional. For example:
- **enum** Season { WINTER, SPRING, SUMMER, FALL }

Or,

- **enum** Season { WINTER, SPRING, SUMMER, FALL; }
- Both the definitions of Java enum are the same.

Example of specifying initial value to the enum constants

```
class EnumExample4{
enum Season{
WINTER(5), SPRING(10), SUMMER(15), FALL(20);
private int value;
private Season(int value){
this.value=value;
} }
public static void main(String args[]){
for (Season s : Season.values())
System.out.println(s+" "+s.value);
 }}
```

Output

```
WINTER 5   SPRING 10        SUMMER 15      FALL 20
```

Java Garbage Collection

- In java, garbage means unreferenced objects.
- Garbage Collection is process of reclaiming the runtime unused memory automatically. In other words, it is a way to destroy the unused objects.
- To do so, we were using free() function in C language and delete() in C++. But, in java it is performed automatically. So, java provides better memory management.

- Advantage of Garbage Collection
- It makes java **memory efficient** because garbage collector removes the unreferenced objects from heap memory.
- It is **automatically done** by the garbage collector(a part of JVM) so we don't need to make extra efforts.

- How can an object be unreferenced?
- By nulling the reference
- By assigning a reference to another
- By anonymous object etc.

1) By nulling a reference:

```
Employee e=new Employee();          e=null;
```

2) By assigning a reference to another:

```
Employee e1=new Employee();
Employee e2=new Employee();
e1=e2;
```

3) By anonymous object:

```
  new Employee();
```

Simple Example of garbage collection in java

```
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();
 } }
```

```
object is garbage collected object is garbage collected
```

Unit I question bank

- **PART A**

- List any four features of java
- What are the tools in JDK ?
- What are the basic concepts of oops?
- What is an object ?
- What are the contents of a java program?
- List out the API packages.
- What are the data types in java?
- What are the types of primitive / built in data types?
- What is a variable? Give example
- What is a literal? Give example
- What are the three types of comment lines in java?
- What are the keyboard input methods in java?
- What are the types of control statements
- What are the visibility modifiers in java?

- what is a constructor?
- Define package
- What is meant by garbage collection?
-

- **PART B & C**
- Explain the features of java
- Describe the concepts of oops
- Describe the structure of java program
- List out the API packages and describe all.
- Write the classification of data types in java
- Discuss about the rules for naming a variable
- Classify the literals and explain each
- Describe briefly the operators in java
- Explain the keyboard input methods in java?
- Describe the control statements in java.
- Explain the conditional statements.
- Explain the looping structures
- Explain the call by value and call by reference.
- Explain the concepts of constructor
- Define package and explain the classifications of package

- **UNIT I COMPLETED**