Java programming  
SYBCA SEMESTER-4

INSTRUCTOR: MIHIR SUDANI

Veer Narmad South Gujarat University

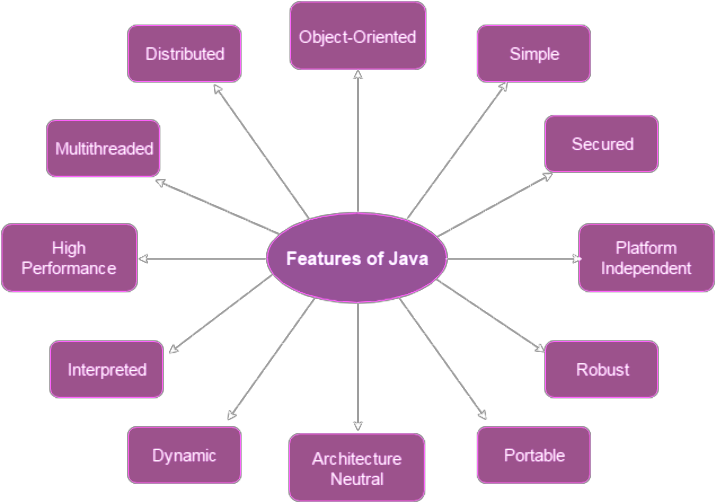
horizontal line

# Unit 1. Introduction to Java

| 1.1 Properties of Java  1.2 Comparison of java with C++  1.3 Java Compiler, Java Interpreter  1.4 Identifier, Literals, Operators, Variables, Keywords, Data Types  1.5 Branching: If – Else, Switch  1.6 Looping: While, Do-while, For  1.7 Type Casting |
| --- |

* **What is java ?**
* Java is a high level, robust, secured,class based and object-oriented programming language introduced by James Gosling in the year 1991.
* Java is platform independent.
* Java is mainly used for application.
* It is widely used in :
* Web applications
* Mobile applications
* Enterprise applications
* Middleware applications
* **History of java :**
* Java is general purpose object oriented programming language.
* Java was developed by James Gosling,Mike Sheridan and Patrick Naughton at Sun Microsystems.
* The team initially develops the language for digital devices such as set-top boxes, TVs, etc.
* James Gosling and his team are called this project "Greentalk" and its file extension was.gt and later became to known as "oak".
* **Why "Oak" ?**
* Oak is a symbol of strength and picked as national tree by many national countries like USA,France,Germany etc.
* Later, in 1995, Oak was renamed Java (from Java coffee, a type of coffee from Indonesia).
* Currently,java is used in internet programming,mobile devices,games and so on.
* As of September 2023,Java 21 is the latest version.
* Currently,java is managed by Oracle.

**1.1 Properties of Java / features of java / characteristics of java**



1. **Simple :**

* Syntax of java based on C++ so easier for programmer to learn it after C++.
* Removed many confusing and rarely used features, e.g., explicit pointers, operator overloading etc.

1. **Secured :**

* Java is secured because :
* No explicit pointer
* Java Programs run inside virtual machine sandbox

1. **Platform independent :**

* Java code can be run on multiple platforms e.g. Windows, Linux, Sun Solaris, Mac/OS etc.
* Java code is compiled by the compiler and converted into bytecode.
* This bytecode is a platform-independent code because it can be run on multiple platforms i.e. Write Once and Run Anywhere(WORA).

1. **Robust :**

* Java is known for strong memory management and error-checking mechanism.
* Java has runtime exception handling feature to provide programming support for robustness.
* The programmer must write the code to deal with exceptions.

1. **Portable :**

* Java is portable because it facilitates you to carry the Java bytecode to any platform.
* It doesn't require any implementation.

1. **Architecture neutral :**

* Java is architecture neutral because there are no implementation dependent features.
* for example, the size of primitive types is fixed. In C programming, int data type occupies 2 bytes of memory for 32-bit architecture and 4 bytes of memory for 64-bit architecture.

1. **Dynamic :**

* Before the development of Java, only static text was displayed in the browser. But using applet program, we can also create animation dynamically on the Internet.

1. **Interpreted :**

* During compilation, Java compiler converts the source code of the program into byte code.
* This byte code can be executed on any system machine with the help of Java interpreter in JVM.
* If we take any other programming language, only a compiler or an interpreter is used to run programs.
* But in Java, we use both compiler and interpreter for the execution of the program.

1. **High performance :**

* Java applications can achieve high performance through features like Just-In-Time (JIT) compilation and efficient memory management.

1. **Multithreaded :**

* Java multithreading allows you to do multiple tasks at the same time. This is possible because modern day computers have multiple CPUs.

1. **Distributed :**

* Java is distributed because it facilitates users to create distributed applications in Java.
* Remote Method Invocation (RMI) and Enterprise Java Beans (EJB) are used for creating distributed applications.

1. **Object-oriented :**

* Java follows an object-oriented programming paradigm and reusable code through classes and objects.
* The main principles or features of object-oriented programming are :

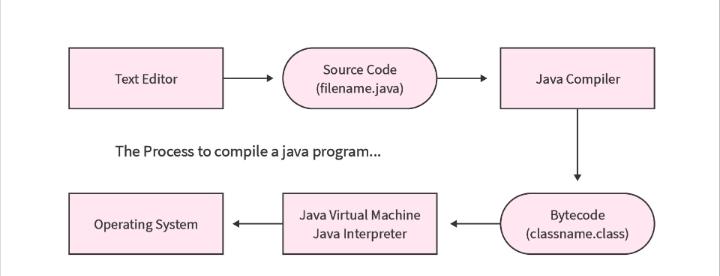
1. Class and Object
2. Abstraction
3. Encapsulation
4. Inheritance
5. Polymorphism

**1.2 Comparison of java with C++**

| **C++** | **Java** |
| --- | --- |
| C++ is platform dependent. | Java is platform independent |
| It is mainly used for system programming. | It is mainly used for application programming. |
| It is a semi object oriented programming language. | it is a pure object oriented programming language. |
| It supports multiple inheritance. | It doesn't support multiple inheritance. |
| It supports both constructor and destructor. | It supports only constructor. |
| It supports (::) operator and operator overloading. | It doesn't support (::) as well as operator overloading. |
| It supports three access specifier like  Public,private,protected. | It supports four access specifier like public,private, protected, default. |
| It supports pointers | It doesn't support pointer. |

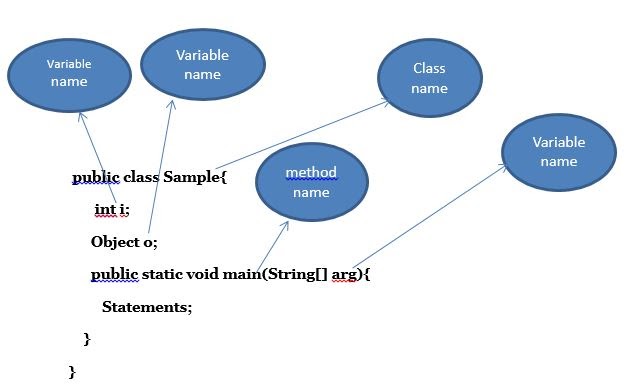
**1.3 Java Compiler, Java Interpreter**

* Java is a hybrid programming language,it uses compiler as well as interpreter.
* **Java compiler :**
* When you write Java code, it's in a human-readable form.
* The Java compiler takes this code and translates it into a lower-level language called bytecode.
* Bytecode is a set of instructions that can be executed by the Java Virtual Machine (JVM).
* **Java Interpreter :**
* The Java interpreter, in this context, is part of the JVM.
* It takes the bytecode produced by the compiler and executes it line by line.
* It converts each bytecode instruction into machine code for the specific computer it is running on.
* This allows your Java program to run on different platforms without modification.

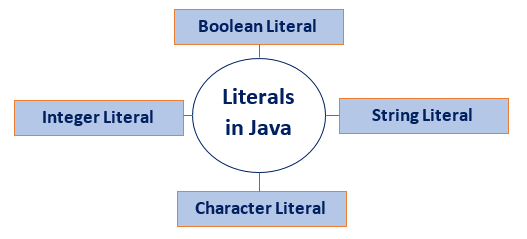


**1.4 Java Tokens / (Identifier, Literals, Operators, Variables, Keywords, Data Types)**

* **Identifier :**
* In java,an identifier is the name of variable ,method,class or packages or interface that is used for the purpose of identification.
* In other word,Identifier refers to name of variable,method ,class and so on.



* Rules for the identifier :
* keywords can't be used as an identifier.
* Identifiers are case-sensitive.
* We can not use whitespace in between identifier.
* Identifiers always start with letters,$ or \_.
* **Literals :**
* Literals in Java,refers to fixed value that do not change during the execution of the program.
* Literals are the constant values assigned to the constant variable.



| **Literals** | **Example** | **Description** |
| --- | --- | --- |
| Integer | int myNum= 42; | 42 is an integer literal |
| Character | char myChar = 'A'; | A is a character literal |
| String | String myStr="hello"; | hello is a string literal |
| Boolean | bollean myBool=true; | true is a boolean literal |

* **Operators :**
* operators are symbols that are used to perform operations on variables and values.
* There are many types of operators which are as follows :

1. **Arithmetic operators**

* Arithmetic operators are used with numeric values to perform common mathematical operations:

| **Operator** | **Operation** |
| --- | --- |
| + | Addition |
| - | Subtraction |
| \* | Multiplication |
| / | Division |
| % | Modulus Operator |

1. **Relational Operators**

* Comparison operators are used to compare two values:

| **Operator** | **Operation** |
| --- | --- |
| == | Equal to |
| < | Less than |
| > | Greater than |
| <= | Less than or equal to |
| >= | Greater than or equal to |
| != | Not equal to |

1. **Logical operators**

* Logical operators are used to combine conditional statements:

| **Operator** | **Operation** |
| --- | --- |
| Logical AND | && |
| Logical OR | || |
| Logical NOT | ! |

1. **Assignment operators**

* Assignment operators are used to assign values to variables:

| **Operator** | **Operation** | **Equivalent to** |
| --- | --- | --- |
| = | X=5 | X=5 |
| += | X+=5 | X=X+5 |
| -= | X-=5 | X=X-5 |
| \*= | X\*=5 | X=X\*5 |
| /= | X/=5 | X=X/5 |
| %= | X%=5 | X=X%5 |

1. **Bitwise operators**

* In Java, bitwise operators manipulate individual bits of primitive data types.

| **Symbol** | **Operator** | **Description** |
| --- | --- | --- |
| & | Bitwise AND | * It is a binary operator denoted   by the symbol &.   * It returns 1 if and only if both bits are 1, else returns 0. |
| | | Bitwise OR | * It is a binary operator denoted by the symbol |. * It returns 1 if and only if any one bits are 1, else returns 0. |
| ^ | Bitwise XOR (exclusive OR) | * It is a binary operator denoted by the symbol ^. * It returns 0 if both bits are the same, else returns 1 |
| ~ | Bitwise NOT | * It is a unary operator denoted by the symbol ~ (tilde). * It make every 1 to 0 and 0 to 1. |
| << | Left Shift | * The left operand value is moved left by the number of bits specified by the right operand. |
| >> | Right Shift | * The left operand value is moved right by the number of bits specified by the right operand. |
| >>> | Unsigned Right Shift | * Shifts bits to the right, filling empty spaces with zeros. |

1. **Conditional(ternary) operators**

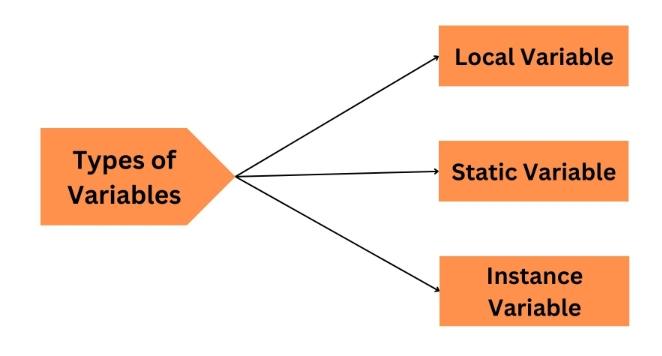
* In Java, the ternary operator is a shorthand way of writing an if-else statement with a single line of code.
* **Syntax :**

| variable = condition ? expression\_If\_True : expression\_If\_False; |
| --- |

* If the condition is true, it evaluates expression\_If\_True; otherwise, it evaluates expression\_If\_False.

| **Example** | **Output** |
| --- | --- |
| class test  {  public static void main(String args[ ])  {  int a=6,b=20;  int max=(a>b)?a:b;  System.out.print(max+" is max");  }  } | 20 is max |

* **Variables :**
* Variable is the name of memory location.
* In other word,we can say that it is user defined name which is the given by user
* Variable can store any types of value.
* Types of variables :



1. **Local variables :**

* A variable which is declared inside the body of the method or method parameter called local variable.

1. **Static variables :**

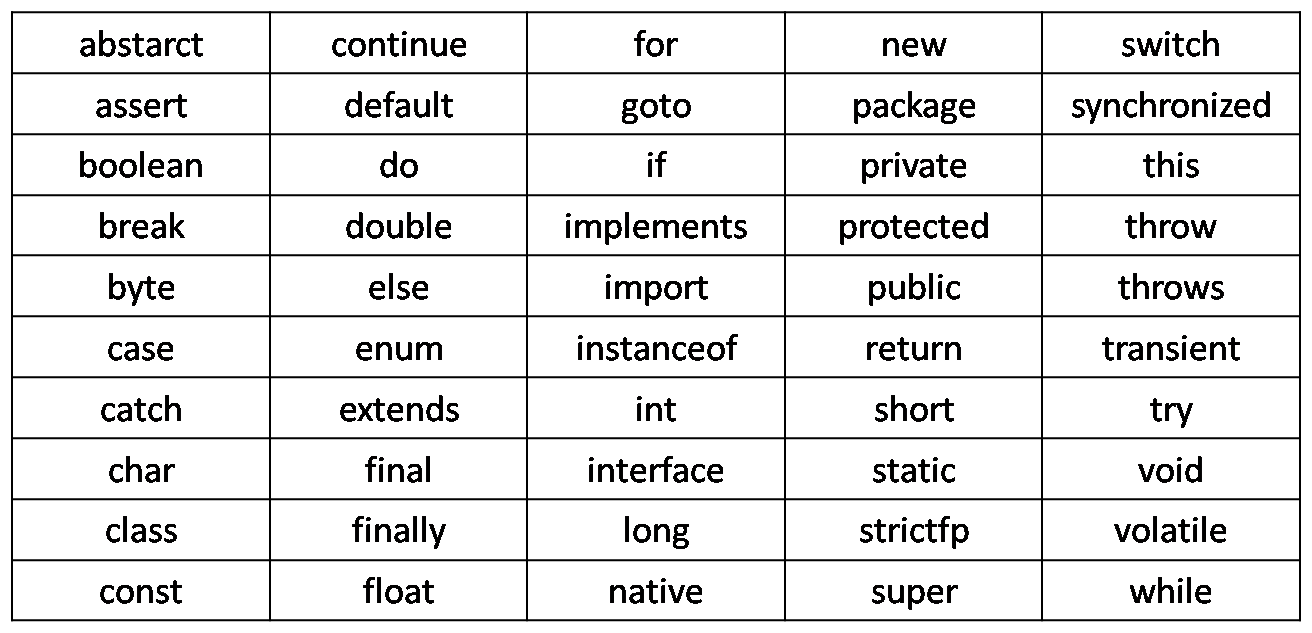
* A variable which is declared with the help of static keyword called static variable.
* You can call a static variable directly or using the class name.

1. **Instance variables :**

* A variable which is declared inside the class but outside of all the methods called instance variable.
* To call instance variables, you have to create an object for it and then with the help of that object you can call the particular instance variable.

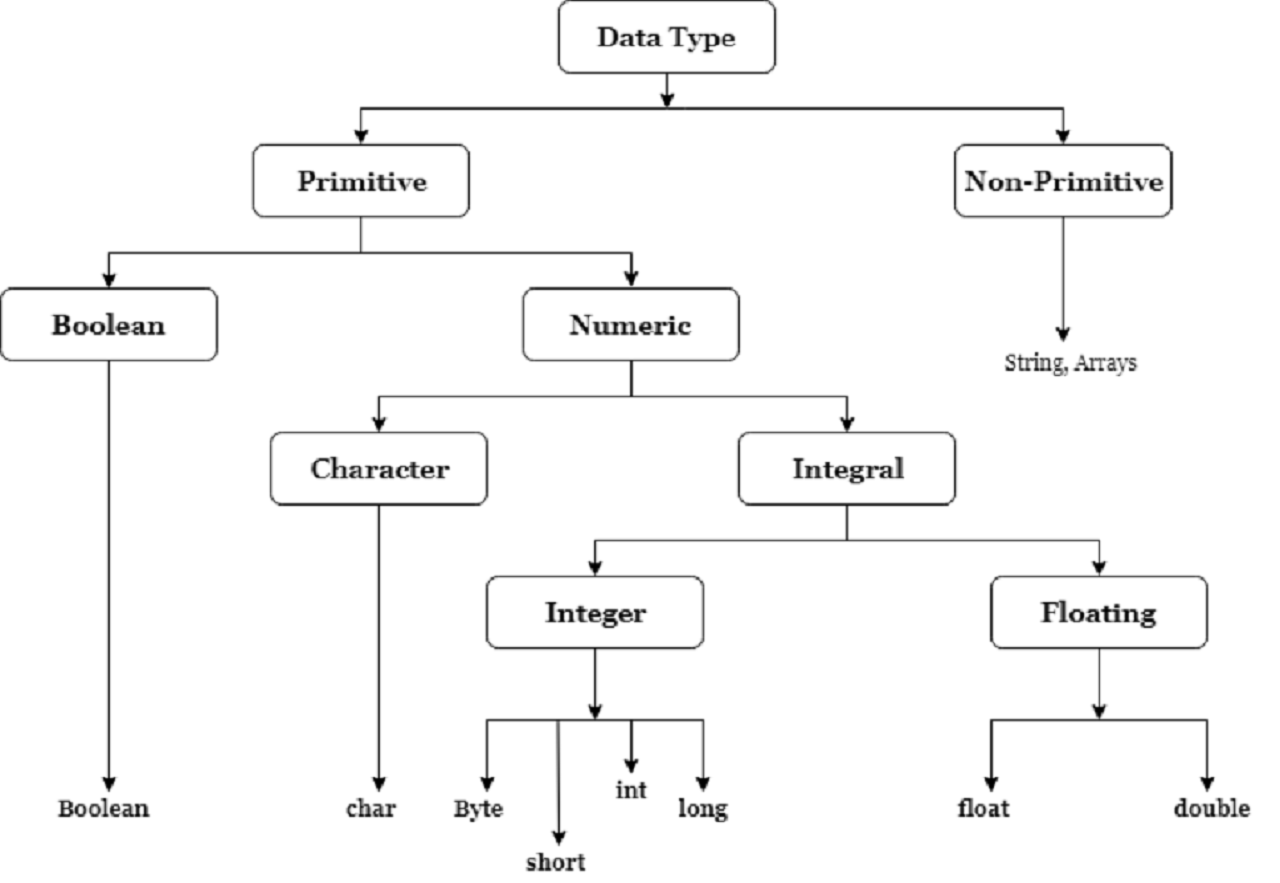
| **Example** | **Output** |
| --- | --- |
| class test  {  **static int b=20;** **//static variable**  **int c=30;**  **// instance variable**  public static void main(String args[ ])  {  test ref=new test( );  **int a=10;** **//local variable**  System.out.println("a :: "+a);  System.out.println("b :: "+b);  System.out.println("c :: "+ref.c);  }  } | a :: 10  b :: 20  c :: 30 |

* **Keywords :**
* Keywords are the reserved word whose meaning is already defined in the compiler called keywords.



* **Data Types :**
* Data-types means container.
* A datatypes defines the type of data where variable will store.
* There are mainly two types of Data Types:

1. Primitive datatypes
2. Non-primitive datatypes



**Primitive datatypes :**

* In java,primitive data types are used to store simple values like numbers, characters, or boolean values.
* It is pre-defined datatypes.
* **Integral Types :**

1. **Byte**

* Byte datatype is used to represent values in range -128 to 127.
* Byte datatype takes 1 byte to store a value into variable.

1. **Short**

* In Java, the "short" data type is a 16-bit signed two's complement integer.
* It has a range from -32,768 to 32,767.
* Short datatype takes 2 byte to store a data.

1. **Int**

* In Java, the int data type is a 32-bit signed two's complement integer.
* It has a range from -2^31 to 2^31 - 1.
* Integer datatype takes 4 byte to store a data.

1. **Long**

* In Java, the long data type is a 64-bit signed two's complement integer.
* It has a larger range compared to int, spanning from -2^63 to 2^63 - 1.
* It takes 8 bytes to store a data.
* **Floating point types :**

1. **Float**

* In Java, the float data type is used to represent single-precision 32-bit floating-point numbers.
* It takes up 4 bytes of memory.

1. **Double**

* In Java, the double data type is used to represent double-precision 64-bit floating-point numbers.
* It occupies 8 bytes of memory.
* **Other types :**

1. **Char**

* In Java, the char data type is used to represent a single 16-bit Unicode character.
* You can assign a single character enclosed in single quotes to a char variable.
* It takes 2 bytes of memory.

1. **Boolean**

* In Java, the boolean data type is used to represent a binary condition, typically denoted as either true or false.
* It is commonly used for logical comparisons and decision-making in control structures.

**Example of primitive datatypes :**

| byte myByte = 127;  short myShort = 32000;  int myInt = 2147483647;  long myLong = 9223372036854775807;  float myFloat = 3.14;  double myDouble = 2.71828;  char myChar = 'A';  boolean myBoolean = true; |
| --- |

**Non-primitive datatypes :**

* Non-primitive datatypes are derived from primitive datatypes.

1. **String**

* Strings are a special type of non-primitive data type, representing sequences of characters.
* **Example :**

| String myString = "Hello, Java!"; |
| --- |

1. **Arrays**

* Arrays can store multiple values of the same data type or objects.
* **Example :**

| int[ ] intArray = {1, 2, 3};  String[ ] stringArray = {"apple", "banana", "cherry"}; |
| --- |

**1.5 Branching: If – Else, Switch**

* **Branching :**
* Branching is a way for your program to make choices and go down different paths based on conditions.

1. **If-Else :**
2. **If statement :**

* It evaluates condition,if condition becomes true then the code inside the if block is executed.
* If condition becomes false then the code inside the if block is skipped.

| **Syntax** | **Example** |
| --- | --- |
| if(condition)  {  // code to be executed  } | class test  {  public static void main(String args[ ])  {  int a=10,b=10;  if(a==b)  {  System.out.print("A and B are same");  }  }  } |

1. **If-else statement :**

* It evaluates condition,if condition becomes true then if block of code is executed.
* If condition becomes false then else block of code is executed

| **Syntax** | **Example** |
| --- | --- |
| if(condition)  {  // code to be executed  }  else  {  // code to be executed  } | class test  {  public static void main(String args[ ])  {  int a=10,b=10;  if(a==b)  {  System.out.print("A and B are same");  }  else  {  System.out.print("A and B are not same");  }  }  } |

1. **If-else ladder :**

* In If-else ladder more than one condition will check.
* In If-else ladder,one if statement, multiple else-if statement and at last one else statement.

| **Syntax** | **Example** |
| --- | --- |
| if(condition)  {  // code to be executed  }  else if(condition)  {  // code to be executed  }  else  {  // code to be executed  } | class test  {  public static void main(String args[ ])  {  int a=10,b=5,c=5;  if(a==b)  {  System.out.print("A and B are same");  }  else if(b==c)  {  System.out.print("B and C are same");  }  else  {  System.out.print("A and C are same");  }  }  } |

1. **Nested if :**

* Whenever we define if-else block,inside this if-else block,we define another if-else block is called nested if statement.
* If-else statement within the another if-else statement.

| **Syntax** | **Example** |
| --- | --- |
| if (condition1)  {  // statement 1.1  if (condition2)  {  // statement 2.1  }  else  {  // statement 2.2  }  }  else  {  // statement 1.2  } | class test  {  public static void main(String args[ ])  {  int x=5,y=10,z=6;  if(x>y)  {  System.out.print("X is max");  if(x>z)  {  System.out.print("X is max");  }  else  {  System.out.print("Z is max");  }  }  else  {  System.out.print("y is max");  }  }  } |

1. **Switch :**

* A switch statement in Java is like a menu where you choose an option, and it helps your program make decisions based on a specific value.
* In simple words, the Java switch statement executes one statement from multiple conditions.
* Switch statement is as like as if-else ladder.

| **Syntax** | **Example** |
| --- | --- |
| switch(expression)  {  case 1:  statement 1;  break;  case 2:  statement 2;  break;  case n:  statement n;  break;  default:  default statement;  } | import java.util.Scanner;  class test  {  public static void main(String args[ ])  {  Scanner s=new Scanner(System.in);  System.out.print("enter a day :: ");  int ch=s.nextInt( );  switch(ch)  {  case 1: System.out.print("monday");  break;  case 2:  System.out.print("tuesday");  break;  case 3:  System.out.print("wednesday");  break;  default:  System.out.print("other day");  }  }  } |

**1.6 Looping: While, Do-while, For**

* In programming language,loops are used to execute a set of instructions/functions repeatedly when some conditions become true.
* There are three types of loop in java :

1. While loop
2. Do-while loop
3. For loop

* Now,we will see all the loop one by one which are as follows:

1. **While loop :**

* While loop is an entry controlled loop.
* In while loop, given condition is checked first and then statements are evaluated as per the given condition.

| **Syntax** | **Example** |
| --- | --- |
| While(condition)  {  //Statements;  } | int i=1;  while(i<=5)  {  System.out.println(i);  i++;  } |

1. **Do-while loop :**

* Do-while loop is an exit controlled loop.
* In do-while loop,statements are evaluated first as per the given condition and then given condition is checked.

| **Syntax** | **Example** |
| --- | --- |
| do  {  // statements;  }  while (expression); | int i=1;  do  {  System.out.println(i);  i++;  }  while(i<=5); |

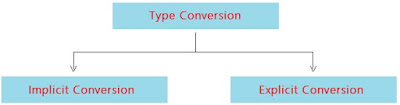
1. **For loop :**

* For loop is also an entry controlled loop.
* For loop is used when you want to repeat the statements with fixed numbers of time.

| **Syntax** | **Example** |
| --- | --- |
| for(init; condition; icr/dcr)  {  //statements;  } | int i;  for(i=1;i<=5;i++)  {  System.out.println(i);  } |

**1.7 Type Casting**

* Type casting or type conversion is a method of converting one datatype to another datatype is called type casting.
* There are two types of typecasting in Java:



1. **Implicit typecasting :**

* This occurs when you convert a smaller data type into a larger one.
* It is automatically performed by the compiler.
* It is also known as automatic type conversion or widening conversion.

| **Example** | **Output** |
| --- | --- |
| int a=10;  double b=a;  System.out.println("int :: "+a);  System.out.println("float :: "+b); | int :: 10  float :: 10.0 |

1. **Explicit typecasting :**

* This is used when you want to convert a larger data type into a smaller one.
* By default,compiler does not allow explicit typecasting.
* You can perform explicit typecasting by specifying the desired type in parentheses before the variable you want to cast.
* It is also known as manual type conversion or narrowing conversion.

| **Example** | **Output** |
| --- | --- |
| double pi=3.14;  int a=(int)pi;  System.out.println("double :: "+pi);  System.out.println("int :: "+a); | double :: 3.14  int :: 3 |