Implementation

Software design and implementation is the stage in the software engineering process at which an executable software system is developed.

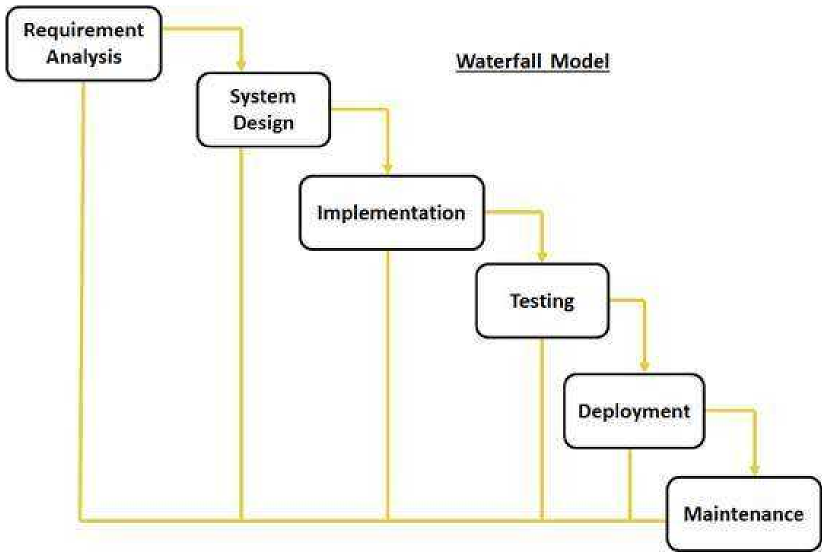
Software design is a creative activity in which you identify software components and their relationships, based on a customer's requirements

Implementation is the process of realizing the design as a program.

Process Models

Water fall model

The Waterfall Model was first Process Model to be introduced. It is also referred to as a linear-sequential life cycle model. It is very simple to understand and use. In a waterfall model, each phase must be completed before the next phase can begin and there is no overlapping in the phases.



The sequential phases in Waterfall model are:

- **Requirement Gathering and analysis:** All possible requirements of the system to be developed are captured in this phase and documented in a requirement specification doc.
- **System Design:** The requirement specifications from first phase are studied in this phase and system design is prepared. System Design helps in specifying hardware and system requirements and also helps in defining overall system architecture.
- **Implementation:** With inputs from system design, the system is first developed in small programs called units, which are integrated in the next phase. Each unit is developed and tested for its functionality which is referred to as Unit Testing.

- **Integration and Testing:** All the units developed in the implementation phase are integrated into a system after testing of each unit. Post integration the entire system is tested for any faults and failures.
- **Deployment of system:** Once the functional and non functional testing is done, the product is deployed in the customer environment or released into the market.
- **Maintenance:** There are some issues which come up in the client environment. To fix those issues patches are released. Also to enhance the product some better versions are released. Maintenance is done to deliver these changes in the customer environment.

RAD Model

RAD model is Rapid Application Development model. In RAD model the components or functions are developed in parallel as if they were mini projects. The developments are time boxed, delivered and then assembled into a working prototype.

This can quickly give the customer something to see and use and to provide feedback regarding the delivery and their requirements.

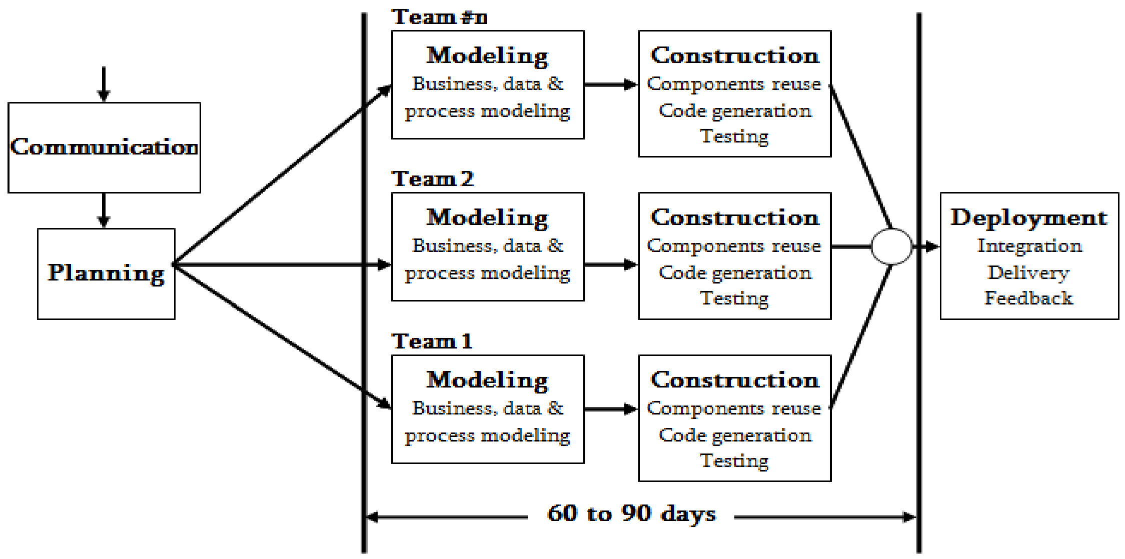


Figure : Flowchart of RAD model

The phases in the rapid application development (RAD) model are:

Business modeling: The information flow is identified between various business functions.

Data modeling: Information gathered from business modeling is used to define data objects that are needed for the business.

Process modeling: Data objects defined in data modeling are converted to achieve the business information flow to achieve some specific business objective.

Application generation: Automated tools are used to convert process models into code and the actualsystem.

Testing and turnover: Test new components and all the interfaces.

Component based Development Model

the model composes applications from prepackaged software components.

Modeling and construction activities begin with the identification of candidate components. These components can be designed as either conventional software modules or object-oriented classes or packages of classes. Regardless of the technology that is used to create the components, the component-based development model incorporates the following steps

• Available component-based products are researched and evaluated for the application domain in question
• Component integration issues are considered.
• A software architecture is designed to accommodate the components.
• Components are integrated into the architecture.
• Comprehensive testing is conducted to ensure proper functionality.

Unified Modeling Language(UML)

- UML is a modeling language for visualizing, specifying, constructing and documenting a software system and its components.

Visualizing

- UML includes both graphical and textual representation
- Notations of UML are visual and well defined

Specifying

- UML helps to build models that are precise, unambiguous and complete

Constructing

- Models can be directly connected to a variety of programming language
- It is possible to map from model in UML to

- Object oriented languages such as Java or C++
- Tables in relational database

Documenting

- UML addresses the documentation of systems architecture and all of its details like requirements, architecture, design, source code, project plan and tests

UML Primary Goals

- The primary goals in the design of UML are:

- Provide users a ready to use, expressive visual modeling language so that they can develop and exchange meaningful models
- Provide extensibility and specialization mechanisms to extend core concepts

- Provide required formal basis for understanding the modeling language
- Encourage the growth of Object Oriented tools
- Supports development concepts

UML Diagrams

1. Class Diagram(static)
2. Use case diagram
3. Behavior diagram(dynamic)
   - Interaction Diagram
     - Sequence Diagram
     - Collaboration Diagram
   - State chart Diagram
   - Activity Diagram

4. Implementation Diagram
   - Component Diagram
   - Deployment diagram

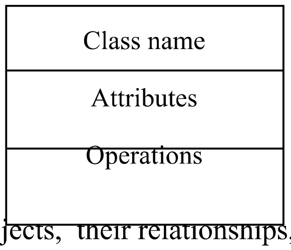
Class diagram

- Class diagram describe the structure of systems in terms of classes and objects.

- Class is a collection of objects that share a set of attributes and operations

- Attributes are data held by objects

- Operations are transformations applied to objects



It is a virtual representation of objects, their relationships, and their structures is for ease of undersatnding

Class Notation:

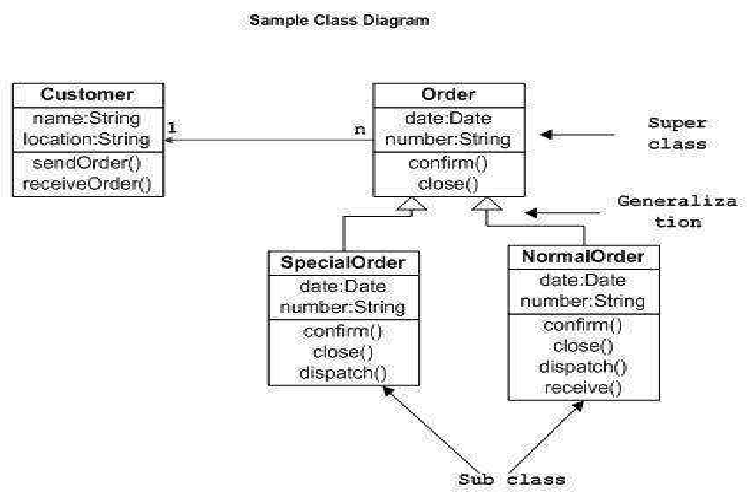
UML *class* is represented by the diagram shown below. The diagram is divided into four parts.

- The top section is used to name the class.
- The second one is used to show the attributes of the class.
- The third section is used to describe the operations performed by the class.
- The fourth section is optional to show any additional components.

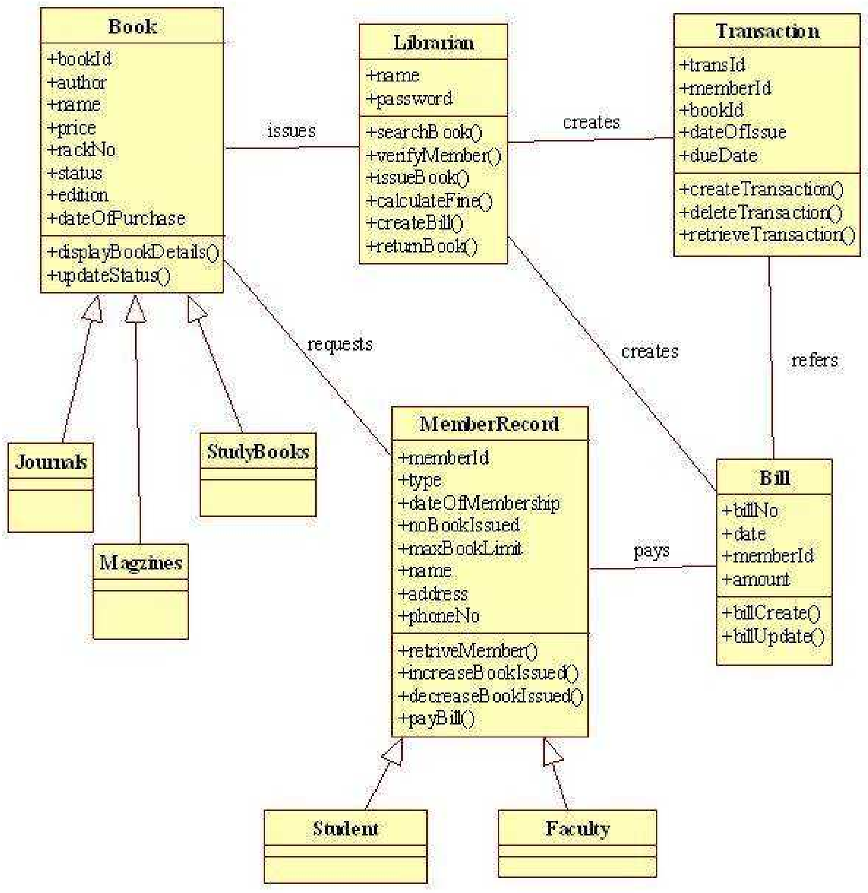
Object Diagram

It is an instance of class diagram

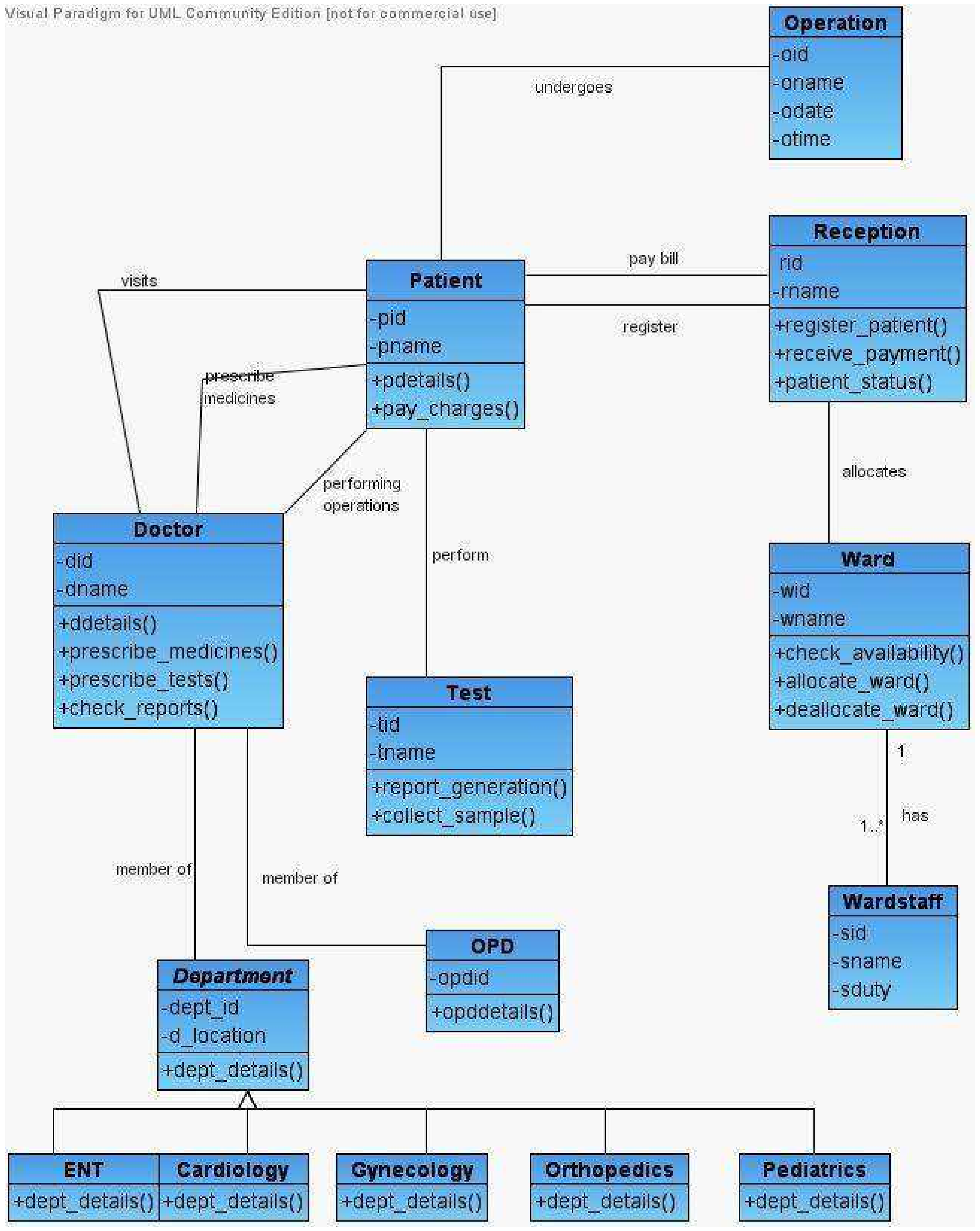
It shows snapshot of the detailed state of the system at a point in time.



Note: + public data, - private data, # protected data



Class Diagram for Library Management System



Class Diagram for Hospital Management System

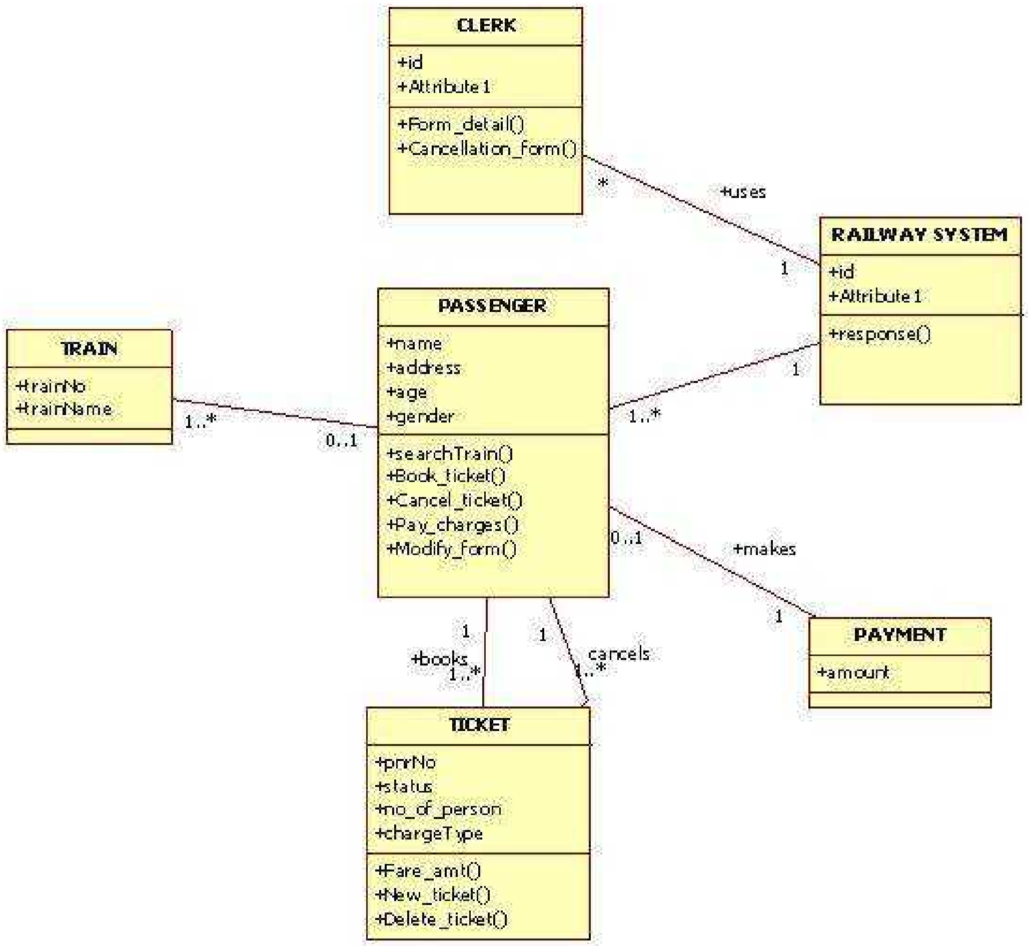


Fig: Class Diagram for railway ticket reservation system

<u>Use case Diagram</u>

Use case diagrams are a set of use cases, actors and their relationships. They represent the use case view of a system.

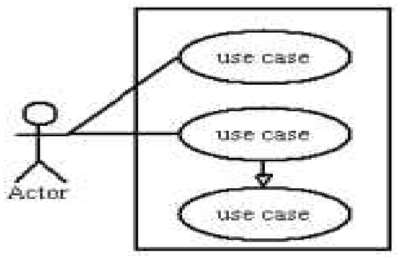
A use case represents a service provided by a system or a particular functionality of a system. So use case diagram is used to describe the relationships among the functionalities and their internal/external controllers. These controllers are known as actors.

The purpose of use case diagram is to capture the dynamic aspect of a system. Use case diagrams are used to gather the requirements of a system including internal and external influences.

Basic Use Case Diagram Symbols and Notations

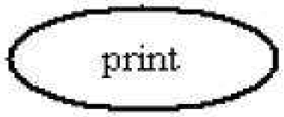
System

Draw your system's boundaries using a rectangle that contains use cases. Place actors outside the system's boundaries.



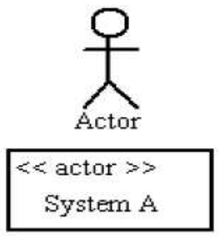
Use Case

Draw use cases using ovals. Label with ovals with verbs that represent the system's functions.



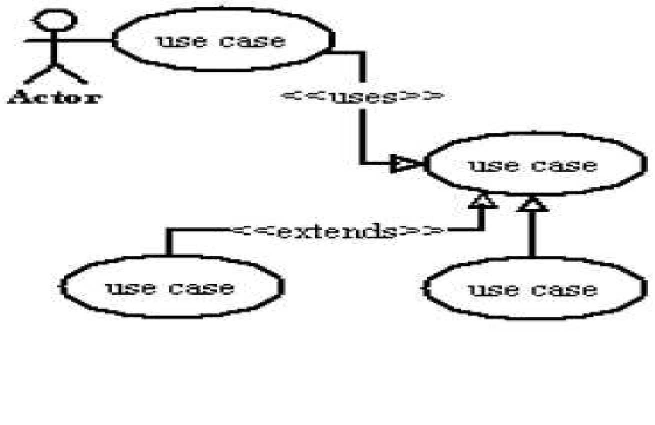
Actors

Actors are the users of a system. When one system is the actor of another system, label the actor system with the actor stereotype. An actor is an entity that must interact with the system under development



Relationships

Illustrate relationships between an actor and a use case with a simple line. For relationships among use cases, use arrows labeled either "uses" or "extends." A "uses" relationship indicates that one use case is needed by another in order to perform a task. An "extends" relationship indicates alternative options under a certain use case.



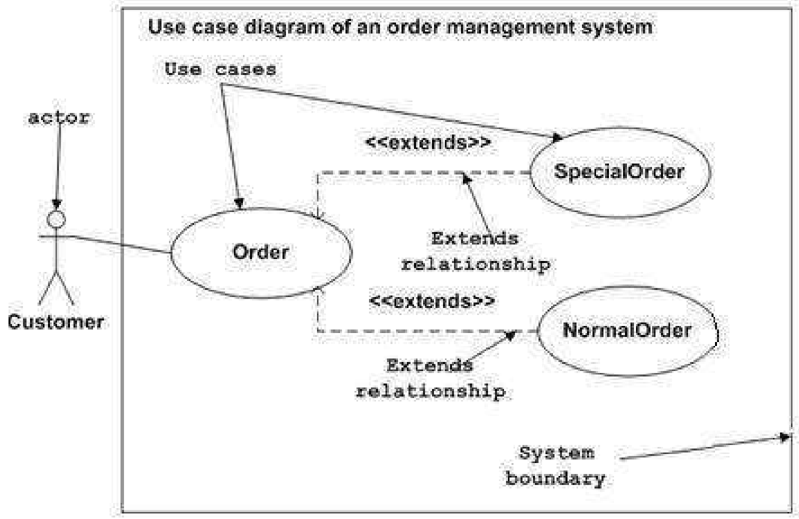


Figure: Sample Use Case diagram

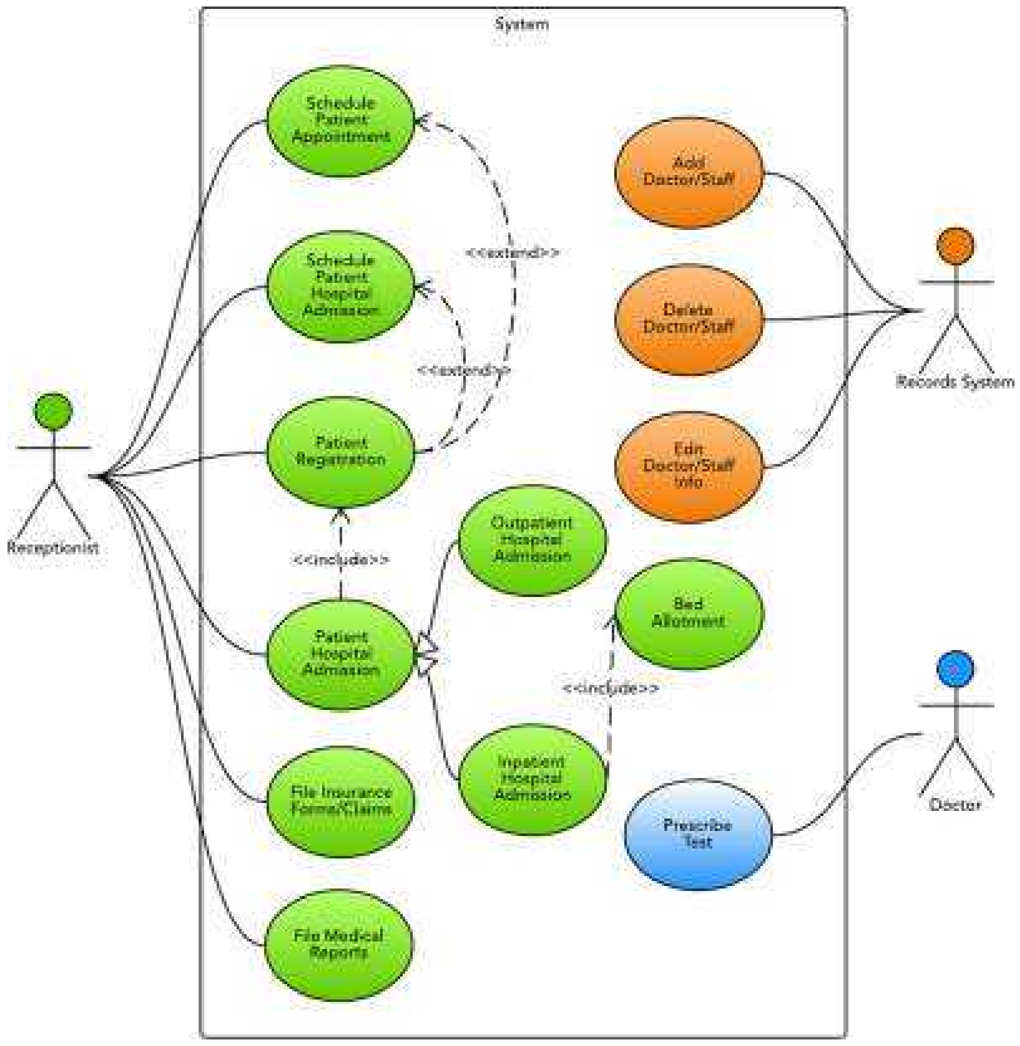


Fig: Use case Diagram for Hospital Management System

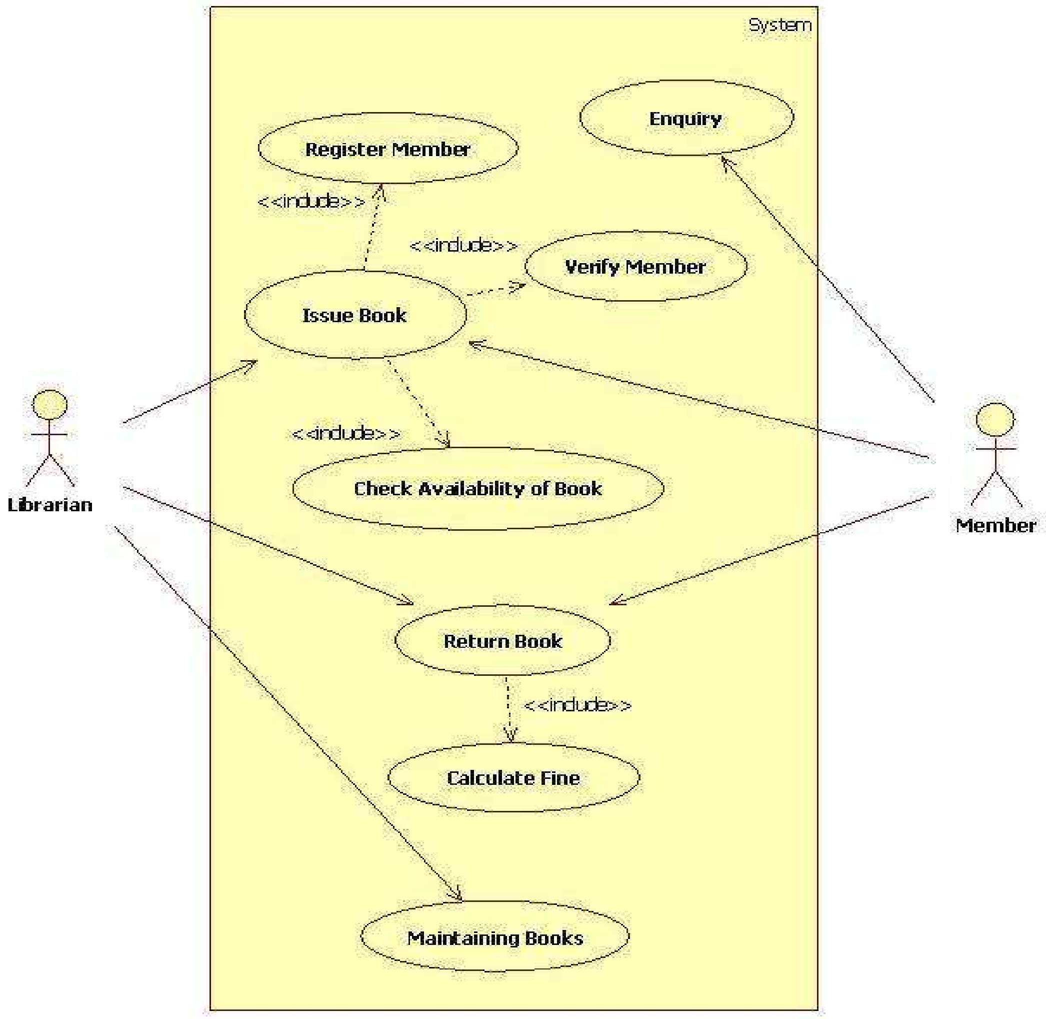


Fig: Use case Diagram for Library Management System

Introduction to Java

Genesis of Java

Java's History

Developed and maintained by ***Oracle***

- James Gosling and Sun Microsystems
- Originally called Oak but it was renamed "Java"
- Aimed at producing an operating environment for networked devices and embedded systems

-Design objectives for the language

Simple, secure, portable, object-oriented, robust, multithreaded, architecture-neutral, interpreted , distributed, dynamic

-It can be integrated into browsers

Dynamic and self executing programs

Java Applications and Applets

Java can be used to create **two types of programs**

-An ***application*** is a program that runs on your computer, under the operating system of that computer.

-An ***applet*** is an application designed to be transmitted over the Internet and executed by a Java-compatible Web browser.

An applet is actually a tiny Java program

An applet is a program that can react to user input and dynamically change

Java Programming Language Platforms

-A Java platform is a particular environment in which Java programming language applications run.

-All Java platforms consist of a Java Compiler, Java Virtual Machine (JVM) and an application programming interface (API).

An ***API*** is a collection of software components that you can use to create other software components or applications.

The API is a library of available java classes, packages and interfaces.

Different Platforms

Java Platform, Standard Edition (J2SE)

- Java SE's API provides the core functionality of the Java programming language.
- J2SE can be used to develop standalone applications or applets.

Java Platform, Enterprise Edition (J2EE)

The Java EE platform provides an API and runtime environment for developing and running large-scale applications

Java platform Micro Edition (J2ME)

J2ME can be used to develop applications for mobile devices such as cell phones.

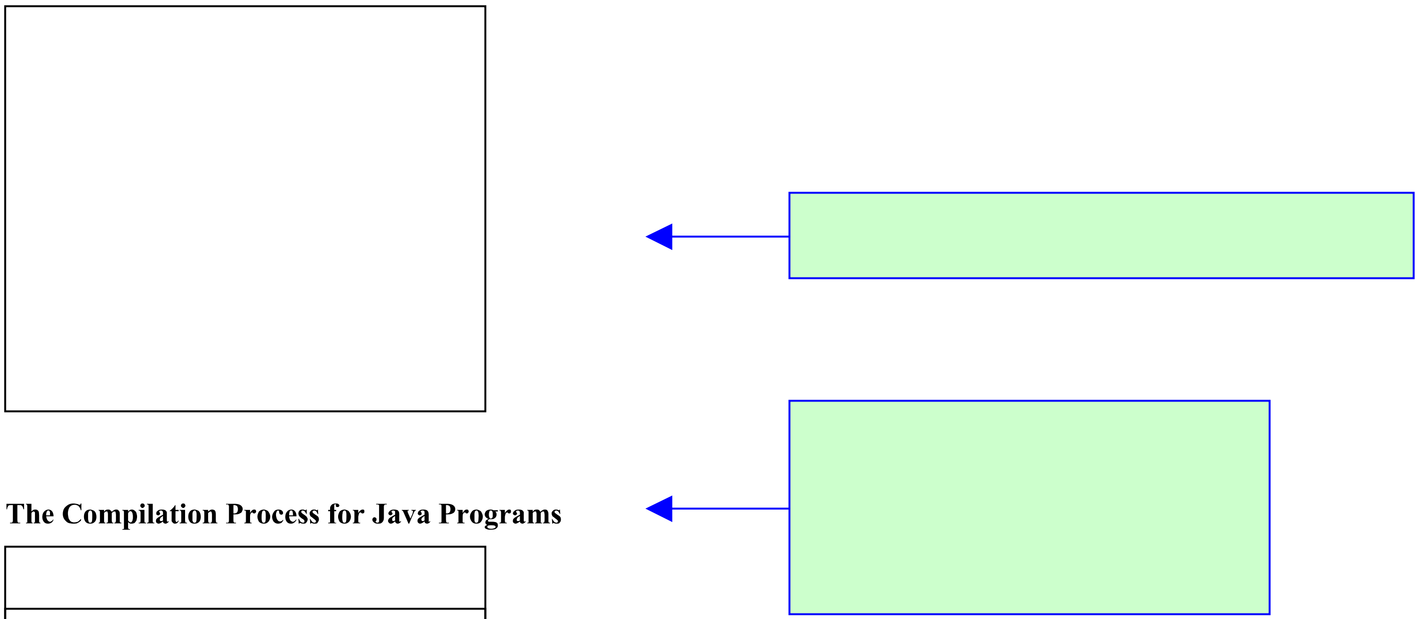
JavaFX

JavaFX is a platform for creating rich internet applications

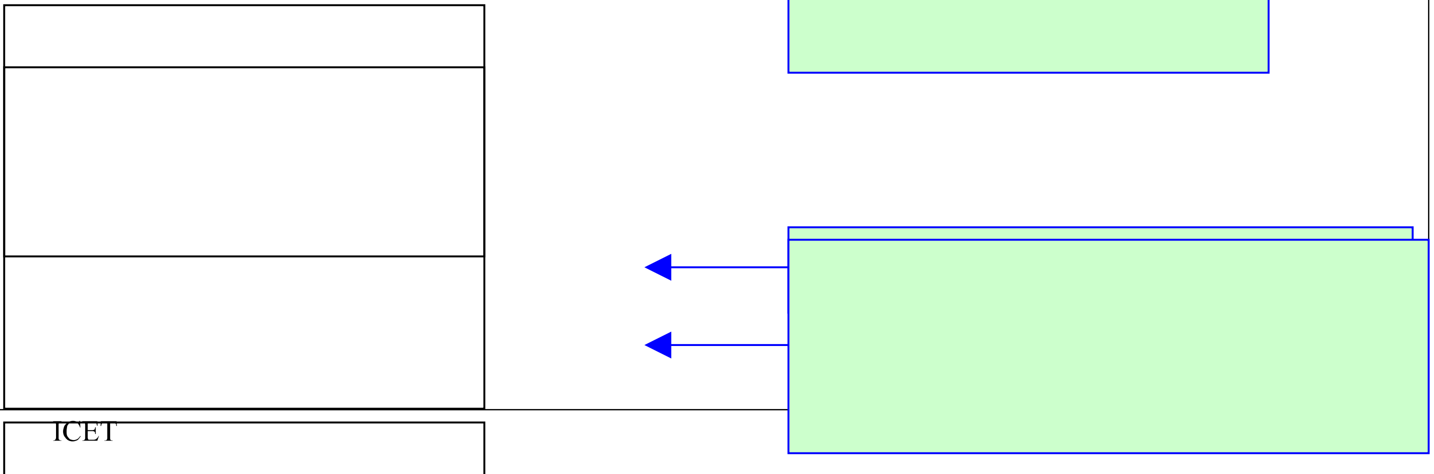
Java Versions

- JDK Alpha and Beta (1995)
- JDK 1.0 (1996)
- JDK 1.1 (1997)
- J2SE 1.2 (1998)
- J2SE 1.3 (2000)
- J2SE 1.4 (2002)
- J2SE 1.5 (2004)
- Java SE 6 (2006)
- Java SE 7 (2011)
- Java SE 8 (2014)

The Compilation Process for Non-Java Programs



The Compilation Process for Java Programs



Java Virtual Machine

How can bytecode be run on any type of computer?

■ As a Java program's bytecode runs, the bytecode is translated into object code by the computer's bytecode interpreter program. The bytecode interpreter program is known as the *Java Virtual Machine,* or *JVM* for short.

■ *Bytecode* is a highly optimized set of instructions designed to be executed by the Java run-time system, which is called the ***Java Virtual Machine* (*JVM*).**

■ JVM is a specification that provides runtime environment in which java bytecode can be executed.

■ JVMs are available for many hardware and software platforms (i.e. JVM is platform dependent).

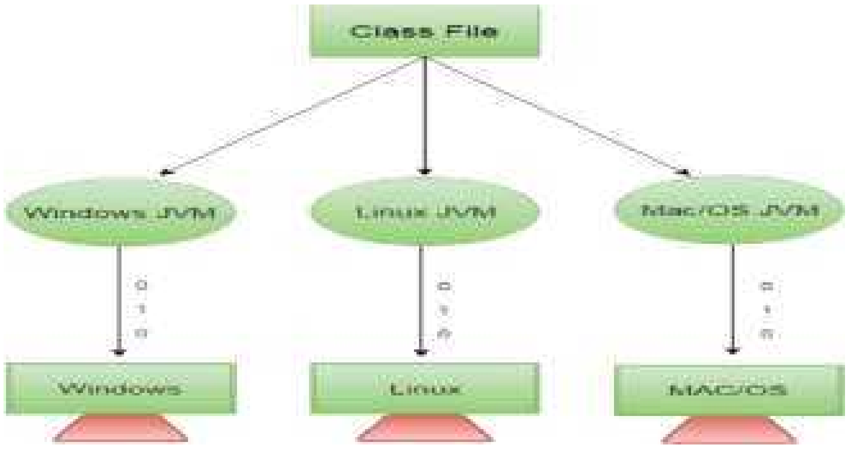
The JVM performs following operation:

- Loads code
- Verifies code
- Executes code
- Provides runtime environment

JVM provides definitions for the:

- Memory area
- Class file format
- Register set
- Garbage-collected heap

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).



<u>Features of Java</u>

- 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

Java Is Simple

- Java was designed to be easy for the programmer
- For Object-oriented programmers, learning Java will be even easier.
- Confusing concepts from C++ are left out of Java or implemented in a cleaner manner. Eg:explicit pointers, operator overloading etc
- In Java, there are a small number of clearly defined ways to accomplish a given task.

Java Is Object-Oriented

- Java is inherently object-oriented.
- Although many object-oriented languages began strictly as procedural languages, Java was designed from the start to be object-oriented
- The object model in Java is simple and easy to extend
- Object-oriented means we organize our software as a combination of different types of objects that incorporates both data and behavior

Java Is Distributed

- Distributed computing involves several computers working together on a network.
- Java handles TCP/IP protocols
- Networking capability is inherently integrated into Java, writing network programs is like sending and receiving data to and from a file.
- RMI and EJB are used for creating distributed applications.

Java Is Interpreted

- You need an interpreter to run Java programs.
- The programs are compiled into the Java Virtual Machine code called byte code.
- The byte code is machine-independent and can run on any machine that has a Java interpreter
- Byte code is easy to translate into machine code for very high performance using JIT compiler

Java Is Robust

Robust simply means strong. Java uses strong memory management. There is lack of pointers that avoids security problem.

There is automatic garbage collection in java. There is exception handling and type checking mechanism in java. All these points make java robust.

- Java compilers can detect many problems that would first show up at execution time in other languages.
- Java has eliminated certain types of error-prone programming constructs found in other languages.
- Java manages memory allocation and deallocation for you
- Java has a runtime exception-handling feature to provide programming support for robustness.

Java Is Secure

- Java implements several security mechanisms to protect your system against harm caused by stray programs.
- Automatic array bounds checking and the lack of manual memory management
- No use of pointers, Exception handling concept etc

Java Is Architecture-Neutral

- Write once, run anywhere
- With a Java Virtual Machine (JVM), you can write one program that will run on any platform.

Java Is Portable

Because Java is architecture neutral, Java programs are portable. They can be run on any platform without being recompiled.

Java's Performance

- The execution speed of Java programs improved significantly due to the introduction of Just-In Time compilation (JIT)
- The Just-In-Time (JIT) compiler is a component of the Java Runtime Environment. It improves the performance of Java applications by compiling bytecodes to machine code at run time
- They can be run on any platform without being recompiled.

Java Is Multithreaded

- Allows you to write programs that do many things simultaneously.

- Multithread programming is smoothly integrated in Java, whereas in other languages you have to call procedures specific to the operating system to enable multithreading

Java Is Dynamic

Java is considered as **Dynamic** because of Bytecode[a class file]. A source code written in one platform, the same code can be executed in any platform [which JDK is installed.]. And it also loads the class files at runtime

- Java programs carry with them substantial amount of run time information
- Java resolves accesses to object at runtime
- Allow dynamic linking

<u>A Simple Java Program – Hello.java</u>

<u>Example 1</u>

```
/*
  Hello World, first application, only output.
*/
import java.io.*;
class Hello
{
  public static void main (String args [])
      {
        System.out.println("Hello World\n");
      }  //end main
}//end class
```

How to get it running

Text in Hello.java file

To compile:

```
javac Hello.java
```

To run:

```
java Hello
```

Java is CASE SENSITIVE!!

File name has to be the same as class name in file.

Need to import necessary class definitions

All statements in Java end with a semicolon.

Whitespace is ignored by compiler

In Java, all code must reside inside a class.

Comments

Single line comments(//)

Multi line comments(*/ */)

Documentation comment(**/ */)

Java.io.*;

Similar to #include<stdio.h>

Access to all the classes defined in java.io

public keyword is an access modifier which represents visibility, it means it is visible to all.

static is a keyword, if we declare any method as static, it is known as static method. The core advantage of static method is that there is no need to create object to invoke the static method. The main method is executed by the JVM, so it doesn't require creating object to invoke the main method. So it saves memory.

String args[ ]

- String args[ ] declares a parameter named args, which is an array of instances of the class String.
- Objects of type String store character strings.

System.out.println("Hello World");

- This line outputs the string "Hello World" followed by a new line on the screen.
- Output is actually accomplished by the built-in println( ) method.
- In this case, println( ) displays the string which is passed to it.
- System is a predefined class that provides access to the system, and out is the output stream that is connected to the console.

Example 2

```
class Example2
{
  public static void main(String args[])
{
     int a;        // this declares a variable called a
    a = 100;   // this assigns a the value 100

     System.out.println("This is num: " +a);
    a = a * 2;
```

```
      System.out.print("The value of num * 2 is ");
      System.out.println(a);
  }
}
```

Java Reserved Words or Keywords

There are 49 reserved keywords currently defined in the Java language.These keywords ,combined with the syntax of the operators and seperators, from the definition of the Java language.These keywords cannot be used as names for a variable,class or methods.

byte	goto	package	private	this
abstract	for	new	protected	throw
boolean	float	native	public	throws
break	finally	long	return	transient
case	final	interface	short	try
catch	extends	int	static	void
char	else	instanceof	strictfp	volatile
class	double	import	super	while
const	do	implements	switch	
continue	default	if	synchronized	

byte, char ,int ,long,short,float,double and boolean are data types in Java

The java **instanceof** operator is used to test whether the object is an **instance of** the specified type

break- used to exit from the loop

continue- used to move control to the beginning of loop

The keywords **const** and **goto** are reserved but not used.

try, catch , throw ,throws and finally are associated with exception handling

package-A **java package** is a group of similar types of classes, interfaces and sub-packages.

interface- An interface is a reference type in Java. It is similar to class. It is a collection of abstract methods

final- It is used to make a variable as a constant.

extends-**extends** is the keyword used to inherit the properties of a class

implements-A class uses the **implements** keyword to implement an interface

private, protected and public are access specifiers

synchronized- Synchronization in java is the capability *to control the access of multiple threads to any shared resource*.

volatile- volatile keyword is used to mark a Java variable as "being stored in main memory"
this-There can be a lot of usage of **java this keyword**. In java, this is a **reference variable** that refers to the current object.
super-The **super** keyword in java is a reference variable which is used to refer immediate parent class object.

return-The **return keyword** is used to **return** from a method when its execution is complete.
assertion- Assertion is a statement in java. It can be used to test your assumptions about the program.
While executing assertion, it is believed to be true. If it fails, JVM will throw an error named AssertionError. It is mainly used for testing purpose.

new - It creates a **Java**object and allocates memory for it. **new** is also used for array creation, as arrays are also objects.
The "**abstract**" keyword can be used on classes and methods.
When a method is declared abstract, the method cannot have a definition
transient is a **Java** keyword which marks a member variable not to be serialized
When a method is marked as **native,** it cannot have a body and must ends with a semicolon
strictfp is a **keyword** in the **Java** programming language that restricts floating-point calculations.
using **strictfp** ensures result of floating point computations is always same on all platforms.
In addition to the keyword,Java reserved the following:true,false and null.These are the values defined by Java.

Identifiers

- An **identifier** is a word used by a programmer to name a variable, method, class, or label.
- Keywords and reserved words may not be used as identifiers.
- An identifier must begin with a ***letter, a dollar sign ($), or an underscore (_)***; subsequent characters may be letter, dollar signs, underscores, or digits.

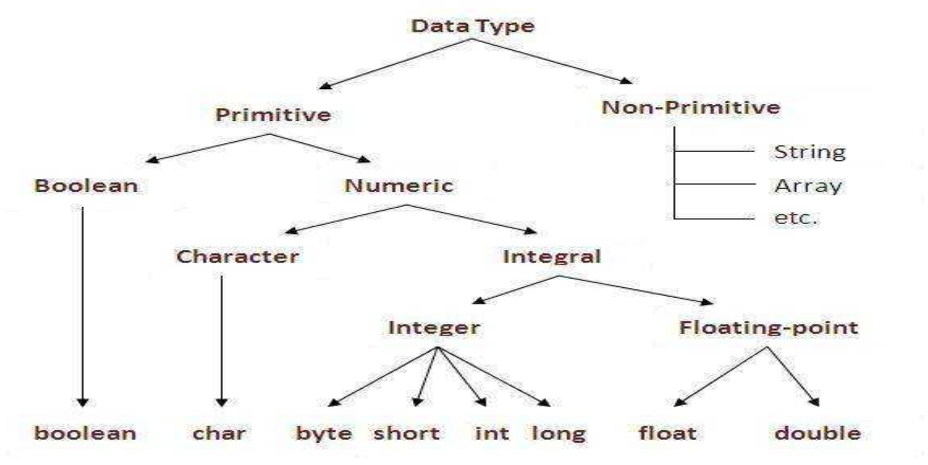
Data types in Java

Data types represent the different values to be stored in the variable. In java, there are two types of data types:

- Primitive data types
- Non-primitive data types

Primitive data types are predefined types of data, which are supported by the programming language.

Non-primitive data types are not defined by the programming language, but are instead created by the programmer. They are sometimes called "reference variables,since they reference a memory location, which stores the data.



Java defines eight simple data types **:** **byte**,**short**,**int**,**long**,**char**,**float**,**double**,and **boolean**.These can put in four groups:

Integers This group includes byte, short, int, and long, which are for whole valued signed numbers.

Floating-point numbers This group includes float and double, which represent numbers with fractional precision.

Characters This group includes char, which represents symbols in a character set, like letters and numbers.

Boolean This group includes boolean, which is a special type for representing true/false

Integer

byte

Variables of type **byte** are especially useful when you're working with **a stream of data from a network or file. Size- 1 byte**

short

It is probably the least-used Java type

This type is mostly applicable to 16-bit computers.

Size-2 byte

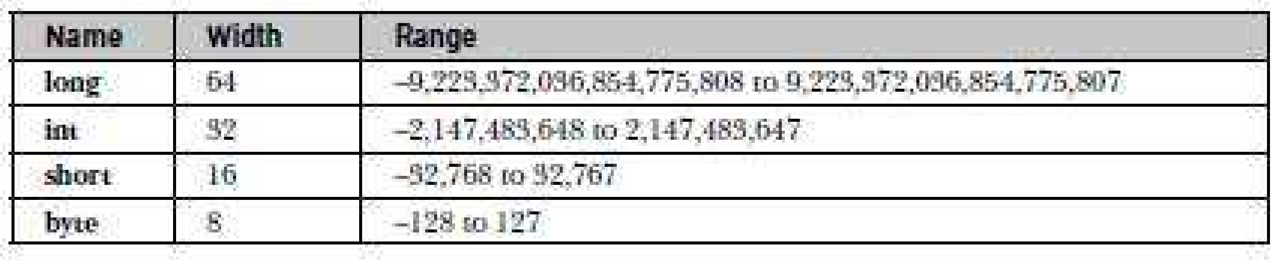
int

The most commonly used integer type is **int**.

Commonly employed to control loops and to index arrays. Size- 4 byte

long

Large enough to hold the desired value. Size- 8 byte



Name	Width	Range
long	64	–9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
int	32	–2,147,483,648 to 2,147,483,647
short	16	–32,768 to 32,767
byte	8	–128 to 127

<u>**Floating Point**</u>

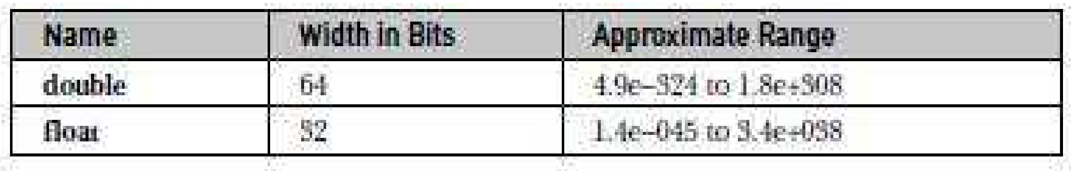
Float

Variables of type **float** are useful when you need a fractional component. Size – 4 byte

Double

All math functions, such as **sin( )**, **cos( )**, and **sqrt( )**, return **double** values.

- many iterative calculations, or are manipulating large-valued numbers, **double** is the best choice.
- Size- 8 byte



Name	Width in Bits	Approximate Range
double	64	4.9e–324 to 1.8e+308
float	32	1.4e–045 to 3.4e+038

<u>**Characters**</u>

- In C/C++, char is 1 byte wide.
- Java uses Unicode to represent characters. It requires 2 bytes.
- *Unicode* defines a fully international character set that can represent all of the characters found in all human languages
- The ASCII character set occupies the first 127 values in the Unicode character set.

<u>**Booleans**</u>

- Java has a simple type, called boolean, for logical values.
- It can have only one of two possible values, **true or false.**
- This is the type returned by all relational operators, such as a < b.

Demonstrate double variables

```
// Compute the area of a circle.
class Area
{
  public static void main(String args[])
{
  double pi, r, a;
  r = 10.8; // radius of circle
  pi = 3.14; // pi, approximately
    a = pi * r * r; // compute area
  System.out.println("Area of circle is " + a);
}
}
```

Demonstrate char variables

```
class Chardemo
{
   public static void main(String args[])
 {
       char ch1, ch2;
       ch1 = 88; // ASCII code for X
       ch2 = 'Y';
             System.out.print("ch1 and ch2: ");
       System.out.println(ch1 + " " + ch2);
   }
}
```

Output-

```
ch1 and ch2: X Y
```

<u>Variable</u>

The variable is the basic unit of storage in a Java program.

<u>Declaring a Variable</u>

type identifier [= *value*][, *identifier* [= *value*] ...] ;

- The *identifier* is the name of the variable.
- Eg:- int a,b,c;

  int d=3;

Dynamic Initialization

```
class Example
{
    public static void main(String args[])
     {
      double a = 3.0, b = 4.0;
            // c is dynamically initialized
      double c = Math.sqrt(a * a + b * b);
System.out.println("Hypotenuse is " + c);
   }
}
```

Scope and Lifetime of Variables

Java allows variables to be declared within any block.

A **block** is begun with an opening curly brace and ended by a closing curly brace. A block defines a *scope.*

The **scope** of a variable defines the section of the code in which the variable is visible.

In Java, the two major scopes are those **defined by a class and those *defined by a method*.**

The **lifetime** of a variable refers to how long the variable exists before it is destroyed.

<u>**Scope example**</u>

```
class Scope
{
  public static void main(String args[])
 {
   int x;                    // known to all code within main
   x = 10;
      if(x == 10)
      {    // start new scope
           int y = 20;    // known only to this block
           // x and y both known here.
          System.out.println("x and y: " + x + " " + y);
          x = y * 2;
       }
y = 100;          // Error! y not known here. x is still known here.
 System.out.println("x is " + x);
 }
}
```

<u>**Lifetime example**</u>

```
class LifeTime
```

```
{
   public static void main(String args[])
  {
     int x;
       for(x = 0; x < 3; x++)
      {
          int y = -1; // y is initialized each time block is entered
          System.out.println("y is: " + y); // this always prints -1
          y = 100;
          System.out.println("y is now: " + y);
     }
   }
}
```

Type conversion

Type Casting(Type conversion). Assigning a value of one type to a variable of another type is known as **Type Casting**.

Example : int x = 10; byte y = (byte)x;.

there are two kinds of **conversion**: implicit and explicit.

An implicit conversion means that a value of one **type** is changed to a value of another **type** without any special directive from the programmer.

Automatic Type casting take place when,

- the two types are compatible
- the target type is larger than the source type

Example :

```
public class Test
{
   public static void main(String[] args)
   {
     int i = 100;
     long l = i;          //no explicit type casting required
     float f = l;         //no explicit type casting required
     System.out.println("Int value "+i);
```

```
      System.out.println("Long value "+l);
      System.out.println("Float value "+f);
   }
  }
```

```
Output :
Int value 100
Long value 100
Float value 100.0
```

Narrowing or Explicit type conversion

When you are assigning a larger type value to a variable of smaller type, then you need to perform explicit type casting.

Example :

```
public class Test
{
   public static void main(String[] args)
   {
     double d = 100.04;
     long l = (long)d;  //explicit type casting required
     int i = (int)l;       //explicit type casting required

     System.out.println("Double value "+d);
     System.out.println("Long value "+l);
     System.out.println("Int value "+i);
      }
  }
```

```
Output :
Double value 100.04
Long value 100
Int value 100
```

Type Promotion Rules

All byte and short values are promoted to **int**

If one operand is long then the whole expression is promoted to **long**

If one operand is float then the whole expression is promoted to **float**

If one operand is double then the whole expression is promoted to **double**

Arrays

Collection of homogenous elements stored in adjacent locations

Arrays can be of any type and dimensions

Array elements can be accessed using an index

One dimensional array

Declaring an array

Syntax: type var-name[];

Eg: int a[];

Allocating an array

Syntax : var-name=new type[size];

Eg : a=new int[10];

Declaring and allocating

Eg: int a[] = new int[10];

These elements will be automatically initialized to Zero

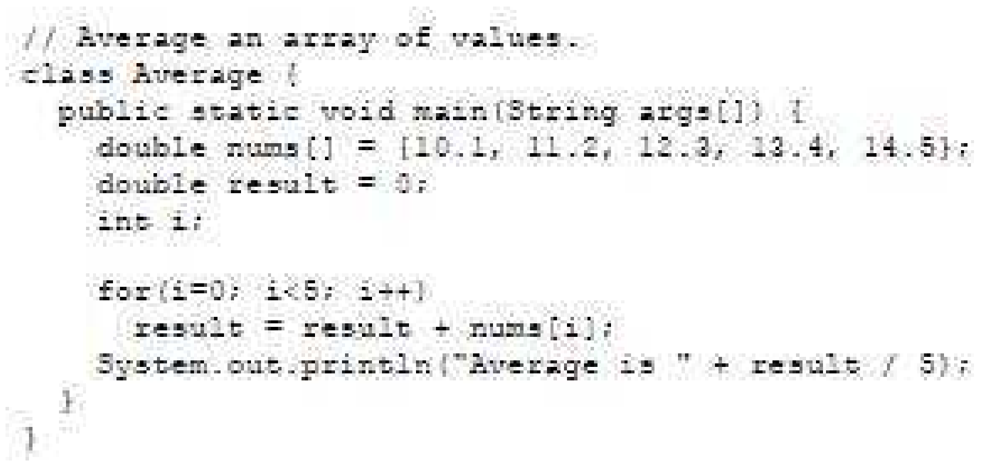
All array index starts at **zero**

Example

```
class Array
{
public static void main(String args[])
{
int i=0;
int a [];
a=new int[10];
a[0]=1;
a[1]=2;
a[2]=3;
for(i=0;i<3;i++)
   System.out.println(a[i]);
   }
}
```

Array initialization

int a[]={1,2,3,4};



```
// Average an array of values.
class Average {
  public static void main(String args[]) {
    double nums[] = {10.1, 11.2, 12.3, 13.4, 14.5};
    double result = 0;
    int i;

    for(i=0; i<5; i++)
      result = result + nums[i];
    System.out.println("Average is " + result / 5);
  }
}
```

<u>**Multi Dimensional Arrays**</u>

Array of arrays

Syntax : type array-name[][]=new type[row][col]

Eg : int a[][]=new int[4][5];

Initializing Multi Dimensional Array

int a[][]={ {1,2,3,4} , {5,6,7,8} };

Example

```
class Two
{
public static void main(String args[])
{
          int a[][]={  {1,2,3,4} , {5,6,7,8} };
int i , j ;
for(i=0;i<2;i++)
 for(j=0;j<4;j++)
 {
    System.out.println(a[i][j] + " ");
   }
}
}
```

Literals

A literal is a program element that directly represents a value

Integral literals may be expressed in decimal, octal, or hexadecimal.

To indicate octal, prefix the literal with 0 (zero)

To indicate hexadecimal, prefix the literal with 0x or 0X;

int decimal = 100;

int octal = 0144;

int hexa = 0x64;

Floating-Point Literals

They can be expressed in either **standard or scientific notation.**

Standard notation consists of a whole number component followed by a decimal point followed by a fractional component.

Eg:- 2.0, 3.14159, and 0.6667

Scientific notation uses a standard-notation, floating-point number plus a suffix that specifies a power of 10 by which the number is to be multiplied.

Eg:- 6.022E23, 314159E–5, and 2e+100.

Boolean Literals

Either takes the value true or false

Character Literal

can be expressed by enclosing the desired character in single quotes,

char c = ' w ';

String Literals

By enclosing a sequence of characters between a pair of double quotes.

Eg:- “Hello World”

Operators

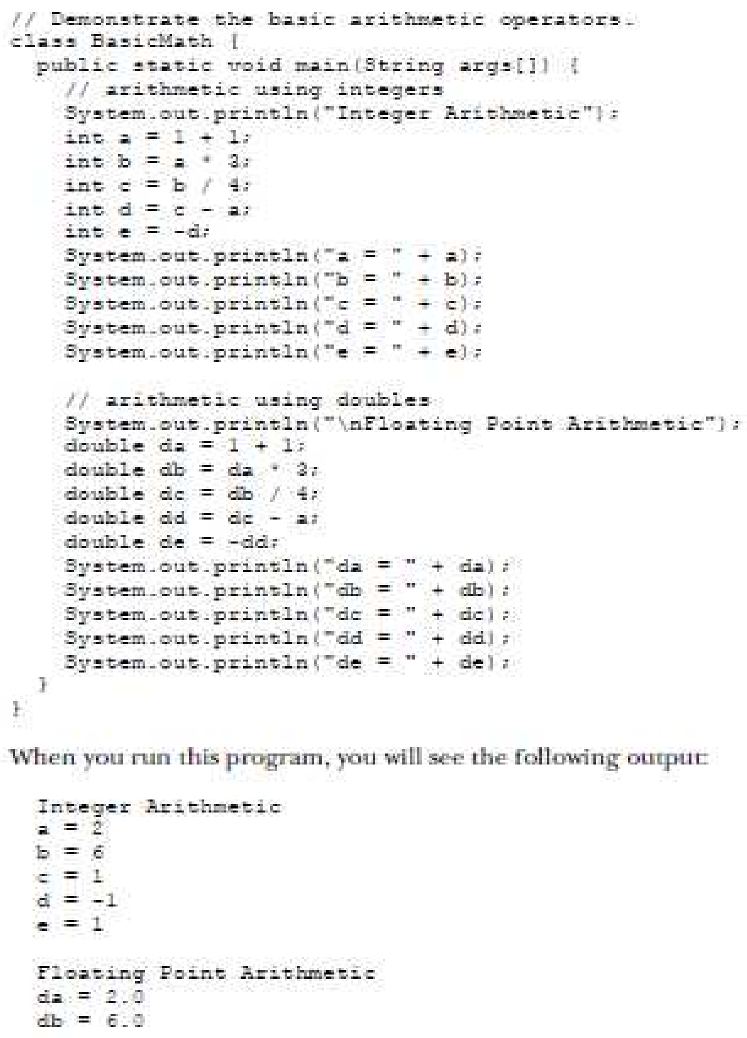
Arithmetic Operators

Operator	**Result**
+	Addition
-	Subtraction(Also Unary Minus)
*	Multiplications
/	Division
%	Modulus
++	Increment
+=	Addition Assignment
-=	Subtraction ‘’
*=	Multiplication

Modulus Operator

It can be applied to floating point as well as integers(in c/c++ only applicable to integers)

Eg: a+=b is equivalent to a=a+b



```
// Demonstrate the basic arithmetic operators.
class BasicMath {
  public static void main(String args[]) {
    // arithmetic using integers
    System.out.println("Integer Arithmetic");
    int a = 1 + 1;
    int b = a * 3;
    int c = b / 4;
    int d = c - a;
    int e = -d;
    System.out.println("a = " + a);
    System.out.println("b = " + b);
    System.out.println("c = " + c);
    System.out.println("d = " + d);
    System.out.println("e = " + e);

    // arithmetic using doubles
    System.out.println("\nFloating Point Arithmetic");
    double da = 1 + 1;
    double db = da * 3;
    double dc = db / 4;
    double dd = dc - a;
    double de = -dd;
    System.out.println("da = " + da);
    System.out.println("db = " + db);
    System.out.println("dc = " + dc);
    System.out.println("dd = " + dd);
    System.out.println("de = " + de);
  }
}
```

When you run this program, you will see the following output:

```
  Integer Arithmetic
  a = 2
  b = 6
  c = 1
  d = -1
  e = 1

  Floating Point Arithmetic
  da = 2.0
  db = 6.0
```

<u>Bitwise Operators</u>

Operator	**Result**
~	**Bitwise Unary NOT**
&	**Bitwise AND**
|	**Bitwise OR**
^	**Bitwise Exclusive OR**
>>	**Shift Right**
>>>	**Shift Right Zero Fill**

<<	**Shift Left**
&=	**Bitwise AND Assignment**
|=	**Bitwise OR „**
^=	**Bitwise Excusive OR „**
>>=	**Shift right „**
>>>=	**Shift right zero fill „**
<<=	**Shift left „**

The Left Shift

Syntax: value<<num;

Shifts all the bits in a value to the left a specified number of times

For each shift a high order bit is shifted out and a zero is brought in on the right

The right shift

Syntax: value>>num;

Shifts all the bits in a value to the right a specified number of times

For each shift a low order bit is shifted out and a zero is brought in on the left

But the MSB bit will not be changed to keep the sign

right Shift fill zero>>>

10111010>>4=11111011

10111010 >>> 4 = 00001011

Bitwise operator Assignment

Combines the assignment and bit wise operators

a>>=4 ie a=a>>4

a|=b   ie a=a|b

<u>Relational Operator</u>

To determine relationship that one operand has to the other

== , !=, >, <, >=, <=

Can be used with int,float,char and boolean

Result is a boolean

Boolean Logical Operators

Operator	**Result**
&	Logical AND
	Logical OR

^	Logical XOR
||	Short circuit OR
&&	Short Circuit AND
!	Logical Unary NOT
&=	AND assignment
|=	OR Assignment
^=	XOR Assignment
==	Equal to
!=	Not Equal to
?:	Ternary if then else

Short circuit Logical Operators

The **&& and || operators "short-circuit",** meaning they don't evaluate the right hand side if it isn't necessary.

The & and | operators, when used as logical operators, always evaluate both sides.

Eg

If(d!=0 && num/d>10)- there is no risk for run time exception when d=0.

If we use & , both sides have to be evaluated, causing run time exception.

Assignment Operator

variable=expression;

Variable must be compactable with expression

= can be used to create chain of assignments

int x,y,z;

x=y=z=100;

Ternary(Three way) operator

expression1? Expression2 : expression3;

If expression1 is true then expression2 is evaluated else expression3 will be evaluated

Control Statements

Cause the flow of execution

3 categories

Selection

Iteration

Jump

Selection Control Statement

Allows to choose different paths of execution based on o/p of an expression or state of a variable according to run time conditions

if

switch

<u>if statement</u>

```
if(condition)
    statement1;
 else
    statement2;
```

- Each statement may be single or compound enclosed in curly brackets
- Condition is an expression that returns a boolean value
- Else clause is optional

```
  Eg:- int a,b,c;
 if(a>b)
    c=a;
 else
    c=b;
```

If-else-if ladder

```
if(condition)
       statement;
     else if(condition)
      statement;
      .
      .
  else
    statement;
```

Executed from the top down

As soon as one if is found true the statement(s) will be executed and the rest is bypassed

If none is true statement(s) in else will be executed

<u>Switch Statement</u>

Multi way branch statement based on value of an expression

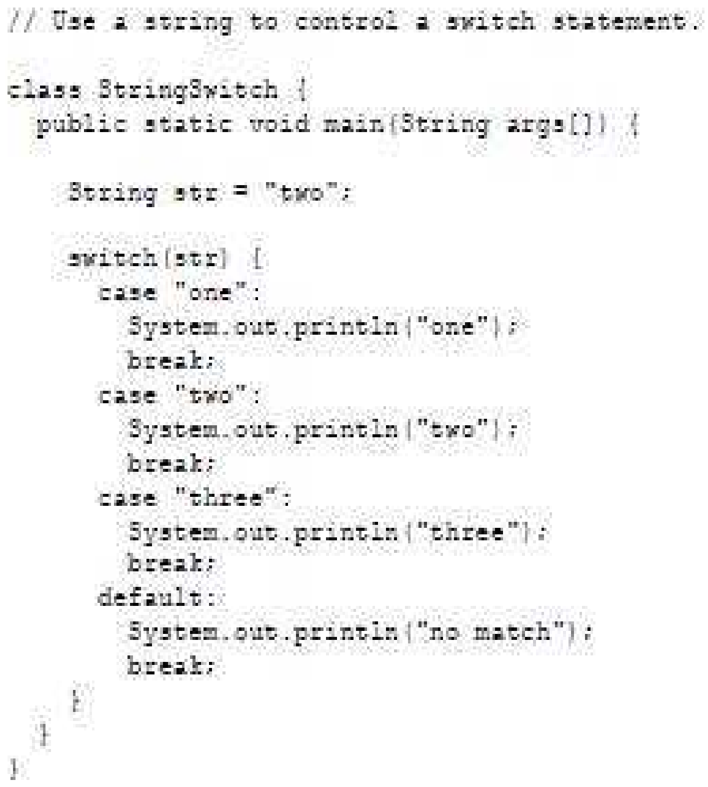
```
  switch(expression)
  {
   case value1:
    //statement sequence
   break;
  case value 2:
  // statement sequence
  break;
  .
  .
 case value n:
 // statement sequence
 break;
 default:
  //statement sequence
 }
```

Example

```
class Test
{
 public static void main(String args[])
 {
 int i;
 for(i=1;i<5;i++)
   switch(i)
   {
    case 1:  System.out.println("One");
             break;
    case 2:  System.out.println("Two");
             break;
    case 3:  System.out.println("Three");
             break;
    case 4:  System.out.println("Four");
             break;
    default: System.out.println("Five");
   }
 }
}
```

Example

```
class Test
{
  public static void main(String args[])
{
  int i;
  for(i=0;i<12;i++)
    switch(i)
    {
     case 0:
     case 1:
     case 2:
     case 3:
     case 4: System.out.println("i is less than 5");
     break;
     case 5:
     case 6:
     case 7:
     case 8:
     case 9: System.out.println("i is less than 10");
                break;
     default: System.out.println("i is 10 or more");
    }
  }
}
```



```
// Use a string to control a switch statement.

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

    String str = "two";

    switch(str) {
      case "one":
        System.out.println("one");
        break;
      case "two":
        System.out.println("two");
        break;
      case "three":
        System.out.println("three");
        break;
      default:
        System.out.println("no match");
        break;
    }
  }
}
```

Nested Switch

```
switch(cond)
{
 case 1:
     switch(cond)
      {
        case 0:  ----
                     break;
        case 1:
                    ----
                     break;
      }
       break;
 case 2:
//……………………………………………………………
}
```

Comparison with if

Switch can only test for equality whereas if can evaluate any type of boolean expression

No two case statements can be identical in a switch block

Switch statement is more efficient than set of nested ifs

Switch is faster because it can perform equality of case constants and expression and both are of same type

<u>**Iteration Statements**</u>

Loops repeatedly executes the same set of instructions until a termination condition is met

while and do- while

for

while

The most fundamental loop – ***Entry controlled loop***

```
while(condition)
{
   //body of loop
}
```

- Condition can be any boolean expression
- Body of the loops is executed as long as the condition is true

do while

Exit control loop

```
 do
{
   // body of loop
} while(condition);
```

Used for menu driven programs

Loop will display the menu once then user gives the choice

For loop

Count control loop

```
 for(initialization;condition;iteration)
{
// body of loop
}
```

Possible to declare the variable inside the initialization portion of for

Eg:-   for(int i=0; i<10;i++)

```
                        {
                        -----
                        }
```

Scope of this variable is limited to the for block

It is possible to include more than one statement in initialization and iteration part of for loop

```
class Comma {
public static void main(String args[])
{
   int first, second ;
  for ( first = 0, second = 10 ; first < second ; first++, second-- )
  {
    System.out.println( first + " " + second ) ;
  }
}
```

Some loop variations

Condition can be any boolean expression

```
 boolean done=false;
 int i=0;
for(;!done;)
 {
    System.out.println(i);
    if(i==10)
      done=true;
   i++;
}
```

Infinite loop

```
for(; ;)
     {
        //body of the loop
      }
```

Nested loops

```
for(i=0;i<m;i++)
  for(j=i;j<n;j++)
    {
```

```
    //body of the inner loop
   }
```

<u>**Jump Statements**</u>

Break, continue and return

break statement has three uses

1. It terminates a statement sequence in switch
2. It can be used to exit a loop
3. It can be used as goto

Java does not have goto statement

Instead break can be used to tell where the execution exactly should resume

Labeled break

Syntax :- break label;

Label is a java identifier

Continue

Continue the loop but by pass the remaining code in the body of loop for the current iteration

```
class Test
{
public static void main(String args[])
{
  for(int i=0;i<10;i++)
  {
    if(i%2!=0)
       continue;
    System.out.println(i);
  }
 }
}
```

Return

Explicitly return from a method/function to the caller function

```
class return
{
 public static void main(String args[])
 {
   boolean t=true;
```

```
  System.out.println("Before return");
  if(t)
   return;
 System.out.println("This wont execute!");
}
}
```

Here control is transferred to Java run time system since main is called by Java Run time system