2016

Learn Java with Shamik

Shamik Mitra

Java on fly. A guide for novice to pro

5/31/2016

Written by: **Shamik Mitra**
Technical Leader at IBM
blog: http://javaonfly.blogspot.in/
facebook : https://www.facebook.com/shamik.mitra.37
mob : **9830471739** mail **:mitrashamik@gmail.com**
Provide Tuition on java, j2ee Struts,Spring,Hibernate,jpa,Hadoop,Devops,Agile,kanban

Learn Java with Shamik

About Me:

Programming is a tool where you can blend your Aritstic sense with Inteligence.I dedicate this book to all my fellow developer or novice who wants know the Java. I try to cover all the tricky situation you can come up while doing programming.

I DEDICATE THIS BOOK TO MY MOTHER LATE SUBHRA MITRA.

Written by: **Shamik Mitra**
Technical Leader at IBM
blog: http://javaonfly.blogspot.in/
facebook : https://www.facebook.com/shamik.mitra.37
mob : **9830471739** mail **:mitrashamik@gmail.com**
Provide Tuition on java, j2ee Struts,Spring,Hibernate,jpa,Hadoop,Devops,Agile,kanban

I have 9 years of professional experience in Java. Currently I work for IBM as a Technical leader. During my service life I was awarded "Eminence &Excellence"," Spark award" "Deep Skill" in IBM. Won the "Jury Award" in Techathon Coding competition which was held over all location in IBM India. I love to share my experience with you so you can learn it very effective way.

I am open to teach one-to-one or group classes. I prefer Skype or Google hangout as online tutoring tool.

If you have any questions you can direct mail me or follow me in Facebook or write it in my blog.

Email:mitrashamik@gmail.com

skype: mitrashamik

facebook:https://www.facebook.com/shamik.mitra.37

blog:http://javaonfly.blogspot.in/

DOWNLOAD LINK

You can download all programs written in this book

Link:https://github.com/shami83/JavaAssignment.git

Then click on clone pr download then download the zip.

Written by: **Shamik Mitra**
Technical Leader at IBM
blog: http://javaonfly.blogspot.in/
facebook : https://www.facebook.com/shamik.mitra.37
mob : **9830471739** mail **:mitrashamik@gmail.com**
Provide Tuition on java, j2ee Struts,Spring,Hibernate,jpa,Hadoop,Devops,Agile,kanban

Written by: **Shamik Mitra**
Technical Leader at IBM
blog: http://javaonfly.blogspot.in/
facebook : https://www.facebook.com/shamik.mitra.37
mob : **9830471739** mail **:mitrashamik@gmail.com**
Provide Tuition on java, j2ee Struts,Spring,Hibernate,jpa,Hadoop,Devops,Agile,kanban

Chapter 1

Core Java Understanding

What is Java?

Java is a high-level programming language originally developed by Sun Microsystems and released in 1995. Java runs on a variety of platforms, such as Windows, Mac OS, and the various versions of UNIX. So java is a platform independent language. This tutorial gives a complete understanding of Java.

Java Latest Release and Founder/History

The latest release of the Java Standard Edition is Java SE 8. With the advancement of Java and its widespread popularity, multiple configurations were built to suite various types of platforms. Ex: J2EE for Enterprise Applications, J2ME for Mobile Applications.

The new J2 versions were renamed as Java SE, Java EE and Java ME respectively. Java is guaranteed to be **Write Once, Run Anywhere.**

James Gosling initiated the Java language project in June 1991 for use in one of his many set-top box projects. The language, initially called **Oak** after an oak tree that stood outside Gosling's office, also went by the name **Green** and ended up later being renamed as **Java**, from a list of random words.

Sun released the first public implementation as Java 1.0 in 1995.

What is Object Oriented Language/OOPS:

Object means a real word entity such as pen, chair, table, Bottle etc. Object-Oriented Programming is a methodology or paradigm to design a program using classes and objects. It simplifies the software development and maintenance by providing some concepts:

Written by: **Shamik Mitra**
Technical Leader at IBM
blog: http://javaonfly.blogspot.in/
facebook : https://www.facebook.com/shamik.mitra.37
mob : **9830471739** mail **:mitrashamik@gmail.com**
Provide Tuition on java, j2ee Struts,Spring,Hibernate,jpa,Hadoop,Devops,Agile,kanban

- Object
- Class
- Inheritance
- Polymorphism
- Abstraction
- Encapsulation

Java satisfies all above concept so java is an Object Oriented Programming Language.

Object

Any entity that has state /properties and behaviour/methods is known as an object. For example: chair, pen, table, keyboard, bike etc. It can be physical and logical. Object is real-time entity in **java world so called,JVM(Java Virtual Machine)**.

In Physics object is which take some places which has properties and which can do some actions Same in Java, Object takes memory space i.e. space in RAM it has properties and it can do some actions like, by**pen we can write so write is an action ,in java term we call it method.**

If we consider a dog, then its state/properties is - name, breed, colour, and the behaviour /methods is - barking, wagging, running

Class

Class is Logical entity or we can call it blueprint. Like for each house there is a plan for that house. So plan is Class and the House itself is an Object

Inheritance

When one object acquires all the properties and behaviours of parent object i.e. known as inheritance. It provides code reusability. It is used to achieve runtime polymorphism.

For Examples, All Vehicle must have wheels and BMW has 4 wheels so If Vehicles is **parent** then **BMW** is child and it inherits wheels property.

Written by: **Shamik Mitra**
Technical Leader at IBM
blog: http://javaonfly.blogspot.in/
facebook : https://www.facebook.com/shamik.mitra.37
mob : **9830471739** mail **:mitrashamik@gmail.com**
Provide Tuition on java, j2ee Struts,Spring,Hibernate,jpa,Hadoop,Devops,Agile,kanban

Polymorphism

When **one task is performed by different ways** i.e. known as polymorphism.

In java, we use method **overloading** and method **overriding** to achieve polymorphism.

example can be to speak behaviour e.g. cat speaks meow, dog barks woof etc.

Abstraction

Hiding internal details and showing functionality is known as abstraction. For example: driving a car, we know how to drive but don't know how the **internal mechanism** actually moves the car.

In java, we use abstract class and interface to achieve abstraction.

Encapsulation

Binding (or wrapping) code and data together into a single unit is known as encapsulation. For example: capsule, it is wrapped with different medicines.

In Java Class wrap properties and methods.

Installing Java:

Download Java from here

http://www.oracle.com/technetwork/java/javase/downloads/index.html

Download Eclipse from here

http://www.eclipse.org/downloads/packages/eclipse-ide-java-developers/marsr

Install java and extract eclipse folder on your local folder

Now open eclipse

Go to File->JavaProject->create project

Create project and happy coading.

Written by: **Shamik Mitra**
Technical Leader at IBM
blog: http://javaonfly.blogspot.in/
facebook : https://www.facebook.com/shamik.mitra.37
mob : **9830471739** mail **:mitrashamik@gmail.com**
Provide Tuition on java, j2ee Struts,Spring,Hibernate,jpa,Hadoop,Devops,Agile,kanban

Chapter 2

Java Basic Syntax of a Class

package<dot separated name> : optional (Address of the class)

import <fqdn of class want to use >:(address of class which are going to used in this class)

<Access Specifier/modifiers><non access modifirs>*class*<Name of Class>

{

<Access Specifier/modifiers><type of property><propertyName>=<intialization value>;

<Access Specifier/modifiers><return Type/void><method Name>(<type of property><propertyName> can be 1...n)

{

Written by: **Shamik Mitra**
Technical Leader at IBM
blog: http://javaonfly.blogspot.in/
facebook : https://www.facebook.com/shamik.mitra.37
mob : **9830471739** mail **:mitrashamik@gmail.com**
Provide Tuition on java, j2ee Struts,Spring,Hibernate,jpa,Hadoop,Devops,Agile,kanban
//business logic goes here

return<return Type>;

}//end of method

}//end of class

Italics : denotes java Keywords

Now First Java Program:

HelloWorld.java

```
package com.example


public class HelloWorld
{



private String msg=”Hello Shamik !!!”
public String getMessage()
{
return msg;
}


public static void main(String args[])//entry point , To run the program it is requred.
{
HelloWorld world = new HelloWorld(); //create a HelloWorld object and   assign  it with world refrence


String msg = world.getMessage();//call method by help of refrence .(dot) Operator use in java to call methods.
System.out.println(msg);
}
```

Written by: **Shamik Mitra**
Technical Leader at IBM
blog: http://javaonfly.blogspot.in/
facebook : https://www.facebook.com/shamik.mitra.37
mob : **9830471739** mail **:mitrashamik@gmail.com**
Provide Tuition on java, j2ee Struts,Spring,Hibernate,jpa,Hadoop,Devops,Agile,kanban

}

Now Right click on the class and go to Run As and click on run as a java application

You will see the output

Hello Shamik!!!

Discussing About Syntax

A class is consisting of properties and methods.

Methods - A method is basically a behaviour or we can think it as a verb. A class can contain many methods. methodis the place where the business logics are written, data is manipulated and all the actions are executed.

Instance Variables /properties/member variables- Each object has its unique set of instance variables/member variables. An object's state is created by the values assigned to these instance variables/properties. In above example msg is member cariable.

Java Best practises and Key things to remember

Case Sensitivity - Java is case sensitive, which means identifier **MSG** and **msg** would have different meaning in Java.

Class Names -class name’s first letter should be in Upper Case.

If several words are used to form a name of the class, each inner word's first letter should be in Upper Case. Like HelloWorld

Written by: **Shamik Mitra**
Technical Leader at IBM
blog: http://javaonfly.blogspot.in/
facebook : https://www.facebook.com/shamik.mitra.37
mob : **9830471739** mail **:mitrashamik@gmail.com**
Provide Tuition on java, j2ee Struts,Spring,Hibernate,jpa,Hadoop,Devops,Agile,kanban

Method Names - All method names should start with a Lower Case letter.

If several words are used to form the name of the method, then each inner word's first letter should be in Upper Case. We called it camelCase.

Like getMessage()

Keywords- Java has reserved keywords so you can't use them as method or variable name. Like class is reserved keyword so you can't use it as a variable name/method name.

Program File Name - Name of the program file should exactly match the public class name.

If File name is **HelloWorld** then public class name should be **HelloWorld.** If file does not contain any public class, then name is immaterial. A file can has only one public class.

Java Identifiers - We call it as properties. Rule as follows

All identifiers should begin with a letter (A to Z or a to z), currency character ($) or an underscore

(_).

After the first character identifiers can have any combination of characters.

A key word cannot be used as an identifier.

Most importantly identifiers are case sensitive.

Examples of legal identifiers: age, $salary, _value, __1_value

Examples of illegal identifiers: 123abc, -salary

Java Modifiers -

There are two categories of modifiers:

- **Access Modifiers:** default, public, protected, private
- **Non-access Modifiers:** final, abstract, strictfp
-

Written by: **Shamik Mitra**
Technical Leader at IBM
blog: http://javaonfly.blogspot.in/
facebook : https://www.facebook.com/shamik.mitra.37
mob : **9830471739** mail **:mitrashamik@gmail.com**
Provide Tuition on java, j2ee Struts,Spring,Hibernate,jpa,Hadoop,Devops,Agile,kanban

Java Variables -

We would see following type of variables in Java:

- Local Variables
- Class Variables (Static Variables/Global variables/static)
- Instance Variables (Non-static variables)

Java Arrays:

Arrays are objects that store multiple variables of the same type. However, an array itself is an object on the heap.

Enums were introduced in java 5.0. Enums restrict a variable to have one of only a few predefined values. The values in this enumerated list are called enums.

For example, if we consider an application for a Shoe shop, it would be possible to restrict the size to small, medium and large. This would make sure that it would not allow anyone to order any size other than the small, medium or large.

Example

```
Class ShoeShop {

   enum ShoeSize{ SMALL, MEDIUM, LARGE }
   public ShoeSize size;
}

public class ShoeShopTest {

   public static void main(String args[]){
      ShoeShop shop = new ShoeShop();
      shop.size = ShoeShop.ShoeSize.LARGE ;
      System.out.println("Shoe Size is: " + shop.size.toString());
   }
}
```

Java Keywords

Written by: **Shamik Mitra**
Technical Leader at IBM
blog: http://javaonfly.blogspot.in/
facebook : https://www.facebook.com/shamik.mitra.37
mob : **9830471739** mail **:mitrashamik@gmail.com**
Provide Tuition on java, j2ee Struts,Spring,Hibernate,jpa,Hadoop,Devops,Agile,kanban

Java has following reserved keywords

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

How to Define a class in Java

A class is a blue print from which individual objects are created.

A sample of a class is given below:

```
public class TechnoStudent{
   public static String  collegeName=”Techno India”;//Class variable/global
   private String name;//member variable/instance variable
```

Written by: **Shamik Mitra**
Technical Leader at IBM
blog: http://javaonfly.blogspot.in/
facebook : https://www.facebook.com/shamik.mitra.37
mob : **9830471739** mail **:mitrashamik@gmail.com**
Provide Tuition on java, j2ee Struts,Spring,Hibernate,jpa,Hadoop,Devops,Agile,kanban

```
  private int age

  public void getName(){
    return name;
  }

  public void getAge(){
        return age;
  }
public String getStream()
{

 String stream = “CSE”;//local variable
 return stream
}


}
```

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. Ex: stream in above example

- **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. Ex: age
- **Class variables:** Class variables are variables declared with in a class, outside any method, with the static keyword. It will be share by all instances of this class Ex: collegeName

Constructor

Every class has a constructor. If we do not explicitly write a constructor for a class, the Java compiler builds a default constructor for that class.

Each time a new object is created, at least one constructor will be invoked. The main rule of **constructors is that they should have the same name as the class. A class can**

Written by: **Shamik Mitra**
Technical Leader at IBM
blog: http://javaonfly.blogspot.in/
facebook : https://www.facebook.com/shamik.mitra.37
mob : **9830471739** mail **:mitrashamik@gmail.com**
Provide Tuition on java, j2ee Struts,Spring,Hibernate,jpa,Hadoop,Devops,Agile,kanban

have more than one constructor. It looks like a method but without return type. If you define constructor explicitly then compile does not injected constructor

Example

```
public class Teacher{
  public Teacher(){
  }

  public Teacher(String name){
    // This constructor has one parameter, name.
  }
}
```

Object Creation

As mentioned previously, a class provides the blueprints for objects. So basically an object is created from a class. In Java, the new key word is used to create new objects.

There are three steps when creating an object from a class:

- **Declaration:** A variable declaration with a variable name with an object type. We call it refrence so we can link the create object. It is like pointer by which we can access the created objects. Like the phone number is a pointer of a person to access.
- **Instantiation:** The 'new' key word is used to create the object. So we can think of it as labour who will actually build the object in jvm from the blue print/class.
- **Initialization:** The 'new' keyword is followed by a call to a constructor. This call initializes the new object. Actually here objects acquire memory in RAM.

Example:

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


   // Following statement would create an object Teacher
     Teacher teacher = new Teacher( "Shamik" );
  }
```

here teacher is reference of newly created object whose name is Shamik.

Written by: **Shamik Mitra**
Technical Leader at IBM
blog: http://javaonfly.blogspot.in/
facebook : https://www.facebook.com/shamik.mitra.37
mob : **9830471739** mail **:mitrashamik@gmail.com**
Provide Tuition on java, j2ee Struts,Spring,Hibernate,jpa,Hadoop,Devops,Agile,kanban

Java Class and File's Contract

There can be only one public class per source file.

(TIP : Best practice...use one class for each 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 Teacher{}* then the source file should be as Teacher.java.
- If the class is defined inside a package, then the package statement should be the first statement in the source file. **Package is used for avoid namespce collition**
- If i**mport 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 says about which classes we are going to use.Remember when a class is outside of the package of **declaration class** for that class **import** statement required.
- 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.

DataTypes

There are two data types available in Java:

- Primitive Data Types, **int,char,float,double,boolean,short,long-- default value is 0**
- Reference/Object Data Types

Reference variables are created using defined **constructors of the classes**. They are used to access objects. These variables are declared to be of a specific type that cannot be changed. For example, Student, Teacher etc.

- Class objects, and various type of array variables come under reference data type.
- Default value of any reference variable is **null**.

Written by: **Shamik Mitra**
Technical Leader at IBM
blog: http://javaonfly.blogspot.in/
facebook : https://www.facebook.com/shamik.mitra.37
mob : **9830471739** mail **:mitrashamik@gmail.com**
Provide Tuition on java, j2ee Struts,Spring,Hibernate,jpa,Hadoop,Devops,Agile,kanban

- A reference variable can be used to refer to any object of the declared type or any compatible type.
- Example: Animal animal = new Animal("Tiger");

A literal is a source code representation of a fixed value. They are represented directly in the code without any computation.

Literals can be assigned to any primitive type variable.

int a=10;

char a='A' // here A is literal

String literal

String name=" Shamik" // here Shamik is literal.

Java Modifiers:

Java has two kind of modifiers

1. Java Access Modifiers.

2. Non access Modifiers.

Java Access Modifiers:

Java has 3 Access modifiers but 4 access level.

3 modifiers are **private,protected,public**

4 access levels are **private,protected,public,default**

If we not mention any access modifiers, then it is known as default in java.

Ex:**String name**; or **String getName()**

Written by: **Shamik Mitra**
Technical Leader at IBM
blog: http://javaonfly.blogspot.in/
facebook : https://www.facebook.com/shamik.mitra.37
mob : **9830471739** mail **:mitrashamik@gmail.com**
Provide Tuition on java, j2ee Struts,Spring,Hibernate,jpa,Hadoop,Devops,Agile,kanban

Private: Most restricted modifiers only accessible in the class which creates it.

Default:Accessible by own class and by the other classes of same package of itsowner class.

Protected:Accessible by own class and by the other classes of same package of itsowner class. All subclasses of its owner class in differentpackage**via Inheritance**.

Public:Accessible by all.

Visibility Matrix

	Own class	Own package	Subclass in Defferent Package	Defferent Package
Private	Yes	No	No	No
Default	Yes	Yes	No	No
Protected	Yes	Yes	Yes(**Inheritence**)	No
Public	Yes	Yes	Yes	Yes

Non access Modifiers

mainly in java we have following non access modifiers

1. Static

2. Final

3. Volatile

4. Synchronize

Static: Static is a java keyword we use it for class level variables or methods. When we need a property or method which we want to invoke without object/instance we make them static remember static method and property is not associated with object /instance .it

associated with class and **only one copy of this static property/method is share among all objects of this class.**

Written by: **Shamik Mitra**
Technical Leader at IBM
blog: http://javaonfly.blogspot.in/
facebook : https://www.facebook.com/shamik.mitra.37
mob : **9830471739** mail **:mitrashamik@gmail.com**
Provide Tuition on java, j2ee Struts,Spring,Hibernate,jpa,Hadoop,Devops,Agile,kanban

Let's take an example**,we want to count number of objects created from a class.**

We know when an object is created its constructor is always called so If we use a Static counter and for each object creation increase the counterthen we cancountthe object instances.

Here is the program

```
package com.example.staticTest;

publicclass ObjectTracking {

	publicstaticintCOUNTER=0;

	public ObjectTracking()
	{
		COUNTER++;
	}

	publicstaticvoid displayCount()
	{
		System.out.println("Number of Objects in JVM " + COUNTER);
	}

	publicstaticvoid main(String[] args) {

		for(inti=0;i<10;i++)
		{
			new ObjectTracking();
		}

		ObjectTracking.displayCount();//call displayCount as static
	}

}
```

Output: Number of Objects in JVM 10

Written by: **Shamik Mitra**
Technical Leader at IBM
blog: http://javaonfly.blogspot.in/
facebook : https://www.facebook.com/shamik.mitra.37
mob : **9830471739** mail **:mitrashamik@gmail.com**
Provide Tuition on java, j2ee Struts,Spring,Hibernate,jpa,Hadoop,Devops,Agile,kanban
Please pay attention to the line ObjectTracking.*displayCount*()

we call *displayCount*() method without create a object but with Class name

it is possible because static is attach to class not object.

Please please remember

to call static we need classname no need object refrence

<Class Name>.<Static Method/Property>

TIP: We also call static variable through object reference but it is a bad practice and compiler will changereference to class name don't get confused it is not object's property.

```
Class StaticTest
{
public static String CLASS_NAME=”StaticTest”

publicstaticvoid main(String[] args) {
                StaticTest test = new StaticTest();
        System.out.println(test. CLASS_NAME);// test will replace by       Static Test

        }
}
```

Here test. CLASS_NAME replaced by StaticTest.CLASS_NAME

Good practice is always creating Static variable name in Capital Letters

if multi word then put _ beetween them

so CLASS_NAME is good practice but className or CLASSNAME is bad practice.

Written by: **Shamik Mitra**
Technical Leader at IBM
blog: http://javaonfly.blogspot.in/
facebook : https://www.facebook.com/shamik.mitra.37
mob : **9830471739** mail **:mitrashamik@gmail.com**
Provide Tuition on java, j2ee Struts,Spring,Hibernate,jpa,Hadoop,Devops,Agile,kanban

Final: Final is a keyword in java. It can be used in three level

a. Variable and Reference

b. Method

c. Class

Variable and Reference:

A final variable can be explicitly initialized only once. A reference variable declared final can never be reassigned to refer to a different object. However, you can change the state of the object. So don't fool with this.

TIP: You can't reassign reference variable to another object but you can change the object State.

With variables, the *final* modifier often is used with *static* to make the constant a class variable.

Method:

A final method cannot be overridden by any subclasses. final modifier prevents a method from being modified in a subclass.

The main intention of making a method final would be that, the content of the method should not be changed by any outside class.

Class: A final class cannot be extended. If a class is marked as final, then no class can inherit any feature from the final class.

Example :

```
package com.example.finalTest;
```

Written by: **Shamik Mitra**
Technical Leader at IBM
blog: http://javaonfly.blogspot.in/
facebook : https://www.facebook.com/shamik.mitra.37
mob : **9830471739** mail **:mitrashamik@gmail.com**
Provide Tuition on java, j2ee Struts,Spring,Hibernate,jpa,Hadoop,Devops,Agile,kanban

```
import java.util.ArrayList;
import java.util.List;

publicfinalclass  FinalTest {

        publicfinalintscore =10;

        publicfinal List<String>greet = new ArrayList<String>();



        publicfinalvoid reassign()
        {
                // score=80; it can't be reassign as it is final
                // greet = new ArrayList<String>(); new ArrayList<String>();
                greet.add("able to change the arraylist state");
        }

}
```

Here FinalTest class marked as final so no one can extend it and allow to change the content of the class.

Method **reassign()** is final so it can't be overridden in subclass (if FinalTest is not final.)

Score is final and assign a value 10. So it can't be reassigning again if you do that compiler will complain.

It is an example of Immutable Class.

Volatile and Synchronize will be discussed later. When we will discuss about Thread.

Written by: **Shamik Mitra**
Technical Leader at IBM
blog: http://javaonfly.blogspot.in/
facebook : https://www.facebook.com/shamik.mitra.37
mob : **9830471739** mail **:mitrashamik@gmail.com**
Provide Tuition on java, j2ee Struts,Spring,Hibernate,jpa,Hadoop,Devops,Agile,kanban

Chapter 3

Java Basic Operators

Java provides a rich set of operators to manipulate variables. We can divide all the Java operators into the following groups:

- Arithmetic Operators
- Relational Operators
- Bitwise Operators
- Logical Operators
- Assignment Operators
- Misc Operators

Arithmetic Operators:

If A=20 qnd B=10 then

	+ (Addition)
1	Adds values on either side of the operator **Example:** A + B will give 30
	- (Subtraction)
2	

Written by: **Shamik Mitra**
Technical Leader at IBM
blog: http://javaonfly.blogspot.in/
facebook : https://www.facebook.com/shamik.mitra.37
mob : **9830471739** mail **:mitrashamik@gmail.com**
Provide Tuition on java, j2ee Struts,Spring,Hibernate,jpa,Hadoop,Devops,Agile,kanban

	Subtracts right hand operand from left hand operand **Example:** A - B will give 10
3	*** (Multiplication)** Multiplies values on either side of the operator **Example:** A * B will give 200
4	**/ (Division)** Divides left hand operand by right hand operand **Example:** A/B will give 2
5	**% (Modulus)** Divides left hand operand by right hand operand and returns remainder **Example:** A % B will give 0
6	**++ (Increment)** Increases the value of operand by 1 **Example:** B++ gives 11
7	**-- (Decrement)** Decreases the value of operand by 1 **Example:** B-- gives 9

- **Relational Operators:**

If A=20 qnd B=10 then

1	**== (equal to)**

Written by: **Shamik Mitra**
Technical Leader at IBM
blog: http://javaonfly.blogspot.in/
facebook : https://www.facebook.com/shamik.mitra.37
mob : **9830471739** mail **:mitrashamik@gmail.com**
Provide Tuition on java, j2ee Struts,Spring,Hibernate,jpa,Hadoop,Devops,Agile,kanban

	Checks if the values of two operands are equal or not, if yes then condition becomes true. **Example:** (A == B) is not true. **!= (not equal to)**
2	Checks if the values of two operands are equal or not, if values are not equal then condition becomes true. **Example:** (A != B) is true. **> (greater than)**
3	Checks if the value of left operand is greater than the value of right operand, if yes then condition becomes true. **Example:** (A > B) is true. **< (less than)**
4	Checks if the value of left operand is less than the value of right operand, if yes then condition becomes true. **Example:** (A < B) is not true. **>= (greater than or equal to)**
5	Checks if the value of left operand is greater than or equal to the value of right operand, if yes then condition becomes true. **Example** (A >= B) is true. **<= (less than or equal to)**
6	Checks if the value of left operand is less than or equal to the value of right operand, if yes then condition becomes true. **example**(A <= B) is not true.

Bitwise Operators:

Java defines several bitwise operators, which can be applied to the integer types, long, int, short, char, and byte.

Bitwise operator works on bits and performs bit-by-bit operation. Assume if a = 60; and b =

Written by: **Shamik Mitra**
Technical Leader at IBM
blog: http://javaonfly.blogspot.in/
facebook : https://www.facebook.com/shamik.mitra.37
mob : **9830471739** mail **:mitrashamik@gmail.com**
Provide Tuition on java, j2ee Struts,Spring,Hibernate,jpa,Hadoop,Devops,Agile,kanban
13; now in binary format they will be as follows:

a = 0011 1100

b = 0000 1101

---

a&b = 0000 1100

a|b = 0011 1101

a^b = 0011 0001

~a = 1100 0011

The following table lists the bitwise operators:

Assume integer variable A holds 60 and variable B holds 13 then:

SR.NO	Operator and Description
	& (bitwise and)
1	Binary AND Operator copies a bit to the result if it exists in both operands. **Example:** (A & B) will give 12 which is 0000 1100
	| (bitwise or)
2	Binary OR Operator copies a bit if it exists in either operand. **Example:** (A | B) will give 61 which is 0011 1101
	^ (bitwise XOR)
3	Binary XOR Operator copies the bit if it is set in one operand but not both. **Example:** (A ^ B) will give 49 which is 0011 0001
	~ (bitwise compliment)
4	Binary Ones Complement Operator is unary and has the effect of 'flipping' bits. **Example:** (~A) will give -61 which is 1100 0011 in 2's complement form due to a signed binary number.
	<< (left shift)
5	Binary Left Shift Operator. The left operands value is moved left by the number of bits specified by the right operand

	Example: A << 2 will give 240 which is 1111 0000
	>> (right shift)
6	Binary Right Shift Operator. The left operands value is moved right by the number of bits specified by the right operand.
	Example: A >> 2 will give 15 which is 1111
	>>> (zero fill right shift)
7	Shift right zero fill operator. The left operands value is moved right by the number of bits specified by the right operand and shifted values are filled up with zeros.
	Example: A >>>2 will give 15 which is 0000 1111

Logical Operators

If A=20 and B=10

Operator	Description
	&& (logical and)
1	Called Logical AND operator. If both the operands are non-zero, then the condition becomes true.
	Example (A && B) is false.
	|| (logical or)
2	Called Logical OR Operator. If any of the two operands are non-zero, then the condition becomes true.
	Example (A || B) is true.
	! (logical not)
3	Called Logical NOT Operator. Use to reverses the logical state of its operand. If a condition is true then Logical NOT operator will make false.
	Example !(A && B) is true.

Assignment Operators:

There are following assignment operators supported by Java language:

Written by: **Shamik Mitra**
Technical Leader at IBM
blog: http://javaonfly.blogspot.in/
facebook : https://www.facebook.com/shamik.mitra.37
mob : **9830471739** mail **:mitrashamik@gmail.com**
Provide Tuition on java, j2ee Struts,Spring,Hibernate,jpa,Hadoop,Devops,Agile,kanban

SR.NO	Operator and Description
1	**=** Simple assignment operator, Assigns values from right side operands to left side operand. **Example:** C = A + B will assign value of A + B into C
2	**+=** Add AND assignment operator, It adds right operand to the left operand and assign the result to left operand. **Example:** C += A is equivalent to C = C + A
3	**-=** Subtract AND assignment operator, It subtracts right operand from the left operand and assign the result to left operand. **Example:**C -= A is equivalent to C = C - A
4	***=** Multiply AND assignment operator, It multiplies right operand with the left operand and assign the result to left operand. **Example:** C *= A is equivalent to C = C * A
5	**/=** Divide AND assignment operator, It divides left operand with the right operand and assign the result to left operand **Example**C /= A is equivalent to C = C / A
6	**%=** Modulus AND assignment operator, It takes modulus using two operands and assign the result to left operand. **Example:** C %= A is equivalent to C = C % A
7	**<<=** Left shift AND assignment operator. **Example**C <<= 2 is same as C = C << 2
8	**>>=**

Written by: **Shamik Mitra**
Technical Leader at IBM
blog: http://javaonfly.blogspot.in/
facebook : https://www.facebook.com/shamik.mitra.37
mob : **9830471739** mail **:mitrashamik@gmail.com**
Provide Tuition on java, j2ee Struts,Spring,Hibernate,jpa,Hadoop,Devops,Agile,kanban

Right shift AND assignment operator

Example C >>= 2 is same as C = C >> 2
&=

9 Bitwise AND assignment operator.

Example: C &= 2 is same as C = C & 2
^=

10 bitwise exclusive OR and assignment operator.

Example: C ^= 2 is same as C = C ^ 2
|=

11 bitwise inclusive OR and assignment operator.

Example: C |= 2 is same as C = C | 2

Conditional Operator (? :)

Conditional operator is also known as the ternary operator. This operator consists of three operands and is used to evaluate Boolean expressions. The goal of the operator is to decide which value should be assigned to the variable. The operator is written as:

Tip: variable x = (expression) ? value if true : value if false

Example

```
package com.example.operator;

publicclass OperatorExample {

	inta=20;
	intb=10;

	publicvoid airithMeticOperation()
	{
		intresult = a+b;
		System.out.println("airithMeticOperation is"+ result);
	}

	publicvoid relationalOperation()
```

Written by: **Shamik Mitra**
Technical Leader at IBM
blog: http://javaonfly.blogspot.in/
facebook : https://www.facebook.com/shamik.mitra.37
mob : **9830471739** mail **:mitrashamik@gmail.com**
Provide Tuition on java, j2ee Struts,Spring,Hibernate,jpa,Hadoop,Devops,Agile,kanban

```
	{
		boolean result = a>b;
		System.out.println("relationalOperation is"+ result);
	}

	public void bitwiseOperation()
	{
		int result = a|b;
		System.out.println("bitwiseOperation is"+ result);
	}

	public void logicalOperation()
	{

		boolean result =(a>10 &&b<=10);
		System.out.println("logicalOperation is"+ result);
	}

	public void assignmentOperation()
	{

		a += 10;
		System.out.println("assignmentOperation is"+ a);
	}

	public void conditionalOperation()
	{

		int result = a>20?a:b;
		System.out.println("conditionalOperation is"+ result);
	}


	public static void main(String[] args) {
		OperatorExample example = new OperatorExample();
		example.airithMeticOperation();
		example.relationalOperation();
		example.bitwiseOperation();
		example.logicalOperation();
		example.assignmentOperation();
		example.conditionalOperation();



	}
```

Written by: **Shamik Mitra**
Technical Leader at IBM
blog: http://javaonfly.blogspot.in/
facebook : https://www.facebook.com/shamik.mitra.37
mob : **9830471739** mail **:mitrashamik@gmail.com**
Provide Tuition on java, j2ee Struts,Spring,Hibernate,jpa,Hadoop,Devops,Agile,kanban
}

Output :
airithMeticOperation is30
relationalOperation istrue
bitwiseOperation is30
logicalOperation istrue
assignmentOperation is30
conditionalOperation is30

Loop & Array

There may be a situation when you need to execute a block of code several number of times. Suppose you have to print 1 to 10000.

Programming languages provide loop for that.

A **loop** statement allows us to execute a statement or group of statements multiple times and following is the general form of a loop statement in most of the programming languages:

There is three type of Loop in java

Written by: **Shamik Mitra**
Technical Leader at IBM
blog: http://javaonfly.blogspot.in/
facebook : https://www.facebook.com/shamik.mitra.37
mob : **9830471739** mail **:mitrashamik@gmail.com**
Provide Tuition on java, j2ee Struts,Spring,Hibernate,jpa,Hadoop,Devops,Agile,kanban

1. **for Loop**

2. **while Loop**

3. **do while loop**

For Loop:

Syntax of for loop

for(initialization; boolean expression or condition ;increment or decrement)

{
//block of code
}

Here is the sequnce flow of for loop

The **initialization** step is executed first, and **only once.** This step allows you to declare and initialize any loop control variables. a semi colon (;) use for separator

- Next, the **Boolean expression** is evaluated. **If it is true**, the body of the loop is executed. If it is **false**, the body of the loop will not be executed and control jumps to the next statement after the for loop.
- After the **body** of the for loop gets executed, the control jumps back up to the increment/decrement statement. This statement allows you to update any loop control variables. **This statement can be left blank with a semicolon at the end**.
- The Boolean expression is now evaluated again. If it is true, the loop executes and the process repeats (body of loop, then update step, then Boolean expression). After the Boolean expression is false, the for loop terminates.

Example

Print 1 to 100 using for loop

```
public class LoopTest {

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

Written by: **Shamik Mitra**
Technical Leader at IBM
blog: http://javaonfly.blogspot.in/
facebook : https://www.facebook.com/shamik.mitra.37
mob : **9830471739** mail **:mitrashamik@gmail.com**
Provide Tuition on java, j2ee Struts,Spring,Hibernate,jpa,Hadoop,Devops,Agile,kanban

```
      for(int x = 10; x < 100; x++) {
         System.out.println("value of x : " + x );

      }
   }
}
```

Each style for Loop

From java 5 onwards we can use each style for loop. Here we don't need to initialize value or increment or decrement variables. We just have to create a temporary variable of the type of Collection or Array(discussed later) that variable holds the value for current Iteration refrence.

Syntax

for(variable type of collection element : collection)

{

//Statement

}

Example :Print Integer array using each type for loop

```
package com.example.loop;

publicclass EachStyleLoop {


                public Integer[] arr=new Integer[]{10,20,30,40,50};

                publicstaticvoid main(String args[]) {

                        EachStyleLoop obj =new EachStyleLoop();

                for(Integer i : obj.arr)
                        System.out.println("value of Integer is : " + i);
```

```
                }



}
```

Here **obj.arr** is type of Integer array so we create a temporary variable **i** of type Integer , Now for each element i will point that element and print its value.

While Loop:

A while loop statement in java programming language repeatedly executes a target statement as long as a given condition is true.

The syntax of a while loop is:

```
while(boolean_expression==true)
{
   //block of code
}
```

Here, **statement(s)** may be a single statement or a block of statements. The **condition** may be any expression, and true is any non-zero value.

When executing, if the *boolean_expression* result is true, then the actions inside the loop will be executed. This will continue as long as the expression result is true.

Example: Print Even numbers between 1 to 100

```
package com.example.loop;

publicclass WhileLoop {

        publicvoid displayEven()
        {
                inti=0;//initialization
                while(i<=100)
                {
                        if(i%2==0)
                        {
                                System.out.println("Even Number is " + i);
                        }
```

```
                    i++;//increment
                }
        }

        publicstaticvoid main(String[] args) {

                WhileLoop obj =new WhileLoop();
                obj.displayEven();
        }

}
```

do While Loop:

A do...while loop is similar to a while loop, except that a do while loop is guaranteed to execute at least one time as condition is tested after body finishes,

Syntax:

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

Please note that the Boolean expression appears at the end of the loop, so the statements in the loop execute once before the Boolean is tested.

If the Boolean expression is true, the control jumps back up to do statement, and the statements in the loop execute again. This process repeats until the Boolean expression is false.

Example: Demo Account act as a real account for first time publicity purpose

```
package com.example.loop;

publicclass DoWhileLoop {

        privatebooleanisDemoAccount=true;
```

Written by: **Shamik Mitra**
Technical Leader at IBM
blog: http://javaonfly.blogspot.in/
facebook : https://www.facebook.com/shamik.mitra.37
mob : **9830471739** mail **:mitrashamik@gmail.com**
Provide Tuition on java, j2ee Struts,Spring,Hibernate,jpa,Hadoop,Devops,Agile,kanban

```
        publicvoid executAccount()
        {

                do
                {
                        System.out.println("Although a Demo account For publicity it will act as real account for first time");
                }
                while(!isDemoAccount);

        }

        publicstaticvoid main(String[] args) {

                DoWhileLoop obj = new DoWhileLoop();
                obj.executAccount();

        }

}
```

Arrays:

As of now we just explore single valued property. If we have to deal with multivalued properties? then an **array is answer it is a data structure,** which stores a fixed-size sequential collection of elements of the same type. An array is used to store a collection of data, but it is often more useful to think of an array as a **collection of variables of the same type**.

Why array is so handy?

Instead of declaring individual variables, such as number0, number1, ..., and number99, you can declare one array variable such as numbers and use arr[0], arr[1], and ..., arr[99] to represent individual variables.

Array declaration

You can declare an array in two ways

Written by: **Shamik Mitra**
Technical Leader at IBM
blog: http://javaonfly.blogspot.in/
facebook : https://www.facebook.com/shamik.mitra.37
mob : **9830471739** mail **:mitrashamik@gmail.com**
Provide Tuition on java, j2ee Struts,Spring,Hibernate,jpa,Hadoop,Devops,Agile,kanban

1. <datatype>[] arrayRef; // best practice . as you can tell by seeing datatype it is an array
2. <datatype> arrayRef[];// not wrong but avoid to use

Array Creation

There is two ways to create array

1. Array declaration with post initialization.
2. Array declaration with initialization.

Array declaration with post initialization:

<datatype>[] arr = new <dataType>[size];
Integer[] arr = new Integer [10];// It will create a Integer array with size 10.

The above statement does two things:

- It creates an array with size 10 using new Integer [10];
- It assigns the reference of the newly created array to the variable arr.

TIP: Please remember when you declare array in above way you must have to provide size, which is always in right side. Don't ever put it to left side or left it blank it is wrong way to declare array.

So Integer[] arr = new Integer[10];// is right
But Integer[10] arr = new Integer[];// is wrong
Integer[] arr = new Integer[];// is wrong

Array declaration with initialization:

<datatype>[] arr = {value0, value1, ..., valuek};//initialization value

String[] arr = {"Shamik","Swastika","Samir"} //etc

Integer[] arr = {10,20,30,40} //etc

Written by: **Shamik Mitra**
Technical Leader at IBM
blog: http://javaonfly.blogspot.in/
facebook : https://www.facebook.com/shamik.mitra.37
mob : **9830471739** mail **:mitrashamik@gmail.com**
Provide Tuition on java, j2ee Struts,Spring,Hibernate,jpa,Hadoop,Devops,Agile,kanban

Multi Dimension Array:

An Array can be multi dimension so it can be 2D or 3D or ND array where D refer the dimension.

A 2D Integer array with size 5X5, we can explain in following term

a. An Integer 2D array which has 5 elements where each element is an Integer array
b. Each element array contains 5 integer value

By java we can declare it with

Integer [][] arr = new Integer[5][5];

So Integer[][] arr = new Integer[2][2];

Pictorial view is

arr[0][0]	arr[0][1]	arr[1][0]	arr[1][1]

Integer 2D array of Size 2 with name arr

One thing to remember array is Zero-based index i.e. it starts from zero so if an array with size 10 then it has 10 cells with number 0 to 9.

We can access a cell through it's index position

So arr[0][0] point to First cell of 2D array and then first cell of element array under the 2D array

Similarly So arr[1][0] point to Second cell of 2D array and then first cell of element array under the 2D array.

TIP: when you initialize an array like Integer[] arr = new Integer[5]; It creates a arrar with length 5 and its each cell will populate to default value of data type so in this case it is 0. If it is a String array or any other Object array, then it's each cell value will be null;

Here arr[0] gives 0 if String arr[0] gives null.

Written by: **Shamik Mitra**
Technical Leader at IBM
blog: http://javaonfly.blogspot.in/
facebook : https://www.facebook.com/shamik.mitra.37
mob : **9830471739** mail **:mitrashamik@gmail.com**
Provide Tuition on java, j2ee Struts,Spring,Hibernate,jpa,Hadoop,Devops,Agile,kanban

One more thing to populate or retrieve value for each cell you have to traverse through each cell. So for loop is required for that general rule is Number of dimension increase number of inner for loop is increased to populate value

Examaple : A program populate and retrieve value from 2D array

```
package com.example.arrytest;

publicclass TwoDArrayTest {


	Integer[][] arr = new Integer[2][2];


	publicvoid insertion()
	{
		for(int i=0;i<2;i++)
		{

			for(int j=0;j<2;j++)
			{

				int res =i*j+1;
				arr[i][j]=res;

				System.out.println("value Inserted in cell arr["+i+"]["+j+"] is " +
res);
			}

		}
	}


	publicvoid traversal()
	{
		for(inti=0;i<2;i++)
		{

			for(int j=0;j<2;j++)
			{

				int res =arr[i][j];
```

Written by: **Shamik Mitra**
Technical Leader at IBM
blog: http://javaonfly.blogspot.in/
facebook : https://www.facebook.com/shamik.mitra.37
mob : **9830471739** mail **:mitrashamik@gmail.com**
Provide Tuition on java, j2ee Struts,Spring,Hibernate,jpa,Hadoop,Devops,Agile,kanban

```
                                System.out.println("value retrive from cell arr["+i+"]["+j+"] is " +
res);
                        }

                }
        }



        publicstaticvoid main(String[] args) {

                TwoDArrayTest obj = new TwoDArrayTest();
                obj.insertion();
                System.out.println("GOING TO RETRIVE");
                obj.traversal();

        }



}
```

Output :

```
value Inserted in cell arr[0][0] is 1
value Inserted in cell arr[0][1] is 1
value Inserted in cell arr[1][0] is 1
value Inserted in cell arr[1][1] is 2
GOING TO RETRIVE
value retrive from cell arr[0][0] is 1
value retrive from cell arr[0][1] is 1
value retrive from cell arr[1][0] is 1
value retrive from cell arr[1][1] is 2
```

Pass an Array to Method

Written by: **Shamik Mitra**
Technical Leader at IBM
blog: http://javaonfly.blogspot.in/
facebook : https://www.facebook.com/shamik.mitra.37
mob : **9830471739** mail **:mitrashamik@gmail.com**
Provide Tuition on java, j2ee Struts,Spring,Hibernate,jpa,Hadoop,Devops,Agile,kanban
Just as you can pass primitive type values to methods, you can also pass arrays to methods.

For example, the following method print the elements in an int array:

```
package com.example.arrytest;

publicclass PassArrayToMethod {

	publicvoid display(int[] array) {
		for (int i = 0; i < array.length; i++) {
		   System.out.print(array[i] + " ");
		  }
		}



	publicstaticvoid main(String[] args) {


		int[] arr = {1,5,8,11,14};
		PassArrayToMethod obj = new PassArrayToMethod();
		obj.display(arr);

	}

}
```

Return Array from Method

A method may also return an array.

Example : the method shown below returns an array that is the reversal of another array:

```
package com.example.arrytest;

publicclass ReturnArrayFromMethod {

	publicint[] reverse(int[] arr) {
		int[] reverseArr = newint[arr.length];

		for (int i = 0, j = reverseArr.length - 1; i < arr.length; i++, j--) {
			reverseArr[j] = arr[i];
		  }
```

```
                return reverseArr;
                }



        publicvoid print(int[] arr)
        {
                for (int i = 0;i <arr.length;i++) {
                        System.out.print(arr[i]+ " ");
                 }

                 System.out.println();

        }


        publicstaticvoid main(String[] args) {

                ReturnArrayFromMethod obj =new ReturnArrayFromMethod();
                int[] arr={1,2,3,4};
                obj.print(arr);
                int[] res = obj.reverse(arr);
                obj.print(res);

        }

}
```

TIP: Always remember Array itself is Object. An array of primitives mean an array holds primitive elements but array itself is an Object

Arrays Class:

Arrays is an helper class which java provides to do some operations on Array easily .The **java.util.Arrays** class contains various static methods for sorting and searching arrays, comparing arrays, and filling array elements. These methods are overloaded for all primitive types.

Written by: **Shamik Mitra**
Technical Leader at IBM
blog: http://javaonfly.blogspot.in/
facebook : https://www.facebook.com/shamik.mitra.37
mob : **9830471739** mail **:mitrashamik@gmail.com**
Provide Tuition on java, j2ee Struts,Spring,Hibernate,jpa,Hadoop,Devops,Agile,kanban

public static int binarySearch(Object[] a, Object key)

Searches the specified array of Object (Byte, Int , double, etc.) for the specified value using the binary search algorithm. The array must be sorted prior to making this call. This returns index of the search key, if it is contained in the list; otherwise, (-(insertion point + 1).

public static boolean equals(long[] a, long[] a2)

2 Returns true if the two specified arrays of longs are equal to one another. Two arrays are considered equal if both arrays contain the same number of elements, and all corresponding pairs of elements in the two arrays are equal. This returns true if the two arrays are equal. Same method could be used by all other primitive data types (Byte, short, Int, etc.)

public static void fill(int[] a, int val)

3 Assigns the specified int value to each element of the specified array of ints. Same method could be used by all other primitive data types (Byte, short, Int etc.)

public static void sort(Object[] a)

4 Sorts the specified array of objects into ascending order, according to the natural ordering of its elements. Same method could be used by all other primitive data types (Byte, short, Int, etc.)

Written by: **Shamik Mitra**
Technical Leader at IBM
blog: http://javaonfly.blogspot.in/
facebook : https://www.facebook.com/shamik.mitra.37
mob : **9830471739** mail **:mitrashamik@gmail.com**
Provide Tuition on java, j2ee Struts,Spring,Hibernate,jpa,Hadoop,Devops,Agile,kanban

Chapter 4

Java Decision making

Decision making structures have one or more conditions to be evaluated or tested by the program, along with a statement or statements that are to be executed if the condition is determined to be true, and optionally, other statements to be executed if the condition is determined to be false

Java programming language provides following types of decision making statements.

1. **If statement**
2. **If else statement**
3. **If else if else**
4. **Switch case**

If Statement:

An **if** statement consists of a Boolean expression followed by one or more statements.

Syntax :

```
if(Boolean expression==true)
{

    //Statements
}
```

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

Example : If number grater that 9 then it is a two digit Number

Written by: **Shamik Mitra**
Technical Leader at IBM
blog: http://javaonfly.blogspot.in/
facebook : https://www.facebook.com/shamik.mitra.37
mob : **9830471739** mail **:mitrashamik@gmail.com**
Provide Tuition on java, j2ee Struts,Spring,Hibernate,jpa,Hadoop,Devops,Agile,kanban

```
package com.example.decision;

publicclass IfTest {


	publicvoid checkIf(int number)
	{
		if(number >9)
		{
			System.out.println("This is 2 digit Number");
		}

	}

	publicstaticvoid main(String[] args) {

		IfTest obj =new IfTest();
		obj.checkIf(10);

	}

}
```

If else statement

An **if** statement can be followed by an optional **else** statement, which executes when the Boolean expression is false. Like in above example if it is a one digit number then we can print the same.

Syntax:

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

Example:

```
package com.example.decision;

publicclassIfElse {
```

Written by: **Shamik Mitra**
Technical Leader at IBM
blog: http://javaonfly.blogspot.in/
facebook : https://www.facebook.com/shamik.mitra.37
mob : **9830471739** mail **:mitrashamik@gmail.com**
Provide Tuition on java, j2ee Struts,Spring,Hibernate,jpa,Hadoop,Devops,Agile,kanban

```
	publicvoid checkIf(int number)
	{
		if(number >9)
		{
			System.out.println("This is 2 digit Number");
		}
		else
		{
			System.out.println("This is 1 digit Number");
		}

	}

	publicstaticvoid main(String[] args) {

		IfElse obj =newIfElse();
		obj.checkIf(8);

	}


}
```

If else if else

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

***TIP*:When using if , else if , else statements there are few points to keep in mind.**

- **An if can have zero or one else's and it must come after any else if's.**
- **An if can have zero to many else if's and they must come before the else.**
- **Once an else if succeeds, none of the remaining else if's or else's will be tested.**

Example; Greet World according to time

```
package com.example.decision;

publicclass Greet {
```

```
	publicvoid greetWorld(int time)
	{
		if(time>0 && time <12)
		{
			System.out.println("Good Morning");
		}
		elseif(time>=12 && time <16)
		{
			System.out.println("Good Afternoon");
		}
		elseif(time>=16 && time <18)
		{
			System.out.println("Good Evening");
		}
		else
		{
			System.out.println("Good Night");
		}

	}


	publicstaticvoid main(String[] args) {

		Greet greet = new Greet();
		greet.greetWorld(17);

	}

}
```

Switch Case:

A **switch** statement allows a variable to be tested for equality against a list of values. Each value is called a case, and the variable being switched on is checked for each case.

Syntax:

```
switch(expression){
case value :
//Statements
break;//optional
case value :
//Statements
break;//optional
```

Written by: **Shamik Mitra**
Technical Leader at IBM
blog: http://javaonfly.blogspot.in/
facebook : https://www.facebook.com/shamik.mitra.37
mob : **9830471739** mail **:mitrashamik@gmail.com**
Provide Tuition on java, j2ee Struts,Spring,Hibernate,jpa,Hadoop,Devops,Agile,kanban

```
//You can have any number of case statements.
default://Optional
//Statements
}
```

The following rules apply to a **switch** statement:

- The variable used in a switch statement can only be integers, convertible integers (byte, short, char), strings and enums
- You can have any number of case statements within a switch. Each case is followed by the value to be compared to and a colon.

- The value for a case must be the same data type as the variable in the switch and it must be a constant or a literal.
- When the variable being switched on is equal to a case, the statements following that case will execute until a *break* statement is reached.
- When a *break* statement is reached, the switch terminates, and the flow of control jumps to the next line following the switch statement.
- Not every case needs to contain a break. If no break appears, the flow of control will *fall through* to subsequent cases until a break is reached.
- A *switch* statement can have an optional default case, which must appear at the end of the switch. The default case can be used for performing a task when none of the cases is true. No break is needed in the default case.

Example: Greet world using Switch

```
package com.example.decision;

publicclass SwitchTest {

        publicvoid greetWorld(int time)
        {


                if(time>0 && time <12)
                {
                        time =0;
```

```
		}
		elseif(time>=12 && time <16)
		{
			time =12;
                                }
		elseif(time>=16 && time <18)
		{
			time =16;

		}
		else
		{
			time =124;


		}

		// set time to 0,12,16,24 as case only support constant or literal
		switch(time)
		{
		case 0:
			System.out.println("Good Morning");
			break;

		case 12:
			System.out.println("Good Afternoon");
			break;
		case 16:
			System.out.println("Good Evening");
			break;
		default:
			System.out.println("Good Night");


		}

	}


	publicstaticvoid main(String[] args) {

		Greet greet = new Greet();
		greet.greetWorld(17);

	}
```

Written by: **Shamik Mitra**
Technical Leader at IBM
blog: http://javaonfly.blogspot.in/
facebook : https://www.facebook.com/shamik.mitra.37
mob : **9830471739** mail **:mitrashamik@gmail.com**
Provide Tuition on java, j2ee Struts,Spring,Hibernate,jpa,Hadoop,Devops,Agile,kanban
}

Chapter 5

AutoBoxing and UnBoxing:

Each primitive type has a Wraaper class in java

The automatic conversion of primitive data types into its equivalent Wrapper type is known as boxing and opposite operation is known as unboxing. This is the new feature of Java5. So java programmer doesn't need to write the conversion code. Prior to java we have to do the conversion by own.

Example of Boxing and Unboxing

```
package com.example.boxing;

publicclass BoxingUnBoxing {


	publicvoid box(Integer i)
	{
		System.out.println("Automatically Box to Integer "  + i);
	}

	publicvoid unbox(int i)
	{
		System.out.println("Automatically unBox to int "  + i);
	}

	publicstaticvoid main(String[] args) {

		BoxingUnBoxing obj =new BoxingUnBoxing();
		int i=2;
		obj.box(i);
```

```
            Integer obInt= new Integer(10);
            obj.unbox(obInt);


        }


}
```

Behind the Scene when you do boxing/unboxing compiler actually perform intermediate step for you

i.e int i=2 became Integer by new Integer (2). Compiler done this for you

in case of unbox Integer 10 became int 10 in by i.intValue(); Compiler done this for you

TIP: Pay attention when doing unboxing actually it performs Ineteger.intValue() on the Integer reference so if the Integer is null (Object default value) then null.intValue(); produce null pointer Exception which is very hard to debug as compiler does unboxing automatically for you .

So in above example if we change this line

```
Integer obInt= null;
obj.unbox(obInt);
```

It will produce null pointer exception.

Prior to java 5 you have to do it by yourself

Varargs (Variable Argument):

The varrags allows the method to accept zero or muliple arguments. Before varargs either we use overloaded method or take an array as the method parameter but it was not considered good because it leads to the maintenance problem. If we don't know how many arguments we will have to pass in the method, varargs is the better approach.

Written by: **Shamik Mitra**
Technical Leader at IBM
blog: http://javaonfly.blogspot.in/
facebook : https://www.facebook.com/shamik.mitra.37
mob : **9830471739** mail **:mitrashamik@gmail.com**
Provide Tuition on java, j2ee Struts,Spring,Hibernate,jpa,Hadoop,Devops,Agile,kanban

Let say We have a class Summation which will do sum of int. Now if there is a method calls sum(int a,int b) then it will sum two int . If requirement change now it will do sum of 3 int we have to change the method or overload the method which is hectic. So get rid of it we use varargs. So remember when we are not sure number of arguments methods take varargs is better approach.

Syntax of varargs

Declaration : <datatype>… refrenceName; i.e String… name;

To call method with varagrs

method(1,2,3….n); where 1,2,3 are number of arguments you want to pass.
i.e

public void callName(String… name)

call it by
callName(“Shamik”,”Samir”) ;

Example:

```
package com.example.varargs;

publicclass VaragsTest {


	publicvoid sum(int ... i)
	{

		int a = i[0];
		int b= i[1];
		int c = i[2];

		int sum = a+b+c;
		System.out.println("Sum is "+ sum);
	}

	publicstaticvoid main(String[] args) {

		VaragsTest obj = new VaragsTest();
		obj.sum(10,20,30);

	}
```

Written by: **Shamik Mitra**
Technical Leader at IBM
blog: http://javaonfly.blogspot.in/
facebook : https://www.facebook.com/shamik.mitra.37
mob : **9830471739** mail **:mitrashamik@gmail.com**
Provide Tuition on java, j2ee Struts,Spring,Hibernate,jpa,Hadoop,Devops,Agile,kanban
}

In method sum we get the value by i[0],i[1] by index. Internally varargs act as array

So to get nth argument you do i[n-1]; as array zero-based.

Pay attention to call,we call sum(10,20,30); for pass 3 arguments but if we need pass n arguments Then call is sum(1,2,….,n); where 1,2,3 is the number of arguments to n ,so do sum of 10 ints
you have to pass 10 arguments in sum method.

i.e obj.sum (1,2,3,4,5,6,7,8,9,10);

and get i[0] to i[9] in method sum then add;

we can change the method like this

```
package com.example.varargs;

publicclass VarargsLoop {

	publicvoid sum(int...i)
	{
		int result=0;

		for(int element : i )
		{
			result=result+element;
		}
		System.out.println("sum is " + result);
	}

	publicstaticvoid main(String[] args) {

		VarargsLoop loop = new VarargsLoop();
		loop.sum(1,2,3,4,5,6,7,8,9,10);

	}

}
```

Written by: **Shamik Mitra**
Technical Leader at IBM
blog: http://javaonfly.blogspot.in/
facebook : https://www.facebook.com/shamik.mitra.37
mob : **9830471739** mail **:mitrashamik@gmail.com**
Provide Tuition on java, j2ee Struts,Spring,Hibernate,jpa,Hadoop,Devops,Agile,kanban

Chapter 6

Java Encapsulation:

Encapsulation is one of the main fundamental OOP concepts. The other three are inheritance, polymorphism, and abstraction.

Encapsulation in Java is a mechanism of wrapping the data (variables) and code acting on the data (methods) together as a single unit.

In encapsulation the variables of a class will be hidden from other classes, and can be accessed only through the methods of their current class, therefore it is also known as data hiding.

Why Encapsulation?

Why we need to hide your data from outside? It is because you don't want your code will break by outsider intentionally. Let say you have a class Movie and Movie can only be seen if age is over 18 now so naturally you have class name **Movie** which has one property **age** and one method say **watchMovie**.

Now without Encapsulation class will look like

```
package com.example.encapsulation;

publicclass Movie {

	public Integer age;


	publicvoid watchMovie()
	{
		if(age.intValue()>18)
		{
			System.out.println("Watching Movie");
		}
```

```
                else
                {
                        System.out.println("You are not permitted to Watch Movie");
                }
        }

        publicstaticvoid main(String[] args) {

                Movie movie=new Movie();
                // intentionally I set the age to null
                movie.age=null;
                movie.watchMovie();

        }

}
```

Pay attention to the line
movie.age=**null**;

here I set the age as null intentionally as I have direct access to property age.so an out-come I am able to break the code of Movie class by **null pointer exception**

No I will change the Movie class in a way so I can achieve encapsulation

To achieve encapsulation in Java

- Declare the variables of a class as private.
- Provide public setter and getter methods to modify and view the variables values.

Now class look like

```
package com.example.encapsulation;

publicclass Movie {

        private Integer age;


        publicvoid watchMovie()
```

Written by: **Shamik Mitra**
Technical Leader at IBM
blog: http://javaonfly.blogspot.in/
facebook : https://www.facebook.com/shamik.mitra.37
mob : **9830471739** mail **:mitrashamik@gmail.com**
Provide Tuition on java, j2ee Struts,Spring,Hibernate,jpa,Hadoop,Devops,Agile,kanban

```
	{
		if(age.intValue()>18)
		{
			System.out.println("Watching Movie");
		}
		else
		{
			System.out.println("You are not permitted to Watch Movie");
		}
	}



	public Integer getAge() {
		returnage;
	}



	publicvoid setAge(Integer age) {
		this.age = age;
	}



	publicstaticvoid main(String[] args) {

		Movie movie=new Movie();
		// intentionally I did not set the age
		movie.setAge(null);
		movie.watchMovie();

	}

}
```

So still I can break the code easily. So what the heck encapsulation doing here?

What benefits I got from make age property private and access the property through getter and setter.

Think??????????????

Written by: **Shamik Mitra**
Technical Leader at IBM
blog: http://javaonfly.blogspot.in/
facebook : https://www.facebook.com/shamik.mitra.37
mob : **9830471739** mail **:mitrashamik@gmail.com**
Provide Tuition on java, j2ee Struts,Spring,Hibernate,jpa,Hadoop,Devops,Agile,kanban

Actually by providing setter and getter you make a provision that you can provide validation when the value is set to class.

For say if we put a validation in setAgemethod, if age is null then we set it to 0, then the program run flawlessly.

Also I can put it into watchMovie method but think if there are many methods in Movie class
And each uses age property then in each method you have to put same validation which is nothing but a hectic and repeatable work. Instead of that if we do it in setAge method it will be in single place.

So here is the beauty of Encapsulation always there is a provision for doing something on the property in future.

TIP: Always go for encapsulation

Updated Code

```
 package com.example.encapsulation;

publicclass Movie {

	private Integer age;


	publicvoid watchMovie()
	{
		if(age.intValue()>18)
		{
			System.out.println("Watching Movie");
		}
		else
		{
			System.out.println("You are not permitted to Watch Movie");
		}
	}



	public Integer getAge() {
		returnage;
	}
```

```
	publicvoid setAge(Integer age) {
		if(age==null)
		{
			age=0;
		}
		this.age = age;
	}



	publicstaticvoid main(String[] args) {

		Movie movie=new Movie();
		// intentionally I did not set the age
		movie.setAge(null);
		movie.watchMovie();

	}

}
```

Written by: **Shamik Mitra**
Technical Leader at IBM
blog: http://javaonfly.blogspot.in/
facebook : https://www.facebook.com/shamik.mitra.37
mob : **9830471739** mail **:mitrashamik@gmail.com**
Provide Tuition on java, j2ee Struts,Spring,Hibernate,jpa,Hadoop,Devops,Agile,kanban

Chapter 7

Association or Composition

Till now we are working with one class where we define methods, property and main methods to run the program. This is not an ideal Object orientation or I can say in java world you got a very little scope where you can work with one class. The main purpose of OOP programing is **Object collaboration.**

What is Object collaboration:

Object collaboration means relation between objects by which object can work together, talk to each other invoke one object's property from another. We can say object doing a certain task unitedly.

Let takes an Example:

Suppose you has a mobile and world wants to know the brand of your mobile?

To achieve this one thing, you can do create a class **Person** create two properties **name**, and a String **mobileBrand** and a method **display**

By which you print the specification

```
package com.example.Association;

publicclass MobileSpecification {


        private String personName="SHAMIK";
        private String mobileBrand="SAMSUNG GALAXY";


        publicvoid display()
        {
```

```
            System.out.println(personName + " has a mobile "+ mobileBrand);
        }

        publicstaticvoid main(String[] args) {

            MobileSpecification obj = new MobileSpecification();
            obj.display();

        }



}
```

So you can achieve the solution beautifully. But if you think minutely as a developer you did a little mistake here. You do not think about future as a developer which **is primary things to think.**

Let say now world wants to know more about your mobile not only the name with this your mobile OS name. RAM size , Touch Screen type etc.

Now What you going to do add more properties in same class MobileSpecification and prints them. But if again requirements change in future? So every time requirement changes you have to alter the same class to adopt it. Now If world wants to know the person Details that is details about you. Then same class MobileSpecification has to be edited.

So either Person or Mobile If requirement changes you have to edit MobileSpecification each time. Which is hectic and error prone.

So what mistake you done?

If you look at the problem, you can discover Person and Mobile is two independent object

If the requirements change in future for one, that should not be affect the second.

So Ideal design should be creating two class Person and Mobile and they will talk to each other by object collaboration or association to perform the job i.e print the specification

Written by: **Shamik Mitra**
Technical Leader at IBM
blog: http://javaonfly.blogspot.in/
facebook : https://www.facebook.com/shamik.mitra.37
mob : **9830471739** mail **:mitrashamik@gmail.com**
Provide Tuition on java, j2ee Struts,Spring,Hibernate,jpa,Hadoop,Devops,Agile,kanban

So You have a Phone that means in java

You own a Phone
You which is Object of a Person class HAS an Object of a Phone.

More precisely Person object has a property called Mobile which is a reference of a Mobile object.

This is call association when One Object holds a **property** which will **points or refer** to **another object**. By this **property** we access the **other objects property or method** take action on them.

TIP: Always go for association it will segregate the responsibility. Always remember Single Class Has Single Responsibility. i.e A class only contains one object and if the object requirements change that class only be modified not the Other.

As here Person and Mobile two different object now if Mobile requirements change only mobile class will be modified not the person and vice versa.

SO, ALWAYS REMEMBER CLASS IS CHANGED FOR A SINGLE REASON IF THERE IS MULTIPLE REASONS FOR WHICH A CLASS NEED TO BE CHANGED THEN IT IS DESIGN FLAW OF DEVELOPER WHICH IS NOT INTENDED.

In java we call this HAS-A relationship, So HAS_A, composition, association all are same meaning in java

We can Say

Person HAS-A MOBILE

That is Person class has a mobile object as it's property

Updated Example

```
package com.example.Association;

publicclass Mobile {
```

```
        private String name="SamSung Galaxy";
        private String os="Android Lollypop";
        privateintram=1;


        public String getName() {
                returnname;
        }
        public String getOs() {
                returnos;
        }
        publicint getRam() {
                returnram;
        }

        publicvoid printSpecification()
        {

                System.out.println("Phone name "+ name);
                System.out.println("Phone OS "+ os);
                System.out.println("Phone RAM "+ ram + "GB");

        }




}


package com.example.Association;

publicclass Person {

        private String name="Shamik";
        private Mobile mobile;

        public String getName() {
                returnname;
        }
```

Written by: **Shamik Mitra**
Technical Leader at IBM
blog: http://javaonfly.blogspot.in/
facebook : https://www.facebook.com/shamik.mitra.37
mob : **9830471739** mail **:mitrashamik@gmail.com**
Provide Tuition on java, j2ee Struts,Spring,Hibernate,jpa,Hadoop,Devops,Agile,kanban

```
	public Mobile getMobile() {
		returnmobile;
	}



	publicvoid setMobile(Mobile mobile) {
		this.mobile = mobile;
	}



	publicstaticvoid main(String[] args) {


		Mobile mobileAssociation = new Mobile();
		Person person=new Person();

		//Associate Mobile object with person. actually person's mobile property now
points to
		//mobileAssociation object

		person.setMobile(mobileAssociation);

		System.out.println(person.getName() + " HAS A Mobile and it's specifiaction
is");
		person.getMobile().printSpecification();//call mobile;s print specification from
person class




	}



}
```

Written by: **Shamik Mitra**
Technical Leader at IBM
blog: http://javaonfly.blogspot.in/
facebook : https://www.facebook.com/shamik.mitra.37
mob : **9830471739** mail **:mitrashamik@gmail.com**
Provide Tuition on java, j2ee Struts,Spring,Hibernate,jpa,Hadoop,Devops,Agile,kanban
Pay attention to the line

person.setMobile(mobileAssociation);

here Associate Mobile object with person. actually person's mobile property now points to mobileAssociation object

Look at the call

person.getMobile().printSpecification();
Here we call mobile;s print specification method from person class.

Written by: **Shamik Mitra**
Technical Leader at IBM
blog: http://javaonfly.blogspot.in/
facebook : https://www.facebook.com/shamik.mitra.37
mob : **9830471739** mail **:mitrashamik@gmail.com**
Provide Tuition on java, j2ee Struts,Spring,Hibernate,jpa,Hadoop,Devops,Agile,kanban

Chapter 8

Inheritance:

Inheritance is one of the main OOP rule.Inheritance can be defined as the process where **one class** acquires the properties (methods and fields) of **another**. With the use of inheritance the information is made manageable in a hierarchical order.

The class which inherits the properties of other is known as **subclass** (derived class, child class) and the class whose properties are inherited is known as **superclass** (base class, parent class).

Inheritance can be achieved in two ways

1. By **exentds** a class
2. By **implements** an interface

Syntax

```
class Superclass

{

}
```

```
class subclass extends superclass
```

Written by: **Shamik Mitra**
Technical Leader at IBM
blog: http://javaonfly.blogspot.in/
facebook : https://www.facebook.com/shamik.mitra.37
mob : **9830471739** mail **:mitrashamik@gmail.com**
Provide Tuition on java, j2ee Struts,Spring,Hibernate,jpa,Hadoop,Devops,Agile,kanban
{

}

Or

interface contract {

}

Class fullfillContract **implements** contract

{

}

Here extends and implements is java keyword.

Why we need Inheritance?

Inheritance the word describe itself. In world there are many occasion an object inherits a property of another. Like think about a School. In Kolkata there are so many schools and what is the common between them?

It must have a name, it has a medium, it has students etc

What is the differences between them?

Is it having AC or not?

Is it fully computerized or not?

Etc.

Now if we write the common properties in a one place and reuse them for all rather than rewrite the same property for each school object.

TIP: By inheritance we can achieve this-- Write once and use properties multiple times without re write.

SO If you found something common between different objects make a superclass define that property once and use it everywhere.

Example of Inheritance:

```
package com.example.inheritence;

publicclass School {

	private String name;
	private String medium;
	public String getName() {
		returnname;
	}
	publicvoid setName(String name) {
		this.name = name;
	}
	public String getMedium() {
		returnmedium;
	}
	publicvoid setMedium(String medium) {
		this.medium = medium;
	}


}
```

Written by: **Shamik Mitra**
Technical Leader at IBM
blog: http://javaonfly.blogspot.in/
facebook : https://www.facebook.com/shamik.mitra.37
mob : **9830471739** mail **:mitrashamik@gmail.com**
Provide Tuition on java, j2ee Struts,Spring,Hibernate,jpa,Hadoop,Devops,Agile,kanban

```
package com.example.inheritence;

publicclass ComuterizedScool extends School{

	private String Computerized="Fully Computerized";
	publicvoid display()
	{
		System.out.println("School Name " + this.getName());
		System.out.println("School Medium " + this.getMedium());
		System.out.println("Mode of teaching " + Computerized );
	}




}
package com.example.inheritence;

publicclass ManualSchool extends School{
	private String mode="Manually using Black Board";
	publicvoid display()
	{
		System.out.println("School Name " + this.getName());
		System.out.println("School Medium " + this.getMedium());
		System.out.println("Mode of teaching " + mode );
	}


}
package com.example.inheritence;

publicclass SchoolManager {
	publicstaticvoid main(String[] args) {

		ComuterizedScool sch = new ComuterizedScool();
		sch.setName("DPS");
		sch.setMedium("ENGLISH");
		sch.display();
		ManualSchool ms=new ManualSchool();
		ms.setName("RAMKRISHNA MISSION");
		ms.setMedium("BENGALI");
		ms.display();
	}

}
```

Written by: **Shamik Mitra**
Technical Leader at IBM
blog: http://javaonfly.blogspot.in/
facebook : https://www.facebook.com/shamik.mitra.37
mob : **9830471739** mail **:mitrashamik@gmail.com**
Provide Tuition on java, j2ee Struts,Spring,Hibernate,jpa,Hadoop,Devops,Agile,kanban

Here we created four classes

1. School.java
2. ComputerizedScool.java
3. ManualSchool.java
4. SchoolManager.java

Although there is two types of school ComputerizedScool and manual school the name and medium is common between them and the mode of teaching is difference between them

So we create a School object which holds common properties name, medium so that both school can inherit those properties. Only the mode property is added in subclass as they are different for each type of school

So here **School** is Superclass and **ManualSchool** and **ComputerizedScool** is Subclass

Now if a third type of school is added to this hierarchy it will extend the School and automatically inherit the name and medium property. This is the power of inheritance. **Define properties in one place use it any where.**

this Keyword:

Please pay attention to display, method of ManualSchool here we use this.getName(); to access the name property.

this always refer to the it's object. Actually when the object is created in SchoolManager using new ManualSchool(); immediately this will refer to newly created object along with ms reference. So this can be thought of more like pronoun in English instead of use reference name, this can be used in class to refer the object.

Please note one thing

Name and medium property defined in School but as we use extends keyword in Manual School and Computerized School class definition those properties are inherit automatically And we can use it by this as if it is the property of Manual or Computerized school.

Written by: **Shamik Mitra**
Technical Leader at IBM
blog: http://javaonfly.blogspot.in/
facebook : https://www.facebook.com/shamik.mitra.37
mob : **9830471739** mail **:mitrashamik@gmail.com**
Provide Tuition on java, j2ee Struts,Spring,Hibernate,jpa,Hadoop,Devops,Agile,kanban

Rules of Inheritance

1. If a Class is Final you cannot extends it for say if School, class is written as public final class School. Then ManualSchool or Computerized school. Can't extends it.
2. private property or method can't be inherited so in above example name and mode is private so it can't be inherited so we can't do this.name or this.medium in subclass but we can do this.getName() as it's setter and getter is public.
3. We call Inheritance as IS-A relationship
4. So **ManualSchool IS-A School** as Manual School contains all property School has but viceversa is not true as School does not has all property subclass has
5. So we can call Subclass = Super Class + additional property.

instanceof Operator

To check IS-A relationship. Java provides instance of operator

So instanceof operator gives a Boolean value

True means IS-A relation satisfies false means not.

instanceof operator returns true down the hierarchy,

Let's take an Example

Suppose BMW extends CAR and Car extend Vehicle

Then we say

BMW instanceof CAR is true.
CAR instanceof Vehicle is true.
BMW instanceof Vehicle is true.
CAR instanceof BMW is false
Vehicle instanceof CAR is false.
Vehicle instanceof BMW is false.

Protected Modifier with inheritance:

Written by: **Shamik Mitra**
Technical Leader at IBM
blog: http://javaonfly.blogspot.in/
facebook : https://www.facebook.com/shamik.mitra.37
mob : **9830471739** mail **:mitrashamik@gmail.com**
Provide Tuition on java, j2ee Struts,Spring,Hibernate,jpa,Hadoop,Devops,Agile,kanban

protected is most critical modifier to understand .

According to the definition it is same as default modifier, which can be accessed by other classes in same package but the only difference is it can also be accessed by subclasses from outside packages through inheritance.

Now there are two catch points

1. What do you mean by accessed?

2. What do you mean by access through inheritance?

1. What do you mean by accessed?

In java you can access properties of other classes by two ways

a. Association (HAS A) 2. inheritance (IS A)

Scenario a.

```
package com.bar;

public class Bar

{

 protected String greet = "Hello";

}
```

Written by: **Shamik Mitra**
Technical Leader at IBM
blog: http://javaonfly.blogspot.in/
facebook : https://www.facebook.com/shamik.mitra.37
mob : **9830471739** mail **:mitrashamik@gmail.com**
Provide Tuition on java, j2ee Struts,Spring,Hibernate,jpa,Hadoop,Devops,Agile,kanban

```
packge com.foo;

public class Foo extends Bar

{

Bar bar = new Bar(); // BAR HAS A relationship with FOO(Association)


bar.greet; //not compile no visibility through association


}
```

Scenario b

```
packge com.foo

Class Foo extends Bar

{




System.out.println(greet); //Inherited so it print Hello


}


package com.bar
```

Written by: **Shamik Mitra**
Technical Leader at IBM
blog: http://javaonfly.blogspot.in/
facebook : https://www.facebook.com/shamik.mitra.37
mob : **9830471739** mail **:mitrashamik@gmail.com**
Provide Tuition on java, j2ee Struts,Spring,Hibernate,jpa,Hadoop,Devops,Agile,kanban
Class Bar

{

protected String greet = "Hello";

}

Association means a class has a reference of another class as you see in Scenario a.

Inheritance means a class inheriting parent class property.

For protected the tricky situation arises when two classes are in different package, as you see Foo is in com.foo and Bar in com.bar now the question is, can Foo access Bar's greet variable which is protected?

The Answer is simply no as by rule protected is visible only through inheritance so although class Foo **extends Bar but we are trying to access greet by association which is not acceptable in java**

On other hand in **Scenario 2** will be compile fine and show Hello message as Foo inherited Bar's greet property.

Now add another criticality

suppose

Written by: **Shamik Mitra**
Technical Leader at IBM
blog: http://javaonfly.blogspot.in/
facebook : https://www.facebook.com/shamik.mitra.37
mob : **9830471739** mail **:mitrashamik@gmail.com**
Provide Tuition on java, j2ee Struts,Spring,Hibernate,jpa,Hadoop,Devops,Agile,kanban

class zoo is in the

packge com.foo now If I write following code snippet in Zoo

```
Foo foo = new Foo();

foo.greet;
```

will it be compile?

Think about it.....

The Answer is no as I said earlier by Association (different package) you can't ever access greet. but in Same package you do.

Types of Inheritance:

We can have different types of inheritance

Written by: **Shamik Mitra**
Technical Leader at IBM
blog: http://javaonfly.blogspot.in/
facebook : https://www.facebook.com/shamik.mitra.37
mob : **9830471739** mail **:mitrashamik@gmail.com**
Provide Tuition on java, j2ee Struts,Spring,Hibernate,jpa,Hadoop,Devops,Agile,kanban

1. **Single Inheritance**: When there is only one superclass and only one subclass we call it single inheritance

Let say Foo is only super class and only Bar extends Foo then it is a single Inheritance

2. **Multilevel Inheritance**: when a subclass has multilevel parent then we call it Multilevel Inheritance**.** Like BMW extends CAR and CAR extends Vehicle we can say Vehicle->CAR->BMW
3. **Hierarchical Inheritance**: When a super class has multiple subclass we called it Hierarchical Inheritance. Like School has two subclasses ManualSchool and Computrized School.
4. **Multiple Inheritance:** When a sub class extends has two or more parents then it called multiple inheritance like Tablet inherits some of Computer and some of Phones property so we can say

Tablet extends Phone and Computer. But unfortunately Java does not directly support Multiple Inheritance

Diamond Problem:

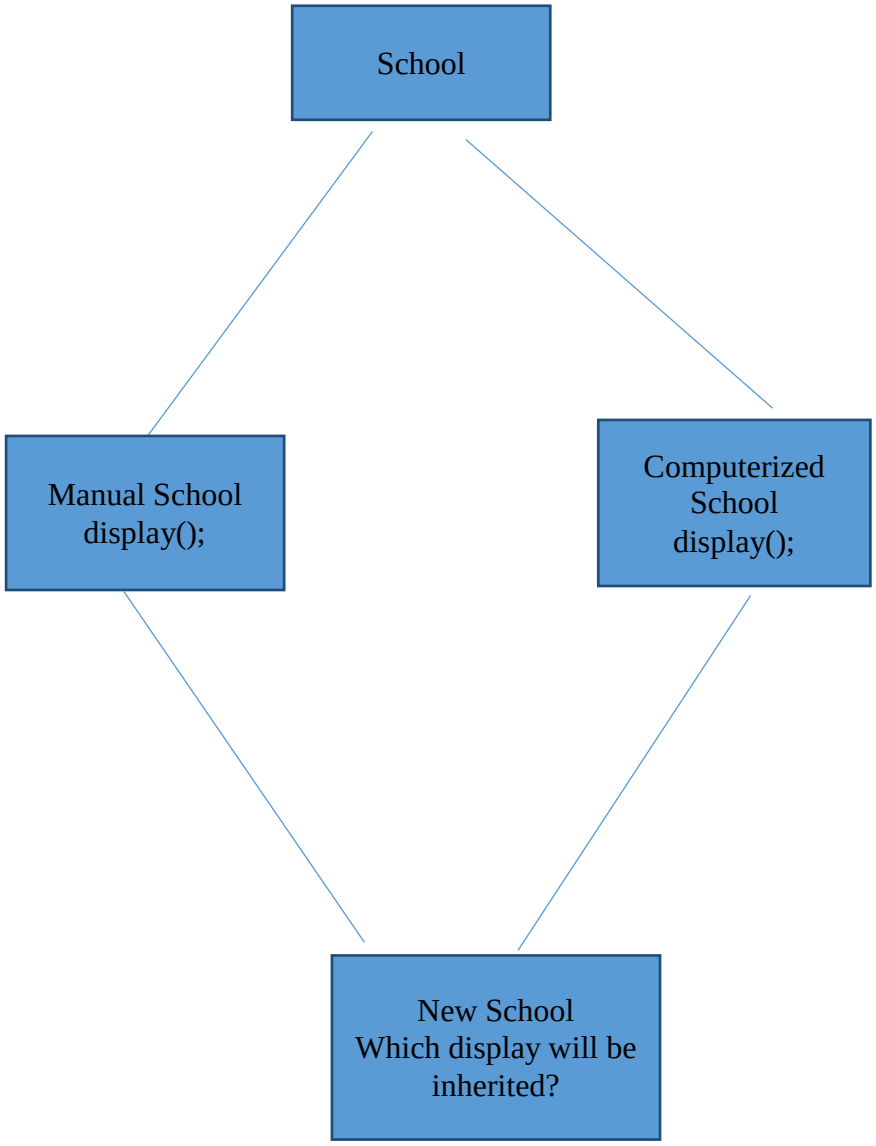
Why java does not support **Multiple Inheritance.** Let takes previous examplewhere manual School and Computerized school inherits School. New Requirements is creating a new Type of school which support both Computerized and manual teaching by Blackboard.

Now the best solution is just extending Computerized school and Manual School so in a new school both properties are inherited. And we can offer both type of teaching.

But doing so we will be faced a real hard problem which java has no answer

Written by: **Shamik Mitra**
Technical Leader at IBM
blog: http://javaonfly.blogspot.in/
facebook : https://www.facebook.com/shamik.mitra.37
mob : **9830471739** mail **:mitrashamik@gmail.com**
Provide Tuition on java, j2ee Struts,Spring,Hibernate,jpa,Hadoop,Devops,Agile,kanban
Pay attention to the method **publicvoid** display() in Computerized and Manual School class it is public so if I extend both Computerized and Manual School both display method will be inherited to child. Now as the method name is same when we invoke display from child which version will be called? Rather which version of display will inherit to the child

JAVA has no Answer to this ambiguous problem. So Java is not supported Multiple inheritance and this problem called Diamond problem as the Shape is like diamond.



Diamond Problem

Written by: **Shamik Mitra**
Technical Leader at IBM
blog: http://javaonfly.blogspot.in/
facebook : https://www.facebook.com/shamik.mitra.37
mob : **9830471739** mail **:mitrashamik@gmail.com**
Provide Tuition on java, j2ee Struts,Spring,Hibernate,jpa,Hadoop,Devops,Agile,kanban

If I said there is no solution or no answer in java, then you people are going to kill me or utter slang words to me. ☺

Because, Nowadays Multiple Inheritance is must like, Tab is inherited from Phone and PC.

So as we all expected **Mighty Java** comes with an answer with this problem

Interface !!!!!!

What is an interface

An interface is a reference type in Java, it is similar to class, it is a collection of **abstract** methods. A class **implements** an interface, thereby inheriting the abstract methods of the interface.

Along with abstract methods an interface may also contain **constants**, default methods, static methods, and nested types. **Method bodies exist only for default methods and static methods.**

Writing an interface is similar to writing a class. But a class describes the attributes and behaviors of an object. And an interface contains behaviors that a class implements. So I can say it is a **contract when a class implement it it should fulfill the contract or in other words has to provide method definition for all declared method in interface.**

Unless the class that implements the interface is abstract, all the methods of the interface need to be **defined** in the class. As those are only declared in interface.

An interface is similar to a class in the following ways:

- An interface can contain any number of methods.
- An interface is written in a file with a **.java** extension, with the name of the **interface** matching the name of the **file**.
- The byte code of an interface appears in a **.class** file.
- Interfaces appear in packages, and their corresponding bytecode file must be in a directory structure that matches the package name.

However, an interface is different from a class in several ways, including:

- You cannot **instantiate** an interface. i.e can't use **new** keyword

Written by: **Shamik Mitra**
Technical Leader at IBM
blog: http://javaonfly.blogspot.in/
facebook : https://www.facebook.com/shamik.mitra.37
mob : **9830471739** mail **:mitrashamik@gmail.com**
Provide Tuition on java, j2ee Struts,Spring,Hibernate,jpa,Hadoop,Devops,Agile,kanban

- An interface does not contain any **constructors**.
- All of the methods in an interface are abstract we can say **declared** no body is present in interface.
- An interface cannot contain **instance** fields. The only fields that can appear in an interface must be declared both **static and final**.
- An interface is not **extended** by a class; it is **implemented** by a class.
- An interface can extend multiple interfaces.so it supports multiple inheritance unlike class. I will tell how it solve Diamond problem later.

Syntax

```
//package <package name>

//Any number of import statements
Import <statements>
public interface NameOfInterface
{
   //Any number of final, static fields
   //Any number of abstract method declarations\
}
```

Interface Rules

- An interface is implicitly abstract. You do not need to use the **abstract** keyword while declaring an interface.
- Each method in an interface is also implicitly **abstract**, so the abstract keyword is not needed.
- Methods in an interface are implicitly public.
- You can declare properties or variables in interface but they are implicitly static and final
- Only static methods can have body in interface as it is not part of instance don’t be fool with that.

Example:

Written by: **Shamik Mitra**
Technical Leader at IBM
blog: http://javaonfly.blogspot.in/
facebook : https://www.facebook.com/shamik.mitra.37
mob : **9830471739** mail **:mitrashamik@gmail.com**
Provide Tuition on java, j2ee Struts,Spring,Hibernate,jpa,Hadoop,Devops,Agile,kanban

```
package com.example.interfaceTest;

publicinterface Workable {

	intwork=10;
	publicvoid work();

}
```

```
package com.example.interfaceTest;

publicclass Man implements Workable{

	publicvoid work() {
		//Workable.work=10;
		System.out.println("Doing work for "+ Workable.work + " hrs");

	}


	publicstaticvoid main(String[] args) {

		Man man = new Man();
		man.work();

	}

}
```

I have created a workable interface with a method named **work**. Please note that although I wrote

Public void work() but compiler makes it
Public abstract void work();

Same, I have created a properties called **work**

Compiler makes it
public static final work=10;

Written by: **Shamik Mitra**
Technical Leader at IBM
blog: http://javaonfly.blogspot.in/
facebook : https://www.facebook.com/shamik.mitra.37
mob : **9830471739** mail **:mitrashamik@gmail.com**
Provide Tuition on java, j2ee Struts,Spring,Hibernate,jpa,Hadoop,Devops,Agile,kanban
Now Man class implements the interface Workable. Earlier I said that interface is like a contract here contract is the work method, so whoever implements the interface Workable it has to fulfill the contract so it has to define the method work which is declared in Workable interface.

Man provide a concreate implementation of the work method. When I created object of man and called the work method it will call the Man's work method.

TIP**: **Methods declared in interface must has to be overridden by the class who implements the interface. Variable declared in interface are by default static final so that is not associated to instance moreover it is final so it can't be reassigned once initialized. So, we can't initialize the work variable in above example.

Key point to remember about Interface

1. An interface can extends another interface
2. An interface can extends multiple interface
3. First concrete class who implements interface has to define all the methods of interface and interface's parent interface if any.
4. A class can implements multiple interface so it's support multiple inheritance

TIP: Burn it in your head, an interface extends another interface but a class implements an interface

Suppose there is interface called workable which has a method called work if a interface ComputerizedWork extends it and it has a method called computerSpecification then A class Robot which implements ComputerizedWork has to redefine work and computerSpecification method.

```
package com.example.interfaceTest;

publicinterface Workable {

        publicvoid work();

}

package com.example.interfaceTest;

publicinterfacecomputerizedWorkextends Workable{

        publicvoid computerSpecification();
```

```
}


package com.example.interfaceTest;

publicclass Robot implements computerizedWork{

	publicvoid work() {
		System.out.println("Doing work using Computer");

	}

	publicvoid computerSpecification() {
		System.out.println("Updated quad core processor");

	}

	publicstaticvoid main(String[] args) {

		Robot obj =new Robot();
		obj.computerSpecification();
		obj.work();

	}

}
```

Solving Diamond Problem Using Interface

If you remember the Diamond problem example, The main problem is Manual Scool and Computerized School both inherits parent display method and re defined it in their own way but when the child object invoke display method which version of parent will be called.

We can solve it by interface as interface is only a contract and it only contains method declaration so if ManualSchool and ComputerizedSchool are interfaces the display method is only declared there so NewSchool will redefine the display method and no one will complain about it !!!!!!!!

Solution

```
package com.example.interfaceTest.DiamondProblem;

publicinterface School {
```

```
	publicvoid display();

}

package com.example.interfaceTest.DiamondProblem;

publicinterface ManualSchool {

	publicvoid display();

}

package com.example.interfaceTest.DiamondProblem;

publicinterfaceComputerizedSchool {

	publicvoid display();

}

package com.example.interfaceTest.DiamondProblem;

publicclass NewSchool implements ManualSchool,ComputerizedSchool{

	publicvoid display() {
		System.out.println("Both Blackboard and Computer");

	}

	publicstaticvoid main(String[] args) {

		NewSchool sch =new NewSchool();
		sch.display();

	}

}
```

Inheritance and constructor hierarchy:

As Java support multilevel inheritance it is possible that a class have multiple predecessor If you remember in constructor section, I said every class has constructor according to that sentence every predecessor has constructor so now the question is when I call the sub-class constructor to instantiate subclass object what will happen?

Written by: **Shamik Mitra**
Technical Leader at IBM
blog: http://javaonfly.blogspot.in/
facebook : https://www.facebook.com/shamik.mitra.37
mob : **9830471739** mail **:mitrashamik@gmail.com**
Provide Tuition on java, j2ee Struts,Spring,Hibernate,jpa,Hadoop,Devops,Agile,kanban
Think about the scenario in terms of inheritance subclass has multilevel parent so this subclass inherited properties from each predecessor so when subclass object instantiate then it needs to populate all the properties including its parents so it is necessary to call all parents constructor to create a successful subclass object.

So when you call new Subclass(); i.e call subclass constructor it calls immediate parent constructor then it's immediate parent constructor and goes on until the last parent is reached.

In this context I want to share a java magic with all of you the first predecessor of every object is **Object** class. There is a class call Object in JVM java create this Object class for you and every object extends this class. So till now as many example I have written every class extends this Object class ,compiler will do this for you so you don't have to bother so much about Object class. **Just rememberObject is the first predecessor of any object created in jvm. So constructor call's end when it call's Objects constructor call;**

Here an Obvious question arise is How to parent class constructor call from child class?

Answer is another java magic, **super** keyword super always refer to parent class it has re-semblance with **this** this refer itself where super refer its immediate parent. Remember one thing the first line in constructor is always **super unless there is an any parametric constructor** but sooner or later one of the constructor's first line will be super. If you write a blank constructor compile put this super keyword for you. So subclass call super, it's immediate parent, same call super in this manner call continues with Object constructor then comeback and execute subclass constructor second line.

Example

```
package com.example.constructor;

publicclass Animal {


	public Animal()
	{
		System.out.println("animal Constructor calls");
	}

}


package com.example.constructor;
```

Written by: **Shamik Mitra**
Technical Leader at IBM
blog: http://javaonfly.blogspot.in/
facebook : https://www.facebook.com/shamik.mitra.37
mob : **9830471739** mail **:mitrashamik@gmail.com**
Provide Tuition on java, j2ee Struts,Spring,Hibernate,jpa,Hadoop,Devops,Agile,kanban

```
publicclass Mamals extendsAnimal{

	public Mamals()
	{
		System.out.println("mamals constructor calls");
	}

}
```

```
package com.example.constructor;

publicclass Man extends Mamals{


	public Man()
	{
		//super(); If you call or not compiler call super for you
		System.out.println("Mans Constructor call");

		System.out.println("Second line call after constructor's call");
	}

	publicstaticvoid main(String[] args) {

		Man man =new Man();


	}

}
```

Output:

```
animal Constructor calls
mamals constructor calls
Mans Constructor call
Second line call after constructor's call
```

Written by: **Shamik Mitra**
Technical Leader at IBM
blog: http://javaonfly.blogspot.in/
facebook : https://www.facebook.com/shamik.mitra.37
mob : **9830471739** mail **:mitrashamik@gmail.com**
Provide Tuition on java, j2ee Struts,Spring,Hibernate,jpa,Hadoop,Devops,Agile,kanban

Chapter 9

Polymorphism:

Polymorphism is the ability of an object to take on many forms. Real world example is Carbon, you heared of diamond, you know charcoal you know about dust ,but all this are different forms of carbon. Same in java the objects inherited from parents inherits it's all property so I can say every subclass is a form of parent class. Subclass cann add it's extra features or redefined it parent's property but one thing is sure as it is inherited form parent it inherits all property of parent

So by logic I can say

Subclass = superclass + extra feature;

So If Animal super class of Mammal and Mammals is super class of Man is

Then following statements are true.

Man IS-A Animal

Man IS-A Mammal

Mammal IS-A Animal

Man IS-A Man

Written by: **Shamik Mitra**
Technical Leader at IBM
blog: http://javaonfly.blogspot.in/
facebook : https://www.facebook.com/shamik.mitra.37
mob : **9830471739** mail **:mitrashamik@gmail.com**
Provide Tuition on java, j2ee Struts,Spring,Hibernate,jpa,Hadoop,Devops,Agile,kanban

But

Mammal IS-A Man is false as Mammals means not only man wheal is also a mamal.

So undoubtedly we can say as child has all property of parent we can assign the child object to parent reference as if we call any method with that reference we can assure child must havethose methods either child inherits it from parent or redefined in his own way.

This is call Polymorphism.

So I can say

Animal a =new Man();

Mamal m = new Man();

As animal and Mamal both are parents of Man class, so I can assign man object to this parent reference this is call **polymorphic assignment this is the most powerful feature of OOPS.**

Polymorphism rules

1. Polymorphism in OOP occurs when a parent class reference is used to refer to a child class object.

2. Any Java object that can pass more than one IS-A test is considered to be polymorphic. In Java, all Java objects are polymorphic since any object will pass the IS-A test for their **own type** and for the class Object.

Written by: **Shamik Mitra**
Technical Leader at IBM
blog: http://javaonfly.blogspot.in/
facebook : https://www.facebook.com/shamik.mitra.37
mob : **9830471739** mail **:mitrashamik@gmail.com**
Provide Tuition on java, j2ee Struts,Spring,Hibernate,jpa,Hadoop,Devops,Agile,kanban

3. **It is important to know that the only possible way to access an object is through a reference variable**. A reference variable can be of only one type. Once declared, the type of a reference variable cannot be changed.

4. The reference variable can be reassigned to other objects provided that it is not declared final. The type of the reference variable would determine the methods that it can invoke on the object.

A reference variable can refer to any object of its declared type or any subtype of its declared type. A reference variable can be declared as a class or interface type.

In java we achieve polymorphism in two ways

1. Overloading
2. Overriding.

Overloading:

Classic overloading definition from book is “If a class have multiple methods by same name but different parameters, it is known as **Method Overloading**.”

Question: why we use same name for a method?

If we have to perform only one operation, having same name of the methods increases the readability of the program.

Suppose you have to perform addition of the given numbers but there can be any number of arguments, if you write the method such as **a(int,int)** for two parameters, and **b(int,int,int)** for three parameters then it may be difficult for you as well as other programmers to understand the behavior of the method because its name differs. So, we perform method overloading to figure out the program quickly.

Written by: **Shamik Mitra**
Technical Leader at IBM
blog: http://javaonfly.blogspot.in/
facebook : https://www.facebook.com/shamik.mitra.37
mob : **9830471739** mail **:mitrashamik@gmail.com**
Provide Tuition on java, j2ee Struts,Spring,Hibernate,jpa,Hadoop,Devops,Agile,kanban

Or think in that way if we have to perfor addition of float,double,int or all the we can say method name is same but parameters will be different.

Overloading Rule

1. Overloaded methods MUST change the argument list.

2. Overloaded methods CAN change the return type.

3. Overloaded methods CAN change the access modifier.

4 Overloaded methods CAN declare new or broader checked exceptions

5.Overloading does not depend on return type if in a class two method has same name same parameters but different return type this will not a valid overloading.

6. Overloading can happen in same class or in parent-child class.

So,

```
public void doX(int i) throws AirithmeticException;

public void doX(long j)

public int doX(double j)

private int doX(String j)

private void doX()

throws Exception;
```

all are valid overloading

Written by: **Shamik Mitra**
Technical Leader at IBM
blog: http://javaonfly.blogspot.in/
facebook : https://www.facebook.com/shamik.mitra.37
mob : **9830471739** mail **:mitrashamik@gmail.com**
Provide Tuition on java, j2ee Struts,Spring,Hibernate,jpa,Hadoop,Devops,Agile,kanban

Overloading Example:

```
package com.example.overloading;

publicclass Adder {

	publicvoid add(int a, int b)
	{
		int result=a+b;
		System.out.println("Result is in int version of add " + result);
	}

	publicvoid add(double a, double b)
	{
		double result=a+b;
		System.out.println("Result is in double version of add " + result);
	}

	publicstaticvoid main(String[] args) {
		Adder obj =new Adder();
		obj.add(1, 2);
		obj.add(1D, 5D);
	}

}
```

Output:

```
Result is in int version of add 3
Result is in double version of add 6.0
```

Why return type nor considered in Overloading?

Because of ambiguity problem. Let say overloading allow return type then I have two methods

```
 int add(int a,int b)
double add(int a,int b)
```

Now when we call

```
Adder adder =new Adder();
```

Written by: **Shamik Mitra**
Technical Leader at IBM
blog: http://javaonfly.blogspot.in/
facebook : https://www.facebook.com/shamik.mitra.37
mob : **9830471739** mail **:mitrashamik@gmail.com**
Provide Tuition on java, j2ee Struts,Spring,Hibernate,jpa,Hadoop,Devops,Agile,kanban

adder.add(1,2);// here what wii be the return type?

Now in above problem what will be the return type double or int, an ambiguity occurs.
Due to this reason return type not allowed in overloading.

Now we are talking about some puzzles.

Suppose I have two methods in a class call

doX(int i,long j)

doX(long j,int i,)

is this a valid overloading???

Answer is Yes although Data signature is different but here methods meaning are same as only arguments are swap their spaces

Another one

a. doX(int i)

b. doX(Integer i)

and we call it by doX(2);

can you guess which method will be called.

Think.....

Written by: **Shamik Mitra**
Technical Leader at IBM
blog: http://javaonfly.blogspot.in/
facebook : https://www.facebook.com/shamik.mitra.37
mob : **9830471739** mail **:mitrashamik@gmail.com**
Provide Tuition on java, j2ee Struts,Spring,Hibernate,jpa,Hadoop,Devops,Agile,kanban

It will call doX(int i)

Remember in overloading **primitive type**has precedence over **Boxing**.

More puzzle

doX(Integer i);

doX(int ... i);

now calling doX(2);

can you tell which version will be called.

It calls doX(Integer i);

Please remember in overloading, **Boxing** has precedence over **varargs** as varargs comes in 1.5 but Boxing concept is there from the start.

Adding more complexity

doX(Integer i)

doX(long i)

calling with do(2)

It will call doX(long i)

Written by: **Shamik Mitra**
Technical Leader at IBM
blog: http://javaonfly.blogspot.in/
facebook : https://www.facebook.com/shamik.mitra.37
mob : **9830471739** mail **:mitrashamik@gmail.com**
Provide Tuition on java, j2ee Struts,Spring,Hibernate,jpa,Hadoop,Devops,Agile,kanban

as in overloading **primitivewidening** has precedence over **Boxing.**

TIP: Widening means if datatype does not found exact version it will call closest grater datatype as here we know int byte short long are in same datatype so do(int i) found closest mismatch do(long i) by widening.

The Last puzzle:

doX(Object i)

doX(Long i)

call with doX(2) can you tell which version will be called?

it calls doX(Object i) version

In Overloading **Boxing then widening** is permissible, mean here conversion is

int->Integer->Object (boxing then Widening)

but doX(Long i) **int->long->Longwidening then boxing** will not allowed.

Overloading Puzzle Rule

1. Primitive widening uses the "smallest" method argument possible.
2. Used individually, boxing and var-args are compatible with overloading.
3. You can't widen from one wrapper type to another. (IS-A fails.)
4. You can't widen and then box. (An int can't become a Long.)
5. You can box and then widen. (An int can become an Object, via Integer.)
6. You can combine var-args with either widening or boxing.

Written by: **Shamik Mitra**
Technical Leader at IBM
blog: http://javaonfly.blogspot.in/
facebook : https://www.facebook.com/shamik.mitra.37
mob : **9830471739** mail **:mitrashamik@gmail.com**
Provide Tuition on java, j2ee Struts,Spring,Hibernate,jpa,Hadoop,Devops,Agile,kanban

Example with boxing varargs

```
package com.example.overloading;

publicclass BoxingVarargs {

	/*public void doIt(int i)
	{
		System.out.println("Call int version"  + i);
	}*/
	publicvoid doIt(Integer i)
	{
		System.out.println("Call Integer version"  + i);
	}

	publicvoid doIt(int... i)
	{
		System.out.println("Call varargs version"  + i);
	}


	publicstaticvoid main(String[] args) {

		BoxingVarargs obj = new BoxingVarargs();
		obj.doIt(2);


	}


}
```

Output:Call Integer version2

Written by: **Shamik Mitra**
Technical Leader at IBM
blog: http://javaonfly.blogspot.in/
facebook : https://www.facebook.com/shamik.mitra.37
mob : **9830471739** mail **:mitrashamik@gmail.com**
Provide Tuition on java, j2ee Struts,Spring,Hibernate,jpa,Hadoop,Devops,Agile,kanban

Example with Widening and boxing

```
package com.example.overloading;

publicclass WidenBoxing {

	publicvoid doIt(Object i)
	{
		System.out.println("Call box and widen int->Integer->Object  "   + i);
	}

	publicvoid doIt(Long i)
	{
		System.out.println("Call widen and box int->long->Long  not possible " + i);
	}

	publicstaticvoid main(String[] args) {

		WidenBoxing obj =new WidenBoxing();
		obj.doIt(2);
	}

}
```

Please note that if you comment the doIt(Object i) method. Compiler would complain no method found doIt(int i) ,as widen then box is not supported.

Last word about overloading in java we sometimes call it static binding or static polymorphism because which method will be called that will decide in compile time based on the reference and method signature.

Written by: **Shamik Mitra**
Technical Leader at IBM
blog: http://javaonfly.blogspot.in/
facebook : https://www.facebook.com/shamik.mitra.37
mob : **9830471739** mail **:mitrashamik@gmail.com**
Provide Tuition on java, j2ee Struts,Spring,Hibernate,jpa,Hadoop,Devops,Agile,kanban

Chapter 10

Overriding

Earlier in Inheritance chapter we talked about super classes and sub classes. If a class inherits a method from its super class, then there is a chance to redefine or override the method provided that it is not marked final.

The benefits of overriding are

Ability to define a behavior that's specific to the subclass type which means a subclass can implement a parent class method based on its requirement.

In object-oriented terms, overriding means to override the functionality of an existing method.

Let say there is a parent class call AreaFinder which has a method called calculateArea

If this AreaFinder has a subclass call Rectangle then it is Rectangle class duty to calculare area of rectangle as parent class does not know about the child or what formula has to use for calculate Rectangle as AreaFinder has broader scope, it does not know the details of child class. In fact, it is not possible, if a parent has 1000 child classes then it is not possible to Know the specific details of each subclass. Rather it is easy to delegate the responsibility to subclass for implement the logic. So it is the right moment for Overriding a method.

Written by: **Shamik Mitra**
Technical Leader at IBM
blog: http://javaonfly.blogspot.in/
facebook : https://www.facebook.com/shamik.mitra.37
mob : **9830471739** mail **:mitrashamik@gmail.com**
Provide Tuition on java, j2ee Struts,Spring,Hibernate,jpa,Hadoop,Devops,Agile,kanban

Example:

```
package com.example.overriding;

publicclassAreaFinder {

	publicvoid calculateArea()
	{
		System.out.println("Not know what to calculate :(");
	}

}
```

```
package com.example.overriding;

publicclass Rectangle extends AreaFinder{

	intlength=20;
	intbreadth=10;

	@Override
	publicvoid calculateArea()
	{
		System.out.println("Area is "+ length*breadth);
	}

	publicstaticvoid main(String[] args) {

		AreaFinder obj =new Rectangle();
		obj.calculateArea();

		AreaFinder obj1 =new AreaFinder();
		obj1.calculateArea();

		Rectangle obj2 =new Rectangle();
```

```
                obj2.calculateArea();




        }

}
```

Output:
Area is 200
Not know what to calculate :(
Area is 200

Pay attention to the first object creation

```
AreaFinder obj =new Rectangle();
obj.calculateArea();
```

This is called polymorphic assignment here parent reference holds child object.
Just I took few seconds to talk more about this

1. As it is a parent reference only method defines in parent are visible and you can call them, suppose you add a new method in a child so that method is not visible by parent reference.

2. Sometime we call overriding as *dynamic method dispatch or runtime polymorphism* as although using parent reference we call calculateArea(); it will not call parent's calculateArea it's call Rectangles calculate Area . May be the reference is parent but runtime jvm decides which Objects method should be called as in JVM parent reference point to

Rectangle Object it will call Rectangle's caculateArea provided caculateArea has been overridden perfectly in Rectangle.

Overriding Rules:

- The argument list should be exactly the same as that of the overridden method.
- The return type should be the same or a subtype of the return type declared in the original overridden method in the superclass it's call **covariant** return.
- The access level cannot be more restrictive than the overridden method's access level. For example: if the superclass method is declared **public** then the overridding method in the sub class cannot be either **private** or **protected**.
- Instance methods can be overridden only if they are inherited by the subclass.

Written by: **Shamik Mitra**
Technical Leader at IBM
blog: http://javaonfly.blogspot.in/
facebook : https://www.facebook.com/shamik.mitra.37
mob : **9830471739** mail **:mitrashamik@gmail.com**
Provide Tuition on java, j2ee Struts,Spring,Hibernate,jpa,Hadoop,Devops,Agile,kanban

- A method declared **final** cannot be **overridden**.
- A method declared **static** cannot be overridden but can be re-declared it is called shadowing .
- If a method cannot be inherited, then it cannot be overridden.
- A subclass within the same package as the instance's superclass can override any superclass method that is not declared private or final.
- A subclass in a different package can only override the non-final methods declared public or protected.
- An overriding method can throw any uncheck exceptions, regardless of whether the overridden method throws exceptions or not. However the overriding method should not throw **checked exceptions** that are **new or broader** than the ones declared by the overridden method. The overriding method can throw **narrower or fewer** exceptions than the overridden method.
- Constructors cannot be overridden.
- Variables can't be Overridden Override is for method.
- Using super keyword, we can call parents method which is overridden in child
- Interface declared method must be overridden in the first concreate class
- Always use @Override annotation to make sure the method has overridden successfully.
- If a method does not Override successfully in child it's ended up with an Overloaded version of that method.

Example

```
package com.example.overriding;

publicclass Animal {


        publicvoid makeSound()
        {
                System.out.println("No Sound");
        }

}
package com.example.overriding;
```

Written by: **Shamik Mitra**
Technical Leader at IBM
blog: http://javaonfly.blogspot.in/
facebook : https://www.facebook.com/shamik.mitra.37
mob : **9830471739** mail **:mitrashamik@gmail.com**
Provide Tuition on java, j2ee Struts,Spring,Hibernate,jpa,Hadoop,Devops,Agile,kanban

```
publicclass Dog extends Animal{

	@Override
	publicvoid makeSound()
	{
		System.out.println("Bark");
	}

}
package com.example.overriding;

publicclass Giraffe extends Animal{

	@Override
	publicvoid makeSound()
	{
		super.makeSound();
	}


}

package com.example.overriding;

publicclass Soundmanager {

	publicstaticvoid main(String[] args) {

		Animal animal = new Dog();
		animal.makeSound();
		animal = newGiraffe();
		animal.makeSound();
	}

}
```

Output:

Bark
No Sound

Here I did polymorphic assignment in SoundManager Class
First it contains Dog instance so it calls Dog's makeSound

Written by: **Shamik Mitra**
Technical Leader at IBM
blog: http://javaonfly.blogspot.in/
facebook : https://www.facebook.com/shamik.mitra.37
mob : **9830471739** mail **:mitrashamik@gmail.com**
Provide Tuition on java, j2ee Struts,Spring,Hibernate,jpa,Hadoop,Devops,Agile,kanban

Secondly I reassign it to Giraffe instance so it called Giraffe Version and if you pay attention to Giraffe makeSound method I call super. makeSound() incase of that it calls the Parentmake Sound method.

TIP: Remember in case of Overiding method name, signature and return type is same in child class . If it is not, then you will have ended up with an overloading. Return type may be subclass, we call it co-variant return

In above example
Animal makeSound() is in parent

Then We can say Dog makeSound() is a valid overriding.

Written by: **Shamik Mitra**
Technical Leader at IBM
blog: http://javaonfly.blogspot.in/
facebook : https://www.facebook.com/shamik.mitra.37
mob : **9830471739** mail **:mitrashamik@gmail.com**
Provide Tuition on java, j2ee Struts,Spring,Hibernate,jpa,Hadoop,Devops,Agile,kanban

Chapter 11

Abstraction

As per dictionary, Abstraction is the quality of dealing with ideas rather than events. It means hiding the implementation part and just show you the part as a user you want. for example, when you consider a driver driving a car, he does not know what is going on behind when he steers the wheel or press the break. But he knows how to operate the car, he needs steering, brake, accelerator to operate the car but don't bother about how they actually work and car moves.

likewise, in Object oriented programming Abstraction is a process of hiding the implementation details from the user, only the functionality will be provided to the user. In other words, user will have the information on **what the object does instead of how it does it.**

In Java Abstraction is achieved using Abstract classes, and Interfaces.

Abstract key words can be applied on

1. Class level
2. Method Level.

Written by: **Shamik Mitra**
Technical Leader at IBM
blog: http://javaonfly.blogspot.in/
facebook : https://www.facebook.com/shamik.mitra.37
mob : **9830471739** mail **:mitrashamik@gmail.com**
Provide Tuition on java, j2ee Struts,Spring,Hibernate,jpa,Hadoop,Devops,Agile,kanban

Abstract Class

A class which contains the abstract keyword in its declaration is known as abstract class.

An abstract class is not a concreate class like we created classes till now.

Abstract Class Rules:

- Abstract class may or may not contain ***abstract methods***ie., methods without body (ex. public void work();)
- But, if a class have at least one **abstract method**, then the class **must** be declared **abstract**.
- If a class is declared abstract it cannot be instantiated. Ex.(you can't do **new Abstractclass()**;)
- To use an abstract class, you have to extendabstract, provide implementations to the abstract methods in it and call the Concreate class but creating it's instance.
- If you inherit an abstract class, you have to provide implementations to all the abstract methods in it.
- An abstract class can extend another abstract class in this case it is possible that child abstract class does not override abstract methods declared in paren. But sooner or later there must be a concreate class in this hierarchy who will implement all the abstract methods of parents.
- Interface has by default abstract method.
- A 100 percent abstract class equivalent to an interface

Example:

```
package com.example.abstractExample;

publicabstractclass TvSpecification {

        private String brand;
        private String mode;
        privateintsize;
        private String led;

        publicTvSpecification()
```

```
        {
                init();
        }

        publicabstractvoid extraSpecification();
        publicabstractvoid init();

        public String getBrand() {
                returnbrand;
        }

        publicvoid setBrand(String brand) {
                this.brand = brand;
        }

        public String getMode() {
                returnmode;
        }

        publicvoid setMode(String mode) {
                this.mode = mode;
        }

        publicint getSize() {
                returnsize;
        }

        publicvoid setSize(int size) {
                this.size = size;
        }

        public String getLed() {
                returnled;
        }

        publicvoid setLed(String led) {
                this.led = led;
        }
        @Override
        public String toString() {
                return"TvSpecification [brand=" + brand + ", mode=" + mode + ", size="
                                + size + " inch , led=" + led + "]";
        }
```

```
}
```

```
package com.example.abstractExample;

publicclass Samsung extends TvSpecification{



	@Override
	publicvoid extraSpecification() {
		System.out.println("HDMI ready");
		System.out.println("100 PMPO woofer");


	}

	@Override
	publicvoid init() {
		this.setBrand("SamSung");
		this.setLed("LCD");
		this.setMode("Color");
		this.setSize(44);

	}

}
```

```
package com.example.abstractExample;

publicclass Sony extends TvSpecification{


	@Override
	publicvoid extraSpecification() {
		System.out.println("Full HD 1080");
		System.out.println("100 PMPO woofer with sorounding sound");
		System.out.println("3D Enabled");
	}
```

```
	@Override
	publicvoid init() {
		this.setBrand("Sony");
		this.setLed("LED");
		this.setMode("Color");
		this.setSize(52);

	}

}

package com.example.abstractExample;

publicclass TvShop {

	publicstaticvoidmain(String[] args) {


		TvSpecification obj = new Samsung();
		System.out.println(obj);
		obj.extraSpecification();
		System.out.println("*****************");

		 obj = new Sony();
		System.out.println(obj);
		obj.extraSpecification();

	}

}
```

Output:

```
TvSpecification [brand=SamSung, mode=Color, size=44 inch , led=LCD]
HDMI ready
100 PMPO woofer
*****************
TvSpecification [brand=Sony, mode=Color, size=52 inch , led=LED]
Full HD 1080
100 PMPO woofer with sorounding sound
3D Enabled
```

Written by: **Shamik Mitra**
Technical Leader at IBM
blog: http://javaonfly.blogspot.in/
facebook : https://www.facebook.com/shamik.mitra.37
mob : **9830471739** mail **:mitrashamik@gmail.com**
Provide Tuition on java, j2ee Struts,Spring,Hibernate,jpa,Hadoop,Devops,Agile,kanban

Chapter 12

Some Extra information but Useful

Object Class :

The **java.lang.Object** class is the root of the class hierarchy. Every class has Object as a superclass. All objects, including arrays, implement the methods of this class.

So every class you created or already in java inherit Object class. It is in **java.lang** package. java.lang package is a special package Integer ,String ,Array this class are in java.lang package and it is need not to import compiler automatically insert it for you.

Object class has some important methods which are very useful.

1	protected Object clone()	This method creates and returns a copy of this object.
2	boolean equals(Object obj)	This method indicates whether some other object is "equal to" this one.
3	protected void finalize()	This method is called by the garbage collector on an object when garbage collection determines that there are no more references to the object.
4	Class<?> getClass()	This method returns the runtime class of this Object.
5	int hashCode()	This method returns a hash code value for the object.
6	void notify()	

Written by: **Shamik Mitra**
Technical Leader at IBM
blog: http://javaonfly.blogspot.in/
facebook : https://www.facebook.com/shamik.mitra.37
mob : **9830471739** mail **:mitrashamik@gmail.com**
Provide Tuition on java, j2ee Struts,Spring,Hibernate,jpa,Hadoop,Devops,Agile,kanban

This method wakes up a single thread that is waiting on this object's monitor.

void notifyAll()

7

This method wakes up all threads that are waiting on this object's monitor.

String toString()

8

This method returns a string representation of the object.

void wait()

9

This method causes the current thread to wait until another thread invokes the notify() method or the notifyAll() method for this object.

void wait(long timeout)

1
0

This method causes the current thread to wait until either another thread invokes the notify() method or the notifyAll() method for this object, or a specified amount of time has elapsed.

void wait(long timeout, int nanos)

This method causes the current thread to wait until another thread invokes the notify() method or the notifyAll() method for this object, or some other thread interrupts the current thread, or a certain amount of real time has elapsed.

boolean equals(Object obj):

1
1

The **java.lang.Object.equals(Object obj)** indicates whether some other object is **"equal to" or meaningful equivalent** to this one. **==** operator checks two objects **memory location** are same or not where as **equals** check two objects are **meaningfully equivalent or not**.

If in subclass you not override equals then the meaning of == and equalsis same

The equals method for class Object implements the most discriminating possible equivalence relation on objects; that is, for any non-null reference values x and y, this method returns true if and only if x and y refer to the same object (x == y has the value true).

Note that it is generally necessary to override the hashCode method whenever this method is overridden, so as to maintain the general contract for the hashCode method,

which states that equal objects must have equal hash codes.

equals Rule:

- It is**reflexive**: for any reference value x, x.equals(x) should return true.
- It is**symmetric**: for any reference values x and y, x.equals(y) should return true if and only if y.equals(x) returns true.
- It is**transitive**: for any reference values x, y, and z, if x.equals(y) returns true and y.equals(z) returns true, then x.equals(z) should return true.
- It is**consistent**: for any reference values x and y, multiple invocations of x.equals(y) consistently return true or consistently return false, provided no information used in equals comparisons on the object is modified.

- For any non-null reference value x, x.equals(null) should return false.

Let say we can say Two employees are equal if and only id employee ID is same for two.

Example:

```
package com.example.ObjectMethod;

publicclass Employee {

	privateintempId;
	private String name;
	publicint getEmpId() {
		returnempId;
	}
	publicvoid setEmpId(int empId) {
		this.empId = empId;
	}
	public String getName() {
		returnname;
	}
	publicvoid setName(String name) {
		this.name = name;
	}


	@Override
```

```
        publicint hashCode() {
                finalint prime = 31;
                int result = 1;
                result = prime * result + empId;
                return result;
        }


        @Override
        publicboolean equals(Object obj) {
                if (this == obj)
                        returntrue;
                if (obj == null)
                        returnfalse;
                if (getClass() != obj.getClass())
                        returnfalse;
                Employee other = (Employee) obj;
                if (empId != other.empId)
                        returnfalse;
                returntrue;
        }



}
package com.example.ObjectMethod;

publicclass EqualsTest {
        publicstaticvoid main(String[] args) {

                Employee emp = new Employee();
                emp.setName("James Bond");
                emp.setEmpId(7);



                Employee emp1 = emp; // two refrence point to same Object so == test
will pass

                Employee emp2 = new Employee();
                emp.setName("James Bond");
                emp.setEmpId(7);// Create a new Object with sa,e parameters so euals
test pass but == test fails as memory location is different.


                System.out.println("== test for emp and emp1 refrence which points to
same object value is " + (emp==emp1));
```

```
                System.out.println("== test for emp and emp2 refrence which points to diff
object value is " + (emp==emp2));

                System.out.println("equals test for emp and emp1 they are same object so
value is " + emp.equals(emp1));

                System.out.println("equals test for emp and emp2 they are meaning fully
equvalent but diff object so value is " + emp.equals(emp1));



        }

}
```

Output:

```
== test for emp and emp1 refrence which points to same object value is true
== test for emp and emp2 refrence which points to diff object value is false
equals test for emp and emp1 they are same object so value is true
equals test for emp and emp2 they are meaning fully equvalent but diff object so value
is true
```

In Employee class I override the Object's equal method and put a logic if empId are same then it returns true.

In Equals Test class I test two objects are meaning fully equal or not by invoking equals method.

hashcode:

The **java.lang.Object.hashCode()** method returns a hash code value for the object. This method is supported for the benefit of hashtables such as those provided by java.util.Hashtable. Actually it generates a int using hashing algorithm so Object can be searched easily. We will talk it later when we discuss about **collection** frame work.

Please remember whenever you override equals please override hashcode. As they ave maintained a contract. If you not override hashcode then uncanny situations can occur. And You scratch the head but not detect the problem. ☺

Hashcode contract:

Whenever it is invoked on the same object more than once during an execution of a Java application, the hashCode() method must consistently return

the same integer, provided no information used in equals() comparisons on the object is modified. This integer need not remain consistent from one execution of an application to another execution of the same application.

■ If two objects are equal according to the equals(Object) method, then calling the hashCode() method on each of the two objects must produce the same integer result.

■ It is NOT required that if two objects are unequal according to the equals(java.lang.Object) method, then calling the hashCode() method on each of the two objects must produce distinct integer results. However, the programmer should be aware that producing distinct integer results for unequal objects may improve the performance of hashtables.

toString Method :

Object class offers this toString() method. Purpose of this method is provide a String representation of the Object. Object is nothing bit a state i.e it has properties which contains certain value till now IF we have to print that value we use **System.out.println**() then print that properties. But If I want to print all the properties or want to know the details about object, toString() is very useful Just override the toString() method and write the property's value. Upon call toString() it will show the state.

Please remember all class inherit toString() method it is up to you, are you want to override it or not but best practice is override the toString() so we can see the details of the object. If you not override it's show a hexadecimal number i.e the **memory location of the object.**

Example:

```
package com.example.toString;

publicclass House {

        private String length="10X10 feet";
        private String color="Pink";
        private String furnished="Furnished";
        public String decoration="Well decorated with deluxe bed";


        //here override the toString method so it can print details
        @Override
```

Written by: **Shamik Mitra**
Technical Leader at IBM
blog: http://javaonfly.blogspot.in/
facebook : https://www.facebook.com/shamik.mitra.37
mob : **9830471739** mail **:mitrashamik@gmail.com**
Provide Tuition on java, j2ee Struts,Spring,Hibernate,jpa,Hadoop,Devops,Agile,kanban

```
        public String toString() {
                return"House [length=" + length + ", color=" + color + ", furnished="
                                + furnished + ", decoration=" + decoration + "]";
        }


        publicstaticvoid main(String[] args) {

                House house = new House();
                System.out.println(house); //here we not call toString but Sys-
tem.out.println always call toString for you.

        }
```

Output: House [length=10X10 feet, color=Pink, furnished=Furnished, decoration=Well decorated with deluxe bed]

Please notice in System.*out*.println(house); I just pass the house reference., But remember System.*out*.println() call toString on the reference internally.

Parametric Constructor:

Constructor can be parametric,so we can define a constructor which can takes argument. But what the heck we need it?

The answer is for design the architecture ☺

As a java developer you have to perform Architect role sometimes,

Suppose you have a car. The most and must important thing is make it legal so we need a **registration number** right? So I can say if a car is not registered that is illegal. But in India there is no place for any illegal thing. So I can say Registration number is a **must has property** in car. But the other property like AC, car decoration, FM are **optional** ,if a car does not have those properties then no problem will occurs. So as a developer your job is to make sure everybody **who owns the car must do the Registration**. Here is the trick you can make a **parametric constructor** which take registration number as input. By doing this you can make sure when someone try to **create a Car**

object he has to provide registration number otherwise he can't create a car object be-

cause there is no blank constructor and as you provide the **constructor**so compiler does not inject any **blank constructor**.

***TIP: Things to remember make sure required properties are always going through constructor argument and optional can go by setter/getter*.**

Example:

```
package com.example.constructor;

publicclass Car {


	private String registrationNumber;//required

	private String ac;//optional


	Car(String registrationNumber)
	{
		this.registrationNumber=registrationNumber;

	}




	public String getAc() {
		returnac;
	}




	publicvoid setAc(String ac) {
		this.ac = ac;
	}




	@Override
	public String toString() {
		return"Car [registrationNumber=" + registrationNumber + ", ac=" + ac
				+ "]";
```

```
	}



	publicstaticvoid main(String[] args) {

		//Car car = new Car(); can't do as no blank constructor is there
		Car car = new Car("WB0442323");
		car.setAc("AC");
		System.out.println(car);
	}
}
```

Output: Car [registrationNumber=WB0442323, ac=AC]

TIP: In inheritance if any of the parent create a parametric constructor and does not provide a blank constructor. Remember child must have the same parametric constructor and Explicitly call parametric constructor of Parent using super(parametric value) , unless If child has a blank constructor it calls parent blank constructor via super() but parent does not have it so compiler will complain and Object will not create.

Example:

```
package com.example.constructor;

publicclass Parent {

	public Parent(String str)
	{
		System.out.println("parent ::" + str);
	}

}

package com.example.constructor;

publicclass Child extends Parent{

	public Child(String str)
	{
		super(str);//Has to invoke parametric version as parent does not has any
blank constructor
```

```
            System.out.println("Child :" + str);
        }

        publicstaticvoid main(String[] args) {

            Child c =new Child("Hello");


        }

}
```

Here in Child constructor we have to call parent's parametric constructor explicitly as parent does not have any blank constructor.

Casting:

In java change a reference to a type to another type call casting. Actually casting involves in parent-child relationship, Polymorphic assignment is an example of casting.

There are two types of casting in java

1. Upcasting : Java permits an object of a subclass type to be treated as an object of any superclass type. This is called upcasting. For Upcasting we don't have to do any explicit casting it does automatically.

2. Downcasting : Java permits an reference of a superclass which point to an subclass object can be casted to that subclass. This is called Downcasting. For Downcasting we have to cast the type explicitely.

Example:

```
package com.example.casting;

publicclass Animal {

        publicvoid sound()
        {
            System.out.println("Generic Sound");
        }
```

```
}

package com.example.casting;

publicclass Dog extends Animal{

        publicvoid sound()
        {
                System.out.println("Bark");
        }

}

package com.example.casting;

publicclass Casting {

        publicstaticvoid main(String[] args) {

             Animal a = new Dog(); //Upcasting
           a.sound();

            Dog d = (Dog)a;//Downcasting
            d.sound();

        }

}
```

Here in Casting class First we did a polymorphic assignment i.e example of Upcasting. But in second scenario We know that ref a points to a Dog object so I can refer it to a Dog reference but “a” is a reference of Animal so we explicitly cast “a” to Dog, we call it Down casting.

Constructor Overloading:

Written by: **Shamik Mitra**
Technical Leader at IBM
blog: http://javaonfly.blogspot.in/
facebook : https://www.facebook.com/shamik.mitra.37
mob : **9830471739** mail **:mitrashamik@gmail.com**
Provide Tuition on java, j2ee Struts,Spring,Hibernate,jpa,Hadoop,Devops,Agile,kanban

As like methods constructor can be overloaded. Using overloaded constructor, we got opportunity to create object by chose a form.

Suppose I want to create and use a Car object in a business logic,where only registration is matter so I only provide that property value not provide the AC property so here constructor overloading required.

Example:

```
package com.example.constructor;

publicclass BMW {

	private String regNo;
	private String AC;


	public BMW()
	{

	}

	public BMW(String regNo)
	{
		this.regNo=regNo;
	}


	public BMW(String regNo,String AC)
	{
		this.regNo=regNo;
		this.AC=AC;
	}



	@Override
	public String toString() {
		return"BMW [regNo=" + regNo + ", AC=" + AC + "]";
	}

	publicstaticvoid main(String[] args) {
		BMW bmw = new BMW();
		BMW bmw1 = new BMW("WB0345");
		BMW bmw2 = new BMW("WB0345","AC");
```

```
                System.out.println("call blank constructor" + bmw);
                System.out.println("call registration only constructor" + bmw1);
                System.out.println("call with full implement constructor" + bmw2);

        }

}
```

Output:
call blank constructorBMW [regNo=null, AC=null]
call registration only constructorBMW [regNo=WB0345, AC=null]
call with full implement constructorBMW [regNo=WB0345, AC=AC]

Static method shadowing:

I know you people are very studios so you remember that static is not associated with object it is something which has one copy and share between object. And Overriding is a phenomena of an object so Static method overriding is not possible If same name method has in parent and child then by the reference it calls the actual version of static method actually reference change to class so this call it's version. It is called method shadowing.

Example:

```
package com.example.Methodshadowing;

publicclassMethodshadowing {


        publicstaticvoid display()
        {
                System.out.println("I am parent's static version");
        }


}
```

```
package com.example.Methodshadowing;

publicclass Child extends Methodshadowing{

        publicstaticvoid display()
        {
```

```
                System.out.println("I am child's static version");
        }



        publicstaticvoid main(String[] args) {
                Methodshadowing obj = new Methodshadowing();//polymorphic assign-
ment
                obj.display();

                Child child = new Child();
                child.display();

        }



}
```

Output:
I am parent's static version
I am child's static version

Please note although I make a polymorphic assignment
Methodshadowing obj = new Methodshadowing();//polymorphic assignment
obj.display();

but display is sataic method so obj.display() replace with Methodshadowing.display();

and call the display method in parent.

Don’t fool with this scenario. Interviewer want to puzzle you but always remember over-riding is related to object ans static not related to objects so Overriding is not possible with static.

Written by: **Shamik Mitra**
Technical Leader at IBM
blog: http://javaonfly.blogspot.in/
facebook : https://www.facebook.com/shamik.mitra.37
mob : **9830471739** mail **:mitrashamik@gmail.com**
Provide Tuition on java, j2ee Struts,Spring,Hibernate,jpa,Hadoop,Devops,Agile,kanban

Chapter 13

Java Exception

Everything is not perfect in this world, how you can Expect in java everything will be perfect?

Exception or trouble or which is not intended or abnormal situation is inevitable.

So java need to handle this so java comes with Exception.

An exception (or exceptional event) is a problem that arises during the **execution** of a program. When an Exception occurs the normal flow of the program is disrupted and the program/Application terminates abnormally, which is not recommended, therefore these exceptions are to be handled.

An exception can occur for many different reasons, below given are some scenarios where exception occurs.

- A user has entered invalid data.
- A file that needs to be opened cannot be found.
- A network connection has been lost in the middle of communications or the JVM has run out of memory.

Some of these exceptions are caused by user error, others by programmer error, and others by physical resources that have failed in some manner.

Based on these we have three categories of Exceptions you need to understand them to know how exception handling works in Java,

Three Kind of Exception in java

3. Checked Exception
4. Unchecked Exception
5. Error

Written by: **Shamik Mitra**
Technical Leader at IBM
blog: http://javaonfly.blogspot.in/
facebook : https://www.facebook.com/shamik.mitra.37
mob : **9830471739** mail **:mitrashamik@gmail.com**
Provide Tuition on java, j2ee Struts,Spring,Hibernate,jpa,Hadoop,Devops,Agile,kanban

Checked Exception:

A checked exception is an exception that occurs at the compile time, these are also called as compile time exceptions. These exceptions cannot simply be ignored at the time of compilation; the Programmer should take care of (handle) these exceptions.

For example, if you use **FileReader** class in your program to read **data from a file**, if the file specified in its constructor doesn't exist, then an ***FileNotFoundException***occurs, and compiler prompts the programmer to handle the exception.

```
import java.io.File;

import java.io.FileReader;



public class FilenotFound {



   public static void main(String args[]){

      File file=new File("C://file.txt");

      FileReader fr = new FileReader(file);

   }



}
```

If you try to compile this compile would complain as You want to read a file using File reader so it is a risky operation as this file can be present or not. So Java does not take risk and instruct you to hadle the situation if file is not found. If you not provide it, then your class is not compile. Compiler will escalate against you!!!! And force you to handle abnormal situation.

Compile will say

```
javac FilenotFound.java

FilenotFound_Demo.java:8: error: unreported exception FileNotFoundException; must
be caught or declared to be thrown

    FileReader fr = new FileReader(file);

                    ^

1 error
```

Unchecked exceptions:

An Unchecked exception is an exception that occurs at the time of execution, these are also called as Runtime Exceptions, these include programming bugs, such as logic errors or improper use of an API. runtime exceptions are ignored at the time of compilation. Compiler not force you to handle abnormal situation as improper execution is due to you and you are responsible for that not java. It is java developer's duty to provide good business logic that can't fail in runtime. The best Patrice is always go for default path if abnormal situation occurs so that program cannot terminate so upon judging the situation you can provide a handle mechanism.

For example, if you have declared an array of size 5 in your program, and trying to call the 6th element of the array then an *ArrayIndexOutOfBoundsExceptionexception* occurs.

```
public class Unchecked {


  public static void main(String args[]){

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

    System.out.println(num[5]);
```

Written by: **Shamik Mitra**
Technical Leader at IBM
blog: http://javaonfly.blogspot.in/
facebook : https://www.facebook.com/shamik.mitra.37
mob : **9830471739** mail **:mitrashamik@gmail.com**
Provide Tuition on java, j2ee Struts,Spring,Hibernate,jpa,Hadoop,Devops,Agile,kanban

```
    }


}
```

If compile and run it throws a run time exception Exception in thread "main"

java.lang.ArrayIndexOutOfBoundsException: 5

at Exceptions.Unchecked_Demo.main(Unchecked.java:8)

Errors:

These are not exceptions at all, but problems that arise beyond the control of the user or the programmer. Errors are typically ignored in your code because you can rarely do anything about an error. For example, if a stack overflow occurs, an error will arise. They are also ignored at the time of compilation. You can't do anything about it.

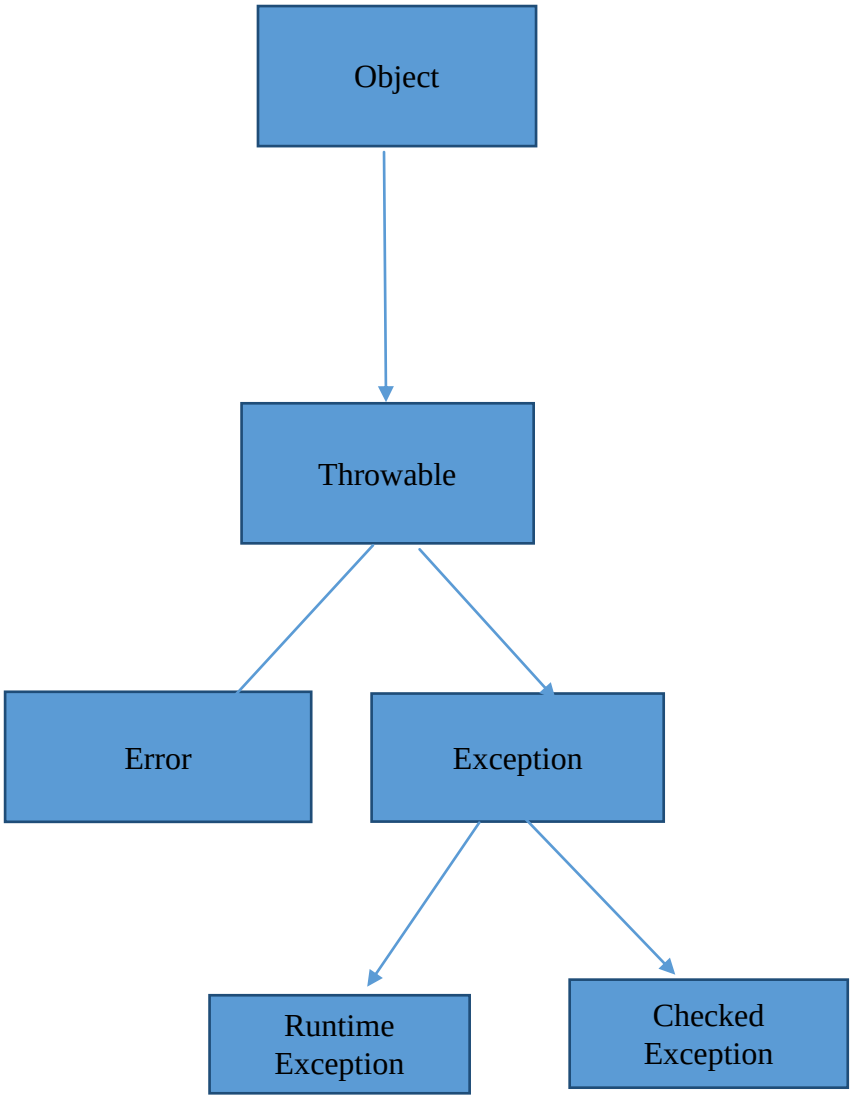
Exception Hierarchy:

All exception classes are subtypes of the **java.lang.Exception** class. The Exception class itself is a subclass of the **Throwable** class.Throwable is inherited from **Object** class Other than the exception class there is another subclass called **Error** which is derived from the **Throwable** class.

Errors are abnormal conditions that happen in case of severe failures, these are not handled by the java programs. Errors are generated to indicate errors generated by the runtime environment. Example : JVM is out of Memory. Normally programs cannot recover from errors.

The Exception class has two main subclasses: IOException class and RuntimeException Class.

Written by: **Shamik Mitra**
Technical Leader at IBM
blog: http://javaonfly.blogspot.in/
facebook : https://www.facebook.com/shamik.mitra.37
mob : **9830471739** mail **:mitrashamik@gmail.com**
Provide Tuition on java, j2ee Struts,Spring,Hibernate,jpa,Hadoop,Devops,Agile,kanban



Exception Hierarchy

Important Exception Method:

1 **public String getMessage()**

Returns a detailed message about the exception that has occurred. This message is initialized in the Throwable constructor.

2 **public Throwable getCause()**

Returns the cause of the exception as represented by a Throwable object.

3 **public String toString()**

Returns the name of the class concatenated with the result of getMessage()

4 **public void printStackTrace() V.V.IMP**

Prints the result of toString() along with the stack trace to System.err, the error output stream.

5 **public StackTraceElement [] getStackTrace() -- imp**

Returns an array containing each element on the stack trace. The element at index 0 represents the top of the call stack, and the last element in the array represents the method at the bottom of the call stack.

6 **public Throwable fillInStackTrace()**

Fills the stack trace of this Throwable object with the current stack trace, adding to any previous information in the stack trace.

Catching Exception:

A method catches an exception using a combination of the **try** and **catch** keywords. A try/catch block is placed around the code that might generate an exception or we can say this block of code is potential danger code. Code within a try/catch block is referred to as protected code, and the syntax for using try/catch looks like the following:

Syntax

```
try
{
//Protected code
}catch(ExceptionName e1)
{
//Catch block
}
```

The code which is prone to exceptions is placed in the try block, when an exception occurs, that exception occurred is handled by catch block associated with it. Every try block should be immediately followed **either by a catch block or finally block**.

A catch statement involves declaring the type of exception you are trying to catch. If an exception occurs in protected code, the catch block (or blocks) that follows the try is checked. If the type of exception that occurred is listed in a catch block, the exception is passed to the catch block much as an argument is passed into a method parameter.

Written by: **Shamik Mitra**
Technical Leader at IBM
blog: http://javaonfly.blogspot.in/
facebook : https://www.facebook.com/shamik.mitra.37
mob : **9830471739** mail **:mitrashamik@gmail.com**
Provide Tuition on java, j2ee Struts,Spring,Hibernate,jpa,Hadoop,Devops,Agile,kanban

Example: an array is declared with 2 elements. Then the code tries to access the 3rd element of the array which throws an exception.

```
package com.example.exception;

publicclass ExceptionTest {

	publicint[] arr = {1,2};

	publicvoid access(){

		try
		{
			System.out.println("Access a 3rd element that does not Exist");
			intresult = arr[2];
		}
		catch(ArrayIndexOutOfBoundsException e)
		{
			e.getMessage();
			e.printStackTrace();
		}



	}

	publicstaticvoid main(String[] args) {

		ExceptionTest obj = new ExceptionTest();
		obj.access();

	}


}
```

Output:

```
Access a 3rd element that does not Exist
java.lang.ArrayIndexOutOfBoundsException: 2
	at com.example.exception.ExceptionTest.access(ExceptionTest.java:12)
	at com.example.exception.ExceptionTest.main(ExceptionTest.java:28)
```

Written by: **Shamik Mitra**
Technical Leader at IBM
blog: http://javaonfly.blogspot.in/
facebook : https://www.facebook.com/shamik.mitra.37
mob : **9830471739** mail **:mitrashamik@gmail.com**
Provide Tuition on java, j2ee Struts,Spring,Hibernate,jpa,Hadoop,Devops,Agile,kanban

Multiple Catch Block:

A try block can be followed by multiple catch blocks. The syntax for multiple catch blocks looks like the following:

Syntax:

```
try
{
//Protected code
}catch(ExceptionType1 e1)
{
//Catch block
}catch(ExceptionType2 e2)
{
//Catch block
}catch(ExceptionType3 e3)
{
//Catch block
}
```

The syntax demonstrates three catch blocks, but you can have **any number of catch** after a single **try**. If an exception occurs in the protected code, the exception is thrown to the first catch block in the list. If the data type of the exception thrown matches ExceptionType1, it gets caught there. If not, the exception passes down to the second catch statement. This continues until the exception either is caught or falls through all catches, in which case the current method stops execution and the exception is thrown down to the previous method on the call stack.

TIP:
Please remember when you use multiple catch block it should be narrower to broder exception or subclass to superclass. Say a try has two block ArithmeticException and Exception if Exception block is prior to Arithmetic catch block it should catch the Exception as it is super class of Exception So Arithmetic block will be unreachable so always use subclass to super class unless compiler would complain.

Example***:***

```
package com.example.exception;

publicclass MultipleCatch {

        publicvoid test()
        {
```

```
                try
                {
                        intresult =1/0;

                }
                catch(ArithmeticException ex)
                {
                        System.out.println("In Arithmetic Exception block");
                        ex.printStackTrace();
                }
                catch(Exception ex)
                {
                        System.out.println("In  Exception block");
                        ex.printStackTrace();
                }
        }

        publicvoid test(String str)
        {
                try
                {
                        str.toLowerCase();

                }
                catch(ArithmeticException ex)
                {
                        System.out.println("In Arithmetic Exception block");
                        ex.printStackTrace();
                }
                catch(Exception ex)
                {
                        System.out.println("In  Exception block");
                        ex.printStackTrace();
                }
        }

        publicstaticvoid main(String[] args) {

                MultipleCatch obj = new MultipleCatch();
                obj.test();
                obj.test(null);

        }

}
```

Output :

Written by: **Shamik Mitra**
Technical Leader at IBM
blog: http://javaonfly.blogspot.in/
facebook : https://www.facebook.com/shamik.mitra.37
mob : **9830471739** mail **:mitrashamik@gmail.com**
Provide Tuition on java, j2ee Struts,Spring,Hibernate,jpa,Hadoop,Devops,Agile,kanban

```
In Arithmetic Exception block
java.lang.ArithmeticException: / by zero
        at com.example.exception.MultipleCatch.test(MultipleCatch.java:9)
        at com.example.exception.MultipleCatch.main(MultipleCatch.java:46)
java.lang.NullPointerException
        at com.example.exception.MultipleCatch.test(MultipleCatch.java:28)
        at com.example.exception.MultipleCatch.main(MultipleCatch.java:47)
In  Exception block
```

Throw and Throws Keywords:

If a method does not handle a checked exception, the method must declare it using the **throws** keyword. The throws keyword appears at the end of a method's signature. You have two choices either take the decision and handle exception when abnormality occurs i.e try/catch either play safely and say I am not going to take decision I pass this to caller to take decision i.e throws.

You can **throw** an exception, either a newly instantiated one or an exception that you just caught, by using the **throw** keyword.

Try to understand the difference between throws and throw keywords, *throws* is used to postpone the handling of a checked exception and *throw* is used to invoke an exception explicitly.

Example**:**

```
package com.example.exception;

publicclass ThrowException {

        publicvoid test() throws ArithmeticException
        {
                intresult=1/0;

        }

        publicvoid test(String str) throws NullPointerException
        {

                try
                {
                        str.toLowerCase();

                }
```

```
                catch(NullPointerException ex){

                        throw ex;
                }


        }


        publicstaticvoid main(String[] args) {

                ThrowException obj= new ThrowException();
                try
                {
                        obj.test();


                }
                catch(ArithmeticException ex)
                {
                        ex.printStackTrace();
                        obj.test(null);
                }
                catch(NullPointerException ex)
                {
                        ex.printStackTrace();
                }


        }

}
```

Output:

```
java.lang.ArithmeticException: / by zero
        at com.example.exception.ThrowException.test(ThrowException.java:7)
        at com.example.exception.ThrowException.main(ThrowException.java:33)
Exception in thread "main" java.lang.NullPointerException
        at com.example.exception.ThrowException.test(ThrowException.java:16)
        at com.example.exception.ThrowException.main(ThrowException.java:40)
```

In test method I handle the exception using try/catch then throw the Exception to caller which is **main** method, here we catch the exception again and print the stack trace.

Written by: **Shamik Mitra**
Technical Leader at IBM
blog: http://javaonfly.blogspot.in/
facebook : https://www.facebook.com/shamik.mitra.37
mob : **9830471739** mail **:mitrashamik@gmail.com**
Provide Tuition on java, j2ee Struts,Spring,Hibernate,jpa,Hadoop,Devops,Agile,kanban

finally block:

The finally block follows a **try block** or a **catch block**. A finally block of code always executes, irrespective of occurrence of an Exception.

Using a finally block allows you to run any cleanup-type statements that you want to execute, no matter what happens in the protected code.

It is good practice to use finally block with try/catch. Finally block always execute irrespective of exception occurs or clean processing but after each case we need to release costly resources like Database Connection, Statement, any large File Stream etc. So finally block is right place for that as it is always executing.

TIP: Pease note that any clean up code has to be written in final block as it executes always unless JVM shutdown using System.exit(0) so in try or catch if last statement is System.exit(0) in that case finally will not be executed.

Syntax:

```
try{

}

catch(Exception1){

{

catch(Exception2){

{



catch(Exception3){

{

finally
```

{

//Always execute

}

Or

try

{

{

finally

{

}

Example:

```
package com.example.exception;

publicclass FinallyTest {

        publicvoid testFinally() throws ArithmeticException
        {
                try
                {
                        System.out.println("In try block");
                        intresult = 1/0;
                }
                catch(ArithmeticException ex)
                {
                        System.out.println("In Exception block");
                        throw ex;
                }
                finally
                {
                        System.out.println("In Finally block");
```

```
                }
        }

        publicstaticvoid main(String[] args) {

                FinallyTest test=new FinallyTest();
                test.testFinally();


        }

}
```

Output:

```
In try block
Exception in thread "main" In Exception block
In Finally block
java.lang.ArithmeticException: / by zero
        at com.example.exception.FinallyTest.testFinally(FinallyTest.java:10)
        at com.example.exception.FinallyTest.main(FinallyTest.java:26)
```

Note that Although exception occurs it execute the finally block.

In case of **System.exit(0)finally** block is not executed.

Example:

```
package com.example.exception;

publicclass SystemExit {

        publicvoid test()
        {
                try{

                        System.out.println("In try block");
                        System.exit(0);

                }
                finally
                {
                        System.out.println("In finally block");
```

```
                }
        }

        publicstaticvoid main(String[] args) {
                SystemExit exit = new SystemExit();
                exit.test();
        }

}
```

Output:

In try block

TIP: Please note that checked exception has to be handled either by try catch block or delegation the responsibility to caller but in both cases you have to handled it programmatically.

In case of Runtime Exception, you have provision not to handle as this can occur in runtime so code can't not take preventive action on them basically it occurs due to bad logic implementation so logic has to be fixed rather than handle it. So try/catch or throws optional here.

TIP: In a code you can java try/catch/finally or try/finally or try/multiple catch/finally or try/catch but never ever any catch or finally block without try.

User Defined Exception:

You can create your own exceptions in Java. Keep the following points in mind when writing your own exception classes:

- All exceptions must be a child of Throwable.
- If you want to write a checked exception that is automatically enforced by the Handle or Declare Rule, you need to extend the **Exception** class.
- If you want to write a **runtime exception**, you need to extend the **RuntimeEx-**

ception class.

We can define our own Exception class as below:

```
package com.example.exception;

publicclassAgeValidationExceptionextends Exception{

	intage;
	String msg;

	AgeValidationException(String msg,int age)
	{
		this.age=age;
		this.msg=msg;
	}

	publicvoid printMessage()
	{
		System.out.println(msg + age);
	}


}
package com.example.exception;

publicclass Movie {
	intage;
	publicvoid watchMovie()
	{
		System.out.println("Movie is very good");
	}

	publicint getAge() {
		returnage;
	}

	publicvoid setAge(int age) throwsAgeValidationException {
		if (age <18)
		{
			thrownewAgeValidationException("Age can not be less than 18
,given ", age);
		}
		this.age = age;
	}
```

```
publicstaticvoid main(String[] args) {

        try
        {
                Movie mv =new Movie();
                mv.setAge(11);
                mv.watchMovie();

        }
        catch(AgeValidationException ex)
        {
                ex.printMessage();
        }

}
```

Here I create my own exception AgeValidationException if age is less than 18 then the person not allowed to watch movie.

TIP: Please note that when you extends Exception class to create your custom exception it is always a checked Exception so it has to be handled or delegate to caller by throws.

To create Custom Runtime Exception , RunTimeException class has to be extended.

One more thing Error cannot be handled but as it is a subclass of Throwable class you can catch it but you can do nothing because it not your programming fault this is due to environment issues which can't be controlled.

Like earthquake you can measure it predict it but not do anything to prevent it.

Chapter 14

Java File and I/O:

In java when you write a program very often you need to read file or write the output in to a file. Java provides standard api for that we called it File and I/O API.

Written by: **Shamik Mitra**
Technical Leader at IBM
blog: http://javaonfly.blogspot.in/
facebook : https://www.facebook.com/shamik.mitra.37
mob : **9830471739** mail **:mitrashamik@gmail.com**
Provide Tuition on java, j2ee Struts,Spring,Hibernate,jpa,Hadoop,Devops,Agile,kanban

Java uses the concept of stream to make I/O operation fast and read and write to a file. The java.io package contains all the classes required for input and output operations.

There are two types of stream in java by which we can read or write in a file or do I/O operations.

InputStream :

Java application uses an input stream to read data from a source, it may be a file,an array, peripheral device or socket.

OutputStream:

Java application uses an output stream to write data to a destination, it may be a file, an array, peripheral device or socket.

Written by: **Shamik Mitra**
Technical Leader at IBM
blog: http://javaonfly.blogspot.in/
facebook : https://www.facebook.com/shamik.mitra.37
mob : **9830471739** mail **:mitrashamik@gmail.com**
Provide Tuition on java, j2ee Struts,Spring,Hibernate,jpa,Hadoop,Devops,Agile,kanban

Stream pictorial view

OutputStream class:

OutputStream is an abstract class. It is the superclass of all classes representing an output stream of bytes. An output stream accepts output bytes and sends them to some source like File, array, Socket etc.

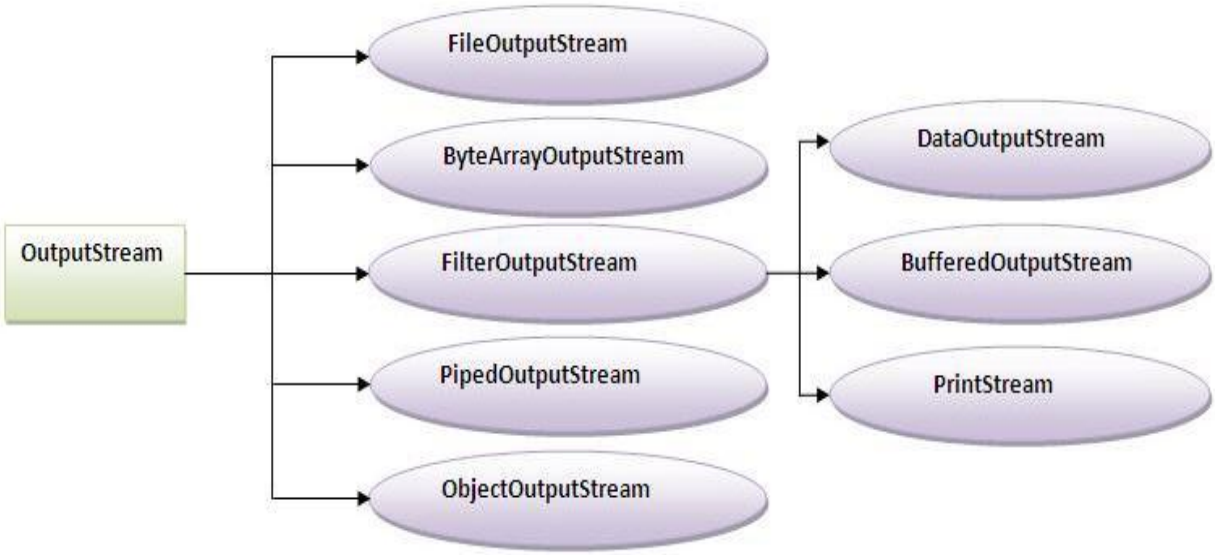
Written by: **Shamik Mitra**
Technical Leader at IBM
blog: http://javaonfly.blogspot.in/
facebook : https://www.facebook.com/shamik.mitra.37
mob : **9830471739** mail **:mitrashamik@gmail.com**
Provide Tuition on java, j2ee Struts,Spring,Hibernate,jpa,Hadoop,Devops,Agile,kanban

Frequent used methods of OutputStream class

Method	Description
public void write(int)throws IOException	used to write a byte to the current output stream.
public void write(byte[])throws IOException:	used to write an array of byte to the current output stream.
public void flush()throws IOException	flushes the current output stream.
public void close()throws IOException:	is used to close the current output stream.

Output Stream Hierarchy:

Written by: **Shamik Mitra**
Technical Leader at IBM
blog: http://javaonfly.blogspot.in/
facebook : https://www.facebook.com/shamik.mitra.37
mob : **9830471739** mail **:mitrashamik@gmail.com**
Provide Tuition on java, j2ee Struts,Spring,Hibernate,jpa,Hadoop,Devops,Agile,kanban



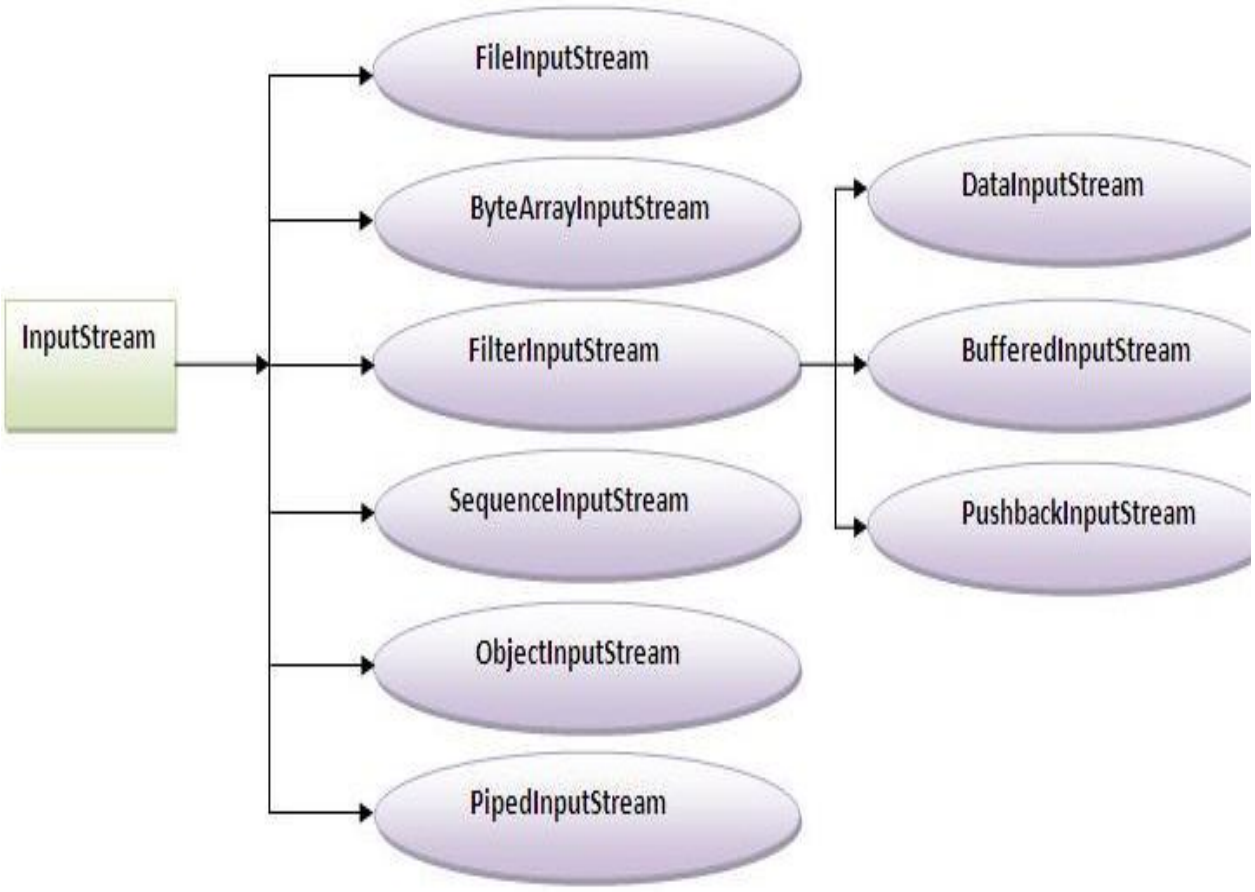
InputStream class

InputStream is an abstract class. It is the superclass of all classes representing an input stream of bytes.

Commonly used methods of InputStream class

Written by: **Shamik Mitra**
Technical Leader at IBM
blog: http://javaonfly.blogspot.in/
facebook : https://www.facebook.com/shamik.mitra.37
mob : **9830471739** mail **:mitrashamik@gmail.com**
Provide Tuition on java, j2ee Struts,Spring,Hibernate,jpa,Hadoop,Devops,Agile,kanban

Method	Description
public abstract int read()throws IOException:	reads the next byte of data from the input stream. It returns -1 at the end of file.
public int available()throws IOException:	returns an estimate of the number of bytes that can be read from the current input stream.
public void close()throws IOException:	is used to close the current input stream.



Written by: **Shamik Mitra**
Technical Leader at IBM
blog: http://javaonfly.blogspot.in/
facebook : https://www.facebook.com/shamik.mitra.37
mob : **9830471739** mail **:mitrashamik@gmail.com**
Provide Tuition on java, j2ee Struts,Spring,Hibernate,jpa,Hadoop,Devops,Agile,kanban

Example :

Write a text in a File

```
package com.example.FileandIO;

import java.io.FileOutputStream;
import java.io.IOException;

publicclass WriteContent {

	FileOutputStream fout=null;

	publicvoid write(String str) throws IOException
	{

		try
		{
			fout=new FileOutputStream("content.txt");
		byteb[]=str.getBytes();//converting string into byte array
		fout.write(b);

		    System.out.println("success...");
		}
		catch(Exception ex)
		{
			ex.printStackTrace();
		}
		finally
		{

			fout.close();

		}
	}

	publicstaticvoid main(String[] args) {

		WriteContent content = new WriteContent();
		try {
			content.write("Java is my favourite language");
```

```
                } catch (IOException e) {
                        e.printStackTrace();
                }

        }


}
```

After running it will generate a file called content.txt under javaAssignment folder.

Write a java program to read from a file

```
package com.example.FileandIO;

importjava.io.File;
import java.io.FileInputStream;
import java.io.IOException;

publicclass ReadContent {
        FileInputStream fin=null;
        publicvoid read() throws IOException
        {

                try{
                        fin=new FileInputStream("content.txt");
                        int i=0;
                        while((i=fin.read())!=-1){
                            System.out.print((char)i);
                          }

                         }catch(Exception e){
                                e.printStackTrace();
                         }
                        finally
                         {
                                fin.close();
                         }

        }

        publicstaticvoid main(String[] args) {
```

```
                ReadContent content = new ReadContent();
                try {
                        content.read();
                } catch (IOException e) {
                        e.printStackTrace();
                }

        }

}
```

Java BufferedOutputStream:

Java BufferedOutputStream class uses an internal buffer to store data. It adds more efficiency than to write data directly into a stream. So, it makes the performance fast.

Java BufferedInputStream

Java BufferedInputStream class is used to read information from stream. It internally uses buffer mechanism to make the performance fast.

Example: File Copy using java BufferdInputStream, BufferedOutputStream

```
package com.example.FileandIO;

import java.io.BufferedInputStream;
import java.io.BufferedOutputStream;
import java.io.FileInputStream;
```

Written by: **Shamik Mitra**
Technical Leader at IBM
blog: http://javaonfly.blogspot.in/
facebook : https://www.facebook.com/shamik.mitra.37
mob : **9830471739** mail **:mitrashamik@gmail.com**
Provide Tuition on java, j2ee Struts,Spring,Hibernate,jpa,Hadoop,Devops,Agile,kanban

```
import java.io.FileOutputStream;

publicclass CopyFileUsingBuferdStream {
        String body="";

        publicvoid read() throws Exception
        {
                FileInputStream fin=null;
                BufferedInputStream bin=null;

                try{

                        fin=new FileInputStream("content.txt");
                   bin=new BufferedInputStream(fin);
                int i;

                while((i=bin.read())!=-1){
                // System.out.print((char)i);
                    System.out.println(new Character((char)i).toString());
                body = body + new Character((char)i).toString();
                   }

                   System.out.println("String is " + body);

                }
                catch(Exception ex)
                {
                        ex.printStackTrace();
                }
                finally
                {
                        fin.close();
                        bin.close();
                }



        }

        publicvoid write() throws Exception
        {
                FileOutputStream fout=null;
                 BufferedOutputStream bout=null;

                try
                {
                        fout=new FileOutputStream("contentCopy.txt");
```

```
                    bout=new BufferedOutputStream(fout);
                byte[] buff = body.getBytes();
                    bout.write(buff);
                    bout.flush();
                    System.out.println("Done");

                }
                catch(Exception ex)
                {
                        ex.printStackTrace();

                }
                finally
                {
                        fout.close();
                }




        }

        publicstaticvoid main(String[] args) {

                CopyFileUsingBuferdStream stream = new CopyFileUsingBu-
ferdStream();
                try
                {
                        stream.read();
                        stream.write();

                }
                catch(Exception ex)
                {
                        ex.printStackTrace();
                }



        }

}
```

File Reader and File Writer:

Written by: **Shamik Mitra**
Technical Leader at IBM
blog: http://javaonfly.blogspot.in/
facebook : https://www.facebook.com/shamik.mitra.37
mob : **9830471739** mail **:mitrashamik@gmail.com**
Provide Tuition on java, j2ee Struts,Spring,Hibernate,jpa,Hadoop,Devops,Agile,kanban

Java FileWriter and FileReader classes are used to write and read data from text files. These are character-oriented classes, used for file handling in java.

Java has suggested not to use the FileInputStream and FileOutputStream classes if you have to read and write the textual information. File Reader and writer is higher level class so it provides some simple but powerful methods to read and write from a source

Useful methods of File Reader and Writer

FileWriter

Method	Description
public void write(String text)	writes the string into FileWriter.
public void write(char c)	writes the char into FileWriter.
public void write(char[] c)	writes char array into FileWriter.
public void close()	closes FileWriter.

FileReader

Method	Description
public int read()	returns a character in ASCII form. It returns -1 at the end of file.
	public void close()

Example: File Copy using File Reader and Writer

```
package com.example.FileandIO;

import java.io.FileReader;
import java.io.FileWriter;

publicclass FileCopyUsingFileReaderWriter {

        FileReader fr;
        FileWriter fw;
        String body="";
```

Written by: **Shamik Mitra**
Technical Leader at IBM
blog: http://javaonfly.blogspot.in/
facebook : https://www.facebook.com/shamik.mitra.37
mob : **9830471739** mail **:mitrashamik@gmail.com**
Provide Tuition on java, j2ee Struts,Spring,Hibernate,jpa,Hadoop,Devops,Agile,kanban

```
publicvoid read()throws Exception
{

        try
        {
                fr=new FileReader("content.txt");
                int i;
                while((i=fr.read())!=-1)
                body = body + new Character((char)i).toString();
                  System.out.println(body);


        }
        catch(Exception ex)
        {
                ex.printStackTrace();
        }
        finally
        {
                System.out.println("Always give close staement here");
                fr.close();
        }

}

publicvoid write() throws Exception
{

        try
        {

                fw=new FileWriter("contenCopyUsingFileWriter.txt");
                fw.write(body);


        }
        catch(Exception ex)
        {
                ex.printStackTrace();
        }
        finally
        {
                fw.close();
        }

}
```

```
        publicstaticvoid main(String[] args) {

                FileCopyUsingFileReaderWriter copy = new FileCopyUsingFileReader-
Writer();
                try {
                        copy.read();
                        copy.write();
                } catch (Exception e) {
                        e.printStackTrace();
                }



        }

}
```

Upon running above example a new File has been created.

Written by: **Shamik Mitra**
Technical Leader at IBM
blog: http://javaonfly.blogspot.in/
facebook : https://www.facebook.com/shamik.mitra.37
mob : **9830471739** mail **:mitrashamik@gmail.com**
Provide Tuition on java, j2ee Struts,Spring,Hibernate,jpa,Hadoop,Devops,Agile,kanban

Chapter 15

Java Inner Classes

In Java, just like methods, variables a class can have another class as its member. Writing a class within another is allowed in Java. The class written within is called the **nested class**, and the class that holds the **inner class** is called the **outer class.**

Syntax

```
class Outer_Demo{
   class Nested_Demo{
   }
}
```

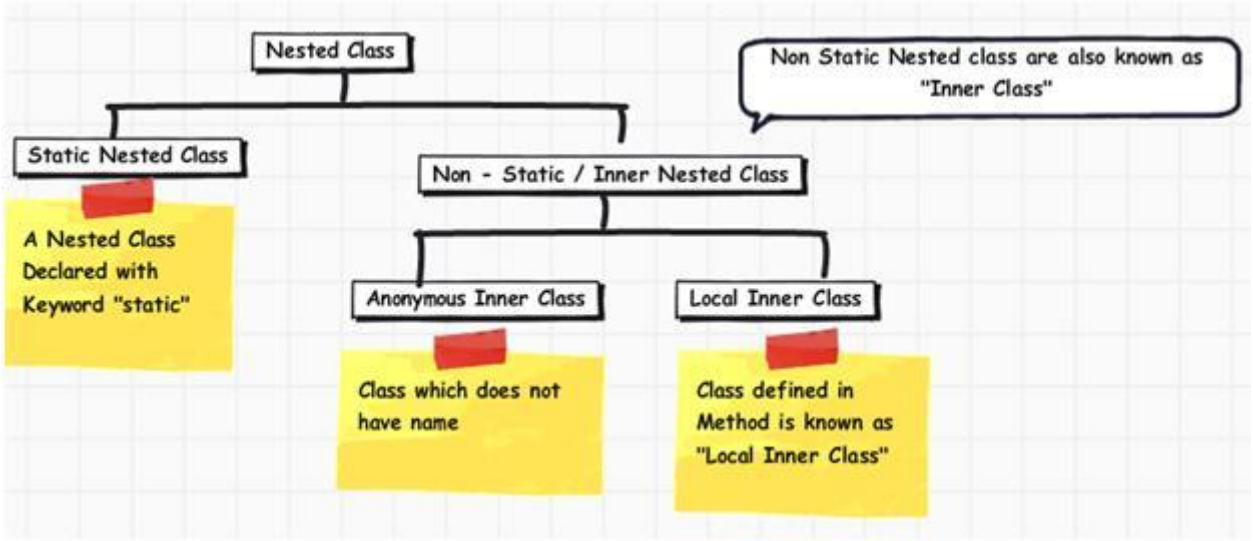
Primarily Nested class is two types

1. **Non Static Nested class**: These are the non-static members of a

class. Act as member variable.

2. **Static Nested class**: These are the static members of a class. So it is not bound to instance so act as a independent class.

Nested Class Hierarchy



Inner Classes (Non-static Nested Classes) :

Written by: **Shamik Mitra**
Technical Leader at IBM
blog: http://javaonfly.blogspot.in/
facebook : https://www.facebook.com/shamik.mitra.37
mob : **9830471739** mail **:mitrashamik@gmail.com**
Provide Tuition on java, j2ee Struts,Spring,Hibernate,jpa,Hadoop,Devops,Agile,kanban

Inner classes are a security mechanism in Java. Till now we know a class cannot be associated with the access modifier **private only public and default is permissible,** but if we have the **class** as a **member** of **other class**, then the inner class can be made **private**. And this is also used to access the private members of a class.

Three types of Inner Classes are available

1. **Inner class**
2. **Method local Inner Class**
3. **Anonymous Class**.

Inner Class:

We can call it class with in a class. So as properties and methods are membervariables of a class in same way Inner class is treat as a member variable.

So we can use inner class in any methods of the Outer class. It can be used from Outside also, Like properties of a class. It can have private or protected access modifiers.

Example:

```
package com.example.InnerClass;

publicclass Outer {

        public String msg="I am Outer class Message";
        public Inner inner;
```

```
        publicclass Inner
        {
                public String msg="I am Inner class Message";

        }

        publicvoid displyInnerClassMessage()
        {
                Inner inner = new Inner();
                System.out.println(inner.msg);


        }



        publicvoid displyOuterClassMessage()
        {
                System.out.println(this.msg);

        }


        publicstaticvoid main(String[] args) {

                Outer outer = new Outer();
                outer.displyInnerClassMessage();
                outer.displyOuterClassMessage();

        }







}


package com.example.InnerClass;

import com.example.InnerClass.Outer.Inner;

publicclass OtherClass {

        publicstaticvoid main(String[] args) {
                Outer outer = new Outer();
```

```
                Inner inner= outer.new Inner();
                System.out.println(inner.msg);
        }

}
```

Some key things to remember

In the above example a class **Inner** has been defined under **Outer So Inner** act as a member variable of Class Outer.

In displyInnerClassMessage() we access the Inner class create a instance then call it msg property because displyInnerClassMessage() is a member of Outer class so it can accesss Inner class easily.

From Other class we can invoke inner class as it is public,but as this is a member of Outerclass so without outer instance I can't access the inner class

So In OtherClass main method

First I create an Outer class object and then using that object I instantiate the Inner class by new

I know the syntax is bit weird.

Outer outer = new Outer();
Inner inner= outer.new Inner();

Remember the syntax in following way

Treat inner class as a property so access a property we use

Outer outer = new Outer.

If I call any property of Outer class we call it like outer.<somemethod>(using dot).

But I am going to access a class so to access a class we need to instantiate it
So we can do it by **new so obviously to instantiate Inner we have to do**

Written by: **Shamik Mitra**
Technical Leader at IBM
blog: http://javaonfly.blogspot.in/
facebook : https://www.facebook.com/shamik.mitra.37
mob : **9830471739** mail **:mitrashamik@gmail.com**
Provide Tuition on java, j2ee Struts,Spring,Hibernate,jpa,Hadoop,Devops,Agile,kanban

new Inner(); It is under Outer so we club two syntax dot and new and got the new syntax

```
Inner inner= outer.new Inner();
```

TIP: Remember in case of Inner class we can use private or protected modifier before class. In Inner class we can access private member of Outer class.

Method local inner class:

In Java, we can write a class within a **method** and this will be a **local type**. Like local variables, the scope of the inner class is **restricted within the method**.

A method-local inner class can be instantiated only within the method where the inner class is defined.

Example :

```
package com.example.InnerClass;

publicclass MethodLocalClass {


        publicvoid display()
        {

                final String prefix ="Hello Student This is : ";

                class DisplayMessage{

                        publicvoid showMessage(String str)
                        {
```

```
                                System.out.println(prefix + str);
                        }
                }

                 DisplayMessage msg = new DisplayMessage();
                 msg.showMessage("Method Local inner Class");


        }

        publicstaticvoid main(String[] args) {

                MethodLocalClass obj = new MethodLocalClass();
                obj.display();

        }

}
```

Note that in above example in display method we have create a new class called DisplayMessage, DisplayMMessage has a method called showMwssage() now in showMessage we access method local variable **prefix,** but always remember it has to be **final** unless compiler will complain. Because this is local variable this is in stack memory and Method local class is a class so it is in heap memory so if it is not final then it's can be changed in class but stack memory's scope is only when method executes. So if it is changed in heap memory then how it this change can be reflected in Stack. For this reason, we make it final so once declared then it can't be changed.

TIP: Method local class, scope only in that method. Even other method in same class can't access it. Note that we can only instantiate this class only in the method where it is declared. Like after declaration we create an instance of DisplayMessage class.

Anonymous Inner Class

An inner class declared without a **class name** is known as an **anonymous inner class**. In case of anonymous inner classes, we declare and instantiate them at the same time. Generally, they are used whenever you need to override the method of a class or an interface. Actually it saves our time, each time we do not have to create a

class to override it rather on the fly we declare it and override the method.

Syntax:

```
AnonymousInner an_inner = new AnonymousInner(){
  public void my_method(){
  ........
  ........
  }
};
```

Example:

```
package com.example.InnerClass;

publicabstractclass Anonymous {

	publicabstractvoid display();

	publicstaticvoid main(String[] args) {

		Anonymous obj = new Anonymous(){

			@Override
			publicvoid display()
			{
				System.out.println("Example of Anonymous class");
			}

		};

		obj.display();

	}

}
```

Notice that in main method we create an Anonymous class and override display method of abstract class Anonymous.

Written by: **Shamik Mitra**
Technical Leader at IBM
blog: http://javaonfly.blogspot.in/
facebook : https://www.facebook.com/shamik.mitra.37
mob : **9830471739** mail **:mitrashamik@gmail.com**
Provide Tuition on java, j2ee Struts,Spring,Hibernate,jpa,Hadoop,Devops,Agile,kanban

Rules of Anonymous class

Anonymous class does not **contain** any name

It cannot extend parent class rather in a same place it creates an instance of a subclass without name and override necessary methods. Like above

```
new Anonymous(){

//body
}
```

As Anonymous class does not contain any class name so it is always reference by parent reference

Anonymous class use for override so if you add a new method in Anonymous class declaration you can't call this method as it is always referenced by parent class refrence.

So

```
Anonymous obj = new Anonymous(){

                @Override
                publicvoid display()
                {
                        System.out.println("Example of Anonymous class");
                }


                publicvoidshow()
                {
                        System.out.println("show never called");
                }



        };// Always remember to put semicolon
```

So show() never ever called.

Always remember to put semicolon at the end of declaration as this is a statement.

Anonymous Inner Class as Argument

If a method accepts an object of an interface, or an abstract class, or a concrete class, then we can implement the interface or extend the abstract class, then pass the object to the method as an argument.

But in all the three cases, you can pass an **anonymous innerclass** to the method. Here is the syntax of passing an anonymous inner class as a method argument:

```
obj.method(new AnonymousClass (){
   public void innerMethod(){
   .....
   .....
   }// end of declaration of anonymous class
});// put semicolon at the end.
```

Example:

```
package com.example.InnerClass;

publicinterface Message {

        publicString msg();
```

```
}
```

```
package com.example.InnerClass;

publicclass AnonymousClassAsArgument {


	publicvoid diaplayMsg(Message obj)
	{
		System.out.println(obj.msg());
	}


	publicstaticvoid main(String[] args) {

		AnonymousClassAsArgument object = new AnonymousClassAsArgu-
ment();

		object.diaplayMsg(new Message(){


			public String msg() {

				return"I am an Anonymous Class pass as an argument";
			}

		}

		);

	}

}
```

Note that the displayMsg(Message obj) take an Object of an Interface Message
So Without create a File, in main method when we call the disPlayMsg method we create an Anonymous class of Message interface implement the msg method and pass that class as Object of Message interface.

```
object.diaplayMsg(

new Message(){

        public String msg() {

            return"I am an Anonymous Class pass as an argument";
        }

}

);//end of method call
```

Static Nested Class

A static inner class is a nested class which is a static member of the outer class. It can be accessed without instantiating the outer class, using other static members. Just like static members, a static nested class does not have access to the **instance variables** and **methods** of the outer class. The syntax of static nested class is as follows:

```
class Outer {
  static class StaticNestedClass{
  }
}
```

Example:

```
package com.example.InnerClass;

publicclass StaticNestedClass {
```

```
        publicstaticclass InnerStaticClass{

                publicvoid display()
                {
                        System.out.println("I am static Nested Class");
                }

        }

        publicstaticvoid main(String[] args) {


                InnerStaticClass obj = new StaticNestedClass.InnerStaticClass();
                obj.display();
        }
}
```

Note that in main method Without create Outer class instance I can instantiate static class. Because static variables are not part of instance.

Some Key Points

1. Static nested class can be instantiated without instance of outer class
2. As the Static nested class is Static scope or global scope so it can't be accessed by member variables that is Outer class method.
3. Static class can be accessed Static member of Outer class

Written by: **Shamik Mitra**
Technical Leader at IBM
blog: http://javaonfly.blogspot.in/
facebook : https://www.facebook.com/shamik.mitra.37
mob : **9830471739** mail **:mitrashamik@gmail.com**
Provide Tuition on java, j2ee Struts,Spring,Hibernate,jpa,Hadoop,Devops,Agile,kanban

Chapter 16

Java Serialization

Java provides a mechanism, called **object serialization** where an object can be represented as a **sequence of bytes** that includes the object's data as well as information about the object's type and the types of data stored in the object. We use Serialization to transfer the state of the Object from one source to another. Suppose I need to pass an object from One JVM to another so we export the object in a file then import them in another JVM so by serialization we can achieve it.

Once a serialized object has been written into a file or any other source, it can be read from the file or source and **deserialized** that is, the type information and bytes that represent the object and its data can be used to recreate the object in memory.

Most impressive is that the entire process is JVM independent, meaning an object can be serialized on one platform and **deserialized** on an entirely different platform.

Written by: **Shamik Mitra**
Technical Leader at IBM
blog: http://javaonfly.blogspot.in/
facebook : https://www.facebook.com/shamik.mitra.37
mob : **9830471739** mail **:mitrashamik@gmail.com**
Provide Tuition on java, j2ee Struts,Spring,Hibernate,jpa,Hadoop,Devops,Agile,kanban

ObjectInputStream and **ObjectOutputStream** are high-level classes that contain the methods for serializing and deserializing an object.

The **ObjectOutputStream** class contains many write methods for writing various data types

Most important method is

public final void writeObject(Object x) throws IOException

Similarly

ObjectInputStream class contains the following method for deserializing an object:

public final Object readObject() throws IOException, ClassNotFoundException

How serialization Works

To do serialization in java, we must implement the java.io.Serializable interface
It is a **Marker Interface that is an interface with no methods in it.**

When JVM sees an Object with Serializable interface it calls default**ObjectOutput-Stream to serialize the object.** Note that JVM do serialization for you**.**

Rules of Serialization

The class must implement the java.io.Serializable interface

All of the fields in the class must be serializable. If a field is not serializable, it must be marked **transient**.

Every property has to be serializable because If an Object you want to serialize suppose there is another Object associated with it HAS-A relation. Now when the Object is

going to serialize another object has to be serialize because Serialize means save the

state of the Object so we can restore them later. In one-word Object graph has to be serialized. So Every Object associated to main Object is must implement Serializable.

transient Keyword

In certain scenario you don't want to serialize a property suppose you want to serialize an Object but it contains another Object which is not a Serializable and it is comes from a jar, so you cannot change it by putting Serializable interface. So What will be the Solution?

Solution is either Skip that Object from Serialization process or Serialize it customly

To do first one we use transient keyword. so if a property marked with transient then it will not take part in serialization so when deserialized it's value will be the default value.

TIP: In java serializing an object to a file, the standard convention in Java is to give the file a .ser extension.

Example:

```
package com.example.serialization;

import java.io.Serializable;

publicclassStudentimplementsSerializable{

	private String name;
	private String subject;
	private String tutionDay;
	privateintfee;
	public String getName() {
		returnname;
	}
	publicvoid setName(String name) {
		this.name = name;
	}
	public String getSubject() {
		returnsubject;
	}
	publicvoid setSubject(String subject) {
		this.subject = subject;
	}
	public String getTutionDay() {
		returntutionDay;
```

```
		}
		publicvoid setTutionDay(String tutionDay) {
			this.tutionDay = tutionDay;
		}
		publicint getFee() {
			returnfee;
		}
		publicvoid setFee(int fee) {
			this.fee = fee;
		}
		@Override
		public String toString() {
			return"Student [name=" + name + ", subject=" + subject
					+ ", tutionDay=" + tutionDay + ", fee=" + fee + "]";
		}




}


package com.example.serialization;

import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;

public class Serialization {

		Student std;
		FileOutputStream fileOut;
		FileInputStream fileIn;

		public void init()
		{
			std = new Student();
			std.setName("Salman Khan");
			std.setSubject("Java");
			std.setFee(2000);
			std.setTutionDay("Sun day");
		}

		public void serialize() throws IOException
```

```
        {
                System.out.println("Before Serialization" + std);
                ObjectOutputStream out=null;

                try
                {
                        fileOut =new FileOutputStream("student.ser");
            out = new ObjectOutputStream(fileOut);
            out.writeObject(std);

            System.out.println("Serialized data is saved in student.ser");

                }
                finally
                {
                        out.close();
            fileOut.close();

                }


        }


        public void deSerialize() throws Exception
        {
                ObjectInputStream in =null;
                try
                {
                        fileIn = new FileInputStream("student.ser");
                   in = new ObjectInputStream(fileIn);
           Student std1 = (Student) in.readObject();
           System.out.println("AFTER Deserializing" + std1);

                }
                finally
                {
                        in.close();
            fileIn.close();

                }



        }

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

```
                Serialization sr = new Serialization();

                try
                {
                        sr.init();
                        sr.serialize();
                        sr.deSerialize();
                }
                catch(Exception ex)
                {
                        ex.printStackTrace();
                }

        }

}
```

Look at the above program where I create a Student Class which implement the Serializable interface so it can be serialized.

Now I serialize this Student class in student.ser file then deserialize this and restored the Student object from student.ser file.

Using ObjectInputStream and ObjectOutputStream we can serialize it and deserialize it.

Now If I want to serialize Student object without the tutionday property just I make it transient.

```
package com.example.serialization;

import java.io.Serializable;

publicclassStudentimplements Serializable{

        private String name;
        private String subject;
        transientprivate String tutionDay;
        privateintfee;
        public String getName() {
                returnname;
        }
        publicvoid setName(String name) {
                this.name = name;
```

```
        }
        public String getSubject() {
                returnsubject;
        }
        publicvoid setSubject(String subject) {
                this.subject = subject;
        }
        public String getTutionDay() {
                returntutionDay;
        }
        publicvoid setTutionDay(String tutionDay) {
                this.tutionDay = tutionDay;
        }
        publicint getFee() {
                returnfee;
        }
        publicvoid setFee(int fee) {
                this.fee = fee;
        }
        @Override
        public String toString() {
                return"Student [name=" + name + ", subject=" + subject
                                + ", tutionDay=" + tutionDay + ", fee=" + fee + "]";
        }



}
```

Now tutionDay is not part of serialization process. So when we restore Student class tutionDay will be set to default value i.e. null;

Custom Serialization Through Code:

Developers Life is not always easy Suppose you have a requirement to serialize a class but you discovered that in this class there is a HAS-A relationship with other class which is not serializable and comes from jar(So you can't modified the class) but by hook or crook you have to serialize this class after all requirement is most important things in developers life. You can do anything but you have to implement the requirement☺.

So what is the solution, If I say there is no way probably you will throw this book and show anger on me. Calm down there is a way you can customize the serialization

process or I can say you also take a part in serialization through following two methods.

private void writeObject(ObjectOutputStream out) throws IOException

private void readObject(ObjectInputStream in) throws IOException

Look closely these methods are private methods so those are not inherited or can be called from outside so JVM will call this method in the time of serialization.

Suppose in above example I want to serialize tutionDay by custom serialize process

So I rewrite the Student class in this way.

```
package com.example.serialization;

import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.io.Serializable;

publicclassStudentCustomimplements Serializable{

	private String name;
		private String subject;
		transientprivate String tutionDay;
		privateintfee;
		public String getName() {
			returnname;
		}
		publicvoid setName(String name) {
			this.name = name;
		}
		public String getSubject() {
			returnsubject;
		}
		publicvoid setSubject(String subject) {
			this.subject = subject;
		}
		public String getTutionDay() {
			returntutionDay;
		}
		publicvoid setTutionDay(String tutionDay) {
			this.tutionDay = tutionDay;
```

```
	}
	publicint getFee() {
		returnfee;
	}
	publicvoid setFee(int fee) {
		this.fee = fee;
	}




	privatevoid writeObject(ObjectOutputStream out) throws IOException {
		System.out.println("call custom write");
	  setTutionDay("MonDay");
	  out.writeObject(this.tutionDay);
	  out.defaultWriteObject();
	 }

	privatevoid readObject(ObjectInputStream in) throws IOException,
ClassNotFoundException {
		System.out.println("call custom read");
		this.tutionDay= (String)in.readObject();
	  in.defaultReadObject();
	 }




	@Override
	public String toString() {
		return"StudentCustom [name=" + name + ", subject=" + subject
				+ ", tutionDay=" + tutionDay + ", fee=" + fee + "]";
	}




}
```

Written by: **Shamik Mitra**
Technical Leader at IBM
blog: http://javaonfly.blogspot.in/
facebook : https://www.facebook.com/shamik.mitra.37
mob : **9830471739** mail **:mitrashamik@gmail.com**
Provide Tuition on java, j2ee Struts,Spring,Hibernate,jpa,Hadoop,Devops,Agile,kanban

```
package com.example.serialization;

import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;

publicclass SerializationDemo {

	StudentCustom std;
	FileOutputStream fileOut;
	FileInputStream fileIn;

	publicvoid init()
	{
		std = new StudentCustom();
		std.setName("Shahrukh Khan");
		std.setSubject("Java");
		std.setFee(2000);
		std.setTutionDay("Sun day");
	}

	publicvoid serialize() throws IOException
	{
		System.out.println("Before Serialization" + std);
		ObjectOutputStream out=null;

		try
		{
			fileOut =new FileOutputStream("studentCustom.ser");
		out = new ObjectOutputStream(fileOut);
		out.writeObject(std);

		System.out.println("Serialized data is saved in studentCustom.ser");

		}
		finally
		{
			out.close();
	fileOut.close();

		}


	}
```

```
        publicvoid deSerialize() throws Exception
        {
                ObjectInputStream in =null;
                try
                {
                        fileIn = new FileInputStream("studentCustom.ser");
                   in = new ObjectInputStream(fileIn);
                   StudentCustom std1 = (StudentCustom) in.readObject();
               System.out.println("AFTER Deserializing" + std1);

                }
                finally
                {
                         in.close();
        fileIn.close();

                }



        }

        publicstaticvoid main(String[] args) {

                SerializationDemo sr = new SerializationDemo();

                try
                {
                        sr.init();
                        sr.serialize();
                        sr.deSerialize();
                }
                catch(Exception ex)
                {
                        ex.printStackTrace();
                }

        }


}
```

Pay attention to StudentCustom Class here to do custom serialization we add two new methods writeObject and readOject. In those method first I do the custom serialization

for tutionDay . I use writeObject to explicitly serialize it then call defaultWriteObject() so JVM can do the serialization for other fields.

Externalizable interface:

There is an another way to do custom serialization. Implementing Externalizable interface. Remember that it is not a marker interface it has two methods

publicvoidreadExternal(ObjectInput in)
publicvoidwriteExternal(ObjectOutput out)

Let assume we want to Serialize Student class using externalizable interface.

Example:

```
package com.example.serialization;

import java.io.Externalizable;
import java.io.IOException;
import java.io.ObjectInput;
import java.io.ObjectOutput;

publicclass StudentExtern implements Externalizable{

	private String name;
	private String subject;
	transientprivate String tutionDay;
	privateintfee;
	public String getName() {
		returnname;
	}
	publicvoid setName(String name) {
		this.name = name;
	}
	public String getSubject() {
		returnsubject;
```

```
	}
	publicvoid setSubject(String subject) {
		this.subject = subject;
	}
	public String getTutionDay() {
		returntutionDay;
	}
	publicvoid setTutionDay(String tutionDay) {
		this.tutionDay = tutionDay;
	}
	publicint getFee() {
		returnfee;
	}
	publicvoid setFee(int fee) {
		this.fee = fee;
	}
	publicvoid readExternal(ObjectInput in) throws IOException,
			ClassNotFoundException {

		System.out.println("Call read External");
		this.name = (String)in.readObject();
		this.subject= (String)in.readObject();
		this.tutionDay = (String)in.readObject();
		this.fee = (Integer)in.readInt();
	}
	publicvoid writeExternal(ObjectOutput out) throws IOException {
		System.out.println("Call Write External");
		out.writeObject(name);
		out.writeObject(subject);
		out.writeObject(tutionDay);
		out.writeInt(fee);


	}
	@Override
	public String toString() {
		return"StudentExtern [name=" + name + ", subject=" + subject
				+ ", fee=" + fee + "]";
	}



}
```

```
package com.example.serialization;

import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;

publicclass ExternalizableDemo {

	StudentExternstd;
	FileOutputStream fileOut;
	FileInputStream fileIn;

	publicvoid init()
	{
		std = newStudentExtern();
		std.setName("Aamir Khan");
		std.setSubject("Java");
		std.setFee(2000);
		std.setTutionDay("Sun day");
	}

	publicvoid serialize() throws IOException
	{
		System.out.println("Before Serialization" + std);
		ObjectOutputStream out=null;

		try
		{
			fileOut =new FileOutputStream("studentExtern.ser");
		out = new ObjectOutputStream(fileOut);
		out.writeObject(std);

		System.out.println("Serialized data is saved in studentExtern.ser");

		}
		finally
		{
```

```
                        out.close();
        fileOut.close();

                }


        }


        publicvoid deSerialize() throws Exception
        {
                ObjectInputStream in =null;
                try
                {
                        fileIn = new FileInputStream("studentExtern.ser");
                   in = new ObjectInputStream(fileIn);
                StudentExtern std1 = (StudentExtern) in.readObject();
               System.out.println("AFTER Deserializing" + std1);

                }
                finally
                {
                        in.close();
        fileIn.close();

                }



        }

        publicstaticvoid main(String[] args) {

                ExternalizableDemo sr = new ExternalizableDemo();

                try
                {
                        sr.init();
                        sr.serialize();
                        sr.deSerialize();
                }
                catch(Exception ex)
                {
                        ex.printStackTrace();
                }

        }
```

```
}
```

Here We serialize StudenExtern class using Externalizable and in readExternal and writeExternal Method we serialize/deserializa the property of StudentExtern usinf writeObject and readObject.

serialVersionUID :

In course of serialization process always a **serialversionId** is generated which is tag to this serialize class and also saved in .ser file. later when the. ser file is about to de-serialize JVM checks the serialization id for class file and .ser file if those two matches the de-serialize done else it throws a **InvalidClassException**.

Developer can also provide **serialversionId** if not then JVM produce it by an algorithm. But **serialversionId** also exists.

Why we need serialVersionUID

Suppose you have Student class and you successfully serialize it. But After serialization a new change request come, new property holyday will be added in Student class. As a developer it is very simple you add a property holyday and provide getter setter. But the problem arises when you de-serialize Student Object from. serfile. When the Student Object Serilized at that time there was no property called holyday but now the class hasso Deserialization fails. The main reason is there was JVM generated serialVersionUID at the time of serialization but when it desrialize again that algorithm runs but this time it creates a new serialVersionUID because new property is added. So mismatch occurs and de-serialization fails.

To overcome this developer, provides a **serialVersionUID**then JVM should not

Written by: **Shamik Mitra**
Technical Leader at IBM
blog: http://javaonfly.blogspot.in/
facebook : https://www.facebook.com/shamik.mitra.37
mob : **9830471739** mail **:mitrashamik@gmail.com**
Provide Tuition on java, j2ee Struts,Spring,Hibernate,jpa,Hadoop,Devops,Agile,kanban

generate it and time of desrializeserialVersionUID is same so de-serialization process passes the test, new **property** act as a **transient** property so intialize with default

value.

Tricky Situation with SerialVersionUID

Suppose After serialization a property is deleted from a class. So when JVM de-serialize the Object from. ser file then it shows a field in. ser file but not is class at this pont de-serialization not proceed. It throws incompatible Exception.

Same with transient if a filed marked transient after serialization same problem will occur.

Please remember this situation although serialVersionUID present still it not de serialize.

Written by: **Shamik Mitra**
Technical Leader at IBM
blog: http://javaonfly.blogspot.in/
facebook : https://www.facebook.com/shamik.mitra.37
mob : **9830471739** mail **:mitrashamik@gmail.com**
Provide Tuition on java, j2ee Struts,Spring,Hibernate,jpa,Hadoop,Devops,Agile,kanban

Chapter 16

Collection Frameworks:

Till now apart from array we talked about properties which are single valued. That is It holds only one value. But in business oftenly situation occurs where you have to deal with multivalue for say find the employee who earn maxium salary. So it is comparision against employees so we need a data structute which holds eployees and then do maximum operation on that data structure.

So data Structure is key things in java not in Java for all language it is important. When we say about multivalued data structure Three thing comes with it.

1. Ordering
2. Sorting
3. Traversing.

Ordering:

Ordering means maintaining a certain sequence. So element in the collection has predefined place and it always maintains. As an Example we know Ascending order, Descending order etc.

Sorting:

Sorting is subset of ordering. Sorting means It is always orderd but maintains a special algorithm so we can say natural order is a kind of sorting.

So all Sorted thing is ordered but all ordered thing is not sorted. Like Sorting by alphabet etc.

Traversing:

When you deal with collection or multiple values you need to access each element and try to do certain operation on it. So accessing each elemen you need to Traverse through the collection. So by the method you traverse is called Traversing.

Java data structure:

Java provides following data structures.

- Enumeration
- BitSet
- Vector
- Stack
- Dictionary
- Hashtable
- Properties

All these classes are now **legacy** and Java-2 has introduced a new framework called **Collections Framework**.

Enumaration:

The Enumeration interface isn't itself a data structure, but it is very important within the context of other data structures. It uses to traverse other data structure. The Enumeration interface defines a means to retrieve successive elements from a data structure.

For example, Enumeration defines a method called **nextElement** that is used to get the next element in a data structure that contains multiple elements.

Bitset

The BitSet class implements a group of bits or flags that can be set and cleared individually.

This class is very useful in cases where you need to keep up with a set of Boolean values; you just assign a bit to each value and set or clear it as appropriate.

Vector

The Vector class is similar to a traditional Java array, except that it can grow as necessary to accommodate new elements. It is dynamic not like fixed size array.

Like an array, elements of a Vector object can be accessed via an index into the vector.

The nice thing about using the Vector class is that you don't have to worry about setting it to a specific size upon creation; it shrinks and grows automatically when necessary. Vector is synchronize so it is thread safe.

Stack

The Stack class implements a last-in-first-out (LIFO) stack of elements.

You can think of a stack literally as a vertical stack of objects; when you add a new element, it gets stacked on top of the others.

When you pull an element off the stack, it comes off the top. In other words, **the last element you added to the stack is the first one to come back off.**

Written by: **Shamik Mitra**
Technical Leader at IBM
blog: http://javaonfly.blogspot.in/
facebook : https://www.facebook.com/shamik.mitra.37
mob : **9830471739** mail **:mitrashamik@gmail.com**
Provide Tuition on java, j2ee Struts,Spring,Hibernate,jpa,Hadoop,Devops,Agile,kanban

Dictionary

The Dictionary class is an abstract class that defines a data structure for mapping keys to values.

This is useful in cases where you want to be able to access data via a particular key rather than an index.

Since the Dictionary class is abstract, it provides only the framework/contract for a key-mapped data structure rather than a specific implementation.

Hashtable:

The Hashtable class provides a means of organizing data based on some user-defined key structure.

For example, in an employee list hash table you could store and sort data based on a key such as CompanyName rather than on a employee's name.

The specific meaning of keys in regard to hash tables is totally dependent on the usage of the hash table and the data it contains. Hashtable is synchronize.

Properties:

Properties is a subclass of Hashtable. It is used to maintain lists of values in which the key is a String and the value is also a String.

The Properties class is used by many other Java classes. For example, it is the type of object returned by System.getProperties( ) when obtaining environmental values.

Written by: **Shamik Mitra**
Technical Leader at IBM
blog: http://javaonfly.blogspot.in/
facebook : https://www.facebook.com/shamik.mitra.37
mob : **9830471739** mail **:mitrashamik@gmail.com**
Provide Tuition on java, j2ee Struts,Spring,Hibernate,jpa,Hadoop,Devops,Agile,kanban

Example

```
package com.example.dataStructure;

import java.util.BitSet;
import java.util.Enumeration;
import java.util.Properties;
import java.util.Stack;
import java.util.Vector;

import com.sun.org.apache.xalan.internal.xsltc.runtime.Hashtable;

publicclass DataStructure {

	publicvoid vector()
	{
		Vector vec= newVector();
		vec.add("Shamik Mitra");
		vec.add("Samir Mitra");
		vec.add("Swastika Mitra");

		printVectorUsingEnumeration(vec);


	}

	publicvoid printVectorUsingEnumeration(Vector v)
	{
		System.out.println("Enumeration Demo ::");
		for(Enumeration e=v.elements();e.hasMoreElements(); )
		{
			String name = (String)e.nextElement();
			System.out.println("name is " + name);
		}
	}

	publicvoid bitSet()
	{
		BitSet bit =new BitSet(8);
		BitSet bit1 =new BitSet(8);
```

```
		for(int i=0;i<8;i++)
		{
			bit1.set(1);
			if(i%2==0)
			{
				bit.set(i);
			}

		}

		System.out.println("BitSet pattern is " + bit);
		System.out.println("Doing operation on BitSet");

		 bit.or(bit1);
		 System.out.println("Now BitSet pattern is " + bit);
	}


	publicvoid stack()
	{
		Stack st = newStack();
		System.out.println("Stack Push");
		st.push(1L);
		st.push(2L);
		st.push(3L);
		System.out.println("Stack Push" + st);

		System.out.println("Stack pop");

		for(int i=0;i<=2;i++)
		{
			System.out.println("POP : " + st.pop());
		}




	}

	publicvoid hastable()
	{
		Hashtable tb = new Hashtable();

		tb.put("name", "shamik Mitra");
		tb.put("Job", "IBM");
		tb.put("hobby", "photography");
		System.out.println("Hashtable " + tb);
```

Written by: **Shamik Mitra**
Technical Leader at IBM
blog: http://javaonfly.blogspot.in/
facebook : https://www.facebook.com/shamik.mitra.37
mob : **9830471739** mail **:mitrashamik@gmail.com**
Provide Tuition on java, j2ee Struts,Spring,Hibernate,jpa,Hadoop,Devops,Agile,kanban

```
                for(Enumeration e=tb.keys();e.hasMoreElements(); )
                {
                        String key = (String)e.nextElement();
                        String value = (String)tb.get(key);
                        System.out.println("Key is " + key + " value is "+ value);
                }

        }

        publicvoid properties()
        {
                Properties tb = new Properties();

                tb.put("INDIA", "DELHI");
                tb.put("BANGLADESH", "DHAKA");
                tb.put("AMERICA", "WASHINGTON DC");
                System.out.println("Hashtable " + tb);
                for(Enumeration e=tb.keys();e.hasMoreElements(); )
                {
                        String key = (String)e.nextElement();
                        String value = (String)tb.get(key);
                        System.out.println("Key is " + key + " value is "+ value);
                }

        }



        publicstaticvoid main(String[] args) {

                DataStructure dt = new DataStructure();
                dt.vector();
                dt.bitSet();
                dt.stack();
                dt.hastable();
                dt.properties();

        }



}
```

Output:

```
Enumeration Demo ::
name is Shamik Mitra
name is Samir Mitra
name is Swastika Mitra
BitSet pattern is {0, 2, 4, 6}
Doing operation on BitSet
Now BitSet pattern is {0, 1, 2, 4, 6}
Stack Push
Stack Push[1, 2, 3]
Stack pop
POP : 3
POP : 2
POP : 1
Hashtable {hobby=photography, Job=IBM, name=shamik Mitra}
Key is hobby value is photography
Key is Job value is IBM
Key is name value is shamik Mitra
Hashtable {INDIA=DELHI, AMERICA=WASHINGTON DC, BANGLADESH=DHAKA}
Key is INDIA value is DELHI
Key is AMERICA value is WASHINGTON DC
Key is BANGLADESH value is DHAKA
```

Collection Framework

Although previous data structures are still in Java but collection framework provide more easy to use and faster access data structure.

The collections framework was designed to meet several goals.

- The Collection framework is very high-performance. The implementations for the fundamental collections (dynamic arrays, linked lists, trees, and hashtables) are

Written by: **Shamik Mitra**
Technical Leader at IBM
blog: http://javaonfly.blogspot.in/
facebook : https://www.facebook.com/shamik.mitra.37
mob : **9830471739** mail **:mitrashamik@gmail.com**

- The framework had to allow different types of collections to work in a similar manner and with a high degree of interoperability.

- Extending and/or adapting a collection had to be easy.

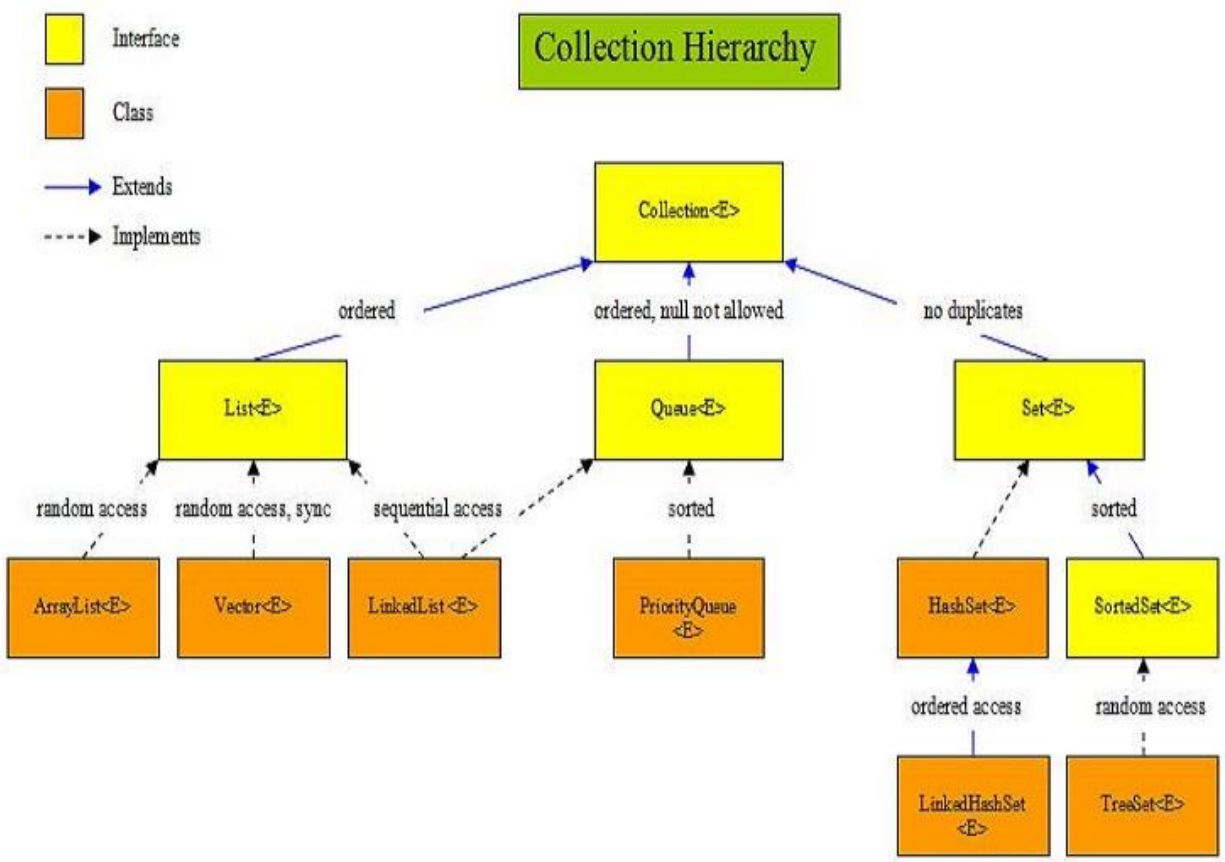
A collections framework is a unified architecture for representing and manipulating collections. All collections frameworks contain the following:

- **Interfaces:** These are abstract data types that represent collections. Interfaces allow collections to be manipulated independently of the details of their representation. In object-oriented languages, interfaces generally form a hierarchy.
- **Implementations, i.e., Classes:** These are the concrete implementations of the collection interfaces. In essence, they are reusable data structures.
- **Algorithms:** These are the methods that perform useful computations, such as searching and sorting, on objects that implement collection interfaces. The algorithms are said to be polymorphic: that is, the same method can be used on many different implementations of the appropriate collection interface.

In addition to collections, the framework defines several map interfaces and classes. Maps store key/value pairs. **Although maps are not *collections*** in the proper use of the term, but they are fully integrated with collections.

Written by: **Shamik Mitra**
Technical Leader at IBM
blog: http://javaonfly.blogspot.in/
facebook : https://www.facebook.com/shamik.mitra.37
mob : **9830471739** mail **:mitrashamik@gmail.com**
Provide Tuition on java, j2ee Struts,Spring,Hibernate,jpa,Hadoop,Devops,Agile,kanban

Collection Hirerchy



Written by: **Shamik Mitra**
Technical Leader at IBM
blog: http://javaonfly.blogspot.in/
facebook : https://www.facebook.com/shamik.mitra.37
mob : **9830471739** mail **:mitrashamik@gmail.com**
Provide Tuition on java, j2ee Struts,Spring,Hibernate,jpa,Hadoop,Devops,Agile,kanban

A closer look to hirerchy

Collection Interfaces

Name	Description
The Collection Interface	This enables you to work with groups of objects; it is at the top of the collections hierarchy.
The List Interface	This extends **Collection** and an instance of List stores an ordered collection of elements.elements are orderd but not sorted.
The Set	This extends Collection to handle sets, which must contain unique elements. Elements are unordered and unsorted.
The SortedSet	This extends Set to handle sorted sets. Elements are orderd and sorted
The Map	This maps interface stores unique keys to values. One null key accepted
The Map.Entry	This describes an element (a key/value pair) in a map. This is an inner class of Map.
The SortedMap	This extends Map so that the keys are maintained in ascending order.

Collection Classes:

Name	Description
AbstractCollection	Implements most of the Collection interface.
AbstractList	Extends AbstractCollection and implements most of the List interface.
AbstractSequentialList	Extends AbstractList for use by a collection that uses sequential rather than ran-

Written by: **Shamik Mitra**
Technical Leader at IBM
blog: http://javaonfly.blogspot.in/
facebook : https://www.facebook.com/shamik.mitra.37
mob : **9830471739** mail **:mitrashamik@gmail.com**
Provide Tuition on java, j2ee Struts,Spring,Hibernate,jpa,Hadoop,Devops,Agile,kanban

	dom access of its elements.
LinkedList	Implements a linked list by extending AbstractSequentialList.
ArrayList	Implements a dynamic array by extending AbstractList.
AbstractSet	Extends AbstractCollection and implements most of the Set interface.
HashSet	Extends AbstractSet for use with a hash table.
LinkedHashSet	Extends HashSet to allow insertion-order iterations.
TreeSet	Implements a set stored in a tree. Extends AbstractSet.
AbstractMap	Implements most of the Map interface.
HashMap	Extends AbstractMap to use a hash table.
TreeMap	Extends AbstractMap to use a tree.

Some Important Class

ArrayList:

The ArrayList class extends AbstractList and implements the List interface. ArrayList supports dynamic arrays that can grow as needed. The internal data structure of ArrayList is an array.

Standard Java arrays are of a fixed length. After arrays are created, they cannot grow or shrink, So the problem is that **you must know in advance how many elements an array will hold**. To get rid of thet we can use ArrayList it has load factor 0.75 that is when the ¾ th of size fills it grows automatically.

When objects are removed, the array may be shrunk.

ArrayList is **Orderd but not Sorted**, It maintains **Order of Insertion**which means the order elements are inserted in ArrayList in that order they are Pop out**.**

ArrayList is **RandomAccess** that means we san get access to any elment by index. ArrayList contains **duplicate** value. Can contains **null** value also.

Written by: **Shamik Mitra**
Technical Leader at IBM
blog: http://javaonfly.blogspot.in/
facebook : https://www.facebook.com/shamik.mitra.37
mob : **9830471739** mail **:mitrashamik@gmail.com**
Provide Tuition on java, j2ee Struts,Spring,Hibernate,jpa,Hadoop,Devops,Agile,kanban

ArrayList importand methods

Name	Description
void add(int index, Object element)	Inserts the specified element at the specified position index in this list. Throws IndexOutOfBoundsException if the specified index is is out of range (index < 0 || index > size()).
boolean add(Object o)	Appends the specified element to the end of this list
boolean addAll(Collection c)	Appends all of the elements in the specified collection to the end of this list, in the order that they are returned by the specified collection's iterator. Throws NullPointerException if the specified collection is null.
void clear()	Removes all of the elements from this list.
Object clone()	Returns a shallow copy of this ArrayList.
boolean contains(Object o)	Returns true if this list contains the specified element. More formally, returns true if and only if this list contains at least one element e such that (o==null ? e==null : o.equals(e)).
void ensureCapacity(int minCapacity)	ncreases the capacity of this ArrayList instance, if necessary, to ensure that it can hold at least the number of elements specified by the minimum capacity argument.
Object get(int index)	Returns the element at the specified position in this list. Throws IndexOutOfBoundsException if the specified index is is out of range (index < 0 || index >= size()).
int indexOf(Object o)	Returns the index in this list of the first occurrence of the specified element, or -1 if the List does not contain this element.
Object remove(int index)	Removes the element at the specified position in this list. Throws IndexOutOf-

Written by: **Shamik Mitra**
Technical Leader at IBM
blog: http://javaonfly.blogspot.in/
facebook : https://www.facebook.com/shamik.mitra.37
mob : **9830471739** mail **:mitrashamik@gmail.com**
Provide Tuition on java, j2ee Struts,Spring,Hibernate,jpa,Hadoop,Devops,Agile,kanban

	BoundsException if index out of range (index < 0 || index >= size()).
int lastIndexOf(Object o)	Returns the index in this list of the last occurrence of the specified element, or -1 if the list does not contain this element.
int size()	Returns the number of elements in this list.
Object[] toArray()	Returns an array containing all of the elements in this list in the correct order. Throws NullPointerException if the specified array is null.
Object[] toArray(Object[] a)	Returns an array containing all of the elements in this list in the correct order; the runtime type of the returned array is that of the specified array.
void trimToSize()	Trims the capacity of this ArrayList instance to be the list's current size.

Example:

```
package com.example.collection;

import java.util.ArrayList;
import java.util.Collection;
import java.util.List;

publicclass ArrayListDemo {

	ListarrayList=newArrayList();

	publicvoid addelement()
	{
		arrayList.add("Shamik");
		arrayList.add("Samir");
		System.out.println("After add collection" + arrayList);

	}
	publicvoid addelement(int index,String name)
	{
		arrayList.add(index, name);
		System.out.println("After add name" + arrayList);

	}
```

```
publicvoid addelement(Collection c)
{
        arrayList.addAll(c);
        System.out.println("After add collection" + arrayList);

}

publicvoid getIndex(String name)
{
        int index = arrayList.indexOf(name);
        System.out.println("Index pos:" + index);
}

publicvoid getElement(int index)
{
        String name =(String)arrayList.get(index);
        System.out.println("element : " + name);
}


publicvoid removeElement(int index)
{
        arrayList.remove(index);
        System.out.println("After Remove : " + arrayList);
}
publicvoid removeElement(String name)
{
        arrayList.remove(name);
        System.out.println("After Remove  by Object: " + arrayList);
}

publicvoid toArray()
{
        for(Object a:  arrayList.toArray())
        {
                System.out.println("Element : " + a.toString());
        }


}


publicstaticvoid main(String[] args) {
        ArrayListDemo demo =new ArrayListDemo();
        demo.addelement();
    demo.addelement(2,"Swastika");
```

Written by: **Shamik Mitra**
Technical Leader at IBM
blog: http://javaonfly.blogspot.in/
facebook : https://www.facebook.com/shamik.mitra.37
mob : **9830471739** mail **:mitrashamik@gmail.com**
Provide Tuition on java, j2ee Struts,Spring,Hibernate,jpa,Hadoop,Devops,Agile,kanban

```
        Collection c= new ArrayList();
        c.add("Mayukh");
        c.add("Ajanta");

            demo.addelement(c);

            demo.getIndex("Shamik");
            demo.getElement(3);
            demo.removeElement(3);
            demo.removeElement("Ajanta");
            demo.toArray();

        }


}
```

Output:

```
After add collection[Shamik, Samir]
After add name[Shamik, Samir, Swastika]
After add collection[Shamik, Samir, Swastika, Mayukh, Ajanta]
Index pos:0
element : Mayukh
After Remove : [Shamik, Samir, Swastika, Ajanta]
After Remove  by Object: [Shamik, Samir, Swastika]
Element : Shamik
Element : Samir
Element : Swastika
```

***TIP*: When you get and object from a list by passing the Object i.e look demo.removeElement("Ajanta"); . it internally calls equals method of the passing Object if match found then remove the Object from list . Here we use String so it calls String's equals method. But if you put your Custom Object in list then equals method must have to be Overridden to work, remove method successfully.**

Written by: **Shamik Mitra**
Technical Leader at IBM
blog: http://javaonfly.blogspot.in/
facebook : https://www.facebook.com/shamik.mitra.37
mob : **9830471739** mail **:mitrashamik@gmail.com**
Provide Tuition on java, j2ee Struts,Spring,Hibernate,jpa,Hadoop,Devops,Agile,kanban

HashSet :

HashSet extends AbstractSet and implements the Set interface. It creates a collection that uses a **hash table**data structure for storage.

A hash table stores information by using a mechanism called **hashing**. In hashing algoritmm, **the informational content of a key is used to determine a unique value**, called its **hash code**. So internally Hashset calls it's contain Objects hascode method. So it is best practice always Override hashcode when you use custom object unless wrong result will be returned.

In Set hash code of an Object is used as the index at which the Object associated with the key is stored. **The transformation of the key into its hash code is performed automatically.**

Hash set is **unordered and unsorted.** So Set does not hold order, elements are pop out from set is unpredictable.

HashSet contains unique element. How HashSet ensures the element is Unique?

Actually it internally calls Object's **equals** method when you add/remove an element from set If the Objects are **meaningfully equivalent** or Object has it equals method properly overridden. If they are **meaningfully equivalent,** then HashSet reject the Object else add the Object in to set.

TIP: Always remember When you add a custom Object in set always override equals and hashcode of the Object. Unless it does not give proper result.

HashSet important methods:

Name	Description
boolean add(Object o)	Adds the specified element to this set if it is not already present.
void clear()	Removes all of the elements from this

Written by: **Shamik Mitra**
Technical Leader at IBM
blog: http://javaonfly.blogspot.in/
facebook : https://www.facebook.com/shamik.mitra.37
mob : **9830471739** mail **:mitrashamik@gmail.com**
Provide Tuition on java, j2ee Struts,Spring,Hibernate,jpa,Hadoop,Devops,Agile,kanban

	set.
Object clone()	Returns a shallow copy of this HashSet instance: the elements themselves are not cloned.
boolean contains(Object o)	Returns true if this set contains the specified element. internally calls equals method.
boolean isEmpty()	Returns true if this set contains no elements.
Iterator iterator()	Returns an iterator over the elements in this set.
boolean remove(Object o)	Removes the specified element from this set if it is present.
int size()	Returns the number of elements in this set (its cardinality).

Example:

```
package com.example.collection;

publicclass Employee {


        private String name;
        private String job;

        public String getName() {
                returnname;
        }
        publicvoid setName(String name) {
                this.name = name;
        }
        public String getJob() {
                returnjob;
        }
        publicvoid setJob(String job) {
                this.job = job;
        }
        @Override
        public String toString() {
                return"Employee [name=" + name + ", job=" + job + "]";
        }
```

```
}


package com.example.collection;

import java.util.HashSet;
import java.util.Iterator;

publicclass HashSetDemo {

	HashSetset = newHashSet();

	publicvoid init()
	{
		Employee emp1 = new Employee();
		emp1.setName("Shamik");
		emp1.setJob("IT");

		Employee emp2 = new Employee();
		emp2.setName("Samir");
		emp2.setJob("GOVT");

		set.add(emp1);
		set.add(emp2);

		System.out.println("SET is " + set);
	}


	publicvoid addelement(Employee emp)
	{
		set.add(emp);
		System.out.println("After adding " + set);
	}

	publicvoid removeElement(Employee emp)
	{
		set.remove(emp);
		System.out.println("After removing " + set);
	}

	publicvoid travarse()
	{
```

```
                for(Iterator it=set.iterator();it.hasNext();)
                {
                        Employee emp =(Employee)it.next();
                        System.out.println("Employee " + emp.getName() + ": JOB : "+
emp.getJob());
                }

        }

        publicstaticvoid main(String[] args) {

                HashSetDemo demo = new HashSetDemo();
                demo.init();

                Employee emp1 = new Employee();
                emp1.setName("Shamik");
                emp1.setJob("IT");
                demo.addelement(emp1);// try to add duplicate element

                Employee emp2 = new Employee();
                emp2.setName("Shamik");
                emp2.setJob("IT");

                demo.removeElement(emp2);

                demo.travarse();

        }

}
```

Output :

```
SET is [Employee [name=Samir, job=GOVT], Employee [name=Shamik, job=IT]]
After adding [Employee [name=Samir, job=GOVT], Employee [name=Shamik, job=IT],
Employee [name=Shamik, job=IT]]
After removing [Employee [name=Samir, job=GOVT], Employee [name=Shamik,
job=IT], Employee [name=Shamik, job=IT]]
Employee Samir: JOB : GOVT
Employee Shamik: JOB : IT
Employee Shamik: JOB : IT
```

Pay attention to the Output although I try to add a duplicate element and want to

remove element but they are added and not removed correspondingly. Why such a behavior it is not the nature of set. Problem is I not override equals and hash-code of Employee Object. Now try to Override them and run the program again.

Updated Employee class

```
package com.example.collection;

publicclass Employee {


	private String name;
	private String job;

	public String getName() {
		returnname;
	}
	publicvoid setName(String name) {
		this.name = name;
	}

	public String getJob() {
		returnjob;
	}
	publicvoid setJob(String job) {
		this.job = job;
	}
	@Override
	public String toString() {
		return"Employee [name=" + name + ", job=" + job + "]";
	}

	@Override
	publicint hashCode() {
		finalint prime = 31;
		int result = 1;
		result = prime * result + ((job == null) ? 0 : job.hashCode());
		result = prime * result + ((name == null) ? 0 : name.hashCode());
		return result;
	}
	@Override
	publicboolean equals(Object obj) {
		if (this == obj)
			returntrue;
		if (obj == null)
			returnfalse;
		if (getClass() != obj.getClass())
```

```
                    returnfalse;
                Employee other = (Employee) obj;
                if (job == null) {
                    if (other.job != null)
                        returnfalse;
                } elseif (!job.equals(other.job))
                    returnfalse;
                if (name == null) {
                    if (other.name != null)
                        returnfalse;
                } elseif (!name.equals(other.name))
                    returnfalse;
                returntrue;
        }




}
```

Output:

```
SET is [Employee [name=Samir, job=GOVT], Employee [name=Shamik, job=IT]]
After adding [Employee [name=Samir, job=GOVT], Employee [name=Shamik, job=IT]]
After removing [Employee [name=Samir, job=GOVT]]
Employee Samir: JOB : GOVT
```

This is the behavior we expected. ☺

HashMap:

HashMap in Java works on **hashing** principle. It is a data structure which allows us to store object and retrieve it by a **key**. Hash functions are used to link key and value in HashMap. Objects are stored by calling **put(key, value)** method of HashMap and retrieved by calling **get(key)** method. When we call put method, **hashcode()** method of the key object is called so that **hash function** of the map can find a **bucket location** to store **value** object, which is actually an index of the internal array. If we dig down the structure we found HashMap internally stores mapping in the form of **Map.Entry** object which contains both key and value object. When you want to retrieve the object, you call the **get()** method and again pass the **key** object. This time again key object generate same **hash code** and we end up at same bucket location. If there is only one object then it is returned the value.

Written by: **Shamik Mitra**
Technical Leader at IBM
blog: http://javaonfly.blogspot.in/
facebook : https://www.facebook.com/shamik.mitra.37
mob : **9830471739** mail **:mitrashamik@gmail.com**
Provide Tuition on java, j2ee Struts,Spring,Hibernate,jpa,Hadoop,Devops,Agile,kanban

The Map interface maps unique keys to values. **A key is an object** that you use to retrieve a value at a later date.

- Given a key and a value, you can store the value in a Map object. After the value is stored, you can retrieve it by using its key. Keys equals and hascode must be overridden.
- Several methods throw a NoSuchElementException when no items exist in the invoking map.
- A ClassCastException is thrown when an object is incompatible with the elements in a map.
- A NullPointerException is thrown if an attempt is made to use a null object and null is not allowed in the map.
- An UnsupportedOperationException is thrown when an attempt is made to change an unmodifiable map.

Important methods of Map

Name	Description
void clear( )	Removes all key/value pairs from the invoking map.
boolean containsKey(Object k)	turns true if the invoking map contains k as a key. Otherwise, returns false. Internally call Keys equals and hascode. Hascode identify the bucket and equals check for meaningfully equivalent.
boolean containsValue(Object v)	Returns true if the map contains v as a value. Otherwise, returns false
Set entrySet( )	Returns a Set that contains the entries in the map. The set contains objects of type Map.Entry. This method provides a set-view of the invoking map.
boolean equals(Object obj)	Returns true if obj is a Map and contains the same entries. Otherwise, returns false.
Object get(Object k)	Returns the value associated with the key k.
int hashCode( )	Returns the hash code for the invoking map.
boolean isEmpty( )	Returns true if the invoking map is empty. Otherwise, returns false.

Written by: **Shamik Mitra**
Technical Leader at IBM
blog: http://javaonfly.blogspot.in/
facebook : https://www.facebook.com/shamik.mitra.37
mob : **9830471739** mail **:mitrashamik@gmail.com**
Provide Tuition on java, j2ee Struts,Spring,Hibernate,jpa,Hadoop,Devops,Agile,kanban

Set keySet( )	Returns a Set that contains the keys in the invoking map. This method provides a set-view of the keys in the invoking map.
Object put(Object k, Object v)	Puts an entry in the invoking map, overwriting any previous value associated with the key. The key and value are k and v, respectively. Returns null if the key did not already exist. Otherwise, the previous value linked to the key is returned.
void putAll(Map m)	Puts all the entries from m into this map.
Object remove(Object k)	Removes the entry whose key equals k.
int size( )	Returns the number of key/value pairs in the map.
Collection values( )	Returns a collection containing the values in the map. This method provides a collection-view of the values in the map.

Example:

```
package com.example.collection;

import java.util.HashMap;
import java.util.Iterator;
import java.util.Map;
import java.util.Map.Entry;

publicclass HashMapDemo {

	Mapmap = newHashMap();


	publicvoid init()
	{
		map.put("Shamik", "IT");
		map.put("Samir", "Govt");
		System.out.println("Map is" + map);
	}

	publicvoid addElement(String key,String value)
	{
		map.put(key, value);
```

```
                System.out.println("After add Map is" + map);
        }

        publicvoid removeElement(String key)
        {
                map.remove(key);
                System.out.println("After remove Map is" + map);
        }

        publicvoid traverseThroughEntrySet()
        {
                System.out.println("Traverse through entryset");
                for(Iterator it=map.entrySet().iterator();it.hasNext();)
                {
                        Entry obj = (Entry) it.next();
                        System.out.println("Key: " + obj.getKey()+ " value : "+
obj.getValue());

                }

        }

        publicvoid traverseThroughKeySet()
        {
                System.out.println("Traverse through keyset");
                for(Iterator it=map.keySet().iterator();it.hasNext();)
                {
                        String key = (String) it.next();
                        String value=(String) map.get(key);
                        System.out.println("Key: " + key+ " value : "+ value);

                }

        }

        publicstaticvoid main(String[] args) {

                HashMapDemo demo = new HashMapDemo();
                demo.init();
                demo.addElement("Swastika", "IT");
                demo.removeElement("Swastika");
                demo.traverseThroughEntrySet();
                demo.traverseThroughKeySet();

        }
```

```
}
```

Output :

```
Map is{Samir=Govt, Shamik=IT}
After add Map is{Samir=Govt, Shamik=IT, Swastika=IT}
After remove Map is{Samir=Govt, Shamik=IT}
Traverse through entryset
Key: Samir value : Govt
Key: Shamik value : IT
Traverse through keyset
Key: Samir value : Govt
Key: Shamik value : IT
```

Key things to remember

Few simple things need to remember, if you want to use your **custom object** as a **Key** of **Map**
1. It has to be immutable .as some one changed the state. of the object. Immutable means you can't change the state of the object. So you can't modify any property of the Object once it's value is assigned. We can acive it through final modifier.

2. It has to be override hashcode. As good your hasing algoritm, distribution of objects in bucket will be as good. Hascode identifies the bucket if hascode does not overridden you can't found the key.
.3. It has to override **equals** method so in time of get it can fetch the Object which are meaningfully equal.

Example:

```
public final class CustomKey{
private final String name = "Shamik";
public final String setName(String name)
{
this.name=name;
}
public final String getName(String name){
return name;
```

```
}
public int hascode()
{
return name.hascode()*1245/7;
}
public boolean equals(Object obj)
{
if(obj instanceof CustomKey)
{
return this.name.equals(obj.name);
}
return false;
}
}
```

Now if name is not final then following scenario can occur

CustomKey refKey= new CustomKey();
Map<CustomKey ,String> map = new HashMap<CustomKey,String>();
map.put(refKey,"Shamik");
refKey.setName("Bubun");
map.get(refKey);
It will return null as when equals method invokes it return false. So immutability is required.

Iterator

Often, you will want to traverse through the elements in a collection. For example, you might want to display each element.

The easiest way to do this is to employ an iterator, which is an object that implements either the **Iterator** or the **ListIterator** interface.

Iterator enables you to traverse through a collection, obtaining or removing elements. ListIterator extends Iterator to allow **bidirectional** traversal of a list and the modification of elements.

Java use External Iterator to iterate over collection.

Example:

Written by: **Shamik Mitra**
Technical Leader at IBM
blog: http://javaonfly.blogspot.in/
facebook : https://www.facebook.com/shamik.mitra.37
mob : **9830471739** mail **:mitrashamik@gmail.com**
Provide Tuition on java, j2ee Struts,Spring,Hibernate,jpa,Hadoop,Devops,Agile,kanban

```
package com.example.collection;

import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;

publicclass IteratorDemo {

	publicListcountryList = newArrayList();

	publicvoid init()
	{
		countryList.add("India");
		countryList.add("Pakistan");
		countryList.add("Srilanka");
		countryList.add("Nepal");

	}

	publicvoid traverse()
	{
		for(Iterator it=countryList.iterator();it.hasNext();)
		{
			String country = (String)it.next();
			System.out.println("Country name is " + country);
		}
	}

	publicstaticvoid main(String[] args) {

		IteratorDemo demo = new IteratorDemo();
		demo.init();
		demo.traverse();

	}

}
```

Output :
Country name is India
Country name is Pakistan
Country name is Srilanka
Country name is Nepal

Written by: **Shamik Mitra**
Technical Leader at IBM
blog: http://javaonfly.blogspot.in/
facebook : https://www.facebook.com/shamik.mitra.37
mob : **9830471739** mail **:mitrashamik@gmail.com**
Provide Tuition on java, j2ee Struts,Spring,Hibernate,jpa,Hadoop,Devops,Agile,kanban

Every collection object has iterator method which returns a subclass Iterator Objects.

Iterator has two important methods

hasNext() and next();

hasNext checks is there is an any element exists from current index. But it does not moves the pointer.

next() gives the next element and increase the index.

Collections Class

The **java.util.Collections** class consists utility static methods that operate on or return collections.Following are the important points about Collections:

- It contains polymorphic algorithms that operate on collections, "wrappers", which return a new **collection backed** by a specified collection.
- The methods of this class all throw a NullPointerException if the collections or class objects provided to them are null.

Collection class utility methods

Name	Description
static <T> boolean addAll(Collection<? super T> c, T... elements)	This method adds all of the specified elements to the specified collection.
static <T> Queue<T> asLifoQueue(Deque<T> deque)	This method returns a view of a Deque as a Last-in-first-out (Lifo) Queue.
static <T> int binarySearch(List<? extends Comparable<? super T>> list, T key)	This method searches the specified list for the specified object using the binary search algorithm.
static <T> int binarySearch(List<? extends T> list, T key, Comparator<? super T< c)	This method searches the specified list for the specified object using the binary search algorithm.

Written by: **Shamik Mitra**
Technical Leader at IBM
blog: http://javaonfly.blogspot.in/
facebook : https://www.facebook.com/shamik.mitra.37
mob : **9830471739** mail **:mitrashamik@gmail.com**
Provide Tuition on java, j2ee Struts,Spring,Hibernate,jpa,Hadoop,Devops,Agile,kanban

static <E> Collection<E> checkedCollection(Collection<E> c, Class<E> type)	This method returns a dynamically typesafe view of the specified collection.
static <E> List<E> checkedList(List<E> list, Class<E> type)	This method returns a dynamically typesafe view of the specified list.
static <K,V> Map<K,V> checkedMap(Map<K,V> m, Class<K> keyType, Class<V> valueType)	This method returns a dynamically typesafe view of the specified map.
static <E> Set<E> checkedSet(Set<E> s, Class<E> type)	This method returns a dynamically typesafe view of the specified set.
static <K,V> SortedMap<K,V> checkedSortedMap(SortedMap<K,V> m, Class<K> keyType, Class<V> valueType)	This method returns a dynamically typesafe view of the specified sorted map.
static <E> SortedSet<E>checkedSortedSet(SortedSet<E> s, Class<E> type)	This method returns a dynamically typesafe view of the specified sorted set.
static <T> void copy(List<? super T> dest, List<? extends T> src)	This method copies all of the elements from one list into another.
static boolean disjoint(Collection<?> c1, Collection<?> c2)	This method returns true if the two specified collections have no elements in common.
static <T> List<T> emptyList()	This method returns the empty list (immutable).
static <K,V> Map<K,V> emptyMap()	This method returns the empty map (immutable).
static <T> Set<T> emptySet()	This method returns the empty set (immutable).
static <T> Enumeration<T> enumeration(Collection<T> c)	This method returns an enumeration over the specified collection.
static <T> void fill(List<? super T> list, T obj)	This method replaces all of the elements of the specified list with the specified element.
static int frequency(Collection<?> c, Object o)	This method returns the number of elements in the specified collection equal to the specified object.
static int indexOfSubList(List<?> source, List<?> target)	This method returns the starting position of the first occurrence of the specified target list within the specified source list, or -1 if there is no such occurrence.

Written by: **Shamik Mitra**
Technical Leader at IBM
blog: http://javaonfly.blogspot.in/
facebook : https://www.facebook.com/shamik.mitra.37
mob : **9830471739** mail **:mitrashamik@gmail.com**
Provide Tuition on java, j2ee Struts,Spring,Hibernate,jpa,Hadoop,Devops,Agile,kanban

static int lastIndexOfSubList(List<?> source, List<?> target)	This method returns the starting position of the last occurrence of the specified target list within the specified source list, or -1 if there is no such occurrence.
static <T> ArrayList<T> list(Enumeration<T> e)	This method returns an array list containing the elements returned by the specified enumeration in the order they are returned by the enumeration.
static <T extends Object & Comparable<? super T>>T max(Collection<? extends T> coll)	This method returns the maximum element of the given collection, according to the natural ordering of its elements.
static <T> T max(Collection<? extends T> coll, Comparator<? super T> comp)	This method returns the maximum element of the given collection, according to the order induced by the specified comparator.
static <T extends Object & Comparable<? super T>>T min(Collection<? extends T> coll)	This method Returns the minimum element of the given collection, according to the natural ordering of its elements.
static <T> T min(Collection<? extends T> coll, Comparator<? super T> comp)	This method returns the minimum element of the given collection, according to the order induced by the specified comparator.
static <T> List<T> nCopies(int n, T o)	This method returns an immutable list consisting of n copies of the specified object.
static <E> Set<E> newSetFromMap(Map<E,Boolean> map)	This method returns a set backed by the specified map.
static <T> boolean replaceAll(List<T> list, T oldVal, T newVal)	This method replaces all occurrences of one specified value in a list with another.
static void reverse(List<?> list)	This method reverses the order of the elements in the specified list.
static <T> Comparator<T> reverseOrder()	This method returns a comparator that imposes the reverse of the natural ordering on a collection of objects that implement the Comparable interface.

Written by: **Shamik Mitra**
Technical Leader at IBM
blog: http://javaonfly.blogspot.in/
facebook : https://www.facebook.com/shamik.mitra.37
mob : **9830471739** mail **:mitrashamik@gmail.com**
Provide Tuition on java, j2ee Struts,Spring,Hibernate,jpa,Hadoop,Devops,Agile,kanban

static <T> Comparator<T> reverseOrder(Comparator<T> cmp)	This method returns a comparator that imposes the reverse ordering of the specified comparator.
static void rotate(List<?> list, int distance)	This method rotates the elements in the specified list by the specified distance.
static void shuffle(List<?> list)	This method randomly permutes the specified list using a default source of randomness.
static void shuffle(List<?> list, Random rnd)	This method randomly permute the specified list using the specified source of randomness.
static <T> Set<T> singleton(T o)	This method returns an immutable set containing only the specified object.
static <T> List<T> singletonList(T o)	This method returns an immutable list containing only the specified object.
static <K,V> Map<K,V> singletonMap(K key, V value)	This method returns an immutable map, mapping only the specified key to the specified value.
static <T extends Comparable<? super T>> void sort(List<T> list)	This method sorts the specified list into ascending order, according to the natural ordering of its elements.
static <T> void sort(List<T> list, Comparator<? super T> c)	This method sorts the specified list according to the order induced by the specified comparator.
static void swap(List<?> list, int i, int j)	This method swaps the elements at the specified positions in the specified list.
static <T> Collection<T> synchronizedCollection(Collection<T> c)	This method returns a synchronized (thread-safe) collection backed by the specified collection.
static <T> List<T> synchronizedList(List<T> list)	This method returns a synchronized (thread-safe) list backed by the specified list.
static <K,V> Map<K,V> synchronizedMap(Map<K,V> m)	This method returns a synchronized (thread-safe) map backed by the specified map.
static <T> Set<T> synchronizedSet(Set<T> s)	This method returns a synchronized (thread-safe) set backed by the specified set.

Written by: **Shamik Mitra**
Technical Leader at IBM
blog: http://javaonfly.blogspot.in/
facebook : https://www.facebook.com/shamik.mitra.37
mob : **9830471739** mail **:mitrashamik@gmail.com**
Provide Tuition on java, j2ee Struts,Spring,Hibernate,jpa,Hadoop,Devops,Agile,kanban

static <K,V> SortedMap<K,V> synchronizedSortedMap(SortedMap<K,V> m)	This method returns a synchronized (thread-safe) sorted map backed by the specified sorted map.
static <T> SortedSet<T> synchronizedSortedSet(SortedSet<T> s)	This method returns a synchronized (thread-safe) sorted set backed by the specified sorted set.
static <T> Collection<T> unmodifiableCollection(Collection<? extends T> c)	This method returns an unmodifiable view of the specified collection.
static <T> List<T> unmodifiableList(List<? extends T> list)	This method returns an unmodifiable view of the specified list.
static <K,V> Map<K,V> unmodifiableMap(Map<? extends K,? extends V> m)	This method returns an unmodifiable view of the specified map.
static <T> Set<T> unmodifiableSet(Set<? extends T> s)	This method returns an unmodifiable view of the specified set.
static <K,V> SortedMap<K,V> unmodifiableSortedMap(SortedMap<K,? extends V> m)	This method returns an unmodifiable view of the specified sorted map.
static <T> SortedSet<T> unmodifiableSortedSet(SortedSet<T> s)	This method returns an unmodifiable view of the specified sorted set.

Example :

```
package com.example.collection;

import java.util.ArrayList;
import java.util.Collection;
import java.util.Collections;
import java.util.List;

publicclass CollectionsUtilityDemo {

	Listlist;

	publicvoid addToCollection()
	{
		list = newArrayList();
```

Written by: **Shamik Mitra**
Technical Leader at IBM
blog: http://javaonfly.blogspot.in/
facebook : https://www.facebook.com/shamik.mitra.37
mob : **9830471739** mail **:mitrashamik@gmail.com**
Provide Tuition on java, j2ee Struts,Spring,Hibernate,jpa,Hadoop,Devops,Agile,kanban

```
                list.add("Shamik");
                list.add("Samir");

                System.out.println("List is " + list);
                Collections.addAll(list, "Swastika","Mayukh");
                System.out.println("Now after adding List is " + list);

        }

        publicvoid binarySearch()
        {
                list = newArrayList();
                list.add("Shamik");
                list.add("Samir");

                int index= Collections.binarySearch(list, "Shamik");
                System.out.println("Index of Search element is " + index);
        index= Collections.binarySearch(list, "Swastika");
                System.out.println("Index of Search element is " + index);      //print the
    insertable position if the element not in the collection

        }

        publicvoid dynamicTypeSafe()
        {
                list = newArrayList();
                list.add("Shamik");
                list.add("Samir");

                Collection<String> tslst= Collections.checkedList(list, String.class);
                System.out.println(tslst);
        }

        publicvoid copyList()
        {
                list = newArrayList(2);
                list.add("Shamik");
                list.add("Samir");

                List destList = newArrayList(2);
                destList.add("Swastika");
                destList.add("Mayukh");


                System.out.println("Source list before copy" + list);
                System.out.println("Destination list before copy" + destList);
                Collections.copy(destList, list);
```

```
		System.out.println("Source list after copy" + list);
		System.out.println("Destination list after copy" + destList);



	}

	publicvoid checkCommonelement()
	{
		list = newArrayList(2);
		list.add("Shamik");
		list.add("Samir");

		List destList = newArrayList(2);
		destList.add("Swastika");
		destList.add("Mayukh");

		boolean uncommon = Collections.disjoint(list, destList);
		System.out.println("List are uncommon " + uncommon);

	}

	publicvoid findFrequency()
	{
		list = newArrayList();
		list.add("Shamik");
		list.add("Samir");
		list.add("Samir");

		int freq= Collections.frequency(list, "Samir");
		System.out.println("Frq of element is " + freq);


	}

	publicvoid findMinMax()
	{
		list = newArrayList();
		list.add(5);
		list.add(15);
		list.add(55);
		list.add(123);
		list.add(-6);

		int min = Collections.min(list);
		int max=  Collections.max(list);
```

```
                System.out.println("Min Value is: "+ min + " -- Max value is: "+ max);
        }

        publicvoid reverseList()
        {
                list = newArrayList();
                list.add(5);
                list.add(15);
                list.add(55);
                list.add(123);
                list.add(-6);

                System.out.println("Before reverse :" + list);
                Collections.reverse(list);
                System.out.println("After reverse :" + list);


        }

        publicvoid rotateList()
        {
                list = newArrayList();
                list.add(5);
                list.add(15);
                list.add(55);
                list.add(123);
                list.add(-6);
                System.out.println("Before rotate :" + list);
                Collections.rotate(list,3);
                System.out.println("After rotate :" + list);
        }

        publicvoid shuffleList()
        {
                list = newArrayList();
                list.add(5);
                list.add(15);
                list.add(55);
                list.add(123);
                list.add(-6);
                System.out.println("Before shuffle :" + list);
                Collections.shuffle(list);
                System.out.println("After shuffle :" + list);


        }
```

Written by: **Shamik Mitra**
Technical Leader at IBM
blog: http://javaonfly.blogspot.in/
facebook : https://www.facebook.com/shamik.mitra.37
mob : **9830471739** mail **:mitrashamik@gmail.com**
Provide Tuition on java, j2ee Struts,Spring,Hibernate,jpa,Hadoop,Devops,Agile,kanban

```
	publicvoid sortList()
	{
		list = newArrayList();
		list.add(5);
		list.add(15);
		list.add(55);
		list.add(123);
		list.add(-6);
		System.out.println("Before sorting :" + list);
		Collections.sort(list);
		System.out.println("After sorting :" + list);


	}

	publicvoid swapPostion()
	{
		list = newArrayList();
		list.add(5);
		list.add(15);
		list.add(55);
		list.add(123);
		list.add(-6);
		System.out.println("Before Swap :" + list);
		Collections.swap(list,0,3);
		System.out.println("After swap :" + list);

	}


	publicstaticvoid main(String[] args) {

		CollectionsUtilityDemo demo = new CollectionsUtilityDemo();
		demo.addToCollection();
		demo.binarySearch();
		demo.dynamicTypeSafe();
		demo.copyList();
		demo.checkCommonelement();
		demo.findFrequency();
		demo.findMinMax();
		demo.reverseList();
		demo.rotateList();
		demo.shuffleList();
		demo.sortList();
		demo.swapPostion();
	}
```

```
}
```

Output:

```
List is [Shamik, Samir]
Now after adding List is [Shamik, Samir, Swastika, Mayukh]
Index of Search element is 0
Index of Search element is -3
[Shamik, Samir]
Source list before copy[Shamik, Samir]
Destination list before copy[Swastika, Mayukh]
Source list after copy[Shamik, Samir]
Destination list after copy[Shamik, Samir]
List are uncommon true
Frq of element is 2
Min Value is: -6 -- Max value is: 123
Before reverse :[5, 15, 55, 123, -6]
After reverse :[-6, 123, 55, 15, 5]
Before rotate :[5, 15, 55, 123, -6]
After rotate :[55, 123, -6, 5, 15]
Before shuffle :[5, 15, 55, 123, -6]
After shuffle :[5, 55, 123, -6, 15]
Before sorting :[5, 15, 55, 123, -6]
After sorting :[-6, 5, 15, 55, 123]
Before Swap :[5, 15, 55, 123, -6]
After swap :[123, 15, 55, 5, -6]
```

Comparator and Comparable

As of now I use Collection.sort method onString or wrapper class and it works fine but million-dollar question is If I want to use it on my Custom class say **Employee** should it would be sorted let see.

```
package com.example.collection;

publicclassEmployee {

        private String name;
        private String job;
```

```
	public String getName() {
		returnname;
	}
	publicvoid setName(String name) {
		this.name = name;
	}

	public String getJob() {
		returnjob;
	}
	publicvoid setJob(String job) {
		this.job = job;
	}
	@Override
	public String toString() {
		return"Employee [name=" + name + ", job=" + job + "]";
	}

	@Override
	publicint hashCode() {
		finalint prime = 31;
		int result = 1;
		result = prime * result + ((job == null) ? 0 : job.hashCode());
		result = prime * result + ((name == null) ? 0 : name.hashCode());
		return result;
	}
	@Override
	publicboolean equals(Object obj) {
		if (this == obj)
			returntrue;
		if (obj == null)
			returnfalse;
		if (getClass() != obj.getClass())
			returnfalse;
		Employee other = (Employee) obj;
		if (job == null) {
			if (other.job != null)
				returnfalse;
		} elseif (!job.equals(other.job))
			returnfalse;
		if (name == null) {
			if (other.name != null)
				returnfalse;
		} elseif (!name.equals(other.name))
			returnfalse;
		returntrue;
	}
```

```
}
```

```
package com.example.collection;

import java.util.ArrayList;
import java.util.Collections;
import java.util.List;

publicclass EmployeeSortManager {

        publicstaticvoid main(String[] args) {

                List empList = newArrayList();
                Employee emp1  = new Employee();
                emp1.setJob("IT");
                emp1.setName("Shamik");

                Employee emp2  = new Employee();
                emp2.setJob("GOVT");
                emp2.setName("Samir");

                Employee emp3  = new Employee();
                emp3.setJob("IT");
                emp3.setName("Swastika");

                empList.add(emp1);
                empList.add(emp2);
                empList.add(emp3);

                System.out.println("Before Sorting" + empList);

                Collections.sort(empList);

                System.out.println("After Sorting" + empList);


        }

}
```

Written by: **Shamik Mitra**
Technical Leader at IBM
blog: http://javaonfly.blogspot.in/
facebook : https://www.facebook.com/shamik.mitra.37
mob : **9830471739** mail **:mitrashamik@gmail.com**
Provide Tuition on java, j2ee Struts,Spring,Hibernate,jpa,Hadoop,Devops,Agile,kanban

Output:

```
Exception in thread "main" Before Sorting[Employee [name=Shamik, job=IT], Employee
[name=Samir, job=GOVT], Employee [name=Swastika, job=IT]]
java.lang.ClassCastException: com.example.collection.Employee cannot be cast to ja-
va.lang.Comparable
        at java.util.Arrays.mergeSort(Unknown Source)
        at java.util.Arrays.sort(Unknown Source)
        at java.util.Collections.sort(Unknown Source)
        at
com.example.collection.EmployeeSortManager.main(EmployeeSortManager.java:30)
```

OOPS… What happens??? Collection.sort method does not able to sort the Employee class strange isn't it?

Please pay attention to the exception message it says java.lang.ClassCastException: com.example.collection.Employee cannot be cast to java.lang.Comparable. that means sort method wants to cast the Employee Objects as Comparable interface.

Comparable interface is use for compare two Object of same class. It has a method call **publicint** compareTo(Object o) which is used to compare an Object with this Objects

1 = means this is grater than Other Object
0= means this is equals with Other Object
-1= means this is equals with Other Object

As Employee Object does not implements Comparable interface so When we invoke Collection.sort it trys to call compareTo method internally but not found so it throws the error but in case of String and Wraaper class Comprarable has been implemented already so Collection. sort method works perfectly

No I change the Employe class as follows.

```
package com.example.collection;

publicclass Employee implementsComparable{


        private String name;
        private String job;
```

Written by: **Shamik Mitra**
Technical Leader at IBM
blog: http://javaonfly.blogspot.in/
facebook : https://www.facebook.com/shamik.mitra.37
mob : **9830471739** mail **:mitrashamik@gmail.com**


```
	public String getName() {
		returnname;
	}
	publicvoid setName(String name) {
		this.name = name;
	}

	public String getJob() {
		returnjob;
	}
	publicvoid setJob(String job) {
		this.job = job;
	}
	@Override
	public String toString() {
		return"Employee [name=" + name + ", job=" + job + "]";
	}

	@Override
	publicint hashCode() {
		finalint prime = 31;
		int result = 1;
		result = prime * result + ((job == null) ? 0 : job.hashCode());
		result = prime * result + ((name == null) ? 0 : name.hashCode());
		return result;
	}
	@Override
	publicboolean equals(Object obj) {
		if (this == obj)
			returntrue;
		if (obj == null)
			returnfalse;
		if (getClass() != obj.getClass())
			returnfalse;
		Employee other = (Employee) obj;
		if (job == null) {
			if (other.job != null)
				returnfalse;
		} elseif (!job.equals(other.job))
			returnfalse;
		if (name == null) {
			if (other.name != null)
				returnfalse;
		} elseif (!name.equals(other.name))
			returnfalse;
		returntrue;
	}
```

```
        publicint compareTo(Object arg0) {

                Employee other= (Employee) arg0;

                returnthis.name.compareTo(other.name);

        }

}
```

Now if I run EmployeeSortManager It will provode following outcome.

Output :

Before Sorting[Employee [name=Shamik, job=IT], Employee [name=Samir, job=GOVT], Employee [name=Swastika, job=IT]]
After Sorting[Employee [name=Samir, job=GOVT], Employee [name=Shamik, job=IT], Employee [name=Swastika, job=IT]]

Note that in employee class we implements Comparable interface and provide an definition of compareTo method.

Now a new Question is arised if the Employee object is coming from a jar or that is implemented by a developer of different company. And you don't have the permission to modify the Source code. But you have to sort **Employee** object which does not implement comparableinterface. What do you do? Is there is any way out? Java has an answer of that situation that is **Comparator** by comprator you can sort any custom class in any order.

Think ,by Comparable only one way you can sort Objects as **compareTo has only one concreate implementation** where as **Comparator** does not require comparable. And from Outside you can provide Comparator incollection. sort method so in any order any way you can sort any Object.

See the Example below

```
package com.example.collection;
```

Written by: **Shamik Mitra**
Technical Leader at IBM
blog: http://javaonfly.blogspot.in/
facebook : https://www.facebook.com/shamik.mitra.37
mob : **9830471739** mail **:mitrashamik@gmail.com**


```
publicclass Employee implementsComparable{


        private String name;
        private String job;

        public String getName() {
                returnname;
        }
        publicvoid setName(String name) {
                this.name = name;
        }

        public String getJob() {
                returnjob;
        }
        publicvoid setJob(String job) {
                this.job = job;
        }
        @Override
        public String toString() {
                return"Employee [name=" + name + ", job=" + job + "]";
        }

        @Override
        publicint hashCode() {
                finalint prime = 31;
                int result = 1;
                result = prime * result + ((job == null) ? 0 : job.hashCode());
                result = prime * result + ((name == null) ? 0 : name.hashCode());
                return result;
        }
        @Override
        publicboolean equals(Object obj) {
                if (this == obj)
                        returntrue;
                if (obj == null)
                        returnfalse;
                if (getClass() != obj.getClass())
                        returnfalse;
                Employee other = (Employee) obj;
                if (job == null) {
                        if (other.job != null)
                                returnfalse;
                } elseif (!job.equals(other.job))
                        returnfalse;
                if (name == null) {
```

```
                        if (other.name != null)
                                returnfalse;
                } elseif (!name.equals(other.name))
                        returnfalse;
                returntrue;
        }
        publicint compareTo(Object arg0) {

                Employee other= (Employee) arg0;

                returnthis.name.compareTo(other.name);

        }









}

package com.example.collection;

import java.util.Comparator;

publicclassEmployeeReverseComparatorimplementsComparator{


        publicint compare(Object o1, Object o2) {
                Employee arg0 = (Employee)o1;
                Employee arg1 = (Employee)o2;

                return arg1.getName().compareTo(arg0.getName());
        }

}


package com.example.collection;

import java.util.Comparator;

publicclassEmployeeJobComparatorimplementsComparator{

        publicint compare(Object o1, Object o2) {
                Employee arg0 = (Employee)o1;
```

```
                Employee arg1 = (Employee)o2;
                return arg0.getJob().compareTo(arg1.getJob());
        }

}
```

```
package com.example.collection;

import java.util.ArrayList;
import java.util.Collections;
import java.util.List;

publicclass EmployeeSortUsingComparator {

        publicstaticvoid main(String[] args) {

                List empList = newArrayList();
                Employee emp1  = new Employee();
                emp1.setJob("IT");
                emp1.setName("Shamik");

                Employee emp2  = new Employee();
                emp2.setJob("GOVT");
                emp2.setName("Samir");

                Employee emp3  = new Employee();
                emp3.setJob("IT");
                emp3.setName("Swastika");

                empList.add(emp1);
                empList.add(emp2);
                empList.add(emp3);

                System.out.println("Before Sorting" + empList);

                Collections.sort(empList,new EmployeeReverseComparator());

                System.out.println("After reverse Sorting" + empList);

                Collections.sort(empList,new EmployeeJobComparator());

                System.out.println("After Sorting by job" + empList);
```

Written by: **Shamik Mitra**
Technical Leader at IBM
blog: http://javaonfly.blogspot.in/
facebook : https://www.facebook.com/shamik.mitra.37
mob : **9830471739** mail **:mitrashamik@gmail.com**
Provide Tuition on java, j2ee Struts,Spring,Hibernate,jpa,Hadoop,Devops,Agile,kanban

```
        }

}
```

Output:

```
Before Sorting[Employee [name=Shamik, job=IT], Employee [name=Samir, job=GOVT], Employee [name=Swastika, job=IT]]
After reverse Sorting[Employee [name=Swastika, job=IT], Employee [name=Shamik, job=IT], Employee [name=Samir, job=GOVT]]
After Sorting by job[Employee [name=Samir, job=GOVT], Employee [name=Swastika, job=IT], Employee [name=Shamik, job=IT]]
```

Look here I have created two comparatorsEmployeeJobComparator and EmployeeReverseComparator

When I sort the Employee I pass these comparator in collection.sort method Collections.*sort*(empList,**new** EmployeeReverseComparator()); Using these comparators I can sort the employee list in different way.

Pay attention to the comparator class it has one method call **compare** which takes two arguments of the class it wants to sort.

Written by: **Shamik Mitra**
Technical Leader at IBM
blog: http://javaonfly.blogspot.in/
facebook : https://www.facebook.com/shamik.mitra.37
mob : **9830471739** mail **:mitrashamik@gmail.com**
Provide Tuition on java, j2ee Struts,Spring,Hibernate,jpa,Hadoop,Devops,Agile,kanban

Chapter 17

Generics:

Still now in collection we add Objects in to Arraylist, set, map(collections) and when we iterate it we type cas the Object to it's actual type and then print's iis content. But this is very risky as collection takes Object and Every Custom Object extentds Objects so we can put anything in collection but when we iterate them and type cast them it will throw type cast Exception.

Example

```
package com.example.generics;

import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;

publicclass ArrayListTest {


        publicstaticvoid main(String[] args) {

                List list = newArrayList();
                list.add("Cat");
                list.add("Dog");
                list.add(new Integer(10));

                for(Iterator it=list.iterator();it.hasNext();)
                {
```

Written by: **Shamik Mitra**
Technical Leader at IBM
blog: http://javaonfly.blogspot.in/
facebook : https://www.facebook.com/shamik.mitra.37
mob : **9830471739** mail **:mitrashamik@gmail.com**
Provide Tuition on java, j2ee Struts,Spring,Hibernate,jpa,Hadoop,Devops,Agile,kanban

```
                        String obj = (String)it.next();
                        System.out.println(obj);
                }


        }

}
```

Output:

```
Cat
Dog
Exception in thread "main" java.lang.ClassCastException: java.lang.Integer cannot be cast to java.lang.String
        at com.example.generics.ArrayListTest.main(ArrayListTest.java:19)
```

Look here I explicitly add an integer in String arrayList so when we traverse it's got class cast Exception.

It would be nice if we can tell explicitly This Collection Only takes String type not Integer then we can get rid of class cast exception by generics we can achive that.

Java **Generic** methods and generic classes enable programmers to specify, with a single method declaration, a set of related methods or, with a single class declaration, a set of related types, respectively.

Generics also provide compile-time type safety that allows programmers to catch invalid types at compile time.

Using Java Generic concept, we might write a generic method for sorting an array of objects, then invoke the generic method with Integer arrays, Double arrays, String arrays and so on, to sort the array elements.

Generic Rules:

Written by: **Shamik Mitra**
Technical Leader at IBM
blog: http://javaonfly.blogspot.in/
facebook : https://www.facebook.com/shamik.mitra.37
mob : **9830471739** mail **:mitrashamik@gmail.com**
Provide Tuition on java, j2ee Struts,Spring,Hibernate,jpa,Hadoop,Devops,Agile,kanban

All generic method declarations have a type parameter section delimited by angle brackets (< and >) that precedes the method's return type (< E > in the next example).

Each type parameter section contains one or more type parameters separated by commas. A type parameter, also known as a type variable, is an identifier that specifies a generic type name.

placeholders for the types of the arguments passed to the generic method, which are known as actual type arguments.

A generic method's body is declared like that of any other method. Note that type parameters can represent only reference types, not primitive types (like int, double and char).

A generic type is only compile time safety so only compile time it exist but runtime there is no existence of generic. So we call it **type erasure.**

Example:

```
package com.example.generics;

import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;

publicclass ArrayListUsingGenerics {
	publicstaticvoid main(String[] args) {

		List<String> list = new ArrayList<String>();
		list.add("Cat");
		list.add("Dog");
		//list.add(new Integer(10));
```

Written by: **Shamik Mitra**
Technical Leader at IBM
blog: http://javaonfly.blogspot.in/
facebook : https://www.facebook.com/shamik.mitra.37
mob : **9830471739** mail **:mitrashamik@gmail.com**
Provide Tuition on java, j2ee Struts,Spring,Hibernate,jpa,Hadoop,Devops,Agile,kanban

```
                for(Iterator<String> it=list.iterator();it.hasNext();)
                {
                        String obj = it.next();
                        System.out.println(obj);
                }


        }

}
```

Look at the Example we intilize the arraylist as
List<String> list = **new** ArrayList<String>();

The angle bracket means generics in angle bracket we use a datatype String that means we can Only pass String Object to this arrayList not an Integer or ant Datatype. Even we can't pass any subclass.

If we want to pass integer compiler will complain that you can't pass Intger as it is a String type ArrayList. So we call it compile time safety. Later when we iterate over this array list we don't have to bother about the elements in the List as we know this will only contain String.

TIP: NOTE that when we use a datatype in angel bracket we can only pass that datatype or subtype element in to that collection. so If I have a super Class Vehicle and a subclass Car if we declare List<Vehicle> then we pass Car,Vehicle object in to it we can only pass vehicle type Object.

Example:

```
package com.example.generics;

publicclass Vehicle {

        private String name;

        public String getName() {
                returnname;
        }

        publicvoid setName(String name) {
                this.name = name;
        }
```

```
}

package com.example.generics;

publicclass Car extends Vehicle{

	private String brand;

	public String getBrand() {
		returnbrand;
	}

	publicvoid setBrand(String brand) {
		this.brand = brand;
	}

}



package com.example.generics;

import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;

publicclass GenericTest {

	publicstaticvoid main(String[] args) {

		List<Vehicle>list = new ArrayList<Vehicle>();

		Vehicle v1 = new Vehicle();
		v1.setName("Rickshaw");

		Car c= new Car();
		c.setName("car");
		c.setBrand("BMW");

		list.add(v1);
		list.add(c);

		for(Iterator<Vehicle> it=list.iterator();it.hasNext();)
		{
			Vehicle obj = it.next();
			System.out.println(obj.getName());
		}
```

```
        }

}
```

So, it is a good news that we can add subtype object in Generics. But can I pass Sub-type List in to method.

Example

```
package com.example.generics;

import java.util.ArrayList;
import java.util.List;

publicclass GenericsMethod {



        publicvoid testSubtype(List<Vehicle> list)
        {
                System.out.println(list);
        }


        publicstaticvoid main(String[] args) {
                List<Car> carList = new ArrayList<Car>();
                Car car = new Car();
                car.setBrand("Audi");
                car.setName("Four Wheeler");

                carList.add(car);

                GenericsMethod test = new GenericsMethod();
                test.testSubtype(carList);// We can't pass Subtype generics to SuperType
Generic argument

        }

}
```

Oops We can't pass Subtype GenericList as the method argument which takes super-type Generics.

TIP: Please burn this point in your head that we can only pass exact type into the method argument, so we can pass List<Vehicle> in to testSubtype(List<Vehicle> list) but not List<Car> although it is a subclass. This is because Compiler sees the Method argument as a List of Vehicle data type so it not allows any other da-tatype even Car which is Subtype of Vehicle.

Same way can you say is following statement is applicable or not

List<Vehicle> list = new ArrayList<Car>();

Think about it……

Yes, you guess it right this is also not permissible for same reason as I Say List of ve-hicle only takes Vehicle not any subtype. But one moment please in case of array this is supported isn't it?

Vechile [] arr = new Car[6]();

Also if a method is test(Vehicle[] arr) I can call it by test(carArr) where

carArr is a Car[] array.

So the question arise why in case of generics polymorphic assignment is not al-lowed?

To answer this question, we need to discuss it in details

As you rememberd I told that Generic is type erasure that is compile time safety that means in runtime type does not matter. It is same as old collection. And here lie the tricks

For a moment let say I can pass List<Car> in to a method test(List<Vehicle> list)

So as by generics definition List<Car> means it will have allowed only car Objectt but

as we successfully pass it in List<Vehicle> which is super type of Car I also add Train Object into the List as Train extends Vehicle.

So In a Car Collection I can add Train object. So runtime error will occur same case can happen for array but array has ArrayStored Exception by this you can identify what goes wrong but in collection as type only exists in compile type there is no point to identify the problem due to that Generic not allow subtype to pass as an argument but

you can add subtype if you pass List<Vehicle> collection in method *test(List<Vehicle> list)* that is in method test you can add *list.add (new Train());* Because you are sure that whatever you add that is subtype of Vehicle.

But Still a big question left to answer, is it not possible pass Subtype List argument in to Parent Type list parameter?

If the answer is no I am sure you will be frustrated and say what the heck I can't avail Polymorhic behavious which is main piller of OOPS. Sun developer are also think about that but the problem is If I pass child type collection in to parent type then I can add any subclass which is not intended. So what is the remedy think…..

Can you guess what I am going to tell?

No more suspense. We can pass child type if and only if in that method developer won't add anything this method only readonly no write operation is permitted. In that case I can pass child type as I am sure that no one manipulates my collection it is only for display purpose. How to achive that

Syntac

List<? extends type>

So according to the above example

If I write

test(List<? extends Vehicle> list) then I can pass list<Car> in it. As I am sure by telling this **I mentiond that I can pass any Object which extends Vehicle and I assure that I won't add anything in to the collection in the test method. I can read the collection but not doing any write operation on it**

Let's take an example where I try to add an element in to collection

Written by: **Shamik Mitra**
Technical Leader at IBM
blog: http://javaonfly.blogspot.in/
facebook : https://www.facebook.com/shamik.mitra.37
mob : **9830471739** mail **:mitrashamik@gmail.com**
Provide Tuition on java, j2ee Struts,Spring,Hibernate,jpa,Hadoop,Devops,Agile,kanban

It will provide foolwing exception

```
package com.example.generics;

import java.util.ArrayList;
import java.util.List;
```

```
package com.example.generics;

import java.util.ArrayList;
import java.util.List;

publicclass GenericsMethodTestUpdated {

	publicvoid testSubtype(List<? extendsVehicle> list)
	{
		list.add(new Integer(2));
		System.out.println("List is" + list);
	}


	publicstaticvoid main(String[] args) {
		List<Car> carList = new ArrayList<Car>();
		Car car = new Car();
		car.setBrand("Audi");
		car.setName("Four Wheeler");

		carList.add(car);

		GenericsMethodTestUpdated test = new GenericsMethodTestUpdated();
		test.testSubtype(carList);

	}


}
```

The method add(capture#1-of ? extends Vehicle) in the type List<capture#1-of ? extends Vehicle> is not applicable for the arguments (Integer)

That means You can't add anything in to collection in the method testSubtype()
If I comment the line, list.add(**new** Integer(2)); it will run perfectly answer will be

Output: List is [Car [brand=Audi]]

Written by: **Shamik Mitra**
Technical Leader at IBM
blog: http://javaonfly.blogspot.in/
facebook : https://www.facebook.com/shamik.mitra.37
mob : **9830471739** mail **:mitrashamik@gmail.com**
Provide Tuition on java, j2ee Struts,Spring,Hibernate,jpa,Hadoop,Devops,Agile,kanban

TIP: Please note that generic syntax List<? extends Object> means it will accept any Object which is subtype of Object but if it is interface still we use extends here
Like List<? implements serializable> is wrong but List<? extends Serializable> is right.
Please memorize that thing.

Be patient still something left to discuss

Can you say

List<Object> list
List<? extends Object> list
List<?> list

Is these three are same?

Think before give an answer.

Please give me the opportunity to give the answer

List<Object > and List<? extends Object> has a huge diffrenece, first one means it only take the object of type Object not any subtype of Object but the second one means it can take an Object which is subtype of Object and we can manipulate the list.
Where as the Third one is wildcard notion List<?> list it has same meaning as List<? extends Object>

Still the question remains, I can pass subtype now but can I manipulate the collection that is can I write in to collection in the method.

Also that is possible in java by using a tricks

List<? super Car> list

By this syntax we are telling that You can pass a collection which is type of Car or super type of Car nothing lower in the inheritance tree and you can manipulate the list by adding Car .

Car IS-A Vehicle
Vehicle IS_A Vehicle

So I am sure if I pass collections of parentsthen adding car is perfect .

Written by: **Shamik Mitra**
Technical Leader at IBM
blog: http://javaonfly.blogspot.in/
facebook : https://www.facebook.com/shamik.mitra.37
mob : **9830471739** mail **:mitrashamik@gmail.com**
Provide Tuition on java, j2ee Struts,Spring,Hibernate,jpa,Hadoop,Devops,Agile,kanban

Example

```
package com.example.generics;

import java.util.ArrayList;
import java.util.List;

publicclass GenericsTestWithSuper {

	publicvoid testSubtype(List<? super Car> list)
	{
		Car v = new Car();
		v.setName("Four Wheeler");
		v.setBrand("BMW");


		list.add(v);
		for(Object c : list)
		{
			Car car= (Car)c;

			System.out.println("Element is " + car);
		}

	}


	publicstaticvoid main(String[] args) {
		List<Vehicle> carList = new ArrayList<Vehicle>();
		Car car = new Car();
		car.setBrand("Audi");
		car.setName("Four Wheeler");

		carList.add(car);

		GenericsTestWithSuper test = new GenericsTestWithSuper();
		test.testSubtype(carList);

	}

}
```

Look at the method testSubtype() here I add a new Car BMW and I pass List<Vehicle> in to list<? super Car>as Vehicle is parent of Car.

Written by: **Shamik Mitra**
Technical Leader at IBM
blog: http://javaonfly.blogspot.in/
facebook : https://www.facebook.com/shamik.mitra.37
mob : **9830471739** mail **:mitrashamik@gmail.com**
Provide Tuition on java, j2ee Struts,Spring,Hibernate,jpa,Hadoop,Devops,Agile,kanban

Still now we learn how to use generics and pass generics as an argument but can you think of if I mix old collection with generics what will happen

Example:

```
package com.example.generics;

import java.util.ArrayList;
import java.util.List;

publicclassGenericwithOldCollection {

	publicvoid testSubtype(List list)
	{
		Car v = new Car();
		v.setName("Four Wheeler");
		v.setBrand("BMW");


		list.add(v);
		list.add(new Integer(2));
		for(Object c : list)
		{
			Car car= (Car)c;

			System.out.println("Element is " + car);
		}

	}


	publicstaticvoid main(String[] args) {
		List<Vehicle> carList = new ArrayList<Vehicle>();
		Car car = new Car();
		car.setBrand("Audi");
		car.setName("Four Wheeler");

		carList.add(car);

		GenericwithOldCollection test = newGenericwithOldCollection();
		test.testSubtype(carList);

	}


}
```

If you mix it then it will be very risky. Compile will warn you by the message

List is a raw type. References to generic type List<E> should be parameterized

But if you not consider it then it acts as old collection and I can add anything to that collection as like here I add Integer object in to Vehicle list so it will throw a run time exception.

Output :

```
Element is Car [brand=Audi]
Element is Car [brand=BMW]
Exception in thread "main" java.lang.ClassCastException: java.lang.Integer cannot be cast to com.example.generics.Car
        at com.example.generics.GenericwithOldCollection.testSubtype(GenericwithOldCollection.java:19)
        at com.example.generics.GenericwithOldCollection.main(GenericwithOldCollection.java:36)
```

What will be scenario if we I wan to pass Raw list in to a Generics. Answer is it will be taken witout any complain.

Still now we only talk about generic initialization and pass Generics in a method. Now we will learn hoew to create a Generic class or Generic method which will take any type of Object

Syntax:

```
Public Class<T>{
T instance;
public void method(T variable)
{
}

}
```

Say We have to write a class which will add any Number tye like float,integer,double etc

Written by: **Shamik Mitra**
Technical Leader at IBM
blog: http://javaonfly.blogspot.in/
facebook : https://www.facebook.com/shamik.mitra.37
mob : **9830471739** mail **:mitrashamik@gmail.com**
Provide Tuition on java, j2ee Struts,Spring,Hibernate,jpa,Hadoop,Devops,Agile,kanban

So we need a class which will take Number type arguments so generics is sultable for it but generics syntx is class<T> so we can't add two generics Type so we need a interface and Anonymious class to add the generics type

```
package com.example.generics;

publicinterfaceAddOperation<T extends Number> {

	public  T add(T lhs, T rhs);

}
```

```
package com.example.generics;

publicclass Adder<T extends Number> {


	publicvoid add(T a, T b,AddOperation<T> addOp)
	{
		T result =addOp.add(a, b);
		System.out.println("Result is" + result);
	}


	publicstaticvoid main(String[] args) {

		Adder<Integer> addInteger = new Adder<Integer>();

		addInteger.add(1, 2, new AddOperation<Integer>(){

			public  Integer add(Integer lhs, Integer rhs) {
				return lhs+rhs;
			}


		});
```

```
        Adder<Long> addLong = new Adder<Long>();

                addLong.add(5L, 6L, new AddOperation<Long>(){

                    public  Long add(Long lhs, Long rhs) {
                        return lhs+rhs;
                    }


                });




            }

}
```

Output :

Result is 3
Result is 11

Use Generics in Method

In previous Example we have seen how to use Generic on your custom Class Now we will see if you want a make a generic metods what the things we need to do

Syntax:

```
public <T> void methodName(T argument)
{

}
```

Please note if we want to make a method as a generic method we have to provide <T>, place holder It is not return type it's tell that this method takes T type argument which is a generic argument at calling time it will be replaced with actual type

Written by: **Shamik Mitra**
Technical Leader at IBM
blog: http://javaonfly.blogspot.in/
facebook : https://www.facebook.com/shamik.mitra.37
mob : **9830471739** mail **:mitrashamik@gmail.com**
Provide Tuition on java, j2ee Struts,Spring,Hibernate,jpa,Hadoop,Devops,Agile,kanban

Example

```
package com.example.generics;

publicclass GenericMethodTest {


        public<T>void printValue(T instance)
        {
                System.out.println("class is "+ instance.getClass() + " value is " + in-
stance);
        }

        publicstaticvoid main(String[] args) {
                GenericMethodTest test = new GenericMethodTest();
                test.printValue(new Integer(2));
                test.printValue(new Float(2));
                test.printValue("Hi");
                test.printValue(new Integer[]{1,2});


        }
```

Output:

```
class is class java.lang.Integer value is 2
class is class java.lang.Float value is 2.0
class is class java.lang.String value is Hi
class is class [Ljava.lang.Integer; value is [Ljava.lang.Integer;@3e25a5

}
```

Written by: **Shamik Mitra**
Technical Leader at IBM
blog: http://javaonfly.blogspot.in/
facebook : https://www.facebook.com/shamik.mitra.37
mob : **9830471739** mail **:mitrashamik@gmail.com**
Provide Tuition on java, j2ee Struts,Spring,Hibernate,jpa,Hadoop,Devops,Agile,kanban

Chapter 18

Java Thread

What is a Thread?

a thread of execution is the smallest sequence of programmed instructions that can be managed independently by a scheduler.

So I can say If I have a program and it need to execute then I must a thread to execute it. Thread can be seen as a Straw by which you can drink a Cococola Where cococola is the program and it is manged/can be drunk by Straw. It is single thread environment. Where Only one thread is involved.

Having said that a question arrised so still now we have written many dozens of program so any thread is involved to that?

The answer is yes in every method we have use public static void main which spwan a **main** thread. Every program we had written is manges by main thread.

Now going to further deep if Two strawsare usedto drink theCococola from bottle then by two straws, two people can drink the same provided, they need some synchronization. It is call Multithread example. So when a program is shared among multiple threads then it calls Multithreaded Environment.

To be precise, Java is a multi threaded programming languageso we can develop multi threaded program using Java. A multi threaded program contains two or more threads

that can run concurrently and each part can handle different task at the same time making optimal use of the available resources specially when your computer has multiple CPUs. So speed up an execution suppose If a task is done by a single man and same task performed by Ten Men then it takes 1/10 th time lesser.

By definition multitasking is when multiple processes share common processing resources such as a CPU. Multi threading extends the idea of multitasking into applications where you can subdivide specific operations within a single application into individual threads. Each of the threads can run in parallel. The OS divides processing time not only among different applications, but also among each thread within an application.

Multi threading enables you to write in a way where multiple activities can proceed concurrently in the same program provided some synchronization so that sharable resourse can't go messy. We will discuss it later.

Thread LifeCycle

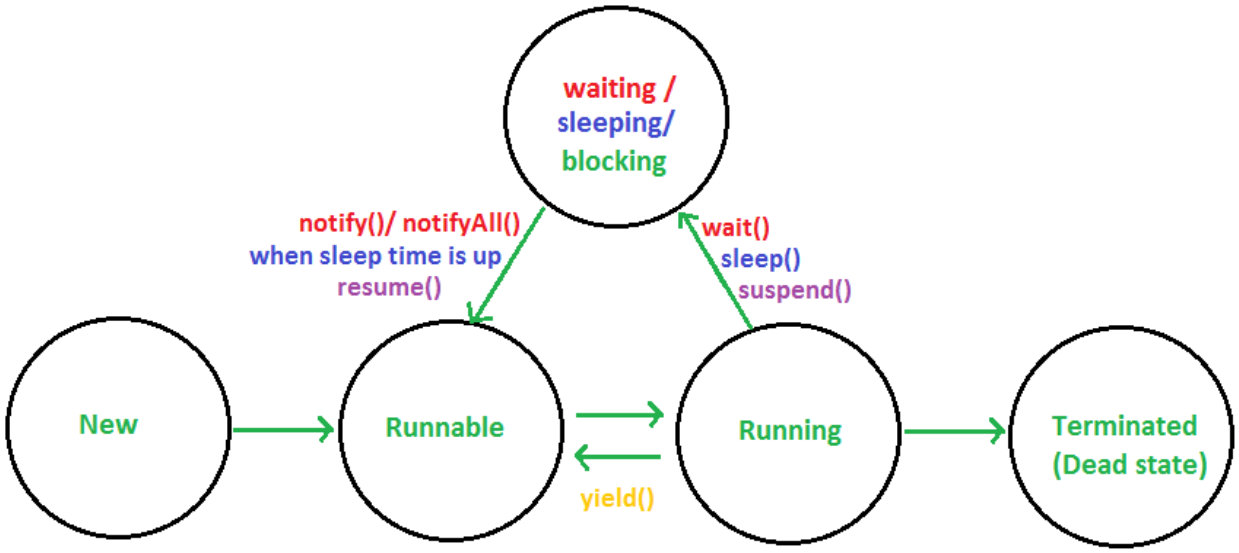


Fig. THREAD STATES

New State: when a new thread spawn, its begins life cycle in the new state. It remains

in this state until the program starts the thread(thread.start() method). It is also referred to as a born thread.

Runnable: After a newly born thread is started, the thread becomes runnable. A thread in this state is considered to be executing its task. Sometimes we called it Thread is in ready state.

Waiting: When a thread is wating for a resource while another thread performs a task on that resource. First thread enters in Wating state. A thread transitions back to the runnable state only when another thread signals the waiting thread to continue executing.

Running: When a thread is in runnable state and thread scheduler picks it up for execution and it start the execution it's enter in running state. From running state, it can go to wating state if thread schedular halts it.

Dead: When a Thread complet it's execution it goes to dead state . the thread is destroyed.

Thread Priorities:

Java thread has a priority that helps the operating system determine the prirority order foe scheduled threads

Java thread priorities are in the range between MIN_PRIORITY (a constant of 1) and MAX_PRIORITY (a constant of 10). By default, every thread is given priority NORM_PRIORITY (a constant of 5). Please note that is java thread priroty not the OS level priority it may possible OS level as priroty level (1-5) but one thing sure higer priroty thread given priroty that loer priroty by Thread scheduler.

Threads with higher priority are more important to a program and should be allocated processor time before lower-priority threads. However, thread priorities cannot guarantee the order in which threads execute and very much platform dependent.

Written by: **Shamik Mitra**
Technical Leader at IBM
blog: http://javaonfly.blogspot.in/
facebook : https://www.facebook.com/shamik.mitra.37
mob : **9830471739** mail **:mitrashamik@gmail.com**
Provide Tuition on java, j2ee Struts,Spring,Hibernate,jpa,Hadoop,Devops,Agile,kanban

Thread Type

Two type of thread is available

a. Normal thread
b. Daemon Thread

Normal thread: Thease thread are associated with program. It may be main thread or spawn from main thread when program shutsdown abruptly JVM wait for Nornmal threads to complete their tasks.

Daemon Thread: This type of thread are not directly associated with program. This threads are executing in background and perform certain task. when program shuts-down abruptly JVM does not wait for Daemon threads to complete their tasks.

Garbagecollector is such type of thread.

Create Thread in Java

There are two ways we can create thread in java.

1. By Extending Thread
2. By implementing Runnable

1. By Extending Thread class :

We can Extends a thread class and override the public void run Method to create a thread.

Example:

```
package com.example.thread;

publicclass CreateThreadbyExtend  extends Thread{


@Override
publicvoid run()
{
	System.out.println("Execute the Thread" + Thread.currentThread().getName());
}

publicstaticvoid main(String[] args) {

	System.out.println("Execute " + Thread.currentThread().getName());

	Thread thread = new CreateThreadbyExtend();
	thread.setName("My Custom thread");
	thread.start();

}




}
```

Output :

```
Execute main
Execute the Thread My Custom thread
```

Look the program carefully Here I extend the Thread class and Override the public void run method. This is the key method where you put your business logic , which will be executed by thread. But look carefully we not call the run method anywhere we call thread.start(). By invoking start method, it internally calls run and create a separate path of execution or I can say it spwan a thread.

TIP: Remember thread.start() internally call run method. If you call run explicitly then it acts as a simple method but not spawn a new thread or separate path of execution. So never call run method programitically

Another point need to be discussed pay attention to the code in main method here I use Thread.currentThread().getName() which will print the name of current name of the thread . In first line I invoke Thread.*currentThread*().getName() so, it prinis the name of main thread which is main. By this we can prove every program runs under main thread after that we create CreateThreadbyExtend Object and invoke start method now when we call Thread.*currentThread*().getName(), now it prints "My custom thread" as currently custom thread is executing.

2. **By Implementing Runnable:**
   Another way to create thread is use Runnable interface and Overide the public void run method. This is most elegant way to create a thread. As it implements interface we can use other interface and also extend any other class. But in case of extending Thread once, you can't extend any Other class. The class which implements the Runnable interface we call it target of the thread.

Example :

```
package com.example.thread;

publicclass CreateThreadByImplement implements Runnable{

	publicvoid run() {


		for(int i=0;i<10;i++)
		{
			System.out.println(Thread.currentThread().getName() + i);

		}

	}


	publicstaticvoid main(String[] args) {

		CreateThreadByImplement runnable = new CreateThreadByImplement();

		Thread th = new Thread(runnable);
		th.setName("Thread First");

		Thread th1 = new Thread(runnable);
```

```
                th1.setName("Thread Second");

                th.start();
                th1.start();

        }

}
```

Output:

```
Thread First0
Thread First1
Thread First2
Thread First3
Thread First4
Thread First5
Thread Second0
Thread Second1
Thread Second2
Thread Second3
Thread Second4
Thread First6
Thread Second5
Thread First7
Thread Second6
Thread First8
Thread Second7
Thread First9
Thread Second8
Thread Second9
```

Here I create a class **CreateThreadByImplement class** implement runnable interface. Look at the output it is random and each time you run the program you may got different results as because Thread scheduler pick up thread then hals that thread takes second one. It is upon Thread scheduler but one thing we gurrenty tha all the thread will execute.

Written by: **Shamik Mitra**
Technical Leader at IBM
blog: http://javaonfly.blogspot.in/
facebook : https://www.facebook.com/shamik.mitra.37
mob : **9830471739** mail **:mitrashamik@gmail.com**
Provide Tuition on java, j2ee Struts,Spring,Hibernate,jpa,Hadoop,Devops,Agile,kanban

Thread Synchronization

Let discuss about a scenario, A Man and his wife share a Debit card ,say initially Account has 10000 rupees. Man go to pub and Wife ges for Shopping. Now they enjoy lot there now time for pay bill. But Here problem arise at same time Man and his wife wants to pay the bill so man insert his debit card and same for his wife in same time so each can see it has 10,000 rupees. man bill is 6000 and his wife bill is 8000 so both are under 10,000 so they can easily pay the bill and left the place. But It leads a very dengarous situation because although Account has 10000 ruppes actually they pais 8000+6000=14000 ruppes so 4000 is going to lost world. If it is continuing bank rupt is inhebitable but where the problem lies?

Problem is lack of Syncronization Some kHow Bank has to manage the transaction and when one transaction is processing somhow for another transaction on that debit card should be wait once first transaction is end then second one can start. In this way we can get rid of thet problem. So manageing this messy condition on sharable resource. Here Acoount is sharable as Man and his wife share this account. We must have to do some sort of synchronization. We can think of Two transaction as two threads as they are separate path of execution but on same Resource that is Account.

When we acces Sharable resource we need to synchronize the action of multiple threads and make sure that only one thread can access the resource at a given point in time. This is implemented using a concept called **monitors**. Each object in Java is associated with a monitor, which a thread can lock or unlock. Only one thread at a time may hold a lock on a monitor.

Java programming language provides a very elegant way of creating threads and synchronizing their task by using **synchronized** blocks. You keep shared resources within this block. Following is the general form of the synchronized statement.

So much of theory I am just going mad. Let’s understand it with example

Example:

```
package com.example.thread;

publicclass Account {

        public Account(int balance)
```

```
        {
                this.balance = balance;
        }

        publicintbalance;

        publicint checkBalance()
        {
                returnbalance;
        }

        publicboolean verifyBalance(int withdrawlAmount)
        {
                return withdrawlAmount<=balance;
        }

        publicvoid withdraw(int amnt)
        {

                        balance = balance -amnt;
                        System.out.println("new Balance is " + balance);
                        return;



        }

        publicvoid credit(int amnt)
        {

                        balance = balance + amnt;

        }


        publicint getBalance() {
                returnbalance;
        }

        publicvoid setBalance(int balance) {
                this.balance = balance;
        }

}
```

Written by: **Shamik Mitra**
Technical Leader at IBM
blog: http://javaonfly.blogspot.in/
facebook : https://www.facebook.com/shamik.mitra.37
mob : **9830471739** mail **:mitrashamik@gmail.com**
Provide Tuition on java, j2ee Struts,Spring,Hibernate,jpa,Hadoop,Devops,Agile,kanban

```
package com.example.thread;

publicclass DebitCard implements Runnable{

	Account acc;
	intwithdrawAmount=0;

	publicint getWithdrawAmount() {
		returnwithdrawAmount;
	}

	publicvoid setWithdrawAmount(int withdrawAmount) {
		this.withdrawAmount = withdrawAmount;
	}

	public DebitCard(Account acc )
	{
		this.acc =acc;
	}

	publicvoid run() {

		int balance = acc.checkBalance();
		System.out.println(Thread.currentThread().getName()+ " Current balance
is " + balance);
		boolean b = acc.verifyBalance(this.getWithdrawAmount());
		try {
			Thread.sleep(1000);
		} catch (InterruptedException e) {
			// TODO Auto-generated catch block
			e.printStackTrace();
		}

		if(b)
		{
			acc.withdraw(this.getWithdrawAmount());
		}


		System.out.println(Thread.currentThread().getName()+ " After withdraw
Current balance is " + acc.checkBalance());


			}


	publicstaticvoid main(String[] args) {
```

```
                Account acc = new Account(10000);
                DebitCard dc=new DebitCard(acc);
                DebitCard dc1=new DebitCard(acc);
                Thread th = new Thread(dc);
                dc.setWithdrawAmount(6000);
                th.setName("Shamik");
                th.start();


                Thread th1 = new Thread(dc1);
                dc1.setWithdrawAmount(8000);
                th1.setName("Swastika");
                th1.start();

        }

}
```

Output :

```
Shamik Current balance is 10000
Swastika Current balance is 10000
new Balance is -4000
new Balance is -4000
Shamik After withdraw Current balance is -4000
Swastika After withdraw Current balance is -4000
```

Carefully inspect the code. Lot to describe here.

I have created an Account Object when we created this we need to put some Amount in this so I pass this as a constructor argument. Makes it a required parameter.

Account has two important methods verifyBalance and withdraw. By verifyBalance We check withdraw amount is less than current balance, if so then we not going to with-draw. In DebitCard class I create Two Debit Card Object point to same Account and Start two withdraw operations one is rs 6000 another rs 8000. But Hey what going on we ended up with a messy state with new account balance -4000.

What goes wrong?

Pay attention to the code written in run we first check the verify then withdraw but here

Written by: **Shamik Mitra**
Technical Leader at IBM
blog: http://javaonfly.blogspot.in/
facebook : https://www.facebook.com/shamik.mitra.37
mob : **9830471739** mail **:mitrashamik@gmail.com**
Provide Tuition on java, j2ee Struts,Spring,Hibernate,jpa,Hadoop,Devops,Agile,kanban

two thread simalteniously check verifyBalance so both get 10000 and they want to withdraw 6000 and 8000 so it is fine and Account ended up with -4000. But if you think minutely here is a catch verifyBalance and withdraw shulb be a unit operation. That is when one thread checks the balance and going to withdraw other thread shuld not interfare. Unless it ended up with dirty state. So synchronization nedded between thread.

When a thread checks and withdraw other should wait, so by synchronization we can solve this problem.

```
package com.example.thread;

publicclass Account {

	public Account(int balance)
	{
		this.balance = balance;
	}

	publicintbalance;

	publicint checkBalance()
	{
		returnbalance;
	}

	publicboolean verifyBalance(int withdrawlAmount)
	{
		return withdrawlAmount<=balance;
	}

	publicvoid withdraw(int amnt)
	{

			balance = balance -amnt;
			System.out.println("new Balance is " + balance);
			return;



	}


	publicsynchronizedvoiddoDebitOperation(int amnt)
	{
		int balance = this.checkBalance();
```

```
                System.out.println(Thread.currentThread().getName()+ " Current balance
    is " + balance);
                boolean b = this.verifyBalance(amnt);
                try {
                        Thread.sleep(1000);
                } catch (InterruptedException e) {
                        // TODO Auto-generated catch block
                        e.printStackTrace();
                }

                if(b)
                {
                        this.withdraw(amnt);
                }


                System.out.println(Thread.currentThread().getName()+ " After withdraw
    Current balance is " + this.checkBalance());



        }


        publicvoid credit(int amnt)
        {

                        balance = balance + amnt;

        }


        publicint getBalance() {
                returnbalance;
        }

        publicvoid setBalance(int balance) {
                this.balance = balance;
        }

    }



    package com.example.thread;

    publicclass SynchDebitCard implements Runnable{
```

```
        Account acc;
        intwithdrawAmount=0;

        publicint getWithdrawAmount() {
                returnwithdrawAmount;
        }

        publicvoid setWithdrawAmount(int withdrawAmount) {
                this.withdrawAmount = withdrawAmount;
        }

        public SynchDebitCard(Account acc )
        {
                this.acc =acc;
        }


        publicvoid run() {

                acc.doDebitOperation(this.getWithdrawAmount());

                        }


        publicstaticvoid main(String[] args) {

                Account acc = new Account(10000);
                SynchDebitCard dc=new SynchDebitCard(acc);
                SynchDebitCard dc1=new SynchDebitCard(acc);
                Thread th = new Thread(dc);
                dc.setWithdrawAmount(6000);
                th.setName("Shamik");
                th.start();


                Thread th1 = new Thread(dc1);
                dc1.setWithdrawAmount(8000);
                th1.setName("Swastika");
                th1.start();

        }

}
```

Output:

```
Shamik Current balance is 10000
new Balance is 4000
Shamik After withdraw Current balance is 4000
Swastika Current balance is 4000
Swastika After withdraw Current balance is 4000
```

Just few tweaks and we magically resolve the problem and Bank saves himself from going to rupt ☺

Let see what we have done. We created a new synchronized method in Account class as this is the sharable resource between two debit card. When transaction goes two debit card checks same account so I create a new operation called doDebitOperation where first I check the balance then perform the operation if all is well. So make verify-Balance and withdraw operation atomic to write it in a Synchronized method.

And just call it from SynchDebit card.

wait and notify:

Let understand it by a story. Suppose you go to KFC to buy our faviourite Kenatucky fried Chicken. You orderd for it but you have to wait few minutes to actually get it served. Why?? Because it need to be processed I mean they take it out from hot chamber process it prepares it in dish take money from you then serve it. So think without paying money or not being processed you can't get it. So some synchronization needed obviously first it has to be processed then only it can be served.

Let understand it with thread perspective. You is a thread and who processed the KFC chicken is another thread KFC chicken is only sharable resource between you and him.

But the condition is first KFC need to be processed then you can get it so if you order it first you can not get it immidietely you have to **wait**once it's processing is finished by processing (another thread) he acknowledge you by saying "Sir Here is your KFC chiken" or **notify** you**.**

TIP:So when multiple threads works on a sharable resource to do in it an orderable fashon we use wait and notify to maintains the order of synchronization. Please note that wait and notify is Object's method not Thread class. Wait and notify always use unse synchronize block or method.

Written by: **Shamik Mitra**
Technical Leader at IBM
blog: http://javaonfly.blogspot.in/
facebook : https://www.facebook.com/shamik.mitra.37
mob : **9830471739** mail **:mitrashamik@gmail.com**
Provide Tuition on java, j2ee Struts,Spring,Hibernate,jpa,Hadoop,Devops,Agile,kanban

Example:

```
package com.example.thread;

publicclass KFC {

	privatestatic KFC kfc = new KFC(10);

	publicstatic KFC getInstance()
	{
		returnkfc;
	}


	privateintorderAmount;
	booleanprocessed=false;
	private KFC()
	{

	}
	private KFC(int amnt)
	{
		this.orderAmount=amnt;
	}

	publicsynchronizedvoid processKFC(int i) throws InterruptedException
	{


				if(!processed)
				{
					System.out.println("Take out KFC piece " + "KFC"+i);

					System.out.println("put it in Hot chamber for 2 mi-
niutes "+ "KFC"+i);
					System.out.println("Decorate "+ "KFC"+i);
					System.out.println("********************");
					processed=true;
					notify();
				}
				else
				{
					wait();
					System.out.println("Wating for customer confirmation
he has finshed " + "KFC");
```

```
                        }


    }

    publicsynchronizedvoid serveKFC(int i) throws InterruptedException
    {


            if(!processed)
            {
                wait();
                System.out.println("Wating eagerly for " + "KFC"+i);
            }
            System.out.println("Serve" + "KFC"+i);
            System.out.println("eat"+ "KFC"+i);
            System.out.println("wating for next "+ "KFC");
            System.out.println("*****************************");
            processed=false;
            notify();

    }



    publicint getOrderAmount() {
        returnorderAmount;
    }
    publicvoid setOrderAmount(int orderAmount) {
        this.orderAmount = orderAmount;
    }
    publicstaticvoid main(String[] args) {


        Thread processor =new Thread(){

            publicvoid run()
            {
                try {
                    for(int i =0 ;
i<KFC.getInstance().getOrderAmount();i++)
                    {
                        KFC.getInstance().processKFC(i);
                        Thread.sleep(1000);
```

```
                                        }

                                    } catch (InterruptedException e) {
                                        // TODO Auto-generated catch block
                                        e.printStackTrace();
                                    }
                                }
                            };

    Thread customer =new Thread(){

                                publicvoid run()
                                {
                                    try {
                                        for(int i =0 ;
i<KFC.getInstance().getOrderAmount();i++)
                                        {
                                            KFC.getInstance().serveKFC(i);
                                            Thread.sleep(1000);
                                        }
                                    } catch (InterruptedException e) {
                                        // TODO Auto-generated catch block
                                        e.printStackTrace();
                                    }
                                }
                            };

                            processor.start();
                            customer.start();

            }

    }
```

Output:

```
Take out KFC piece KFC0
put it in Hot chamber for 2 miniutes KFC0
Decorate KFC0
******************
ServeKFC0
eatKFC0
wating for next KFC
****************************
Take out KFC piece KFC1
put it in Hot chamber for 2 miniutes KFC1
```

```
Decorate KFC1
**********
Wating eagerly for KFC1
ServeKFC1
eatKFC1
wating for next KFC
**********
Take out KFC piece KFC2
put it in Hot chamber for 2 miniutes KFC2
Decorate KFC2
**********
Wating eagerly for KFC2
ServeKFC2
eatKFC2
wating for next KFC
**********
Take out KFC piece KFC3
put it in Hot chamber for 2 miniutes KFC3
Decorate KFC3
**********
ServeKFC3
eatKFC3
wating for next KFC
**********
Take out KFC piece KFC4
put it in Hot chamber for 2 miniutes KFC4
Decorate KFC4
**********
ServeKFC4
eatKFC4
wating for next KFC
**********
Take out KFC piece KFC5
put it in Hot chamber for 2 miniutes KFC5
Decorate KFC5
**********
Wating eagerly for KFC5
ServeKFC5
eatKFC5
wating for next KFC
**********
Take out KFC piece KFC6
put it in Hot chamber for 2 miniutes KFC6
Decorate KFC6
**********
ServeKFC6
eatKFC6
```

```
wating for next KFC
**************************
Take out KFC piece KFC7
put it in Hot chamber for 2 miniutes KFC7
Decorate KFC7
******************
Wating eagerly for KFC7
ServeKFC7
eatKFC7
wating for next KFC
**************************
Take out KFC piece KFC8
put it in Hot chamber for 2 miniutes KFC8
Decorate KFC8
******************
Wating eagerly for KFC8
ServeKFC8
eatKFC8
wating for next KFC
**************************
Take out KFC piece KFC9
put it in Hot chamber for 2 miniutes KFC9
Decorate KFC9
******************
ServeKFC9
eatKFC9
wating for next KFC
**************************
```

Here I create a KFC class with two methods processKFC and serveKFC then two threads Customer and processor acting on KFC if Customer first takes control over KFC, then he has to wait as processed flag is flase. Once processor thread processes the KFC piece it notifies customer and customer then eat the delicious KFC and wait for next piece. ☺

Thread Join

join method is another way to do inter thread comminucation. By join method invoked on a thread tells that current execution thread is wating for the thread whose join method has been called. After finish of that thread current Thread resume the processing.

Example

```
package com.example.thread;

publicclass ThreadJoinexample {

	publicstaticvoid main(String[] args) throws InterruptedException {


		System.out.println("Current thread is " +
Thread.currentThread().getName());
		System.out.println(Thread.currentThread().getName() + " : I am going to
wait for Thread shamik");
		Thread t = new Thread()
		{
			publicvoid run()
			{
				for(int i=0;i<10;i++)
				{
					System.out.println(Thread.currentThread().getName()
+ " :Executing and Counting " + i);
				}
			}
		};

		t.setName("Shamik");


		t.start();
		t.join();
		System.out.println(Thread.currentThread().getName()+  " :resume now");
	}

}
```

Output :

```
Current thread is main
main : I am going to wait for Thread shamik
Shamik :Executing and Counting 0
Shamik :Executing and Counting 1
Shamik :Executing and Counting 2
Shamik :Executing and Counting 3
```

```
Shamik :Executing and Counting 4
Shamik :Executing and Counting 5
Shamik :Executing and Counting 6
Shamik :Executing and Counting 7
Shamik :Executing and Counting 8
Shamik :Executing and Counting 9
main :resume now
```

In above program Main thread call join on Thread shamik so main is wating for shamik to finish it's job then resume it's execution.

Thread Sleep

If we invoke sleep on current thread it goes to awiting state but it does not release the resource or It does not release Object monitor. Another version is you can pass time in to Tsleeo method. In that case Thread going to wait satae for that period of time then resume its execution.

Thread Deadlock

Deadlock is such a scenario when one thread is wating for a resource that is occupied by another thread and wating for this resource for ever as that thread also want to grab a resource but can't due to unavialibilty or in use by another thread.

TIP: Suppose thraer is two threads call A and B and A accurie a resource R1 and wait for resource R2 and Thread B accuire R2 wait fir R1 then the deadlock arise as A wait for R2 which is accuires by Thread B and vice versa so they wait for ever . deadlock is very bad thing always design your soft ware deadlock free.

Example

```
package com.example.thread.deadlock;

publicclass DeadLock {
publicstaticvoid main(String[] args) {
final String resource1 = "Shamik Mitra";
final String resource2 = "Swastika Mitra";
```

```
    // t1 tries to lock resource1 then resource2
         Thread t1 = new Thread() {
    publicvoid run() {
    synchronized (resource1) {
                System.out.println("Thread 1: locked resource 1");

    try { Thread.sleep(100);} catch (Exception e) {}

    synchronized (resource2) {
                 System.out.println("Thread 1: locked resource 2");
               }
             }
           }
         };

    // t2 tries to lock resource2 then resource1
         Thread t2 = new Thread() {
    publicvoid run() {
    synchronized (resource2) {
               System.out.println("Thread 2: locked resource 2");

    try { Thread.sleep(100);} catch (Exception e) {}

    synchronized (resource1) {
                 System.out.println("Thread 2: locked resource 1");
               }
             }
           }
         };


         t1.start();
         t2.start();
       }
     }
```

Output :

Thread 1: locked resource 1

Thread 2: locked resource 2

Written by: **Shamik Mitra**
Technical Leader at IBM
blog: http://javaonfly.blogspot.in/
facebook : https://www.facebook.com/shamik.mitra.37
mob : **9830471739** mail **:mitrashamik@gmail.com**
Provide Tuition on java, j2ee Struts,Spring,Hibernate,jpa,Hadoop,Devops,Agile,kanban

Written by: **Shamik Mitra**
Technical Leader at IBM
blog: http://javaonfly.blogspot.in/
facebook : https://www.facebook.com/shamik.mitra.37
mob : **9830471739** mail **:mitrashamik@gmail.com**
Provide Tuition on java, j2ee Struts,Spring,Hibernate,jpa,Hadoop,Devops,Agile,kanban

Written by: **Shamik Mitra**
Technical Leader at IBM
blog: http://javaonfly.blogspot.in/
facebook : https://www.facebook.com/shamik.mitra.37
mob : **9830471739** mail **:mitrashamik@gmail.com**
Provide Tuition on java, j2ee Struts,Spring,Hibernate,jpa,Hadoop,Devops,Agile,kanban

Written by: **Shamik Mitra**
Technical Leader at IBM
blog: http://javaonfly.blogspot.in/
facebook : https://www.facebook.com/shamik.mitra.37
mob : **9830471739** mail **:mitrashamik@gmail.com**
Provide Tuition on java, j2ee Struts,Spring,Hibernate,jpa,Hadoop,Devops,Agile,kanban

Written by: **Shamik Mitra**
Technical Leader at IBM
blog: http://javaonfly.blogspot.in/
facebook : https://www.facebook.com/shamik.mitra.37
mob : **9830471739** mail **:mitrashamik@gmail.com**
Provide Tuition on java, j2ee Struts,Spring,Hibernate,jpa,Hadoop,Devops,Agile,kanban

.

Written by: **Shamik Mitra**
Technical Leader at IBM
blog: http://javaonfly.blogspot.in/
facebook : https://www.facebook.com/shamik.mitra.37
mob : **9830471739** mail **:mitrashamik@gmail.com**
Provide Tuition on java, j2ee Struts,Spring,Hibernate,jpa,Hadoop,Devops,Agile,kanban

Written by: **Shamik Mitra**
Technical Leader at IBM
blog: http://javaonfly.blogspot.in/
facebook : https://www.facebook.com/shamik.mitra.37
mob : **9830471739** mail **:mitrashamik@gmail.com**
Provide Tuition on java, j2ee Struts,Spring,Hibernate,jpa,Hadoop,Devops,Agile,kanban

Written by: **Shamik Mitra**
Technical Leader at IBM
blog: http://javaonfly.blogspot.in/
facebook : https://www.facebook.com/shamik.mitra.37
mob : **9830471739** mail **:mitrashamik@gmail.com**
Provide Tuition on java, j2ee Struts,Spring,Hibernate,jpa,Hadoop,Devops,Agile,kanban

Written by: **Shamik Mitra**
Technical Leader at IBM
blog: http://javaonfly.blogspot.in/
facebook : https://www.facebook.com/shamik.mitra.37
mob : **9830471739** mail **:mitrashamik@gmail.com**
Provide Tuition on java, j2ee Struts,Spring,Hibernate,jpa,Hadoop,Devops,Agile,kanban