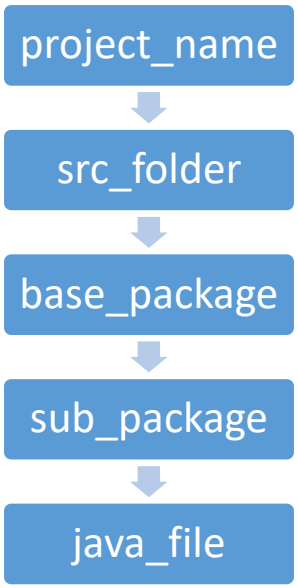
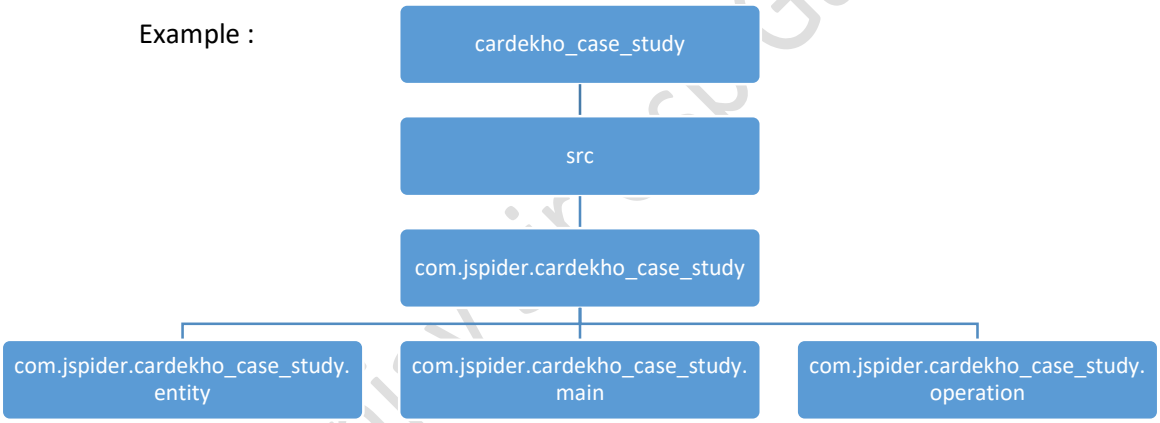
PROJECT STRUCTURE



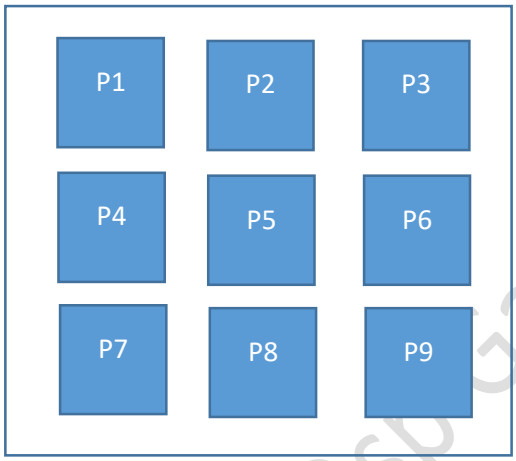
Example :



- **Base package naming convention**
  - Format:
    - Domain.host_name.project_name
  - Eg.
    - com.jspiders.cardekho_case_study
- **Sub package Naming convention**
  - Format:
    - Base_package_name.package_name
  - Eg.
    - Com.jspiders.cardekho_case_study.main

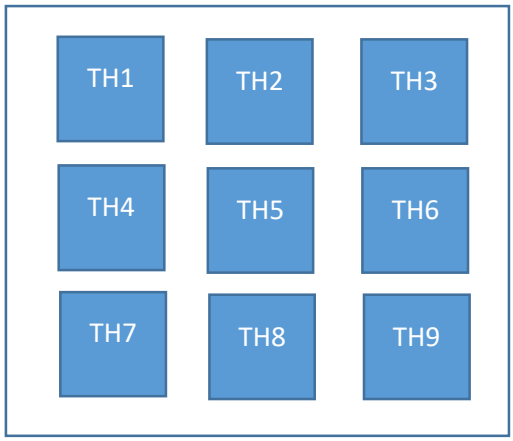
Multithreading

- To understand the concept of multithreading we need to understand what is multitasking
- **Multitasking**
  - Performing more than one task at the same time is known as multitasking
  - A task the end goal that has to achieved are known process of task
  - Hence it can be said that task is combination of multiple process
    - Processes-



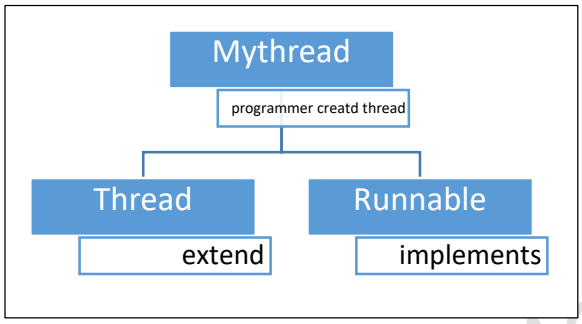
TASK

    - In order to achieve the completion of the task in lesser time or reduce the time taken for the task completion we need to implement multitasking
- **Multiprocessing**
  - Executing more than one task at the same time is known as multiprocessing
  - As process is step involved in achieving the task completion, it is also divided into smaller unit
  - The smallest unit of a process is known as a thread
  - Hence it can be said that the process of the combination of more than one (multiple) thread



Process

- **Thread**
  - Thread can be defining as the smallest unit of a process or a lightweight process
  - Execution of more than one thread at the same time is known as multithreading
  - In java thread as considered as special class which can be created in two different ways
    - By extending Thread class
    - By implementing Runnable interface



  - **start**()
    - It is nonstatic method present in thread class
    - To start a class as a thread, we need to call this method with its object ref variable
  - **run**()
    - It is a method from runnable interface which is also overridden in Thread class
    - The execution logic of a thread has to be declared in this method
    - We don't need to explicitly call this method the start method calls it implicitly
  - **Note –**
    - **If the start method is not use and the run method is called directly then the class behaves like a normal class and not as a thread**


```
package com.jspjkliders.multithreading.thread;

public class MyThread1 extends Thread {
	@Override
	public void run() {
		System.out.println("My thread 1 is running now !");
	}

}
package com.jspiders.multithreading.main;

import com.jspiders.multithreading.thread.MyThread1;

public class ThreadMain {

	public static void main(String[] args) {
		MyThread1 myThread1 = new MyThread1();
		myThread1.start();
	}
}
```

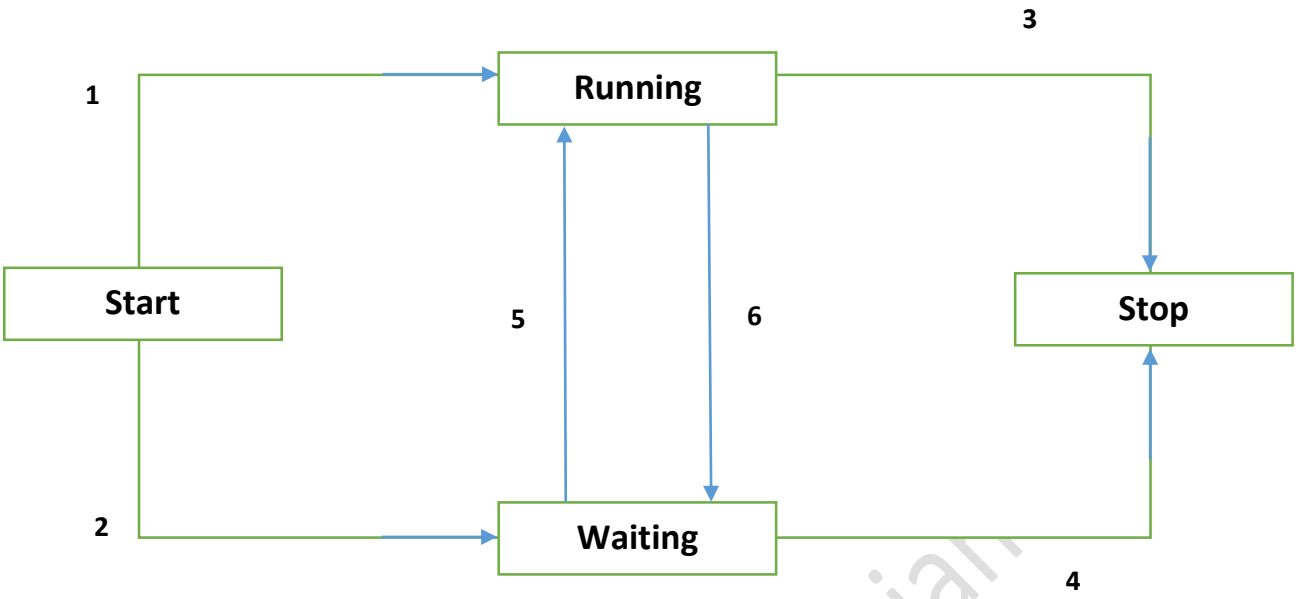
- When thread is created by implementing the runnable interface, then we do not have the access of the start method with the object of the user defined Thread class (Mythread2)
- Hence we need to create the object of the Thread class directly & pass the object of the user define Thread class as an argument to the overloaded constructor of the Thread class
- Now with the object of Thread class, we can directly access the start method which will implicitly call the run method present in the user define Thread class

Q. What is the better way to create Thread? and Why?

- Implementing the interface its better way to create a thread
- Because when we extend Thread class with another class at that time we cannot extends multiple class at a same time
- There should be diamond problem happen
- To avoid this restriction, we can implement the Runnable interface, in this scenario we can extend another class also
- That's why the implement the Runnable interface is the better way to create a Thread

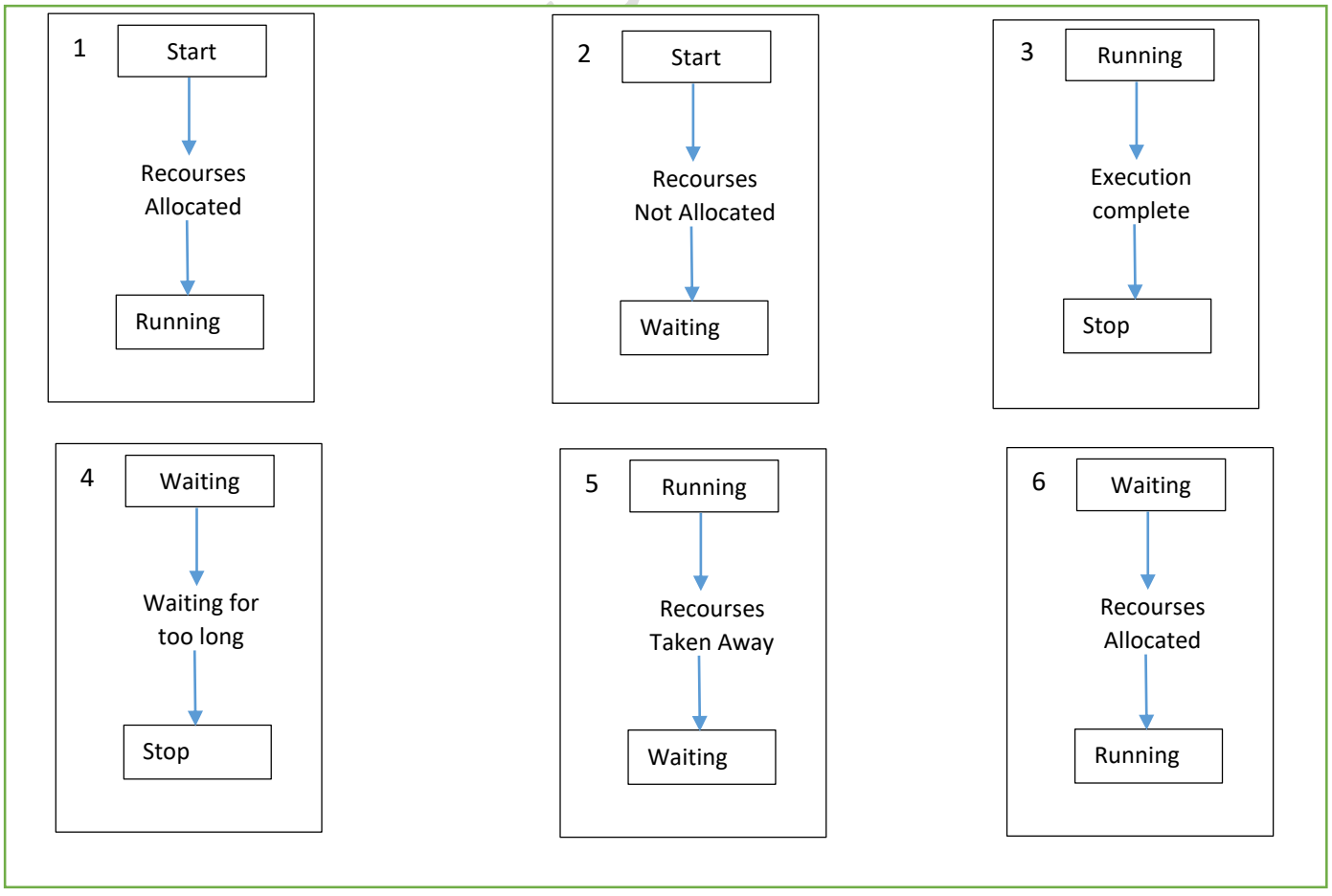

```
package com.jspiders.multithreading.thread;

public class MyThread2 extends Thread{

	@Override
	public void run() {
		System.out.println("Mythread2 is running now ! ");
	}

}
package com.jspiders.multithreading.main;


import com.jspiders.multithreading.thread.MyThread1;
import com.jspiders.multithreading.thread.MyThread2;

public class ThreadMain2 {
	public static void main(String[] args) {

			MyThread1 myThread1 = new MyThread1();
			MyThread2 myThread2 = new MyThread2();

			myThread1.start();
			myThread2.start();
	}
}
```

- Life Cycle of Thread



- Whenever a thread is initializing it moves to **start** phase
- If the thread is executing, then we can say that it is in **running** phase
- If the thread is started but not executing, then we can say that it is in **waiting** phase
- When thread is terminated or ended then we can say that it case in **stop** phase

Transition of Thread



- Executing Multiple Thread Simultaneously

```java
package com.jspiders.multithreading.thread;

public class MyThread1 extends Thread{
	@Override
	public void run() {
		for (int i = 0; i < 5; i++) {
			System.out.println("MyThread 1 is Now Running...");
		}
	}
}
```

```java
package com.jspiders.multithreading.thread;

public class MyThread2 extends Thread{

	@Override
	public void run() {
		for (int i = 0; i < 5; i++) {
			System.out.println("MyThread 2 is Now Running...");
		}
	}
}
```

```java
package com.jspiders.multithreading.main;


import com.jspiders.multithreading.thread.MyThread1;
import com.jspiders.multithreading.thread.MyThread2;

public class ThreadMain2 {
	public static void main(String[] args) {

			MyThread1 myThread1 = new MyThread1();
			MyThread2 myThread2 = new MyThread2();

			myThread1.start();
			myThread2.start();
	}
}
```

Expected O/p –

```
MyThread 1 is Now Running...
MyThread 1 is Now Running...
MyThread 1 is Now Running...
MyThread 1 is Now Running...
MyThread 1 is Now Running...
MyThread 2 is Now Running...
MyThread 2 is Now Running...
MyThread 2 is Now Running...
MyThread 2 is Now Running...
MyThread 2 is Now Running...
```

Actual Output-

```
MyThread 1 is Now Running...
MyThread 2 is Now Running...
MyThread 2 is Now Running...
MyThread 2 is Now Running...
MyThread 2 is Now Running...
MyThread 2 is Now Running...
MyThread 1 is Now Running...
MyThread 1 is Now Running...
MyThread 1 is Now Running...
MyThread 1 is Now Running...
```

- **Note -**
  - The actual output the above program is not constant it is happening due due to component known as 'ThreadSheduler'
- **Thread Scheduler**
  - It is a component in multithreading which is responsible to manage the lifecycle of the Thread
  - Whenever the Thread is initialise(started) the thread scheduler is responsible to allocate a dedicated stack for the execution of that thread
- **Properties Of Thread**
  - There are three properties for every Thread
    - Id
    - Name
    - Priority
  - This priority we can access by using the helper methods (getter & setters)
    - Name
      - getName()
      - setName(String Name)
    - priority
      - getPriority()
      - setPriority(int priority)
- The default name for each Thread starts from 'Thread-0'
- The default priority for each Thread is always 5

```
package com.jspiders.multithreading.thread;

public class MyThread3 extends Thread {

	@Override
	public void run() {
		System.out.println("Thread name : " + getName());
		System.out.println("Thread priority : " + this.getPriority());
	}
}

package com.jspiders.multithreading.main;

import com.jspiders.multithreading.thread.MyThread3;

public class ThreadMain3 {

	public static void main(String[] args) {

		MyThread3 myThread3 = new MyThread3();
		myThread3.setName("Thread3");
		myThread3.setPriority(7);
		myThread3.start();
	}
```

- getName()
  - it is a nonstatic method present in the Thread class it is used to retrieve the name of a thread
- setName()
  - it is a nonstatic method present in the Thread class it is used to modify the name of a thread
- getPriority()
  - it is nonstatic method from Thread class
  - it is used to retrieve the priority of a thread
- setPriority()
  - it is a nonstatic method from Thread class it is used to modify the priority of thread

```
package com.jspiders.multithreading.thread;

public class MyThread4 implements Runnable {

	@Override
	public void run() {
		System.out.println("Name of thread : "
					+ Thread.currentThread().getName());
		System.out.println("Priority of thread : "
					+ Thread.currentThread().getPriority());
	}
}

package com.jspiders.multithreading.main;

import com.jspiders.multithreading.thread.MyThread4;

public class ThreadMain4 {

	public static void main(String[] args) {

		MyThread4 myThread4 = new MyThread4();
		Thread thread = new Thread(myThread4);
		thread.setName("Thread4");
		thread.setPriority(2);
		thread.start();

	}
}
```

- **CurrentThread()**
  - This is a static method present in a Thread class
  - It is used to return the reference of the currently executing Thread Object

```java
package com.jspiders.multithreading.thread;

public class MyThread5 extends Thread {

	@Override
	public void run() {
		for(int i = 1; i <= 5; i++) {
			if (i == 3) {
				stop();
			}
			System.out.println(getName() + " is now running");
		}
	}

}
package com.jspiders.multithreading.main;

import com.jspiders.multithreading.thread.MyThread5;

public class ThreadMain5 {

	public static void main(String[] args) {
		MyThread5 myThread5 = new MyThread5();
		myThread5.setName("5");
		myThread5.start();
	}
}
```

- **stop()**
  - it is a nonstatic method present in a thread class
  - it is used to stop the execution of the currently executing thread forcefully
  - It is a deprecated method which is already marked for removal
- Deprecated - No Longer In Use

```java
package com.jspiders.multithreading.resource;

public class Account {

	private int balance;

	public Account(int balance) {
		this.balance = balance;
	}

	public int checkBalance() {
		return balance;
	}

	public synchronized void deposit(int amount) {
		System.out.println("Trying to deposit " + amount + " Rs. ");
		balance += amount;
		System.out.println("Deposit successfull. ");
		System.out.println("Account balance : " + checkBalance());
	}

	public synchronized void withdraw(int amount) {
		System.out.println("Trying to withdraw " + amount + " Rs. ");
		if (balance >= amount) {
			balance -= amount;
			System.out.println("Withdraw successfull. ");
			System.out.println("Account balance : " + checkBalance());
		} else {
			System.out.println("Insufficient balance. ");
		}
	}
}
```

```java
package com.jspiders.multithreading.thread;

import com.jspiders.multithreading.resource.Account;

public class Husband extends Thread {

	private Account account;

	public Husband(Account account) {
		this.account = account;
	}

	@Override
	public void run() {
		account.deposit(5000);
		account.withdraw(2000);
	}

}
```

```java
package com.jspiders.multithreading.thread;

import com.jspiders.multithreading.resource.Account;

public class Wife extends Thread {

	private Account account;

	public Wife(Account account) {
		this.account = account;
	}

	@Override
	public void run() {
		account.deposit(2000);
		account.withdraw(5000);
	}
}
```

```java
package com.jspiders.multithreading.main;

import com.jspiders.multithreading.resource.Account;
import com.jspiders.multithreading.thread.Husband;
import com.jspiders.multithreading.thread.Wife;

public class AccountMain {

	public static void main(String[] args) {
		Account account = new Account(10000);
		Husband husband = new Husband(account);
		Wife wife = new Wife(account);

		husband.start();
		wife.start();
	}
}
```

Expected Output	Actual Output
Trying to deposit 5000 Rs. Deposit successfull. Account balance : 15000 Trying to withdraw 2000 Rs. Withdraw successfull. Account balance : 13000 Trying to deposit 2000 Rs. Deposit successfull. Account balance : 15000 Trying to withdraw 5000 Rs. Withdraw successfull. Account balance : 10000	Trying to deposit 5000 Rs. Deposit successfull. Account balance : 15000 Trying to deposit 2000 Rs. Deposit successfull. Account balance : 17000 Trying to withdraw 5000 Rs. Withdraw successfull. Account balance : 12000 Trying to withdraw 2000 Rs. Withdraw successfull. Account balance : 10000

- **Shared Resource**
  - Resources which is access by more than one Thread Simultaneously Is called as a shared resource
  - A resource can be variable, a method, an object, a class or even memory
  - If resource is static member else if it is a nonstatic member then it is known as nonstatic resource
- **Data inconstancy in multithreading**
  - When multiple threads are operating each thread are allocated
  - It this threads are unaware of the operation performed on shared resource by another thread
  - Hence Thread operated on resource, independently which leads to ‘data inconstancy’
  - The data in consistency caused due to multithreading can be avoid with help of ‘Synchronization’

Assignment 1

Write the details of all the methods used in multithreading in the below format

Method 1 .

- **Method Name** : start()
- **class** : Thread Class
- **method signature** : **public void start()**
- **return type** : void
- **modifier** :
- **Access modifier** : public
- **Exception** : IlligalThreadStateException()

Method 2 .

- **Method Name** : run()
- **interface** : Runnable
- **method signature** : **void run()**
- **return type** : void
- **modifier** :
- **Access modifier** : default
- **Exception** :

Method 3 .

- **Method Name** : getName()
- **class** : Thread
- **method signature** : **public final String getName()**
- **return type** : String
- **modifier** : final
- **Access modifier** : public
- **Exception** :

Method 4 .

- **Method Name** : setName()
- **Class** : Thread
- **method signature** : **public final synchronized void setName(String Name)**
- **return type** : void
- **modifier** : final , synchronized
- **Access modifier** : public
- **Exception** : NullPointerException throw

Method 5 .

- **Method Name** : getPriority()
- **Class** : Thread

- **method signature : public final int getPriority()**
- **return type** : int
- **modifier** : final
- **Access modifier** : public
- **Exception** :

Method 6.

- **Method Name** : setPriority()
- **class/interface** : thread
- **method signature** : **public final void setPriority(int newPriority)**
- **return type** : void
- **modifier** : final
- **Access modifier** : public
- **Exception** : **IllegalArgumentException**

Method 7.

- **Method Name** : currentThread()
- **class/interface** : Thread
- **method signature** : **public static native Thread currentThread()**
- **return type** : Thread
- **modifier** : static , native
- **Access modifier** : **public**
- **Exception** :

Method 8.

- **Method Name** : stop()
- **class/interface** : Thread
- **method signature** : **public final void stop()**
- **return type** : void
- **modifier** : final
- **Access modifier** : **public**
- **Exception** : UnsupportedOperationException

Method 9.

- **Method Name** : wait()
- **Class** : Object
- **method signature** : **public final void wait()**
- **return type** : void
- **modifier** : final
- **Access modifier** : **public**
- **Exception** : InterruptedException

Method 10.

- **Method Name** : notify()

- **class/interface** : Object
- **method signature** : **public final native void notify()**
- **return type** : void
- **modifier** : static , native
- **Access modifier** : **public**
- **Exception** :

Method 11.

- **Method Name** : notifyAll()
- **class/interface** : Object
- **method signature** : **public final native void notifyAll()**
- **return type** : native
- **modifier** : final , native
- **Access modifier** : **public**
- **Exception** :

Method 12.

- **Method Name** : sleep()
- **Class** : Thread
- **method signature** : **public static void sleep(long millis)**
- **return type** : void
- **modifier** : static
- **Access modifier** : **public**
- **Exception** : **InterruptedException**

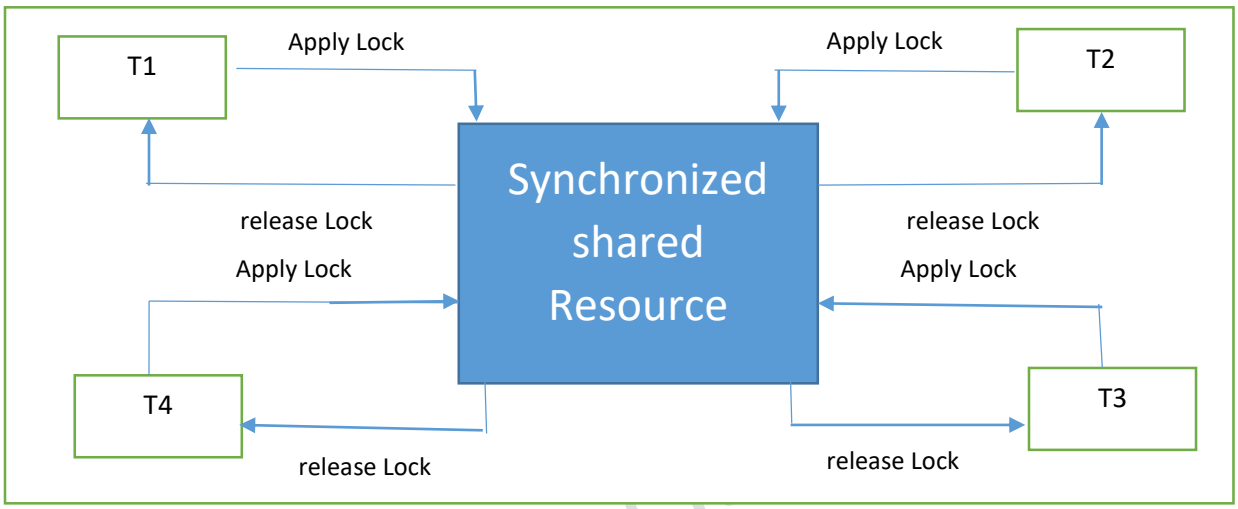
Assignment No. 2

Write a difference between notify () and notifyAll().

	Notify()	**notifyAll()**
1.	Wake up single Thread	Wake up all Thread

Synchronization

- it is a process which is used to avoid the data inconstancy caused due to multithreading while operating a shared resource
- synchronization is implement with the help of synchronized keyword
- when a shared resource is made synchronized then whenever a thread gets access that resource then it applies lock on resource
- it is means if shared resources is synchronized then only one thread can access at the time
- all the other thread that access the resources until lock on that resource released



- **Lock in multithreading**
  - in multithreading there are two types of locks
    - class Lock
    - object lock
- **class Lock**
  - if synchronized shared resource is static member then lock applied on it will be class lock
- **Object Lock**
  - If synchronized shared resource is non static member, the lock applied on it will be object lock

- To implement synchronization in multithreading: -

```
package com.jspiders.multithreading.resource;

public class Account {

	private int balance;

	public Account(int balance) {
		this.balance = balance;
	}

	public int checkBalance() {
		return balance;
	}

	public synchronized void deposit(int amount) {
		System.out.println("Trying to deposit " + amount + " Rs. ");
		balance += amount;
		System.out.println("Deposit successfull. ");
		System.out.println("Account balance : " + checkBalance());
	}

	public synchronized void withdraw(int amount) {
		System.out.println("Trying to withdraw " + amount + " Rs. ");
		if (balance >= amount) {
			balance -= amount;
			System.out.println("Withdraw successfull. ");
			System.out.println("Account balance : " + checkBalance());
		} else {
			System.out.println("Insufficient balance. ");
		}
	}
}
```

- In this case as synchronized keyword is applied on the shared resources that is deposit & withdraw method, it can be access either by husband or wife thread at a time
- Hence if one thread is already operation on the shared resource then other thread cannot operate unit first thread done executing
- This will help as to achieved consistency of find output

Wait()

- It is nonstatic methd which is used to put the current executing thread from the running phase to the waiting phase forcefully
- The thread whose execution is forcefully calls cannot resume it execution automatically

notify()

- it is a nonstatic method which is used to call a waiting phase to running phase
- this method is capable to resolve only one waiting thread

notifyAll()

- it is similar to notify method but it is capable to remove the execution of all the waiting thread

sleep()

- It is a static method which is used to pause the execution of the currently executing thread for a specific time interval
- Unlike the wait method, the sleep method does not need the notify method call
- It means if a thread is pause with the help of sleep method then it will then resume its execution automatically after the define time period is completed
- This method is an overloaded method
- The most commonly use method signature is as follows

  sleep(long millisecond)

```
package com.jspiders.multithreading.main;

public class sleepDemo {
	public static void main(String[] args) {
		String msg = "This is the sleep method";
		for (int i = 0; i < msg.length(); i++) {
			System.out.println(msg.charAt(i));
			try {
				Thread.sleep(1000);
			} catch (InterruptedException e) {
				// TODO Auto-generated catch block
				e.printStackTrace();
			}
		}
	}
}
```

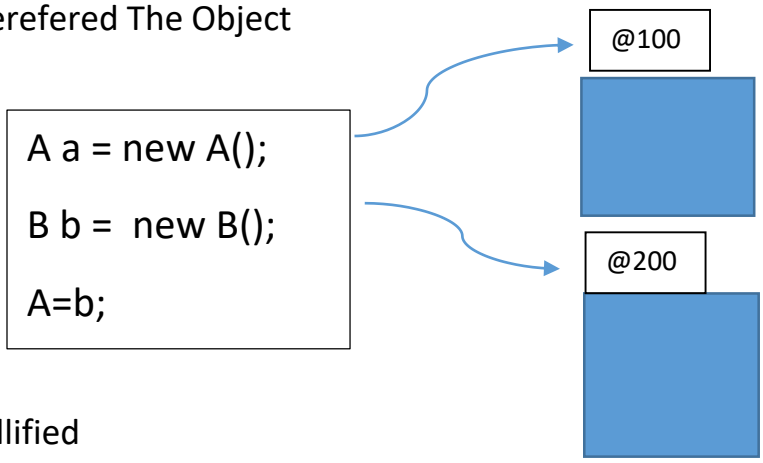
Daemon Threads

- In java there are two types of threads that is user defined thread and demon thread
- The user define thread are the treads which are created by the programmer with the help of the Thread class or Runnable interface
- Such user defines Threads have to be called for execution explicitly by using start method
- On the other hand, Demon Threads are predefined in Java and are called for execution implicitly by JVM whenever required
- It means The user is not responsible to call daemon Thread for execution and also its execution cannot be see
- Note - > The most important Daemon Thread in java is ‘Garbage Collection’

Garbage Collector

- The Garbage Collection Demean Threads Helps Java to Manage the Memory
- The Garbage Collection Demean is responsible to remove the unusable objects from the memory
- It is implicitly called by JVM in two Cases That is
  1. When is Object is deferred
  2. When an Object is Nullified

1.derefered The Object



2. Nullified

```
A a= new A();
a = null;
```

Advantages of Garbage Collection

- It makes JAVA efficient in memory management
- So now we as programmer do not need to worry about the memory management in java

Dead Lock

- When multiple thread is executed on shared resource and Their execution is intern dependent and also each Thread has acquired (applied) lock on resource which is required by another thread and wise versa then none of the Thread will be capable Of releasing The Lock from the resource and hence it cannot be given to any Other Thread
- In Such a situation none of the thread will be able to complete the exception
- This is Known as The Dead Lock Situation

Synchronized Block

- Synchronized block allows us to make all the resources (variable, object, classes) as synchronized by default
- It is block of code which is having the keyword synchronized
- Syntax of Synchronized Block Is as follows

```
Synchronized
{
……..
}
```

```java
package com.jspiders.multithreading.resources;

public class Resourse {
	public String res1 =  "1st resource";
	public String res2 = "2nd resource";
}
```

```java
package com.jspiders.multithreading.thread;

import com.jspiders.multithreading.resources.Resourse;

public class Thread1 extends Thread {
	Resourse resource;

	public Thread1(Resourse resource) {
		this.resource = resource;
	}

	@Override
	public void run() {
		synchronized (resource.res1) {
			System.out.println(getName()+ " applied Lock on "+ resource.res1);
			synchronized (resource.res2) {
				System.out.println(getName()+ " applied Lock on "+ resource.res2);
			}
			System.out.println(getName()+" released "+ resource.res1);
		}
		System.out.println(getName()+" released "+ resource.res2);
	}

}
```

```java
package com.jspiders.multithreading.thread;

import com.jspiders.multithreading.resources.Resourse;

public class Thread2 extends Thread{
	Resourse resource;

	public Thread2(Resourse resource) {
		this.resource = resource;
	}

	@Override
	public void run() {
		synchronized (resource.res2) {
			System.out.println(getName()+ " applied Lock on "+ resource.res1);
			synchronized (resource.res1) {
				System.out.println(getName()+ " applied Lock on "+
resource.res2);
			}
			System.out.println(getName()+" released "+ resource.res1);
		}
		System.out.println(getName()+" released "+ resource.res2);
	}

}
```

notifyAll()

```java
package com.jspiders.multithreading.thread;

import com.jspiders.multithreading.resources.Cake;

public class CakeMaker extends Thread {

	Cake cake;

	public CakeMaker(Cake cake) {
		super();
		this.cake = cake;
	}
	public void run() {

		cake.bakeCakes(5);
	}
}
```

```java
package com.jspiders.multithreading.main;

import com.jspiders.multithreading.resources.Resourse;
import com.jspiders.multithreading.thread.Thread1;
import com.jspiders.multithreading.thread.Thread2;

public class ThreadMain4 {

	public static void main(String[] args) {

		Resourse resource = new Resourse();

		Thread1 thread1 = new Thread1(resource);
		Thread2 thread2 = new Thread2(resource);

		thread1.setName("Thread 1");
		thread2.setName("Thread 2");

		thread1.getName();
		thread1.getPriority();
		thread1.setPriority(7);

		thread1.start();
		thread2.start();
		thread2.notify();

	}

}
```

```java
package com.jspiders.multithreading.resources;

public class Cake {
	private int avialableCakes;

	public synchronized void orderCake(int orderedcakes) {

		System.out.println("Trying to order " + orderedcakes + " Cakes. ");

		if (avialableCakes < orderedcakes) {

			System.out.println("Cakes are not available Could you please
wait for some time..!");

			bakeCakes(orderedcakes- avialableCakes);
			try {
				this.wait();
			} catch (InterruptedException e) {
				// TODO Auto-generated catch block
				e.printStackTrace();
			}
		}
		avialableCakes -= orderedcakes;
		System.out.println(orderedcakes+" Ordered Cakes Delivered..!");

	}
	public synchronized void bakeCakes(int bakedcakes) {

		System.out.println("Trying to Bake " + bakedcakes + " Cakes.");
		avialableCakes += bakedcakes;
		System.out.println("Available Cakes " + avialableCakes + ".");

		this.notifyAll();
	}

}
```

```java
package com.jspiders.multithreading.main;

import com.jspiders.multithreading.resources.Cake;
import com.jspiders.multithreading.thread.Friend3;
import com.jspiders.multithreading.thread.Friend4;

public class CakeShop {

	public static void main(String[] args) {

		Cake cake=new Cake();

		Friend3 friend3=new Friend3(cake);
		Friend4 friend4=new Friend4(cake);

		friend3.start();
		friend4.start();
	}
}
```

```java
package com.jspiders.multithreading.thread;

import com.jspiders.multithreading.resources.Cake;

public class Friend4 extends Thread {

	Cake cake;

	public Friend4(Cake cake) {
		super();
		this.cake = cake;
	}

	@Override
	public void run() {

		cake.orderCake(10);
	}
}
```

```java
package com.jspiders.multithreading.thread;

import com.jspiders.multithreading.resources.Cake;

public class Friend3 extends Thread {

	private Cake cake;

	public Friend3(Cake cake) {
		super();
		this.cake = cake;
	}
	@Override
	public void run() {

		cake.orderCake(5);
	}
}
```

File Handling

- File is an entity were we can store the data as well as we can manipulate the data
- If we are writing some data to the file in that case that file will be considered as Target entity
- If we are reading some data from the file in that case that file will be considered as a source entity

Note –

Performing operation on file (CRUD operation) is called as File Handling

Operation on a File

1. Create a file
2. Fetch(get) the information about the file
3. Delete a File
4. Write data to a file
5. Read data from the file

File Class

- File is a concrete class present in java.io package
- We need to create an Object for the File Class
- In Order to perform operations on a file
- This file Class Contains overloaded constructor. Generally, Use constructor is as below

```
File (String path_name)
{

}
```

Path Name

- The value for the path name given in two ways
  1. By mentioning file name along with its extension in this case the path will be project folder This is also called as **default path** name
  2. A user can provide a folder path along with a file name and extension this path is called as an **absolute path**

createNewFile()

- It is a nonstatic method present inside the file class
- It is used to create a new file in the specified folder

```java
package com.jspider.filehandling.operations;

import java.io.File;
import java.io.IOException;

public class CreateFileDemo1 {
	public static void main(String[] args) {
		File file  = new File("Demo.txt");
		if (file.exists()) {
			System.out.println("File is Already Created ");
		} else {
			try {
				file.createNewFile();
				System.out.println("File is created");
			} catch (IOException e) {

				System.out.println("File is Not created");
			}
		}
		File file2 = new File("D:/WEJA2/filehandling/demo2.txt");
		try {
			file2.createNewFile();
			System.out.println("File is created");
		} catch (IOException e) {
			System.out.println("File is Not created");
		}
	}
}
```

exists()

- It is a nonstatic method present inside the file class
- It is used to check the whether the specified file is present or not
- If file is present it returns True otherwise it returns falls

```
package com.jspider.filehandling.operations;

import java.io.File;
import java.io.IOException;

public class CreateFileDemo1 {
	public static void main(String[] args) {
		File file  = new File("Demo.txt");
		if (file.exists()) {
			System.out.println("File is Already Created ");
		} else {

			try {
				file.createNewFile();
				System.out.println("File is created");
			} catch (IOException e) {

				System.out.println("File is Not created");
			}

		}
		File file2 = new File("D:/WEJA2/filehandling/demo2.txt");

		try {
			file2.createNewFile();
			System.out.println("File is created");
		} catch (IOException e) {

			System.out.println("File is Not created");
		}

	}
}
```

Delete File by Java Program

```
package com.jspider.filehandling.operations;

import java.io.File;

public class DeleteFile {

	public static void main(String[] args) {

		File file = new File("Demo.txt");

		if (file.exists()) {
			file.delete();
			System.out.println("File deleted");
		} else {
			System.out.println("File Does NOt Exist");
		}
	}

`
```

Fileinfo

```
package com.jspider.filehandling.operations;

import java.io.File;

public class FileInfo {

	public static void main(String[] args) {
		File file = new File("Demo.txt");

		System.out.println(file.getName());
		System.out.println(file.getAbsolutePath());
		System.out.println(file.length());

		if (file.canExecute()) {
			System.out.println("File Will Be Executable");
		} else {
			System.out.println("File not Executable");
		}
		if (file.canRead()) {
			System.out.println("File will Readable");
		} else {
			System.out.println("File not Readable");
		}
		if (file.canWrite()) {
			System.out.println("File Can Writable");
		} else {
			System.out.println("File not Writable");
		}
	}
}
```

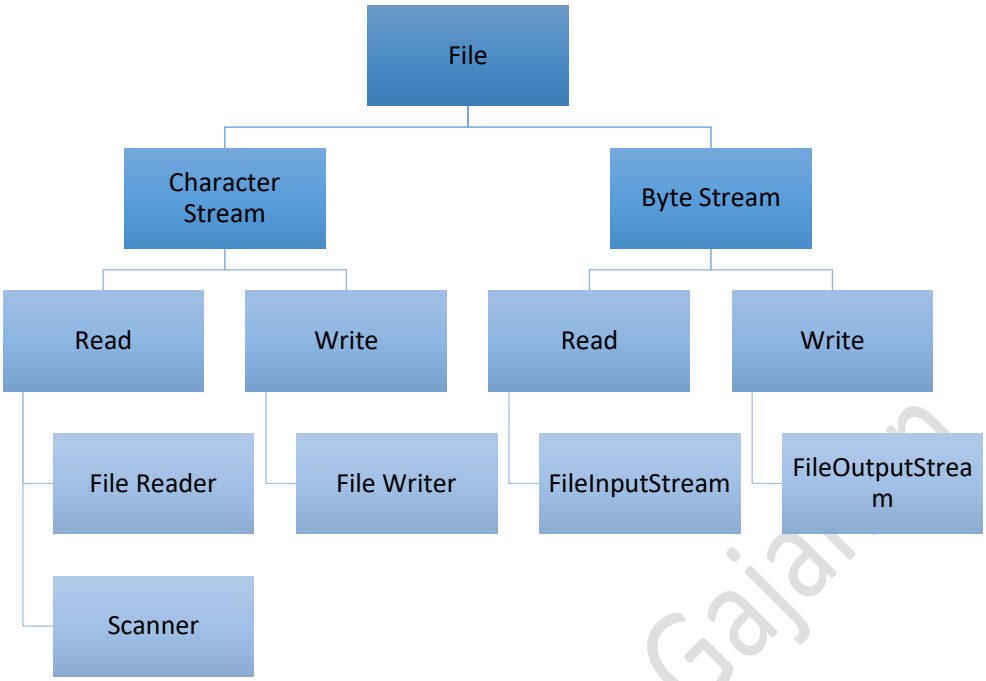
Reading and Writing Operation On file

- Reading And Writing operation on file depend on the type of data that particular file handling
- The data is categorized in two types
    1. Use Character stream data
    2. Byte Stream Data

    To perform read operation on files which handles character stream data we use reader classes to perform write operation which handles character stream data we use writer classes

    TO Read Operation On a file which handles byte stream data we use input stream classes

To perform write operation on file which handles byte stream data we use FileOutputStream



Write File of Char Stream

```
package com.jspider.filehandling.operations;

import java.io.File;
import java.io.FileWriter;
import java.io.IOException;

public class CharStreamWrite {
	public static void main(String[] args) throws IOException {
		File file = new File("Demo.txt");
		if (file.exists()) {
			FileWriter fileWriter = new FileWriter(file);
			fileWriter.write("File Write By Java Program");
			System.out.println("File Write Successfully");
			fileWriter.close();
		} else {
			file.createNewFile();
			System.out.println("New File Will Be Created");
			FileWriter fileWriter = new FileWriter(file);
			fileWriter.write("File Write By Java Program");
			System.out.println("File Write Successfully");
			fileWriter.close();
		}
	}
}
```

Byte Stream Data Write

```
package com.jspider.filehandling.operations;

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

public class ByteStreamWrite {
	public static void main(String[] args) throws IOException {
		File file = new File("Demo.txt");
		if (file.exists()) {
			FileOutputStream fileOutputStream = new FileOutputStream(file);
			fileOutputStream.write(5454);
			System.out.println("Data Written to the File");
			fileOutputStream.close();
		} else {
			file.createNewFile();
			System.out.println("New File Created");
			FileOutputStream fileOutputStream = new FileOutputStream(file);
			fileOutputStream.write(5454);
			System.out.println("Data Written to the File");
			fileOutputStream.close();
		}
	}
}
```

Reading The Data Using FIleReader & Scanner

```
package com.jspider.filehandling.operations;

import java.io.File;
import java.io.FileReader;
import java.io.IOException;
import java.util.Scanner;
public class CharStreamRead {
	public static void main(String[] args) throws IOException {
		File file = new File("Demo.txt");
		if (file.exists()) {
			FileReader fileReader = new FileReader(file);
			System.out.println(fileReader.read());
			System.out.println("Data fetched From the file Filereader");
			fileReader.close();

			Scanner scanner =  new Scanner(file);
			while (scanner.hasNextLine()) {
				System.out.println(scanner.nextLine());
			}
			System.out.println("Data fetched from the Scanner");
			scanner.close();

		} else {
			System.out.println("File Not FOund");
		}

	}
```

ByteStreamRead

```
package com.jspider.filehandling.operations;

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

public class ByteStreamRead {

	public static void main(String[] args) throws IOException {

		File file = new File("Demo.txt");

		if (file.exists()) {
			FileInputStream fileInputStream = new FileInputStream(file);
			System.out.println(fileInputStream.read());
			System.out.println("Data fetched from the file");
			fileInputStream.close();
		} else {
			System.out.println("File Not Found");
		}
	}

}
```

SERIALIZATION AND DESERIALIZATION

- Serialization
  - It is the process of converting java object into byte stream format
  - To perform Serialization, we use writeObjectMethod()
- Deserialization
  - It is a process of converting byte stream data into java Object
  - For deserialization we use read object method of ObjectInputStream Class

Serializable

- Serializable is a marker interface
- The class whose object undergoes serialization and deserialization process
- That class should implement serializable interface

Marker Interface

- Marker interface is an empty interface
- It used to mark particular class for specific task
- It provides special abilities to a class
  (write marker interface present in java and use)

- java.io.Serializable:

  - This marker interface was used to indicate that a class could be serialized, meaning its object instances could be converted into a byte stream for storage or transmission and then reconstructed later.
- java.util.EventListener:
  - This marker interface was used to identify interfaces that defined event listener types. Implementing this interface indicated that a class could receive events generated by event sources.
- java.rmi.Remote:
  - This marker interface was used in the Java Remote Method Invocation (RMI) framework to indicate that a remote object could be invoked from a different Java Virtual Machine (JVM) over a network.

- To perform **serialization**, we need to have objects of File Output Stream and Object output stream Classes
- To perform **deserialization**, we need to have objects of File Input Stream and object input stream classes

```java
package com.jspider.serializationAndDeserialization;

import java.io.Serializable;

public class Student implements Serializable{

	private static final long serialVersionUID = 1L;
	private int Id;
	private String Name;
	private String Email;
	private int age;

	public Student(int id, String name, String email, int age) {
		super();
		Id = id;
		Name = name;
		Email = email;
		this.age = age;
	}
	@Override
	public String toString() {
		return "Student [Id=" + Id + ", Name=" + Name + ", Email=" + Email + ",
age=" + age + "]";
	}
}
package com.jspider.serializationAndDeserialization;

import java.io.File;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.ObjectOutputStream;

public class Serialization {

	public static void main(String[] args) throws IOException {
		File file = new File("Student.txt");
		if (file.exists()) {
			FileOutputStream fileOutputStream = new FileOutputStream(file);
			ObjectOutputStream objectOutputStream = new
ObjectOutputStream(fileOutputStream);

			objectOutputStream.writeObject(new Student(1, "Laxman",
"Laxman@gmail.com", 24));
			System.out.println("Object Written Successfully");
			fileOutputStream.close();
			objectOutputStream.close();

		} else {
			file.createNewFile();
			FileOutputStream fileOutputStream = new FileOutputStream(file);
			ObjectOutputStream objectOutputStream = new
ObjectOutputStream(fileOutputStream);
			objectOutputStream.writeObject(new Student(1, "Laxman",
"Laxman@gmail.com", 24));
			System.out.println("Object Written Successfully");
			fileOutputStream.close();
			objectOutputStream.close();
		}
	}
}
```

```java
package com.jspider.serializationAndDeserialization;

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

public class Deserialization {

	public static void main(String[] args) throws IOException,
ClassNotFoundException {

		File file = new File("Student.txt");

		if (file.exists()) {

			FileInputStream fileInputStream = new FileInputStream(file);
			ObjectInputStream objectInputStream = new
ObjectInputStream(fileInputStream);
			Student student = (Student) objectInputStream.readObject();
			System.out.println(student);
			System.out.println("Object is fetched from the file");

			objectInputStream.close();
			fileInputStream.close();

		}
		else {
			System.out.println("File is not Found ");
		}
	}
```


Assignment

Method 1.

- **Method Name** : createNewFile()
- **Class** : File
- **method signature** : **public Boolean createNewFile()**
- **return type** : boolean
- **modifier** : non static
- **Access modifier** : **public**
- **Exception** : **IoException**

Method 2.

- **Method Name** : exists()
- **Class** : File
- **method signature** : **public Boolean exists()**
- **return type** : boolean

- **modifier** : non static
- **Access modifier** : **public**

Method 3.

- **Method Name** : delete()
- **Class** : File
- **method signature** : **public Boolean delete()**
- **return type** : boolean
- **modifier** : non static
- **Access modifier** : **public**

Method 4.

- **Method Name** : getName()
- **Class** : File
- **method signature** : **public String getName()**
- **return type** : String
- **modifier** : non static
- **Access modifier** : **public**

Method 5.

- **Method Name** : getAbsulatePath()
- **Class** : File
- **method signature** : **public String getAbsulatePath()**
- **return type** : String
- **modifier** : non static
- **Access modifier** : **public**

Method 6.

- **Method Name** : length()
- **Class** : File
- **method signature** : **public long length()**
- **return type** : long
- **modifier** : non static
- **Access modifier** : **public**

Method 7.

- **Method Name** : canExecute()
- **Class** : File
- **method signature** : **public boolean canExecute()**
- **return type** : boolean
- **modifier** : non static

- **Access modifier : public**

Method 8.

- **Method Name : canRead()**
- **Class : File**
- **method signature : public boolean canRead()**
- **return type : boolean**
- **modifier : non static**
- **Access modifier : public**

Method 9.

- **Method Name : canWrite()**
- **Class : File**
- **method signature : public boolean canWrite()**
- **return type : boolean**
- **modifier : non static**
- **Access modifier : public**

Method 10.

- **Method Name : write()**
- **Class : Writer**
- **method signature : public void write(String str)**
- **return type : void**
- **modifier : non static**
- **Access modifier : public**
- **Exception : IoException**

Method 11.

- **Method Name : close()**
- **Class : OutputStream**
- **method signature : public void close()**
- **return type : void**
- **modifier : non static**
- **Access modifier : public**
- **Exception : IoException**

Method 12.

- **Method Name : read()**
- **Class : File**
- **method signature : public int read()**
- **return type : int**
- **modifier : non static**
- **Access modifier : public**
- **Exception : IoException**

Method 13.

- **Method Name** : hasNextLine()
- **Class** : Scanner
- **method signature** : **public Boolean hasNextLine()**
- **return type** : Boolean
- **modifier** : non static
- **Access modifier** : **public**
- **Exception** :

Method 14.

- **Method Name** : next()
- **Class** : Scanner
- **method signature** : **public String next()**
- **return type** : String
- **modifier** : non static
- **Access modifier** : **public**
- **Exception** :

Method 15.

- **Method Name** : writeObject(Object obj)
- **Class** : ObjectOutputStream
- **method signature** : **public final void writeObject(Object obj)**
- **return type** : void
- **modifier** : final
- **Access modifier** : **public**
- **Exception** : **IoException**

Method 16.

- **Method Name** : readObject()
- **Class** : ObjectInputStream
- **method signature** : **public final Object readObject()**
- **return type** : Object
- **modifier** : final
- **Access modifier** : **public**
- **Exception** : **IoException, ClassNotFoundException**

Design Patterns

- Design a patterns are the predefined structures which can be used to avoid recurring development issues
- Using Design patterns, we can avoid the development issues but we unable to solve development issues

- Based on the type of issue that the design patterns are dealing with they are classified into two types
    1. **Creational Design Patterns**
    2. **Structural Design Patterns**

1.Creational Design Patterns

- The design patterns which deal with the development issues associated with object creation
- Widely used creational design patterns are as below
    - Singleton design pattern
    - Factory Design Pattern
    - Builder design pattern

2.Structural Design Patterns

- The design patterns which deal with the development issues associated with structures of classes and interfaces
- Widely use structured design pattern is as below
    - Adopter Design Pattern

Singleton design pattern

- Singleton design pattern using this design pattern we can restrict user to create multiple objects for single class
- In singleton design pattern we are going to make the constructor of a particular class as private
- In order to access to private constructor, we define helper method
- Singleton design pattern classified into 2 type:
    - Lazy instantiation
        - In lazy instantiation we will declare object reference variable of a particular class
        - But we don't initial it
        - We will initialize that object reference variable whenever user will request for a object of a class
    - Eager instantiation
        - In eager instantiation we will declare Object reference variable of a particular class
        - And we initialize it so by doing this we are keeping an object of a class ready and
        - we will return the same object to the user whenever requested
        -

Factory Design Pattern

- This design pattern is us
- ed to create objects whenever user will request for an object
- If an application contains several classes and the object for all the classes need to be created. But there is a chance that sum of the created Objects will be left unused This leads to wastage of memory
- This problem can be avoided by using **Factory Design Pattern**
- With the help of its factory design patterns we can create objects during runtime (During program execution)

Factory Design Pattern

```
package com.jspider.designpatterns.creational;
public class NormalTea implements Beverage {

	@Override
	public void order() {

		System.out.println("Normal tea is Ordered ");
	}
}
package com.jspider.designpatterns.creational;
public class GreenTea implements Beverage{

	@Override
	public void order() {
		System.out.println("Green Tea is Ordered");
	}
}
package com.jspider.designpatterns.creational;
public class BlackTea implements Beverage{

	@Override
	public void order() {
		// TODO Auto-generated method stub
		System.out.println("Black Tea is Ordered ");
	}
}
package com.jspider.designpatterns.creational;

public class IceTea implements Beverage{

	@Override
	public void order() {
		// TODO Auto-generated method stub
		System.out.println("Ice Tea is Ordered. ");
	}
}

}
```

SingletonEgger instantiation

```
package com.jspider.designpatterns.creational;

public class SingletonEgger {

	private static SingletonEgger singletonEgger = new SingletonEgger();

	private SingletonEgger()
	{

	}
	public static SingletonEgger getObject()
	{
		return singletonEgger;
	}

}
```

```
package com.jspider.designpatterns.Main;

import com.jspider.designpatterns.creational.SingletonEgger;

public class SingletonEgerMain {

	public static void main(String[] args) {

		SingletonEgger singletonEgger = SingletonEgger.getObject();

		System.out.println(singletonEgger);

		SingletonEgger singletonEgger2 = SingletonEgger.getObject();

		System.out.println(singletonEgger2);

		SingletonEgger singletonEgger3 = SingletonEgger.getObject();

		System.out.println(singletonEgger3);
	}

}
```

Lazy instantiation

```
package com.jspider.designpatterns.creational;

public class SingletonLazy {

	private static SingletonLazy singletonLazy;

	private SingletonLazy() {
		System.out.println("Constructor accessed ! ");
	}

	public static SingletonLazy getObject() {
		if (singletonLazy == null) {
			singletonLazy = new SingletonLazy();
			}
		return singletonLazy;
	}
}
```

```
package com.jspider.designpatterns.Main;

import com.jspider.designpatterns.creational.SingletonLazy;

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

		SingletonLazy singletonLazy  =  SingletonLazy.getObject();
		System.out.println(singletonLazy);

		SingletonLazy singletonLazy1  =  SingletonLazy.getObject();
		System.out.println(singletonLazy1);

		SingletonLazy singletonLazy2  =  SingletonLazy.getObject();
		System.out.println(singletonLazy2);
	}
}
```

```java
package com.jspider.designpatterns.creational;

public interface Beverage {

	void order();
}
```

```java
package com.jspider.designpatterns.Main;

import java.util.Scanner;

import com.jspider.designpatterns.creational.Beverage;
import com.jspider.designpatterns.creational.BlackTea;
import com.jspider.designpatterns.creational.GreenTea;
import com.jspider.designpatterns.creational.IceTea;
import com.jspider.designpatterns.creational.NormalTea;

public class TeaFactory {

	private static Beverage beverage;

	public static void main(String[] args) {
		Factory().order();
	}

	public static Beverage Factory() {
		System.out.println("Select tea to Order : ");
		System.out.println("1.Normal tea\n"+"2.Black tea\n"+"3.Green
Tea\n"+"4.Ice Tea");

		Scanner scanner = new Scanner(System.in);
		int choice = scanner.nextInt();

		switch (choice) {
		case 1:{
			beverage =  new NormalTea();
			break;
		}
		case 2:{
			beverage =  new BlackTea();
			break;
		}
		case 3:{
			beverage =  new GreenTea();
			break;
		}
		case 4:{
			beverage =  new IceTea();
			break;
		}
		default:
			System.out.println("Invalid Choice Try Again !");
			Factory();
		}
		scanner.close();
		return beverage;

	}
```

Builder Design Pattern

- It used to create complex Objects If an application contains classes with too many properties which are required to be initialized in the expected sequence which looks impossible to perform this problem can be solved using builder design pattern
- In this builder design pattern, we need to take the help of another class called as builder class which has access to the contractor of complex class which is aware of all the properties of complex object

```
package com.jspider.designpatterns.creational;

public class Contact {

	private String firstName;
	private String middleName;
	private String email;
	private long mobileNumber;
	private long landlineNumber;
	private String adress;
	private String gender;
	private int age;
	private String dateOfBirth;

	public Contact(String firstName, String middleName, String email, long
mobileNumber, long landlineNumber,
			String adress, String gender, int age, String dateOfBirth) {
		super();
		this.firstName = firstName;
		this.middleName = middleName;
		this.email = email;
		this.mobileNumber = mobileNumber;
		this.landlineNumber = landlineNumber;
		this.adress = adress;
		this.gender = gender;
		this.age = age;
		this.dateOfBirth = dateOfBirth;
	}

	@Override
	public String toString() {
		return "Contact [firstName=" + firstName + ", middleName=" + middleName
+ ", email=" + email
				+ ", mobileNUumber=" + mobileNumber + ", landlineNUmber=" +
landlineNumber + ", adress=" + adress
				+ ", gender=" + gender + ", age=" + age + ", dateOfBirth=" +
dateOfBirth + "]";
	}
```

```java
package com.jspider.designpatterns.creational;

public class ContactBuilder {

	private String firstName;
	private String middleName;
	private String email;
	private long mobileNumber;
	private long landlineNumber;
	private String adress;
	private String gender;
	private int age;
	private String dateOfBirth;
	public ContactBuilder firstName(String firstName) {
		this.firstName = firstName;
		return this;
	}
	public ContactBuilder middleName(String middleName) {
		this.middleName = middleName;
		return this;
	}
	public ContactBuilder email(String email) {
		this.email = email;
		return this;
	}
	public ContactBuilder mobileNumber(long mobileNumber) {
		this.mobileNumber = mobileNumber;
		return this;
	}
	public ContactBuilder landlineNumber(long landlineNumber) {
		this.landlineNumber = landlineNumber;
		return this;
	}
	public ContactBuilder gender(String gender) {
		this.gender = gender;
		return this;
	}
	public ContactBuilder adress(String adress) {
		this.adress = adress;
		return this;
	}
	public ContactBuilder age(int age) {
		this.age = age;
		return this;
	}
	public ContactBuilder dateOfBirth(String dateOfBirth) {
		this.dateOfBirth = dateOfBirth;
		return this;
	}
	public Contact getContact()
	{
		Contact contact = new Contact(firstName, middleName, email,
mobileNumber, landlineNumber, adress, gender, age, dateOfBirth);
		return contact;
	}
}
```

```java
package com.jspider.designpatterns.Main;

import com.jspider.designpatterns.creational.Contact;
import com.jspider.designpatterns.creational.ContactBuilder;

public class ContactMain {

	public static void main(String[] args) {

		Contact contact = new
ContactBuilder().firstName("Raju").gender("Male").age(40).getContact();

		System.out.println(contact);
	}
}
```

Adapter Design Pattern

- This design pattern is used to adopt properties of one entity and behaviors of another entity using one intermediate class called as adapter class so this adapter class is going to inherit the properties of one class and implement the behaviors of one interface

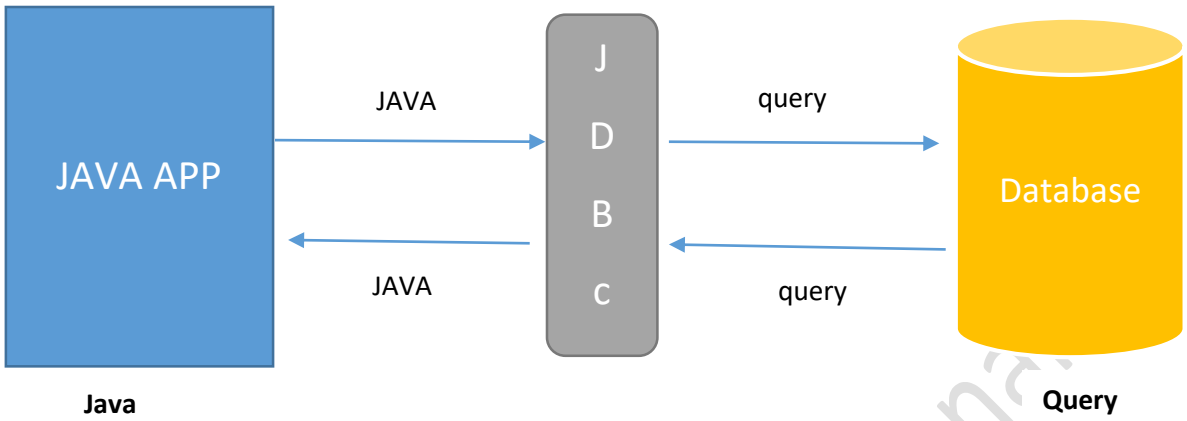

```java
package com.jspider.designpatterns.structural;

public class Employee {

	private int id;
	private String name;
	private String email;
	public int getId() {
		return id;
	}
	public void setId(int id) {
		this.id = id;
	}
	public String getName() {
		return name;
	}
	public void setName(String name) {
		this.name = name;
	}
	public String getEmail() {
		return email;
	}
	public void setEmail(String email) {
		this.email = email;
	}

}
```

```
package com.jspider.designpatterns.structural;

public interface Event {

	void womensDay(Adapter adapter);

	void mensDay(Adapter adapter);

}

package com.jspider.designpatterns.structural;
public class Adapter extends Employee implements Event{
	public void womensDay(Adapter adapter) {
		adapter.setId(1);
		adapter.setName("Radha");
		adapter.setEmail("radha@gmail.com");
		System.out.println("Chief guest for the womens day event is "
+adapter.getName());

	}

	public void mensDay(Adapter adapter) {
		adapter.setId(1);
		adapter.setName("Ramesh");
		adapter.setEmail("ramesh@gmail.com");
		System.out.println("Chief guest for the Mens Day event is "+
adapter.getName());

	}

}

package com.jspider.designpatterns.Main;

import com.jspider.designpatterns.structural.Adapter;

public class AdapterMain {

	public static void main(String[] args) {
		Adapter adapter = new Adapter();
		adapter.womensDay(adapter);
		adapter.mensDay(adapter);
	}
}
```

JDBC (java database connectivity**)**

- JDBC stands for java database connectivity
- It is used to establish the communication between the java application and the database application
- It is required because java application understands only java programming language whereas the database application understands only query language

- As both the applications are incompatible to understands each other, hence JDBC needs to add has mediator between them
- JDBC is capable of understanding and processing java language as well as query language

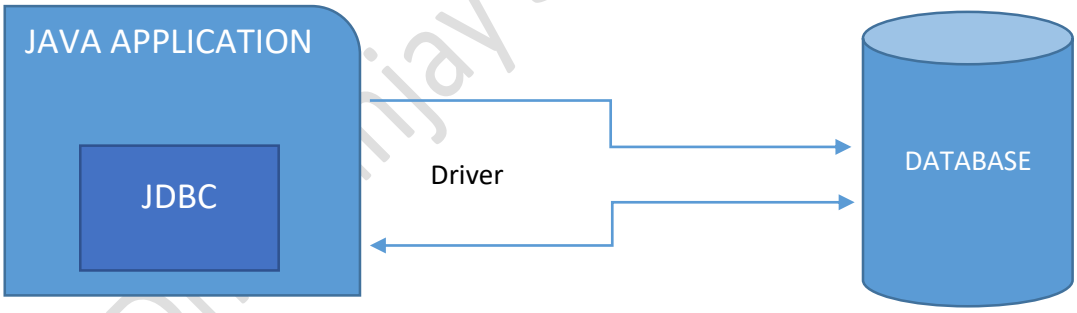


- Establishing communication and database is required in order to store the data from java application to the database in order to accessing in the feature

Note –

Any application or technology which helps us to establish communication between to different application is considered a type of **API** hens JDBC can be called as an API

There is No other technology other than JDBC which can helps u
s to interact between java application and database



- The JDBC logic is part of java application itself
- Hence in order to send and receive data between java application and database JDBC appoints the driver class
- The driver class is only responsible to carry the data given by JDBC to the database and the output given by the database back to JDBC
- Hence It can be said that the drover class provides a bridge of communication
  - **5 steps of JDBC**
    1. Load The driver class

2. Open Connection
3. Create / prepare statement
4. Process the result
5. Close connection

Prerequisites of JDBC

- In order to access the classes, the classes and interfaces required for JDBC operations, we need to add an external JAR file 'Mysql-connector/j'
- Follow the below steps to require the
  1. Go to google and search for 'Maven Repository'
  2. Go to the first link in the result
  3. In maven repository search for 'Mysql-conncetor/j'
  4. Select the version similar to the version installed in your system
  5. From the selected version, download the JAR file
  6. Go the project in eclipse and right click on the project name
  7. In the dropdown go to 'Build Path' and then go to '*Configure Build* Path'
  8. In the next window go to the '*libraries and click on add external jars button*'
  9. Open the downloaded jar file then click on and apply and close button
  10. After the add file jar file is added externally we can see a section

Note- Adding a jar file externally into a project, increases the load on the build path of that project. This results in delayed build process

1. Load The Driver Class
   - For Loading The Driver class, we need to take help of 'forName()'
   - It is static method from a predefined class in java named 'Class'
   - This method can be use to load any class with the help of its path
   - The path of the class is given as a string Argument to the method
     - Driver class Path
       - "Com.mysql.cj.jdbcDriver"
2. Open Connection
   - In order to open the connection between the java application and the database application, we need to take help of overloaded method called as 'getConnection()'
   - It is static method present in the class named 'DriverManager'
   - The conection can be establish in 3 diffferent ways
     i. getConnection(String url);
     ii. getConnection(String url, properties info);
     iii. getConnection(String url,String user, String password);
   - To establish the connection we need to provide the database url

- Database URL format

protocol : sub-protocol://host_name:port_no/db_name?user = username & password = password

1. Protocol
   - A protocol defines which technology is being use for the url
2. Sub protocol
   - The sub protocol is defines the technology for which the technology defiled in protocol is used
3. Host name
   - It defines that the database application is located in which system exactly
   - If the database application is present in the same system as that of the java application, then the host name is declared as 'Localhost'.
   - But if the database application is in sum different system then the host name is declared as the Ip address that system
4. Port Number
   - It defines the exact location of the database application inside the system identified by host name
5. Database Name
   - It defines that which database is currently under operation
6. User
   - It defines the username of the account in which the database is present
7. Password
   - It defines the password of the account that being used

Eg.

Jdbc:mysql://loacalhost:3306/weja2?user=root&password=root

```
package com.jspiders.jdbc.select;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;

public class SelectDemo {
	public static void main(String[] a0rgs) {

		try {
			Class.forName("com.mysql.cj.jdbc.Driver");
			Connection connection =
DriverManager.getConnection("jdbc:mysql://localhost:3306/weja2?user=root&password=ro
ot");

			Statement statement = connection.createStatement();
			ResultSet resultSet = statement.executeQuery("select * from emp");

			while (resultSet.next())
			{
				System.out.println(resultSet.getInt(1) + "  "+
resultSet.getString(2));
			}
			connection.close();
			statement.close();
			resultSet.close();

		} catch (ClassNotFoundException e) {
			// TODO Auto-generated catch block
			e.printStackTrace();
		} catch (SQLException e) {
			// TODO Auto-generated catch block
			e.printStackTrace();
		}
	}
}
```

```java
package com.jspiders.jdbc.select;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;

public class SelectDemo3 {
	private static Connection connection;
	private static Statement statement;
	private static ResultSet resultSet;
	private static String driverPath = "com.mysql.cj.jdbc.Driver";
	private static String dburl = "jdbc:mysql://localhost:3306/weja2";
	private static String user = "root";
	private static String password = "root";
	private static String query;

	public static void main(String[] args) {

		try {
			//1. Load The driver Class
			Class.forName(driverPath);

			//2.open Connection
			connection = DriverManager.getConnection(dburl, user, password);

			//3.create and prepare Statement
			statement = connection.createStatement();
			query = "select * from emp";
			resultSet = statement.executeQuery(query);

			//process the result

			while (resultSet.next()) {
				System.out.println(resultSet.getString(1)+ " || "
							+  resultSet.getString(2)
						);
			}

		} catch (ClassNotFoundException | SQLException e) {
		}finally {
			try {
				if (connection != null) {
					connection.close();
				}
				if (statement != null) {
					statement.close();
				}
				if (resultSet != null) {
					resultSet.close();
				}
			} catch (SQLException e2) {
				// TODO: handle exception
			}
		}
```

Third Way of Connection

- In the previous two ways of connection, the database creadintial (username and passwords) were expose in the program hence it may lead to the illegal access of our data
- In order to avoid Such security issue, we can use the third way of connection in which database credential are not expose in the program
- Instead we will declare the database credentials in '.properties' File
- This file can be loaded in an object of Properties class and then this object can be Pass as an argument to the get connection method along with the dburl
- In this case the dburl will not contain the username and password

Note- The database credentials are declared in properties file in the format of key value pair

Creating Properties File

- **Step 1**
  - Create a new General Folder under the project folder directly.
  - The name of this folder will be 'resources'.
- **Step 2**
  - Under The resources folder create a new general file.
  - The name of this file will be 'db_info' and it will be mandatory to provide the extension as '**.properties**'
- **Step 3**
  - In the Properties file we will declare the database credential as follows
    - user=root
    - password=root
- **Note –** The username and password will be implicitly loaded from the properties object

```java
package com.jspiders.jdbc.select;

import java.io.FileInputStream;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.Statement;
import java.util.Properties;

public class SelectDemo4 {
	private static Connection connection;
	private static Statement statement;
	private static ResultSet resultSet;
	private static FileInputStream file;
	private static Properties properties = new Properties();
	private static String dburl = "jdbc:mysql://localhost:"
					+ "3306/weja2";
	private static String driverPath = "com.mysql.cj.jdbc.Driver";
	private static String filePath = "D:\\WEJA2\\jdbc\\resources\\db_info.properties";
	private static String query;

	public static void main(String[] args) {
		try {
			Class.forName(driverPath);

			file = new FileInputStream(filePath);
			properties.load(file);
			connection = DriverManager.getConnection
					(dburl, properties);

			statement = connection.createStatement();
			query = "select * from emp";
			resultSet = statement.executeQuery(query);

			while(resultSet.next()) {
				System.out.println(resultSet.getString(1) + " | "
						+ resultSet.getString(2));
			}

		} catch (Exception e) {
			e.printStackTrace();
		} finally {
			try {
				if (connection != null) {
					connection.close();
				}
				if (statement != null) {
					statement.close();
				}
				if (resultSet != null) {
					resultSet.close();
				}
				if (file != null) {
					file.close();
				}
			} catch (Exception e) {
				e.printStackTrace();
			}
		}
	}
```

- **getConnection()**
  - It is a static method present in DriverManager class it is used to establish connection for JDBC
  - It is responsible to return the object of Connection Interface
- **createStatment()**
  - It is a nonstatic method present in the Connection interface
  - It is used to obtain the object of Statement Inetface
- **executeQuery()**
  - It is a nonstatic method present in Statement Interface which accepts a string argument that will be consider as the query to be executable
  - This method is responsible to return the object of Resultset interface
  - The output of the query is loaded in ResultSet Object
- **next()**
  - It is a nonstatic method present in ResultSet Interface it is used to check if a next object/Value is present or not
  - If present, then this method returns 'True' as the output else it returns the 'false'
- **get[Datatype]()**
  - From the ResultSet Object we can retrieve any value according to its datatype based on its column position (column index)
  - The methods that for available for this operation are
    - getString()
    - getInt()
    - getLong()
    - getDouble(). etc
  - This Methods accepts an integer argument which considered as the column index
- **Load()**
  - It is a nonstatic method present in Properties class
  - It is used to load the contain from an input file as properties for the Project
- **getProperty()**
  - It is a nonstatic method in the properties class which is used to retrieve the value of a property based on its key
  - This method accepts a String argument which is considerd as the key of a property
- **Static Query**
  - The queries that hold or contain all the required values before compilation are known as static queries
  - In JDBC, the static queries are compiled at the time of execution itself
  - To handle static queries in JDBC, we can make use of the Statement interface
  - The PreparedStatement interface can also handeled static quieries.
  - It is because the PreparedStatement interface the child of Statement interface
- **Dynamic Queries**

- The queries which can be compile without the actual values are known as dynamic queries
- The dynamic queries are compiled by the database and kept ready for execution
- In the dynamic query, we need to reserve the place for actual values so that it satisfies or follows the required syntax. This is done with help of 'placeholder' (?)
- A placeholder in the query represents that a value will be asiigned in its position before the executing the query
- In JDBC , the dynamic queries can be handled by the prepared statement interface
- The object of its interface can be obtain with the help of method called as 'PreparedStatement()' which is present as a nonstatic method in the Connection interface
- Here as the query has to be compiled before the execution hence the query will be passed as a string argument to the preparedStatement()

```
package com.jspiders.jdbc.dynamic;

import java.io.FileInputStream;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.util.Properties;

public class DynamicSelect {

	private static Connection connection;
	private static PreparedStatement preparedStatement;
	private static ResultSet resultSet;
	private static Properties properties = new Properties();
	private static FileInputStream file;
	private static String filePath =
"D:\\WEJA2\\jdbc\\resources\\db_info.properties";
	private static String query;
	public static void main(String[] args) {
		try {
			file = new FileInputStream(filePath);
			properties.load(file);
			Class.forName(properties.getProperty("driverPath"));

			connection =
DriverManager.getConnection(properties.getProperty("dburl"),properties);
			query= "select * from emp where sal > ?";

			preparedStatement = connection.prepareStatement(query);
			preparedStatement.setInt(1, 2000);
			resultSet = preparedStatement.executeQuery();

			System.out.println("EMPNO   ENAME     JOB     \t          MGR\t\t HIREDATE
SAL      COMM");
			while (resultSet.next()) {
				System.out.println(resultSet.getString(1) + "\t"+
resultSet.getString(2) + "\t"+ resultSet.getString(3)+ "
\t\t"+resultSet.getString(4)+"\t\t" + resultSet.getString(5)+"\t" +
resultSet.getString(6) + "\t " + resultSet.getString(7));
			}
		} catch (Exception e) {
			e.printStackTrace();
		} finally {
			try {
				if (file != null) {
					file.close();
				}
				if (connection != null) {
					connection.close();
				}
				if (resultSet != null) {
					connection.close();
				}
				if (preparedStatement != null) {
					preparedStatement.close();
				}
			} catch (Exception e2) {
				// TODO: handle exception
			}
```

Note –

- a dynamic query can be compiled without the actual values but it cannot be executed without them.
- It means the actual values for the respective placeholder have to provide before executing query
- It can be done by using the setter methods for required datatype from the PreparedStatement interface
    - Eg. setInt(),setString(),setDouble(),setLong(),etc….
    - This method accepts two arguments
        - The first argument is considered has the position of a placeholder in the query. This argument has to be an integer value
        - The second argument is considered as the actual value to be assigned to the specific place holder. This argument can be of any datatype depending on the setter method which is being used

- Difference Between Dynamic query and Static Query

Static	Dynamic
• In static query, the values required for execution are hardcoded	• In dynamic query, the values required for execution can be represented as a placeholder(?)
• The values are present in the query before it gets compile	• The query can be compiled without the actual values
• The query is compiled during at a time of execution	• The query is compiled and kept ready before it is required to be executed
• Two change the values, changing the query is required	• The values can be change without changing the query
• Each time new values are required, then the query will be compiled for each time	• The query can be compiled once and can be executed multiple times with new values
• To handled static queries we can use Statement interface or PreparedStatement interface	• In dynamic we can only use PreparedStatement interface to handle Dynamic queries

Statement interface	PreparedStatement inteface
• This interface is used to handle only static queries	• This interface can be used to handled the static as well as dynamic quries
• It is the parent interface of PreparedStatement	• It is the child interface of Statement
• Its object can be Obtained with the help of a nonstatic method from connection interface known as createStatement()	• Its object can be obtained with the help of a nonstatic method from Connection interface known as prepareStatement(“”). This methods accepts the query as an String argument
• The executeQuery and executeQuery methods from this Interface accepts the query as a String argument	• The executeQuery and executeUpdate method from this interface does not accepts any argument

- Stored Procedure
  - Like method in java there are stored procedure in database
  - We can define multiple statements (queries) in a single stored procedure
  - This statements will be executed collectively when the stored procedure is called for execution
  - Hence the until the stored procedure is called, the statements will only be stored in the database but not executed

```java
package com.jspiders.jdbc.dynamic;

import java.io.FileInputStream;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;

import java.util.Properties;

public class DynamicInsert {

	private static Connection connection;
	private static PreparedStatement preparedStatement;
	private static int result;
	private static Properties properties = new Properties();
	private static FileInputStream file;
	private static String filePath =
"D:\\WEJA2\\jdbc\\resources\\db_info.properties";
	private static String query;

	public static void main(String[] args) {
		try {
			file = new FileInputStream(filePath);
			properties.load(file);

			Class.forName(properties.getProperty("driverPath"));

			connection =
DriverManager.getConnection(properties.getProperty("dburl"),properties);

			query= "insert into stud_info values(?,?,?,?)";

			preparedStatement = connection.prepareStatement(query);
			preparedStatement.setInt(1, 10);
			preparedStatement.setString(2, "kailas");
			preparedStatement.setString(3, "kailas@gmail.com");
			preparedStatement.setLong(4, 7412543654L);

			result = preparedStatement.executeUpdate();

			System.out.println("Query Ok ! "+result+" row(s) inserted !");
		} catch (Exception e) {
			e.printStackTrace();
		} finally {
			try {
				if (file != null) {
					file.close();
				}
				if (connection != null) {
					connection.close();
				}
				if (preparedStatement != null) {
					preparedStatement.close();
				}
			} catch (Exception e2) {
				e2.printStackTrace();
			}
		}
	}
}
```

```
package com.jspiders.jdbc.dynamic;

import java.io.FileInputStream;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.util.Properties;

public class DynamicSelect {

	private static Connection connection;
	private static PreparedStatement preparedStatement;
	private static ResultSet resultSet;
	private static Properties properties = new Properties();
	private static FileInputStream file;
	private static String filePath =  "D:\\WEJA2\\jdbc\\resources\\db_info.properties";
	private static String query;

	public static void main(String[] args) {
		try {
			file = new FileInputStream(filePath);
			properties.load(file);

			Class.forName(properties.getProperty("driverPath"));

			connection =
DriverManager.getConnection(properties.getProperty("dburl"),properties);

			query= "select * from emp where sal > ?";

			preparedStatement = connection.prepareStatement(query);
			preparedStatement.setInt(1, 2000);
			resultSet = preparedStatement.executeQuery();

			System.out.println("EMPNO   ENAME     JOB    \t         MGR\t\t HIREDATE
SAL      COMM");
			while (resultSet.next()) {
				System.out.println(resultSet.getString(1) + "\t"+
resultSet.getString(2) + "\t"+ resultSet.getString(3)+ "
\t\t"+resultSet.getString(4)+"\t\t" + resultSet.getString(5)+"\t" + resultSet.getString(6) +
"\t " + resultSet.getString(7));
			}
		} catch (Exception e) {
			e.printStackTrace();
		} finally {
			try {
				if (file != null) {
					file.close();
				}
				if (connection != null) {
					connection.close();
				}
				if (resultSet != null) {
					connection.close();
				}
				if (preparedStatement != null) {
					preparedStatement.close();
				}
			} catch (Exception e2) {
				e2.printStackTrace();
			}
		}
	}
}
```

```java
package com.jspiders.jdbc.dynamic;

import java.io.FileInputStream;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.util.Properties;

public class DynamicUpdate {

	private static FileInputStream file;
	private static String filePath = "D:\\WEJA2\\jdbc\\resources\\db_info.properties" ;
	private static Properties properties = new Properties();
	private static Connection connection;
	private static PreparedStatement preparedStatement;
	private static String query;
	private static int result;

	public static void main(String[] args) {
		try {
			file = new FileInputStream(filePath);

			properties.load(file);

			Class.forName(properties.getProperty("driverPath"));
			connection =
DriverManager.getConnection(properties.getProperty("dburl"),properties);
			query = "update stud_info set contact = 9999999999 where sname Like ?";
			preparedStatement = connection.prepareStatement(query);
			preparedStatement.setString(1, "%s");
			result = preparedStatement.executeUpdate();
			System.out.println("Query Ok ! " + result + " row(s) affectet !");


		} catch (Exception e) {
			e.printStackTrace();
		}finally {
			try {
				if (file != null) {
					file.close();
				}
				if (connection != null) {
					connection.close();
				}
				if (preparedStatement != null) {
					preparedStatement.close();
				}
			} catch (Exception e2) {
				e2.printStackTrace();
			}
		}
	}
}
```

```
package com.jspiders.jdbc.dynamic;

import java.io.FileInputStream;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.util.Properties;

public class DynamicDelete {

	private static FileInputStream file;
	private static Properties properties = new Properties();
	private static PreparedStatement preparedStatement;
	private static String query;
	private static int result;
	private static Connection connection;
	private static String filePath = "D:\\WEJA2\\jdbc\\resources\\db_info.properties";

	public static void main(String[] args) {
		try {
			file = new FileInputStream(filePath);
			properties.load(file);

			Class.forName(properties.getProperty("driverPath"));
			connection =
DriverManager.getConnection(properties.getProperty("dburl"),properties);

			query = "delete from stud_info where sname  like ?";
			preparedStatement = connection.prepareStatement(query);
			preparedStatement.setString(1, "%s");
			result = preparedStatement.executeUpdate();
			System.out.println("Query Ok ! , "+result + " row(s) affected");

		} catch (Exception e) {
			e.printStackTrace();
		}finally {
			try {
				if (file != null) {
					file.close();
				}
				if (preparedStatement != null) {
					preparedStatement.close();
				}
				if(connection != null)
				{
					connection.close();
				}
			} catch (Exception e2) {
				e2.printStackTrace();
			}
		}
	}
}
```

Stored Procedure

Syntax

```
delimitor /
create procedure procedure_name()
begin
        statement 1;
        statement 2;
        .
        .
        Statement n;
End /
Delimiter ;
```

Calling the stored procedure for a execution

Syntax : call procedure_name();

Eg. Call proc1();

Note :

- JDBC uses row data as query parameters but
- As we know java will process the data in Object format
- Hence It becomes the responsibility of programmer to establish the relationship between the data from database and the java object data
- A table in database must be considered as a class in java , Columns of the table must be considered as the properties of the class and rows in the table must be considered as the objects of the class

```java
package com.jspiders.jdbc.object;

public class Student {

	private int sid;
	private String name;
	private String email;
	private long contact;
	public int getSid() {
		return sid;
	}
	public void setSid(int sid) {
		this.sid = sid;
	}
	public String getName() {
		return name;
	}
	public void setName(String name) {
		this.name = name;
	}
	public String getEmail() {
		return email;
	}
	@Override
	public String toString() {
		return "Student [sid=" + sid + ", name=" + name + ", email=" + email + ",
contact=" + contact + "]";
	}
	public void setEmail(String email) {
		this.email = email;
	}
	public long getContact() {
		return contact;
	}
	public void setContact(long contact) {
		this.contact = contact;
	}


}
```

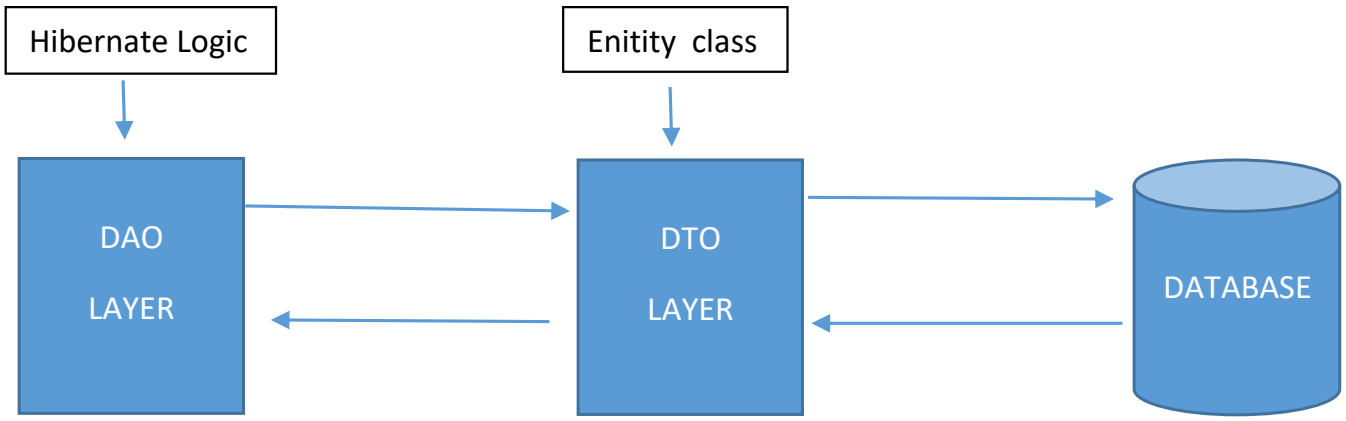
Note

- Step to load the driver class in JDBC operations, is deprecated
- It means JDBC will implicitly identify and load the Driver Class

- **Drawbacks of JDBC**
    1. IN JDBC being java developer we need to write SQL quires it means to write JDBC program, a programmer is forced to have the SQL knowledge
    2. For every JDBC operation, we need to define and use the database properties
    3. In JDBC, there is no implicit relationship between the relational database at the java objects
- Note –
    1. Due to this drawbacks, JDBC become a **deprecated** technology
- **ORM**
    1. ORM stands for Object Relational Mapping
    2. ORM was the develop to overcome the drawbacks of JDBC
    3. All though the ORM technology is build upon the JDBC framework itself.
    4. It means the ORM technology internally works on JDBC logic only
    5. In ORM we are not forced to write a SQL queries. It means while using ORM technology, SQL knowledge is not mandatory
    6. In ORM, the database properties can be declared just once and use multiple times
    7. As the name suggest, the objects from java application are implicitly mapped in relational database. It means a class(entity) is implicitly mapped as a table, properties of the class are mapped as the column as the table an objects of the class are mapped as the rows of the table

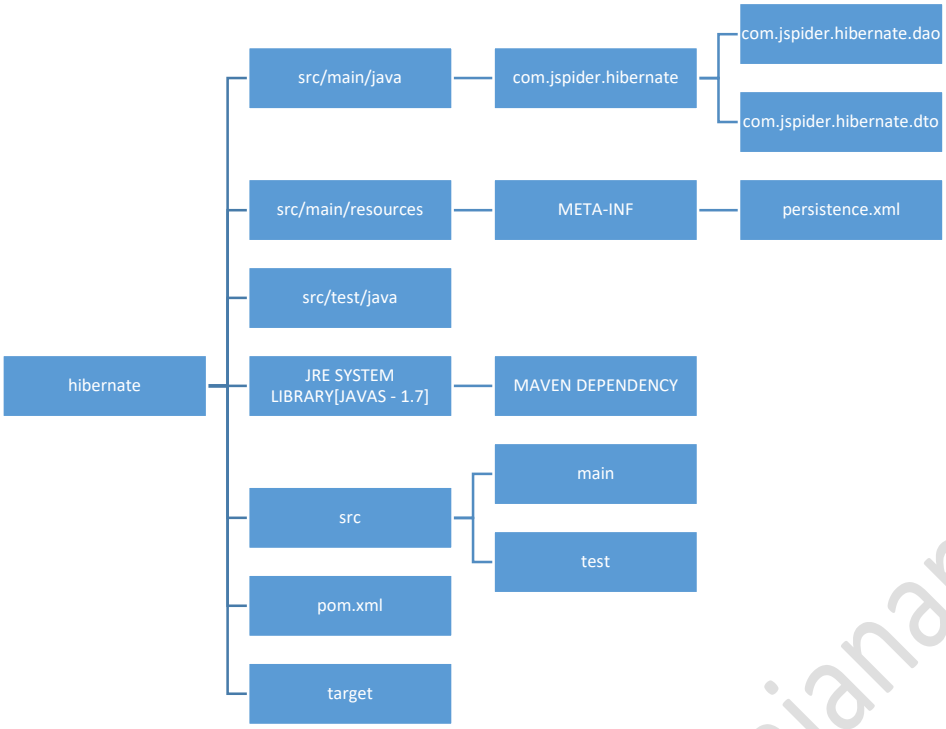
Note - One of the most ORM technology is Hibernate.

- **JPA**
    1. JPA stands for JAVA persistence API
    2. JPA is used to make a migration between one ORM tool to another, easier hence Hibernate is can also be use along with JPA and it will be known as hibernate JPA
- **Hibernate Layers**



- Hibernate structure consist of two layers
    1. DTO
    2. DAO
        - **DTO** layer consist of the entity classes required in the application. This entity classes are the data that is transfer between the java application and the database. Hence the name the **D**ata **T**ransfer **O**bject.
        - The **DAO** layer consist of the hibernate logic required for the application. The classes in DAO layer will be responsible to perform operations (access) on the entity classes from DTO layer. Hence the name **D**ata **A**ccess **O**bject.
    3. Note – This layers are represented as Packages in the projects.

- Steps to create a Maven project for hibernate
    1. Step 1
        - Click ctr+N and search for 'Maven'
    2. Step 2
        - Select 'Maven Project' and click on next
    3. Step 3
        - Click on next
    4. Step 4
        - In the 'select and Archetype' window search for 'org.apache.maven', and find the quick start Archetype of the version 1.4 and then click On Next
    5. Step -5
        - In the next window enter the group id in the format of 'domain.hostname' eg. 'com.jspider'
    6. Step -6
        - In the artifact id enter the project name eg. 'hibernate'
    7. Step – 7
        - Untick the check box for 'run archetype generation interactively' then click on finish
- Dependancy
    1. MySQL connector/j
    2. Hibernate core 5.6.15
    3. Hibernate Entity Manager 5.6.15
    4. Project Lombok(optional)
- Note –
    1. these dependencies are supposed to be
    2. added inside the dependencies tag in the pom.xml file
    3. Any required Dependency can be obtained from 'Maven Repository'
- Project Structure for Hibernate



- Configuring the maven Project for hibernate implementation
  1. Step1
     - Add the required Dependencies in pom.xml file
  2. Step 2
     - Create the packages for the DAO and DTO layer
  3. Step -3
     - To define the database properties, we need to create a specific structure for the 'persistence.xml' file. The structure can be created as follows
       - Inside the project folder, create a new source folder by the name src/main/resources
       - Inside the source folder create a new general folder by the name 'META-INF'
       - Inside the META-INF folder, create a new xml file by the name 'persistence'
- Adding Scema definition in persistence.xml
  1. Step - 1
     - Go to maven dependency section
  2. Step - 2
     - Find a jar file by the name 'javax.persistence-api-2.2.jar' and open it
  3. Step – 3
     - Find a package by the name 'javax.persistence' and open it
  4. Step-4
     - Scroll down the bottom of the package to find a file by the name 'persistence_2_1.xsd' and open it
  5. Step 5

- From the file, copy the content from line number 50 to 56 and paste it in the persistence.xml file

6. Step-6
    - Remove the 3 dots (…) present within the opening and closing persistence tag and then add 'persistence-unit'

- Defining database property in persistence.xml file
    1. Step-1
        - Inside the persistence unit tag, add a properties tag
    2. Step -2
        - For defining a single property, we need to declare a property tag inside the properties tag
    3. Step-3
        - For each property, the name and value has to be define
- Properties required in persistence.xml
    1. Driver path
    2. Dburl
    3. Username
    4. Password
    5. Dialect
    6. Mapping protocol
    7. Display query

1. Driver Path
    a. name= "javax.persistence.jdbc.driver" value= "com.mysql.cj.jdbc.Driver"
2. Dburl
    a. name= "javax.persistence.jdbc.url" value= "jdbc:mysql://localhost:3306/weja2"
3. username
    a. name= "javax.persistence.jdbc.user" value= "root"
4. password
    a. name= "javax.persistence.jdbc.password" value= "root"
5. Dialact
    a. name= "hibernate.dialect" value= "org.hibernate.dialect.MySQL8Dialect"
6. Mapping Protocol
    a. name= "hibernate.hbm2ddl.auto" value= "update"
7. Display Query
    a. name= "hibernate.show_sql" value= "true"

- adding the value for dilect property
  - Step 1
    - Go to maven dependency section and open a jar file name 'hibernate.core(version).jar'
  - Step-2
    - From the jar file, open the package named 'org.hibernate.dialect' and search for 'MYSQL8dilect.class' file
  - Step-3
    - Open the class file, copy the qualified name of that class and paste it in the value attribute for 'hibernate.dilect' property
- **Note –**
  - **the properties declared/define in persistence unit tag can be access by the named value set for the persistence unit**
- **@Entity**
  - This notation is used to mark a class as an entity class
  - To perform hibernate operations on an object of any class, it must be mark as an entity class
  - It is present in javax.persistence package
- **@Id**
  - This notation is used to mark a property as the primary key of the table corresponding to its entity class
  - It is present in javax.persistence package


```java
package com.jspider.hibernate.dto;

import javax.persistence.Entity;
import javax.persistence.Id;

import lombok.Data;

@Data
@Entity

public class EmployeeDTO {

	@Id
	private int id;
	private String ename;
	private String email;
	private long contact;
	private String address;

}
```

```java
package com.jspider.hibernate.dao;

import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;
import javax.persistence.EntityTransaction;
import javax.persistence.Persistence;

import com.jspider.hibernate.dto.EmployeeDTO;

public class OperationDAO {

	private static EntityManagerFactory entityManagerFactory;
	private static EntityManager entityManager;
	private static EntityTransaction entityTransaction;

	private static void openConnection()
	{
		entityManagerFactory = Persistence.createEntityManagerFactory("Employee");
		entityManager = entityManagerFactory.createEntityManager();
		entityTransaction = entityManager.getTransaction();
	}

	private static void closeConnection() {

		if (entityManager != null) {
			entityManager.close();
		}
		if (entityManagerFactory != null) {
			entityManagerFactory.close();
		}
		if (entityTransaction.isActive()) {
			entityTransaction.rollback();
		}
	}

	public static void main(String[] args) {

		openConnection();

		EmployeeDTO Emp1 = new EmployeeDTO();

		Emp1.setId(2);
		Emp1.setEname("Ramesh");
		Emp1.setContact(98456321424L);
		Emp1.setAddress("Dange Chouk");
		Emp1.setEmail("Ramesh@123.com");

		entityTransaction.begin();

		entityManager.persist(Emp1);

	entityTransaction.commit();

		closeConnection();

	}
}
```

```java
package com.jspider.hibernate.dto;

import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;
import javax.persistence.EntityTransaction;
import javax.persistence.Persistence;

public class EmployeeDAO2 {
	public static EntityManagerFactory entityManagerFactory;
	public static EntityManager entityManager;
	public static EntityTransaction entityTransaction;
	public static void openConnection() {

		entityManagerFactory = Persistence.createEntityManagerFactory("Employee");
		entityManager =  entityManagerFactory.createEntityManager();
		entityTransaction = entityManager.getTransaction();

	}
	public static void closeConnection()
	{
		if (entityManagerFactory != null) {
			entityManagerFactory.close();
		}
		if (entityManager != null) {
			entityManager.close();
		}
		if (entityTransaction != null) {
			if (entityTransaction.isActive()) {
				entityTransaction.rollback();
			}
		}
	}
	public static void main(String[] args) {
		openConnection();
		entityTransaction.begin();

		EmployeeDTO emp1 = new EmployeeDTO();

		emp1.setId(3);
		emp1.setAddress("mumbai");
		emp1.setContact(98532146214L);
		emp1.setEmail("Suresh@gmail.com");
		emp1.setEname("suresh");

		entityManager.persist(emp1);

		EmployeeDTO emp2 = new EmployeeDTO();

		emp2.setId(4);
		emp2.setAddress("pune");
		emp2.setContact(98532146214L);
		emp2.setEmail("Suresh@gmail.com");
		emp2.setEname("shubham");

		entityManager.persist(emp2);
		entityTransaction.commit();
		closeConnection();
	}
}
```

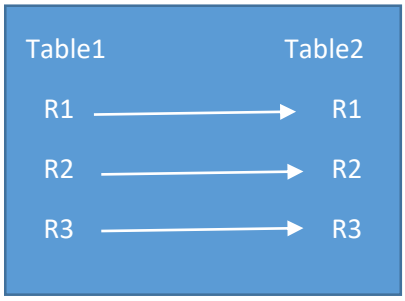
```java
package com.jspider.hibernate.dao;

import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;
import javax.persistence.EntityTransaction;
import javax.persistence.Persistence;

import com.jspider.hibernate.dto.EmployeeDTO;

public class EmployeeDAO3 {

	private static EntityManagerFactory entityManagerFactory;
	private static EntityManager entityManager;
	private static EntityTransaction entityTransaction;

	private static void openConnection() {

		entityManagerFactory = Persistence.createEntityManagerFactory("Employee");
		entityManager= entityManagerFactory.createEntityManager();
		entityTransaction = entityManager.getTransaction();

	}

	private static void closeConnection() {
		if (entityManagerFactory != null) {
			entityManagerFactory.close();
		}
		if (entityManager != null) {
			entityManager.close();
		}
		if (entityTransaction != null) {
			if (entityTransaction.isActive()) {
				entityTransaction.rollback();
			}
		}
	}

	public static void main(String[] args) {
		openConnection();
		entityTransaction.begin();

		EmployeeDTO emp = entityManager.find(EmployeeDTO.class, 3);
		entityTransaction.commit();
		System.out.println(emp);
		closeConnection();

	}

}
```

```java
package com.jspider.hibernate.dao;

import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;
import javax.persistence.EntityTransaction;
import javax.persistence.Persistence;

import com.jspider.hibernate.dto.EmployeeDTO;

public class EmployeeDAO5 {

	private static EntityManagerFactory entityManagerFactory;
	private static EntityManager entityManager;
	private static EntityTransaction entityTransaction;

	private static void openConnection() {

		entityManagerFactory =
Persistence.createEntityManagerFactory("Employee");
		entityManager= entityManagerFactory.createEntityManager();
		entityTransaction = entityManager.getTransaction();

	}

	private static void closeConnection() {
		if (entityManagerFactory != null) {
			entityManagerFactory.close();
		}
		if (entityManager != null) {
			entityManager.close();
		}
		if (entityTransaction != null) {
			if (entityTransaction.isActive()) {
				entityTransaction.rollback();
			}
		}
	}

	public static void main(String[] args) {
		openConnection();
		entityTransaction.begin();
		EmployeeDTO emp = entityManager.find(EmployeeDTO.class, 2);
		emp.setAddress("Mumbai");
		entityManager.persist(emp);
		entityTransaction.commit();
		closeConnection();
	}

}
```

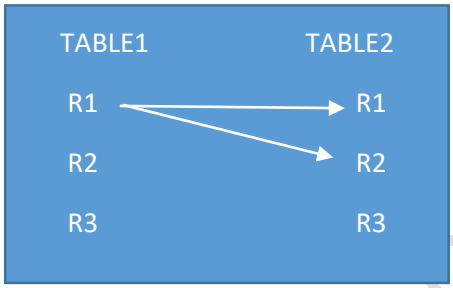
```java
package com.jspider.hibernate.dao;

import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;
import javax.persistence.EntityTransaction;
import javax.persistence.Persistence;

import com.jspider.hibernate.dto.EmployeeDTO;

public class EmployeeDAO4 {

	private static EntityManagerFactory entityManagerFactory;
	private static EntityManager entityManager;
	private static EntityTransaction entityTransaction;

	private static void openConnection() {

		entityManagerFactory = Persistence.createEntityManagerFactory("Employee");
		entityManager= entityManagerFactory.createEntityManager();
		entityTransaction = entityManager.getTransaction();

	}

	private static void closeConnection() {
		if (entityManagerFactory != null) {
			entityManagerFactory.close();
		}
		if (entityManager != null) {
			entityManager.close();
		}
		if (entityTransaction != null) {
			if (entityTransaction.isActive()) {
				entityTransaction.rollback();
			}
		}
	}

	public static void main(String[] args) {
		openConnection();
		entityTransaction.begin();
		entityManager.remove(entityManager.find(EmployeeDTO.class, 3));
		entityTransaction.commit();
		closeConnection();

	}

}
```

- **HIBERNATE MAPPING**
- Hibernate mapping is used to define or establish relationship between two entities.
- We have 4 types of hibernate mapping
    1. One to one mapping
        - When a record in table 1 is related to exactly one record from table 2 then it is called as one to one mapping
        - If the relationship is defined inside either one of the tables then it is called as unidirectional mapping

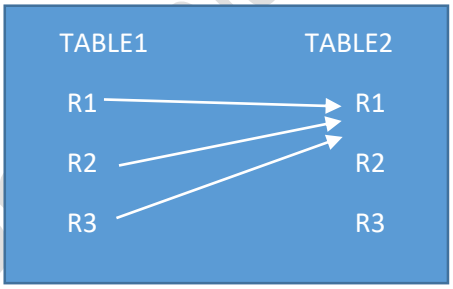
- If the relationship is define inside the both the table, then it is called as bidirectional mapping



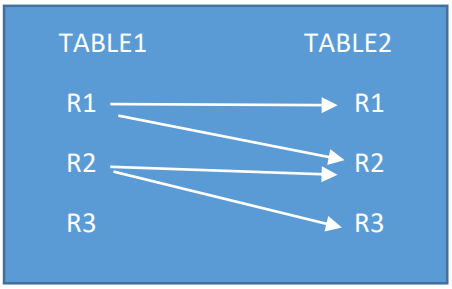
2. One to many mapping
   - When the record from table one is related to more than one records from table 2 then it is called as one to many mapping



3. Many to one mapping
   - When more than one record from table one are related to exactly one record from table to the it is called as many to one mapping



4. Many to many mapping
   - When more than one records from table one are related to more than one records from table two then it is called as many to many mapping
   - If the relationship is defined either of the table, then it is called as unidirectional mapping
   - If the relationship is defined inside the both table, then it is called as bidirectional mapping



- One to one unidirectional

```
package com.jspider.hibernate.dto;

import javax.persistence.Entity;
import javax.persistence.Id;

import lombok.Data;

@Data
@Entity
public class AdharCard {
	@Id
	private int id;
	private long adharNumber;
	private String dateOfIssue;

}
package com.jspider.hibernate.dto;

import javax.persistence.Entity;
import javax.persistence.Id;
import javax.persistence.OneToOne;

import lombok.Data;

@Data
@Entity
public class Person {
	@Id
	private int id;
	private String name;
	private String email;
	@OneToOne
	private AdharCard adharCard;

}
```

```java
package com.jspider.hibernate.dao;

import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;
import javax.persistence.EntityTransaction;
import javax.persistence.Persistence;

import com.jspider.hibernate.dto.AdharCard;
import com.jspider.hibernate.dto.Person;

public class PersonDAO {

	private static EntityManagerFactory entityManagerFactory;
	private static EntityManager entityManager;
	private static EntityTransaction entityTransaction;

	private static void openConnection()
	{
		entityManagerFactory = Persistence.createEntityManagerFactory("Employee");
		entityManager = entityManagerFactory.createEntityManager();
		entityTransaction = entityManager.getTransaction();
	}

	private static void  closeConnection() {
		if (entityManagerFactory != null) {
			entityManagerFactory.close();
		}
		if (entityManager != null) {
			entityManager.close();
		}
		if (entityTransaction != null) {
			if (entityTransaction.isActive()) {
				entityTransaction.rollback();
			}
		}
	}

	public static void main(String[] args) {
		openConnection();
		entityTransaction.begin();
		Person per1 = new Person();
		per1.setName("Ramesh");
		per1.setId(1);
		per1.setEmail("ramesh@gmail.com");

		AdharCard adharCard = new AdharCard();

		adharCard.setAdharNumber(123456987456L);
		adharCard.setId(1);
		adharCard.setDateOfIssue("12/11/2014");
		entityManager.persist(adharCard);

		per1.setAdharCard(adharCard);

		entityManager.persist(per1);
		entityTransaction.commit();
		closeConnection();
	}
}
```

- One to Many Unidirectional Mapping

```
package com.jspider.hibernate.dto;

import javax.persistence.Entity;
import javax.persistence.Id;

import lombok.Data;

@Entity
@Data
public class Employee {
	@Id
	private int id;
	private String name;
	private double salary;
	private String email;

}
package com.jspider.hibernate.dto;

import java.util.List;

import javax.persistence.Entity;
import javax.persistence.Id;
import javax.persistence.OneToMany;

import lombok.Data;

@Data
@Entity
public class Company {
	@Id
	private int id;
	private String name;
	private String adress;
	private String email;
	@OneToMany
	private List <Employee> employee;
}
```

```java
package com.jspider.hibernate.dao;

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

import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;
import javax.persistence.EntityTransaction;
import javax.persistence.Persistence;

import com.jspider.hibernate.dto.Company;
import com.jspider.hibernate.dto.Employee;

public class CompanyDAO {
	private static EntityManagerFactory entityManagerFactory;
	private static EntityManager entityManager;
	private static EntityTransaction entityTransaction;
	private static void openConnection() {
		//write here to open connection
	}

	private static void closeConnection() {
		if (entityManagerFactory != null) {
			entityManagerFactory.close();
		}
		if (entityManager != null) {
			entityManager.close();
		}
		if (entityTransaction != null) {
			if (entityTransaction.isActive()) {
				entityTransaction.rollback();
			}
		}

	}

	public static void main(String[] args) {
		openConnection();
		entityTransaction.begin();

		Company company = new Company();
		company.setId(1);
		company.setName("Infosys");
		company.setEmail("infosys@gmail.com");
		company.setAdress("pune");

		Employee emp1 = new Employee();
		emp1.setId(1);
		emp1.setName("remesh");
		emp1.setEmail("ramesh@123.com");
		emp1.setSalary(2400);

		Employee emp2 = new Employee();
		emp2.setId(2);
		emp2.setName("suresh");
		emp2.setEmail("suresh@123.com");
		emp2.setSalary(2500);

		Employee emp3 = new Employee();
		emp3.setId(3);
		emp3.setName("mahesh");
		emp3.setEmail("mahesh@123.com");
		emp3.setSalary(2600);

		List<Employee> employees = new ArrayList<>();
		employees.add(emp1);
		employees.add(emp2);
		employees.add(emp3);

		company.setEmployee(employees);

		entityManager.persist(emp1);
		entityManager.persist(emp2);
		entityManager.persist(emp3);

		entityManager.persist(company);

		entityTransaction.commit();
		closeConnection();
	}
}
```

- In case one to many of unidirectional mapping hibernate will create one extra table called as mapping table to establish relationship between 2 entities
- Many TO One

```
package com.jspider.manytoone.dto;

import javax.persistence.Entity;
import javax.persistence.Id;
import javax.persistence.ManyToOne;

import lombok.Data;

@Data
@Entity
public class Player {

	@Id
	private int id;
	private String name;
	private int jerseyNmber;
	private int age;
	@ManyToOne
	private Team team;
}
package com.jspider.manytoone.dto;

import javax.persistence.Entity;
import javax.persistence.Id;

import lombok.Data;


@Data
@Entity
public class Team {
	@Id
	private int id;
	private String name;
	private String country;

}
```

```java
package com.jspider.manytoone.dto;

import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;
import javax.persistence.EntityTransaction;
import javax.persistence.Persistence;

public class PlayerDao {

	private static EntityManagerFactory entityManagerFactory;
	private static EntityManager entityManager;
	private static EntityTransaction entityTransaction;

	private static void openConnection() {
		entityManagerFactory = Persistence.createEntityManagerFactory("Employee");
		entityManager = entityManagerFactory.createEntityManager();
		entityTransaction = entityManager.getTransaction();
	}

	private static void closeConnection() {
		if (entityManagerFactory != null) {
			entityManagerFactory.close();
		}
		if (entityManager != null) {
			entityManager.close();
		}
		if (entityTransaction != null) {
			if (entityTransaction.isActive()) {
				entityTransaction.rollback();
			}
		}

	}

	public static void main(String[] args) {
		openConnection();
		entityTransaction.begin();

		Player player1 = new Player();
		player1.setId(1);
		player1.setName("virat");
		player1.setJerseyNmber(18);
		player1.setAge(35);

		Player player2 = new Player();
		player2.setId(2);
		player2.setName("rohit");
		player2.setJerseyNmber(45);
		player2.setAge(36);


		Player player3 = new Player();
		player3.setId(3);
		player3.setName("hardik");
		player3.setJerseyNmber(30);
		player3.setAge(32);

		Team team = new Team();
		team.setId(1);
		team.setName("Team A");
		team.setCountry("India");

		player1.setTeam(team);
		player2.setTeam(team);
		player3.setTeam(team);

		entityManager.persist(team);
		entityManager.persist(player1);
		entityManager.persist(player2);
		entityManager.persist(player3);

		entityTransaction.commit();
		closeConnection();

	}
}
```

Many to Many

```
package com.jspider.manytoone.dto;


import javax.persistence.Entity;
import javax.persistence.Id;

import lombok.Data;

@Data
@Entity
public class Course {

	@Id
	private int id;
	private String name;
	private double fees;
}
```

```
package com.jspider.manytoone.dto;

import java.util.List;

import javax.persistence.Entity;
import javax.persistence.Id;
import javax.persistence.ManyToMany;

import lombok.Data;

@Data
@Entity
public class Student {
	@Id
	private int id;
	private String name;
	private String email;
	private String adress;

	@ManyToMany
	private List<Course>courses;

}


//in arraylist we can pass the value directly also
//eg.
//List<Course>course2=Arrays.asList(course1,course2);
```

```java
package com.jspider.manytoone.dao;

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

import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;
import javax.persistence.EntityTransaction;
import javax.persistence.Persistence;

import com.jspider.manytoone.dto.Course;
import com.jspider.manytoone.dto.Student;

public class StudentDao {

	private static EntityManagerFactory entityManagerFactory;
	private static EntityManager entityManager;
	private static EntityTransaction entityTransaction;

	private static void openConnection() {
		//open connection initialise
	}
	private static void closeConnection() {
		//close connection statement
	}
	public static void main(String[] args) {
		openConnection();
		entityTransaction.begin();

		Student student1 = new Student();
		student1.setId(1);
		student1.setName("Rahul");
		student1.setEmail("rahul@gmail.com");
		student1.setAdress("pune");

		Student student2 = new Student();
		student2.setId(2);
		student2.setName("Rakesh");
		student2.setEmail("rakesh@gmail.com");
		student2.setAdress("kolhapur");

		Course java = new Course();
		java.setId(1);
		java.setName("core Java");
		java.setFees(15000.00);

		Course sql = new Course();
		sql.setId(2);
		sql.setName("SQL");
		sql.setFees(24000.00);

		List<Course> courses1 = new ArrayList<>();
		courses1.add(java);
		courses1.add(sql);

		List<Course> courses2 = new ArrayList<>();
		courses2.add(sql);
		courses2.add(java);

		student1.setCourses(courses1);
		student2.setCourses(courses2);

		entityManager.persist(java);
		entityManager.persist(sql);

		entityManager.persist(student1);
		entityManager.persist(student2);

		entityTransaction.commit();
		closeConnection();
	}
}
```

'One to one bidirectional'

```java
package com.jspider.hibernate.dto;

import javax.persistence.JoinColumn;

import javax.persistence.Entity;
import javax.persistence.Id;
import javax.persistence.OneToOne;

import lombok.Data;

@Data
@Entity
public class VotingCard {

	@Id
	private int id;
	private String cardNumber;
	private String dateOfIssue;

	@OneToOne

	@JoinColumn

	private Voter voter;


}
```

```java
package com.jspider.hibernate.dto;

import javax.persistence.Entity;
import javax.persistence.Id;
import javax.persistence.OneToOne;

import lombok.Data;

@Data
@Entity
public class Voter {
	@Id
	private int id;
	private String name;
	private String adress;

	@OneToOne (mappedBy = "votingCard")

	private VotingCard votingCard;

}
```

```java
package com.jspider.hibernate.dao;

import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;
import javax.persistence.EntityTransaction;
import javax.persistence.Persistence;

import com.jspider.hibernate.dto.Voter;
import com.jspider.hibernate.dto.VotingCard;

public class VoterDao {

	private static EntityManagerFactory entityManagerFactory;
	private static EntityManager entityManager;
	private static EntityTransaction entityTransaction;

	private static void openConnection()
	{
		entityManagerFactory = Persistence.createEntityManagerFactory("Employee");
		entityManager = entityManagerFactory.createEntityManager();
		entityTransaction = entityManager.getTransaction();
	}

	private static void  closeConnection() {
		if (entityManagerFactory != null) {
			entityManagerFactory.close();
		}
		if (entityManager != null) {
			entityManager.close();
		}
		if (entityTransaction != null) {
			if (entityTransaction.isActive()) {
				entityTransaction.rollback();
			}
		}
	}

	public static void main(String[] args) {

		openConnection();
		entityTransaction.begin();

		Voter voter1 = new Voter();
		voter1.setId(1);
		voter1.setName("Ramesh");
		voter1.setAdress("pune");

		VotingCard votingCard = new VotingCard();
		voter1.setVotingCard(votingCard);

		votingCard.setId(11);
		votingCard.setDateOfIssue("02/06/2015");
		votingCard.setCardNumber("VOTE101");
		votingCard.setVoter(voter1);

		entityManager.persist(votingCard);
		entityManager.persist(voter1);

		entityTransaction.commit();
		closeConnection();
	}

}
```

```java
package com.jspider.hibernate.dto;

import java.util.List;

import javax.persistence.Entity;
import javax.persistence.Id;
import javax.persistence.ManyToMany;

import lombok.Data;

@Entity
@Data
public class Customer {

	@Id
	private int id;
	private String name;
	private String email;
	private String adress;

	@ManyToMany
	private List<Product> product;

}
package com.jspider.hibernate.dto;

import java.util.List;

import javax.persistence.Entity;
import javax.persistence.Id;
import javax.persistence.ManyToMany;

import lombok.Data;

@Data
@Entity
public class Product {

	@Id
	private int id;
	private String Name;
	private double price;

	@ManyToMany
	@JoinTable(joinColumns = @JoinColumn(referencedColumnName = "id"),inverseJoinColumns =
@JoinColumn(referencedColumnName = "id"))
	private List<Customer>customers;

}
```

```java
package com.jspider.hibernate.dao;

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

import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;
import javax.persistence.EntityTransaction;
import javax.persistence.Persistence;

import com.jspider.hibernate.dto.Customer;
import com.jspider.hibernate.dto.Product;

public class CustomerDao {


	public static EntityManagerFactory entityManagerFactory;
	public static EntityManager entityManager;
	public static EntityTransaction entityTransaction;

	public static void openConnection() {
		//open connections statements

	}

	public static void closeConnection()
	{
		//closeConnection statements
	}

	public static void main(String[] args) {
		openConnection();
		entityTransaction.begin();

		Customer customer1 = new Customer();
		customer1.setId(1);
		customer1.setName("Shubham");
		customer1.setEmail("shubham@gmail.com");
		customer1.setAdress("Sangali");

		Customer customer2 = new Customer();
		customer2.setId(2);
		customer2.setName("Rohan");
		customer2.setEmail("Rohan@gmail.com");
		customer2.setAdress("Satara");

		Product product1 = new Product();
		product1.setId(101);
		product1.setName("soap");
		product1.setPrice(45.65);

		Product product2 = new Product();
		product2.setId(201);
		product2.setName("colgate");
		product2.setPrice(101.65);

		List<Customer> customers1 = new ArrayList<>();
		customers1.add(customer1);

		List<Customer> customers2 = new ArrayList<>();
		customers2.add(customer1);
		customers2.add(customer2);

		List<Product> products1 = new ArrayList<>();
		products1.add(product2);

		List<Product>products2 = new ArrayList<>();
		products2.add(product1);
		products2.add(product2);

		customer1.setProducts(products1);
		customer2.setProducts(products2);

		product1.setCustomers(customers2);
		product1.setCustomers(customers1);

		entityManager.persist(product2);
		entityManager.persist(product1);

		entityManager.persist(customer1);
		entityManager.persist(customer2);


		entityTransaction.commit();
		closeConnection();
	}
}
```

- Hibernate Mapping annotation
  - @OneToOne
    - It is used to define one to one relationship between 2 entities
    - It is a property level annotation
  - @OneToMany
    - It is used to define one to many re
    - It is a property level annotation
  - @ManyToOne
    - It is used to many to one relationship between 2 entities
    - It is a property level annotation
  - @ManyToMany
    - It is used to define Many to many relationships between two entities.
    - It is property level annotation
- Data redundancy in bidirectional mapping
  - In one to one bi directional mapping hibernate will add foreign key columns in both the entity tables but foreign key column inside one table is sufficient to establish the relationship between two tables so we can normalize one of the tables using @joincolums annotation and Mapped By attribute
  - In case of Many to many bidirectional mapping hibernate will create two mapping tables but one mapping table is sufficient to establish the relationship between two tables so we have to normalize either one of the mapping tables. We can make use of @jointables annotations and mapped by attributes *"@JoinTable(joinColumns = @JoinColumn(referencedColumnName = "id"),inverseJoinColumns = @JoinColumn(referencedColumnName = "id"))"*

```java
package com.jspider.hibernate.dto;

import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.Id;
import javax.persistence.Table;

import lombok.Data;

@Data
@Entity
@Table(name = "student_info")
public class Student {
	@Id
	@Column(name = "student_Id")
	private int id;
	@Column(name = "student_Name")
	private String name;
	@Column(name = "student_Email")
	private String email;

}
```

```java
package com.jspider.hibernate.dto;

import java.util.List;

import javax.persistence.CascadeType;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.Id;
import javax.persistence.OneToMany;
import javax.persistence.Table;

import lombok.Data;

@Data
@Entity
@Table(name = "school_info")
public class School {
	@Id
	@Column(name = "school_Id")
	private int id;
	@Column(name = "school_Name")
	private String name;
	@Column(name = "school_Adress")
	private String adress;
	@OneToMany(cascade = CascadeType.PERSIST)
	private List<Student>students;

}
```

```java
@GeneratedValue(strategy = GenerationType.IDENTITY)
```

if we are use this for Id then hibernate add automatically id no need to set value for ID

```java
package com.jspider.hibernate.dao;

import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;

import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;
import javax.persistence.EntityTransaction;
import javax.persistence.Persistence;

import com.jspider.hibernate.dto.School;
import com.jspider.hibernate.dto.Student;

public class SchoolDao {

	private static EntityManagerFactory entityManagerFactory;
	private static EntityManager entityManager;
	private static EntityTransaction entityTransaction;

	private static void openConnection()
	{
		//open connection statement
	}

	private static void  closeConnection() {
		//close connection statement
	}

	public static void main(String[] args) {
		openConnection();
		entityTransaction.begin();

		Student stud1 = new Student();
		stud1.setId(1);
		stud1.setName("Shubham");
		stud1.setEmail("shubham@gmail.com");

		Student stud2 = new Student();
		stud2.setId(2);
		stud2.setName("Rohan");
		stud2.setEmail("rohan.com");

		School school = new School();
		school.setId(1);
		school.setName("COEP");
		school.setAdress("Pune");


		List<Student>students = new ArrayList<>();
		students.add(stud1);
		students.add(stud2);

//		school.setStudents(students);
		school.setStudents(Arrays.asList(stud1,stud2));

//		entityManager.persist(stud1);
//		entityManager.persist(stud2);

		entityManager.persist(school);

		entityTransaction.commit();
		closeConnection();
	}

}
```

- More annotations in hibernate
  - @Table
    - This annotation is used to give user defined name to the table, it takes one attribute called as name
  - @Column
    - This is used to give user define name to the column
    - It takes one attribute called as name
  - We can make use of one attribute called as Casted along with all the mapping annotations it is used to perform castending operation on entities
- **JPQL (java persistence Query Language)**
  - JPQL stands for Java Persistence Query Language
  - It is used to write customize quiries. Seens we have different different ORM tools available in the market each orm tool will understand one specific query language that is hibernate orm tool will understand HQL (hibernate Query Language) and I-batis or My-batis orm tool are understands DQL (Dynamic Query Language). In this case both the query languages are incompatible to each other so it will be difficult to migrate from one orm tool to another orm tool to overcome this problem JPQL is introduced. JPQL is understandable for all the ORM tools
  - **NOTE-** so in JPQL instead of using table name and column name we will make use of class name and its properties

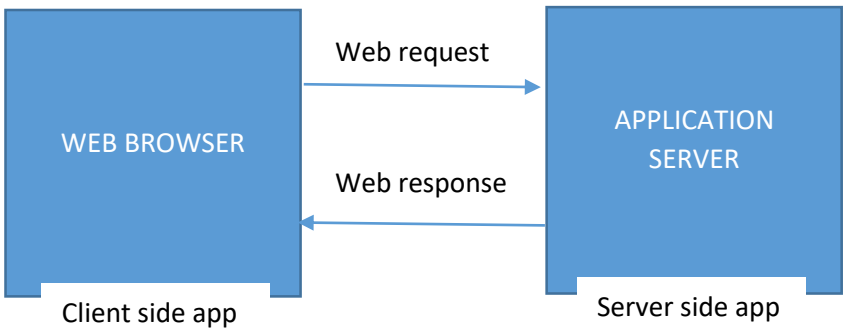

```
package com.jspider.hibernate.dao;

import java.util.Iterator;
import java.util.List;

import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;
import javax.persistence.EntityTransaction;
import javax.persistence.Persistence;
import javax.persistence.Query;

import com.jspider.hibernate.dto.EmployeeDTO;

public class EmployeeDAO6 {
	public static EntityManagerFactory entityManagerFactory;
	public static EntityManager entityManager;
	public static EntityTransaction entityTransaction;
	public static void openConnection() {
		//open

	}
	public static void closeConnection()
	{
		//close connection
	}
	public static void main(String[] args) {

		openConnection();
		entityTransaction.begin();

		Query query = entityManager.createQuery("SELECT emp FROM EmployeeDTO emp");
		List<EmployeeDTO> employees = query.getResultList();

		for (EmployeeDTO employee : employees) {
			System.out.println(employee);
		}
		entityTransaction.commit();
		closeConnection();
	}
}
```

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

        openConnection();
        entityTransaction.begin();

        Query query = entityManager.createQuery("DELETE FROM EmployeeDTO WHERE ID = 2");
        int row  = query.executeUpdate();

        System.out.println(row + "row(S) affected");

        entityTransaction.commit();
        closeConnection();

    }
```

- Create DataBase Did not exist
  - This particular statement can be use to create specified database automatically .if it doesnot exists

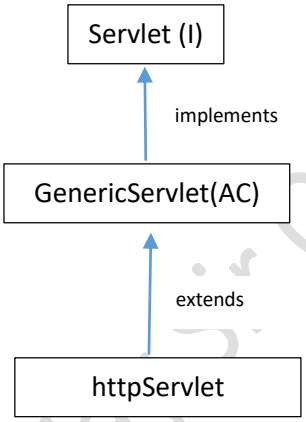
SERVLET

- Servlet is the technology (only one) or java API which is used to accept web request and we can generate web response in web based applications
- Web Application
  - An application which is present inside a server and which can be access using standard web browser.
- Web Brrowser
  - Web browser is the application using which we can generate the web request and we can recive web response
- Web request
  - A web request is used to access a specific web resourse from the server
- Web response
  - The sent web request will be identified by the servlet and respective web response will be generated by the servlet



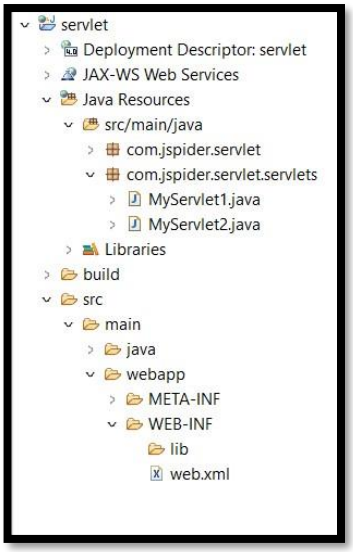
- URL
  - URL stands for uniform resource locator it is used to locate the web resource present inside web application.
  - Using this URL we can access any particular web resource using any device or using any System
  - Format
    - Protocol://host_name or Domain_name :port_number/name_of_resource?query_String#fragment_id
    - Eg. JDBC://localhost:3306/weja2
    - Protocol
      - The protocol indicates the weather a request is protocol dependent or protocol independent and also it indicates which protocol has been used to access the specific web resource
    - Hostname or Domain name
      - Indicates the location of the server in which the requested web resources is present
    - Port number
      - The port number indicates the exact location of the application inside an identified server
    - Name_Of_Resource
      - It indicates the location of a requested web resource inside an identified application
    - Query String
      - In order to send some data from client side application to the server side application trough the url then it declared as query string
      - It will be in the form of key value pair multiple key value pair will be separated using '&' symbol
    - Fragment_Id
      - It is the id of the page which is currently being displayed
- Web Resource
  - It is the resource present inside the web application. There are two types of web resources
    - Static Web resource
    - Dynamic web resource
  - Static web resource
    - The resource which generates static or fix response is called as static web resource
  - Dynamic web resource

        - The resource which generate dynamic response (changing response) is called as dynamic web resource
- Static Web Application
    - The application which contains only static web resources is called static web application
- Dynamic web Application
    - If application contents dynamic web resources the it is called as dynamic web application
- Note –
    - Static web resources can be develop using html
    - Dynamic web resources can be develop using servlets and JSP
- Servlet in java
    - It is a special class in java
    - Servlet hierarchy



- There are two ways create servlet in java
    - By extending GenericServlet abstract class
        - If we create a servlet by extending generic abstract class, then that particular servlet will become protocol independent
    - By extending HttpServlet class
        - We can create a servlet class in java by extending http servlet class
        - The servlet class which extends HttpsServlet class will become protocol dependent
- Step 2 install Apache TomCat Server
    - Step 1
        - In a browser search for apache tomcat server download
    - Step-2
        - Click on the first website link
    - Step-3
        - Download the respective zip file based on your system configuration
    - Step-4

        - Go to your download section and extract that zip file
    - Step-5
        - Go to your eclipse IDE click on window option then click on show view option then click on server option
        - Server section will be added in your ide then click on servers one link will be displayed (no servers are available click this link to create a new server)
    - Step-6
        - Click on that particular link then one window will be appeared in that window just click on apache drop down menu
        - Select respective tomcat server version after that click on next then window will get appeared as server install directory
        - Then click on browse and select the folder which contains extracted file (double click on that particular folder)
        - Click on next and then click on finish
- Step to create Dynamic Project
    - Step 1
        - Ctr+1 one window will appear as a select a wizard
        - In filter section type dynamic web project then select Dynamic web project option and click on enter
    - Step-2
        - One window will appear as dynamic web project then provide a project name and then click on next
        - Again click on next
        - In web module window tick the checkbox to generate web.xml deployment descriptor
        - Then click on finish
- **Structure**



- Web.xml
  - It is an entry point for the dynamic web project
  - It is also called as deployment descriptor
  - Web.xml can be used to configure user created servlets

```java
package com.jspider.servlet.servlets;

import java.io.IOException;

import javax.servlet.GenericServlet;
import javax.servlet.ServletException;
import javax.servlet.ServletRequest;
import javax.servlet.ServletResponse;

public class MyServlet1 extends GenericServlet {


	private static final long serialVersionUID = 1L;

	@Override
	public void service(ServletRequest req, ServletResponse res) throws ServletException, IOException {
		System.out.println("Hello world");

	}

}
```

Web.xml

```xml
<servlet>
	<servlet-name>MyServlet1</servlet-name>
	<servlet-class>com.jspider.servlet.servlets.MyServlet1</servlet-class>
</servlet>

<servlet-mapping>
	<servlet-name>MyServlet1</servlet-name>
	<url-pattern>/MyServlet1</url-pattern>
</servlet-mapping>
```

- Web Server
  - It is a software which is capable of accepting web request and send back web response
  - Web container/servlet container it is a component of web server which contain servlets. It is responsible for managing servlet life cycle

```java
package com.jspider.servlet.servlets;

import java.io.IOException;
import java.io.PrintWriter;

import javax.servlet.GenericServlet;
import javax.servlet.ServletException;
import javax.servlet.ServletRequest;
import javax.servlet.ServletResponse;

public class MyServlet2 extends GenericServlet{

	private static final long serialVersionUID = 1L;

	@Override
	public void service(ServletRequest req, ServletResponse res) throws ServletException, IOException {
		res.setContentType("text/html");
		PrintWriter writer = res.getWriter();
		writer.println("<h1>Hello world</h1>");
	}

}
```

- Note - servlet container will create object for servlet request and servlet response
- 2nd way of creating servlet n java

```java
package com.jspider.servlet.servlets;

import java.io.IOException;
import java.io.PrintWriter;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class MyServlet3 extends HttpServlet {

	private static final long serialVersionUID = 1L;

	@Override
	protected void doGet(HttpServletRequest req, HttpServletResponse resp) throws
ServletException, IOException {

		resp.setContentType("text/html");
		PrintWriter writer = resp.getWriter();
		writer.println("<h1>hiiii<h1/>");
	}


}
```

```java
package com.jspider.servlet.servlets;

import java.io.IOException;
import java.io.PrintWriter;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class MyServlet4 extends HttpServlet {

	private static final long serialVersionUID = 1L;

	@Override
	protected void doGet(HttpServletRequest req, HttpServletResponse resp) throws
ServletException, IOException {
		doPost(req, resp);
	}

	@Override
	protected void doPost(HttpServletRequest req, HttpServletResponse resp) throws
ServletException, IOException {
		resp.setContentType("text/html");
		PrintWriter writer = resp.getWriter();
		writer.println("<h1>hiiii whats up<h1/>");
	}

}
```

```java
package com.jspider.servlet.servlets;

import java.io.IOException;
import java.io.PrintWriter;
import java.util.Date;

import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

@WebServlet("/MyServlet5")
public class MyServlet5 extends HttpServlet {

	private static final long serialVersionUID = 1L;

	@Override
	protected void doGet(HttpServletRequest req, HttpServletResponse resp)
throws ServletException, IOException {
		Date date = new Date();


		resp.setContentType("text/html");
		resp.setHeader("Refresh", "1");
		PrintWriter writer = resp.getWriter();
		writer.println("<h1>"+date+"<h1/>");
	}
}
```

- @webServlet
  - This annotation is used to provide the url pattern for the user define servlet. If we make use of this particular annotation, then configuring the user created servlet inside web.xml is not required
- Implicit mapping methods
  - doGet()
    - Whenever the request is send a particular servlet a doGet method will be called automatically
    - It is used to provide a execution logic for that particular servlet
  - doPost()
    - doPost method will not be called automatically but we can make use of this method to provide the execution logic for the servlet

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Addition</title>
</head>
<body>
	<div align="centre">
		<form action="MyServlet6">
			<table>
				<tr>
					<td>Num1</td>
					<td><input type="text" name="num1"></td>
				</tr>
				<tr>
					<td>Num2</td>
					<td><input type="text" name="num2"></td>
				</tr>
			</table>
			<input type="submit" value="Add">
		</form>
	</div>

</body>
</html>
```

```
package com.jspider.servlet.servlets;

import java.io.IOException;
import java.io.PrintWriter;

import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

@WebServlet("/MyServlet6")
public class MyServlet6 extends HttpServlet {

	private static final long serialVersionUID = 1L;

	@Override
	protected void doGet(HttpServletRequest req, HttpServletResponse resp)
throws ServletException, IOException {
		int num1 = Integer.parseInt(req.getParameter("num1"));
		int num2 = Integer.parseInt(req.getParameter("num2"));
		int sum = num1 + num2;

		resp.setContentType("text/html");
		PrintWriter writer = resp.getWriter();
		writer.println("<h1>"+sum+"</h1>");
	}

}
```

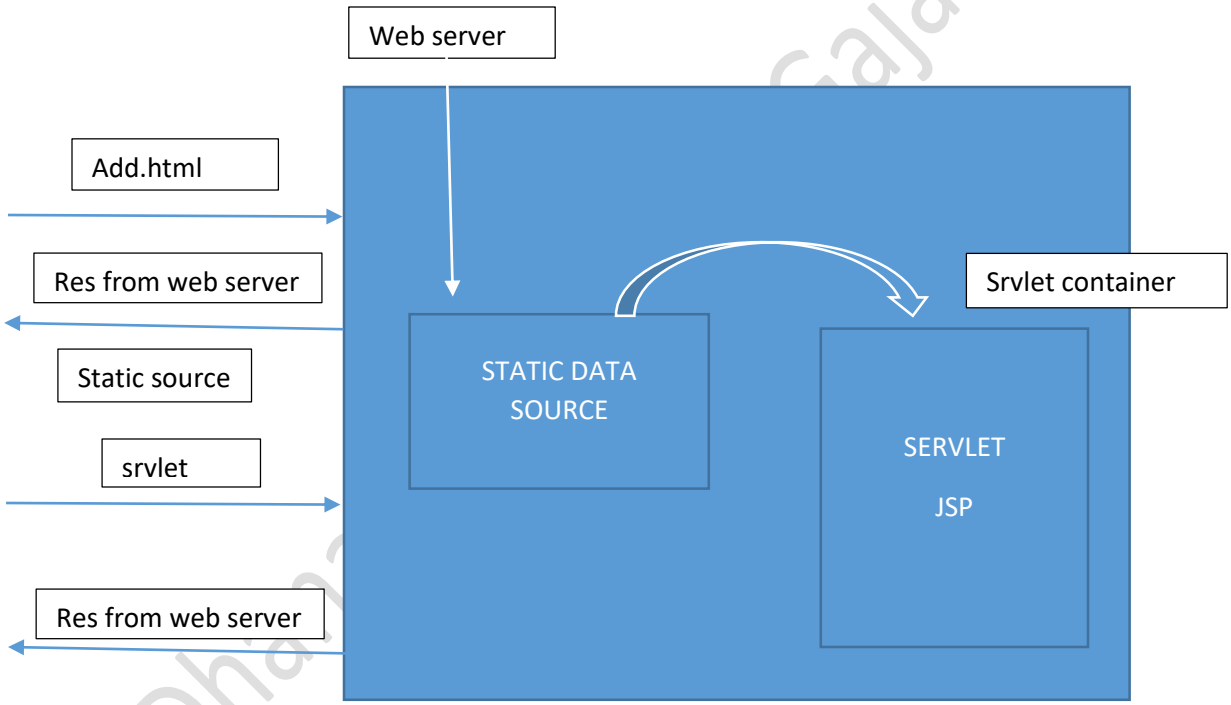
```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Addition</title>
</head>
<body>
	<div align="center">
		<form action="MyServlet7" method="post">
			<table>
				<tr>
					<td>Num1</td>
					<td><input type="text" name="num1"></td>
				</tr>
				<tr>
					<td>Num2</td>
					<td><input type="text" name="num2"></td>
				</tr>
			</table>
			<input type="submit" value="Add">
		</form>
	</div>
</body>
</html>
```

```
package com.jspider.servlet.servlets;

import java.io.IOException;
import java.io.PrintWriter;

import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

@WebServlet("/MyServlet7")
public class MyServlet7 extends HttpServlet {

	private static final long serialVersionUID = 1L;


	@Override
	protected void doPost(HttpServletRequest req, HttpServletResponse
resp) throws ServletException, IOException {
		int num1 = Integer.parseInt(req.getParameter("num1"));
		int num2 = Integer.parseInt(req.getParameter("num2"));
		int sum = num1 + num2;

		resp.setContentType("text/html");
		PrintWriter writer = resp.getWriter();
		writer.println("<h1>"+sum+"</h1>");
	}
}
```

- Note
  - The html pages as to create under web app folder
  - The input name given inside the html document will be considered as parameter inside the servlet

- Difference between doget and dopost

doGet	doPost
• The doget() is called implicitly by the request • doGet() is used when some data has to be retrieve from the server • data send to the url by using doget method will be exposed in the url	• The dopost() has to be called explicitly • doPost() is used when some data has to be send to the server • data send through the url by using dopost method will be hidden in the url



- Servlet LifeCycle
  - Whenever a request is send for particular servlet it will go through some lifecycle phases
    1. Class Loading
       - The servlet class will get loaded
    2. Instantiation
       - Object will be created for that particular servlet
    3. Initialization
       - init() will be invoke only once
    4. Service
       - service() will invoke n times
    5. Destroy
       - destroy() will be invoke once

Student.html

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Insert title here</title>
</head>
<body>
	<div align="center">
		<form action="save" method="post">
			<table>
				<tr>
					<td>Name</td>
					<td><input type="text" name="name"></td>
				</tr>
				<tr>
					<td>Email</td>
					<td><input type="email" name="email"></td>
				</tr>
				<tr>
					<td>SQL</td>
					<td><input type="checkbox" value="SQL"
name="course"></td>
				</tr>
				<tr>
					<td>Core Java</td>
					<td><input type="checkbox" value="Core Java"
name="course"></td>
				</tr>
				<tr>
					<td>WebTech</td>
					<td><input type="checkbox" value="WebTech"
name="course"></td>
				</tr>
			</table>
			<input type="submit" value="Register">
		</form>
	</div>
</body>
</html>
```

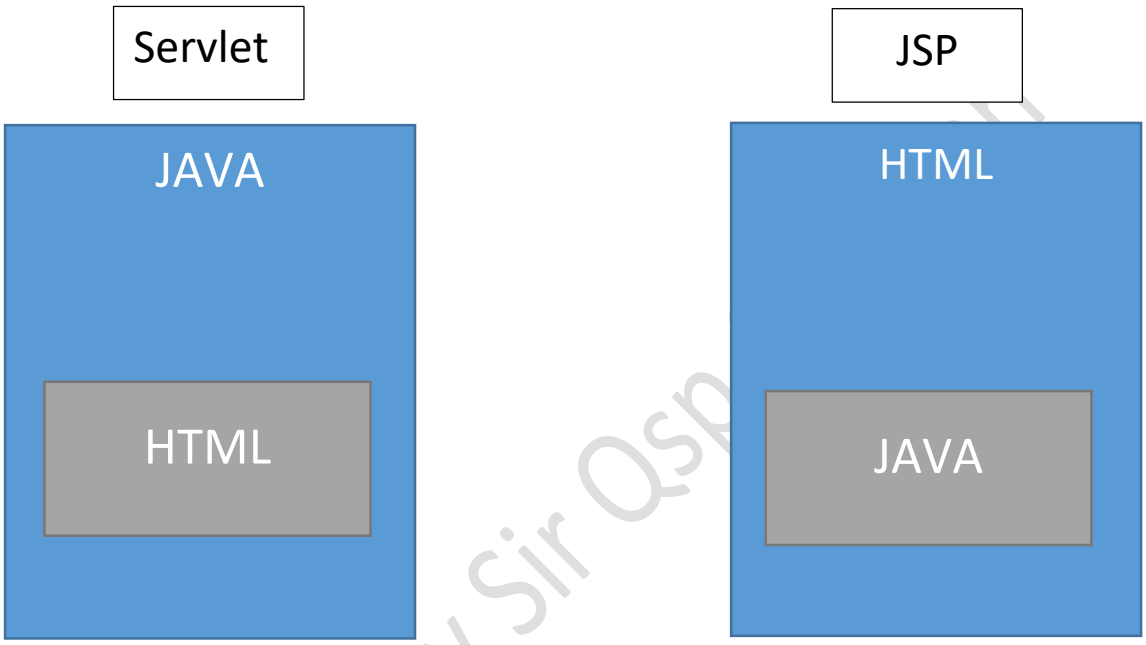
- JSP
  - JSP Stands for jakatra server pages
  - It is used to overcome the drowbacks of servlets
- Drowbacks of Servlet
  - In servlet we need to write complex java code inorder to display even a small HTML contect. so in servlet the focus is move on JAVA rather than HTML

```java
package com.jspider.servlet.servlets;

import java.io.IOException;
import java.io.PrintWriter;

import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

@WebServlet("/save")
public class SaveStudent extends HttpServlet  {


	private static final long serialVersionUID = 1L;

	@Override
	protected void doPost(HttpServletRequest req, HttpServletResponse
resp) throws ServletException, IOException {
		String name=req.getParameter("name");
		String email = req.getParameter("email");
		String[] courses=req.getParameterValues("course");
		resp.setContentType("text/html");
		PrintWriter writer = resp.getWriter();
		writer.println("<h1>Student Info:</h1>");
		writer.println("<h2>" + name + "</h2>");
		writer.println("<h2>" + email + "</h2>");
		writer.println("<h2>Courses:</h2>");
		if (courses != null) {
			for (String course : courses) {
				writer.println("<h2>" + course + "</h2>");
			}
		} else {
			writer.println("<h2>Courses not available</h2>");
		}
	}

}
```

- Note - >
  - These drowbacks caused the servlet technology to become determined
  - JSP is based on servlet technology only but it comes with its own advantages over servlet.
  - In JSP we can focus more on HTML content rather than complex java code
  - We can make use Java language inside JSP file by making use of JSP tags

- These are 5 JSP tags
  1. Scriptlet tag <% %>
  2. Declaration tag <%! %>
  3. Expression tag <%= %>
  4. Difective tag <%@ %>
  5. Action tag :

- <jsp: include page = “” > </jsp: include>
- <jsp: forword page= “”></jsp: forword>



```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Insert title here</title>
</head>
<body>
<div align="center">
	<form action="Add.jsp">
		<table>
			<tr>
				<td>Num1</td>
				<td><input type="text" name="num1"></td>
			</tr>
			<tr>
				<td>Num2</td>
				<td><input type="text" name="num2"> </td>
			</tr>
		</table>
		<input type="submit" value="Add">
	</form>
</div>
</body>
</html>
<%@ page language="java" contentType="text/html; charset=UTF-8"
	pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Insert title here</title>
</head>
<body>
	<%!int num1, num2, sum;%>
	<%
	num1 = Integer.parseInt(request.getParameter("num1"));
	num2 = Integer.parseInt(request.getParameter("num2"));
	sum = num1 + num2;
	%>
	<h1>
		Sum of
		<%=num1%>
		and
		<%=num2%>
		=
		<%=sum%>

	</h1>
</body>
</html>
```

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<jsp:include page="Sub.jsp"></jsp:include>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Insert title here</title>
</head>
<body>
    <%!int num1, num2, sum;%>
    <%
    num1 = Integer.parseInt(request.getParameter("num1"));
    num2 = Integer.parseInt(request.getParameter("num2"));
    sum = num1 + num2;
    %>
    <h1>
        Sum of
        <%=num1%>
        and
        <%=num2%>
        =
        <%=sum%>

    </h1>
</body>
</html>
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>

<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Insert title here</title>
</head>
<body>
    <%!int num1, num2, sub;%>
    <%
    num1 = Integer.parseInt(request.getParameter("num1"));
    num2 = Integer.parseInt(request.getParameter("num2"));
    sub = num1 - num2;
    %>
    <h1>
        The Diffrence Beetween
        <%=num1%>
        and
        <%=num2%>
        is
        <%=sub%>
    </h1>
```

```jsp
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<jsp:forward page="Sub.jsp"></jsp:forward>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Insert title here</title>
</head>
<body>
    <%!int num1, num2, sum;%>
    <%
    num1 = Integer.parseInt(request.getParameter("num1"));
    num2 = Integer.parseInt(request.getParameter("num2"));
    sum = num1 + num2;
    %>
    <h1>
        Sum of
        <%=num1%>
        and
        <%=num2%>
        =
        <%=sum%>

    </h1>
</body>
</html>
```

```jsp
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Insert title here</title>
</head>
<body>
    <%!int num1, num2, sub;%>
    <%
    num1 = Integer.parseInt(request.getParameter("num1"));
    num2 = Integer.parseInt(request.getParameter("num2"));
    sub = num1 - num2;
    %>
    <h1>
        The Diffrence Beetween
        <%=num1%>
        and
        <%=num2%>
        is
        <%=sub%>
    </h1>
```

- implicit Objects
    - The objects which are present in JSP file for use bydefault are called as implicit objects
        1. Eg. Request, response, out
- Life cycle of JSP
    - Every JSP file will be translated to corresponding servlet class
    - After the translation the life cycle of jsp will remain same as that of servlet life cycle
        1. Translation phase
            - Before beginning of life cycle a JSP file is internally translated into servlet file. This is how we can say the JSP technology is based on servlet technology
            - After the translation JSP file can accept web request and generate web response just like servlet life
        2. Class Loading
            - The particular servlet class will be loaded
        3. Instantiation
            - Object will be created for the particular servlet class
        4. Initialization
            - JSP – init() method will be invoke only once
        5. Service
            - JSP-Sevice() method will be invoke for n-time
        6. Destroy
            - JSP-destroy() will be invoke only Once