Advance Java Notes

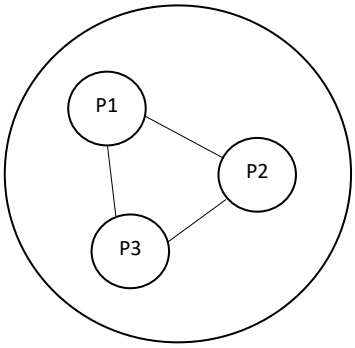
Multi-Threading

Multi-Tasking:

Multi-tasking can be defined as executing more than one tasks simultaneously.

Task:

Task is the end goal to be achieved. Or task is the collection of more than one processes.

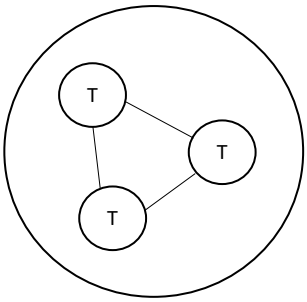


Multi-processing:

Multi-processing can be defined as executing more than one processes simultaneously.

Process:

Process is the collection of more than one threads.



Multi-threading:

Multi-threading can be defined as executing more than one threads simultaneously.

Thread:

Thread is a smallest individual part of a task. Thread can be defined as a light weight process.

Thread is a small program to be executed.

Thread in Java:

Thread is a special class in Java.

Creating a thread in Java:

There are two ways for creating a thread in Java-

1. By extending 'Thread' class.
2. By implementing 'Runnable' interface.

First way of creating Thread

1. **By extending "Thread" class:**
   a. A thread can be created using a 'Thread' class which is present inside 'java.lang' package.
   b. We need to override 'run()' method from 'Runnable' interface.
   c. Thread class is an implementing class of 'Runnable' interface.
   d. In order to execute or run user defined thread, we need to make use of 'start()' method.

   start() method:

   It is a non-static method which is present inside the 'Thread' class.

   Note:

   - A newly created thread will be allocated with dedicated stack area during its execution.
   - If we invoke 'run()' method of user defined thread without making use of 'start()' method, in this case a dedicated stack area will not get allocated for that particular 'run()' method. Instead of that, the 'run()' method will get loaded inside the current stack area.

Mythread.java

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

public class MyThread extends Thread{

	@Override

	public void run() {

	System.out.println("Hello from MyThread");

	}
```

```
}
```

ThreaMain1.java

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

import com.jspiders.multithreading.thread.MyThread;

public class ThreadMain1 {

	public static void main(String[] args) {

		MyThread myThread = new MyThread();

		myThread.start();

	}

}
```

Second way of creating Thread

2. **By implementing 'Runnable' interface:**
   a. The Thread can be created using 'Runnable' interface which is present 'java.lang' package.

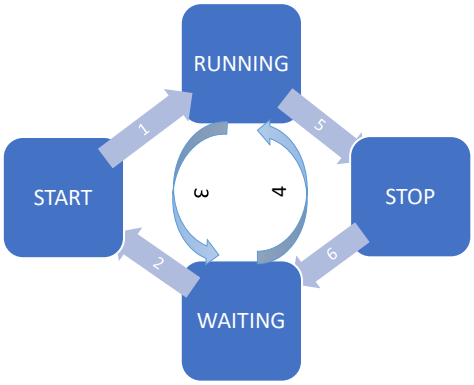
MyThread2.java

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

public class MyThread2 implements Runnable{

	@Override

	public void run() {

	System.out.println("Hello from MyThread2");

	}

}
```

ThreadMain2.java

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

import com.jspiders.multithreading.thread.MyThread2;

public class ThreadMain2 {

	public static void main(String[] args) {

		MyThread2 myThread2 = new MyThread2();

		Thread thread = new Thread(myThread2);

		thread.start();

	}

}
```

Thread Lifecycle:



Thread lifecycle has 4 phases as start, running, waiting, stop. Also, it has 6 transitions as below.

1. Start to Running
2. Start to Waiting
3. Running to Waiting
4. Waiting to Running
5. Running to Stop
6. Waiting to Stop

Thread Scheduler:

It is a component of JVM (Java Virtual Machine) which manages the life cycle of a thread.

1. **Start to Running**
   a. Once object is created for user-defined 'Thread' and 'start()' method is invoked, then the 'Thread' will be in 'START' phase.
   b. Start to Running: When CPU (Central Processing Unit) is assigned to a 'Thread' for its execution, then the 'Thread' will move from 'START' phase to 'RUNNING' phase.

2. **Start to Waiting**
   a. When CPU is not assigned to a 'Thread' for its execution, then the 'Thread' will move from 'START' phase to 'WAITING' phase.

3. **Running to Waiting**
   a. When the assigned CPU taken back from a 'Thread', then that 'Thread' will move from 'RUNNING' to 'WAITING' phase.

4. **Waiting to Running**
   a. When the CPU is assigned back again to a 'Thread' for its execution, then that 'Thread' will move from 'WAITING' to 'RUNNING' phase.

5. **Running to Stop**
   a. When the execution of a 'Thread' is completed. Then that 'Thread' will move from 'RUNNING' to 'STOP' phase.

6. **Waiting to Stop**
   a. If a 'Thread' is waiting for too long time, then that 'Thread' will move from 'WAITING' to 'STOP' phase.

Executing more than one Threads simultaneously

MyThread3.java

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

public class MyThread3 extends Thread{

	@Override

	public void run() {

	    for (int i = 1; i <= 5; i++) {

	    System.out.println("Hello from MyThread3");

	    }

	}

}
```

MyThread4.java

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

public class MyThread4 extends Thread{

	@Override

	public void run() {

	     for(int i = 1; i <=5; i++) {

	     System.out.println("Hello from MyThread4");

	     }

	}

}
```

ThreadMain3.java

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

import com.jspiders.multithreading.thread.MyThread3;

import com.jspiders.multithreading.thread.MyThread4;

public class ThreadMain3 {

	public static void main(String[] args) {

		MyThread3 myThread3 = new MyThread3();

		MyThread4 myThread4 = new MyThread4();

		myThread3.start();

		myThread4.start();

	}

}
```

Thread Properties:

Every 'Thread' created will have some properties as mentioned below:

1. **Id**
   The value for 'Id' property will be assigned by the JVM. A programmer cannot modify the 'Id' value. But a programmer can fetch the 'Id' value using 'getId()' method.

   getId() method:
   It is a non-static method present inside 'Thread' class.

2. **Name**
   This property of a 'Thread' will have default value for a newly created 'Thread'. A programmer can modify the value of this property as well as can fetch the value of this property using 'setName()' , 'getName()' methods respectively.

   setName() method:
   It is a non-static method present inside 'Thread' class.

   getName() method:
   It is a non-static method present inside 'Thread' class.

3. **Priority**
   The default value of the 'Priority' for newly created 'Thread' is 5. The maximum priority value is 10 and the minimum is 1. A programmer can modify the priority value as well as can fetch the priority value using 'setPriority()' , 'getPriority()' methods respectively.

   setPriority() method:
   It is a non-static method present inside 'Thread' class.

   getPriority() method:
   It is a non-static method present inside 'Thread' class.

1. The return type of 'getId()' method is 'long'.
2. The return type of 'getName()' method is 'String'.
3. The return type of 'getPriority()' method is 'int'.
4. The 'setName()' method accepts 'String' type argument.
5. The 'setPriority()' method accepts 'int' type argument.

MyThread5.java

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

public class MyThread5 extends Thread{

	@Override

	public void run() {

		System.out.println("Hello from MyThread5");

		System.out.println("Id = "+ this.getId());

		System.out.println("Name = "+ this.getName());

		System.out.println("Priority = "+ this.getPriority());

	}
}
```

ThreadMain4.java

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

import com.jspiders.multithreading.thread.MyThread5;

public class ThreadMain4 {

	public static void main(String[] args) {

		MyThread5 myThread5 = new MyThread5();

		myThread5.setName("MyThread5");

		myThread5.setPriority(8);

		myThread5.start();

	}

}
```

The allocation of a CPU to a thread is based on certain things as below:

1. The priority of a 'Thread' will take into a consideration.
2. It follows the principle of First Come First Serve (FIFS).
3. It depends on the system configuration.

Account.java

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

public class Account {

	private double accountBalance;

	public void deposit(double amount) {

		System.out.println("Amout of Rupees " + amount

		+ " has been credited...");

		accountBalance += amount;

		System.out.println("Current account balance is " +
accountBalance);
```

```
	}

	public void withdraw(double amount) {

		System.out.println("Amount of Rupees " + amount

		+ " has been debited...");

		accountBalance -= amount;

		System.out.println("Current account balance is " +
accountBalance);

	}

}
```

Husband.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(10000);

		account.withdraw(2000);

	}

}
```

Wife.java

```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);

	}
}
```

AccountMain.java

```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();

		Husband husband = new Husband(account);
```

```
        Wife wife = new Wife(account);

        husband.start();

        wife.start();

    }
}
```

When multiple threads access a shared resource being unaware of each other's operation on a shared resource leads to data inconsistency nothing but we will get undesired outcome.

In the above program, 'Husband' and 'Wife' threads accessing 'Account' class methods being unaware of each other's operation on the methods of 'Account' class which leads to inconsistent final account balance.

Shared Resource:

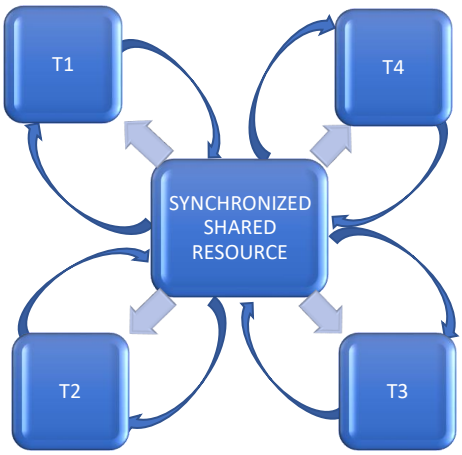
Shared Resource is a resource which is accessed by more than 1 threads. A method or a variable can be referred as a shared resource.

The above problem of data inconsistency can be overcome with the help of synchronization.

Synchronization:

Synchronization can be achieved using synchronized keyword.

If resource is synchronized then a thread which is accessing it will apply a lock on it. So that another thread cannot access that particular resource at the same time.



Types of Locks:

There are 2 types of locks

1. Class lock
2. Object lock

1. **Class lock:**
   If the lock is applied on static synchronized shared resource, then it is called as Class Lock.

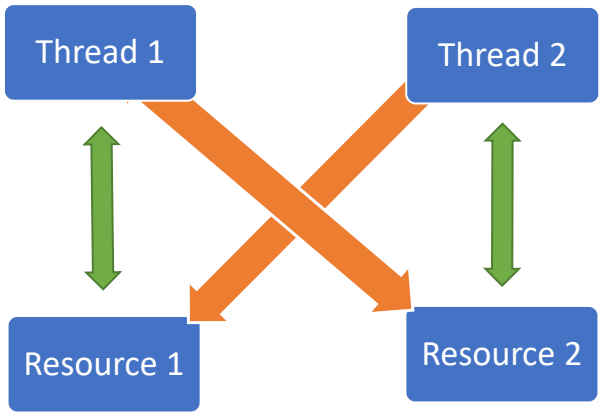
2. **Object lock:**
   If the lock is applied on non-static synchronized shared resource, then it is called as Object Lock.

Account.java

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

public class Account {

	private double accountBalance;

	public synchronized void deposit(double amount) {

		System.out.println("Amout of Rupees " + amount + " has been

		credited...");

		accountBalance += amount;

		System.out.println("Current account balance is " +
```

```
            accountBalance);

    }

    public synchronized void withdraw(double amount) {

        System.out.println("Amount of Rupees " + amount + " has been

        debited...");

        accountBalance -= amount;

        System.out.println("Current account balance is " +

        accountBalance);

    }
}
```

Dead Lock:



In the above diagram, Thread 1 will apply the lock on Resource 1. Similarly, Thread 2 will apply lock on Resource 2.

Now Thread 1 tries to apply lock on Resource 2 at the same time Thread 2 tries to apply lock on Resource 1 which is not possible. Thread 1 will get access to Resource 2 when Thread 2 release the lock on Resource 2 but Thread 2 will not release a lock applied on Resource 2 until and unless it gets access to Resource 1. But Thread 1 will not release a lock applied on Resource 1 until and unless it gets access to Resource 2. This leads to an abnormal situation called as Dead Lock.

MyResource.java

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

public class MyResource {

	public String resource1 = "Resource1";

	public String resource2 = "Resource2";

}
```

MyThread6.java

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

import com.jspiders.multithreading.resource.MyResource;

public class MyThread6 extends Thread{

	private MyResource myResource;

	public MyThread6(MyResource myResource) {

	    this.myResource = myResource;

	}

	@Override

	public void run() {

	    synchronized (myResource.resource1) {

	    System.out.println("Lock is applied on Resource 1 by thread6");

	    synchronized (myResource.resource2) {

	        System.out.println("Lock is applied on Resource 2 by

	        thread6");

	        }

	    }

	}

}
```

MyThread7.java

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

import com.jspiders.multithreading.resource.MyResource;

public class MyThread7 extends Thread{

	private MyResource myResource;

	public MyThread7(MyResource myResource) {

	    this.myResource = myResource;

	}

	@Override

	public void run() {

	    synchronized (myResource.resource2) {

	    System.out.println("Lock is applied on Resource2 by thread7");

	        synchronized (myResource.resource1) {

	        System.out.println("Lock is applied on Resource1 by

	        thread7");

	        }

	    }

	}

}
```

ThreadMain5.java

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

import com.jspiders.multithreading.resource.MyResource;

import com.jspiders.multithreading.thread.MyThread6;

import com.jspiders.multithreading.thread.MyThread7;

public class ThreadMain5 {

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

```
        MyResource myResource = new MyResource();

        MyThread6 myThread6 = new MyThread6(myResource);

        MyThread7 myThread7 = new MyThread7(myResource);

        myThread6.start();

        myThread7.start();

    }
}
```

Methods in Multi-Threading:

1. currentThread()
   It is a static method present inside 'Thread' class which is used to get the object of currently executing thread.

MyResource1.java

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

public class MyResource1 {

    public void resource() {

        System.out.println("Lock is applied on resource by " +

        Thread.currentThread().getName());

    }
}
```

MyThread8.java

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

import com.jspiders.multithreading.resource.MyResource1;

public class MyThread8 extends Thread{

    private MyResource1 myResource1;

    public MyThread8(MyResource1 myResource1) {

        this.myResource1 = myResource1;

    }
```

```
	@Override
	public void run() {
		myResource1.resource();
	}
}
```

MyThread9.java

```
package com.jspiders.multithreading.thread;
import com.jspiders.multithreading.resource.MyResource1;
public class MyThread9 extends Thread{
	private MyResource1 myResource1;
	public MyThread9 (MyResource1 myResource1) {
		this.myResource1 = myResource1;
	}
	@Override
	public void run() {
		myResource1.resource();
	}
}
```

ThreadMain6.java

```
package com.jspiders.multithreading.main;
import com.jspiders.multithreading.resource.MyResource1;
import com.jspiders.multithreading.thread.MyThread8;
import com.jspiders.multithreading.thread.MyThread9;
public class ThreadMain6 {
	public static void main(String[] args) {
		MyResource1 myResource1 = new MyResource1();
```

```
        MyThread8 myThread8 = new MyThread8(myResource1);

        myThread8.setName("MyThread8");

        MyThread9 myThread9 = new MyThread9(myResource1);

        myThread9.setName("MyThread9");

        myThread8.start();

        myThread9.start();

    }
}
```

2. stop()
   It is a non-static method present inside 'Thread' class which is used to terminate or stop the execution of thread or to move a thread from RUNNING to STOP phase forcefully. This method is deprecated means not in use. Because putting a thread form RUNNING to STOP phase forcefully before its complete execution is not recommended.

MyThread10.java

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

public class MyThread10 extends Thread{

    @SuppressWarnings("deprecation")

    @Override

    public void run() {

        for(int i = 1; i <= 5; i++) {

            System.out.println("Hello form MyThread10");

            if(i == 3) {

                stop();

            }

        }

    }
}
```

ThreadMain7.java

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

import com.jspiders.multithreading.thread.MyThread10;

public class ThreadMain7 {

	public static void main(String[] args) {

		MyThread10 myThread10 = new MyThread10();

		myThread10.start();

	}

}
```

3. sleep()
   It is a static method present inside a 'Thread' class which is used to calls the execution of a thread for certain time. This method accepts an argument of type long which has time in milliseconds. For which the execution of a thread will be paused.

MyThread11.java

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

public class MyThread11 extends Thread{

	@Override

	public void run() {

		String message = "Java is a Programming Language.";

		char[] charArray = message.toCharArray();

		for (int i = 0; i < charArray.length; i++) {

			System.out.println(charArray[i]);

	}

	try {

		sleep(500);

	} catch (InterruptedException e) {

		e.printStackTrace();
```

```
        }
    }
}
```

ThreadMain8.java

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

import com.jspiders.multithreading.thread.MyThread11;

public class ThreadMain8 {

    public static void main(String[] args) {

        MyThread11 myThread11 = new MyThread11();

        myThread11.start();

    `}
}
```

4. wait()
   It is a non-static method present inside 'Object' class. It is used to pause the execution of a thread.
5. notify()
   It is a non-static method present inside 'Object' class. It is used to resume the execution of a thread.
6. notifyAll()
   It is a non-static method present inside 'Object' class. It is used to resume the execution of all the threads.
   All the threads which having put in waiting using wait() method.

Food.java

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

public class Food {

private boolean available;

public synchronized void order() {

System.out.println("Order is received");

if(available) {

System.out.println("Order delivered.");
```

```
}
else {
System.out.println("Food is not available");
System.out.println("wait for some time");
}
try {
wait();
} catch (InterruptedException e) {
e.printStackTrace();
}
if(available) {
System.out.println("Order is Delivered.");
}
}
public synchronized void prepare() {
System.out.println("Food is getting prepared.");
System.out.println("Food is prepared.");
available = true;
notify();
}
}
```

Customer.java

```
package com.jspiders.multithreading.thread;
import com.jspiders.multithreading.resource.Food;
public class Customer extends Thread{
private Food food;
```

```
public Customer (Food food) {

this.food = food;

}

@Override

public void run() {

food.order();

}

}
```

Restaurant.java

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

import com.jspiders.multithreading.resource.Food;

public class Restaurant extends Thread{

private Food food;

public Restaurant (Food food) {

this.food = food;

}

@Override

public void run() {

food.prepare();

}

}
```

ThreadMain9.java

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

import com.jspiders.multithreading.resource.Food;

import com.jspiders.multithreading.thread.Customer;

import com.jspiders.multithreading.thread.Restaurant;
```

```
public class ThreadMain9 {

public static void main(String[] args) {

Food food = new Food();

Customer customer = new Customer(food);

customer.start();

Restaurant restaurant = new Restaurant(food);

restaurant.start();

}

}
```

Difference between wait() and sleep():

Wait() method	Sleep() method
It is a non-static method present inside 'Object' class.	It is a static method present inside 'Thread' class.
Wait() method will not accept any argument.	Sleep() method will accept an argument of type long i.e. time in milliseconds.
If an execution of a thread is paused using wait() method then in order to resume its execution, we need to make use of notify() or notifyAll() methods.	If an execution of a thread is paused using sleep() method then the execution of a paused thread will be resumed after specified time.

Daemon Thread:

A thread can be of 2 types.

1. User created thread
2. In-built or pre-defined thread.

Demon thread is a pre-defined, low priority thread which will be executed by the JVM whenever required.

Garbage Collector is the important Demon thread used in Java. It is used to manage the memory efficiently since it removes de-referred or nullified objects from the heap memory.

De-referred objects:

Demo demo = new Demo();

demo = new Demo();

Nullifying the Object:

demo = null;

Advantages of Garbage Collector:

1. Garbage collector will take care of wasted memory. Since it removes non-accessible objects from the memory. And makes Java strong in memory management.
2. Programmer did not to worry about memory wastage or efficient memory utilization.

File Handling

File is an entity we can store the data and we can manipulate and can organize the data according to the requirement.

1. **Source File**
   If a data is getting fetched from the file. In this case, file will act as a source file.

2. **Target file**
   If the data is getting written to the file. In this case, file will act as a target file.

Performing CRUD operations on the file is called as File Handling.

Operations which can be performed on the file are listed below:

1. Creating a new file
2. Deleting a file
3. Fetching information about the file
4. Writing data to the file
5. Reading data from the file

1. **Creating a new file**
   CreateNewFile.java

```java
package com.jspiders.filehandling.operation;

import java.io.File;

import java.io.IOException;

public class CreateNewFile {

	public static void main(String[] args) {

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

		try {

			boolean status = file.createNewFile();

			if (status) {

				System.out.println("File Created successfully");

			}else {

				System.out.println("File already exists.");
```

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

Note: in order to perform any operation on the file, we need an object of 'File' class present inside 'java.io' package.

The constructor of 'File' class will accept one String type argument that is location of the file along with the name of the file with its extension.

If we mention only the file name along with extension then default location will be considered i. e. the 'Project' folder this is called as Default path.

We can mention the exact location for the file which is called as Absolute Path.

CreateNewFile.java

```
package com.jspiders.filehandling.operation;

import java.io.File;

import java.io.IOException;

public class CreateNewFile1 {

    public static void main(String[] args) {

        File file = new File("D:\\File\\Demo.txt");

        try {

            boolean status = file.createNewFile();

            if (status) {

                System.out.println("File is created");

            }else {

                System.out.println("File already exists");

            }
```

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

createNewFile():

It is a non-static method present inside 'File' class. It returns 'true' if the file is not present at specified location. Otherwise, it returns 'false'.

2. **Deleting a file**

   DeleteFile.java

```
package com.jspiders.filehandling.operation;

import java.io.File;

public class DeleteFile {

      public static void main(String[] args) {

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

          if (file.exists()) {

              boolean status = file.delete();

if (status) {

System.out.println("File is deleted");

}else {

System.out.println("File is not deleted");

}

}else {

System.out.println("File does not exist");

}
```

```
}

}
```

delete():

It is a non- static method present inside ‘File’ class. It is used to delete the file. It returns two values if the file is deleted successfully then it returns ‘true’ otherwise, it returns ‘false’.

exist():

It is a non-static method present inside ‘File’ class. it is used to check whether file is present at specified location or not.

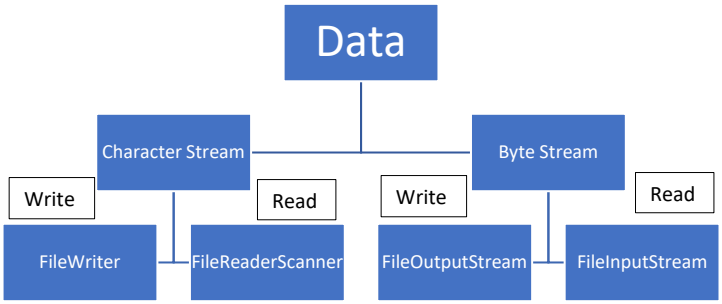
Fetching information about the file:

FileInfo.java

```
package com.jspiders.filehandling.operation;

import java.io.File;

public class FileInfo {

public static void main(String[] args) {

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

if (file.exists()) {

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

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

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

if (file.canWrite()) {

System.out.println("File is writable");

}else {

System.out.println("File is not writable");

}

if (file.canRead()) {
```

```
System.out.println("File is Readable");

}else {

System.out.println("File is not readable");

}

if (file.canExecute()) {

System.out.println("File is executable");

}else {

System.out.println("File is not executable");

}

}else {

System.out.println("File does not exist.");

}

}

}
```

Reading and writing operations on the File:



CharStreamWrite.java

```
package com.jspiders.filehandling.operation;

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("D:/File/Demo.txt");

if (file.exists()) {

FileWriter fileWriter = new FileWriter(file);

fileWriter.write("Java is a high level programming language.");

System.out.println("Data is written to the file");

fileWriter.close();

}else {

boolean status = file.createNewFile();

if (status) {

System.out.println("File is created");

FileWriter fileWriter = new FileWriter(file);

fileWriter.write("Java is a high level programming language");

System.out.println("Data is written to the file");

fileWriter.close();

}else {

System.out.println("File is not created");

}

}

}

}
```

ByteStreamWrite.java

```
package com.jspiders.filehandling.operation;

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("D:/File/Demo.txt");

          if (file.exists()) {

          FileOutputStream fileOutputStream = new FileOutputStream(file);

              fileOutputStream.write(1244);

              System.out.println("Data is written to the file");

              fileOutputStream.close();

          }else {

              boolean status = file.createNewFile();

              if (status) {

                  System.out.println("File is created");

          FileOutputStream fileOutputStream = new FileOutputStream(file);

                  fileOutputStream.write(1244);

                  System.out.println("Data is written to the file");

                  fileOutputStream.close();

      }else {

System.out.println("File is not created");

}

}

}

}
```

CharStreamRead.java

```
package com.jspiders.filehandling.operation;

import java.io.File;
```

```java
import java.io.FileOutputStream;

import java.io.IOException;

public class ByteStreamWrite {

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

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

if (file.exists()) {

FileOutputStream fileOutputStream = new FileOutputStream(file);

fileOutputStream.write(1244);

System.out.println("Data is written to the file");

fileOutputStream.close();

}else {

boolean status = file.createNewFile();

if (status) {

System.out.println("File is created");

FileOutputStream fileOutputStream = new FileOutputStream(file);

fileOutputStream.write(1244);

System.out.println("Data is written to the file");

fileOutputStream.close();

}else {

System.out.println("File is not created");

}

}

}

}
```

ByteStreamRead.java

```java
package com.jspiders.filehandling.operation;
```

```java
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("D:/File/Demo.txt");

if (file.exists()) {

FileInputStream fileInputStream = new FileInputStream(file);

System.out.println(fileInputStream.read());

fileInputStream.close();

System.out.println("Data is fetched from the file");

}else {

System.out.println("File does not exist");

}

}

}
```

getName():

It is a non-static method present inside a ‘File’ class which is used to fetch the name of the file.

getAbsolutePath():

It is a non-static method present inside a ‘File’ class which is used to fetch exact location of the file.

length():

It is a non-static method present inside ‘File’ class which is used to fetch number of characters present inside a file.

canWrite():

It is a non-static method present inside the ‘File’ class it returns ‘True’ if the file is writable otherwise ‘False’.

canRead():

It is a non-static method present inside the ‘File’ class. It returns ‘True’ if the file is readable otherwise ‘False’.

canExecute():

It is a non-static method present inside the ‘File’ class. It returns ‘True’ if the file is executable otherwise ‘False’.

FileWriter:

It is a concrete class present inside ‘java.io’ package.

FileWriter class is the child class of ‘OutputStreamWrite’ class which is the child class of ‘Writer’ class which is abstract in nature.

FileOutputStream:

This is a concrete class present inside ‘java.io’ package.

‘FileOutputStream’ class is the child class of ‘OutputStream’ class which is abstract in nature.

FileReader:

It is a concrete class present inside ‘java.io’ package.

It is a child class

FileInputStream:

It is a concrete class present inside ‘java.io’ package.

It is a child class of InputStream Class which is abstract in nature.

Serialization and De-serialization:

Serialization:

It is the process of converting ‘Java object’ into ‘byte code‘ format.

De-serialization:

It is the process of converting the data in the ‘byte code’ format into original ‘Java object’.

Need for serialization and de-serialization:

When two ‘Java’ applications communicate with each other over a network, then the ‘Java’ object form the application has to be converted into ‘Network supported format’ in order to make it travel over a network. The data which is travelling over a network has to be converted in its original form i. e. ‘Java object’ in order to receive it inside a ‘Java’ application.

Note:

In above case, we are considering ‘byte code’ format as a ‘Network Supported Format’.

Serializable Interface:

It is a marker interface used to mark a class as ‘serializable’. It means objects of that class are now eligible for ‘Serialization’ and ‘De-serialization’ process.

Marker Interface:

It is an empty interface used to mark a class for specific task.

User.java

```
package com.jspiders.serializationanddeserialization.object;

import java.io.Serializable;

public class User implements Serializable{

	private static final long serialVersionUID = 1L;

	private int id;

	private String name;

	private String email;

	private long mobile;

	public User(int id, String name, String email, long mobile) {

		this.id = id;

		this.name = name;

		this.email = email;

		this.mobile = mobile;

	}

	@Override

	public String toString() {

		return "User [id=" + id + ", name=" + name + ", email=" + email

		+ ", mobile=" + mobile + "]";

	}

}
```

Serialization.java

```java
package com.jspiders.serializationanddeserialization.serialization;

import java.io.File;

import java.io.FileOutputStream;

import java.io.IOException;

import java.io.ObjectOutputStream;

import com.jspiders.serializationanddeserialization.object.User;

public class Serialization {

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

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

		FileOutputStream fileOutputStream = new FileOutputStream(file);

		ObjectOutputStream objectOutputStream = new

		ObjectOutputStream(fileOutputStream);

		objectOutputStream.writeObject(new User(1, "Ram",

		"ram@gmail.com", 986645345611));

		System.out.println("Object has been serialized.");

		objectOutputStream.close();

		fileOutputStream.close();

	}

}
```

Deserialization.java

```java
package com.jspiders.serializationanddeserialization.serialization;

import java.io.File;

import java.io.FileInputStream;

import java.io.IOException;

import java.io.ObjectInputStream;

import com.jspiders.serializationanddeserialization.object.User;
```

```java
public class Deserialization {

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

	ClassNotFoundException {

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

		FileInputStream fileInputStream = new FileInputStream(file);

		ObjectInputStream objectInputStream = new

		ObjectInputStream(fileInputStream);

		User user = (User) objectInputStream.readObject();

		System.out.println(user);

		System.out.println("Object is deserialized");

		objectInputStream.close();

		fileInputStream.close();

	}
}
```

Design Patterns

Design patterns are the pre-defined or existing programming structures which can be used to meet the requirements or to avoid the recurring development issues w. r. t. application development.

Design patterns catalog:

Design patterns catalog consist of different design patterns discovered by developers.

If any developer comes up with new design pattern, then it can be added to the design patterns catalog.

Based on the requirement or the issues with which a design pattern deals with, design patterns are classified into different types.

Widely used design patterns are as below:

1. Creational design patterns
2. Structural design patterns

1. **Creational design patterns:**
   Creational design patters deal with the requirements or the issues related to object creation process in the application.

   Widely used creational design patterns are-
   a. Singleton design pattern
   b. Factory design pattern
   c. Builder design pattern

   a. Singleton Design Patterns:
   Singleton design pattern is used to restrict user from creating multiple objects for the same class.
   Singleton design pattern is classified into two types.

   i. Singleton Lazy
   In this case, the object will be created for the class when the user will request for it.

   ii. Singleton Eager
   In this case, the object will be created for the class during class loading process itself even though the user is not requesting for the object.

SingletonLazy.java

```
package com.jspiders.designpatterns.creational;

public class SingletonLazy {

	private static SingletonLazy singletonLazy;

	private SingletonLazy() {

		System.out.println("Constructor is invoked");
```

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

SingletonLazyMain.java

```
package com.jspiders.designpatterns.main;
import com.jspiders.designpatterns.creational.SingletonLazy;
public class SingletonLazyMain {
    public static void main(String[] args) {
        SingletonLazy singletonLazy1 = SingletonLazy.getObject();
        System.out.println(singletonLazy1);
        SingletonLazy singletonLazy2 = SingletonLazy.getObject();
        System.out.println(singletonLazy2);
        SingletonLazy singletonLazy3 = SingletonLazy.getObject();
        System.out.println(singletonLazy3);
        SingletonLazy singletonLazy4 = SingletonLazy.getObject();
        System.out.println(singletonLazy4);
    }
}
```

SingletonEager.java

```
package com.jspiders.designpatterns.creational;
public class SingletonEager {
```

```
	private static SingletonEager singletonEager = new SingletonEager();

	private SingletonEager() {

		System.out.println("Constructor is invoked");

	}

	public static SingletonEager getObject() {

		return singletonEager;

	}
}
```

SingletonEagerMain.java

```
package com.jspiders.designpatterns.main;

import com.jspiders.designpatterns.creational.SingletonEager;

public class SingletonEagerMain {

	public static void main(String[] args) {

		SingletonEager singletonEager1 = SingletonEager.getObject();

		System.out.println(singletonEager1);

		SingletonEager singletonEager2 = SingletonEager.getObject();

		System.out.println(singletonEager2);

		SingletonEager singletonEager3 = SingletonEager.getObject();

		System.out.println(singletonEager3);

		SingletonEager singletonEager4 = SingletonEager.getObject();

		System.out.println(singletonEager4);

	}
}
```

In above programs we have achieved singleton design pattern by following steps.

Step 1: We have made the constructor of a class ‘private’ in nature. In order to restrict its access outside the class.

Step 2: But in order to access that private constructor outside the class indirectly, we have defined one helper method which returns the object of that particular class.

b. **Factory design patterns**

Factory design patters is used to create the objects in the application whenever there is a requirement for that object.

In the application, we have multiple classes for which we need to create the objects. If we create the objects for classes in application and we give them ready for the use then there is a chance that some of the objects will remain unused which leads to memory wastage. In order to avoid this issue in application and to make our application more efficient in memory utilization, we can make use of factory design pattern where objects will be created whenever they are required.

Beverages.java

```
package com.jspiders.designpatterns.creational;

public interface Beverages {

	void order();

}
```

MasalaTea.java

```
package com.jspiders.designpatterns.creational;

public class MasalaTea implements Beverages{

	@Override
	public void order() {

	    System.out.println("Thank You! Masala Tea is ordered...");

	}

}
```

GreenTea.java

```
package com.jspiders.designpatterns.creational;

public class GreenTea implements Beverages{

	@Override
	public void order() {
```

```
            System.out.println("Thank You! Green Tea is ordered...");
        }
}
```

LemonTea.java

```
package com.jspiders.designpatterns.creational;

public class GreenTea implements Beverages{

        @Override
        public void order() {
            System.out.println("Thank You! Green Tea is ordered...");
        }
}
```

GingerTea.java

```
package com.jspiders.designpatterns.creational;

public class GingerTea implements Beverages{

        @Override
        public void order() {
            System.out.println("Thank You! Ginger Tea is ordered...");
        }
}
```

FactoryMain.java

```
package com.jspiders.designpatterns.main;

import java.util.Scanner;

import com.jspiders.designpatterns.creational.Beverages;

import com.jspiders.designpatterns.creational.GingerTea;
```

```java
import com.jspiders.designpatterns.creational.GreenTea;

import com.jspiders.designpatterns.creational.LemonTea;

import com.jspiders.designpatterns.creational.MasalaTea;

public class FactoryMain {

	public static void main(String[] args) {

		try {

			factory().order();

		} catch (NullPointerException e) {

			System.out.println("No Tea Ordered...");

			e.printStackTrace();

		}

	}

	@SuppressWarnings("resource")

	private static Beverages factory() {

		Beverages beverages = null;

		System.out.println("Enter 1 to order Masala Tea\nEnter 2 to
order Green Tea\nEnter 3 to order Lemon Tea\nEnter 4 to order Ginger
Tea");

		System.out.println("Enter Your Choice!!!");

		Scanner scanner = new Scanner(System.in);

		int choice = scanner.nextInt();

		switch (choice) {

		case 1:

			beverages = new MasalaTea();

		break;

		case 2:

			beverages = new GreenTea();
```

```
            break;

            case 3:

                beverages = new LemonTea();

            break;

            case 4:

                beverages = new GingerTea();

            break;

            default:

                System.out.println("Invalid Choice!!!");

            break;

        }

        return beverages;

        }
}
```

c. Builder design pattern

Builder design pattern is used to create complex object in the application.

Complex object:

The object which has too many properties is called as complex object.
Issues we face while creating complex object inside the application using all arguments constructor.

i. We need to remember all the properties of an object.
ii. We must know the type of each property of an object.
iii. We must know the sequence in which the constructor is accepting the arguments.
iv. We must pass all the arguments to create complex object.

In order to avoid issues while creating complex object, we can make use of Builder Design Patterns. Where we have to make use of 1 helper class which will help us to create complex objects. This intermediate class is aware of all the properties of complex object.

2. Structural Design Patterns:

Structural Design Patterns deal with the requirements or the issues associated with structure of a class or an interface in the application.
Widely used Structural design patterns is Adapter Design Pattern.

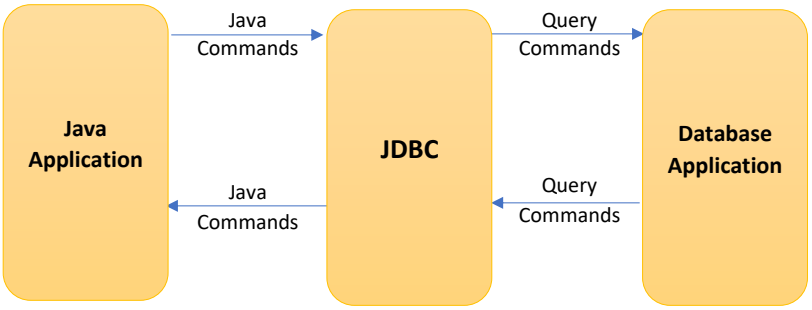
Adapter Design Pattern:
Adapter design pattern is used to build indirect relationship between two entities. We have to make use of an adapter entity (class) which will inherit the properties of above two entities leads to building indirect relationship between those two entities. Same design pattern can be used to combine the properties of two entities together without establishing the direct relationship between those two entities.

Q1. WAJP to demonstrate synchronization in multi-threading.
Q2. WAJP to serialize and deserialize a java object.

JDBC (Java Database Connectivity)

- JDBC stands for Java Database Connectivity.
- It is a technology used to connect Java application with the Database application.
- It is the base technology or the only technology present to connect any Java application with any Database application.
- Since the Java application and the Database application both are incompatible with each other. As the Java application understands Java commands and the Database application understands Query commands.

- JDBC technology will act as the mediator between the Java application and the Database application. In order to make them communicate with each other, as JDBC understands both Java commands as well as Query commands.

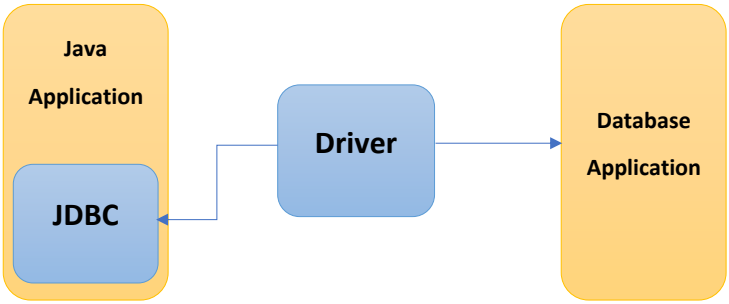


Need for connecting Java application with the Database application:

- Inside the Java application, the data cannot be stored permanently. In order to fetch that data in future, the data has to be stored permanently. The database is only place where we can store the data permanently and also, we can fetch it or manipulate it whenever required.
- JDBC is also called as a Java API (Application Programming Interface)

API:

- API stands for Application Programming Interface.
- API is a software which will help two different applications communicate with each other.
- Since JDBC is the part of Java application, it cannot directly communicate with Database application. So, JDBC appoints one driver to communicate with Database application.



- ‘java.sql’ package represents JDBC API.

Note:
Each and every Database application has dedicated driver which can be used to communicate with them.

Driver:
Driver is a software which is not the part of Java application. It is the external resource which has to be integrated (added) with the Java application.

Steps to download MySQL connector:

1. Open the browser and search for Maven Repository.
2. Click on the link of Mavan Repository.
3. In the search bar, type MySQL Connector and then click on Search.
4. Click on the option as MySQL Connector/J.
5. Click on the appropriate version of MySQL Connector.
6. In the files section, click on the ‘**.jar**’ file, so MySQL Connector will be downloaded.

Steps to use JDBC in a Java Program:

1. Load and register the driver.
2. Open the connections between Java application and Database application.
3. Create or prepare the statement.
4. Execute the statement.
5. Process the result.
6. Close the connections between Java application and Database application.
7. De-register the driver.

Steps to add MySQL Connector (.jar file) to the java project:

1. Right click on the project.
2. Go to Build Path> Click on Configure Build Path… option.
3. Under Libraries section initially only in-built library will be present.
4. Now click on add external JAR option and then browse to the specific location where downloaded MySQL Connector file is present.
5. Select the MySQL Connector and then click on Open.
6. Now, the .jar file will be added to Libraries section then click on Apply and Close.

db_url (Database url):
Format:
API : dataset_app : // host_name : port_Number / database_name
Ex.
jdbc:mysql://localhost:3305/weja4

Methods used to execute the statements:

1. **execute():**

it returns 'Boolean' value. If we execute DQL (Data Query Language) statements then it returns 'true' otherwise it returns 'false'.

Note:
We can execute any type of command using execute().

2. **executeUpdate():**
   it returns 'integer' value. We can execute only DML (Date Manipulation Language) statements using executeUpdate().
   The 'integer' value return by the executeUpdate() represents number of rows affected after executing DML statements.

3. **executeQuery():**
   it is used to execute only DQL (Data Query Language) statements.
   The return type of executeQuery() is 'resultSet' object.

Task:
Wajp to insert 5 records of a student's inside table student using statement:
Id, name, email, age

1. **Ramesh**
2. **Suresh**
3. **Ganesh**
4. **Ajay**
5. **Vijay**

WAJP to display information of the student whose name contains "esh".
WAJP to display information of the students whose name contains "jay".

Statement Hierarchy:

Diff between Statement and PreparedStatement:

Statement	PreparedStatement
It is a parent interface of PreparedStatement interface.	It is a child interface of Statement interface.
Statement is used to execute static queries.	PreparedStatement is used to execute dynamic queries.
Statement is less secure than PreparedStatement.	PreparedStatement is more secure than Statement.
Statement is less efficient than PreparedStatement.	PreparedStatement is more efficient than Statement.

Stored Procedure:
It is a set of queries to be executed. Like methods in Java, we have stored procedures in SQL.
Syntax:

```
delimiter /
create procedure procedure_name()
begin
statements;
end
delimiter ;
```

CallableStatement:
CallableStatement is used to call stored procedures for their execution from the Java application.

Hibernate JPA

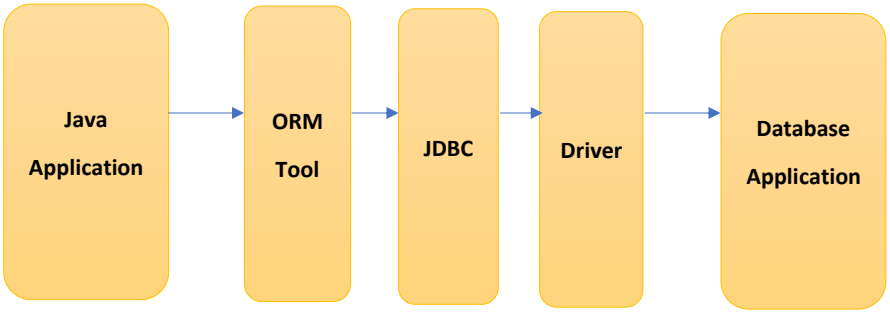
Major Drawbacks of JDBC:

1. As a Java developer, we are forced to write SQL queries.
2. We have to remember database properties as well as properties of a table.
3. In order to perform any operation on the database, we have to write a boiler plate code.
4. We cannot achieve direct relationship between Java object and the data in the database.

These drawbacks can be overcome using ORM (Object Relational Mapping) tools.
ORM Tools:

- ORM stands for Object Relational Mapping.
- ORM tool is used to build a direct relationship between Java object and data in the Database.
- A java class will be mapped as table in the database and Java objects will be mapped as tuple (records) in the database.

Note:
ORM tools used or implemented based on JDBC technology itself.



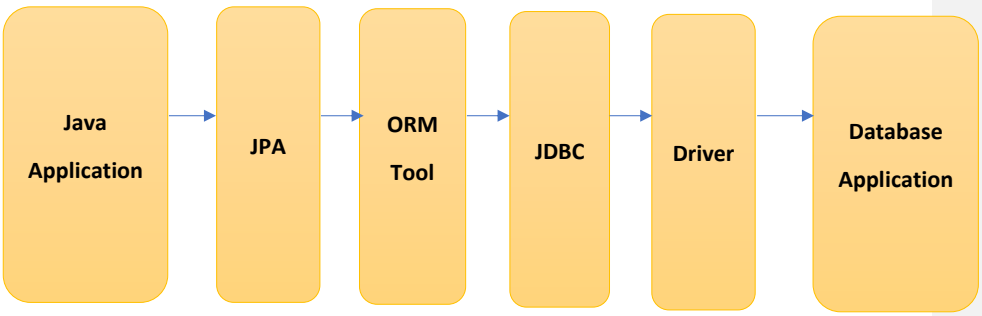
Different ORM tools available in the market.

1. Hibernate
2. Eclipse Link
3. Top Link
4. I Batis/ My Batis

- Hibernate ORM tool is an open source as well as feature rich ORM tool.
- Like Hibernate we have different ORM tools available in the market which are incompatible with each other.
- That makes migration from one to another ORM tool difficult.
- In order to overcome this problem, JPA (Java Persistance API) is implemented.

JPA (Java Persistance API):

- It is a Java specifications which will set common standards for all the ORM tools which eliminates the difficulty in migration from one to another ORM tool.
- It is the responsibility of one of the ORM tools to implement the JPA.
- Here Hibernate ORM tool is implementing JPA so it is called as Hibernate JPA.



Hibernate Layers:

1. DTO
2. DAO

1. **DTO**
   DTO stands for Data Transfer Object.
   This layer is responsible to hold entities in the application.

2. **DAO**
   DAO stands for Data Access Object.
   This layer is responsible to hold Hibernate logic.

Note:

Hibernate layers will be represented as packages in the application.

Steps to create Maven project (archetype-quickstart):

1. Press ctrl + N, in the select wizard search for Maven project.
2. Select the Mavan project option and click on Next.
3. In the next window, do not skip the archetype selection then click on Next.
4. In the next window, select All catalogs then in Filter section type 'org.apache.maven' and then select 'maven-archetype-quickstart' with the version 1.4. after that click on Next.
5. Provide the group-id and artifact-id for the project. And then untick a checkbox 'run archetype generation interactively'. Then click on Finish.

Dependencies required for Hibernate JPA:

1. MySQL Connector/J
2. Hibernate Core Relocation
3. Project Lombok (optional)

pom.xml (project object model):

pom stands for Project Object Model. 'pom.xml' is used to add external dependencies to the project.

Steps to create 'persistent.xml' file:

1. Select the project name and press Ctrl + N to create one source folder under the project.

2. In a select wizard, search for Source Folder. Then select the Source Folder and name it as 'src/main/resources'.
3. Create a General folder under 'src/main/resources' source folder and name it as 'META-INF'.
4. Under 'META-INF' create one '.xml' file with the name 'persistence.xml'.

Annotations used in Hibernate JPA project:

1. @Data
2. @Entity
3. @Id

1. **@Data**
   This annotation is present inside Lombok package. It is used to add getters, setters, toString(), hashCode(), equals(), etc. to the class. It is a class level annotation.

2. **@Entity**
   This annotation is present inside 'javax.persistence' package. It is a class level annotation used to mark a class as an Entity.

3. **#Id**
   This annotation is present inside 'javax.persistence' package. It is a property level annotation used to mark that property for 'primary key' constraint inside the table.

Notes:

1. Properties of a class will be mapped as columns in the table.
2. 'javax.persistence' package represents JPA.
3. Hibernate JPA provides abstraction layer to perform CRUD operations on the Java Objects.

JPQL:

- JPQL stands for Java Persistence Query Language.
- JPQL is used to write customized queries.
- As different ORM tools understands different query languages like Hibernate understands Hibernate Query Language, My-Batis/ I-Batis understands Data Query Language.
- So, all the query languages are incompatible to each other.
- JPQL is understandable by all the ORM tools using which we can overcome the problem of incompatibility among different-different query languages.

Note:

JPQL is similar to SQL itself but in JPQL, we will use class name instead of table name and properties instead of column names.

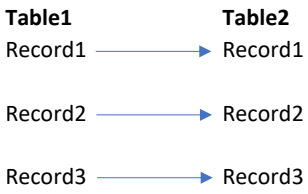
Hibernate Mapping:

Hibernate Mappings are used to establish relationship between two entities.

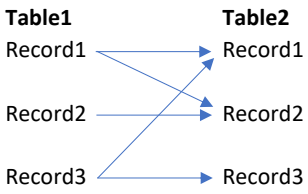
There are 4 types of hibernate mapping:

1. One to One
2. One to Many
3. Many to One
4. Many to Many

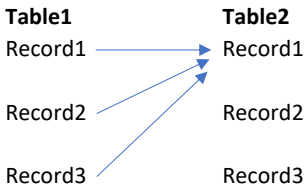
1. **One to One Mapping:**
   One to One mapping can be defined as if each record of table one is related to exactly one record of table two is called as one to one mapping.



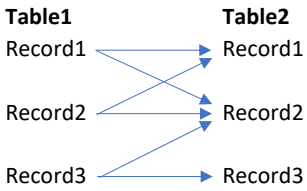
2. **One to Many Mappings:**
   One to Many mappings can be defined as if each record of table one is related to more than one records of table two then it is called as one to many mappings.



3. **Many to One Mappings:**
   Many to One mapping can be defined as if more than one records of table one is related to exactly one record of table two then it is called as Many to One mapping.



4. **Many to Many Mappings:**
   Many to Many mappings can be defined as if more than one records of table one are related to more than one records of table two then it is called as Many to Many mappings.



Unidirectional Mapping:

If the relationship is defined in either one of entities then it is called as Unidirectional mapping.

Bi-directional Mapping:

If the relationship is defined in both the entities then it is called as Bi-directional mapping.

Some of the miscellaneous annotations used in Hibernate JPA:

1. **@GeneratedValue**
   This annotation is used to generate the values for ID property of an Entity class.

2. **@Table**
   This annotation is used to modify the table properties.

3. **@Column**
   This annotation is used to modify column properties.

Dependent Class:

The class inside which the relationship is defined is called as Dependent Class. Because if we want to create an object for this class then we have to create the object for a class which will be the property of dependent class.

Dependency Class:

The class whose object is required to create the object of dependent class is called Dependency Class.

Note:

- Dependent class is also called as Source Entity.
- Dependency class is also called as Target Entity.

Note:

In many-to-many bi-directional mapping, one redundant mapping table will get created which can be eliminated using 'mappedBy' attribute.

Servlets

Web Application or Web-Based application:

Web applications are those applications which are present inside distant systems (servers). And which can be accessed using standard web browsers.

Web Browser:

Web browser is an application which is used to generate web request for particular resource or to accept web response.

Web Resource:

Web resource is the information or the data present on the web or internet.

Static Web Resource:

Static web resource can be defined as the data or the information which will not change based on the user is called as static web resource.

Dynamic Web Resource:

It can be defined as the data or the information changes based on user is called as dynamic web resources.

Static Web Application:

The web application which contains static web resources is called as Static Web Application.

Dynamic Web Application:

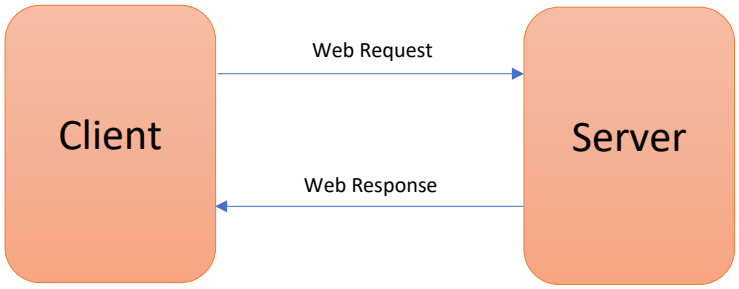
The web application which contains dynamic web resources is called as Dynamic Web Applications.

Note:

Using HTML, we can develop static web resources.

Using Servlets and JSP's, we can develop dynamic web resources.

Client-Server Architecture:



Web Request:

If the client is sending a request for particular web resource available on the web using standard web browsers is called as Web Request.

Web Response:

The data or the information or a message sent back by the server to the client for particular web request is called as Web Response.

Servlet:

- Servlet is the technology which is used to accept web request inside a Java application and to generate web response from the Java application.
- Servlet is also called as Java API.
- We can send a web request for particular resource present inside a Java application using URL.

URL:

URL stands for Uniform Resource Locator which is used to locate or to identify web resource available on the web.

URL Format:

protocol**: //** host_name **:** port_number**/**path_to_resource**?**Query_String**#**Fragment_id

Commented [JP1]: URI

Protocol:

It is the set of rules and regulations to be followed in order to transfer an information over the web.

Host Name:

It defines the location of the server inside which an application is running.

Port Number:

It defines the exact location of an application.

Path to resource:

It defines the name or identifying factor of particular web resource.

Query String:

It is used to send some data from client to the server.

Fragment ID:

It defines the unique ID value associated with each and every web resource.

Steps to create Dynamic Web Project:

1. Press Ctrl + N then search for Dynamic Web Project. Select the Dynamic Web Project and click on Next >.
2. The Next window, provide the Project Name and click on Next >.
3. In the Next window, we can see the source folders available in a project, without any modification click on Next >.
4. In the Next window, Tick the checkbox saying “Generate web.xml deployment descriptor” then click on Finish.

Structure of Dynamic Web Project:

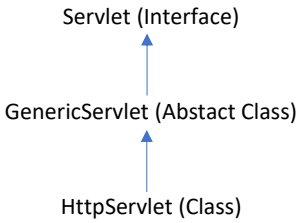
>servlets
    >Deployment Descriptor: servlets
    >JAX-WS Web Services
    >Java Resources
        >src/main/java
        >Libraries
>build
>src
    >main
        Java
    >webapp
        >META-INF
        >WEB-INF
            lib

web.xml

web.xml:
"web.xml" is also called as Deployment Descriptor.

Note:
Servlet is a special class in Java.

Servlet Hierarchy:



Creation of Servlets in Java:
In Java, we can create Servlet using Servlet Interface, GenericServlet Abstract Class and HttpServlet Class.

Note:
Usually, the most preferred way of creating a Servlet is just by using HTTPServlet Class. 'javax.servlet' package represents Servlet API.

Note:
Generic servlet is protocol independent servlet. Whereas HTTP servlet is protocol dependent servlet.

doGet():
It is a HTTP mapping method present inside HttpServlet class which is used to map Web Request associated with HTTP method as Get.

doPost():
It is a HTTP mapping method present inside HttpServlet class which is used to map web request associated with HTTP method as Post.

Note:
For each web request the associated HTTP method is Get by default.

Servlet Life Cycle:

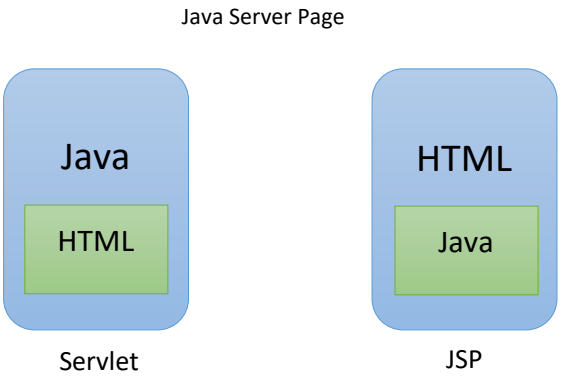
1. Class Loading
2. Instantiation
3. Initialization => init() => 1 time will invoke
4. Service => service() => n times will invoke
5. Destruction => destroy() => 1 time will invoke

Servlet Container:
It is a component of web server which manages servlet life cycle.

Drawbacks of Servlet:

1. Writing an HTML code inside a servlet is bit challenging which can be overcome using JSP where we can write the HTML code in more easy way.

JSP:
JSP stands for Java Server Page which is used to overcome drawback of servlet.



Note:

Since, servlet is the base technology to accept web request and to generate web response, JSP will be converted into servlet by the component called as JSP translator.

JSP Tags:

- JSP tags are used to write Java code inside JSP.
- There are 5 types of JSP tags as mentioned below:

1. Declaration ------ <%! %>
2. Scriptlet ------ <% %>
3. Expression ------ <%= %>
4. Directive ------ <%@ %>
5. Action

1. **Declaration Tag:**

Declaration Tag is used to declare Java variables or methods.

2. **Scriptlet Tag:**
   Scriptlet Tag is used to write Java logic.
3. **Expression Tag:**
   Expression Tag is used to print Java variables.
4. **Directive Tag:**
   Directive Tag is used to write import statements.

Note:

- Using HTML document, we cannot create dynamic web page. But using JSP we can create Dynamic web page. Since, we can write Java code inside a JSP.
- '.jsp' files should be created under webapp folder.

JSP Life Cycle:

1. Translation from JSP to Servlet => JSP translator
2. Class loading
3. Instantiation
4. Initialization => jspInit()
5. Service => _jspService()
6. Destruction => jspDestroy()

Note:

In the first life cycle, Servlet is faster than JSP. But from the second life cycle onwards, JSP is faster than Servlet.

Spring Framework

- Spring is an open-source, light-weight framework used to develop Enterprise Java Applications in more productive way.
- Spring Framework is dedicated only for Java.
- Spring Framework has following modules:

1. Spring Core
2. Spring MVC
3. Spring REST
4. Spring Boot

1. **Spring Core**
   This module of Spring Framework deals with the Core features of Spring Framework.

2. **Spring MVC**
   This module of Spring Framework deals with the MVC Architecture which is used to develop monolithic applications.

3. **Spring REST**
   This module of Spring Framework deals with the implementation of RESTful API's.

4. **Spring Boot**
   This module of Spring Framework deals with the development of Server-side applications called as Web-Services, Micro-services.

Spring Core

Spring Framework has two important features.

1. Inversion of control
2. Dependency injection

1. **Inversion of Control**
   This feature of Spring Framework deals with creation of "beans" in the application context. Because of this feature, a developer is no longer responsible to create "beans" in the application context.

2. **Dependency injection**
   In the application context, “beans” will be dependent on each other. Whenever there is a requirement of a created bean as a dependency then injecting or inserting that dependency can be done by the Spring Framework itself.

   Note:
   “beans” are the light-weight objects created by the Spring Framework.

- In the Spring Framework, beans are managed by Spring Containers.
- Spring Containers are the components of Spring Framework which will be responsible for Inversion of Control (IOC) and Dependency Injection (DI).

Bean Factory:

- It is one of the Spring Containers which will create the beans and insert the beans wherever required.
- Bean Factory will refer some configuration in order to create the beans and inject the beans.
- This configuration can be done in two ways:
  1. XML configuration
  2. Annotation configuration

Dependencies required in order to access Core Features of Spring Framework:

i. Spring Core
ii. Spring Context
iii. Lombok

1. **XML Configuration:**

Annotations used in Spring Core Module:

1. **@Bean**
   It is a method level annotation used to mark a method as responsible for Bean Creation.

2. **@Component**
   It is a class level annotation used to mark a class as component that means eligible for Bean Creation.

3. **@ComponentScan**
   It is a class level annotation used over a configuration class. which is used to scan the components in the Application Context.

4. **@AutoWired**
   It is a property level annotation used to inject required dependencies in the Application Context.

5. **@Value**
   It is a property level annotation used to initialize properties of an object.

6. **@Scope**
   This annotation is used along with @Component or @Bean. This annotation is used to define the scope of the Bean in Application Context which takes 2 values.
   i. Singleton
      If a scope of a Bean is singleton, then a single object or Bean will be created for the Application Context. By default, the scope will be singleton.

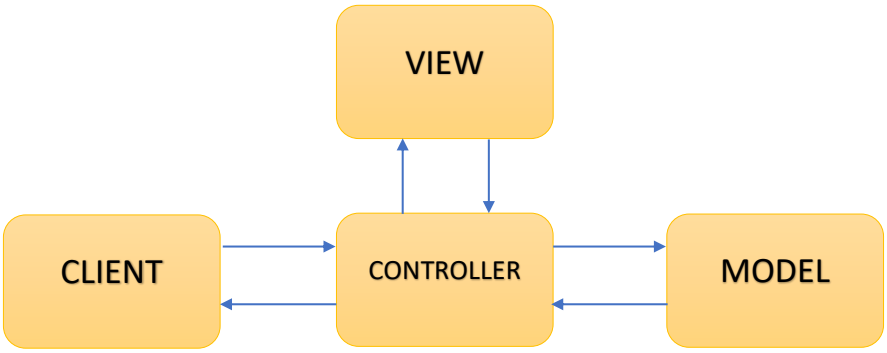
   ii. Prototype
      If a scope of a Bena is prototype, then a new Bean will be created when requested in the Application Context.

Spring MVC

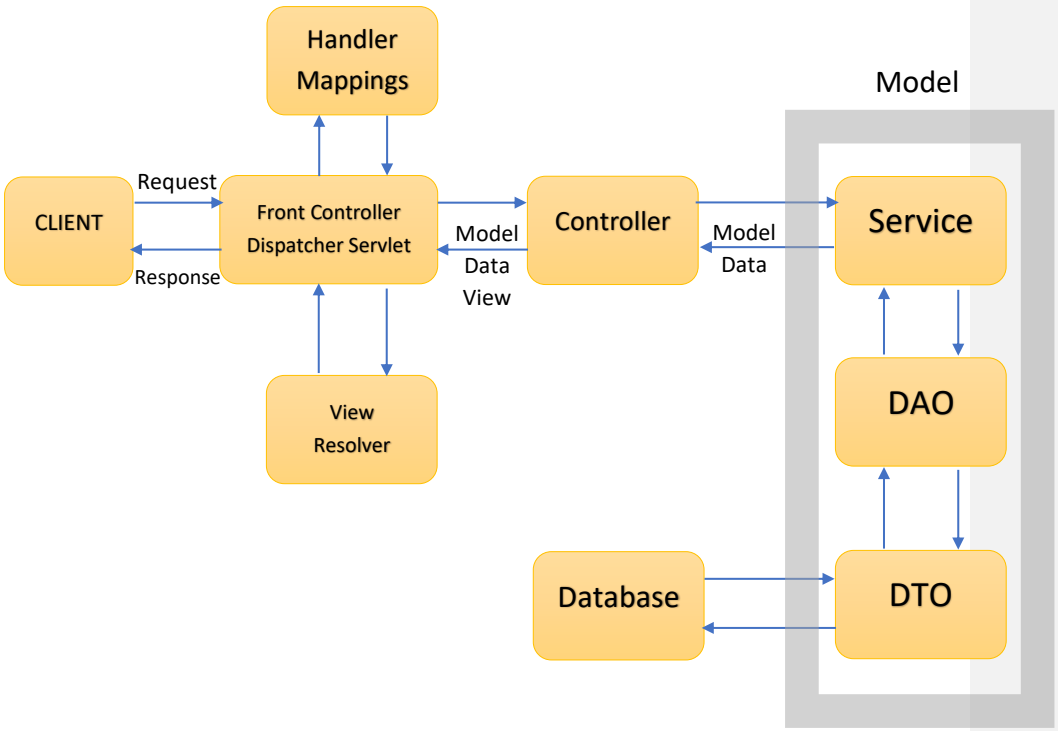
MVC:

- MVC stands for Model, View, Controller.
- It is an architectural design pattern using which we can develop monolithic applications.

Structure of MVC:



Detailed Structure of Spring MVC Application:



Steps to create Maven project (archetype-webapp):

1. Press ctrl + N, in the select wizard search for Maven project.
2. Select the Mavan project option and click on Next.
3. In the next window, do not skip the archetype selection then click on Next.
4. In the next window, select All catalogs then in Filter section type 'org.apache.maven' and then select 'maven-archetype-webapp' with the version 1.4. after that click on Next.
5. Provide the group-id and artifact-id for the project. And then untick a checkbox 'run archetype generation interactively'. Then click on Finish.

Configuration of Dispatcher Servlet inside web.xml:

1. In 'web.xml' file, make use of <servlet> tag inside which make use of <servlet-name> tag. There mentions the name as "Dispatcher".
2. Inside the same <servlet> tag, make <servlet-class> tag where we have to mention the qualified name of "DispatcherServlet".
3. To get the qualified name of "DispatcherServlet", press Ctrl + LShift + T and then search for "DispatcherServlet".
4. Double click on the class shown as "DispatcherServlet".
5. Copy the qualified name and paste inside <servlet-class> tag.
6. Make use of <servlet-mapping> tag in which mention the same servlet name inside <servlet-name> tag.
7. Make use of <url-pattern> tag inside <servlet-mapping> tag which will have '/' as a url pattern for "DispatcherServlet".

Annotations used in Spring MVC:

1. **@Controller**
   It is a class level annotation which is used to mark class as a Controller class which contains the methods to which a web request will be mapped.

2. **@RequestMapping**
   It is a method level annotation used to map a web request to particular method present inside Controller class. This annotation takes attributes as 'path', 'method'. 'path' defines path to resource which is the part of 'url', 'method' defines the HTTP method should be associated with the request.

3. **@RequestParam**
   This annotation is used to map request parameters to the respective arguments in the method. This annotation takes 'name' attribute which defines the name associated with request parameter.

Default objects in Spring MVC:

1. **ModelMap:**
   ModelMap object is used to transfer Model data to the respective 'view' page.

2. **HttpSession:**
   HttpSession object is used to keep the track of signed in user throughout the application context.

Note:

Session:

Session is the time duration between 'sign_in' and 'sign_out'.

Spring REST

Spring REST module is used to develop RESTful API's.

RESTful API:

API's which will help client-side applications to communicate with server-side applications which have been developed using REST architecture.

REST:

REST stands for Representational State Transfer.

Key features of REST Architecture:

1. **Client-Server Architecture.**
   In REST architecture, client-side application and server-side application are developed independently.
2. **Resource based**
   REST architecture is resource based means a client will send a request for a resource and the resource will be given back as a response to the client.
3. **Uniform Interface**
   The server-side application which has been built using REST architecture provides uniform interface to multiple client-side applications for communication.
4. **Stateless Communication**
   All the server-side applications which have been developed using REST architecture will be stateless in nature means server-side applications will not keep the track of requests coming from the client-side applications.
5. **Presentation**
   In REST architecture the data which is getting transferred from client-side application to the server-side application or vice-versa is presentable means it can be either in the form of XML or JSON.
   JSON is the widely used form of data.
6. **Cacheable**
   REST architecture provides provision for data caching.
7. **Layered System**
   In client-server architecture, some extra services can be added like API gateways, Load Balancer, etc. to make client-server architecture more efficient.

In client-side applications, the most common form of data representation is JSON (Java Script Object Notation).

Format of JSON:

{

“Key1”: “value1”,

“key2”: “value2”,

“key_n”: “value_n”

}

Marshalling:

The process of converting JSON into Java Object is called as Marshalling.

Un-Marshalling:

The process of converting Java Object into JSON is called as Un-Marshalling.

Dependencies required to implement RESTful application:

1. mySql connector
2. Hibernate Core Relocation
3. Java Servlet API
4. Spring Core
5. Spring Context
6. Spring Web MVC
7. Jackson Core
8. Jackson data-bind
9. Project Lombok (Optional)