Ctrl+spac: used to get the suggestion

Alt+shift+s: - used to add getters, setter, toString (), hashcode () and constructor.

Ctrl and hover on that particular class, method.

J2EE (JAVA 2ND ENTERPRISES EDITION)

Date: 04.09.2023

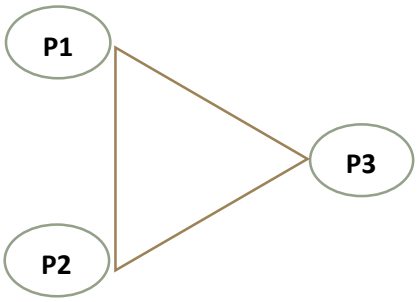
1st Part	2nd Part
1.MULTITHREADING	1.JDBC
2.FILE HANDLING	2.HIBERNATE WITH JPA
3.SERIALIZATION & DESERIALIZATION	3.SERVLETS & JSP
4.DESIGN PATTERN	4.SPRING FRAMEWORK

❖MULTITHREADING: -

: -In order to understand multithreading, we need to understand multitasking.

- **Multitasking: -** It is the process of performing more than one task simultaneously/at the same time, is called as multitasking.

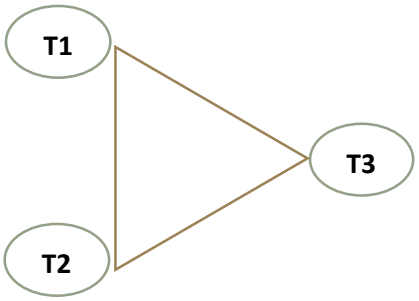
→Task: -Task is an end goal to be achieve or it is a group of more than one processor.



Task

- **Multiprocessing: -** It is the process of executing more than one process simultaneously/at the same time, is called as multiprocessing.

→Process: - Process can be defined as group of more than one sub-process called as threads.



Process

- **MULTITHREADING: -** It is the process of executing more than one thread simultaneously/ at the same time, is called as multithreading.
  →Thread: - Thread can be define as a smallest unit of task or a light-weight sub-process, is called as thread.

- **Thread in java: -** A thread is a special class in java which is contains a small program.

  →There are two ways to creating a thread in java: -
  1. By extending thread class.
  2. By implementing Runnable interface.

  →1st way of creating user define threads.
  Ex: -
  (1)

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

public class MyThread1 extends Thread {

	@Override
	public void run() {
		// TODO Auto-generated method stub
		System.out.println("Hello from MyThread1. ");
	}

}
```

  (2)
  →2nd way to creating user define thread

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

public class MyThread2 implements Runnable {

	@Override
	public void run() {
		// TODO Auto-generated method stub

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

	}


}
```

(3)

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

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

public class ThreadMain {

	public static void main(String[]args) {

		MyThread1 myThread1 = new MyThread1();
		myThread1.start();

		MyThread2 myThread2 = new MyThread2();
		Thread thread = new Thread(myThread2);
		thread.start();



	}
}
```

Output: -

```
Hello from MyThread1.
Hello from MyThread2.
```

run (): - It is abstract method present inside runnable interface.

start (): - It is non-static method present inside thread class.

→**Thread Properties: -** Each user define thread will have three properties

1. **id: -** The value for this property will be assign by **JVM.**
   getId (): - This method is used to access the id value of a thread, it is non-static in nature, present inside thread class.

2.Name: -

setName (): -This method is used to give user define name for a thread, it is non-static in nature, present inside thread class.

getName (): - This method is used to access the name of a thread.

: -It is non-static in nature, present inside thread class.

3.Priority: -

setPriority (): - This method is used to give the priority value for the user define thread.

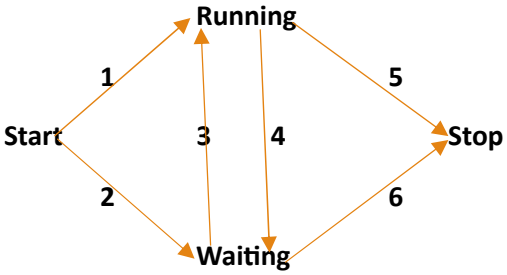
: - The default value of priority will be 5.

: - Min is 1.

: - Max is 10.

getPriority (): - This method is used to access the priority of a thread.

→Life cycle of a thread: -



→There are four phases of a thread in its life cycle.

1.Start phase: - when a new thread is created and start () is invoke, then that particular thread will be in start phase.

2.Running phase: -When a thread starts its execution, then that particular thread will be in running phase.

3.Waiting phase: - When a thread paused its execution, then that particular thread will be in waiting phase.

4.Stop phase: - Once the execution of a thread is done, then that particular thread will be stop phase.

➔ There are six transitions during life cycle of a thread: -

1.Start to running phase: -: - When a thread scheduler gives CPU to a thread for its execution then, that particular thread will go from start phase to running phase.

2.Start to waiting phase: -When thread scheduler doesn't give a CPU to thread for its execution then that particular thread will go from start phase to waiting phase.

3.Running to waiting phase: - When a thread scheduler takes back a CPU from the thread then that particular thread will go from running phase to waiting phase.

4.Waiting to running phase: - When a thread scheduler gives a CPU to a thread which was in waiting phase for its execution then that particular thread will go from waiting to running phase.

5.Running to stop phase: - When an execution of thread is completed than that particular thread will go from running to stop phase.

6.Waiting to stop phase: - When a thread is waiting for longer time then automatically that thread will go from waiting to stop phase.

Date: 13.09.2023

→ **In multithreading each thread will be allocated one dedicated stack.**

: - If we invoke run () for particular thread through start () then that run () will be loaded inside an independent stack area.

: - If we invoke run () of a particular thread directly then dedicated stack won't be allocated for that stack instant that run () will be loaded inside current stack area.

Ex: -

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

public class Account {

	private double accountBalance;

	public Account(double accountBalance) {
		this.accountBalance = accountBalance;
	}

	public synchronized void deposit(double amount) {

		accountBalance += amount;
		System.out.println("Amount of  rupees" + amount + "has been
credited.");
		System.out.println("Updated acount balance" + accountBalance);
	}

	public synchronized void withdraw(double amount) {

		if (amount > accountBalance) {
			System.out.println("Insufficient acount balance.");
		}
		else {
			accountBalance -= amount;
			System.out.println("Amount of rupees " + amount + " has been
debited.");
			System.out.println("Updated account balance = " +
accountBalance);
		}
	}

}
```

Ex: -

```
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);
	}

}
```

Ex: -

```
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(10000);
	}
}
```

Ex: -

```
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();
	}
}
```

Ex: -

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

public class MyThread3 extends Thread{
	@Override
	public void run() {
		// TODO Auto-generated method stub
			for(int i =1; i <=5; i++) {
				System.out.println(this.getName() + " is running");
			}
	}

}
```

Ex: -

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

public class MyThread4 extends Thread {

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

Ex: -

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

import com.jspiders.multithreading.thread.MyThread3;
import com.jspiders.multithreading.thread.MyThread4;

public class ThreadMain1 {

	public static void main(String[] args) {

		MyThread3 myThread3 =new MyThread3();
		myThread3.setName("Thread3");
		MyThread4 myThread4 =new MyThread4();
		myThread4.setName("Thread4");

		myThread3.start();
		myThread4.start();
	}
}
```

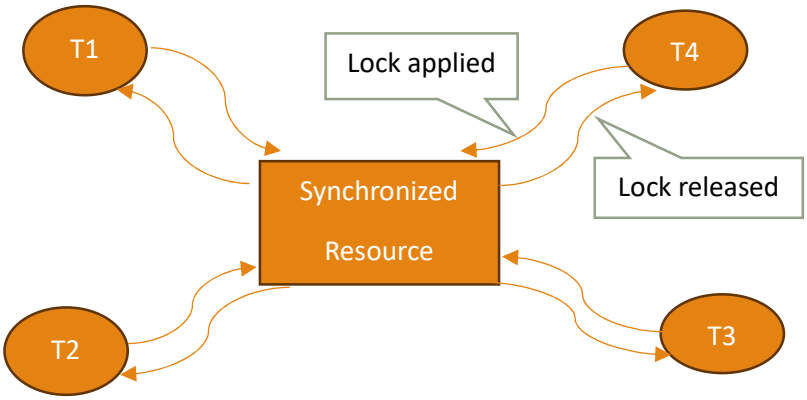
Date: 14.09.2023

- **Shared Resource: -**

  : - The resource which is accessed by multiple threads is called as shared resource.
  : - the method of a variable can be considering as shared resource.
  : - When multiple threads operate on single shared resource, leads to data inconsistency.
  : - Since all the threads are unaware of each other's operation on the shared resource.
  : - In the recent program husband and wife threads accessing deposit and withdraw method of account class, at the same time and being unaware of each other's operation on both the methods.
  : - That leads to inconsistent output.
  : - This data inconsistency can be avoided using synchronization.

- **Synchronization: -**

  : - It is a process of achieving consistent data.
  : - Synchronization can be achieve using **synchronization keyword**.
  : - It is a non-access modifier.
  : - which can be used with variable or methods.
  : - Synchronized resource will allow only one thread at a time to accessing.
  : - When a thread gets an access to a synchronized resource, it will apply lock on it
  : - So that other threads cannot access that particular resource.



→Types of Synchronized

1. Class Lock: -If synchronized is static in nature then the lock applied on it, is called as class lock.
2.Object lock: - If synchronized resource is non-static in nature, then the lock applied on it, is called as object lock.

Ex: -

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

public class Account {

	private double accountBalance;
```

```
	public Account(double accountBalance) {
		this.accountBalance = accountBalance;
	}

	public synchronized void deposit(double amount) {

		accountBalance += amount;
		System.out.println("Amount of  rupees " + amount + " has been
credited.");
		System.out.println("Updated acount balance " + accountBalance);
	}

	public synchronized void withdraw(double amount) {

		if (amount > accountBalance) {
			System.out.println("Insufficient account balance.");
		}
		else {
			accountBalance -= amount;
			System.out.println("Amount of rupees " + amount + " has been
debited.");
			System.out.println("Updated account balance = " +
accountBalance);
		}
	}

}
```

(2)
Ex: -

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


public class Resource {

	public void message() {

		for (int i = 0; i < 5 ; i++) {

			System.out.println("Hello from "+
Thread.currentThread().getName());
		}
	}
}
```

Date: 15.09.2023

Stop (): - It is a non-static method present inside thread class.

: - It is used to move a thread **running state to stop state forcefully.**

: - This method is deprecated, means **it is not recommended for use**, and set for removal from the library.

: - To pull a thread from running state to stop state, before its complete execution is not recommended.

Ex: -

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

public class MyThread7 extends Thread {

	@Override
	public void run() {

		for(i = 1; i<=5;i++) {
			if(i==3) {
				this.stop();
			}
			System.out.println(this.getName() + " is running");
		}
	}

}
```

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

import com.jspiders.multithreading.thread.MyThread7;

public class ThreadMain3 {

	public static void main(String[] args) {

		MyThread7 myThread7 = new MyThread7();
		myThread7.setName("Thread7");
		myThread7.start();
	}
}
```

Date: 16.09.2023

(wait(), notify(), notifyAll(): - its present inside an object class , not thread class **)**

wait(): - It is a non-static method present inside an object class.

: - It is used to put a thread from running phase to waiting phase **forcefully, based on some condition.**

: - If a thread has been put into waiting state using wait() method, then in order to resume its execution, we need to make use of: -

notify(): -It is a non-static method present inside an object class.

: - It is used to move a thread a thread in waiting state to a running state.

: -

: - this method is used to resume, the execution of only one thread.

→notifyAll (): -

: - It Is a non-static method present inside object class.

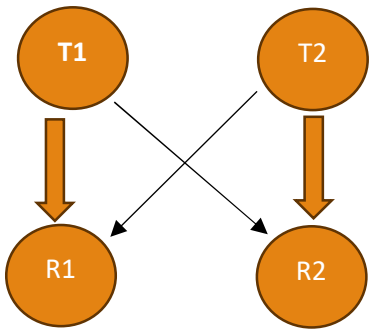
: - It is used to move multiple threads or more than one threads from waiting phase to running phase.

- **sleep (long** time in millisecond)**: -**
  : - It is static method present inside the thread class.
  : - It is used to move a thread from running states to waiting state.
  : - It accepts one lock type argument that is time in milli-second.
  : - sleep () will calls the execution of a thread for specified time.
  : - once time gets accessed the thread will resume, its execution automatically.

→Difference between wait () and sleep ()

wait ()	sleep ()
1.it is non-static method	**1.It is a static method**
2.it is present inside object class	**2.it is present inside thread class**
3.its don't accept any argument	**3.it is accepting a long type argument (**that is time in milli second)
4.If a thread has been moved from running to waiting states using wait method, then in order to resume its execution. We need to make use of notify () or notifyAll ()	**4.If a thread has been moved from running states to waiting state using sleep (). then the particular thread will resume, its execution, after the specified time.**

- **Dead Lock: -**
  : -



: - In above diagram thread1 will apply the lock on resouce1.
: - thread2 will apply the lock on rsource2.
: -Thread1 wants to access resource2, and at the same time thread2 wants to access resource1 but thread1 will not release the lock on resource1 until and unless it gets access to resource2.
: -Similarly, thread2 will not release the lock on resource2.untill and unless it gets access to resource1.
: -Its leads to one abnormal situation, called as Dead lock. where program will not get terminated.

Date: 20.09.2023

➢ Deamon Thread: -

: -In java, the threads created by extending the Thread class or by implementing the runnable interface are considered as user defined threads.

: - Although there are some predefine threads in java which are not created by the user and are known as **Deamon Threads.**

: - The Deamon threads are called for execution implicitly by the **JVM** whenever required.

: - The execution of the Deamon thread is not visible to the programmer.

: - Deamon threads are low priority threads.

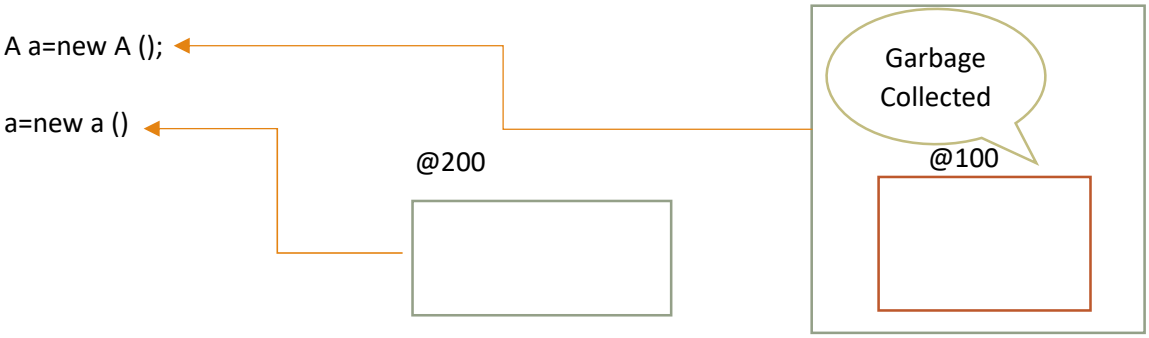
➢ Garbage Collection: -

: - GC is one of the most important Deamon threads in java, it helps to make java strong in memory management.
: - The JVM implicitly called the GC thread, whenever a garbage value is detected in the memory.
: - the GC thread identifies the garbage value and removes it from the memory.

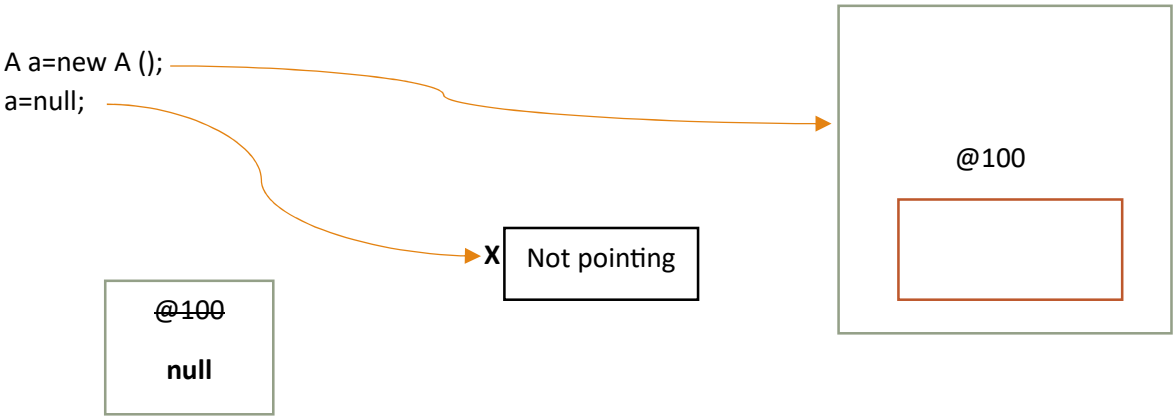
: - The GC thread is invoked by the JVM in two cases
1.Derefereing
2.Nullinifying

1.Derefereing

: - Whenever a variable is re-initialized then we can say that the previous value which it was pointing to has been dereferred.
: - Such dereferred value will not be accessible and has to be garbage collected



2.Nullinifying
: - If a variable is initialized and later Nullified, then it will no more point/refer towards the value created in the memory.
: - Hence the created value will become inaccessible and will be garbage collected.



→**Advantages of GC: -**

: - 1. The GC thread **makes java Robust.**

: - 2. Due to the GC thread, the programmer **does not have to take care of managing the memory in JAVA.**

Date: 20.09.2023

➢ File Handling: -

: - The process of performing the operation on a file is known as file handling.
Or handling the file is known as file handling.

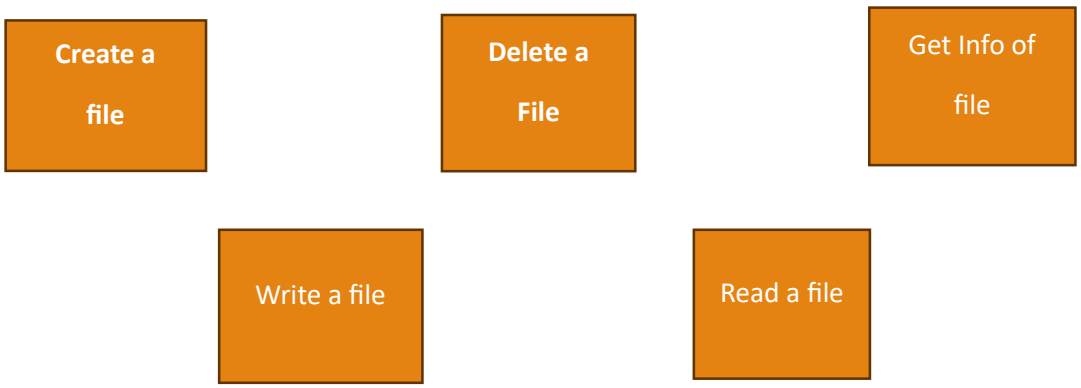
→**File: -** File can be considered as an entity.

: - When the data is being retrieved from the file, then the file is referred as a 'source entity'.

: - When a data as being send to the file, then the file is referred as 'target entity'.

: - To deal with a file in java, we need to create an object of a class named 'File'.

→**Operation in file handling: -** the operation that can be perform on file in java, are as follows: -



: - The operation of creating, deleting and getting the information of a file are not dependent upon the data of a file.

: - Whereas the operation of read and write dependent upon the data which the file holds or can hold.

Ex;

```
package com.jspiders.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");
			try {
					file.createNewFile();
					System.out.println("File successfully created.
");

			} catch (IOException e) {
				e.printStackTrace();
				System.out.println("File is not created. ");

			}
	}

}
```

: - the file class is present in 'java.io package.

: - the file class has overloaded constructors.

: - the most commonly used constructer is the one which accepts as String argument.

: - the String argument is considered as the 'pathname' of the file.

- **Create new file () method: -**

  : - It is non-static method present inside the file class.

  : - This method is used to create file based on the path name given in the file class constructor.

  : - This method overwrites the existing file when a file with same name and extension is being created.

  Ex: -

```
package com.jspiders.filehandling.operations;

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

public class CreateFileDemo2 {

	public static void main(String[] args) {

		File file = new File("Test.txt");
```

```
		if(file.exists()) {
			System.out.println("File already exists. ");
		}else {
			try {
				file.createNewFile();
				System.out.println("File is created. ");
			}
			catch(IOException e) {
				e.printStackTrace();
				System.out.println("File is not created. ");

			}
		}

	}
}
```

Date: 22.09.2023

→ **exists (): -** It is a non-static method present inside file class

: - It will return true, if a file is present at specify location otherwise it will return false.

→**Absolute path: -**

: - Absolute path defines the exact location of a file.

Ex: -

```
	package com.jspiders.filehandling.operations;

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

public class CreateFileDemo3 {

		public static void main(String[] args) {

			File file =new File("F:/J2EE/filehandling/Test1.txt");
			if (file.exists()) {
				System.out.println("File already exists.");

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

				} catch (IOException e) {
					e.printStackTrace();
					System.out.println("File is not created. ");

				}

			}
		}
}
```

→delete() : -

: - it is non-static method present inside file class.

: - It is used to delete a file at specified location.

Ex: -

```
package com.jspiders.filehandling.operations;

import java.io.File;

public class DeleteFIle {

	public static void main(String[] args) {

		File file = new File("F:/J2EE/filehandling/test1.txt");
		if (file.exists()) {

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

		} else {
			System.out.println("File does not exists.");

		}
	}
}
```

Date: 23.09.2023

→getName (): - It is a non-static method present inside file class.

: - It is use to fetch name of the file along with its extension.

→ getAbsolutePath (): -

: - It is a non-static method present inside file class.

: - It returns the exact location of a file.

→canRead (): -

: - It is a non-static method present inside file class.

: - It returns true, if the file is readable.

: - Otherwise, it returns false.

→canWrite (): -

: - It is a non-static method present inside file class.

: - It returns true, if the file is writable.

: - otherwise, it returns false.

→canExecute (): -

: - It is a non-static method present inside file class.

: - It returns true, if the file is executable.

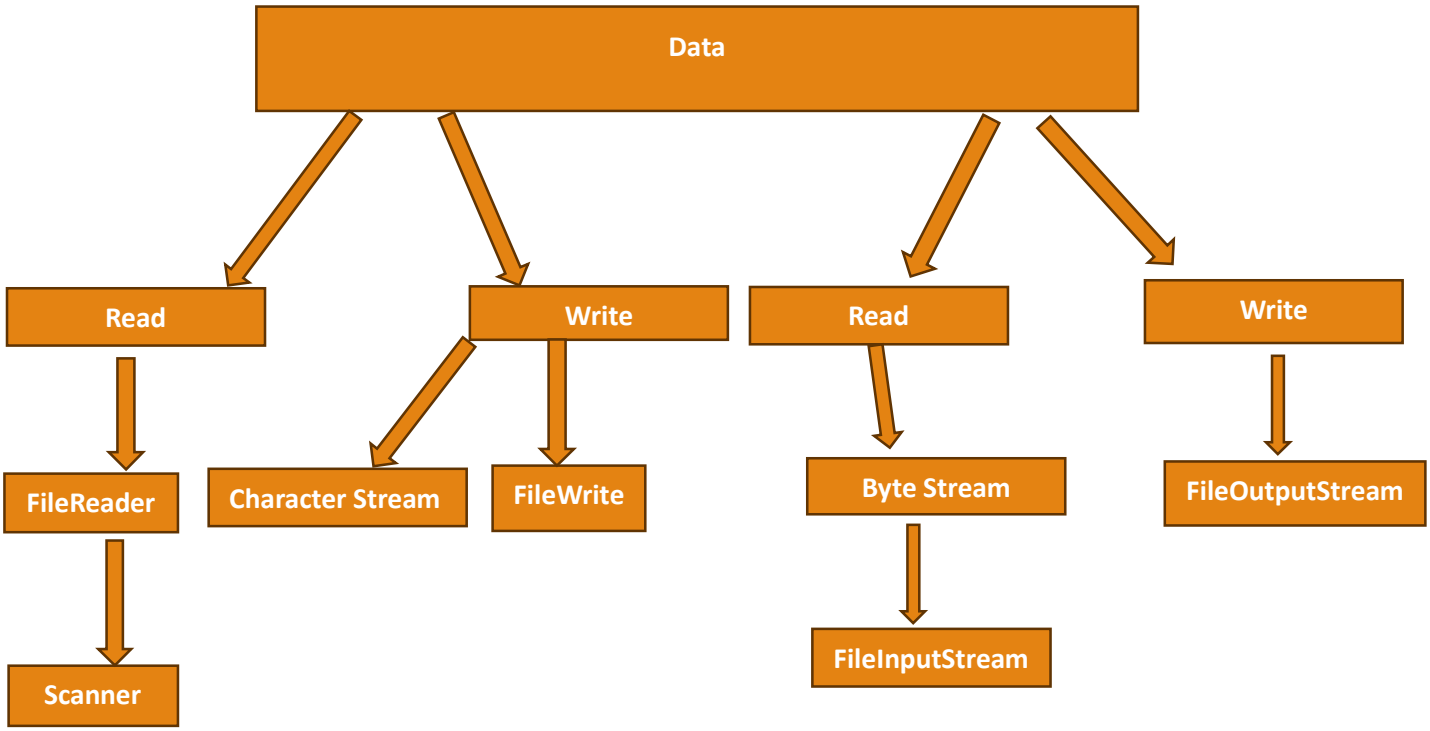
: - otherwise, it returns false.

→length (): -

: - it is a non-static method present inside a file class.

: - It returns number of characters present inside a file.

→Read and Write operation on fille



→CharStreamWrite.java

Ex: -

```
package com.jspiders.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("Hello World!!");
                        System.out.println("Data is written to the file. ");
                        fileWriter.close();
                }
                else {
                        System.out.println("File does not exists.");
                        file.createNewFile();
                        System.out.println("File is created");
                        FileWriter fileWriter = new FileWriter(file);
                        fileWriter.write("Hello world");
                        System.out.println("Data is written to the file");
                        fileWriter.close();
                }
        }

}
```

→**ByteStreamWrite.java**

:-

```
package com.jspiders.filehandling.operations;

import java.io.File;
import java.io.FileOutputStream;
import java.io.FileWriter;
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(1000);
                        System.out.println("Data is written to the file.");
                        fileOutputStream.close();
                }
                else {
                        System.out.println("File does not exists.");
                        file.createNewFile();
                        System.out.println("File is created");
                        FileOutputStream fileOutputStream=new FileOutputStream(file);
                        fileOutputStream.write(1000);
                        System.out.println("Data is written to the file.");
                        fileOutputStream.close();
                }
        }
}
```

Note: - To perform write operation on the file we can make use of Writer classes.

and OutStream classes.

Date: 25.09.2023

→CharStreamDemo1.java

Ex: -

```
package com.jspiders.filehandling.operations;

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

public class CreateStreamDemo1 {
	public static void main(String[] args) throws IOException {

		File file =new File("Demo.txt");
		if (file.exists()) {

				FileReader fileReader =new FileReader(file);
				int value = fileReader.read();
				System.out.println(value);
				System.out.println("Data is fetched from the
file.");
				fileReader.close();

			}else {
				System.out.println("File does not exists.");
			}

		}

	}
```

→CharStreamReadDemo2.java

Ex: -

```
package com.jspiders.filehandling.operations;

import java.io.File;
import java.io.FileNotFoundException;
import java.util.Scanner;

public class CharStreamReadDemo2 {

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

		File file=new File("Demo .txt");
		if (file.exists()) {

			Scanner scanner=new Scanner(file);
			while(scanner.hasNextLine());{
```

```
                        System.out.println(scanner.nextLine());
                }

                System.out.println("Data is fatched from the file.");
                scanner.close();


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

}
```

→ByteStreamRead.java

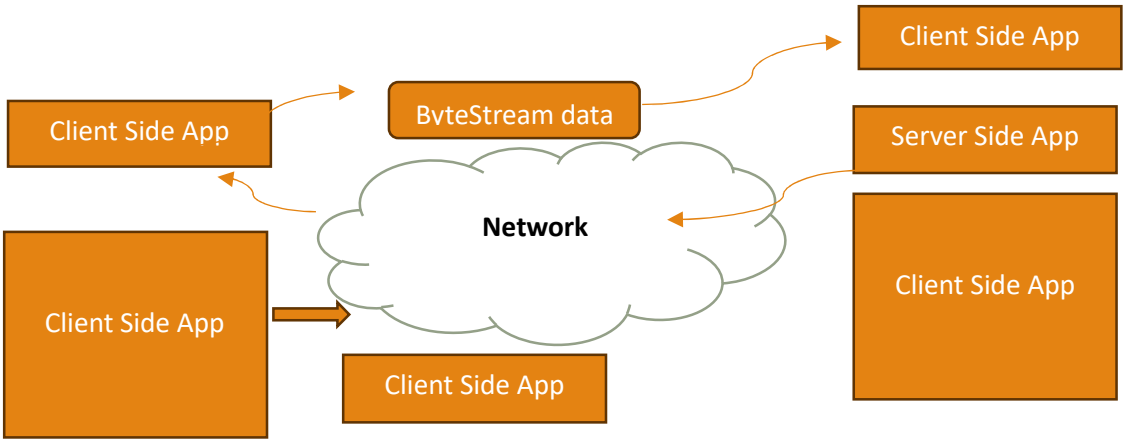
Ex: -

```
package com.jspiders.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("Text.txt");
        if (file.exists()) {

                FileInputStream fileInputStream=new FileInputStream(file);
                int value = fileInputStream.read();
                System.out.println("value");
                System.out.println("Data is factched from the file.");
                fileInputStream.close();
        }
        else {
                System.out.println("File does not exists.");
        }
    }

}
```

Date: 26.09.2023

Note: - To perform read operation on file, we can make use of reader classes or inputStream classes.

- **Serialization: -** It is the process of converting java object into ByteStream format is called as Serialization.
- **Deserialization: -** It is the process of converting ByteStream format into original java object is called as deserialization.



→Serializable interface: -

It is marker interface, which has to be implemented by a class whose objects are eligible for serialization and deserialization process.

: - **Marker interface**: - It is an empty interface which can be implemented to provide some special abilities to a class.

→For serialization: - Write object method of ObjectOutputStream class or writeObject () will be use.

→For deserialization: - Read object method of ObjectInputStream class or readObject () will be use.

Date: 27.09.2023

→Serial.java

Ex: -

```
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 Serial {

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

		File file = new File("File.txt");
		if (file.exists()) {
			System.out.println("File already exists.");
			FileOutputStream fileOutputStream = new
FileOutputStream(file);
```

```
                    ObjectOutputStream objectOutputStream = new
ObjectOutputStream(fileOutputStream);
                    objectOutputStream.writeObject(new User
(1,"Ramesh","ramesh@gmail.com","ramesh@123"));
                    System.out.println("Object is written to the file.");
                    objectOutputStream.close();
                    fileOutputStream.close();

             } else {
                    file.createNewFile();
                    System.out.println("File is created.");
                    FileOutputStream fileOutputStream = new
FileOutputStream(file);
                    ObjectOutputStream objectOutputStream = new
ObjectOutputStream(fileOutputStream);
                    objectOutputStream.writeObject(new User
(1,"Ramesh","ramesh@gmail.com","ramesh@123"));
                    System.out.println("Object is written to the file.");
                    objectOutputStream.close();
                    fileOutputStream.close();


             }

       }
}
```

→Deserial.java

Ex: -

```
package com.jspiders.serializationanddeserialization.deserialization;

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

import com.jspiders.serializationanddeserialization.object.User;

public class Deserial {

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



             File file = new File("File.txt");
             if (file.exists()) {

                    FileInputStream fileInputStream = new FileInputStream(file);
                    ObjectInputStream objectInputStream = new
ObjectInputStream(fileInputStream);
                    User user = (User) objectInputStream.readObject();
                    System.out.println(user);
                    System.out.println("Object is read from the file.");
                    objectInputStream.close();
                    fileInputStream.close();
```

```
		}
		else {
			System.out.println("File does not exits.");
		}

	}
}
```

(2)

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

import java.io.Serializable;

public class User implements Serializable {


	private static final long serialVersionUID = 1L;
		int id;
		String name;
		String email;
		String password;


		public User(int id, String name, String email, String password) {}


		@Override
		public String toString() {
			return "User [id=" + id + ", name=" + name + ", email=" +
email + ", password=" + password + "]";
		}









}
```

Date: 28.09.2023

No class (Due to Ganpati visarjan**)**

Date: 29.09.2023

➢ Design Patterns: -

: - Design patterns are some predefined/ existing programming structure or methodologies which can be used to meet **Specific requirements** for the application and **to avoid issues** occurring during application development process.

→**Design Patterns Catalogue: -** It is the place where design patterns are available.

: - A newly discovered design patterns can be added to the design patterns catalogue.

→Types of Design Patterns

: -Based on the type of requirement/issues with which a particular design pattern is dealing with.

: - It is classified into several types.

Widely use Design patterns are: -

1.Creational Design Patterns.

2.Structural Design Patterns.

→1.Creational Design Patterns: -

: - These design patterns deal with the requirements or the issues associated/related with

object creation in an application.

: - Widely used creational Design patterns are: -

1.Singleton Design pattern.

2.Factory Desing Pattern.

3.Builder Design Pattern.

→2.Structural Design Patterns: - There design patterns deal with the requirements or issues associated with the structure of classes and interfaces in an application.

→Widely used structural design pattern.

→ 1.Singleton Design pattern.

: - Singleton design pattern can be used to **restrict** user from creating multiple objects of a particular class.

: - This requirement can be achieved by making constructor of the class private in nature.

: - To access that constructor outside the class, we can implement one helper method.

Date: 30.09.2023

→SingletonLazy.java

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

public class SingletonLazy {

		private static SingletonLazy singletonLazy;
		private SingletonLazy() {

		}
		public static SingletonLazy getObject(){
			if (singletonLazy ==null) {

				singletonLazy = new SingletonLazy();
```

```
		}
			return singletonLazy;
		}

}
```

(2)

→SingletonEager .java

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

public class SingletonEager {



	private static SingletonEager singletonEager =new SingletonEager();

			private SingletonEager() {
	}

			public static SingletonEager getObject() {
				return singletonEager;

			}
}
```

→SingletonEagerMain.java

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



public class SingletonEagerMain {

		public static void main(String[] args) {


			SingletonEager object1= SingletonEager.getObject();
			SingletonEager object2= SingletonEager.getObject();
			SingletonEager object3= SingletonEager.getObject();



			System.out.println(object1);
			System.out.println(object2);
			System.out.println(object3);
		}
}
```

→ **lazy instantiation: -** In lazy instantiation will be created when we invoke getObject ()

→**Eager Instantiation: -** In eager instantiation object will be created during class loading process, since the object reference variable is static in nature.

→Factory design patterns: -

: - This design pattern is used to create objects for the classes when required.

: - If an application contains several classes, then it is required to create objects of those classes and keep them ready.

: - But there is a chance that some of the objects might remain unused, which leads wastage of memory.

: - This is not the efficient way of memory utilization.

: - This problem can be avoided using **factory design pattern**.

Ex: -

→Beverage.java

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

public interface Beverage {

	void order();
}
```

Note: - Functional interface

→MasalaTea.java

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

	public class MasalaTea implements Beverage {

		@Override
		public void order() {

				System.out.println("Masala tea is ordered.");

		}
	}
```

→GingerTea.java

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

public class GingerTea implements Beverage {

	@Override
	public void order() {

			System.out.println("Ginger tea is ordered.");

	}


}
```

→LemonTea.java

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

public class LemonTea implements Beverage {

	@Override
	public void order() {
				System.out.println("Lemon tea is ordered.");

	}


}
```

→GreenTea.java

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

public class GreenTea implements Beverage {

	@Override
	public void order() {
		System.out.println("Green tea is ordered.");

	}


}
```

→FactoryMain.java

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

import java.util.Scanner;

public class FactoryMain {

	private static Beverage beverage;

	public static void main(String[] args) {

		try {
			factory().order();
		} catch (NullPointerException e) {
			e.printStackTrace();
			System.out.println("Beverage is not ordered.");
		}

	}

	private static Beverage factory() {

		Scanner scanner = new Scanner(System.in);
```

```
            System.out.println("Enter 1 to order Masala tea\nEnter 2 to order
Ginger tea\n"
                        + "Enter 3 to order Lemon tea\nEnter 4 to order Green
tea");
            System.out.println("Enter your choice");
            int choice = scanner.nextInt();
            switch (choice) {
            case 1:
                  beverage = new MasalaTea();
                  break;
            case 2:
                  beverage = new GingerTea();
                  break;
            case 3:
                  beverage = new LemonTea();
                  break;
            case 4:
                  beverage = new GreenTea();
                  break;
            default:
                  System.out.println("Invalid choice");
            }
            scanner.close();
            return beverage;
      }

}
```

Date: 02.10.2023

Date: 03.10.2023

→Builder Design Pattern: -

: - Builder design pattern is used to **create complex objects.**

→Complex object: - If the object has **too many properties** that object is called as complex object.

: - It is not possible to create complex object using **All arguments constructor**, since the object has too many properties and programmer has to **remember the datatype of each argument** and the sequence in which constructor is accepting the arguments.

: - To overcome this problem, we can make use of builder design pattern where **one intermediate builder class** will be there, which is aware of all the properties of complex object.

→Contact.java

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

public class Contact {

            private String firstName;
            private String lastName;
            private String dob;
            private long mobile;
            private long work;
```

```
	private long home;
	private int landline;
	private String email;
	private String website;
	private String address;


	public Contact(String firstName, String lastName, String dob, long
mobile, long work, long home, int landline,
			String email, String website, String address) {
		super();
		this.firstName = firstName;
		this.lastName = lastName;
		this.dob = dob;
		this.mobile = mobile;
		this.work = work;
		this.home = home;
		this.landlines = landline;
		this.email = email;
		this.website = website;
		this.address = address;
	}


	@Override
	public String toString() {
		return "Contact [firstName=" + firstName + ", lastName=" +
lastName + ", dob=" + dob + ", mobile=" + mobile
				+ ", work=" + work + ", home=" + home + ",
landline=" + landline + ", email=" + email + ", website="
				+ website + ", address=" + address + "]";
	}



}
```

Date: 04.10.2023

→ContactBuilder.java

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

public class ContactBuilder {

	private String firstName;
	private String lastName;
	private String dob;
	private long mobile;
	private long work;
	private long home;
	private int landline;
	private String email;
	private String website;
	private String address;


	public ContactBuilder firstName(String firstName) {
```

```
            this.firstName = firstName;
            return this;
      }
      public ContactBuilder lastName(String lastName) {
            this.lastName= lastName;
            return this;
      }
      public ContactBuilder dob(String dob) {
            this.dob=dob;
            return this;
      }
      public ContactBuilder mobile(long mobile) {
            this.mobile= mobile;
            return this;
      }
      public ContactBuilder work(long work) {
            this.work=work;
            return this;
      }
      public ContactBuilder home(long home) {
            this.home=home;
            return this;
      }
      public ContactBuilder landline(int landline) {
            this.landline=landline;
            return this;
      }
      public ContactBuilder email(String email) {
            this.email=email;
            return this;
      }
      public ContactBuilder website(String website) {
            this.website=website;
            return this;
      }
      public ContactBuilder address(String address) {
            this.address=address;
            return this;
      }
      public Contact buildContact() {
            // TODO Auto-generated method stub
            return new
Contact(firstName,lastName,dob,mobile,work,home,landline,email,website,address);
      }
}
```

→Adapter Design Pattern: -

: - It is a structural design pattern which is use to combine the properties and behaviours of two different entities without any direct relationship between those two entities.

: - Here, we have to make use of one more intermediate entity called as adapter, which inherits properties of one entity and behaviours of another entity.

→Game.java

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

public interface Game {

	void football();

	void hockey();

}
```

→Student.java

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

public class Student {

	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;
	}


}
```

→StudentAdapter.java

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

public class StudentAdapter extends Student implements
Game {

	@Override
	public void football() {

		setId(1);
		setName("Ankit");
		setEmail("ankit@gamil.com");
```

```
                                        System.out.println(getName() + " is a
captain of football team.");

                                }

                                @Override
                                public void hockey() {

                                        setId(2);
                                        setName("Rajesh");
                                        setEmail("rajesh@gamil.com");

                                        System.out.println(getName() + " is a
captain of hockey team.");
                                }

}
```

→Teacher.java

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

public class Teacher {

	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;
	}




}
```

→TeacherAdapter.java

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

public class TeacherAdapter extends Teacher implements Game {
```

```
	@Override
	public void football() {

		setId(1);
		setName("Ankit");
		setEmail("ankit@gmail.com");

		System.out.println(getName() + " is a captain of football team.");

	}

	@Override
	public void hockey() {

		setId(2);
		setName("Rajesh");
		setEmail("rajesh@gamil.com");

		System.out.println(getName() + " is a captain of hockey team.");
	}


}
```

→AdapterMain.java

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

public class AdapterMain {

	public static void main(String[] args) {
		StudentAdapter studentAdapter= new StudentAdapter();
		studentAdapter.football();
		studentAdapter.hockey();


		TeacherAdapter teacherAdapter = new TeacherAdapter();
		teacherAdapter.football();
		teacherAdapter.hockey();
	}

}
```

Date: 05.10.2023

➢ JDBC (Java Database Connectivity):

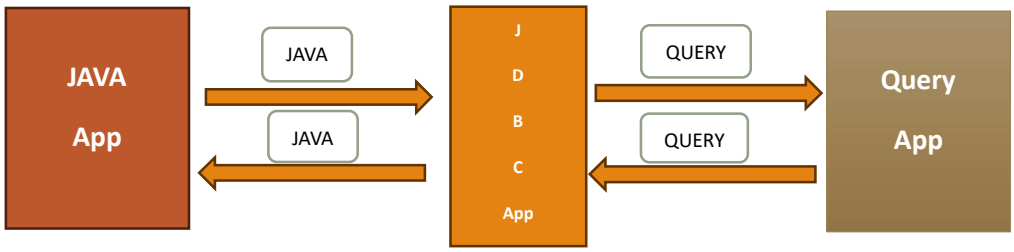
: - JDBC stands for Java Database Connectivity.

: - It is the only technology which can be used to connect any java application (java application: the application which has been built by using java programming language) **with any database application**.

: - JDBC can be called as **JAVA API.**

→ **API: -** API Stands for **Application Programming Interface.**

: - It a software which can be use by **one application to communicate with another application.**



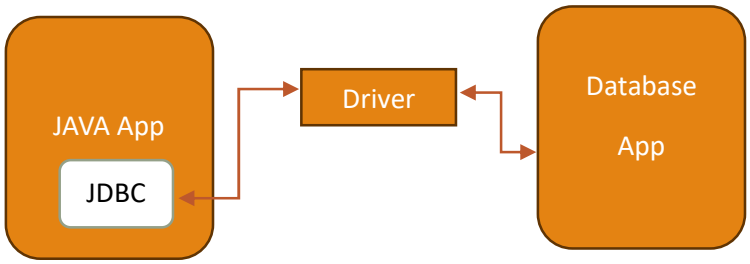
: - Since, **Java application does not understand Query language** and **database application does not understand java programming language**, that means there are incompatible with each other.

: - If java application has to be connected with database application, we have to make use of **JDBC API,** which **converts java commands** into **query commands** and **vice-versa**.

→Need for storing the data outside the java application.

: - We cannot store any data within the application permanently but if we want that data in future that data has to be stored outside the application.

Note: - JDBC is a package present inside java system library as java.sql package.



: - JDBC is a part of java application so that it cannot communicate directly with the database application.

: - JDBC will make use of a driver to communicate with database application.

→Driver: - It is a software which acts as a **bridge between Java application and Database application.**

: - It is an external resource which we have to add to our java application.

Note: - Each and very database application will have its **own dedicated driver software**

→Pre-requisites for use of JDBC Technology.

Step-1: If we want to connect our java application with database application, we have to add the connector software for respective database application to our java application.

Step-2: Add MySQL Connector (jar file) to java project.

Step-3: In a browser search for **maven repository.**

Step-4: Click on the first link shown on the browser.

Inside maven

Step-5: In search bar search for MySQL Connector, then click on search.

Step-6: Click on MySQL connector /J.

Step-7: Click on appropriate/version of MySQL connector or check the uses or click the version which has more uses.

Step-8: In file section, we have an option called as **jar** click on that, your MySQL connector (**jar file**) will be downloaded.

Step-9: Right click on your java project, go to **build path,** then select **Configure build path.**

Step-10: Go to library section, select/click on class bar, then click on **add external jars option**

Step-11: Browser for the location of download MySQL connector jar file in your system.

Step-12: Select the jar file and then click on open, at the end click on **apply & close.**

Last: Now our java application is ready to communicate with MySQL database application.

→**Steps to use JDBC in a program:**

Step-1: We have to register the Driver.

Step-2: Open the connection between java application and database application.

Step-3: Create/prepare the statements (query to executed).

Step-4: Execute the statement (query to be executed) and process the result.

Step-5: Close the connection between java application and database application (It is just like file handling process for security purpose).

Step-6: De-register the Driver.

→**Driver (**it is a concrete class**):** com.mysql.cj.jdbc.Driver

: - It is an implementing class of driver interface.

: - The qualified name of a Driver class id com. mysql.cj. jdbc. Driver

→**URL (Uniform resource locator):** It is the path for database application with which java application is to be connected.

Syntax/Format

API: database_app_name://localhost (IP address): port_number/ [database_name]

→**API:** API represents, the technology used for database connection (JDBC Technology).

→**database_name:** It represents the name of the database application with which java application is to be connected (mysql).

→**localhost (IP address):** It represents the address/location of a system/computer inside which the database application is present.

→**port_number:** It represents the exact location of the database application inside the recognized system.

→**[database_name]:** It represents the name of the database inside the database application.

Date: 10.10.2023

- **Execute () method**

: - It is a non-static method (abstract) present inside statement interface.

: - It returns **boolean** values.

: - We can execute all types of statements using execute () method.

: - It will return true, if we execute **DQL statement** otherwise for **all others statements**, it will return **false.**

- **ExecuteUpdate () method:**

: - It is a non-static method (abstract) present inside statement interface.

: - It returns Interger type of data.

: - It returns **no. of rows affected** in the table after executing **DML Statement/command.**

: - We can execute only **DML commands** using **executeUpdate ().**

- **executeQuery () method: -**

: - It is non-static (abstract ()) present inside statements.

: - It returns the object of type **ResultSet.**

: - executeQuery () is used to execute **DQL statement.**

Date: 13.10.2023

→**JDBCInsert.java**

```
package jdbc;

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

public class JDBCInsert {
```

```
	public static void main(String[] args) throws SQLException {
		//static query
		//Register Driver

		Driver driver =new com.mysql.cj.jdbc.Driver();
		DriverManager.registerDriver(driver);


		//open connection

		Connection connection=
DriverManager.getConnection("jdbc://localhost:3306/weja3?user=root&&password=root"
);



		//create statement
		Statement statement = connection.createStatement();

		//Execute statement
		statement.execute("INSERT INTO student VALUES(1,'ram',
'ram@gmail.com',1000");

		//Process result
		System.out.println("Data inserted.");

		//Close the connection
		statement.close();
		connection.close();

		//DeRegister Driver
		DriverManager.deregisterDriver(driver);
	}

}
```

→JDBCSelect2.java:

```
package jdbc;

import java.io.File;
import java.io.FileReader;
import java.io.IOException;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.Properties;

public class JDBCSelect2 {

	private static Connection connection;
	private static Statement statement;
	private static ResultSet resultSet;
	private static String query;

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

```
		try {
			connection = openConnection();
			statement = connection.createStatement();
			query = "SELECT * FROM student";
			resultSet = statement.executeQuery(query);
			while (resultSet.next()) {
				System.out.println(resultSet.getInt(1));
				System.out.println(resultSet.getString(2));
				System.out.println(resultSet.getString(3));
				System.out.println(resultSet.getDouble(4));
			}
		} catch (ClassNotFoundException | SQLException | IOException e) {
			e.printStackTrace();
		} finally {
			try {
				closeConnection();
			} catch (SQLException e) {
				e.printStackTrace();
			}
		}

	}

	private static Connection openConnection() throws SQLException,
ClassNotFoundException, IOException {

		Class.forName("com.mysql.cj.jdbc.Driver");
		File file = new File("F:\\J2EE\\jdbcinfo.txt");
		FileReader fileReader = new FileReader(file);
		Properties properties = new Properties();
		properties.load(fileReader);
		return DriverManager.getConnection(properties.getProperty("url"),
properties);

	}

	private static void closeConnection() throws SQLException {

		if (resultSet != null) {
			resultSet.close();
		}
		if (statement != null) {
			statement.close();
		}
		if (connection != null) {
			connection.close();
		}

	}

}
```

→JDBCInsert2:

```
package jdbc;

import java.sql.Connection;
import java.sql.DriverManager;
```

```java
import java.sql.SQLException;
import java.sql.Statement;
import com.mysql.cj.jdbc.Driver;

public class JDBCInsert2 {

	private static Driver  driver;
	private static Connection connection;
	private static Statement statement;
	private static String query;

	public static void main(String[] args) {

		try {
			connection = openConnection();
			statement= connection.createStatement();
			query = "INSERT INTO student
VALUES(4,'Umesh','umesh@gmail.com',20000);";
			int row = statement.executeUpdate(query);
			System.out.println(row + "row(s) afftected.");

		} catch (SQLException e) {
			e.printStackTrace();
		}finally {
			try {
				closeConnection();

			} catch (SQLException e) {
				e.printStackTrace();
			}
		}
	}

		private static Connection openConnection() throws SQLException {
		driver = new com.mysql.cj.jdbc.Driver();
		return
DriverManager.getConnection("jdbc:mysql://localhost:3306/weja3","root", "root");
	}
		private static void closeConnection() throws SQLException {

			if (statement != null) {
				statement.close();
			}
			if (connection != null) {
				connection.close();
			}
			DriverManager.deregisterDriver(driver);
		}


}
```

→**JDBCSelect.java**

```java
package jdbc;

import java.io.File;
import java.io.FileReader;
import java.io.IOException;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.Properties;

public class JDBCSelect {

	private static Connection connection;
	private static Statement statement;
	private static String query;

	public static void main(String[] args) {

		try {
			connection = openConnection();
			statement = connection.createStatement();
			query = "SELECT * FROM student";
			statement.execute(query);
			ResultSet resultSet = statement.getResultSet();
			while (resultSet.next()) {
				System.out.println(resultSet.getInt(1));
				System.out.println(resultSet.getString(2));
				System.out.println(resultSet.getString(3));
				System.out.println(resultSet.getInt(4));
				System.out.println(resultSet.getDouble(5));
			}
		} catch (ClassNotFoundException | SQLException | IOException e) {
			e.printStackTrace();
		} finally {
			try {
				closeConnection();
			} catch (SQLException e) {
				e.printStackTrace();
			}
		}

	}

	private static Connection openConnection() throws SQLException,
ClassNotFoundException, IOException {

		Class.forName("com.mysql.cj.jdbc.Driver");
		File file = new File("F:\\J2EE\\jdbc_info.txt");
		FileReader fileReader = new FileReader(file);
		Properties properties = new Properties();
		properties.load(fileReader);
		return DriverManager.getConnection(properties.getProperty("url"),
properties);

	}

	private static void closeConnection() throws SQLException {
```

```
			if (statement != null) {
				statement.close();
			}
			if (connection != null) {
				connection.close();
			}

		}

}
```

Date: 14.10.2023

Ex: -

```
package jdbc;

import java.sql.Connection;
import java.sql.Driver;
import java.sql.DriverManager;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.Scanner;

public class JDBCInsert3 {

	private static Driver driver;
	private static Connection connection;
	private static Statement statement;
	private static String query;

	public static void main(String[] args) {

		Scanner scanner = new Scanner(System.in);
		System.out.println("Enter student id.");
		int id = scanner.nextInt();
		scanner.nextLine();
		System.out.println("Enter student name.");
		String name = scanner.nextLine();
		System.out.println("Enter student email.");
		String email = scanner.nextLine();
		System.out.println("Enter student fees.");
		double fees = scanner.nextDouble();
		scanner.close();

		try {
			connection = openConnection();
			statement = connection.createStatement();
			query = "INSERT INTO student VALUES(" + id + ",'" + name +
"','" + email + "'," + fees + ")";
			int row = statement.executeUpdate(query);
			System.out.println(row + " row(s) affected.");
		} catch (SQLException e) {
			e.printStackTrace();
		} finally {
			try {
				closeConnection();
```

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

    }

    private static Connection openConnection() throws SQLException {

        driver = new com.mysql.cj.jdbc.Driver();
        DriverManager.registerDriver(driver);
        return
DriverManager.getConnection("jdbc:mysql://localhost:3306/weja3", "root", "root");

    }

    private static void closeConnection() throws SQLException {

        if (statement != null) {
            statement.close();
        }
        if (connection != null) {
            connection.close();
        }
        DriverManager.deregisterDriver(driver);

    }

}
```

→Difference between Statement and PreparedStatement: -

Statement	PreparedStatement
1.Statement is a **parent interface of preparedStatement interface.**	1.PreparedStatement is a **child interface of statement interface.**
2.While creating the statement, **we need not to pass the query.**	2.While preparing the statement, **we need to pass a query.**
3.Methods for execution **will take a query as an argument.**	3.Methods for execution **will not take query as an argument.**
4.Statement is **not secured while executing dynamic queries.**	4.PreparedStatement is **secured while executing dynamic queries.**
5.Statement is slower as compare to preparedStatement.	**5.PreparedStatement is faster as compare to statement.**

Date: 16.10.2023

Ex: -

```
package jdbc;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.SQLException;
import java.util.Scanner;
```

```java
public class JDBCInsert4 {

	private static Connection connection;
	private static PreparedStatement preparedStatement;
	private static String query;

	public static void main(String[] args) {

		Scanner scanner = new Scanner(System.in);
		System.out.println("Enter student id.");
		int id = scanner.nextInt();
		scanner.nextLine();
		System.out.println("Enter student name.");
		String name = scanner.nextLine();
		System.out.println("Enter student email.");
		String email = scanner.nextLine();
		System.out.println("Enter student age.");
		int age = scanner.nextInt();
		System.out.println("Enter student fees.");
		double fees = scanner.nextDouble();
		scanner.close();

		try {
			connection = openConnection();
			query = "INSERT INTO student VALUES(?,?,?,?)";
			preparedStatement = connection.prepareStatement(query);
			preparedStatement.setInt(1, id);
			preparedStatement.setString(2, name);
			preparedStatement.setString(3, email);
			preparedStatement.setDouble(4, fees);
			int row = preparedStatement.executeUpdate();
			System.out.println(row + "row(s) affected.");

		} catch (SQLException e) {
			e.printStackTrace();
		} finally {
			try {
				closeConnection();

			} catch (SQLException e) {
				e.printStackTrace();
			}
		}
	}

	public static Connection openConnection() throws SQLException {
		return
DriverManager.getConnection("jdbc:mysql://localhost:3306/weja3", "root", "root");
	}

	private static void closeConnection() throws SQLException {

		if (preparedStatement != null) {
			preparedStatement.close();
		}
		if (connection != null) {
			connection.close();
		}
	}
```

```
}
```

→

; --> delimiter

→Store Procedure:

: - Like methods in java we have stored procedure in SQL.

: - Store procedure can be used to execute set of queries.

Date: 17.10.2023

→Syntax to create a store procedure: -

DELIMITER/

CREATE PROCEDURE procedure_name()

BEGIN

Statement1;

Statement2;

.

.

Statement n;

END/

DELIMITER;

→Syntax to call the store procedure: -

CALL procedure_name ();

Note: - CallableStatement interface from java.sql package can be used to call the stored procedures in the database.

: - CallableStatement interface is a child interface of PreparedStatement interface.

Date: 19.10.2023

Date: 20.10.2023

Q1: - Write a JDBC program to insert five records of a student into a table.

Ans:

```java
package jdbc.practice;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.SQLException;
import java.util.Scanner;

public class StudentJDBC {


	private static Connection  connection;
	private static PreparedStatement preparedStatement;
	private static String query;

	public static void main(String[] args) throws ClassNotFoundException {
		Scanner scanner = new Scanner(System.in);

		try {
			connection = openConnection();
			query= "INSERT INTO student VALUES(?,?,?,?)";
			preparedStatement= connection.prepareStatement(query);
			for(int i=1;i <=5; i++) {
				System.out.println("Enter student id.");
				int id = scanner.nextInt();
				scanner.nextLine();
				System.out.println("Enter student name.");
				String  name = scanner.nextLine();
				System.out.println("Enter student email.");
				String email = scanner.nextLine();
				System.out.println("Enter student fees.");
				double fees = scanner.nextDouble();
				preparedStatement.setInt(1, id);
				preparedStatement.setString(2, name);
				preparedStatement.setString(3, email);
				preparedStatement.setDouble(4, fees);
				preparedStatement.addBatch();

			}
			scanner.close();
			int[] res = preparedStatement.executeBatch();
			System.out.println(res.length + " row(s) affected.");

		} catch (SQLException e) {

			e.printStackTrace();
		}

	}

	private static Connection openConnection() throws ClassNotFoundException,
SQLException {
		Class.forName("com.mysql.cj.jdbc.Driver");
		return
DriverManager.getConnection("jdbc:mysql://localhost:3306/weja3","root","root");
	}
	private static void closeConnection() throws SQLException {
		if (preparedStatement != null) {
			preparedStatement.close();
```

```
			}
			if (connection != null) {
				connection.close();
			}
		}

}
```

Q2: - WA JDBC program to fetch the students whose name contains " esh".

Ans:

```
package jdbc.practice;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.Scanner;

public class JDBCFetchStudentName {

	private static Connection connection;
	private static PreparedStatement preparedStatement;
	private static ResultSet resultSet;
	private static String query;

	public static void main(String[] args) {

		Scanner scanner = new Scanner(System.in);

		try {
			connection = openConnection();
			query = "SELECT * FROM student WHERE Sname like '%esh'";
			preparedStatement = connection.prepareStatement(query);
			//preparedStatement.setInt(1, id);
			resultSet = preparedStatement.executeQuery();
			while (resultSet.next()) {
				System.out.println(resultSet.getInt(1));
				System.out.println(resultSet.getString(2));
				System.out.println(resultSet.getString(3));
				System.out.println(resultSet.getDouble(4));
			}
		} catch (ClassNotFoundException | SQLException e) {
			e.printStackTrace();
		} finally {
			try {
				closeConnection();
			} catch (SQLException e) {
				e.printStackTrace();
			}
		}

	}

	private static Connection openConnection() throws ClassNotFoundException,
SQLException {
```

```
            Class.forName("com.mysql.cj.jdbc.Driver");
            return
DriverManager.getConnection("jdbc:mysql://localhost:3306/weja3", "root", "root");

      }

      private static void closeConnection() throws SQLException {

            if (resultSet != null) {
                  resultSet.close();
            }
            if (preparedStatement != null) {
                  preparedStatement.close();
            }
            if (connection != null) {
                  connection.close();
            }

      }

}
```

Q3: - WA JDBC program to create a database.

Ans:

```
package jdbc.practice;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.SQLException;

public class JDBCCreateDatabase {

      private static Connection connection;
      private static PreparedStatement preparedStatement;
      private static String query;


      public static void main(String[] args) {
            try {
                  connection =openConnection();
                  query="CREATE DATABASE weja300";
                  preparedStatement=connection.prepareStatement(query);
                  int row=preparedStatement.executeUpdate();
                  System.out.println(row+ " rows affected");

            } catch (ClassNotFoundException | SQLException e) {
                  e.printStackTrace();
            }
            finally {
                  try {
                        closeConnection();
                        } catch (Exception e2) {
                              e2.printStackTrace();
```

```
                }
            }
        }
        private static void closeConnection() {


        }


        private static Connection openConnection() throws SQLException,
ClassNotFoundException {
            Class.forName("com.mysql.cj.jdbc.Driver");
            return
DriverManager.getConnection("jdbc:mysql://localhost:3306","root","root");
        }

}
```

Q4: WA JDBC program to update fees of students as 35000, who have fees greater than 50000.

Ans:

```
//Q4:  WA JDBC program to update fees of students as 35000, who have fees
greater than 50000.

package jdbc.practice;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.SQLException;

public class JDBCUpdateFees {
        private static Connection connection;
        private static PreparedStatement  preparedStatement;
        private static String query;

        public static void main(String [] args)
        {
                try {
                        connection=openConnnection();
                        query="UPDATE student SET fees=35000 where fees>5000";
                        preparedStatement=connection.prepareStatement(query);
                        int row=preparedStatement.executeUpdate();
                        System.out.println(row+ " rows affected");
                } catch (ClassNotFoundException | SQLException e) {
                        e.printStackTrace();
                }
                finally {
                        try {
                                closeConnection();
                        } catch (Exception e2) {
                        }
                }
        }

        private static void closeConnection() throws SQLException {
                if(preparedStatement!=null) {
                        preparedStatement.close();
                }
```

```
            if(connection!=null)
            {
                  connection.close();
            }
      }

      private static Connection openConnnection() throws ClassNotFoundException,
SQLException {
            Class.forName("com.mysql.cj.jdbc.Driver");
            return
DriverManager.getConnection("jdbc:mysql://localhost:3306/weja3","root","root") ;
      }



}
```

Q5: - WA JDBC program to fetch all students who are having age between 18 and 28 (make use of store procedure).

Ans:

```
//Q4:  WA JDBC program to update fees of students as 35000, who have fees
greater than 50000.

package jdbc.practice;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.SQLException;

public class JDBCUpdateFees {
      private static Connection connection;
      private static PreparedStatement  preparedStatement;
      private static String query;

      public static void main(String [] args)
      {
            try {
                  connection=openConnnection();
                  query="UPDATE student SET fees=35000 where fees>5000";
                  preparedStatement=connection.prepareStatement(query);
                  int row=preparedStatement.executeUpdate();
                  System.out.println(row+ " rows affected");
            } catch (ClassNotFoundException | SQLException e) {
                  e.printStackTrace();
            }
            finally {
                  try {
                        closeConnection();
                  } catch (Exception e2) {
                  }
            }
      }

      private static void closeConnection() throws SQLException {
            if(preparedStatement!=null) {
                  preparedStatement.close();
            }
```

```
            if(connection!=null)
            {
                  connection.close();
            }
      }

      private static Connection openConnnection() throws ClassNotFoundException,
SQLException {
            Class.forName("com.mysql.cj.jdbc.Driver");
            return
DriverManager.getConnection("jdbc:mysql://localhost:3306/weja3","root","root") ;
      }


}
```

Date: 21.10.2023

→Drawbacks of JDBC: -

1.A programmer should have a knowledge about a **database application and SQL**.
2.In order to perform a single operation on database table, **programmer has to write a boiler plate code (JDBC steps).**
3.It is not possible to achieve **direct relationship between a java object and record in the table.**

- **ORM (Object Relational Mapping) Tool: -**
  : - ORM stands for Object Relational Mapping.
  : - ORM tool can be used to overcome the draw backs of JDBC technology**.**
  : - ORM tool implements JDBC technology itself. It provides the abstraction layer between JDBC and the programmer.

→JPA

: - JPA stands for Java Persistence API.

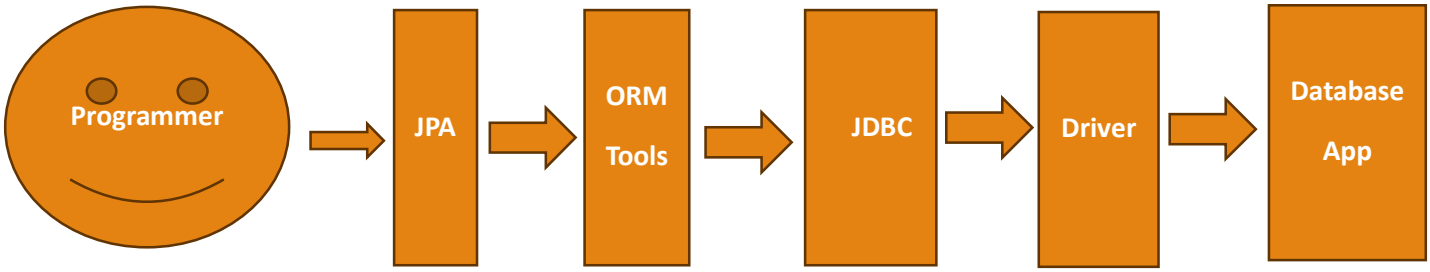
: - It is a specification which contains interfaces, abstract classes, abstract methods.

: - Like hibernate, we have several ORM tools available in the market.

: - It is the responsibility of one of these ORM tools to provide the implementation for JPA.

: - If we make use of JPA then the migration from one ORM tool to another ORM tool becomes easy because all the ORM tools follow a common standard set by JPA.

: - Without JPA migration from one ORM tool to another ORM tool is a tedious/long/lengthy process because all the ORM tools are incompatible with each other.



→**Different ORM tools as follows:**

1.Hibernate

2.Top Link

3.Elclipse link

4.I-Batis/my Bits

: - Hibernate is a widely used ORM tool.

: - JPA along with hibernate can be called **Hibernate JPA** (In this case hibernate will provide the implementation for JPA).

Date: 23.10.2023

Note: - ORM tools will map java classes to the tables inside your database and will map java objects to the records in the tables.

: - Properties of a class. Will be map as columns of a table.

→**Steps to create Maven project (archetype quick start):**

Step-1: Press **control+n**.

Step-2: one window will appear as select a wizard. There types **Maven Project.**

Now select maven project then click on next.

: - **Don't tick a check box as** 'Create a Simple project'.

Step-3: Click on next.one window will appear as

In the next window in filter section 'org. apache. maven'.

Step-4: Select **Archetype as quick start (Version 1.4)**, then click on next.

Step-5: In the next window in group id section provide Domain_name.post_name

Mention project name.

Step-6: Until the check box as '**run archetype generation interactively'**, then click on finish.

→**Structure of Maven project (archetype as quick start)**

- # Hibernate

src/main/java

src/test/java

JRE System Library

MAVEN Dependency

src

Target

Pom.xml

→**POM:** POM stands for Project Object Model.

: - pom.xml file is used to define the **dependencies**, which are required **to build a project**.

: - Pom.xml contents information about the project and the objects required to build that project.

→**Adding dependencies to com.xml**

Step-1: in browser search for maven repository.

Step-2: Click on the

Step-3: search for the required dependency.

Step-4: Select the appropriate version.

Step-5: Scroll down, there we can find a xml code format for that particular dependency.

Step-6: just single click on that code, the xml code will be copied to the click board.

Step-7: paste that xml code inside com.xml file under dependencies tag.

: - Dependencies required for hibernate project.

1st: **-** MySQL connector/j

2nd: - Hibernate core relocation

3rd: - Project Lombok [Optional].

Date: 25.10.2023

→**We have to do some modification inside the maven project so that we can use that project structure to implement hibernate logics.**

Step 1st: Select the project then press **control+n** one window will appear**.**

Step 2nd: - in a filter section **type folder** from that **select source folder**. The name of the source folder is **"src/main/resources".**

Step 3rd: Inside that source folder**,** we have to create **one general folder.** Select the created **source folder** press **control+n** some window will appear, in filter section type folder, now select **general folder.** The name of the folder is **"META-INF".**

Step 4th: Inside **META-INF** folder, we have to create on **xml file.** Select the META-INF folder then press **control+n** one window will appear inside filter section **type xml file.** Then select a xml file. The name of the xml file should be **"persistence"**.

Note: - Inside created **persistence. xml file,** we have to define **properties related** to **JDBC and properties related hibernate ORM tool.**

→Hibernate layers: -

1.DAO (Data Access Object**).**

2.DTO (Data Transfer Object**).**

1.DAO (Data Access Object**): -**This hibernates layer contains the core hibernate logic.

2.DTO (Data Transfer Object**): -** This hibernates layer contains entity classes.

Note: - Hibernate layers will be represented as packages inside the project.

Date: 26.10.2023

→@Entity

→@Id

→@Data

→@Entity: - This annotation from javax.persistence package is used to mark a class as entity (It is a class level annotation).

: - If a class is annotated with this annotation, then hibernate ORM tool will recognize this entity class and will create the respective table inside the database.

→@Id: - This annotation from javax.persistence package is used to mark one of the properties of an entity class to be treated as **Primary key** inside the table (It is property level annotation).

→@Data: - This annotation from Project Lombok is used to add getters(), setters() toString() method, equals method(), hashCode () method to the class.

INSERT OPERATION persist () method
→Student.java: -

Ex: -
```java
package com.qspiders.hibernate.dto;


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

import lombok.Data;

@Entity
@Data
public class Student {

	@Id
	private int SId;
	private String Sname;
	private String email;
	private int age;
	private double fees;
```

```
}
```

→StudentDAO.java

Ex: -

```
package com.qspiders.hibernate.dao;

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

import com.qspiders.hibernate.dto.Student;

public class StudentDAO {

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

	public static void main(String[] args) {

		openConnection();
		 entityTransaction.begin();

		 Student student = new Student();
		 student.setSId(13);
		 student.setSname("Ashish");
		 student.setEmail("ashish@gmail.com");
		 student.setAge(25);
		 student.setFees(35000);


		 entityManager.persist(student);

		 entityTransaction.commit();
		closeConnection();
	}

	private static void openConnection() {

		entityManagerFactory =
Persistence.createEntityManagerFactory("student");
		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();
                    }
                }
        }

}
```

→ SELECT OPERATION find() method
→StudentDAO.java

Ex: -

```
package com.qspiders.hibernate.dao;

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

import com.qspiders.hibernate.dto.Student;

public class StudentDAO2 {



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


        public static void main(String[] args) {

                openConnection();
                entityTransaction.begin();

                Student student = entityManager.find(Student.class,5);
                System.out.println(student);

                entityTransaction.commit();
                closeConnection();
        }


        private static void openConnection() {
                entityManagerFactory=
Persistence.createEntityManagerFactory("student");
                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();
                    }
              }
        }
}
```

→ UPDATE OPERATION 1) find () method

2) persist () method

→StudentDAO.java

Ex: -

```
package com.qspiders.hibernate.dao;

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

import com.qspiders.hibernate.dto.Student;

public class StudentDAO3 {



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

        public static void main(String[] args) {

              openConnection();
              entityTransaction.begin();

              Student student = entityManager.find(Student.class,1);
              student.setAge(25);
              entityManager.persist(student);

              entityTransaction.commit();
              closeConnection();
        }


        private static void openConnection() {

              entityManagerFactory=
Persistence.createEntityManagerFactory("student");
              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();
			}
		}

	}

}
```

→ DELETE OPERATION remove() method

→StudentDAO4

Ex: -

```
package com.qspiders.hibernate.dao;

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

import com.qspiders.hibernate.dto.Student;

public class StudentDAO4 {



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

	public static void main(String[] args) {

		openConnection();
		entityTransaction.begin();

		Student student = entityManager.find(Student.class, 3);
		entityManager.remove(student);

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

```
	}

	private static void openConnection() {

	entityManagerFactory=Persistence.createEntityManagerFactory("student");
		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();
			}
		}
	}

}
```

→**JPQL: -** JPQL stands for java persistence query language, it is used to write programmer defined queries.

: - It is similar to SQL but it uses class name as a table name & properties of a class instead of column names.

: - All the ORM tools available, have their own query languages, which are incompatible with each other.

Ex: - Hibernate uses hibernate query (HQL), I-Batis uses dynamic query language (DQL).

: - **JPQL** is understandable to all the ORM tool.

→StudentDAO5.java

Ex: -
```
package com.qspiders.hibernate.dao;

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.qspiders.hibernate.dto.Student;

public class StudentDAO5 {
```

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

	public static void main(String[] args) {


		openConnection();
		entityTransaction.begin();

		Query query = entityManager.createQuery("SELECT student from
Student student WHERE SId= ? 1");
		query.setParameter(1,2);
		@SuppressWarnings("unchecked")
		List<Student> students = query.getResultList();
		for(Student student : students) {
			System.out.println(student);
		}

		entityTransaction.commit();
		closeConnection();

	}


	private static void openConnection() {

		entityManagerFactory =
Persistence.createEntityManagerFactory("student");
		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();
			}
		}
	}
}
```

Date: 01.11.2023

➢ Hibernate Mappings: -

: - It is used to established relationship between two entities.

→Type of one mapping: -

: - unidirectional mapping: - If relationship (has a) is defined inside one of the entities is called as unidirectional mapping.

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

→Source Entity: - The class which contains an object of other class as a property is called as source entity, it is also called as **owning side.**

→Target Entity: - The class whose object is a property of another class is called as target entity. It is also called as **non-owing side.**

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

1. One to One mapping: - If instance of one entity belongs to exactly one instance of another entity is called as one to one mapping.

2.One to many mapping: - If instance of one entity belongs to more than one instances of another entity then it is called as one to many mapping.

3.Many to one mapping: - If more than one instance of one entity belongs to exactly one instance of another entity is called as many to one mapping.

4.Many to many mapping: - If more than one instance of one entity belongs to more than one instances of another entity, then it is called as many to many mapping.

Date: 04.11.2023

➔ **CompanyDAO2.java**

```
➔ package com.qspiders.onetomanyuni.dao;
➔
➔ import javax.persistence.EntityManager;
➔ import javax.persistence.EntityManagerFactory;
➔ import javax.persistence.EntityTransaction;
➔ import javax.persistence.Persistence;
➔
➔ import com.qspiders.onetomanyuni.dto.Company;
➔
➔ public class CompanyDAO2 {
➔
➔    private static EntityManagerFactory entityManagerFactory;
➔    private static EntityManager entityManager;
➔    private static EntityTransaction entityTransaction;
➔
➔    public static void main(String[] args) {
➔
➔          openConnection();
➔          entityTransaction.begin();
➔
➔          Company company = entityManager.find(Company.class, 1);
```

```
            entityManager.remove(company);

            entityTransaction.commit();
        closeConnection();

        }
        private static void openConnection() {

            entityManagerFactory =
    Persistence.createEntityManagerFactory("company");
            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();
                }
            }
        }

    }
```

→comapnyDAO3.java

```
package com.qspiders.onetomanyuni.dao;

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

import com.qspiders.onetomanyuni.dto.Company;

public class CompanyDAO3 {

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

      public static void main(String[] args) {

            openConnection();
            entityTransaction.begin();

            Company company = entityManager.find(Company.class, 1);
            System.out.println(company);

            entityTransaction.commit();
```

```
		closeConnection();

	}

	private static void openConnection() {

		entityManagerFactory =
Persistence.createEntityManagerFactory("company");
		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();
			}
		}
	}

}
```

→EmployeeDAO.java

```
package com.qspiders.onetomanyuni.dao;

import java.util.List;

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

import com.qspiders.onetomanyuni.dto.Company;
import com.qspiders.onetomanyuni.dto.Employee;

public class EmployeeDAO {

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

	public static void main(String[] args) {

		openConnection();
		entityTransaction.begin();

		Company company = entityManager.find(Company.class, 1);
		List<Employee> employees = company.getEmployees();
		for (Employee employee : employees) {
```

```
                if (employee.getId() == 2) {
                        employees.remove(employee);
                }
            }
            company.setEmployees(employees);
            entityManager.persist(company);

            Employee employee = entityManager.find(Employee.class, 2);
            entityManager.remove(employee);

            entityTransaction.commit();
            closeConnection();

      }

      private static void openConnection() {

            entityManagerFactory =
Persistence.createEntityManagerFactory("company");
            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();
                  }
            }
      }

}
```

Date: 06.11.2023

→Many to one unidirectional mapping: -

→Employee.java

```
package com.qspiders.onetomanyuni.dao;

import java.util.List;

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

import com.qspiders.onetomanyuni.dto.Company;
import com.qspiders.onetomanyuni.dto.Employee;
```

```java
public class EmployeeDAO {

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

	public static void main(String[] args) {

		openConnection();
		entityTransaction.begin();

		Company company = entityManager.find(Company.class, 1);
		List<Employee> employees = company.getEmployees();
		for (Employee employee : employees) {
			if (employee.getId() == 2) {
				employees.remove(employee);
			}
		}
		company.setEmployees(employees);
		entityManager.persist(company);

		Employee employee = entityManager.find(Employee.class, 2);
		entityManager.remove(employee);

		entityTransaction.commit();
		closeConnection();

	}

	private static void openConnection() {

		entityManagerFactory =
Persistence.createEntityManagerFactory("company");
		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();
			}
		}
	}

}
```

→Company.java

```java
package dto;

import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;

import lombok.Data;

@Entity
@Data
public class Company {

	@Id
	@GeneratedValue(strategy = GenerationType.IDENTITY)
	private int id;
	@Column(nullable = false, unique = true)
	private String name;
	@Column(nullable = false)
	private String address;

}
```

→EmployeeDAO.java

```java
package dao;

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

import dto.Company;
import dto.Employee;

public class EmployeeDAO {

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

	public static void main(String[] args) {

		openConnection();
		entityTransaction.begin();

		Company company = new Company();
		company.setName("Tata");
		company.setAddress("Pune");
		entityManager.persist(company);

		Employee employee1 = new Employee();
		employee1.setName("Ramesh");
		employee1.setEmail("ramesh@gmail.com");
		employee1.setSalary(50000);
		employee1.setCompany(company);
		entityManager.persist(employee1);
```

```
            Employee employee2 = new Employee();
            employee2.setName("Mahesh");
            employee2.setEmail("mahesh@gmail.com");
            employee2.setSalary(60000);
            employee2.setCompany(company);
            entityManager.persist(employee2);

            entityTransaction.commit();
            closeConnection();

      }

      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();
                  }
            }
      }

}
```

→EmployeeDAO2

```
package dao;

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

import dto.Employee;

public class EmployeeDAO2 {

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

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

```
		openConnection();
		entityTransaction.begin();

		Employee employee = entityManager.find(Employee.class, 2);
		entityManager.remove(employee);

		entityTransaction.commit();
		closeConnection();

	}

	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();
			}
		}
	}

}
```

→EmployeeDAO3

```
package dao;

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

import dto.Employee;

public class EmployeeDAO3 {

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

	public static void main(String[] args) {

		openConnection();
		entityTransaction.begin();
```

```
		Employee employee = entityManager.find(Employee.class, 1);
		System.out.println(employee);

		entityTransaction.commit();
		closeConnection();

	}

	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();
			}
		}
	}

}
```

Date: 07.11.2023

→many to many unidirectional mapping: -

Student.java

Ex: -

```
package dto;

import java.util.List;

import javax.persistence.CascadeType;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.ManyToMany;

import lombok.Data;

@Entity
```

```java
@Data
public class Student {

	@Id
	@GeneratedValue(strategy = GenerationType.IDENTITY)
	private int id;
	@Column(nullable = false)
	private String name;
	@Column(nullable = false, unique = true)
	private String email;
	private int age;
	@ManyToMany(cascade = CascadeType.PERSIST)
	private List<Course> courses;

}
```

→Course.java

```java
package dto;

import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;

import lombok.Data;

@Entity
@Data
public class Course {

	@Id
	@GeneratedValue(strategy = GenerationType.IDENTITY)
	private int id;
	@Column(nullable = false, unique = true)
	private String name;
	private double fees;

}
```

→StudentDAO.java

```java
package dao;

import java.util.List;

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

import dto.Course;
import dto.Student;

public class StudentDAO {
```

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

	public static void main(String[] args) {

		openConnection();
		entityTransaction.begin();

		Student student = entityManager.find(Student.class, 2);
		List<Course> courses = student.getCourses();
		Course courseToBeDeleted = null;
		for (Course course : courses) {
			if (course.getName().equals("J2EE")) {
				courseToBeDeleted = course;
			}
		}
		courses.remove(courseToBeDeleted);
		student.setCourses(courses);
		entityManager.persist(student);

		Course course = entityManager.find(Course.class, 4);
		entityManager.remove(course);

		entityTransaction.commit();
		closeConnection();

	}

	private static void openConnection() {

		entityManagerFactory =
Persistence.createEntityManagerFactory("student");
		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();
			}
		}

	}

}
```

→**Student.java**

```java
package dao;

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

import dto.Student;

public class StudentDAO2 {

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

	public static void main(String[] args) {

		openConnection();
		entityTransaction.begin();

		Student student = entityManager.find(Student.class, 2);
		entityManager.remove(student);

		entityTransaction.commit();
		closeConnection();

	}

	private static void openConnection() {

		entityManagerFactory =
Persistence.createEntityManagerFactory("student");
		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();
			}
		}

	}

}
```

Date: 08.11.2023

- **Bidirectional Mapping: -**

➔ **One to one bidirectional mapping**

→person.java

```
package dto;

import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.OneToOne;

import lombok.Getter;
import lombok.Setter;

@Entity
@Getter
@Setter
public class Person {

	@Id
	@GeneratedValue(strategy = GenerationType.IDENTITY)
	private int id;
	@Column(nullable = false)
	private String name;
	@Column(nullable = false, unique = true)
	private String email;
	@OneToOne
	private AadharCard aadharCard;

	@Override
	public String toString() {
		return "Person [id=" + id + ", name=" + name + ", email=" + email +
"]";
	}

}
```

→AadharCard.java

```
package dto;

import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.OneToOne;

import lombok.Getter;
import lombok.Setter;

@Entity
@Getter
@Setter
public class AadharCard {
```

```
	@Id
	@GeneratedValue(strategy = GenerationType.IDENTITY)
	private int id;
	@Column(nullable = false, unique = true)
	private long aadharNumber;
	@OneToOne(mappedBy = "aadharCard")
	private Person person;

	@Override
	public String toString() {
		return "AadharCard [id=" + id + ", aadharNumber=" + aadharNumber +
"]";
	}

}
```

→PersonDAO.java

```
package dao;

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

import dto.AadharCard;
import dto.Person;

public class PersonDAO {

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

	public static void main(String[] args) {

		openConnection();
		entityTransaction.begin();

		Person person = new Person();
		person.setName("Sukesh");
		person.setEmail("sukesh@gmail.com");

		AadharCard aadharCard = new AadharCard();
		aadharCard.setAadharNumber(9876543210121);
		aadharCard.setPerson(person);
		entityManager.persist(aadharCard);

		person.setAadharCard(aadharCard);
		entityManager.persist(person);

		entityTransaction.commit();
		closeConnection();

	}

	private static void openConnection() {
```

```
			entityManagerFactory =
Persistence.createEntityManagerFactory("person");
			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();
				}
			}

		}

}
```

→PersonDAO2.java

```
package dao;

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

import dto.AadharCard;
import dto.Person;

public class PersonDAO2 {

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

		public static void main(String[] args) {

			openConnection();
			entityTransaction.begin();

			Person person = entityManager.find(Person.class, 1);
			entityManager.remove(person);

			AadharCard aadharCard = person.getAadharCard();
			entityManager.remove(aadharCard);

			entityTransaction.commit();
			closeConnection();

		}
```

```
	private static void openConnection() {

		entityManagerFactory =
Persistence.createEntityManagerFactory("person");
		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();
			}
		}

	}

}
```

Date: 09.11.2023

→PersonDAO3.java

```
package dao;

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

import dto.AadharCard;

public class PersonDAO3 {

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

	public static void main(String[] args) {

		openConnection();
		entityTransaction.begin();

		AadharCard aadharCard = entityManager.find(AadharCard.class, 1);
		System.out.println(aadharCard);

		System.out.println(aadharCard.getPerson());

		entityTransaction.commit();
```

```
			closeConnection();

		}

		private static void openConnection() {

			entityManagerFactory =
Persistence.createEntityManagerFactory("person");
			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();
				}
			}

		}

}
```

→One to Many Bidirectional Mapping

→Company.java

```
package dto;

import java.util.List;

import javax.persistence.CascadeType;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.OneToMany;

import lombok.Getter;
import lombok.Setter;

@Entity
@Getter
@Setter
public class Company {

	@Id
	@GeneratedValue(strategy = GenerationType.IDENTITY)
	private int id;
```

```
	@Column(nullable = false, unique = true)
	private String name;
	@Column(nullable = false)
	private String address;
	@OneToMany(cascade = CascadeType.ALL, mappedBy = "company")
	private List<Employee> employees;

	@Override
	public String toString() {
		return "Company [id=" + id + ", name=" + name + ", address=" +
address + "]";
	}

}
```

→Employee.java

```
package dto;

import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.ManyToOne;

import lombok.Getter;
import lombok.Setter;

@Entity
@Getter
@Setter
public class Employee {

	@Id
	@GeneratedValue(strategy = GenerationType.IDENTITY)
	private int id;
	@Column(nullable = false)
	private String name;
	@Column(nullable = false, unique = true)
	private String email;
	private double salary;
	@ManyToOne
	private Company company;

	@Override
	public String toString() {
		return "Employee [id=" + id + ", name=" + name + ", email=" + email
+ ", salary=" + salary + "]";
	}

}
```

→CompanyDAO.java

```
package dao;
```

```java
import java.util.Arrays;

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

import dto.Company;
import dto.Employee;

public class CompanyDAO {

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

	public static void main(String[] args) {

		openConnection();
		entityTransaction.begin();

		Employee employee1 = new Employee();
		employee1.setName("Raju");
		employee1.setEmail("raju@gmail.com");
		employee1.setSalary(50000);

		Employee employee2 = new Employee();
		employee2.setName("Shyam");
		employee2.setEmail("shyam@gmail.com");
		employee2.setSalary(45000);

		Employee employee3 = new Employee();
		employee3.setName("Baburao");
		employee3.setEmail("baburao@gmail.com");
		employee3.setSalary(60000);

		Employee employee4 = new Employee();
		employee4.setName("Anjali");
		employee4.setEmail("anjali@gmail.com");
		employee4.setSalary(70000);

		Company company = new Company();
		company.setName("Mahindra");
		company.setAddress("Mumbai");

		employee1.setCompany(company);

		employee2.setCompany(company);

		employee3.setCompany(company);

		employee4.setCompany(company);

company.setEmployees(Arrays.asList(employee1, employee2, employee3, employee4));
		entityManager.persist(company);

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

```
	}

	private static void openConnection() {

		entityManagerFactory =
Persistence.createEntityManagerFactory("company");
		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();
			}
		}

	}

}
```

→CompanyDAO2.java

```
package dao;

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

import dto.Company;

public class CompanyDAO2 {

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

	public static void main(String[] args) {

		openConnection();
		entityTransaction.begin();

		Company company = entityManager.find(Company.class, 2);
		entityManager.remove(company);

		entityTransaction.commit();
		closeConnection();

	}
```

```
	private static void openConnection() {

		entityManagerFactory =
Persistence.createEntityManagerFactory("company");
		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();
			}
		}

	}

}
```

→CompanyDAO3.java

```
package dao;

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

import dto.Company;
import dto.Employee;

public class CompanyDAO3 {

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

	public static void main(String[] args) {

		openConnection();
		entityTransaction.begin();

		Employee employee = entityManager.find(Employee.class, 6);
		System.out.println(employee);

		Company company = employee.getCompany();
		System.out.println(company);

		entityTransaction.commit();
```

```
			closeConnection();

		}

		private static void openConnection() {

			entityManagerFactory =
Persistence.createEntityManagerFactory("company");
			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();
				}
			}

		}

}
```

Date: 10.11.2023

Many to One we have to do it by ourself.

→ Many to Many bidirectional mapping: -

→Student.java

```
package dto;

import java.util.List;

import javax.persistence.CascadeType;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.ManyToMany;

import lombok.Getter;
import lombok.Setter;

@Entity
@Getter
@Setter
```

```
public class Student {

	@Id
	@GeneratedValue(strategy = GenerationType.IDENTITY)
	private int id;
	@Column(nullable = false)
	private String name;
	@Column(nullable = false, unique = true)
	private String email;
	private int age;
	@ManyToMany(cascade = CascadeType.PERSIST)
	private List<Course> courses;

	@Override
	public String toString() {
		return "Student [id=" + id + ", name=" + name + ", email=" + email +
", age=" + age + ", courses=" + courses
				+ "]";
	}

}
```

→Course.java

```
package dto;

	import java.util.List;

	import javax.persistence.Column;
	import javax.persistence.Entity;
	import javax.persistence.GeneratedValue;
	import javax.persistence.GenerationType;
	import javax.persistence.Id;
	import javax.persistence.ManyToMany;

	import lombok.Getter;
	import lombok.Setter;

	@Entity
	@Getter
	@Setter
	public class Course {

		@Id
		@GeneratedValue(strategy = GenerationType.IDENTITY)
		private int id;
		@Column(nullable = false, unique = true)
		private String name;
		@Column(nullable = false)
		private double fees;
		@ManyToMany(mappedBy = "courses")
		private List<Student> students;

		@Override
		public String toString() {
			return "Course [id=" + id + ", name=" + name + ", fees=" +
fees + "]";
		}
```

```
	}
```

→StudentDAO.java

```
package dao;

import java.util.Arrays;

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

import dto.Course;
import dto.Student;

public class StudentDAO {

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

	public static void main(String[] args) {

		openConnection();
		entityTransaction.begin();

		Course course1 = new Course();
		course1.setName("SQL");
		course1.setFees(10000);

		Course course2 = new Course();
		course2.setName("CORE JAVA");
		course2.setFees(15000);

		Course course3 = new Course();
		course3.setName("WEB TECH");
		course3.setFees(20000);

		Course course4 = new Course();
		course4.setName("J2EE");
		course4.setFees(25000);

		Student student1 = new Student();
		student1.setName("Ramesh");
		student1.setEmail("ramesh@gmail.com");
		student1.setAge(25);

		Student student2 = new Student();
		student2.setName("Suresh");
		student2.setEmail("suresh@gmail.com");
		student2.setAge(26);

		Student student3 = new Student();
		student3.setName("Mahesh");
		student3.setEmail("mahesh@gmail.com");
		student3.setAge(22);
```

```
            Student student4 = new Student();
            student4.setName("Umesh");
            student4.setEmail("umesh@gmail.com");
            student4.setAge(24);

            course1.setStudents(Arrays.asList(student1, student2));
            course2.setStudents(Arrays.asList(student3, student4));
            course3.setStudents(Arrays.asList(student1, student4));
            course4.setStudents(Arrays.asList(student2, student3));

            student1.setCourses(Arrays.asList(course1, course3));
            entityManager.persist(student1);
            student2.setCourses(Arrays.asList(course1, course4));
            entityManager.persist(student2);
            student3.setCourses(Arrays.asList(course2, course4));
            entityManager.persist(student3);
            student4.setCourses(Arrays.asList(course2, course3));
            entityManager.persist(student4);

            entityTransaction.commit();
            closeConnection();

      }

      private static void openConnection() {

            entityManagerFactory =
Persistence.createEntityManagerFactory("student");
            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();
                  }
            }

      }

}
```

→StudentDAO2.java

```
package dao;

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

```java
import javax.persistence.Persistence;

import dto.Student;

public class StudentDAO2 {

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

	public static void main(String[] args) {

		openConnection();
		entityTransaction.begin();

		Student student = entityManager.find(Student.class, 4);
		entityManager.remove(student);

		entityTransaction.commit();
		closeConnection();

	}

	private static void openConnection() {

		entityManagerFactory =
Persistence.createEntityManagerFactory("student");
		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();
			}
		}

	}

}
```

→StudentDAO3.java

```java
 package dao;

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

```java
import dto.Student;

public class StudentDAO3 {

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

	public static void main(String[] args) {

		openConnection();
		entityTransaction.begin();

		Student student = entityManager.find(Student.class, 3);
		System.out.println(student);

		entityTransaction.commit();
		closeConnection();

	}

	private static void openConnection() {

		entityManagerFactory =
Persistence.createEntityManagerFactory("student");
		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();
			}
		}

	}

}
```

Date: 11.11.2023

14.11.2023

→PersonDAO2

```java
package dao;

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

import dto.AadharCard;
import dto.Person;

public class PersonDAO2 {

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

    public static void main(String[] args) {

        openConnection();
        entityTransaction.begin();

        // Find the Person entity by its ID
        Person person = entityManager.find(Person.class, 2);

        if (person != null) {
            // Remove the Person entity
            entityManager.remove(person);

            // Access and remove the associated AadharCard entity
            AadharCard aadharCard = person.getAadharCard();
            if (aadharCard != null) {
                entityManager.remove(aadharCard);
            }
        } else {
            System.out.println("Person entity with ID 1 not found.");
        }

        entityTransaction.commit();
        closeConnection();
    }

    private static void openConnection() {
        entityManagerFactory = Persistence.createEntityManagerFactory("person");
        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();
            }
        }
```

```
    }
}
```

→PersonDAO3.java: -

```
package dao;

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

import dto.AadharCard;

public class PersonDAO3 {

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

	public static void main(String[] args) {

		openConnection();
		entityTransaction.begin();

		AadharCard aadharCard = entityManager.find(AadharCard.class, 1);
		System.out.println(aadharCard);

		System.out.println(aadharCard.getPerson());

		entityTransaction.commit();
		closeConnection();

	}

	private static void openConnection() {

		entityManagerFactory =
Persistence.createEntityManagerFactory("person");
		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();
			}
		}
```

```
        }
}
```

Date: 15.11.2023

Date: 20.11.2023

➢ Servlets:

→Web Application/Web-based Application: -

: Web Application: - An application, which is present inside a server and which can be access through internet using standard web browsers is called as web application/ web-based application.

➔ Web Browser

: - It an application, which is used to access resources (data or the information) available on the internet.

➔ **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 application.

➔ **Web Resources: -**

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

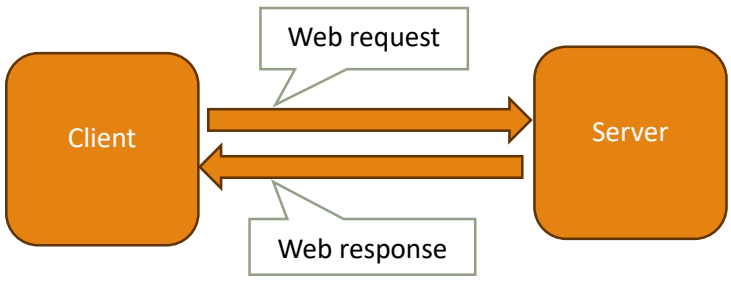
➔ **Static web resource:**

: - The data/information in the web resource will not change with respect to the user then it is called as static web resource.

➔ **Dynamic web resource: -**

: - The data/information in the web resource will change with respect to user then it is called as dynamic web resource.

➔ **Client server architecture: -**



➔ **Web request: -** If the request is sent from the client to the server using standard web browsers for specific web resource, then it is called as web request.

➔ **Web response:** - Response, which is sent back to the client from the server for the received request, then it is called as web response.

Note: - Using HTML, we can **build only static web application**.

: - In order to build dynamic web application, **we make use of servlets and JSPs.**

➔ **Servlet: -** Servlet is the technology or a **java API,** which can be used to accept web request inside the java application and the sent back web response from the java application.

➔ **Servlet in java: -** Servlet is a **special class** in java.

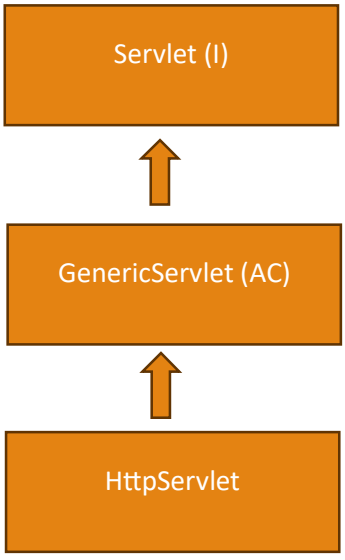
➔ **Creating servlet in java: -** Servlets can be created in **three ways**:

1st: - By implementing **Servlet interface.**

2nd: - By extending **GenericServlet abstract class.**

3rd: - By extending **HttpServlet partially abstract class.**

➔ **Servlet Hierarchy**



→URL (Uniform resources locater).

: - It is used to send the request to the server for specific web resource.

➔ **Format for URL: -**

protocol://host_name:port_no/path_to_resource?Query_String#fragment_id

→**protocol**: - Protocol defines/represents rules and regulations to send the request.

→**host_name**: - It represents the server on which an application is hosted.

→**port_no: -** It represents the exact location of an application.

→**path_to_resource: -** It represents the location of the resource present inside an application.

→**Query_String: -** It represents the data to be sent along with the request to the server.

→**fragment_id**: - It represents the unique id associated with each and every web resource.

Date: 21.11.2023

→**Step to install Apache tomcat 9**

1st: - In web browser search for Apache tomcat 9 download.

2nd: - Click on the 1st link and in the redirected page under core section, click on the appropriate link based on system configuration (32bit or 64bit) to download the Zip file.

3rd: - Extract the downloaded zip file.

4th: - In the Eclipse under server section click on the link to add the server.

5th: - Select the appropriate name of the server provider under that select the version we have downloaded then click on next.

6th: - In the next window browse for the location where zip file has been extracted.

7th: - Select the extracted folder and then click on next.

8th: - Click on next and then click on finish.

➔ **Step to create dynamic web project**

1st: - Press **control+n** and inside filter section **type dynamic**.

2nd: - From the **web folder** select the **dynamic web project** and click on next.

3rd: - Provide the name for the project and **check for the runtime environment.**

4th: - Then click on **next**.

5th: - Again, click on **next**, now in web module page **click the check box to generate web.xml file**

6th: - Then click on **finish**.

➔ **Structure of dynamic web project.**

>servlets

>Development Descriptor: servlets

>JAX-WS Web Services

>Java Resources

>Build

>src

->main

-java

->webapp

-META-INF

->WEB-INF

-lib

web.xml

Date: 22.11.2023

→Creating a servlet by extending generic servlet abstract class.

→MyServlet1.java

```java
package com.qspiders.servlets;

import java.io.IOException;

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

@WebServlet("/MyServlet1")

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
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
	xmlns="http://xmlns.jcp.org/xml/ns/javaee"
	xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
http://xmlns.jcp.org/xml/ns/javaee/web-app_4_0.xsd"
	id="WebApp_ID" version="4.0">
```

```
	<display-name>servlets</display-name>
	<welcome-file-list>
		<welcome-file>index.html</welcome-file>
		<welcome-file>index.jsp</welcome-file>
		<welcome-file>index.htm</welcome-file>
		<welcome-file>default.html</welcome-file>
		<welcome-file>default.jsp</welcome-file>
		<welcome-file>default.htm</welcome-file>
	</welcome-file-list>
	<servlet>
		<servlet-name>MyServlet1</servlet-name>
		<servlet-class>com.jspiders.servlets.servlet.MyServlet1</servlet-
class>
	</servlet>
	<servlet>
		<servlet-name>MyServlet2</servlet-name>
		<servlet-class>com.jspiders.servlets.servlet.MyServlet2</servlet-
class>
	</servlet>
	<servlet>
		<servlet-name>MyServlet3</servlet-name>
		<servlet-class>com.jspiders.servlets.servlet.MyServlet3</servlet-
class>
	</servlet>
	<servlet>
		<servlet-name>MyServlet4</servlet-name>
		<servlet-class>com.jspiders.servlets.servlet.MyServlet4</servlet-
class>
	</servlet>
	<servlet-mapping>
		<servlet-name>MyServlet1</servlet-name>
		<url-pattern>/MyServlet1</url-pattern>
	</servlet-mapping>
	<servlet-mapping>
		<servlet-name>MyServlet2</servlet-name>
		<url-pattern>/MyServlet2</url-pattern>
	</servlet-mapping>
	<servlet-mapping>
		<servlet-name>MyServlet3</servlet-name>
		<url-pattern>/MyServlet3</url-pattern>
	</servlet-mapping>
	<servlet-mapping>
		<servlet-name>MyServlet4</servlet-name>
		<url-pattern>/MyServlet4</url-pattern>
	</servlet-mapping>
</web-app>
```

→Creating a servlet by extending HttpServlet class

→Myservelt2.java

```
package com.qspiders.servlets;

import java.io.IOException;

import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
```

```java
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

@WebServlet("/MyServlet2")

public class MyServlet2 extends HttpServlet{

	private static final long serialVersionUID = 1L;

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

		System.out.println("Hello Ankit!!!");
	}
}
```

→**Web.xml**

```xml
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
	xmlns="http://xmlns.jcp.org/xml/ns/javaee"
	xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
http://xmlns.jcp.org/xml/ns/javaee/web-app_4_0.xsd"
	id="WebApp_ID" version="4.0">
	<display-name>servlets</display-name>
	<welcome-file-list>
		<welcome-file>index.html</welcome-file>
		<welcome-file>index.jsp</welcome-file>
		<welcome-file>index.htm</welcome-file>
		<welcome-file>default.html</welcome-file>
		<welcome-file>default.jsp</welcome-file>
		<welcome-file>default.htm</welcome-file>
	</welcome-file-list>
	<servlet>
		<servlet-name>MyServlet1</servlet-name>
		<servlet-class>com.jspiders.servlets.servlet.MyServlet1</servlet-
class>
	</servlet>
	<servlet>
		<servlet-name>MyServlet2</servlet-name>
		<servlet-class>com.jspiders.servlets.servlet.MyServlet2</servlet-
class>
	</servlet>
	<servlet>
		<servlet-name>MyServlet3</servlet-name>
		<servlet-class>com.jspiders.servlets.servlet.MyServlet3</servlet-
class>
	</servlet>
	<servlet>
		<servlet-name>MyServlet4</servlet-name>
		<servlet-class>com.jspiders.servlets.servlet.MyServlet4</servlet-
class>
	</servlet>
	<servlet-mapping>
		<servlet-name>MyServlet1</servlet-name>
		<url-pattern>/MyServlet1</url-pattern>
	</servlet-mapping>
```

```
	<servlet-mapping>
		<servlet-name>MyServlet2</servlet-name>
		<url-pattern>/MyServlet2</url-pattern>
	</servlet-mapping>
	<servlet-mapping>
		<servlet-name>MyServlet3</servlet-name>
		<url-pattern>/MyServlet3</url-pattern>
	</servlet-mapping>
	<servlet-mapping>
		<servlet-name>MyServlet4</servlet-name>
		<url-pattern>/MyServlet4</url-pattern>
	</servlet-mapping>
</web-app>
```

Note: - in java javax. servlet package represents java servlet API.

Since HttpServlet is a subclass of generic servlet will prefer creating servlet using HttpServlet class (Since using HttpServlet we ca access the features of generic servlet along with added features of HttpServlet).

→Difference between GenericServlet and HttpServlet

GenericServlet	HttpServlet
1. GenericServlet is an abstract class, present inside javax. Servlet package.	1. HttpServlet is a partially abstract class present inside javax. servlet. HttpSerlet package.
2. GenericServlet is a parent class of HttpServlet.	2.HttpServlet is implementing class of genericServlet.
3. We have to override service () method to provide the implementation for the servlet.	3.We have to override Http () methods to provide the implementation for servlet.
4. GenericServlet is protocol independent.	**4.HttpServlet is protocol dependent.**

→Http () Methods

1. **doPost() : -** When user wants to send some data along with the web request to the server then we make use of doPost () method.
2. **doGet ():** - When user wants to retrieve some data from the server then we make use of doGet () method.

: - Whenever a request is sent for the servlet created using HttpServlet, then by default doGet () method will be executed.

Date: 25.11.2023

→student_info.html

```
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>Insert title here</title>
```

```
</head>
<body>
	<div align="center">
		<form action="student_info" method="post">
			<table border="1px solid">
				<tr>
					<td>Name</td>
					<td><input type="text" name="name"></td>
				</tr>
				<tr>
					<td>Email</td>
					<td><input type="text" name="email"></td>
				</tr>
				<tr>
					<td>SQL</td>
					<td><input type="checkbox" name="course"
value="SQL"></td>
				</tr>
				<tr>
					<td>Core Java</td>
					<td><input type="checkbox" name="course"
value="Core Java"></td>
				</tr>
				<tr>
					<td>Web Technology</td>
					<td><input type="checkbox" name="course"
						value="Web Technology"></td>
				</tr>
				<tr>
					<td>J2EE</td>
					<td><input type="checkbox" name="course"
value="J2EE"></td>
				</tr>
			</table>
			<input type="submit" value="Submit">
		</form>
	</div>
</body>
</html>
```

→MyServlet3.java

```
package com.qspiders.servlets;

import java.io.IOException;

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("/MyServlet3")

public class MyServlet3 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 {
		System.out.println("Hey Buddy!!!!");
	}
}
```

→MyServlet4.java

```
package com.qspiders.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("/MyServlet4")

public class MyServlet4 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("Welcome to the servlet Mr. Ankit!!!");
	}
}
```

→MyServlet6.java

```
package com.qspiders.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;
```

```java
@WebServlet("/add")
public class MyServlet5 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 of " +num1 + "and" + num2 + " = " + sum +
"</h1>");
	}
}
```

→**MyServlet6.java**

```java
package com.qspiders.servlets;

import java.io.IOException;

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("/student_info")
public class MyServlet6 extends HttpServlet {

	private static final long serialVersionUID = 1L;

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

		String name = req.getParameter("name");
		System.out.println(name);
		String email = req.getParameter("email");
		System.out.println(email);
		String[] courses = req.getParameterValues("course");

			for(String course : courses) {
				System.out.println(course);
			}

	}
}
```

→**MyServlet7.java**

```java
package com.qspiders.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("/add2")
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 of" + num1 + " and " + num2 + " = " + sum +
"</h1>");
	}
}
```

→Dispatching request from one servlet to another servlet

Include (): - This method is used to dispatch the request from one servlet to another .

: - Responses of both the servlets will be included.

→MyServlet8.java

```java
package com.qspiders.servlets;

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

import javax.servlet.RequestDispatcher;
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("/MyServlet8")
public class MyServlet8 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>Response from MyServlet8.</h1>");

		RequestDispatcher requestDispatcher =
req.getRequestDispatcher("MyServlet9");
		requestDispatcher.include(req, resp);
	}
}
```

→MyServlet9.java

```
package com.qspiders.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("/MyServlet9")
public class MyServlet9 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>Response from MyServlet9.</h1>");
	}
}
```

→ forword ()

: - This method is use to dispatch the request from one servlet to another.

: - Response of a servlet from which the request is getting dispatched will be **excluded** and the response of the servlet to which request has been dispatch will be included.

→MyServlet10.java

```
package com.qspiders.servlets;

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

import javax.servlet.RequestDispatcher;
```

```java
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("/MyServlet10")
public class MyServlet10 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>Response from MyServlet10.</h1>");

		RequestDispatcher requestDispatcher =
req.getRequestDispatcher("MyServlet11");
		requestDispatcher.forward(req, resp);

	}
}
```

→MyServlet11.java

```java
package com.qspiders.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("/MyServlet11")

public class MyServlet11 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>Response from MyServlet11.</h1>");
	}
}
```

→**sendRedirect (): -** This method is used to dispatch the request from one servlet to another servlet.

: - It works like Forword () method but a new request will be generated for the servlet to which the request has to be dispatch.

→Myservlet12.java

```java
package com.qspiders.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("/MyServlet12")
public class MyServlet12 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>Response from Myservlet12.</h1>");

		resp.sendRedirect("MyServlet13");
	}
}
```

→MyServlet13.java

```java
package com.qspiders.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("/MyServlet13")
public class MyServlet13 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>Response from Myservlet13.</h1>");
}
}
```

Date: 27.11.2023

→Servlet Life Cycle

: - Whenever a request is sent to a servlet, that servlet **undergoes some steps/phases**

1. **Class loading process for servlet: -**

: - When the request is sent for particular servlet then that particular servlet class will be loaded.

2. **Instantiation for servlet: -**

: - An object for servlet class will be created.

3. **Initialization of servlet: -**

: - In this phase **init () method** of servlet will be called and executed.

4. **Service will be provided: -**

: - In this phase **service () method** will be called and executed.

5. **Destruction of servlet: -**

: - In this phase **destroy () method** will be called and executed.

→Throughout servlet life cycle only service will be provided multiple times, otherwise all other phases will take place only once.

Note: - Servlet container will manage servlet life cycle. Servlet container will create an object for servlet class and will invoke life cycle methods of servlet.

Ex: - package com.jspiders.servlets.servlet;

Import javax.servlet.Servlet;

.

.

.

.

@WebServlet(“/MyServlet14”)

```
Public class MyServlet14 implements Servlet {
Static {
SOPln(“Class is loaded.”);
}
{
SOPln(“Object is created.”);
}
@override
Public ServletConfig getServletConfig () {
Return null;
}
@override
Public String getServletInfo() {
Return null;
}
@Override
Public void init(ServletConfig config) throws SerlvetException{
SOPln(“init () is in invoked.”);
}
@override
Public void service(SerlvetRequest req, SerlvletResponse res) throws ServletException, IOException{
SOPln(“service () is invoked.”);
}
@override
Public void destroy () {
SOPln(“destroy is () invoked.”)
}
}
```

❖ JSP: -

: - JSP stands for **Java Server Pages.**

Date: 28.11.2023

→**Servlet: -** Inside servlet the focus is more on java code rather than HTML code.

→**JSP: -** In JSP the more focus is on HTML code rather than java code.

→**JSP Tags**

1. **Declaration tag (<%! %>)**
2. **Scriptlet tag (<% %>)**
3. **Expression tag (<%= %>)**
4. **Directive tag (<%@ %>)**
5. **Action tags:**

I.JSP Include

II.JSP Forword

1.Declaration tag (<%! %>): - This tag is used to declare java variables and methods inside JSP.

2.Scriptlet Tag (<% %>): - This tag is used to define java logic inside JSP.

3.Expression tag (<%= %>): - This is used to print variables on the browser.

4.Directive tag (<% @ %>): - This tag is used to make use of import statements inside JSP.

5.Action tags: Action tags are used to perform some action on JSP files.

I.JSP include (jsp: include page=" "): - This tag is used to dispatch request from one JSP file to another JSP files.

Here, responses of both JSP files will be included

II.JSP Forword (jsp : forwords page=" "): - This tag is used to dispatch the request from one JSP file to another JSP files.

Here, the response of a JSP file from which the request has been dispatch is excluded and response of a JSP file to which the request has been dispatch is included.

Note: - JSP file must be created under webapp folder.

→Add.html

```
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>Insert title here</title>
</head>
<body>
	<div align="center">
		<form action="add.jsp">
			<table border="1px solid">
				<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>
```

→Add.jsp

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<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 =
		<%=sum%></h1>
</body>
</html>
```

→MyJSP1.jsp

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>Insert title here</title>
</head>
<body>
      <h1>Response from MyJSP1</h1>
</body>
</html>
<jsp:include page="MyJSP2.jsp"></jsp:include>
```

→MyJSP2.jsp

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>Insert title here</title>
</head>
<body>
      <h1>Response from MyJSP2</h1>
</body>
</html>
```

→MyJSP3.jsp

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>Insert title here</title>
</head>
<body>
      <h1>Response from MyJSP2</h1>
</body>
</html>
<jsp:include page="MyJSP4.jsp"></jsp:include>
```

→MyJSP4.jsp

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
```

```
<title>Insert title here</title>
</head>
<body>
	<h1>Response from MyJSP4</h1>
</body>
</html>
```

28.11.2023:

- **JSP life Cycle: -**

1. JSP will be translated into servlet class by JSP translator.
2. The respective servlet class of translated JSP will be loaded.
3. Object will be created for that servlet class.
4. jspInt () method will be invoked.
5. jspService () method will be invoked.
6. jspDestroy () method will be invoked.

Note: - Except service phase of a JSP life cycle all other phases will take place only once throughout the life cycle.

: - And service phases will take place multiple times.

Note: - The java logic present inside JSP file will be loaded inside jspServlet () method of a servlet class.

02.12.2023: