Contents

Java programming

- Java is a high level, class based, OOP language.
- By using Java, various types of applications are capable of developing.
    1. Java → Desktop apps / standalone apps
    2. JSP → Web apps
    3. Native android → mobile apps

What is JVM?

- Java virtual machine (JVM) enables you to run java applications.
- Java compiler will convert high level java code into bytecode, then JVM will convert bytecode into machine code.

- Java is a platform independent language because java code will execute on top of JVM.
- Because of that, java can be executed in any operating system.

What is JRE?

- Java runtime environment (JRE) is a software package which provides class libraries, JVM and other necessary components to run Java applications.

What is JDK?

- Java development kit (JDK) is a software development kit which contains JRE, compiler, JavaDoc etc.
- JDK will convert high level program to machine code in java application development.
- To execute java applications in any computer JDK is an essential thing.

Java variables

- A variable is a memory allocation of random-access memory which is capable of holding relevant data in it.
- Declaring variables will create memory allocation in RAM.
- Each variable should have a unique variable name which works as an identifier to identify them separately.

Java data types

- Data type will decide what type of data which works with declared variables.
- In Java all variables should declared before it uses.
- In Java following data types are available,
  - → int – works with integer data
  - → String – works with string values / words.
  - → double – works with numerical data with floating points.
  - → bool – stores 1,0 or true, false.

Example:
```
int num=10;
String name=abc;
Double no=12.5;
```

Java operators

- operator is a symbol which we are using in programming to meet certain requirements.
- Depending on the contribution of these operators it can be categorized as follows.
  1) Arithmetic operators → +, -, /, *
  2) Assignment operators → =
  3) Relational operators → <, >, <=, >=
  4) Logical operators → &&, ||

Java basic inputs and outputs

Java outputs

In java there are 3 in built functions can be used to display content on java applications.

System.out.println();

This method will display content and move the cursor to the next line.

System.out.print();

This method will display content and remain the cursor on the same line.

System.out.printf();

This method is similar with printf method in C programming.

Example:

```
public static void main(String[] args) {
        System.out.println("Hello world");
    }
```

Printing variables in Java

```
public static void main(String[] args) {
        double number = 10.6;
        System.out.println(5);
        System.out.println(number);
```

```
public static void main(String[] args) {
        double number = 10.6;
        System.out.println("I am " + "awesome");
        System.out.println("number = "+number);
    }
```

Q] Create a java application to declare two integer variables and initialize them with values. Get the answers for the basic arithmetic operations and display them on the application.

```
public static void main(String[] args) {
      int num1=10;
      int num2=5;

       System.out.println("Summation is "+(num1+num2));
       System.out.println("Subtraction is "+(num1-num2));
       System.out.println("division is "+(num1/num2));
       System.out.println("Multiplication is "+(num1*num2));
   }
```

Java inputs

- In Java user inputs has been taken in a different way.
- When taking user inputs following methods should followed.
    1. Import the class library file
    2. Create a class object using imported library
    3. Through the created object get user inputs.
- Above class library is known as java.util.Scanner;
- When taking multiple inputs one class object is required inside the class file.
- When expecting the user input values to the java application scanner object should create as follows,

    Scanner obj = new Scanner(system.in);

In above object creation object name is the only variable that can be change.

```
import java.util.Scanner;    //should add this library file to get user inputs. (Before the main class)


public class Mavenproject1 {

   public static void main(String[] args) {
    Scanner obj = new Scanner(System.in); //connect through an object (obj is the object name here)

    //getting float input
     System.out.println("Enter float: "); // What should user insert as the input?
     float myfloat = obj.nextFloat(); ();   //In the next line, user will insert the input.
     System.out.println("Float entered = "+ myfloat); //displaying the final output.


    //getting double input
     System.out.println("Enter float: ");
     double myDouble = obj.nextDouble();
     System.out.println("Double entered = "+ myDouble);

    //getting String input
     System.out.println("Enter text: ");
     String mystring = obj.next();
     System.out.println("Text entered = "+ mystring);
   }
```

1. Enter your name and display it

```
public static void main(String[] args) {
    Scanner input = new Scanner(System.in);

     System.out.println("Enter your name: ");
     String name = input.next();
     System.out.println("Your name is "+ name);
  }
```

2. create a Java application to get two numerical inputs from the user and display the multiplication value from the app.

```
public static void main(String[] args) {
    Scanner object = new Scanner(System.in);
     System.out.println("Enter first number: ");
     int num1 = object.nextInt();
     System.out.println("Enter second number: ");
     int num2 = object.nextInt();

     System.out.println("Multiplication is "+ (num1*num2));
  }
```

3. Check whether the user inserted number is odd or even.

```
public static void main(String[] args) {
    Scanner object = new Scanner(System.in);
     System.out.println("Enter a number: ");
     int num = object.nextInt();

     if(num%2==0)
        System.out.println("number is even");
     else
        System.out.println("number is odd");
  }
```

Java loops

- Loops can be used in iterative programs in java programming.
- Loops are capable of repeating the given code until the condition is true.
- In Java programming for loops, while loops, and do while loop are commonly used.

For loop

For loop consists with three expressions and until the condition is true loop will iterate.

Syntax:

```
for (iteration; condition; increment/decrement)
                {
                        Body of the loop;
                }
```

While loop

This loop again can be used for iterative programs until the condition is true.

Syntax

```
while(condition)
                {
                        Body of the loop;
                        Increment/ decrement;
                }
```

Do while loop

This loop will execute at least one iteration even the condition is false.

Reason for that is the condition of do while loop is checked at the end.

Syntax:

```
do
                {
                        Body of the loop;
                        Increment/ decrement;
                }
                While(condition);
```

Q] display first 10 natural numbers with 3 loops.

```
public static void main(String[] args) {
    int a;
    for(a=1;a<=10;a++)
    {
       System.out.println(a);
    }
  }
```

```
public static void main(String[] args) {
    int a=1;
    while(a<=10)
    {
       System.out.println(a);
       a++;
    }
  }
```

```
public static void main(String[] args) {
    int a=1;
    do
    {
       System.out.println(a);
       a++;
    }
    while(a<=10);
  }
```

2. display 100, 90, 80…. With 3 loops.

```
public static void main(String[] args) {
    int x;
    for(x=100;x>=0;x=x-10)
    {
       System.out.println(x);
    }
  }
```

```
public static void main(String[] args) {
    int a=100;
    while(a>=0)
    {
       System.out.println(a);
       a=a-10;
    }
```

```
public static void main(String[] args) {
    int x=100;
    do
    {
       System.out.println(x);
       x=x-10;
    }
    while(x>=0);
  }
```

Java arrays

- Arrays are capable of holding multiple elements in same data type.
- When declaring an array valid array name and a data type should provide.
- Giving the size of the array is not mandatory at the point of declaration.

0	1	2	3	4	5
23	17	11	8	7	29

```
public static void main(String[] args) {
    Scanner obj  = new Scanner(System.in);   //Connecting through an object
    int [] MyArray = new int [10];   //defining the array. Same as C#

    for(int i=0;i<10;i++)
      {
        System.out.println("Enter a value");   //getting the user inputs for 10 times by using for loop
        MyArray[i] = obj.nextInt();   //in the next line, user will insert the input.
      }
      for(int y=0;y<10;y++)
      {
        System.out.println(MyArray[y]);  //display the values by using this for loop.
      }
  }
```

PRACTICAL 01

Java Empty Array

- When declaring an array first the data type should mention with square bracket, and it should follow with a valid array name.
- Syntax

  Data Type [ ] validArrayName ;

  Ex: String [] cars;

Declaring an array with a fixed size

- In Java arrays can be created with a fixed size.
- Which means values cannot be inserted more than the defined size of the array.

  int [] intArray = new int [20];

Task 01

A) **Declare an integer array size of 10 and take the user inputs to the array. Using an appropriate loop display the values in the array.**
B) **Find the maximum value inside the array.**
C) **Display the above array values in reverse order.**
D) **Find out how many odd numbers and even numbers contain inside the array.**
E) **Declare another integer array size of 10. Multiply the above array elements by 10 and assign them to the new array. Display the new array values.**

*[In the code, answer is in each color]

```
public static void main(String[] args) {
      Scanner obj = new Scanner(System.in);
      int [] MyArray = new int [10];
       int max=0;
       for(int i=0;i<10;i++) //enter values
          {
             System.out.println("Enter a value");
             MyArray[i] = obj.nextInt();
          }

          for(int y=0;y<10;y++) //display values
          {
             System.out.println(MyArray[y]);
          }

          for(int i=0;i<10;i++) //find max
          {
             if(MyArray[i]> max)
             {
                max=MyArray[i];
             }
          }
          System.out.println("Maximum values is "+ max);

          for(int y=9;y>=0;y--) //reverse order
          {
             System.out.println(MyArray[y]);
          }

          int even=0,odd=0;
          for(int z=0;z<=9;z++) //odd even count
          {
             if(MyArray[z]%2==0)
                even=even+1;
             else
                odd=odd+1;
          }

          System.out.println("Even count = " + even);
          System.out.println("Odd count = " + odd);

          int [] newarray = new int[10]; //multiply 10
             {
           for(int x=0;x<=9;x++)
           {
              newarray[x] = MyArray[x]*10;
           }
           for(int y=0;y<10;y++)
          {
             System.out.println(newarray[y]);
          }
              }
       }
```

Sort the above created new array in ascending order.

```
//ascending order
        int j,key;
        for(int i=0;i<10;i++)
        {
           j=i-1;
           key=data[i];

           while(j>=0 && data[j]>key)
           {
              data[j+1]=data[j];
              j=j-1;
           }
           data[j+1]=key;
        }
        for(int i=0;i<10;i++)
        {
           System.out.println(data[i]);
        }
```

1. create a Java application to get the size of an integer array from the user and based on that create an integer array. Get the user inputs to the array from the user and find the maximum value which contains inside the array.

```
public static void main(String[] args) {
      Scanner obj = new Scanner(System.in);
      int size;
      int max=0;
       System.out.println("Enter the size of the array: ");
       size=obj.nextInt();

       int [] myarray = new int[size];
       for(int i=0;i<size;i++)
       {
          System.out.println("Enter a value: ");
          myarray[i]=obj.nextInt();
       }
        for(int i=0;i<size;i++) //find max
          {
             if(myarray[i]> max)
             {
                max=myarray[i];
             }
          }
          System.out.println("Maximum values is "+ max);
}
```

Java classes

- Java class is an external file which contains in your java project.
- The class file is a blueprint for the objects.
- By using a single class file, N number of objects can be created which helps to do the code reusability.

Creating a Java class

- Java classes can be created with the keyword called ''class''.
- The boundary of the class will define by using curly brackets.
- A class can contain two members called fields/ variables & methods/ functions.

- Fields are being used to store data and methods are being used to perform some operations.

Structure of a class

```
class CassName{
        //fields
        //methods
    }
```

Example:

```
class MyClass{
      //variable can be declared --> global variables
      public void MyMethod()
      {
         //variable --> local
      }
    }
```

```
class Bicycle{
        //state or field
        private int gear = 5;

        //behavior or method
        public void braking(){
           System.out.println("Working of braking");
        }
    }
```

Java Objects

- An object is called an instance of a class. Which means object can be used as a reference tool to the class component.
- By using a single class N number of class objects can be created.

Syntax:

```
className object = new className( );
```

example

```
//for Bicycle class
    Bicycle sportsBicycle = new Bicycle();
    Bicycle touringBycycle = new Bicycle();
    }
```

Accessing members of a class

- To access a member from another class, class object and the .operator will be used.
- But the accessibility will be decided by the access specifier of the member.

Example

```
class Bicycle{
      //field of class
      int gear = 5;

      //method of class
      void braking(){
         ...
      }
   }

   //create object
   Bicycle sportsBicycle = new Bicycle();

   //access field and method
   sportsBicycle.gear;
   sportsBicycle.braking();
```

Java methods

- Methods can be used to perform special functionalities inside an application.
- Methods can be used to divide a large program into small code chunks which can be used again inside the application.
- In Java there are two types of methods.
    1. User defined methods – developers/ users will create these methods based on the requirements.
    2. Standard library methods – these are known as in-built methods which available from imported class libraries.
- In Java, default specifier is public but in C# its private.

Declaring a java method

- A method should have,
    1. Access specifier
    2. Return type
    3. Valid method name
    4. Parameters if applicable
    5. Method body

Syntax

```
returnType methodName ( ) {
//method body
}
```

```
public class App1 {

   public static void main(String[] args) {
        int num1 = 25;
        int num2 = 15;

        //create an object of Main
        App1 obj = new App1();
        //calling method
        int result = obj.addNumbers(num1,num2);
        System.out.println("sum is " + result);
   }
```

```
//create a method
        public int addNumbers(int a, int b)
        {
           int sum = a+b;
           //return value
           return sum;
        }
```

1. Create a Java application and add a class to display hello world.

```
public class NewClass {
    public void mymethod()
    {
        System.out.println("Hello world");
    }
}
```

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

        NewClass obj = new NewClass();
        obj.mymethod();
    }
```

2. Create a java application to get username and age to display in a separate java class file.

```
public class NewClass {
    public void mymethod()
    {
        Scanner obj = new Scanner(System.in);
        System.out.println("Enter your name: ");
        String name = obj.next();
        System.out.println("Enter your age: ");
        int age = obj.nextInt();
        System.out.println("Your name is "+ name);
        System.out.println("Your age is "+ age);
    }
}
```

```
public class Mavenproject3 {

    public static void main(String[] args) {

        NewClass obj = new NewClass();
        obj.mymethod();
    }
}
```

3. Create a Java application to get two numerical values and display the summation in the class itself.

```
public class NewClass {
    public void mymethod()
    {
        Scanner obj = new Scanner(System.in);
        System.out.println("Enter number one: ");
        int num1 = obj.nextInt();
        System.out.println("Enter number two: ");
        int num2 = obj.nextInt();

        System.out.println("Summation is "+
(num1+num2));
    }
}
```

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

        NewClass obj = new NewClass();
        obj.mymethod();
    }
```

PRACTICAL 02

2. Create a java application to check whether user inserted number 01 is divisible by user inserted number 02 and display the output in same class file.

```
public void mymethod()
   {
      Scanner obj = new Scanner(System.in);
      System.out.println("Enter number one: ");
      int num1 = obj.nextInt();
      System.out.println("Enter number two: ");
      int num2 = obj.nextInt();

     if(num1%num2==0)
     {
        System.out.println("number one is divisible by number 02");
     }
     else
         System.out.println("not divisible");
   }
```

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

      NewClass obj = new NewClass();
      obj.mymethod();
   }
```

1. Create a Java application to check whether user is eligible for voting once user's age passed as parameter to the separate class. Display the output inside the separate class.

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

      Scanner obj = new Scanner(System.in);
      System.out.println("Enter your age: ");
      int age = obj.nextInt();

     NewClass objc = new NewClass();
     objc.mymethod(age);
   }
```

```
public void mymethod(int Age)
   {
      if(Age>=18)
      {
         System.out.println("Eligible for voting");
      }
      else
      {
         System.out.println("Not eligible for voting");
      }
   }
```

3) Create a Java application to get two numerical inputs and pass them as parameters to another class. Inside the added class check the maximum number and return the maximum value out of the function. Display the returned value accordingly.

```
public int mymethod(int number1, int
number2)
   {
      if(number1 > number2)
         return number1;
      else
         return number2;
   }
```

```
public static void main(String[] args) {
     Scanner obj = new Scanner(System.in);
     System.out.println("Enter number 01");
     int num1 = obj.nextInt();
     System.out.println("Enter number 02");
    int num2 = obj.nextInt();

     NewClass objc = new NewClass();
    int max = objc.mymethod(num1, num2);
     System.out.println("max is"+max);
  }
```

4. Create a java application to get the size of an array and pass that value as a parameter to another class. Inside the class create an array based on parameter value and get user inputs to the array. From the class, return the summation of 0^{th} index and the last index of the array and display the summation inside the main method.

```
public class NewClass {
   public int mymethod(int Size)
   {
      Scanner obj = new Scanner(System.in);
      int[] myarr = new int[Size];
      for(int i=0;i<Size;i++)
      {
         System.out.println("Enter a value: ");
         myarr[i]= obj.nextInt();
      }
      return myarr[0]+myarr[Size-1];
   }
}
```

```
public static void main(String[] args) {
       Scanner obj = new Scanner(System.in);
       System.out.println("Enter the size");
       int size= obj.nextInt();


        NewClass objc = new NewClass();
      int sum= objc.mymethod(size);
       System.out.println("Sum is"+sum);
    }
```

Java encapsulation

- Encapsulation refers to hiding data inside a single class but make available the value of the variable using method implementation.
- Data hiding is a way of restricting the access of our data members by hiding the implementation details.

Getters and setters

- Get method will always be a public method which returns the value of the private variable.
- Set method is again a public method which set some values to private variables.
- Set method does not return any value out of the function and most of the time, it is a parameterized method.

  Example:

```
public int Sum;
public void setsum(int a)
{
                Sum = a;
}
Public int getsum()
{
    return Sum;
}
```

Q] create a class called EncapData.java and create two private variables to store radius value and pi value.

Inside the main class you have to get the radius value from the user and pass it to EncapData.java class.

Inside EncapData.java class create getters and setters to find the area of the circle and to find the circumference of the circle.

Return the answers from the EncapData class. Display the answers inside the Main class.

```
public class EncapData {
   private double rad;
   private double pi=3.14;

   public void setRad(double radius)
   {
      rad=radius;
   }
   public double getArea()
   {
      return pi * rad * rad;
   }
   public double getCircumference()
   {
      return 2 * pi * rad;
   }
}
```

```
public class Mavenproject4 {

   public static void main(String[] args) {
      Scanner obj = new Scanner(System.in); //creating an object
      System.out.println("Enter the radius value: "); //getting radius value
      double userradius = obj.nextDouble();  // declaring radius value

      EncapData objEncap = new EncapData(); //creating an object
      objEncap.setRad(userradius); //calling radius value through the object

      System.out.println("Area is "+ objEncap.getArea()); //display area
      System.out.println("Circumference is "+ objEncap.getCircumference()); //display circumference
   }
}
```

Java inheritance

- In inheritance we are creating a new class by using an existing class.
- In such situation, existing class will become parent class or base class. New class will become child class or derived class.
- All the public, protected fields and methods which contain inside parent class, will available to the child class.
- In Java extends keyword will use to perform inheritance.

Syntax:

```
class Animal {
        //methods and fields
}
//use of extends keyword
//to perform inheritance

class Dog extends Animal {
//methods and fields of Animal
//methods and fields of Dog
}
```

- You can customize the child class by adding methods and fields to the child class.

Java inheritance is – a relationship

- Inheritance can identify by using the keyword is a.
- This keyword will use when explaining questions and tasks in computing.

  Example: Orange is a fruit.

  In above example orange will become the child class and fruit is the parent class.

Task 01:

Create a console application with two added classes called animal and cat. Cat is the derived class and animal is the base class. Inside the animal class create a method. Inside the method print I am an animal. Inside the cat class create a method and display I have four legs. Inside the main method create relevant class object and display as follows.

I am an animal I have four legs.

```
public class Animal {
   public void animalmethod()
   {
      System.out.print("I am an animal ");
   }
}
```

```
public class Cat extends Animal {
   public void catmethod()
   {
      System.out.print("I have four legs");
   }
}
```

```
public static void main(String[] args) {
      Cat objb = new Cat();
      objb.animalmethod();
      objb.catmethod();
   }
```

Task 02:

Modify the above program by declaring a variable inside the animal class an assigning an integer value 4 to the variable. Get the following print statement by using modified application.

I am an animal I have 4 legs

```
public class Animal {
   public int legs=4;
   public void animalmethod()
   {
      System.out.print("I am an animal ");
   }
}
```

```
public class Cat extends Animal {
   protected void catmethod()
   {
      System.out.print("I have " + legs + "legs");
   }
}
```

```
public static void main(String[] args) {
      Cat objb = new Cat();
      objb.animalmethod();
      objb.catmethod();
   }
```

Java polymorphism

- In OOP programming you cannot include two or more methods with same method signature inside a single class.
- The reason for that is when methods get called, compiler will confuse with the method that you are trying to call.

  Example:

```
Class student
{
        public void StudentMarks (int y)    → method signature
        {
                -
        }
        public void StudentMarks (int x)
        {
                -
        }
}
```

In above example, both given methods are containing the same method signature. Because of that in OOP programming, you are not allowed to use it like that.

In Polymorphism it describes how a single method, can use in different formats.

- In OOP Polymorphism can be used in two different ways.
    1. Method overriding
    2. Method overloading

Java method overriding

- Method overriding always use with inheritance OOP concept.
- Inside the parent class it's possible to declare the method and inside the child class, same method signature can be declared by overriding parent class method.
- Since it uses inheritance, both parent class method and child class method available to the child class.

Example:

```
Class language {
        public void displayInfo() {
                System.out.println("Common English Language");
                }
        }
Class java extends language {
@override
Public void displayInfor() {
        System.out.println("Java programming language");
        }
}

Class Main {
Public static void Main(String [] args) {
//create an object of java class
Java j1 = new java();
J1.displayInfo();

//create an object of language class
Language l1 = new language();
L1.displayInfo();
}
}
```

- In above example to call the super class method an object should create using super class name.
- To call the child class method an object should create by using child class name.

Java method overloading

- In this polymorphism type we can contain n number of methods with same method name by differentiating the parameter list. This concept is known as method overloading.
- In this concept same method will perform different operations based on the parameters.

  Example:

```
void func() { ...}
void func(int a) {...}
void func(double a) {...}
void func(int a, float b) {....}
```

PRACTICAL 03

07/ 10/ 2022

Q] Create a Java application with two added classes called employee and manager. Manager is a employee and inside the employee class create a method to get employee's name and age as user input values. Override the same method inside manager class to get same user inputs. Run the application to get employee and manager details separately.

```
public class Manager extends Employee {
   public void emp()
   {
      Scanner obj = new Scanner(System.in);
      System.out.println("Enter your name: ");
      String EmpName = obj.next();
      System.out.println("Enter your age: ");
      int EmpAge = obj.nextInt();
   }
}
```

```
public class Employee {
   public void emp()
   {
      Scanner obj = new Scanner(System.in);
      System.out.println("Enter your name: ");
      String EmpName = obj.next();
      System.out.println("Enter your age: ");
      int EmpAge = obj.nextInt();
   }
}
```

```
public class Mavenproject5 {
   public static void main(String[] args) {
      Employee objE = new Employee();
      objE.emp();

      Manager objM = new Manager();
      objM.emp();
   }
}
```

Java Constructors

- Constructor is a special method type which using OOP programming.
- Constructor does not have a return type and always constructor name should be class name.
- Constructors will execute at the point of object creation by using the class name.
- There are two types of constructors in OOP programming.
    1. Default constructor
    2. Parameterized constructor

Default constructor

- There should be only one default constructor in any given class.
- Default constructor can have an access specifier and usually it's public.

```
Class Manager
      {
       Public Manager()
      {
         //can perform any operation
       }
```

Parameterized constructor

- There can be n number of parameterized constructors in a single class by differentiating parameter list.
- Usually, parameterized constructors define under public access specifier.

```
Class Manager
      {
       Public Manager(int x)
      {
         //can perform any operation
       }
   Public Manager(String y, double x)
      {
          //can perform any operation
      }
```

Calling default constructor

- When creating the class object from another class default constructor will get called.
  Ex: Manager ObjManager= new Manager()

Calling parameterized constructor

- When creating the class object parameters should pass accordingly to get call parameterized constructor.
  Ex: - Manager objMg = new Manager(45);
  Manager objMg = new Manager("Alex",34.25);

Task 01

Create a java application which contain class called student. Inside the student class there should be a variable to store student name. Inside the default constructor initialize the student's name as Alex. Add a parameterized constructor to the same class which passes a string value to the constructor. Display the student's name by using both constructors. Create the objects accordingly and call both constructors to display the name.

```
public class Student {
   String name;

   public Student()
   {
      name = "Alex";
      System.out.println(name);
   }
   public Student(String x)
   {
      name = x;
      System.out.println(name);
   }
}
```

```
public class Mavenproject5 {
   public static void main(String[] args) {
      Student obj = new Student();
      Student objS = new Student("Sam");
   }
}
```

Task 01

Create a java application which contains a class called Student with three constructors as follows.

1. Default constructor which takes students name as a user input.
2. Parameterized constructor which stores the student ID inside Student class variable.
3. Parameterized constructor which overrides student name.

Create the main class accordingly and run the application.

Java abstraction

- In this OOP concept, we are hiding the implementation details of a method and showing only the functionality to the user.
- For an example, when sending a message, we only bother about sending and receiving messages rather than worrying about the backend process. In abstraction, similar kind of thing happens.

Abstract classes and methods

- In OOP programming, abstraction can be achieved by using abstract classes or interfaces.
- When implementing abstraction, a keyword called abstract will use which is not an access specifier. This keyword will use in classes and methods.

Abstract class

- Abstract class is a restricted class which cannot be access by other classes using class objects.
- Abstract class should have at least one abstract method.
- An abstract class can also contain non abstract methods also.
- When accessing abstract classes, it should access using inheritance.

Abstract method

- Abstract methods do not have implementation. In abstract class abstract methods are only defining.
- When defining abstract methods accessibility level and return type can be decided.
- The body of the abstract method will be written inside child class using method overriding concept.

  Example:

```
absract class Animal{
      public abstract void animaSound();
      public void sleep()
      {
         System.out.println("Zzz");
      }
   }
```

- Using above animal class, we're not capable of creating a class object since animal class is an abstract class.

  Example:

```
class Pig extends Animal {
   public void animalSound()
   {
      //the body of animalsound() is provided here
      System.out.println("The pig says: wee wee");
   }
}
```

```
abstract class Animal {
   //abstract method (does not have a body)
   public abstract void animalSound();
   //Regular method
   public void sleep()
   {
      System.out.println("Zzz");
   }
}
```

```
public static void main(String[] args) {
      Pig myPig = new Pig(); //create a
pig object
      myPig.animalSound();
      myPig.sleep();
   }
```

1. Add an abstract method to the animal class called animalWeight. override the method inside sub class. Inside the sub class display animal weight is 50kg.
2. Try to create an object by using abstract class name and check the possibility. \
   When creating class objects by using abstract classes an application is throwing a syntax error. Because of that creating objects by using abstract classes are not possible.

Task 02

Create an abstract class called employee and add an abstract method called employeeInfo. Manager is an employee and inside manager class override the abstract method and get manager's name as an input. Display the manager's name inside override method.

```
public static void main(String[] args) {
      Manager objm = new Manager();
      objm.employeeInfo();
   }
```

```
abstract class employee {
   public abstract void employeeInfo();
}
```

```
class Manager extends employee{
   public void employeeInfo()
   {
      Scanner obj = new Scanner(System.in);
      System.out.println("Enter manager's name ");
      String name = obj.next();
      System.out.println("Mnager's name is "+ name);
   }
}
```

PRACTICAL 03

14/ 10/ 2022

Java interfaces

- Interface is another way of implementing abstraction in java.
- Inside an interface we can declare methods, but methods do not have implementation. Which means an interface is a completely abstract class.

```
interface Animal
  {
     public void animalSond(); //interface method (does not have a body)
     public void run(); //interface method (does not have a body)
  }
```

Above mentioned methods will automatically become abstract methods since it did not contain any implementations.

- To access the methods inside interfaces another class should added, and that class should implements from the interface by using the keyword implements. In above operation something similar to inheritance happens an instead of extends keywords implements keyword will use.
- The body of abstract methods should provide inside implement class.

```
public class Mavenproject8 {
   public static void main(String[] args) {
      Pig mypig = new Pig();
      mypig.animalSound();
      mypig.sleep();
   }
}
```

```
public interface Animal {
   public void animalSound();
   public void sleep();
}
```

```
public class Pig implements Animal{
   public void animalSound()
   {
      System.out.println("The pig says: ");
   }
   public void sleep()
   {
      System.out.println("ZZZ");
   }
}
```

Modify the above program by adding an abstract method called AnimalShape. And get user inputs the color of the animal and display it inside the abstract method implementation.

Create an abstract method called animalFeet which takes an integer parameter that is number of feet which animal contains. Get the user input from the user and display the number of feet inside abstract method.

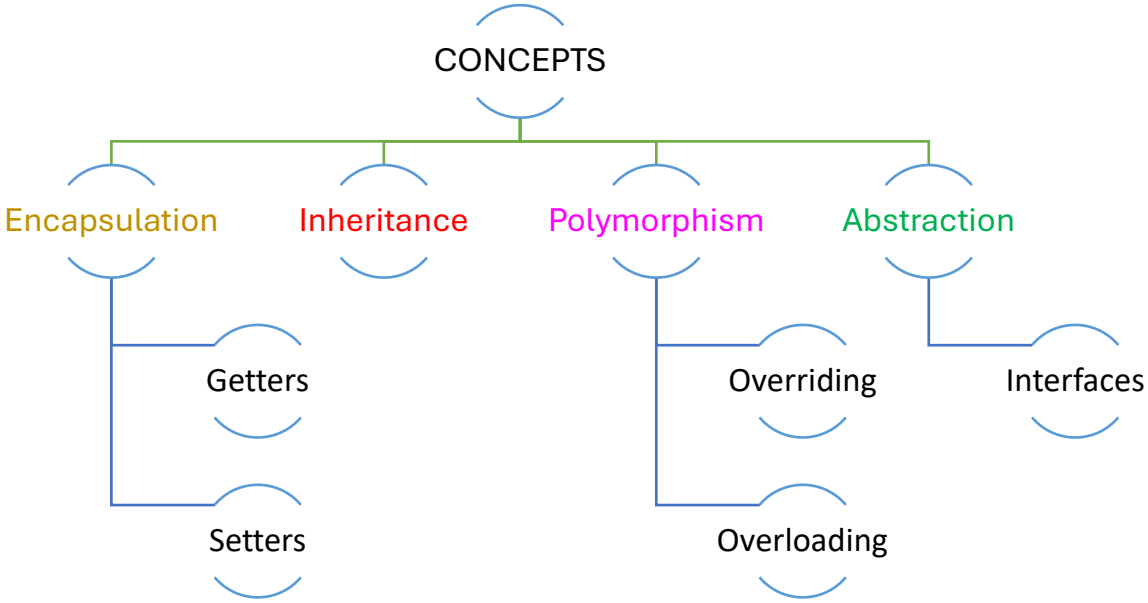
Modify the above program by adding an abstract method called AnimalAge which takes age of the animal as an integer parameter. From the method check whether animal is an adult or child and return one or zero accordingly. From the main method, check whether the return value is one, and display the output as an adult. If return value is zero, output value should be child.

```
public interface Animal {
   public void animalSound();
   public void sleep();
   public void AnimalShape();
   public int AnimalFeet(int feet);
   public int AnimalAge(int age);
}
```

```
public static void main(String[] args) {
      Pig mypig = new Pig();
      mypig.animalSound();              //one
      mypig.sleep();                  //two
      mypig.AnimalShape();              //three

      Scanner obj = new Scanner(System.in);
      System.out.println("enter the number of feet of the animal ");
      mypig.AnimalFeet(feet);           //four


      System.out.println("enter age: ");
      int age = obj.nextInt();

      if(mypig.AnimalAge(age)==1)       //five
         System.out.println("adult");
      else
         System.out.println("child");
   }
```

```
public class Pig implements Animal{
   public void animalSound()
   {
      System.out.println("The pig says: ");
   }
   public void sleep()
   {
      System.out.println("ZZZ");
   }
   public void AnimalShape()
   {
      Scanner obj = new Scanner(System.in);
      System.out.println("enter the color of the animal ");
      String color = obj.next();
      System.out.println("Color is "+color);
   }
   public int AnimalFeet(int feet)
   {

      System.out.println("Number of feet is "+feet);
      return 0;
   }
   public int AnimalAge(int age)
   {

      if(age>=18)
         return  1;
      else
         return 0;
   }
}
```

Short Note




```
public int Sum;
public void setsum(int a)
{
        Sum = a;
}
Public int getsum()
{
    return Sum;
}
```

```
class Animal {
        //methods and fields
}

class Dog extends Animal {
//methods of Animal
//methods of Dog
}
```

```
Class student
{
        public void SMarks (int y)
        {
                -
        }
        public void SMarks (int x)
        {
                -
        }
}
```

```
absract class Animal{
    public abstract void
animaSound();
    public void sleep()
    {

System.out.println("Zzz");
    }
  }
```

Encapsulation: hiding data inside a class but make available the value of variable using implementation.

Setters: with parameter no return type

Getters: no parameters with return type

Inheritance: All the public, protected fields and methods which contain inside parent class, will available to the child class.

Keyword extends

Inheritance: All the public, protected fields and methods which contain inside parent class, will be available to the child class.

Keyword extends

Polymorphism – how a single method can use in different formats.

Overriding – same method name, but the implementation is different.

Overloading – same method name, different parameters

Abstraction: can't create objects.

Keyword abstract

Interfaces: no implementation.

Keyword implements

Constructors: Default → same as class name

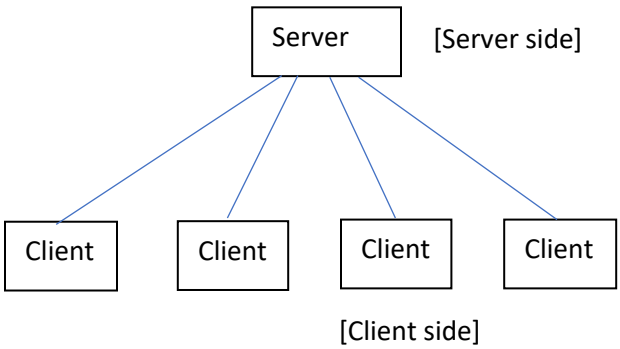
Parameterized → differentiate parameter list

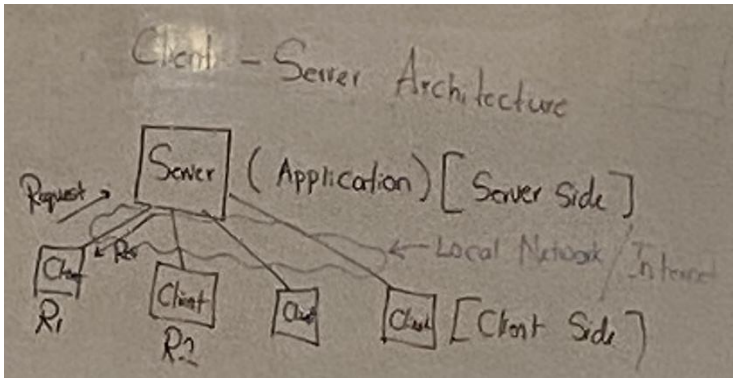
Java servlet

- Java servlets are working based on client server architecture.
- Which means servlets are resides on server side.

Client – server architecture

- This is a software architecture which used in application development.
- In this architecture there are two main components called client and server.
- Server is a dedicated device which contains all application data and clients are connecting to the server and making requests from the server.
- In this architecture there can be n number of clients who are connecting to a single server.
- Clients are making request to the server and server will process them, then respond back to the client computer over the network (Internet).
- In java servlets it discusses about a technology called server-side programming/ server sides scripting.



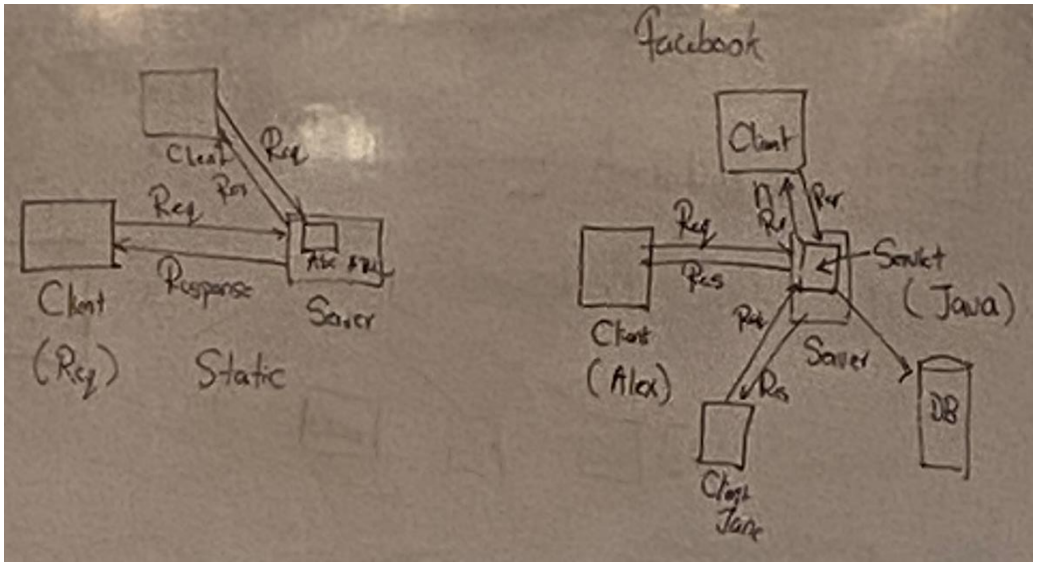


Java servlet

- Servlet is a technology which used to do server-side programming as a server-side scripting language to create dynamic web pages.
- Servlet technology is using java programming language. Because of that this is a robust and scalable technology.
- Before servlets a technology called common gateway interface (CGI) is used. CGI is a server-side scripting language which has identical drawbacks.

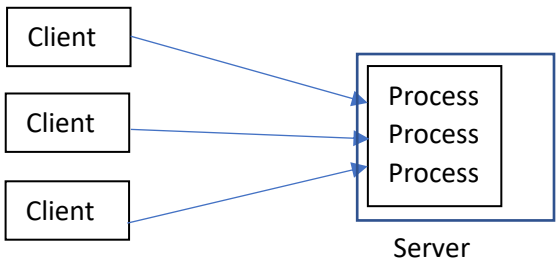
What is a Servlet?

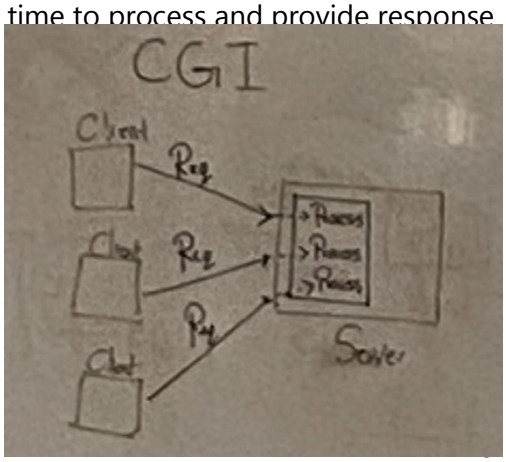
- Servlets are being used to create dynamic web applications because the content of dynamic pages are different from user to user.
- Servlet will do the configuration based on the user with the help of database and will create dynamic web pages in client side.
- The main advantage of servlet is we cannot configure specific hard coded web pages specifically user to user. Servlets will help to do dynamic configurations for specific users when creating dynamic pages.



CGI (Common Gateway Interface)

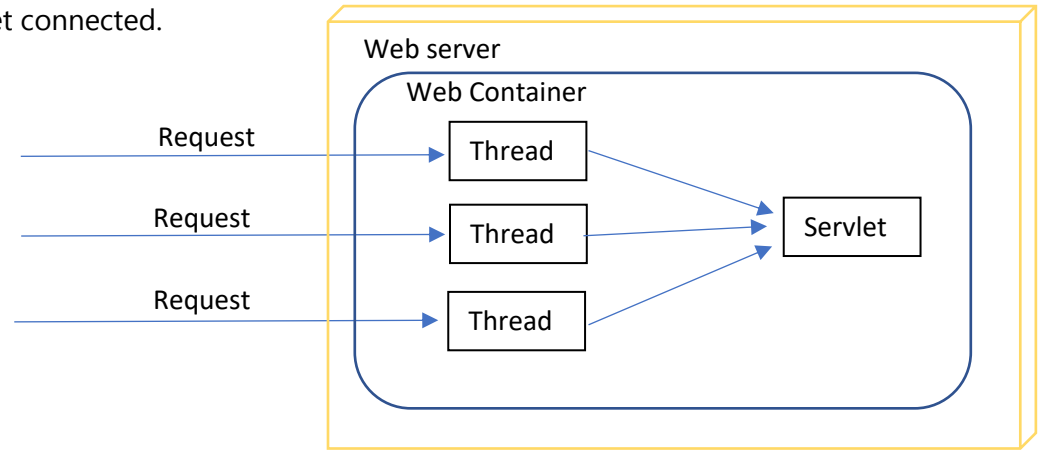
- In CGI for each client request there will be a dedicated process.
- Since processors are heavy weight, it takes more time to execute and provide a response to the client computer.
- If there are large number of requests, it takes more time to process and provide response because memory will get overloaded.





Advantages of servlets

- In servlets each request will convert as a thread inside the server.
- Since threads are lightweight, and easy to process threads will execute quicker than processors and provide response back to the client.
- Since threads are lightweight server won't get overloaded even though number of clients get connected.



Web terminologies

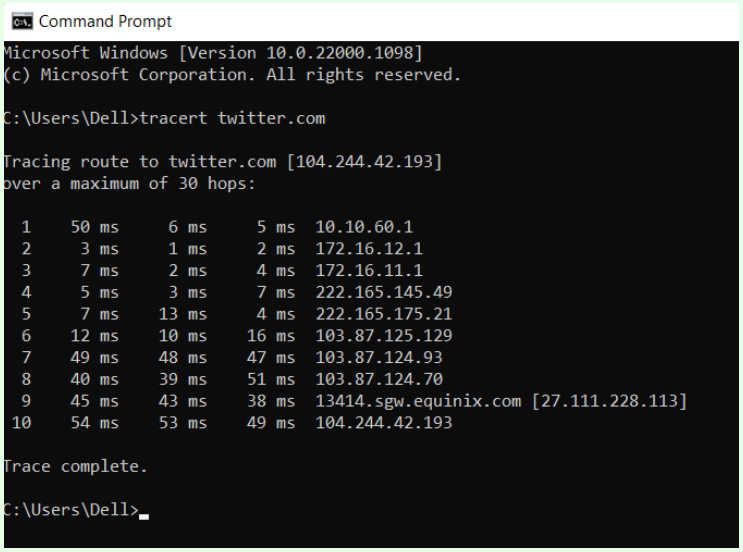
1. HTTP - This protocol used to establish connection between the client and server.
    - In modern day HTTPS protocol also used for a secured connection between client and server.
2. HTTP Request - This is the request which sent by client to server by containing all relevant information.
3. Container - This using Java to create dynamic web pages in server site

19/ 10/ 2022

PRACTICAL 04

Website

1. Website is a collection of web pages which interconnected together.
2. Each website has a most prominent page which identified as the home page of the website.
3. Each website has a dedicated URL and IP address which used to access the website.
4. There are two types of websites,
    - Static websites
    - Dynamic websites



Q] find the IP addresses of following websites.

1. youtube.com 172.217.194.91
2. Facebook.com 157.240.15.35
3. Twitter.com 104.244.42.193

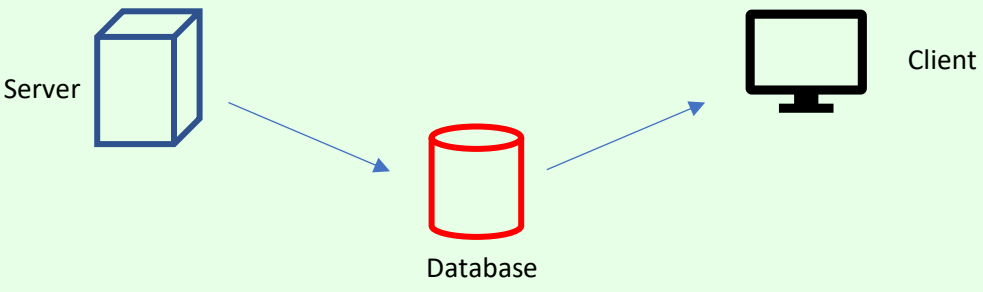
`

Static websites

- The content of the web page will not change from user to user when using static web pages.
- Static pages are directly connected with servers and retrieve data.

Dynamic websites

- Information of dynamic websites will change from user to user.
- This differentiation has been done by using a database or content management system (CMS).



1. Create a java class to calculate and display the area of a shape.
    a. Create two methods to calculate the area of a rectangle and a circle using method overloading.
    b. Display the values.

```
public class Shape {

   public void area(int length, int width)
   {
      System.out.println("Area of a rectangle is "+ length*width);
   }
   public void area(int radius)
   {
      System.out.println("Area of a circle is"+ 3.14*radius*radius);
   }
}
```

```
public static void main(String[] args) {
      Scanner obj = new Scanner(System.in);


      Shape objx = new Shape();
      objx.area(7);
      objx.area(45, 7);
   }
```

2. Create a class called animal.
    a. Implement 2 methods named make_sound and move.
    b. Create 3 subclasses extending the animal class called cat, fish and bird.
    c. Assign unique methods which are specific to its own class.
    d. Create objects from the extended classes and print their unique sounds and moves using the methods from the super class.

HTTP request

- In client server applications clients are making requests from the server. This request is known as HTTP request.
- This request is transmitting over the network and in real world scenario there are multiple requests are received by the server.
- Since this request follows HTTP protocol the behavior of this request will define by the protocol.
- http request contains multiple information among those following three should be essential.
    1. Request line – this contains the information which requested by the server computer.
    2. Source IP address and port – since there are multiple clients connected to the server, server should identify to which client it should response.

       Since client computer also capable of running two separate applications servers should know to which application it should response. From the port number that can be identified.
    3. Destination IP address and port number – this will define to which server the request should made. IP address of the server will define the server location of the network.

       In some situations, the server might contain number of applications. In that situation destination port number will define to which application the request should make.

HTTP request methods

- When creating form applications, we have to include attributes such as action, method.
- Action will define to which page these submitted data should transfer.
- When transferring the data, it uses a rule set under HTTP protocol.
- Request methods will define how the transaction should happen.
- In HTTP there are multiple request methods such as GET, POST, HEAD, TRACE etc.
- Above request methods are having different levels of configuration. Among those GET and POST request are commonly used.

GET vs POST request

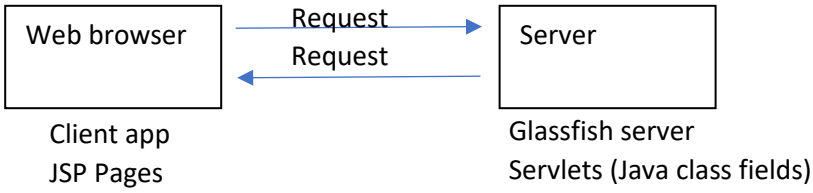
- Get request will display all submitted data on the URL, which is not a secured method since also submitted data can monitor through the URL history.
- Post request method will not expose the submitted data since it cannot be monitored later. Because of that post is a secured method.
- In client sever web development, get method also used under different requirements.

JSP Pages

- JSP is a technology to create the front end of a web application. JSP is capable of containing HTML, JSP tags.
- JSP pages will connect with servlets which contains the logical program of the web application.
- By using JSP expressions java code can also include inside it.
- Theoretically JSP page will execute as the client application and servlets will execute as the server application.

Virtual servers

- This will help to create a client server environment inside the IDE.
- Java web application consists with JSP pages and servlets.
- When execute a java web application JSP pages will loaded into web browser and servlets are execute in virtual servers.
- In Java web there are two main servers known as
    1. Glassfish server
    2. Apache tom cat server
- When configuring the project, you have to select which server that you use.
- Depending on the selected server servlets will get loaded to the virtual server and creates a client server environment.



- In industrial application development server will be located remote place and it will transact data using a network
- In IDEs servers and the JSP files are located in same device and execute inside local host.

09/11/2022 lec9

JSP

01. create a java web application to display your name, batch and degree program in 3 separate lines.

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-
8">
    <title>JSP Page</title>
  </head>
  <body>
    <h2>Chamali Ranasinghe</h2>
    <h2>Software engineering</h2>
    <h2>21.1</h2>
  </body>
</html>
```

02. use internal CSS and modify the above three text lines as you wish.

```
<h2 style="color:red; text-align: center">Chamali Ranasinghe</h2>

    <h2 style="color:green; text-align: center">Software
engineering</h2>

    <h2 style="color: blueviolet; text-align: center">21.1</h2>
```

03. Change the background color into blue and font color into white.

```
<title>JSP Page</title>
      <style>
         body
         {
            background-color: darkblue;
         }
         h2
         {
            color: white;
         }
      </style>
   </head>
   <body>
      <h2 >Chamali</h2>
      <h2 >Software engineering</h2>
      <h2 >21.1</h2>
```

04. Add two pages and link it.

```
<body>
      <h1>Page 01</h1>
      <a href="page02.jsp">click</a>
   </body>
```

```
<body>
      <h1>Page 02</h1>
      <a href="index.jsp">back</a>
   </body>
```

05. create a form

```
<body>
    <h1 style="text-align: center">Sign up</h1>
    <table>
    <form method="post" action="">
      <tr>
        <td>
          Name
        </td>
        <td>
          <input type="text" name="Uname"><br>
        </td>
      </tr>
      <tr>
        <td>
          Age
        </td>
        <td>
          <input type="text" name="Uage"><br>
        </td>
      </tr>
      <tr>
        <td>
          Address
        </td>
        <td>
          <input type="text" name="Uaddress"><br>
        </td>
      </tr>

      <tr>
        <td>
          ID number
        </td>
        <td>
          <input type="text" name="UID"><br>
        </td>
      </tr>
      <tr>
        <td>
          <input type="submit" name="submit" value="Submit" style="color: darkblue">
        </td>
        <td>
          <input type="reset" name="cancel" value="Cancel"style="color: red">
        </td>
      </tr>
    </form>
      </table>
  </body>
```

Java servlets

- Servlet is a java class file which can be used to do back-end data handling.
- Web application might contain number of JSP front end files, but all of them might not required a backend data processing.
- Create a servlet inside the web application if only it uses a back-end process.
- Which means all front-end pages should not require a servlet page.
- Since servlet is a java class file, it contains member variables and member methods.
- When creating a servlet there are three in built methods.
    1. ProcessRequest ( )
        → When servlet is not handling data, this method will invoke.
    2. doGet ( )
        → This method will invoke once we use GET as the HTTP request method. As the form method if we use get this will activate.
    3. doPost ( )
        → This method will activate once we use POST as the method request.

- To display content on the servlet, PrintWriter object reference should be created.

    PrintWriter out = response.getWriter();

- In above statement out is an object reference which can use any valid name.
- Using the object reference with the help of print inbuilt method, content can be display on servlet.

    out.print("Hello I am servlet");

    out.print("this is config servlet");

```
@Override
protected void doPost(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {

    PrintWriter out = response.getWriter();
    out.print("Hello i am servlet");

}
```

- Inside the servlet there should be appropriate variables to extract front-end formed data to back-end servlet.
- Using the controller name which defines in front-end each textbox values can be extracted.

  String Name = request.getParameter("Uname");
- In above code using request.getParameter in built method will extract data from textbox controller to the name variable.

```
protected void doPost(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {

        PrintWriter out = response.getWriter();
        String name = request.getParameter("Uname");
        out.println(name);
        String age = request.getParameter("Uage");
        out.println(age);
        String address = request.getParameter("Uaddress");
        out.println(address);
        String id = request.getParameter("UID");
        out.println(id);
    }
```

Q] create a JSP page and check whether user is eligible for voting or not.

```
PrintWriter out = response.getWriter();
      String name = request.getParameter("Uname");
      out.println(name);
      int age = Integer.parseInt(request.getParameter("Uage"));
      out.println(age);

      if(age >=18)
      {
         out.println("eligible for voting");
      }
      else
         out.println("not eligible for voting");
```

```
<body>
      <form action="test1" method="POST">
         Name:
         <input type="text" name="Uname"><br>
         Age:
         <input type="text" name="Uage"><br>
         <input type="submit" value="submit">
      </form>
   </body>
```

Q2] Explain what is meant by a session and a cookie.

Cookies → client-side files that are stored on a local computer and contain user information. Sessions → server-side files that store user information.

Q3] explain the difference between session and a cookie.

Cookie	Session
Cookies expire after the user specified lifetime.	The session ends when the user closes the browser or logs out of the program.
It can only store a limited amount of data.	It is able to store an unlimited amount of information.

Cookies are stored on a limited amount of data.	A session can store an unlimited amount of data.

Q] create a java class named BankAccount with following private fields. Create appropriate getters and setters to access and modify them. Create 2 objects using BankAccount class and input values for the fields using setters. Print the values to the console using getters.

```
public static void main(String[] args) {
        Scanner obj = new Scanner(System.in);
        System.out.println("Enter the account number: ");
        int acc = obj.nextInt();

        System.out.println("Enter the account holder: ");
        String acch = obj.next();

        System.out.println("Enter the account balance: ");
        Float accb = obj.nextFloat();

        BankAccount c = new BankAccount();
        c.setAcc(acc, acch, accb);

        System.out.println("Account number: "+c.getAcc());
        System.out.println("Account holder: "+ c.getHol());
        System.out.println("Balance is: "+ c.getba());
    }
```

```
class BankAccount {
    private int AccountNumber;
    private String AccountHolder;
    private Float Balance;

    public void setAcc(int accno, String acch, Float accbalance)
    {
        AccountNumber = accno;
        AccountHolder = acch;
        Balance = accbalance;
    }
    public int getAcc()
    {
        return AccountNumber;
    }
    public String getHol()
    {
        return AccountHolder;
    }
    public float getba()
    {
        return Balance;
    }
}
```

Q] create an abstract class called calculate and create an abstract method calculate with 2 int parameters. Create a class as Addition and display the relevant calculations.

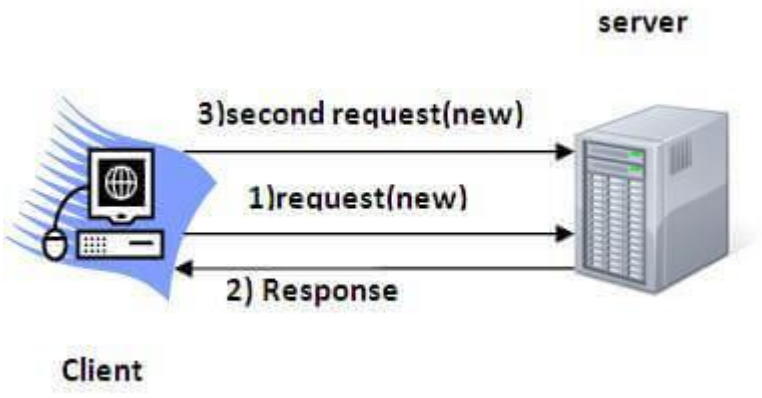
```
public class Addition extends Calculate{
   @Override
   public int calculate(int num1, int num2) {
      return num1+num2;
   }
```

```
abstract class Calculate {

    public abstract int calculate(int num1, int num2);

}
```

```
public static void main(String[] args) {
      Scanner obj = new Scanner(System.in);
      Addition x = new Addition();

      System.out.println("Additio is "+x.calculate(2, 4));
   }
```

Session tracking in servlet

- In web applications storing data which related to clients are important to run web application smoothly.
- The advantage of remembering this information is to run the web application smoothly with low response time.
- In HTTP protocol all requests which sent to the server will identify as new requests.
- Which means server should know the user details from request to request. This happens since HTTP is a stateless protocol. (Stateless - not maintaining the data of a user)
- Due to that reason using session tracking technologies we maintain user information throughout the web application.
- Below diagram explains how HTTP protocol transact (communicate) with server.

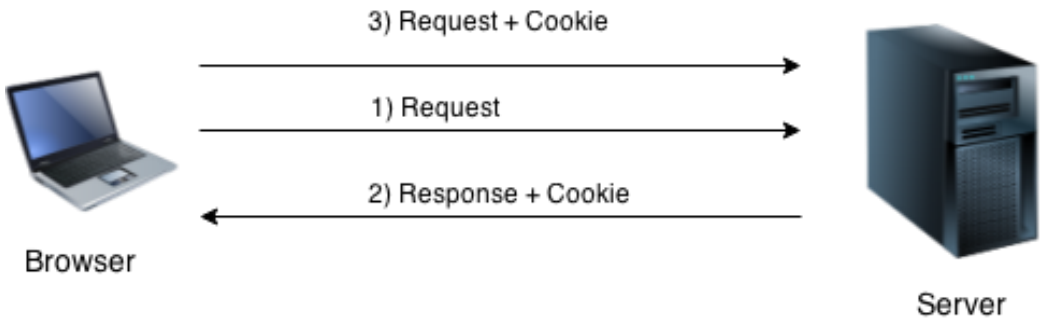


- There are 4 main technologies are being used for session tracking.
    1. Cookies
    2. Hidden form field
    3. URL rewriting
    4. HTTP sessions
- Among above technologies cookies and sessions are mostly used in web development.
- Reason for that is sessions are being used in server side and cookies are being used in client side.

- For verification purposes both of these technologies are being used together in web development.

Cookies in servlets

- When client sending the request to the server, server will generate a step of information which is known as a cookie.
- When server responding back to the client cookie will append (bind/ merge) to the response and send back to the client
- When client requesting again from the server created cookie also get attached. From that server will identify who is the owner of this request.
- The created cookie will store inside client end until it destroys by client itself.
- The main advantage of a cookie is it store in the client end with less memory consumption.
- The main disadvantage is if client disable the cookie, it won't exist anymore.



Java database connection (JDBC)

- Java web applications are capable of connecting with different types of databases such as MYSQL, Oracle etc.
- When connecting web application with the database a relevant connector should add to the web application accordingly.
- These connectors will create the medium to communicate with databases from the application. Note that connectors will change based on the database.
- These connectors are existing as .java files which we should include to the project library.
- When configuring the servlet we should import java.sql.*; to get inbuilt connection classes.
- Once we use the imported library, multiple classes will be available for the servlet file.
- Out of those classes Connection and Statement classes are being used to create instance and initialize with a null value at the beginning.

  Example: Connection con = null;

  Statement st = null;

```
@Override
protected void doPost(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {

    String Name = request.getParameter("uname");
    PrintWriter out = response.getWriter();

    Connection con = null; //created a connection instance. con is the object name.
    Statement st = null; //another instance of a statement class
}
```

- When connecting two different applications (front-end & back-end) there might be exceptions. Because of that exception handling should use in JDBC.
- Added jar file should invoke in back-end program using following code.

```
Class.forName("com.mysql.jdbc.Driver");
```

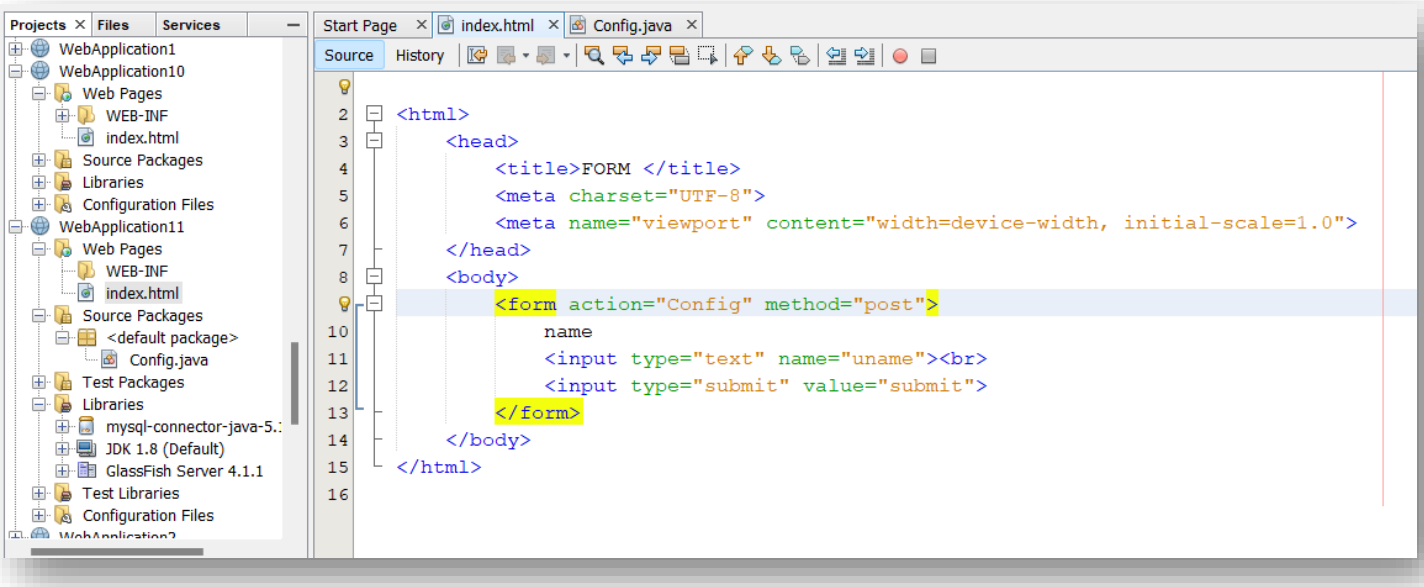
- By using DriverManager in built class your application should inform to which database that you are connecting.
- The URL of the database should pass as a parameter along with device MYSQL username and password.

Username = root password=null

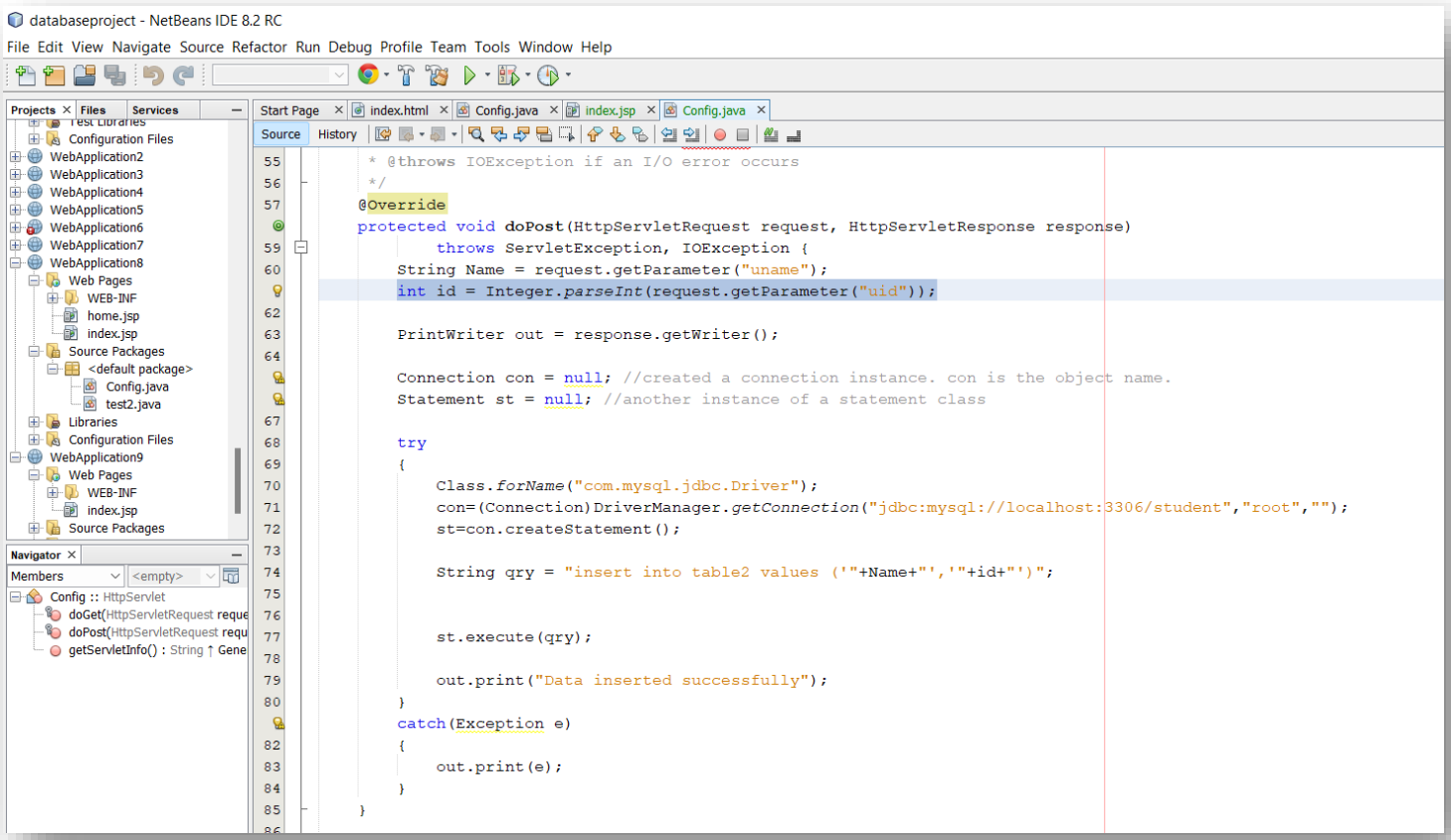
```
con=(Connection)DriverManager.getConnection("jdbc:mysql://localhost:3306/student","root","");
```

```
protected void doPost(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {

    String Name = request.getParameter("uname");
    PrintWriter out = response.getWriter();

    Connection con = null; //created a connection instance. con is the object name.
    Statement st = null; //another instance of a statement class

    try
    {
      Class.forName("com.mysql.jdbc.Driver");
      con=(Connection)DriverManager.getConnection("jdbc:mysql://localhost:3306/student","root","");
      st=con.createStatement();

      String qry = "insert into info values ('"+Name+"')";

      st.execute(qry);

      out.print("Data inserted successfully");
    }
    catch(Exception e)
    {
      out.print(e);
    }
  }
```

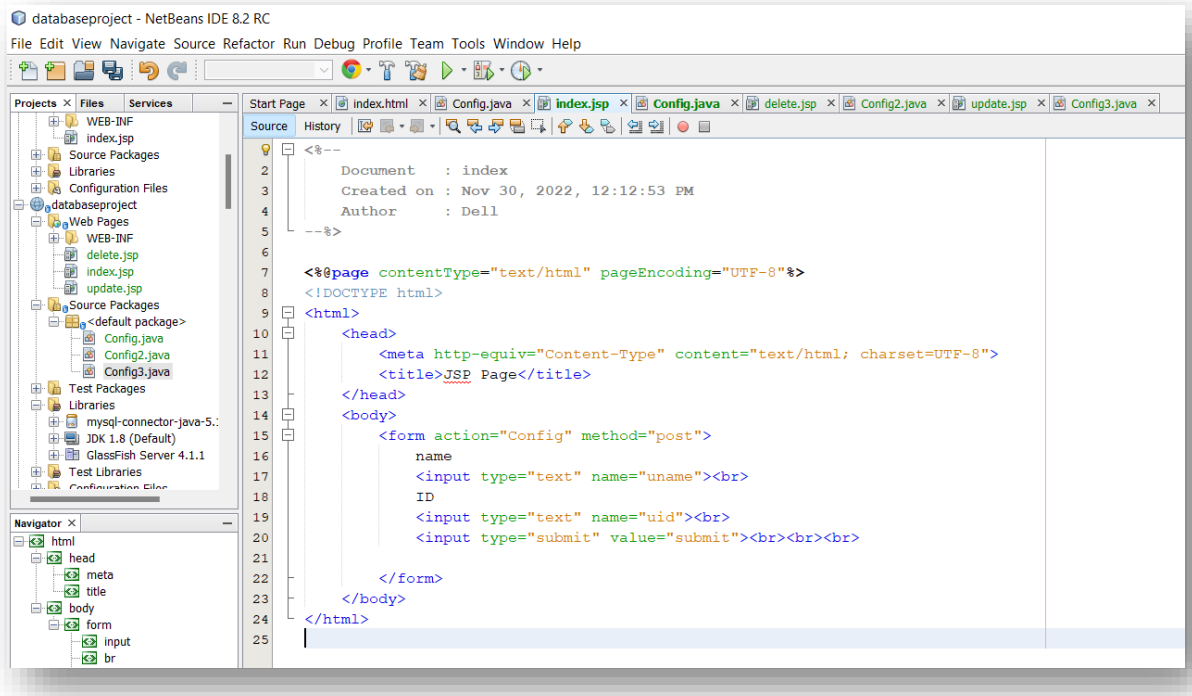
```java
@Override
protected void doPost(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {

    String Name = request.getParameter("uname");
    PrintWriter out = response.getWriter();

    Connection con = null; //created a connection instance. con is the object name.
    Statement st = null; //another instance of a statement class

    try
    {
        Class.forName("com.mysql.jdbc.Driver");
        con=(Connection)DriverManager.getConnection("jdbc:mysql://localhost:3306/student","root","");
        st=con.createStatement();

        String qry = "insert into info values ('"+Name+"')";

        st.execute(qry);

        out.print("Data inserted successfully");
    }
    catch(Exception e)
    {
        out.print(e);
    }
}

@Override
public String getServletInfo() {
    return "Short description";
}

}
```



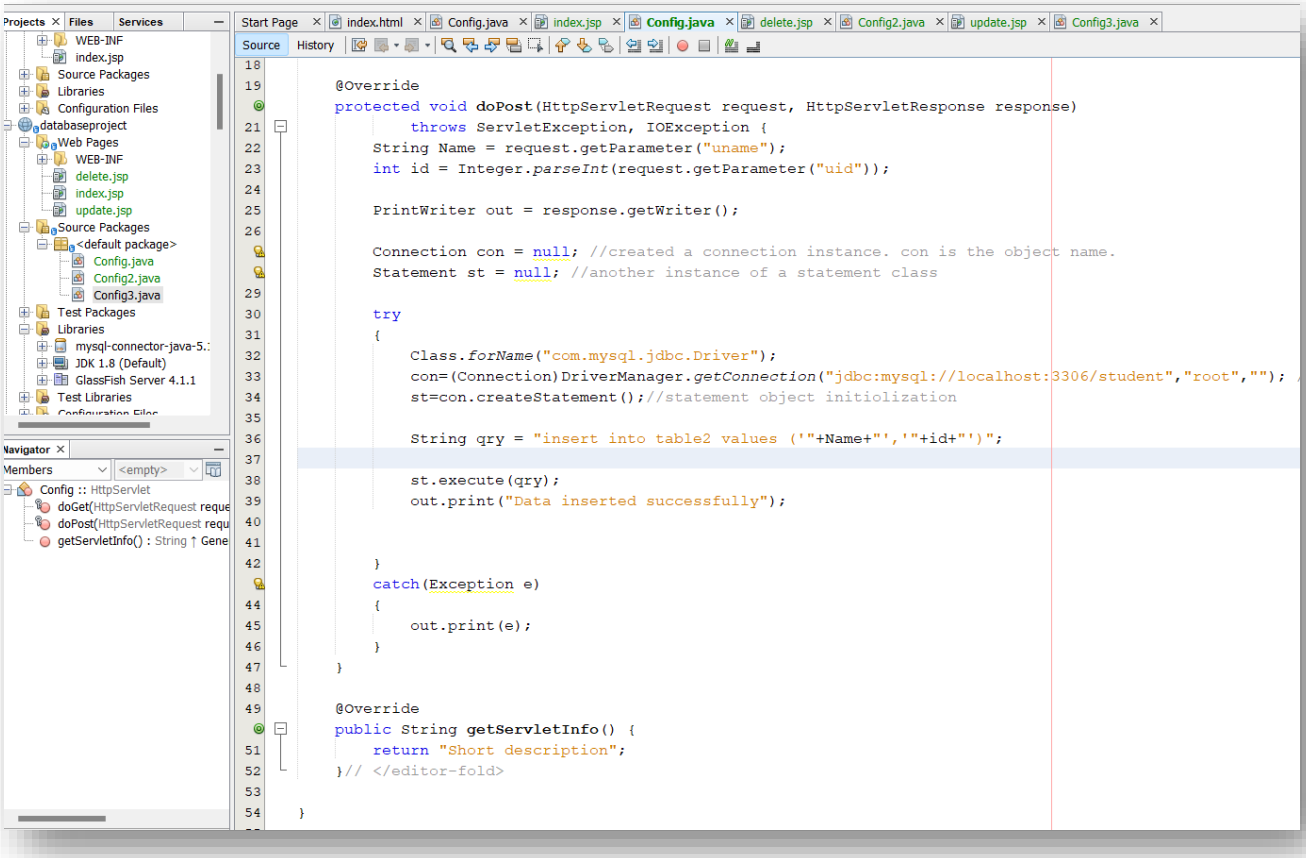
Q] get an integer value for the database.



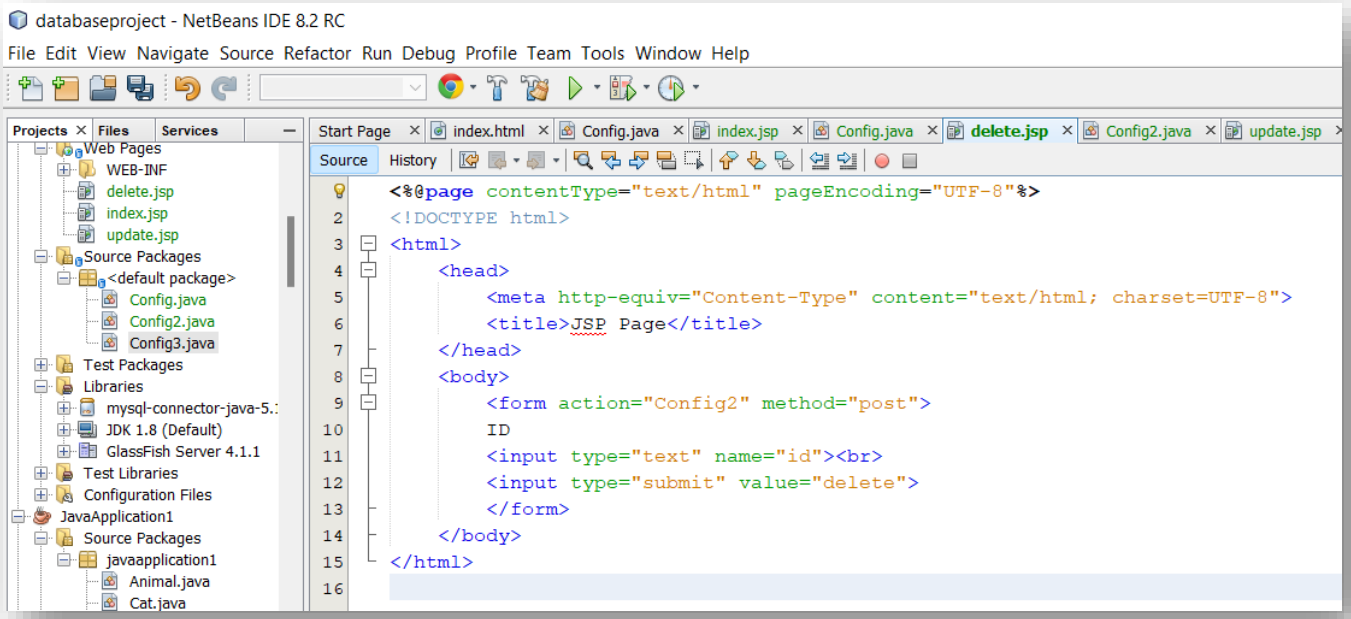
INSERT


databaseproject - NetBeans IDE 8.2 RC
File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help
Projects Files Services
Start Page index.html Config.java index.jsp Config.java delete.jsp Config2.java update.jsp Config3.java
Source History
<%--
Document : index
Created on : Nov 30, 2022, 12:12:53 PM
Author : Dell
--%>
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>JSP Page</title>
</head>
<body>
<form action="Config" method="post">
name
<input type="text" name="uname">
ID
<input type="text" name="uid">
<input type="submit" value="submit">

</form>
</body>
</html>
Navigator
html
head
meta
title
body
form
input
br

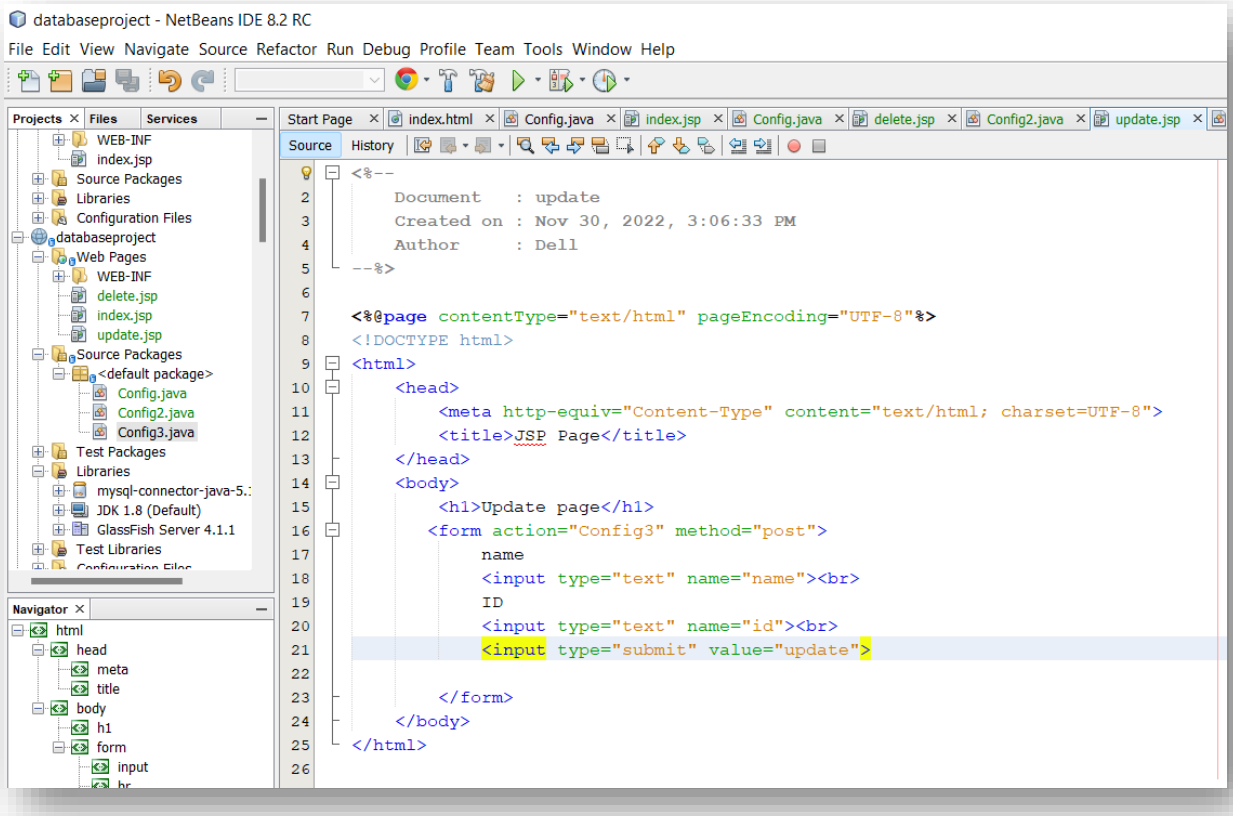

Start Page index.html Config.java index.jsp Config.java delete.jsp Config2.java update.jsp Config3.java
Source History
@Override
protected void doPost(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
String Name = request.getParameter("uname");
int id = Integer.parseInt(request.getParameter("uid"));
PrintWriter out = response.getWriter();
Connection con = null; //created a connection instance. con is the object name.
Statement st = null; //another instance of a statement class
try
{
Class.forName("com.mysql.jdbc.Driver");
con=(Connection)DriverManager.getConnection("jdbc:mysql://localhost:3306/student","root","");
st=con.createStatement();//statement object initiolization
String qry = "insert into table2 values ('"+Name+"','"+id+"')";
st.execute(qry);
out.print("Data inserted successfully");
}
catch(Exception e)
{
out.print(e);
}
}
@Override
public String getServletInfo() {
return "Short description";
}// </editor-fold>
}

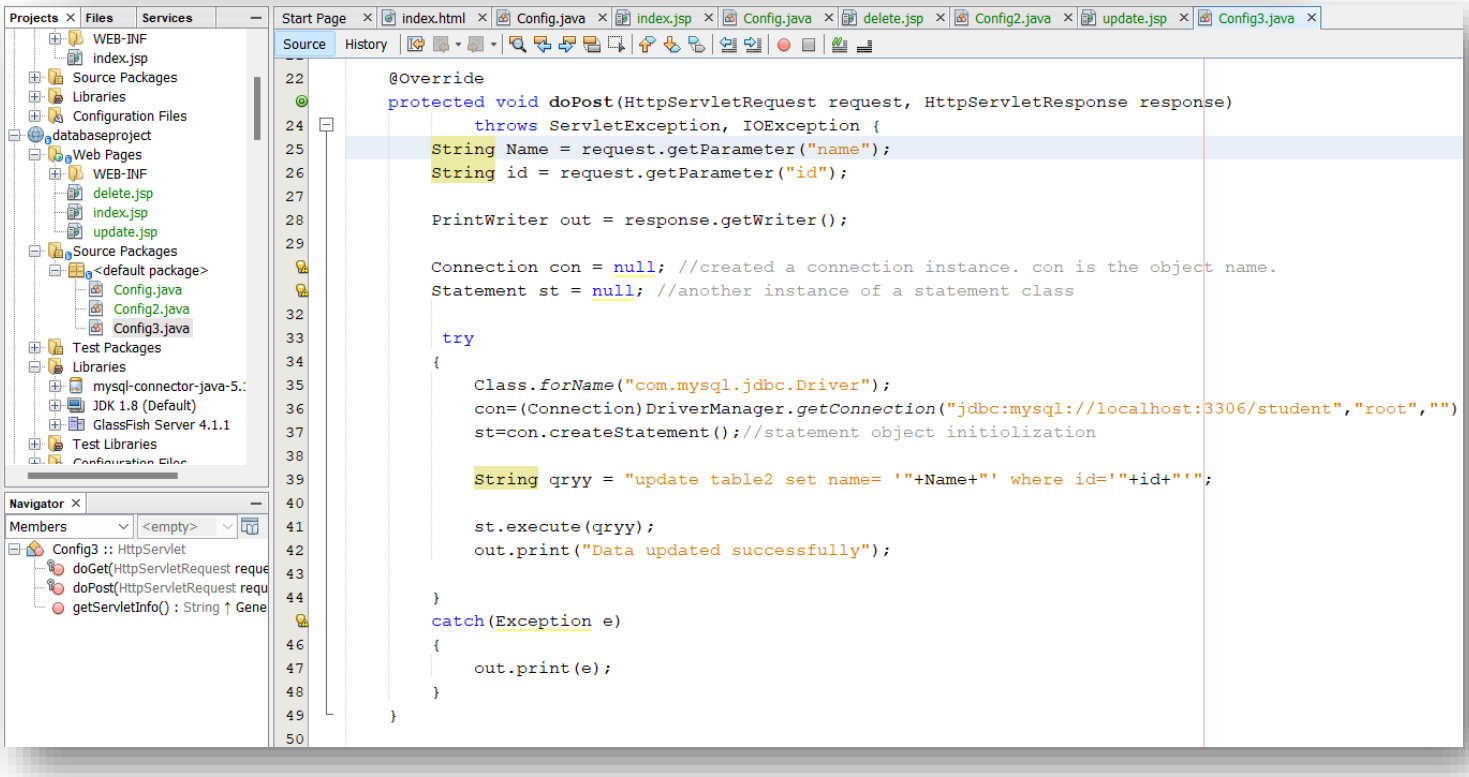
DELETE




```
@Override
protected void doPost(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {

    String id = request.getParameter("uid");

    PrintWriter out = response.getWriter();

    Connection con = null; //created a connection instance. con is the object name.
    Statement st = null; //another instance of a statement class

    try
    {
        Class.forName("com.mysql.jdbc.Driver");
        con=(Connection)DriverManager.getConnection("jdbc:mysql://localhost:3306/student","root","");
        st=con.createStatement();//statement object initiolization

        String qryy = "delete from table2 where id='"+id+"'";

        st.execute(qryy);
        out.print("Data deleted successfully");


    }
    catch(Exception e)
    {
        out.print(e);
    }
}
```

UPDATE


databaseproject - NetBeans IDE 8.2 RC
File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help
Projects Files Services
WEB-INF
index.jsp
Source Packages
Libraries
Configuration Files
databaseproject
Web Pages
WEB-INF
delete.jsp
index.jsp
update.jsp
Source Packages
<default package>
Config.java
Config2.java
Config3.java
Test Packages
Libraries
mysql-connector-java-5.
JDK 1.8 (Default)
GlassFish Server 4.1.1
Test Libraries
Navigator
html
head
meta
title
body
h1
form
input
Start Page
index.html
Config.java
index.jsp
Config.java
delete.jsp
Config2.java
update.jsp
Source
History
<%--
Document : update
Created on : Nov 30, 2022, 3:06:33 PM
Author : Dell
--%>
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>JSP Page</title>
</head>
<body>
<h1>Update page</h1>
<form action="Config3" method="post">
name
<input type="text" name="name">
ID
<input type="text" name="id">
<input type="submit" value="update">
</form>
</body>
</html>


Projects Files Services
WEB-INF
index.jsp
Source Packages
Libraries
Configuration Files
databaseproject
Web Pages
WEB-INF
delete.jsp
index.jsp
update.jsp
Source Packages
<default package>
Config.java
Config2.java
Config3.java
Test Packages
Libraries
mysql-connector-java-5.
JDK 1.8 (Default)
GlassFish Server 4.1.1
Test Libraries
Navigator
Members
<empty>
Config3 :: HttpServlet
doGet(HttpServletRequest reque
doPost(HttpServletRequest requ
getServletInfo() : String ↑ Gene
Start Page
index.html
Config.java
index.jsp
Config.java
delete.jsp
Config2.java
update.jsp
Config3.java
Source
History
@Override
protected void doPost(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
String Name = request.getParameter("name");
String id = request.getParameter("id");
PrintWriter out = response.getWriter();
Connection con = null; //created a connection instance. con is the object name.
Statement st = null; //another instance of a statement class
try
{
Class.forName("com.mysql.jdbc.Driver");
con=(Connection)DriverManager.getConnection("jdbc:mysql://localhost:3306/student","root","")
st=con.createStatement();//statement object initiolization
String qryy = "update table2 set name= '"+Name+"' where id='"+id+"'";
st.execute(qryy);
out.print("Data updated successfully");
}
catch(Exception e)
{
out.print(e);
}
}

Writing java codes in JSP page

Start Page | index.html | Config.java | index.jsp | Config.java | delete.jsp | Config2.java | update.jsp | Config3.java | display.jsp

Source | History

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<%@page import="java.sql.*"%>
<!DOCTYPE html>
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
        <title>JSP Page</title>
    </head>
    <body>
        <h1>Display Data</h1>
        <% //by using these tags we can write java codes
            Connection con = null;
            Statement st = null;
            ResultSet rs = null;

            try
            {
                Class.forName("com.mysql.jdbc.Driver");
                con=(Connection)DriverManager.getConnection("jdbc:mysql://localhost:3306/student","root","");
                st=con.createStatement();

                String qry = "select * from table2";
                rs = st.executeQuery(qry); //getting the values which retrieveing from the db and assign them to the rs

                while(rs.next())
                {
                    out.println(rs.getString(1)); //1 is the column number
                    out.println(rs.getString(2));
                }
            }
            catch(Exception e)
            {
                out.print(e);
            }
        %>
    </body>
</html>
```