

Practical Record File

B. Tech. (CSE)

Advanced Java Programming (**CSE4019)**

|  |  |  |  |
| --- | --- | --- | --- |
| **Course Title** | Advanced Java Programming | | |
| **Course Code** | CSE4019 | | |
| **Date of Start** | 24-March-2023 | **Submission Date** |  |
| **Verified By (Faculty)** | | **Approved By** | |
| **Dr. Anand Motwani**  Assistant Professor (G-II),  Computing Science & Engineering | | PC / Dean | |

INDEX

|  |  |  |
| --- | --- | --- |
| Experiment no. | Experiment Name | Page |
| 1 | Installation of Java and setting up PATH variables. Execute (Compiling and Running) simple Java programs. | 5 - 6 |
| 2 | To study basic structure of JAVA program, Class, Main method, console input/output (I/O) using simple program. | 7 - 8 |
| 3 | To demonstrate the concept of Classes and Objects in JAVA. To demonstrate the concepts of encapsulation and abstraction | 8 - 10 |
| 4 | To demonstrate the use of access modifiers | 11 - 13 |
| 5 | To demonstrate the concept of Simple Inheritance and Multilevel Inheritance in JAVA. Demonstrate various kinds of Inheritance | 13 - 15 |
| 6 | To demonstrate the concept of implementing Multiple Inheritance using Interfaces in JAVA. | 16 - 17 |
| 7 | To demonstrate the use and concept of arrays in JAVA. | 18 - 19 |
| 8 | Demonstrate Exception handling in Java. To create custom exception classes in Java | 20 - 21 |
| 9 | To demonstrate the concept of Concurrency and Multi-threading in JAVA | 22 - 24 |
| 10 | To demonstrate the use of Input /Output Serialization and Object Serialization | 24 - 27 |
| 11 | Explore various Bean development environment. | 27 - 30 |
| 12 | Demonstrate the implementation of simple Java Beans. | 31 - 32 |
| 13 | Demonstrate the working of Java beans using a GUI form | 32 - 35 |
| 14 | Create a custom component using java beans. | 35 - 37 |
| 15 | Demonstrate the use of Using the BeanInfo Class. | 37 - 40 |
| 16 | Demonstrate the Database connection establishment with Java application using java.sql API. | 40 - 42 |
| 17 | Demonstrate the use of Statement and ResultSet class for executing the SQL query and getting the results respectively. | 42 - 44 |
| 18 | Demonstrate the CRUD operations in Java with MySQL | 44 - 46 |
| 19 | Write a Java program to Insert data in the database table in a batch. | 46 - 49 |
| 20 | Develop a small application with GUI using SWING with a MySQL database. Connect the application with database using JDBC | 49 |
| 21 | Demonstrate the use of AWT package by creating a simple GUI form with label and buttons. | 52 - 54 |
| 22 | Demonstrate the use of Layouts (FlowLayout etc.) classes from AWT package by creating a simple GUI form. | 52 - 54 |
| 23 | Demonstrate the use of various components from swing package for implementing a simple form and ActionListener for the button. Form can be 1. Login form 2. Job application form 3. Pizza ordering form. 4. Voltage calculator. | 55 - 59 |
| 24 | ActionListener for the button and mouse click event. | 60 - 62 |
| 25 | Demonstrate the implementation of various swing controls (JTabbedPane, JPanel, JScrollBar, File Chooser etc.) on Form (JFrame) | 63 - 65 |
| 26 | Installation of WindowBuilder extension over Eclipse. | 66 - 67 |
| 27 | Depict the life cycle of JSP using example program. | 68 - 69 |
| 28 | Demonstrate the creation of JSP in Eclipse | 69 - 70 |
| 29 | Demonstrate the interaction between HTML, JSP, and the web application server | 71 - 73 |
| 30 | Implement / Demonstrate the concept of Servlet. | 73 - 75 |
| 31 | Create a web application using Servlet. | 75 - 77 |
| 32 | Perform connection-oriented Socket Programming in networking. | 78 - 81 |

|  |  |  |
| --- | --- | --- |
| Name | Registration Number | Experiments |
| Mayank Srivastava | 20BCE10446 | 11, 13, 14, 15, |
| Ayush Rai | 20BCE10938 | 2.8,16, 17, 18, 19,20 |
| Shauryan Bhardwaj | 20BCE10389. | 6,7,9,12,21,22,23,24,25,26 |
| Aayush Yadav | 20BCE10401 | 1,3,4,5,10,27,28,29,30,31,32 |

Date:28/03/2023

|  |  |
| --- | --- |
| **Exercise (Objective)** | 1. Installation of Java and setting up PATH variables. Execute (Compiling and Running) simple Java programs. |
| **Prerequisites** | The Prerequisites include:   1. A computer running a supported operating system (Windows, macOS, or Linux). 2. Internet access to download Java Development Kit (JDK). |
| **Underlying Concept / Theory** | Java is a popular programming language widely used for developing various applications. To start programming in Java, you need to install the Java Development Kit (JDK), which includes the Java compiler and runtime environment. Additionally, you need to set up the PATH variable to enable the command-line execution of Java programs. |
| **Problem Statement** | The problem is to install Java and set up the necessary environment variables to compile and run Java programs. |
| **Algorithm / Step-by-step instructions** | 1. Download JDK:    * Go to the Oracle website.    * Accept the license agreement.    * Download the JDK for your operating system. 2. Install JDK:    * Run the downloaded JDK installer.    * Follow the installation wizard instructions.    * Choose the installation location.    * Complete the installation process. 3. Set up PATH variable (Windows):    * Open "Environment Variables."    * Click "Edit the system environment variables."    * In the "System Properties" window, click "Environment Variables."    * In the "System Variables" section, select "Path" and click "Edit."    * Click "New" and add the JDK's "bin" directory path.    * Click "OK" to save the changes. 4. Set up PATH variable (macOS and Linux):    * Open a terminal.    * Run the command: **nano ~/.bash\_profile**.    * Add: **export PATH="/usr/local/bin:$PATH"** at the end.    * Press "Ctrl + X," "Y," and "Enter" to save. 5. Verify JDK installation:    * Open a terminal or command prompt.    * Run: **java -version**.    * Verify that Java version information is displayed. 6. Write a simple Java program:    * Open a text editor.    * Write the Java code: 7. Compile and run the Java program:    * Open a terminal or command prompt.    * Navigate to the directory with the Java file.    * Compile with: **javac HelloWorld.java**.    * If successful, a **HelloWorld.class** file is created.    * Run with: **java HelloWorld**.    * Verify the output: "Hello, World!". |
| **Output** | Hello World, should be displayed |
| **Conclusion / Remarks** | By following the step-by-step instructions provided, you should be able to successfully install Java, set up the necessary PATH variables, and compile/run simple Java programs. Java is a widely used programming language, and having it properly installed and configured is essential for Java development. |
| **Exercises /**  **Challenges** | PATH Variable Configuration: Setting up the PATH variable correctly is crucial for executing Java programs from the command line. Double-check that you added the correct path to the JDK's "bin" directory. Typos or incorrect paths can prevent the Java compiler and runtime from being recognized. |

Date:28/03/2023

|  |  |
| --- | --- |
| **Exercise (Objective)** | 2. To study basic structure of JAVA program, Class, Main method, console input/output (I/O) using simple program. |
| **Prerequisites** | Basic concept about class and input and output in java using Scanner and print statement. |
| **Underlying Concept / Theory** | 1. The main method must be declared as public, static, and void. It takes an array of strings (String[] args) as a parameter, which allows command-line arguments to be passed to the program.  2. Input/Output (I/O) is the process of interacting with the user or reading/writing data to external sources, such as the console, files, or network connections. |
| **Problem Statement** | To study basic structure of JAVA program, Class, Main method, console input/output (I/O) using simple program. |
| **Algorithm / Step-by-step instructions** | 1) Create the Main function and Import java.util.Scanner.  2) Open Scanner method.  2) Read the String from user.  3) Return the name of string as Hello - {name}. |
| **Code** | **import** java.util.Scanner;  **public** **class** Main {  **public** **static** **void** main(String[] args) {  // Create a Scanner object to read input from the console  Scanner sc = **new** Scanner(System.***in***);  // Prompt the user to enter their name  System.***out***.print("Enter your name: ");  // Read the user's input as a string  String name = scanner.nextLine();  // Print a greeting message with the user's name  System.***out***.println("Hello, " + name + " !");  // Close the scanner to free up resources  sc.close();  }  } |
| **Output** | Successfully print the name |
| **Conclusion / Remarks** |  |
| **Exercises /**  **Challenges** | Understanding the concepts of Main classes different type of package in java file |

Date:29/03/2023

|  |  |
| --- | --- |
| **Exercise (Objective)** | 3. To demonstrate the concept of Classes and Objects in JAVA. To demonstrate the concepts of encapsulation and abstraction |
| **Prerequisites** | Basic understanding of Java programming language, variables, and methods. |
| **Underlying Concept / Theory** | Classes and objects are fundamental concepts in object-oriented programming. A class is a blueprint or template that defines the structure and behavior of objects, whereas an object is an instance of a class. Encapsulation is the mechanism of wrapping data and methods into a single unit (class) and controlling access to that unit. Abstraction is the concept of providing simplified interfaces (methods) to interact with objects, hiding the internal details. |
| **Problem Statement** | The problem is to demonstrate the concept of classes and objects in Java, as well as the concepts of encapsulation and abstraction. Create classes with encapsulated data and demonstrate abstraction by providing simplified interfaces for accessing and manipulating that data. |
| **Algorithm / Step-by-step instructions** | 1. Create a class, e.g., **Student**, with private instance variables representing student attributes. 2. Define public setter and getter methods to access and modify the private variables. 3. Implement additional methods as needed, such as **displayInfo()** to display student information. 4. Create an object of the class in the main method. 5. Use setter methods to set values for the student attributes. 6. Use the **displayInfo()** method to print the student information. 7. Use getter methods to retrieve the values and print them. 8. Compile and run the program. |
| **Code** | class Student {  private String name;  private int age;  private String rollNumber;   public void setName(String name) {  this.name = name;  }   public String getName() {  return name;  }   public void setAge(int age) {  this.age = age;  }   public int getAge() {  return age;  }   public void setRollNumber(String rollNumber) {  this.rollNumber = rollNumber;  }   public String getRollNumber() {  return rollNumber;  }   public void displayInfo() {  System.*out*.println("Name: " + name);  System.*out*.println("Age: " + age);  System.*out*.println("Roll Number: " + rollNumber);  } }  // Main class public class EncapsulationDemo {  public static void main(String[] args) {  *// Create an object of the Student class* Student student = new Student();   *// Set values using setter methods* student.setName("John Doe");  student.setAge(20);  student.setRollNumber("ABC123");   *// Display student information using the displayInfo method* student.displayInfo();   *// Access information using getter methods* String name = student.getName();  int age = student.getAge();  String rollNumber = student.getRollNumber();   System.*out*.println("Name: " + name);  System.*out*.println("Age: " + age);  System.*out*.println("Roll Number: " + rollNumber);  } } |
| **Output** |  |
| **Conclusion / Remarks** | Classes and objects in Java provide a way to model real-world entities and define their properties and behaviors. Encapsulation allows for data hiding and controlling access to class members, ensuring data integrity and security. Abstraction simplifies interaction with objects by providing a clear and simplified interface. By encapsulating data and providing abstraction, code maintainability and reusability are enhanced. |
| **Exercises /**  **Challenges** | Understanding the concepts of classes and objects and their relationships. |

Date:30/03/2023

|  |  |
| --- | --- |
| **Exercise (Objective)** | 4. To demonstrate the use of access modifiers |
| **Prerequisites** | Basic understanding of Java programming language and familiarity with class, variables, and methods. |
| **Underlying Concept / Theory** | Access modifiers in Java are keywords that define the visibility and accessibility of variables, methods, and classes. There are four access modifiers in Java: **public**, **private**, **protected**, and the default (no explicit modifier).   * **public**: Public members are accessible from anywhere within the program. * **private**: Private members are only accessible within the same class. * **protected**: Protected members are accessible within the same class, subclass, and same package. * Default (no explicit modifier): Default members are accessible within the same package only. |
| **Problem Statement** | Implementing two interfaces in the Class to show that multiple inheritance is achievable by using Interfaces. |
| **Algorithm / Step-by-step instructions** | 1. Create a class named **AccessModifiersDemo**. 2. Declare variables with different access modifiers: **public**, **private**, **protected**, and default. 3. Define methods with different access modifiers: **public**, **private**, **protected**, and default. 4. In the **main** method, create an instance of the **AccessModifiersDemo** class. 5. Access and modify the variables using the appropriate access modifiers. 6. Call the methods using the appropriate access modifiers. 7. Compile and run the program. |
| **Code** | public class AccessModifiersDemo {  public int publicVariable;  private int privateVariable;  protected int protectedVariable;   public void publicMethod() {  System.*out*.println("This is a public method.");  }   private void privateMethod() {  System.*out*.println("This is a private method.");  }   protected void protectedMethod() {  System.*out*.println("This is a protected method.");  }   void defaultMethod() {  System.*out*.println("This is a default method.");  }   public static void main(String[] args) {  AccessModifiersDemo demo = new AccessModifiersDemo();  demo.publicVariable = 10;  demo.privateVariable = 20;  demo.protectedVariable = 30;  demo.publicMethod();  demo.privateMethod();  demo.protectedMethod();  demo.defaultMethod();  } } |
| **Output** |  |
| **Conclusion / Remarks** | Access modifiers in Java provide control over the visibility and accessibility of class members. They allow you to define the appropriate level of encapsulation and enforce proper data hiding. Public members can be accessed from anywhere, private members are restricted to the class itself, protected members can be accessed within the class, subclass, and same package, and default members are accessible within the same package. |
| **Exercises /**  **Challenges** | Ensuring that the access modifiers align with the desired level of encapsulation and information hiding. |

Date:31/03/2023

|  |  |
| --- | --- |
| **Exercise (Objective)** | 5. To demonstrate the use of access modifiers |
| **Prerequisites** | Basic understanding of Java programming language, classes, and objects. |
| **Underlying Concept / Theory** | Inheritance is a fundamental concept in object-oriented programming (OOP) that allows a class to inherit properties and behaviors from another class. It enables code reuse and promotes the concept of a "is-a" relationship.  There are various types of inheritance in Java, including:   1. Simple Inheritance: In simple inheritance, one class (child class) inherits properties and methods from another class (parent class). 2. Multilevel Inheritance: Multilevel inheritance involves a chain of inheritance, where a child class becomes the parent class for another child class, forming a hierarchical structure. |
| **Problem Statement** | The problem is to demonstrate the concept of simple inheritance and multilevel inheritance in Java. Create classes that inherit properties and methods from other classes to showcase different types of inheritance. |
| **Algorithm / Step-by-step instructions** | 1. Create a parent class with some properties and methods. 2. Create a child class that inherits from the parent class. 3. Add additional properties and methods specific to the child class, if needed. 4. Create a grandchild class that inherits from the child class. 5. Add additional properties and methods specific to the grandchild class, if needed. 6. In the main method or a separate test method, create an instance of the grandchild class. 7. Access and invoke inherited methods from the parent and child classes. 8. Invoke methods specific to the grandchild class. 9. Compile and run the program. |
| **Code** | *// Parent class* *class* Animal {  void eat() {  System.*out*.println("The animal eats.");  } }  // Child class inheriting from Animal class Dog extends Animal {  void bark() {  System.*out*.println("The dog barks.");  } }  // Grandchild class inheriting from Dog class Labrador extends Dog {  void run() {  System.*out*.println("The Labrador runs.");  } }  // Main class public class InheritanceDemo {  public static void main(String[] args) {  Labrador labrador = new Labrador();  labrador.eat(); *// Inherited from Animal* labrador.bark(); *// Inherited from Dog* labrador.run(); *// Defined in Labrador class* }} |
| **Output** |  |
| **Conclusion / Remarks** | Inheritance is a powerful mechanism in Java that allows classes to inherit properties and methods from other classes, promoting code reuse and establishing hierarchical relationships. Simple inheritance enables one class to inherit from another, while multilevel inheritance involves a chain of inheritance. By utilizing inheritance, you can create a class hierarchy and define specific behaviors at different levels of the hierarchy. |
| **Exercises /**  **Challenges** | Dealing with potential conflicts or ambiguity when multiple classes in the hierarchy define methods with the same name and signature. |

Date:3/04/2023

|  |  |
| --- | --- |
| **Exercise (Objective)** | 6. To demonstrate the concept of implementing Multiple Inheritance using Interfaces in JAVA. |
| **Prerequisites** | The Prerequisites include:  a) Understanding of Inheritance which is nothing but deriving properties from an already created class and using it to avoid redundancy.  b)Need of Interfaces which is aimed at achieving abstraction. |
| **Underlying Concept / Theory** | Multiple Inheritance is the way to be able to derive properties of more than 1 parent class in the child class. This is achieved by implementing more than 1 interface in the child class. The keyword used is ***“implements”.*** |
| **Problem Statement** | Implementing two interfaces in the Class to show that multiple inheritance is achievable by using Interfaces. |
| **Algorithm / Step-by-step instructions** | 1) Creation of two interfaces named Animal and FastAnimal using keyword interface  2)Declaring and Defining a default and a static method in the two interfaces respectively.  3) Implementing both interfaces in the Class named Tiger  4) Driver class to create object of Tiger and calling the method show1() defined in the class which calls the methods of interfaces using super keyword for the default method. |
| **Code** | **package** javaWinterInterimSem;  **interface** Animal// Understand static and default methods.  {  **public** **static** **final** String ***tname***="Hello";  **void** name(String tname);  **default** **void** show(){  System.***out***.println("I am default method in Animal interface");  }  }  **interface** FastAnimal  {  **void** run(**int** a);  **static** **void** roar() { // static method  System.***out***.println("I am static method in FastAnimal interface");  }  }  **class** Tiger **implements** Animal,FastAnimal // implementing 2 different interfaces  {  **int** a;  String y;  **public** **void** name(String name)  {  y=name;  }  **public** **void** run(**int** a)  {  **this**.a=a;  System.***out***.println("My speed is "+a+" Km/hour");  }  **void** show1()  {  Animal.**super**.show(); // using super keyword to call the show method    FastAnimal.*roar*();// calling the static method  }  }  **public** **class** interface1 {  **public** **static** **void** main(String args[]) {  Tiger ob=**new** Tiger();  ob.show1();    }  } |
| **Output** |  |
| **Conclusion / Remarks** | Multiple Inheritance is achievable by using interfaces in java. |
| **Exercises /**  **Challenges** | Here we have not declared any abstract method within the interfaces which is the main purpose of an interface, however from Java8 we can have default and static methods. This was one of the related challenges assigned in class. |

Date:03/04/2023

|  |  |
| --- | --- |
| **Exercise (Objective)** | 7. To demonstrate the use and concept of arrays in JAVA. |
| **Prerequisites** | The Prerequisites include:  a) Knowledge of different data types in java.  b) Brief knowledge of why we need data structures. |
| **Underlying Concept / Theory** | An array is a container object that holds a fixed number of values of a single type. The length of an array is established when the array is created. After creation, its length is fixed. Here, we learnt different ways to create an array and some basic operations on array. |
| **Problem Statement** | Different ways to create arrays and how to perform operations such as sorting on the data stored in the array. |
| **Algorithm / Step-by-step instructions** | 1)Declaring and initializing array with data values.  2)Running the for loop as per the length of the array.  3)Sorting the array using Bubble Sorting technique |
| **Code** | **public** **class** arr1 {  **public** **static** **void** main(String[] args) {  **int** arr[]= {6,1,5,3,2,4};  /\*  Method 2  int intArray[]; //declaring array  intArray = new int[20]; // allocating memory to array and then asking usinger to input values using Scanner class\*/  **for**(**int** x=0;x<arr.length;x++) {  System.***out***.println(arr[x]);  }  //Sort  **for**(**int** x=0;x<arr.length-1;x++) {  **for**(**int** y=0;y<arr.length-1-x;y++) {  **if**(arr[y]<arr[y+1]) {  **int** t=arr[y+1];  arr[y+1]=arr[y];  arr[y]=t;  }  }  }  System.***out***.println("The sorted array is : ");  **for**(**int** x=0;x<arr.length;x++) {  System.***out***.print(arr[x]+” ”);  }  }  } |
| **Output** |  |
| **Conclusion / Remarks** | The two methods for Array Creation understood and sorting the array elements successfully done. |
| **Exercises /**  **Challenges** | Different exercises that were undertaken included performing searching on the array elements like Linear and Binary searching. |

Date:04/04/2023

|  |  |
| --- | --- |
| **Exercise (Objective)** | 8. Demonstrate Exception handling in Java. To create custom exception classes in Java |
| **Prerequisites** | The Prerequisites include:  a) Understanding of what causes exceptions in the program which are due to some abnormal condition.  b)Why we need custom exceptions is because to provide exception handling that is not present or to modify an already existing Exception. |
| **Underlying Concept / Theory** | The try-catch block is used to handle exceptions in Java. It allows you to specify a block of code that might throw an exception and define how to handle it if it occurs. You should understand the syntax and usage of try-catch blocks |
| **Problem Statement** | Implementing two of Exception handling in java |
| **Algorithm / Step-by-step instructions** | 1)Write the code which might be throw the Expectation.  2) Handle the exception: we will wrap in the try catch statement, If your code can throw different types of exceptions, you can include multiple catch blocks.  3) The calling code can handle the exception by enclosing the method call or code block that may throw an exception within its own try-catch block. |
| **Code** | import java.io.IOException;  import java.lang.ArithmeticException;  import java.util.InputMismatchException;  import java.util.Scanner;  class Exception1 {  public static void main(String[] args) {  Scanner s = new Scanner(System.in);  int z = 0;  //System.out.println("Output = "+c);  try {  int x = s.nextInt();  int y = s.nextInt();  z = x/y; //these operation actually throws an exception object.  }  catch(ArithmeticException e) {  System.out.println("/ by zero is not allowed. Something went wrong");  }  catch (InputMismatchException m) {  System.out.println("Enter correct inputs please.");  }  finally {  System.out.println("This will be executed besides exception.");  System.out.println("Output = "+z);  }  System.out.print("Hello, This will run if exceptions are handled properly."); // run upto here  }  } |
| **Output** |  |
| **Conclusion / Remarks** | Here we catch all the possible of expectation in code above. |
| **Exercises /**  **Challenges** | How to Handel the all type of exception in code. |

Date:05/04/2023

|  |  |
| --- | --- |
| **Exercise (Objective)** | 9.To demonstrate the concept of Concurrency and Multi-threading in JAVA. |
| **Prerequisites** | The Prerequisites include:  a) Understanding of the classes that help achieve concurrency and Multi-threading in java  b) Understanding how Java concurrency is achieved using threads that are made available for different processes by the classes and interfaces. |
| **Underlying Concept / Theory** | Java Concurrency is the capability of the Java platform to run multiple operations simultaneously. In Java, a thread can be created by extending the thread class or implementing Runnable Interface. |
| **Problem Statement** | Doing a specific task by using a single thread and then two threads are used to finish the task parallely. |
| **Algorithm / Step-by-step instructions** | 1.Initialize the array. Store the starting time. Create a function that simply runs over the array elements. Call this in the main method. Record the final time.  2. Print the time taken by single thread.  3.Now, divide the array into two halves and create two threads each functioning on half of the array.  4.Record the time just like above.  5.Compare the time and the time taken during parallel is visibly less than doing it in a single go. |
| **Code** | **public** **class** MultiThread1 {  **public** **static** **void** main(String[] args) {  // create a task to be performed  **int**[] data = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12 };  **long** startTime = System.*currentTimeMillis*();  *performTask*(data);  **long** endTime = System.*currentTimeMillis*();  System.***out***.println("Single-threaded task completed in " + (endTime - startTime) + " milliseconds");    // create two threads to perform the same task in parallel  Thread thread1 = **new** Thread(() -> *performTask*(data, 0, data.length / 2));  Thread thread2 = **new** Thread(() -> *performTask*(data, data.length / 2, data.length));  startTime = System.*currentTimeMillis*();  thread1.start();  thread2.start();  **try** {  thread1.join();  thread2.join();  } **catch** (InterruptedException e) {  e.printStackTrace();  }  endTime = System.*currentTimeMillis*();  System.***out***.println("Multi-threaded task completed in " + (endTime - startTime) + " milliseconds");  }    **private** **static** **void** performTask(**int**[] data) {  **for** (**int** i = 0; i < data.length; i++) {  // simulate a time-consuming task  **try** {  Thread.*sleep*(100);  } **catch** (InterruptedException e) {  e.printStackTrace();  }  System.***out***.println("Thread " + Thread.*currentThread*().getName() + " processing element " + data[i]);  }  }    **private** **static** **void** performTask(**int**[] data, **int** start, **int** end) {  **for** (**int** i = start; i < end; i++) {  // simulate a time-consuming task  **try** {  Thread.*sleep*(100);  } **catch** (InterruptedException e) {  e.printStackTrace();  }  System.***out***.println("Thread " + Thread.*currentThread*().getName() + " processing element " + data[i]);  }  }  } |
| **Output** |  |
| **Conclusion / Remarks** | Multithreaded did the job parallel and hence took much less time than doing the task using single thread. |
| **Exercises /**  **Challenges** | Implementing Runnable Interface to do two tasks that are finding Odd and finding even numbers in parallel using Multi-threading. |

Date:06/04/2023

|  |  |
| --- | --- |
| **Exercise (Objective)** | 10. To demonstrate the use of Input /Output Serialization and Object Serialization |
| **Prerequisites** | Basic understanding of Java programming language, classes, and objects. |
| **Underlying Concept / Theory** | Inheritance is a fundamental concept in object-oriented programming (OOP) that allows a class to inherit properties and behaviors from another class. It enables code reuse and promotes the concept of a "is-a" relationship.  There are various types of inheritance in Java, including:   1. Simple Inheritance: In simple inheritance, one class (child class) inherits properties and methods from another class (parent class). 2. Multilevel Inheritance: Multilevel inheritance involves a chain of inheritance, where a child class becomes the parent class for another child class, forming a hierarchical structure. |
| **Problem Statement** | The problem is to demonstrate the use of input/output serialization and object serialization in Java. Implement a program that reads and writes objects to/from files using serialization. |
| **Algorithm / Step-by-step instructions** | 1. Create a class, e.g., Student, that implements the Serializable interface. 2. Define instance variables and relevant methods in the class. 3. Implement the serializeObject() method to serialize an object and write it to a file. 4. Implement the deserializeObject() method to deserialize an object from a file. 5. In the main method, create an object of the class. 6. Call the serializeObject() method to serialize the object and save it to a file. 7. Call the deserializeObject() method to deserialize the object from the file. 8. Display the deserialized object using the displayInfo() method. 9. Compile and run the program. |
| **Code** | import java.io.\*;  // Class representing a student class Student implements Serializable {  private String name;  private int age;  private String rollNumber;   public Student(String name, int age, String rollNumber) {  this.name = name;  this.age = age;   this.rollNumber = rollNumber;  }   public String getName() {  return name;  }   public int getAge() {  return age;  }   public String getRollNumber() {  return rollNumber;  } }  // Main class public class SerializationDemo {  public static void main(String[] args) {  *// Create a student object* Student student = new Student("John Doe", 20, "ABC123");   *// Serialize the object to a file*  *serializeObject*(student, "student.ser");   *// Deserialize the object from the file* Student deserializedStudent = (Student) *deserializeObject*("student.ser");   *// Display the deserialized object* System.*out*.println("Deserialized Student:");  System.*out*.println("Name: " + deserializedStudent.getName());  System.*out*.println("Age: " + deserializedStudent.getAge());  System.*out*.println("Roll Number: " + deserializedStudent.getRollNumber());  }   *// Serialize an object to a file* private static void serializeObject(Object object, String filename) {  try {  FileOutputStream fileOut = new FileOutputStream(filename);  ObjectOutputStream out = new ObjectOutputStream(fileOut);  out.writeObject(object);  out.close();  fileOut.close();  System.*out*.println("Object serialized successfully.");  } catch (IOException e) {  e.printStackTrace();  }  }  *// Deserialize an object from a file* private static Object deserializeObject(String filename) {  Object object = null;  try {  FileInputStream fileIn = new FileInputStream(filename);  ObjectInputStream in = new ObjectInputStream(fileIn);  object = in.readObject();  in.close();  fileIn.close();  System.*out*.println("Object deserialized successfully.");  } catch (IOException | ClassNotFoundException e) {  e.printStackTrace();  }  return object;  } } |
| **Output** |  |
| **Conclusion / Remarks** | Serialization is the process of converting an object into a byte stream, which can be saved to a file or transmitted over a network. Deserialization is the reverse process of reconstructing the object from the byte stream. Input/output serialization involves reading and writing objects to/from files, while object serialization focuses on the serialization and deserialization of objects. |
| **Exercises /**  **Challenges** | Serialization and deserialization operations can throw various exceptions, such as IOException and ClassNotFoundException. Proper error handling and exception management are essential to ensure the program functions correctly. |

Date:06/04/2023

|  |  |
| --- | --- |
| **Exercise (Objective)** | 11. Explore various Bean development environment. |
| **Prerequisites** | 1) Basic of Java Programming such all concept about array string list and so on.  2) Object-Oriented Programming (OOP) Principles . |
| **Underlying Concept / Theory** | Component-based Development: Bean development environments promote a component-based development approach. A component is a self-contained module that encapsulates functionality and can be easily reused and composed with other components to build applications. Beans serve as the building blocks of these components.  Properties: Properties can be accessed and modified through getter and setter methods, allowing for encapsulation and controlled access to the internal state of the Bean.  Events and Event Handling: Beans can generate events to indicate specific occurrences or changes within the component. Event handling mechanisms enable other components or listeners to respond to these events.  Customization and Configuration: Beans offer customization and configuration options, allowing developers or users to modify their behavior.  Design-Time and Runtime Environments: Bean development environments typically support both design-time and runtime environments.  Bean Persistence: Persistence refers to the ability of Beans to be serialized and stored to disk or transferred over a network.  Interoperability: Bean development environments focus on promoting interoperability between different components and frameworks. construction of complex, modular applications.  Framework Integration: Bean development environments often provide integration with existing frameworks and technologies. |
| **Problem Statement** | Explore various Bean development environment. |
| **Algorithm / Step-by-step instructions** | Sorting: If a Bean development environment includes functionality for working with collections or lists of objects, algorithms such as sorting algorithms (e.g., bubble sort, merge sort, quicksort) might be employed to sort the data based on specific criteria.  Searching: Algorithms for searching, such as linear search, binary search, or more advanced algorithms like hash-based search or tree-based search, may be used within a Bean development environment when working with data structures that require efficient searching operations.  Event Processing: When handling events within a Bean development environment, algorithms may be used to process and respond to events in an efficient and timely manner. This might involve event dispatching algorithms, event handling strategies, or algorithms for event correlation and filtering.  Data Transformation: In Bean development environments that deal with data transformation or manipulation, algorithms can be used for tasks like data validation, data formatting, or data conversion between different formats.  Graph Algorithms: If the Bean development environment involves working with graphs or networks, algorithms such as breadth-first search, depth-first search, or shortest path algorithms (e.g., Dijkstra's algorithm) might be utilized for graph traversal or path finding. |
| **Code** | **public** **class** Employee **implements** java.io.Serializable{  **private** **int** id;  **private** String name;  **public** Employee(){}  **public** **void** setId(**int** id){**this**.id=id;}  **public** **int** getId(){**return** id;}  **public** **void** setName(String name){**this**.name=name;}  **public** String getName(){**return** name;}  }  **package** mypack;  **public** **class** Test{  **public** **static** **void** main(String args[]){  Employee e=**new** Employee();//object is created  e.setName("Arjun");//setting value to the object  System.out.println(e.getName());  }} |
| **Output** |  |
| **Conclusion / Remarks** | Bean development environments promote a component-based approach, allowing developers to create modular and reusable software components known as Beans |
| **Exercises /**  **Challenges** | Implerting the java beans in our Project |

Date:07/04/2023

|  |  |
| --- | --- |
| **Exercise (Objective)** | 12. Demonstrate the implementation of simple Java Beans. |
| **Prerequisites** | The Prerequisites include:  a) Knowledge of Constructors, getters and setters and Serializability. |
| **Underlying Concept / Theory** | Java Bean **it is a reusable software component. A bean encapsulates many objects into one object so that we can access this object from multiple places. Moreover, it provides easy maintenance. It has a default constructor, getters and setters and should be Serializable.** |
| **Problem Statement** | Simple implementation of a Java Bean Class and using the component in other class. |
| **Algorithm / Step-by-step instructions** | 1. Create a Java Bean Class with the name Employee and implement Serializable interface.  2) Create default constructor and the getter and setter methods for private data variables.  3) Create Test class and use the Ja  va Bean Component by just creating an object and calling the setter and the getter methods. |
| **Code** | **public** **class** Employee **implements** java.io.Serializable{  **private** **int** id;  **private** String name;  **public** Employee(){}  **public** **void** setId(**int** id){**this**.id=id;}  **public** **int** getId(){**return** id;}  **public** **void** setName(String name){**this**.name=name;}  **public** String getName(){**return** name;}  }  **package** mypack;  **public** **class** Test{  **public** **static** **void** main(String args[]){  Employee e=**new** Employee();//object is created  e.setName("Arjun");//setting value to the object  System.out.println(e.getName());  }} |
| **Output** |  |
| **Conclusion / Remarks** | Java Bean Created and used in the Test class. |
| **Exercises /**  **Challenges** | Create a GUI based Java Bean and use it in other classes. |

Date:10/04/2023

|  |  |
| --- | --- |
| **Exercise (Objective)** | 13. Demonstrate the working of Java beans using a GUI form |
| **Prerequisites** | Basic knowledge about Java beans, AWT, and Swing |
| **Underlying Concept / Theory** | Java Bean it is a reusable software component. A bean encapsulates many objects into one object so that we can access this object from multiple places. Moreover, it provides easy maintenance. It has a default constructor, getters and setters and should be Serializable. |
| **Problem Statement** | Creating a GUI form and adding functionality using java beans |
| **Algorithm / Step-by-step instructions** | Follow the following steps  1. Create private data members  2. Create a no argument constructor  3. Create public getters, and setters for private data members  4. Create a GUI using AWT, and Swing  5. On clicking the button, the value of data members changes |
| **Code** | // imports  import java.awt.\*;  import java.awt.event.\*;  import javax.swing.\*;  // implementing java.io.Serializable interface  public class BeanGUI extends JPanel implements java.io.Serializable {  // private data members  private String name;  private int age;  private JButton button;  private JLabel nameLabel;  private JLabel ageLabel;  private JTextField nameField;  private JTextField ageField;    // creating a GUI component  public BeanGUI() {  setLayout(new GridLayout(3, 2));  nameLabel = new JLabel("Name:");  add(nameLabel);  nameField = new JTextField();  add(nameField);  ageLabel = new JLabel("Age:");  add(ageLabel);  ageField = new JTextField();  add(ageField);  button = new JButton("Submit");  add(button);  // on the click of a button value of name, and age gets changed, and then printing the value of name and age  button.addActionListener(new ActionListener() {  public void actionPerformed(ActionEvent e) {  name = nameField.getText();  age = Integer.parseInt(ageField.getText());  firePropertyChange("name", null, name);  firePropertyChange("age", null, age);  System.out.println(getName() + " "+getAge());  }  });  }  // public getters and setters  public String getName() {  return name;  }    public void setName(String name) {  String oldName = this.name;  this.name = name;  firePropertyChange("name", oldName, name);  }    public int getAge() {  return age;  }    public void setAge(int age) {  int oldAge = this.age;  this.age = age;  firePropertyChange("age", oldAge, age);  }    public static void main(String[] args) {  JFrame frame = new JFrame("MyBean");  frame.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);  BeanGUI bean = new BeanGUI();  frame.add(bean);  frame.pack();  frame.setVisible(true);    }  } |
| **Output** | Inserting image... |
| **Conclusion / Remarks** | Created a simple GUI form using AWT, and Swing, and making it reusable using Java Beans |
| **Exercises /**  **Challenges** | Connecting the form to a database and storing its value(response) on a database |

Date:10/04/2023

|  |  |
| --- | --- |
| **Exercise (Objective)** | 14. Create a custom component using java beans. |
| **Prerequisites** | Basic knowledge about Java beans, AWT, and Swing |
| **Underlying Concept / Theory** | Java Bean it is a reusable software component. A bean encapsulates many objects into one object so that we can access this object from multiple places. Moreover, it provides easy maintenance. It has a default constructor, getters and setters and should be Serializable. |
| **Problem Statement** | Creating a custom component using Java Beans |
| **Algorithm / Step-by-step instructions** | Follow the following steps  1. Create private data members  2. Create a no argument constructor  3. Create public getters, and setters for private data members  4. Create a GUI using AWT, and Swing  5. Insert the custom components in another class |
| **Code** | CustomComponents.java  // imports  import java.awt.\*;  import javax.swing.\*;    public class CustomComponents extends JButton {  // private data members  private String text;  // no argument constructor  public CustomComponents() {  super();  }  // public getters and setters  public String getText() {  return text;  }    public void setText(String text) {  this.text = text;  super.setText(text);  }  // returning a custom component  public Dimension getPreferredSize() {  return new Dimension(100, 30);  }  }  Custom.java  // imports    import javax.swing.JFrame;    public class Custom {  public static void main(String[] args) {  // creating an instance of CustomComponents  CustomComponents c = new CustomComponents();  // setting value of text  c.setText("Click me");  // creating a new frame  JFrame frame = new JFrame("MyBean");  frame.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);  // adding the custom component in the frame  frame.add(c);  frame.pack();  frame.setVisible(true);  }  } |
| **Output** |  |
| **Conclusion / Remarks** | Created a simple custom component using Java Beans, AWT, and Swing, and added it into a frame in another class |
| **Exercises /**  **Challenges** | Created an entire GUI using custom components |

Date:11/04/2023

|  |  |
| --- | --- |
| **Exercise (Objective)** | 15. Demonstrate the use of Using the BeanInfo Class. |
| **Prerequisites** | Basic knowledge about Java beans, and BeanInfo Class |
| **Underlying Concept / Theory** | The BeanInfo class is used to interface to create a BeanInfo class and provide explicit information about the methods, properties, events, and other features of your beans |
| **Problem Statement** | Creating a BeanInfo class to a Java Beans |
| **Algorithm / Step-by-step instructions** | Follow the following steps   1. Name your BeansInfo class 2. Subclass SimpleBeansInfo, this is a convenience class that implements BeansInfo methods to return null or an equivalent no-op value 3. Override the appropriate methods to return the properties, methods or even events that you want exposed. 4. Optionally associate an icon with the target Bean 5. Specify the target Bean class, and, if the Bean has a customizer, specify it also |
| **Code** | BeansInfo.java  // imports  import java.beans.\*;  import java.lang.reflect.Method;  public class BeansInfo extends SimpleBeanInfo{  // giving information about the properties in the beans  public PropertyDescriptor [] getPropertyDescriptors() {  try{  PropertyDescriptor temp = new PropertyDescriptor("temp",  Temperature.class);  PropertyDescriptor pd[] = {temp};  return pd;  }  catch(Exception e){  System.out.println(e);  }  return null;  }  // giving information about the methods in the java beans  public MethodDescriptor[] getMethodDescriptors() {  try{  Class cl = Temperature.class;  Class args[] = { };  Method cToF = cl.getMethod("cToF", args);  MethodDescriptor cToFDesc = new MethodDescriptor(cToF);  cToFDesc.setShortDescription("Convert Celsius to Fahrenheit");  Method fToC = cl.getMethod("fToC", args);  MethodDescriptor fToCDesc = new MethodDescriptor(fToC);  fToCDesc.setShortDescription("Convert Fahrenheit to Celsius");  MethodDescriptor [] md = {cToFDesc, fToCDesc};  return md;  }  catch(Exception e) {  System.out.println("Exception thrown");  }  return null;  }  }  Temperature.java    public class Temperature implements java.io.Serializable{  // private data member  private double temp;  public Temperature() {  temp = 0.0;  }  // public getters and setters  public double getTemp() {  return temp;  }  public void setTemp(double t) {  temp = t;  }  // methods  public void cToF() {  temp = temp \* (9.0/5.0)+32.0;  }  public void fToC() {  temp = (temp - 32.0) \* (5.0 / 9.0);  }  public static void main(String[] args) {  // Creating an instance of BeansInfo class  BeansInfo BeansInfo = new BeansInfo();  int n = BeansInfo.getPropertyDescriptors().length;  // Printing the description of properties of the beans  for(int i = 0; i < n; i++) {  System.out.println(BeansInfo.getPropertyDescriptors()[i]);  }  System.out.println();  int m = BeansInfo.getMethodDescriptors().length;  // Printing the description of methods of the beans  for(int i = 0; i < n; i++) {  System.out.println(BeansInfo.getMethodDescriptors()[i]);  }  System.out.println();  }  } |
| **Output** | java.beans.PropertyDescriptor[name=temp; propertyType=double; readMethod=public double classCodes.Temperature.getTemp(); writeMethod=public void classCodes.Temperature.setTemp(double)]    java.beans.MethodDescriptor[name=cToF; shortDescription=Convert Celsius to Fahrenheit; method=public void classCodes.Temperature.cToF()] |
| **Conclusion / Remarks** | Created a BeansInfo class for a Java beans |
| **Exercises /**  **Challenges** |  |

Date:12/04/2023

|  |  |
| --- | --- |
| **Exercise (Objective)** | 16. Demonstrate the Database connection establishment with Java application using java.sql API. |
| **Prerequisites** | Basic concept of my sql and concept about acess modiflier. |
| **Underlying Concept / Theory** | Importing the Libraries, importing the sql driver then the establish the connection executing the query of MySQL |
| **Problem Statement** | To Connect the java file with my sql with jdbc concept. |
| **Algorithm / Step-by-step instructions** | 1)Import necessary class and establish connection .  2) Establish a database connection and create the statement. Define Sql query.  3)Execute the query and obtain Result object.  4)Process the result. And close the connection. |
| **Code** | **import** java.sql.Connection;  **import** java.sql.DriverManager;  **import** java.sql.ResultSet;  **import** java.sql.Statement;  **public** **class** Java {  **static** Connection *conn* ;  **static** Statement *stmt*;  **static** ResultSet *rs* ;  **static** String *url*;  **static** String *name*;  **static** String *pass*;  **public** **static** **void** main(String[] args) {  **try** {  Class.*forName*("com.mysql.cj.jdbc.Driver"); // it connect database to my jdbc;  String url = "jdbc:mysql://localhost/complain";// it is database connection name which host locally  String name = "root"; // to establish the particular connection which particular database  String pass = "admin123"; // this is password of the  *conn* = DriverManager.*getConnection*(url, name, pass);// Passing url name and pass to connection (conn) String  System.***out***.println("Connection successfully establish");  } **catch** (Exception e) {  e.printStackTrace();  }  }  } |
| **Output** |  |
| **Conclusion / Remarks** | The Connection Establish successfully |
| **Exercises /**  **Challenges** | The issue in Practice is to connector with sql database. |

Date:13/04/2023

|  |  |
| --- | --- |
| **Exercise (Objective)** | 17. Demonstrate the use of Statement and ResultSet class for executing the SQL query and getting the results respectively. |
| **Prerequisites** | Basic concept of my sql and concept about acess modiflier. |
| **Underlying Concept / Theory** | The use of the Statement and ResultSet classes for executing SQL queries and retrieving results is to interact with a relational database in a Java application. |
| **Problem Statement** | To Connect the java file with my sql with jdbc concept and retriving the data from sql. With help of result set |
| **Algorithm / Step-by-step instructions** | 1)Import necessary class and establish connection .  2) Establish a database connection and create the statement. Define Sql query.  3)Execute the query and obtain Result object.  4)Process the result. And close the connection. |
| **Code** | **import** java.sql.Connection;  **import** java.sql.DriverManager;  **import** java.sql.ResultSet;  **import** java.sql.Statement;  **public** **class** Java {  **static** Connection *conn* ;  **static** Statement *stmt*;  **static** ResultSet *rs* ;  **static** String *url*;  **static** String *name*;  **static** String *pass*;  **static** **void** Connection() {  **try** {  *stmt* = *conn*.createStatement(); // create the statement for connection  *rs* = *stmt*.executeQuery("SELECT \* FROM complain"); // executing query of getting all complain from database  **while**(*rs*.next()) { // running the loop when rs.next() printing the valueW  System.***out***.print("Name: "+ *rs*.getString("studName") + " RoomNumber : "  + *rs*.getString("studroomNumber") + " Phone: "+*rs*.getString("studPhone") +  " \n student Complain: "+*rs*.getString("studComplain")  + " Status:- "+*rs*.getString("studStatus")+ "\n");  }  }**catch**(Exception exc){  System.***out***.println("exc");  }  }  **public** **static** **void** main(String[] args) {  **try** {  Class.*forName*("com.mysql.cj.jdbc.Driver"); // it connect database to my jdbc;  String url = "jdbc:mysql://localhost/complain";// it is database connection name which host locally  String name = "root"; // to establish the particular connection which particular database  String pass = "admin123"; // this is password of the SQL database  *conn* = DriverManager.*getConnection*(url, name, pass);// Passing url name and pass to connection (conn) String  System.***out***.println("Connection successfully establish");  *Connection*(); //Calling the static function Connection  } **catch** (Exception e) {  e.printStackTrace();  }  }  } |
| **Output** |  |
| **Conclusion / Remarks** | Hence! We have successfully retrieved the data from the complaint database, and it is shown on the terminal. |
| **Exercises /**  **Challenges** | When the database is connected with jdbc server it retrieves the data in Result Set. |

Date:13/04/2023

|  |  |
| --- | --- |
| **Exercise (Objective)** | 18. Demonstrate the CRUD operations in Java with MySQL |
| **Prerequisites** | Basic concept of my sql and concept about acess modiflier. And class and object |
| **Underlying Concept / Theory** | To Perform Different type of CRUD, perform such as Create Read and Update and delete operation. With help of Switch function in java |
| **Problem Statement** | To Perform all kind SQL query such as Create Update and Delete operation |
| **Algorithm / Step-by-step instructions** | 1)Import necessary class and establish connection.  2) Establish a database connection and create the statement. Define Sql query.  3) Creating CRUD function separately and calling them when the user wants to call.  3)Execute the query and obtain Result object.  4)Process the result. And close the connection. |
| **Code** | **import** java.sql.Connection;  **import** java.sql.DriverManager;  **import** java.sql.PreparedStatement;  **import** java.sql.ResultSet;  **import** java.sql.Statement;  **import** java.util.Scanner;    **public** **class** Java {  **static** Connection *conn* ;  **static** Statement *stmt*;  **static** ResultSet *rs* ;  **static** String *url*;  **static** String *name*;  **static** String *pass*;  **static** **void** Showdata() {  **try** {  *stmt* = *conn*.createStatement(); // create the statement for connection  *rs* = *stmt*.executeQuery("SELECT \* FROM complain"); // executing query of getting all complain from database  **while**(*rs*.next()) { // running the loop when rs.next() printing the value  System.***out***.print("Name: "+ *rs*.getString("studName") + " RoomNumber : "  + *rs*.getString("studroomNumber") + " Phone: "+*rs*.getString("studPhone") +  " \n student Complain: "+*rs*.getString("studComplain")  + " Status:- "+*rs*.getString("studStatus")+ "\n");  }  }**catch**(Exception exc){  System.***out***.println("exc");  }  }  **static** **void** Insertdata( String studName,String studroomNumber,String studPhone,String studComplain) {  **try** {  **if**((studPhone.length())>=10 || (studroomNumber.length())>=3 ) {// Checking Phone and room is valid or not  System.***out***.println("Complain raised");  }**else** {  System.***out***.println("Invaid number or invaild room number");  **return**;  }  // Excuting the Query of inserting Sql Query in database  PreparedStatement pstmt = *conn*.prepareStatement("INSERT INTO `complain` VALUES (?,?,?,?,?)");  pstmt.setString(1, studName);  pstmt.setString(2, studroomNumber);  pstmt.setString(3, studPhone);  pstmt.setString(4, studComplain);  pstmt.setString(5,"not done");  System.***out***.println(studName);  pstmt.executeUpdate();  System.***out***.print("Insert Successfully");  }**catch**(Exception ex){  System.***out***.println(ex);  }  }  **static** **void** Updatedata(String room) {  **try** {  *stmt* = *conn*.createStatement();  **int** studroomNumber=Integer.*parseInt*(room);  // the Sql Query for Updating the data.  String sql = "UPDATE complain SET studStatus = 'completed' WHERE studroomNumber ="+studroomNumber;  *stmt*.executeUpdate(sql); // Executing the update command.  // Retrieving the data from complain table and from complain database.  System.***out***.print("Update Successfully");  }**catch**(Exception exc){  System.***out***.println(exc);  }  }  **static** **void** Deletedata(String room) {  **try** {  *stmt* = *conn*.createStatement();  **int** studroomNumber=Integer.*parseInt*(room);  // the Sql Query for Deleting the data.  String sql = "Delete from complain WHERE studroomNumber ="+studroomNumber;  *stmt*.executeUpdate(sql); // Executing the update command.  // Retrieving the data from complain table and from complain database.  *rs* = *stmt*.executeQuery("SELECT \* FROM complain");  System.***out***.print("Delete Successfully");  }**catch**(Exception exc){  System.***out***.println(exc);  }  }  **public** **static** **void** main(String[] args) {  **try** {  Class.*forName*("com.mysql.cj.jdbc.Driver"); // it connect database to my jdbc;  String url = "jdbc:mysql://localhost/complain";// it is database connection name which host locally  String name = "root"; // to establish the particular connection which particular database  String pass = "admin123"; // this is password of the  *conn* = DriverManager.*getConnection*(url, name, pass);// Passing url name and pass to connection (conn) String  Scanner sc= **new** Scanner(System.***in***); // creating the choice  System.***out***.print("1 For Creating the value.");  System.***out***.print("2 For Reading the value.");  System.***out***.print("3 For Updating the values");  System.***out***.print("4 For Delecting the value.");  **int** Choise = sc.nextInt();//Accepting the choice.  // Switch statement with int data type  **switch** (Choise) {  // Case1 for Inserting  **case** 1:  System.***out***.println("Write Studname follow by, Student Room number, Phone no, Complain ");  String Name=sc.next();  String Room=sc.next();  String Phone= sc.next();  String Com=sc.next();  Insertdata(Name,Room,Phone,Com);  **break**;  // Case2 for Reading  **case** 2:  Showdata(); //Calling the static function Connection  **break**;  // Case3 for Updating  **case** 3:  System.***out***.print("Write the Room Number:- ");  String Roomno=sc.next();  Updatedata(Roomno);  **break**;  // Case4 for Deletion  **case** 4:  System.***out***.print("Write the Room Number:- ");  String roomno=sc.next();  Deletedata(roomno);  **break**;  // Default case  **default**:  System.***out***.println("Input Invalid State");  }  System.***out***.println("Connection successfully establish");  sc.close();    } **catch** (Exception e) {  e.printStackTrace();  }  }  } |
| **Output** |  |
| **Conclusion / Remarks** | Hence! We have successfully retrieved the data, Update and Delete, Insert from the complaint database, and it is shown on the terminal. |
| **Exercises /**  **Challenges** | When the database is connected with jdbc server it retrieves the data in Result Set. |

Date:13/04/2023

|  |  |
| --- | --- |
| **Exercise (Objective)** | 19. Write a Java program to Insert data in the database table in a batch. |
| **Prerequisites** | Basic concept of my sql and concept about acess modiflier. And class and object |
| **Underlying Concept / Theory** | Instead of executing individual INSERT statements for each record, batching allows you to group multiple records together and send them as a single unit to the database server. |
| **Problem Statement** | To Add Bulk of data to SQL in one Click. |
| **Algorithm / Step-by-step instructions** | 1) We create the Connection and Statement and prepare the SQL Statement.  2) Add the Add Batched Records for inserting in Bulk then executing the Execute the Batch statement to see the output. and then Commit and Close Resources: |
| **Code** | **import** java.sql.Connection;  **import** java.sql.DriverManager;  **import** java.sql.PreparedStatement;  **import** java.sql.ResultSet;  **import** java.sql.\*;  **import** java.sql.Statement;  **import** java.util.Scanner;    **public** **class** Project {  **static** Connection *conn* ;  **static** Statement *stmt*;  **static** ResultSet *rs* ;  **static** String *url*;  **static** String *name*;  **static** String *pass*;  **static** **void** Batch() {  **try** {  *stmt* = *conn*.createStatement(); // create the statement for connection  String insertQuery = "INSERT INTO persons (PersonID,LastName, FirstName) VALUES (?,?, ?)";  PreparedStatement pmt = *conn*.prepareStatement(insertQuery);    // Enable batch processing  *conn*.setAutoCommit(**false**);    // Add multiple data rows to the batch  pmt.setInt(1, 123);  pmt.setString(2, "Yadav");  pmt.setString(3, "Ayush");  pmt.addBatch();    pmt.setInt(1, 124);  pmt.setString(2, "Singh");  pmt.setString(3, "ayush");  pmt.addBatch();    pmt.setInt(1, 125);  pmt.setString(2, "rai");  pmt.setString(3, "ayush");  pmt.addBatch();  // Execute the batch  pmt.executeBatch();    // Commit the transaction  *conn*.commit();    System.***out***.println("Data inserted successfully in batch.");  } **catch** (SQLException e) {  System.***err***.println("Error: " + e.getMessage());  }  }        **public** **static** **void** main(String[] args) {  **try** {  Class.*forName*("com.mysql.cj.jdbc.Driver"); // it connect database to my jdbc;  String url = "jdbc:mysql://localhost/ayush";// it is database connection name which host locally  String name = "root"; // to establish the particular connection which particular database  String pass = "admin123"; // this is password of the SQL Database  *conn* = DriverManager.*getConnection*(url, name, pass);// Passing url name and pass to connection (conn) String  *Batch*();  } **catch** (Exception e) {  e.printStackTrace();  }  }  } |
| **Output** |  |
| **Conclusion / Remarks** | Hence the Successfully we added the bulk data in MySQL |
| **Exercises /**  **Challenges** | Use the executeBatch() method to execute the batched INSERT statements. This sends the entire batch to the database server in one communication round-trip |

Date:14/04/2023

|  |  |
| --- | --- |
| **Exercise (Objective)** | 20. Develop a small application with GUI using SWING with a MySQL database. Connect the application with database using JDBC |
| **Prerequisites** | Basic concept of my sql and concept about acess modiflier. And class and object |
| **Underlying Concept / Theory** | All kind of frame you should know like Radio Button checkbox and many more, You should have prior knowledge about use of swing jdbc |
| **Problem Statement** | Develop the mini-Project which includes jdbc and swing frame work. |
| **Algorithm / Step-by-step instructions** | 1) Develop the Layout of Project (various kind of text flied, text Area, JButton ) etc,  2) Create the connection and Statement to MySQL Server.  3)Crate various kinds of Button on which we add Listener Property on it.  4)Execute the different command on click of button. |
| **Code** |  |
| **Output** |  |
| **Conclusion / Remarks** | To Execute the All variable in single java file. |
| **Exercises /**  **Challenges** | We use ALL Concept of jdbl and Swing in one project that might be challenging task. |

Date:17/04/2023

|  |  |
| --- | --- |
| **Exercise (Objective)** | 21. Demonstrate the use of AWT package by creating a simple GUI form with label and buttons. |
| **Prerequisites** | The Prerequisites are having knowledge about the different widgets that are available under the AWT package. |
| **Underlying Concept / Theory** | **Java AWT** (Abstract Window Toolkit) is an API to develop Graphical User Interface (GUI) or windows-based applications in Java. Java AWT components are platform-dependent i.e. components are displayed according to the view of operating system. AWT is heavy weight i.e. its components are using the resources of underlying operating system (OS). |
| **Problem Statement** | A simple GUI form that contains a label and a button. |
| **Algorithm / Step-by-step instructions** | 1.Import all classes from awt package.  2.Inside the constructor create a frame, a button, a label and add the widgets to the frame using add method of the Frame class.  3.Set the size of the frame, layout and visibility.  4. Create object of the class inside the main method. |
| **Code** | **package** GUI1;  **import** java.awt.\*;  **public** **class** Testawt {  Testawt(){  Frame fm=**new** Frame();  Button btn=**new** Button("Hello World");  Label lb=**new** Label("Welcome to java Graphics");  fm.add(btn);  fm.add(lb);  fm.setSize(300,300);  fm.setLayout(**new** FlowLayout());  fm.setVisible(**true**);  fm.setBackground(Color.***red***);  }  **public** **static** **void** main(String[] args) {  // **TODO** Auto-generated method stub  Testawt ob=**new** Testawt();  }  } |
| **Output** |  |
| **Conclusion / Remarks** | The form successfully created that contains a label and a button using the classes from the AWT package. |
| **Exercises /**  **Challenges** | The exercise assigned was to use more components to create a form. |

Date:17/04/2023

|  |  |
| --- | --- |
| **Exercise (Objective)** | 22. Demonstrate the use of Layouts (FlowLayout etc.) classes from AWT package by creating a simple GUI form. |
| **Prerequisites** |  |
| **Underlying Concept / Theory** | The Java FlowLayout class is used to arrange the components in a line, one after another (in a flow). It is the default layout of the applet or panel. |
| **Problem Statement** | Implementation of Flow layout by applying it to some components. |
| **Algorithm / Step-by-step instructions** | 1.Import the awt and swing classes.  2.Create object of the JFrame class. Create 10 objects of JButton.  3.Add the JButton objects to the Jframe object .  4.Set the Layout of the Frame object and also provide the horizontal and vertical gaps for better view.  5.In the Driver class create the object of the Driver Class. |
| **Code** | // import statement  **import** java.awt.\*;  **import** javax.swing.\*;    **public** **class** AwtFlowLayout  {  JFrame frameObj;    // constructor  AwtFlowLayout()  {  // creating a frame object  frameObj = **new** JFrame();    // creating the buttons  JButton b1 = **new** JButton("1");  JButton b2 = **new** JButton("2");  JButton b3 = **new** JButton("3");  JButton b4 = **new** JButton("4");  JButton b5 = **new** JButton("5");  JButton b6 = **new** JButton("6");  JButton b7 = **new** JButton("7");  JButton b8 = **new** JButton("8");  JButton b9 = **new** JButton("9");  JButton b10 = **new** JButton("10");      // adding the buttons to frame  frameObj.add(b1); frameObj.add(b2); frameObj.add(b3); frameObj.add(b4);  frameObj.add(b5); frameObj.add(b6); frameObj.add(b7); frameObj.add(b8);  frameObj.add(b9); frameObj.add(b10);    // parameterized constructor is used  // where alignment is left  // horizontal gap is 20 units and vertical gap is 25 units.  frameObj.setLayout(**new** FlowLayout(FlowLayout.***LEFT***, 20, 25));      frameObj.setSize(300, 300);  frameObj.setVisible(**true**);  }  // main method  **public** **static** **void** main(String argvs[])  {  **new** AwtFlowLayout();  }  } |
| **Output** |  |
| **Conclusion / Remarks** | Frame Layout successfully set and results satisfied the problem statement thoroughly |
| **Exercises /**  **Challenges** | The challenges included trying other layouts such as Border, Grid, Box Layouts. |

Date:18/04/2023

|  |  |
| --- | --- |
| **Exercise (Objective)** | 23. Demonstrate the use of various components from swing package for implementing a simple form and ActionListener for the button. Form can be 1. Login form 2. Job application form 3. Pizza ordering form. 4. Voltage calculator. |
| **Prerequisites** | The Prerequisites include:  1.Knowledge of Swing and Awt classes and the different components they provide with. Such as JPanel, JcheckBox, JRadioButton etc. |
| **Underlying Concept / Theory** | The Pizza ordering system shall contain 3 different frames for deciding size, toppings and way the food has to be delivered. Once the Order is placed, the details shall get stored in the database. The database connection is made using JDBC concepts. |
| **Problem Statement** | Creating a Simple form for Pizza Ordering system. |
| **Algorithm / Step-by-step instructions** | 1. Extend the Jframe class and implement the ActionListener.  2.Set the layout using Border layout . Create 3 separate panels and add the checkboxes, radio buttons respectively.  3.Make Connection with the database.  4.Inside the ActionPerformed method get the source where the event has occurred and take relevant actions, get the values and insert them inside the table created under the database. |
| **Code** | package swing\_project\_bookshop1;  import javax.swing.\*;  import java.awt.\*;  import java.awt.event.\*;  import java.sql.\*;  public class PizzaSystem extends JFrame implements ActionListener {    private JRadioButton smallSize, mediumSize, largeSize;    private JCheckBox pepperoni, mushrooms, onions, sausage, bacon;    private JRadioButton delivery, pickup;    private JButton orderButton;    private Connection conn;    public PizzaSystem() {  setTitle("Pizza Order Page");  setSize(400, 300);  setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);  setLayout(new BorderLayout());    JPanel sizePanel = new JPanel();  sizePanel.setBorder(BorderFactory.createTitledBorder("Pizza Size"));    smallSize = new JRadioButton("Small");  mediumSize = new JRadioButton("Medium");  largeSize = new JRadioButton("Large");  ButtonGroup sizeGroup = new ButtonGroup();  sizeGroup.add(smallSize);  sizeGroup.add(mediumSize);  sizeGroup.add(largeSize);    sizePanel.add(smallSize);  sizePanel.add(mediumSize);  sizePanel.add(largeSize);    JPanel toppingsPanel = new JPanel();  toppingsPanel.setBorder(BorderFactory.createTitledBorder("Toppings"));    pepperoni = new JCheckBox("Pepperoni");  mushrooms = new JCheckBox("Mushrooms");  onions = new JCheckBox("Onions");  sausage = new JCheckBox("Sausage");  bacon = new JCheckBox("Bacon");    toppingsPanel.add(pepperoni);  toppingsPanel.add(mushrooms);  toppingsPanel.add(onions);  toppingsPanel.add(sausage);  toppingsPanel.add(bacon);    JPanel deliveryPanel = new JPanel();  deliveryPanel.setBorder(BorderFactory.createTitledBorder("Delivery or Pickup"));    delivery = new JRadioButton("Delivery");  pickup = new JRadioButton("Pickup");  ButtonGroup deliveryGroup = new ButtonGroup();  deliveryGroup.add(delivery);  deliveryGroup.add(pickup);    deliveryPanel.add(delivery);  deliveryPanel.add(pickup);    orderButton = new JButton("Order");  orderButton.addActionListener(this);    add(sizePanel, BorderLayout.NORTH);  add(toppingsPanel, BorderLayout.CENTER);  add(deliveryPanel, BorderLayout.SOUTH);  add(orderButton, BorderLayout.EAST);    setVisible(true);    try {  Class.forName("com.mysql.jdbc.Driver");  conn = DriverManager.getConnection("jdbc:mysql://localhost:3306/pizza","root","");  } catch (Exception e) {  e.printStackTrace();  }  }    public void actionPerformed(ActionEvent e) {  if (e.getSource() == orderButton) {  String size = "";  if (smallSize.isSelected()) {  size = "Small";  } else if (mediumSize.isSelected()) {  size = "Medium";  } else if (largeSize.isSelected()) {  size = "Large";  }    String toppings = "";    if (pepperoni.isSelected()) {  toppings += "Pepperoni ";  }  if (mushrooms.isSelected()) {  toppings += "Mushrooms ";  }  if (onions.isSelected()) {  toppings += "Onions ";  }  if (sausage.isSelected()) {  toppings += "Sausage ";  }  if (bacon.isSelected()) {  toppings += "Bacon ";  }    String deliveryOption = "";  if (delivery.isSelected()) {  deliveryOption = "Delivery";  } else if (pickup.isSelected()) {  deliveryOption = "Pickup";  }    try {  Statement stmt = conn.createStatement();  String sql = "INSERT INTO orderspizza\_table (size, toppings, delivery) VALUES ('" + size + "', '" + toppings + "', '" + deliveryOption + "')";  stmt.executeUpdate(sql);  JOptionPane.showMessageDialog(this, "Order placed successfully!");  } catch (SQLException ex) {  ex.printStackTrace();  JOptionPane.showMessageDialog(this, "Error placing order.");  }  }  }    public static void main(String[] args) {  new PizzaSystem();  }  } |
| **Output** |  |
| **Conclusion / Remarks** | The Pizza delivery successfully created with some functionality added. |
| **Exercises /**  **Challenges** | Created similar forms such as enrollment form, Voltage Calculator. |

Date:01/05/2023

|  |  |
| --- | --- |
| **Exercise (Objective)** | 24. Demonstrate the implementation of ActionListener for the button and mouse click event. |
| **Prerequisites** | The Prerequisite required is the knowledge of the different components of swing and Awt to which you will add the ActionListener and MouseListener. |
| **Underlying Concept / Theory** | The Java ActionListener is notified whenever you click on the button or menu item. It is notified against ActionEvent. The ActionListener interface is found in java.awt.event [package](https://www.javatpoint.com/package). It has only one method: actionPerformed().Java MouseListener is notified whenever you change the state of mouse. It is notified against MouseEvent. The MouseListener interface is found in java.awt.event package. It has five methods. |
| **Problem Statement** | Simple implementation to show how to add Action Listener and mouse Listener to any component. |
| **Algorithm / Step-by-step instructions** | 1.Create a simple class and a Constructir.In it Declare and define some textfields and a button.  2.Now, to the button add Action Listener and implement the actionPerformed() method and do the necessary action.  3.Similarly to the mouse add the Mouse listener and implement the 5 abstract methods it has and take the necessary actions. |
| **Code** | **import** java.awt.\*;  **import** java.awt.event.\*;  **public** **class** ActionListenerExample {  TextField l;  ActionListenerExample(){    Frame f=**new** Frame("ActionListener Example");  l=**new** TextField();  TextField tf=**new** TextField();  tf.setBounds(50,50, 150,20);  Button b=**new** Button("Click Here");  b.setBounds(50,100,60,30);  l.setBounds(50,150,60,30);  b.addActionListener(**new** ActionListener(){  **public** **void** actionPerformed(ActionEvent e){  tf.setText("Welcome");  }  });  f.add(b);f.add(tf);  f.add(l);  f.addMouseListener(**new** MouseListener() {    **public** **void** mouseClicked(MouseEvent e) {  l.setText("Mouse Clicked");  }  **public** **void** mouseEntered(MouseEvent e) {  l.setText("Mouse Entered");  }  **public** **void** mouseExited(MouseEvent e) {  l.setText("Mouse Exited");  }  **public** **void** mousePressed(MouseEvent e) {  l.setText("Mouse Pressed");  }  **public** **void** mouseReleased(MouseEvent e) {  l.setText("Mouse Released");  }    });  f.setSize(400,400);  f.setLayout(**null**);  f.setVisible(**true**);  }    **public** **static** **void** main(String[] args) {  **new** ActionListenerExample();  }} |
| **Output** |  |
| **Conclusion / Remarks** | Implemented and understood the usage of the Interfaces ActionListener and MouseListener. |
| **Exercises /**  **Challenges** | Create a frame and change its color on different Mouse Events. And also do different operations on clicking buttons or Menu Items. |

Date:2/05/2023

|  |  |
| --- | --- |
| **Exercise (Objective)** | 25. Demonstrate the implementation of various swing controls (JTabbedPane, JPanel, JScrollBar, File Chooser etc.) on Form (JFrame). |
| **Prerequisites** | The Prerequisite required is knowledge of swing and awt classes and their implementation to be able to use different components. |
| **Underlying Concept / Theory** | The object of JFileChooser class represents a dialog window from which the user can select file. It inherits JComponent class.The JTabbedPane class is used to switch between a group of components by clicking on a tab with a given title or icon. It inherits JComponent class. |
| **Problem Statement** | Implement JTabbedPane, JScrollBar and JfileChooser. |
| **Algorithm / Step-by-step instructions** | 1.Create a Frame and add tabbedPane and add 3 panels to it.  2. In one panel add a a Scroll Bar and for the other we add the File Chooser.  3.For File Chooser we create a Menu Bar and add actionListener to it within which we create object of JFileChooser class and get the path of selected file and read its contents into the text area. |
| **Code** | **import** javax.swing.\*;  **import** java.awt.event.\*;  **import** java.io.BufferedReader;  **import** java.io.File;  **import** java.io.FileReader;  **import** java.io.\*;  **import** javax.swing.JFileChooser;  **public** **class** TabbedPane **implements** ActionListener {  JFrame f;  JMenuBar mb;  JMenu file;  JMenuItem open;  JTextArea ta;  TabbedPane(){  f=**new** JFrame();  JTextArea ta=**new** JTextArea(200,200);  JPanel p1=**new** JPanel();  p1.add(ta);  JPanel p2=**new** JPanel();  JPanel p3=**new** JPanel();  JTabbedPane tp=**new** JTabbedPane();  tp.setBounds(50,50,200,200);  tp.add("main",p1);  tp.add("visit",p2);  tp.add("help",p3);  f.add(tp);    // Scroll Bar  JScrollBar s=**new** JScrollBar();  s.setBounds(100,100, 50,100);  p2.add(s);        // JFileChooser  open=**new** JMenuItem("Open File");  open.addActionListener(**this**);  file=**new** JMenu("File");  file.add(open);  mb=**new** JMenuBar();  mb.setBounds(0,0,800,20);  mb.add(file);  ta=**new** JTextArea(800,800);  ta.setBounds(0,20,800,800);  p3.add(mb);  p3.add(ta);    f.setSize(400,400);  f.setLayout(**null**);  f.setVisible(**true**);  }  **public** **void** actionPerformed(ActionEvent e) {  **if**(e.getSource()==open){  JFileChooser fc=**new** JFileChooser();  **int** i=fc.showOpenDialog(f);  **if**(i==JFileChooser.***APPROVE\_OPTION***){  File f=fc.getSelectedFile();  String filepath=f.getPath();  **try**{  BufferedReader br=**new** BufferedReader(**new** FileReader(filepath));  String s1="",s2="";  **while**((s1=br.readLine())!=**null**){  s2+=s1+"\n";  }  ta.setText(s2);  br.close();  }**catch** (Exception ex) {ex.printStackTrace(); }  }  }  }  **public** **static** **void** main(String[] args) {  **new** TabbedPane();  }} |
| **Output** |  |
| **Conclusion / Remarks** | The Tabbed Pane, Scroll bar and File Chooser successfully implemented. |
| **Exercises /**  **Challenges** | Create a form that uses more such components like JProgress bar etc. |

Date:03/05/2023

|  |  |
| --- | --- |
| **Exercise (Objective)** | 26. Installation of WindowBuilder extension over Eclipse. |
| **Prerequisites** | The Prerequisites include having latest eclipse IDE installed on the system. |
| **Underlying Concept / Theory** | Eclipse WindowBuilder is composed of Eclipse SWT Designer and Eclipse Swing Designer and makes it very easy to create Java GUI applications without spending a lot of time writing code. Use the WYSIWYG visual designer and layout tools to create simple forms to complex windows; the Java code will be generated for you. |
| **Problem Statement** | We need to install WindowBuilder and start creating simple forms. |
| **Algorithm / Step-by-step instructions** | Open Eclipse: Launch Eclipse IDE on your computer.  Navigate to the Eclipse Marketplace: Go to the "Help" menu in the Eclipse toolbar and select "Eclipse Marketplace." This will open the Eclipse Marketplace window.  Search for WindowBuilder: In the Eclipse Marketplace window, type "WindowBuilder" in the search bar located in the top right corner and press Enter.  Select the WindowBuilder entry: Look for the WindowBuilder entry in the search results and click on it.  Choose your Eclipse version: On the WindowBuilder page, you will see a drop-down menu labeled "Select Eclipse Version." Choose the version of Eclipse that you are using. If you are unsure, you can check by going to "Help" > "About Eclipse."  Click "Go to installation": Once you have selected your Eclipse version, click on the "Go to installation" link on the right side of the page. This will take you to the download page for WindowBuilder.  Download WindowBuilder: On the download page, you will see a list of available download options. Choose the appropriate download link for your operating system and click on it to start the download.  Install WindowBuilder: After the download is complete, locate the downloaded file on your computer and extract its contents if necessary. Then, follow the installation instructions provided with the download to install WindowBuilder into your Eclipse IDE.  Restart Eclipse: Once the installation is complete, restart Eclipse to activate the WindowBuilder extension.  Verify the installation: After restarting Eclipse, you can verify the installation by creating a new Java project or opening an existing one. Right-click on a package or source file in the Package Explorer, and from the context menu, select "Open With" > "WindowBuilder Editor." If the WindowBuilder editor opens without any errors, it means that the installation was successful. |
| **Code** |  |
| **Output** | Windows builder successfully installed. |
| **Conclusion / Remarks** | We can utilize the features provided by WindowBuilder to visually design and layout the GUI components. |
| **Exercises /**  **Challenges** | Creating a form using the features of WindowsBuilder. |

Date:05/05/2023

|  |  |
| --- | --- |
| **Exercise (Objective)** | 27. Depict the life cycle of JSP using example program |
| **Prerequisite** | Basic understanding of JSP, HTML, and Java programming. |
| **Underlying Concept / Theory** | JSP has a well-defined life cycle that consists of several phases:   1. Translation: The JSP engine translates the JSP into a servlet. 2. Compilation: The translated JSP is compiled into bytecode by the servlet container. 3. Initialization: The servlet container initializes the JSP by creating an instance of the servlet class. 4. Execution: The JSP is executed by invoking the service() method of the servlet, which generates the dynamic content. 5. Destruction: When the JSP is no longer needed or the server shuts down, the servlet container destroys the JSP instance. |
| **Problem Statement** | The problem is to depict the life cycle of a JSP (JavaServer Pages) by implementing a simple JSP program. Understand and demonstrate the different phases of the JSP life cycle, including translation, compilation, initialization, execution, and destruction. |
| **Algorithm / Step-by-step instructions** | 1. Create a new JSP file with a **.jsp** extension. 2. Inside the JSP, use directives such as **<%@ page %>** to specify the language and import necessary classes. 3. Write the HTML content within the **<html>** tags. 4. Embed Java code within **<% %>** tags to include dynamic content or perform server-side logic. 5. Compile the JSP file using a JSP container or an integrated development environment (IDE). 6. Deploy the compiled JSP file to a servlet container or application server. 7. Access the JSP file through a web browser, which triggers the JSP life cycle. 8. The JSP engine translates the JSP file into a servlet. 9. The servlet container compiles the servlet into bytecode. 10. The container initializes the JSP servlet by creating an instance of the servlet class. 11. The JSP servlet's **service()** method is invoked, generating the dynamic content. 12. The output is sent back to the client's browser for rendering. 13. When the JSP is no longer needed or the server shuts down, the servlet container destroys the JSP instance. |
| **Code** | <%@ **page** language=*"java"* %>  <%@ **page** import=*"java.util.Date"* %>  <**html**>  <**head**>  <**title**>JSP Life Cycle Example</**title**>  </**head**>  <**body**>  <**h1**>JSP Life Cycle Example</**h1**>  <**p**>Current Date and Time: <%= new Date() %></**p**>  </**body**>  </**html**> |
| **Output** |  |
| **Conclusion / Remarks** | The life cycle of a JSP involves several phases, including translation, compilation, initialization, execution, and destruction. Understanding the JSP life cycle is crucial for developing and deploying JSP applications. By following the life cycle, JSP files can be translated, compiled, and executed by the servlet container, allowing dynamic content to be generated and served to clients. |
| **Exercises /**  **Challenges** | **Managing Dependencie**: JSPs may rely on external resources or libraries, which need to be properly managed and configured for the JSP to function correctly. |

Date:08/05/2023

|  |  |
| --- | --- |
| **Exercise (Objective)** | 28. Demonstrate the creation of JSP in Eclipse. |
| **Prerequisite** | 1. Install Eclipse IDE. 2. Set up a web server like Apache Tomcat. |
| **Underlying Concept / Theory** | JSP is a technology used to create dynamic web pages by embedding Java code within HTML. Eclipse is an integrated development environment that supports JSP development and provides tools for creating, editing, and running JSP files. |
| **Problem Statement** | The problem is to demonstrate the creation of a JSP (JavaServer Pages) file using Eclipse IDE. Develop a simple JSP program that displays a message on a web page. |
| **Algorithm / Step-by-step instructions** | 1. Create a new JSP file with a **.jsp** extension. 2. Inside the JSP, use directives such as **<%@ page %>** to specify the language and import necessary classes. 3. Write the HTML content within the **<html>** tags. 4. Embed Java code within **<% %>** tags to include dynamic content or perform server-side logic. 5. Compile the JSP file using a JSP container or an integrated development environment (IDE). 6. Deploy the compiled JSP file to a servlet container or application server. 7. Access the JSP file through a web browser, which triggers the JSP life cycle. 8. The JSP engine translates the JSP file into a servlet. 9. The servlet container compiles the servlet into bytecode. 10. The container initializes the JSP servlet by creating an instance of the servlet class. 11. The JSP servlet's **service()** method is invoked, generating the dynamic content. 12. The output is sent back to the client's browser for rendering. 13. When the JSP is no longer needed or the server shuts down, the servlet container destroys the JSP instance. |
| **Code** | <%@ **page** language=*"java"* %>  <**html**>  <**head**>  <**title**>My First JSP</**title**>  </**head**>  <**body**>  <**h1**>Hello, JSP!</**h1**>  <**p**>Welcome to my first JSP page.</**p**>  </**body**>  </**html**> |
| **Output** |  |
| **Conclusion / Remarks** | By using Eclipse IDE and following the provided steps, a JSP file can be created and executed within a web project. Eclipse provides a convenient environment for JSP development, allowing developers to combine HTML and Java code seamlessly to build dynamic web pages. |
| **Exercises /**  **Challenges** | Setting up a web server like Apache Tomcat and configuring it within Eclipse can be challenging, especially for beginners. |

Date:10/05/2023

|  |  |
| --- | --- |
| **Exercise (Objective)** | 29. Demonstrate the creation of JSP in Eclipse. |
| **Prerequisite** | 1. Install a web application server such as Apache Tomcat. 2. Configure the web application server in Eclipse or a preferred IDE. |
| **Underlying Concept / Theory** | HTML is used for creating the user interface and capturing user input. JSP is responsible for processing the user input and generating dynamic content. The web application server receives the HTTP request, invokes the appropriate JSP file, and sends the generated response back to the client. |
| **Problem Statement** | The problem is to demonstrate the interaction between HTML, JSP (JavaServer Pages), and a web application server. Develop a simple web application that takes user input through an HTML form, processes it using JSP, and displays the result on a web page. |
| **Algorithm / Step-by-step instructions** | 1. Create an HTML file with the desired user interface using HTML tags. 2. Define an HTML form that captures user input and specifies the action URL to the JSP file for processing. 3. Create a JSP file that retrieves the user input from the request object, performs the necessary processing, and generates the dynamic content. 4. Deploy the HTML and JSP files to the web application server. 5. Start the web application server and access the HTML file through a web browser. 6. Enter the required input in the form and submit it. 7. The web application server receives the request, identifies the associated JSP file, and passes the request to the JSP. 8. The JSP retrieves the user input, processes it, and generates the dynamic content. 9. The web application server sends the generated response back to the client's browser. 10. The client's browser displays the processed result on the web page. |
| **Code** | <!**DOCTYPE** html>  <**html**>  <**head**>  <**title**>Input Form</**title**>  </**head**>  <**body**>  <**h1**>Input Form</**h1**>  <**form** action=*"Q29.jsp"* method=*"post"*>  <**label** for=*"name"*>Enter your name:</**label**>  <**input** type=*"text"* name=*"name"* required>  <**br**>  <**input** type=*"submit"* value=*"Submit"*>  </**form**>  </**body**>  </**html**>  -------- JSP CODE ------------------  <%@ **page** language=*"java"* %>  <%@ **page** import=*"java.util.Date"* %>  <**html**>  <**head**>  <**title**>Process JSP</**title**>  </**head**>  <**body**>  <**h1**>Greeting</**h1**>  <%  String name = request.getParameter("name");  out.println("Hello, " + name + "!");  %>  <**p**>Current Date and Time: <%= new Date() %></**p**>  </**body**>  </**html**> |
| **Output** | -------------------------------------------------------------------------- |
| **Conclusion / Remarks** | By using Eclipse IDE and following the provided steps, a JSP file can be created and executed within a web project. Eclipse provides a convenient environment for JSP development, allowing developers to combine HTML and Java code seamlessly to build dynamic web pages. |
| **Exercises /**  **Challenges** | Setting up a web server like Apache Tomcat and configuring it within Eclipse can be challenging, especially for beginners. |

Date:12/05/2023

|  |  |
| --- | --- |
| **Exercise (Objective)** | 30. Implement / Demonstrate the concept of Servlet |
| **Prerequisite** | 1. Java Development Kit (JDK) installed on your machine. 2. A web application server like Apache Tomcat or Jetty. |
| **Underlying Concept / Theory** | Servlets are Java classes that extend the javax.servlet.http.HttpServlet class. They are used to handle HTTP requests and generate responses. Servlets are executed on the server-side and provide dynamic functionality to web applications. |
| **Problem Statement** | The problem is to demonstrate the creation of a JSP (JavaServer Pages) file using Eclipse IDE. Develop a simple JSP program that displays a message on a web page. |
| **Algorithm / Step-by-step instructions** | 1. Create a Java class that extends javax.servlet.http.HttpServlet. 2. Override the appropriate doGet or doPost method based on the HTTP method you want to handle (GET or POST). 3. Inside the method, write the code to generate the desired response. 4. Compile the Java class to generate the .class file. 5. Package the compiled .class file into a WAR (Web ARchive) file. 6. Deploy the WAR file to a web application server. 7. Start the web application server and access the servlet using its URL. |
| **Code** | import jakarta.servlet.ServletException;  import jakarta.servlet.annotation.WebServlet;  import jakarta.servlet.http.HttpServlet;  import jakarta.servlet.http.HttpServletRequest;  import jakarta.servlet.http.HttpServletResponse;  import java.io.IOException;  /\*\*  \* Servlet implementation class Q30  \*/  public class Q30 extends HttpServlet {  private static final long ***serialVersionUID*** = 1L;  /\*\*  \* **@see** HttpServlet#HttpServlet()  \*/  public Q30() {  super();  // **TODO** Auto-generated constructor stub  }    /\*\*  \* **@see** HttpServlet#doGet(HttpServletRequest request, HttpServletResponse response)  \*/  protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {  // **TODO** Auto-generated method stub  response.setContentType("text/html");  response.getWriter().println("<h1>Hello, Servlet!</h1>");  }    /\*\*  \* **@see** HttpServlet#doPost(HttpServletRequest request, HttpServletResponse response)  \*/  protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {  // **TODO** Auto-generated method stub  doGet(request, response);  }  } |
| **Output** |  |
| **Conclusion / Remarks** | Servlets are a powerful component of Java web development that enables the handling of HTTP requests and generating dynamic responses. By extending the HttpServlet class and overriding the appropriate methods, developers can implement servlets to process user requests and generate custom responses. |
| **Exercises /**  **Challenges** | Handling Different HTTP Methods: Servlets can handle various HTTP methods like GET, POST, PUT, DELETE, etc. Handling different methods and processing their associated data can introduce complexity. |

Date:14/05/2023

|  |  |
| --- | --- |
| **Exercise (Objective)** | 31. Implement / Demonstrate the concept of Servlet |
| **Prerequisite** | 1. Java Development Kit (JDK) installed on your machine. 2. A web application server like Apache Tomcat or Jetty. 3. Configuration of the web application server in your IDE. |
| **Underlying Concept / Theory** | A web application using servlets involves building a dynamic web application that utilizes servlets to handle client requests, process data, and generate dynamic responses. It typically involves multiple servlets working together to achieve specific functionalities and is deployed on a web server |
| **Problem Statement** | The problem is to create a web application using Servlets in Java. Develop a simple web application that allows users to calculate the area of a rectangle by entering its length and width. The application should display the calculated area on a web page. |
| **Algorithm / Step-by-step instructions** | 1. Create an HTML file with a form that collects the length and width of the rectangle. 2. Specify the form's action URL to point to the Servlet that will handle the request. 3. Create a Servlet class that extends **javax.servlet.http.HttpServlet**. 4. Override the **doPost** method to handle the POST request from the form submission. 5. Inside the **doPost** method, extract the length and width parameters from the request. 6. Calculate the area of the rectangle using the extracted length and width. 7. Set the content type of the response to "text/html". 8. Use the **getWriter** method of the response object to write the calculated area as HTML. 9. Package the Servlet class along with any required configuration files into a WAR (Web ARchive) file. 10. Deploy the WAR file to a web application server. 11. Start the web application server and access the web application through a web browser. 12. Enter the length and width of the rectangle in the form and submit it. 13. The web application server will invoke the Servlet, process the form data, calculate the area, and display it on the web page. |
| **Code** | **Index.html ------------------------**  <!**DOCTYPE** html>  <**html**>  <**head**>  <**title**>Rectangle Area Calculator</**title**>  </**head**>  <**body**>  <**h1**>Rectangle Area Calculator</**h1**>  <**form** action=*"calculate"* method=*"post"*>  <**label** for=*"length"*>Enter length:</**label**>  <**input** type=*"number"* name=*"length"* required>  <**br**>  <**label** for=*"width"*>Enter width:</**label**>  <**input** type=*"number"* name=*"width"* required>  <**br**>  <**input** type=*"submit"* value=*"Calculate Area"*>  </**form**>  </**body**>  </**html**>  **Servlet – calculate.java ----------------------------------------------**  import jakarta.servlet.ServletException;  import jakarta.servlet.annotation.WebServlet;  import jakarta.servlet.http.HttpServlet;  import jakarta.servlet.http.HttpServletRequest;  import jakarta.servlet.http.HttpServletResponse;  import java.io.IOException;    /\*\*  \* Servlet implementation class Q31  \*/  public class Q31 extends HttpServlet {  private static final long serialVersionUID = 1L;    /\*\*  \* @see HttpServlet#HttpServlet()  \*/  public Q31() {  super();  // TODO Auto-generated constructor stub  }    /\*\*  \* @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse response)  \*/  protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {  // TODO Auto-generated method stub  response.getWriter().append("Served at: ").append(request.getContextPath());  }    /\*\*  \* @see HttpServlet#doPost(HttpServletRequest request, HttpServletResponse response)  \*/  protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {  // TODO Auto-generated method stub  int length = Integer.parseInt(request.getParameter("length"));  int width = Integer.parseInt(request.getParameter("width"));  int area = length \* width;    response.setContentType("text/html");  response.getWriter().println("<h1>Rectangle Area: " + area + "</h1>");  }    } |
| **Output** | ----------------------------------------------------------------------------- |
| **Conclusion / Remarks** | Servlets provide a powerful mechanism for handling HTTP requests and generating dynamic responses in Java web applications. By implementing a Servlet and configuring it within a web application server, we can easily create web applications that interact with users, process their input, and generate calculated results dynamically. |
| **Exercises /**  **Challenges** | **Servlet Configuration**: Configuring the Servlet in the web application server correctly, including mapping the Servlet to the appropriate URL and ensuring proper deployment of the application. |

Date:16/05/2023

Date:18/05/2023

|  |  |
| --- | --- |
| **Exercise (Objective)** | 32. Implement / Demonstrate the concept of Servlet |
| **Prerequisite** | Java Development Kit (JDK) installed on your machine. |
| **Underlying Concept / Theory** | Socket programming allows communication between client and server applications over a network. In this example, we create a client application that establishes a connection with a server application. The client sends a message to the server, and the server responds with the reversed version of the message. |
| **Problem Statement** | The problem is to create a connection-oriented client-server application using socket programming in Java. Develop a client application that sends a message to a server application, and the server application responds with a reversed version of the message. |
| **Algorithm / Step-by-step instructions** | 1. Implement the server side by creating a **ServerSocket** object and specifying a port number (e.g., 8080). 2. Accept client connections using the **accept** method of the **ServerSocket** class, which blocks until a client connects. 3. Create **BufferedReader** and **PrintWriter** objects for reading from and writing to the client's socket. 4. Read the message sent by the client using the **readLine** method and reverse it. 5. Write the reversed message to the client using the **println** method of the **PrintWriter** object. 6. Close the reader, writer, client socket, and server socket. 7. Implement the client side by creating a **Socket** object and specifying the server's IP address or hostname and port number. 8. Create **BufferedReader** and **PrintWriter** objects for reading from and writing to the server's socket. 9. Write a message to the server using the **println** method of the **PrintWriter** object. 10. Read the reversed message from the server using the **readLine** method of the **BufferedReader** object. 11. Print the reversed message to the console. 12. Close the reader, writer, and socket. |
| **Code** | **Client.java ------------------------**  import java.io.\*;  import java.util.\*;  import java.net.Socket;  public class client {  public static void main(String args[])throws IOException {    Socket socket=new Socket("localhost",4444);  Scanner sc = new Scanner(System.in);    //Input Data Stream  System.out.print("Enter message bits: ");  String message = sc.nextLine();  System.out.print("Enter generator: ");  String generator = sc.nextLine();  int data[] = new int[message.length() + generator.length() - 1];  int divisor[] = new int[generator.length()];  for(int i=0;i<message.length();i++)  data[i] = Integer.parseInt(message.charAt(i)+"");  for(int i=0;i<generator.length();i++)  divisor[i] = Integer.parseInt(generator.charAt(i)+"");    //Calculation of CRC  for(int i=0;i<message.length();i++)  {  if(data[i]==1)  for(int j=0;j<divisor.length;j++)  data[i+j] ^= divisor[j];  }    //Display CRC  String checksum="";  System.out.print("The checksum code is: ");  for(int i=0;i<message.length();i++)  data[i] = Integer.parseInt(message.charAt(i)+"");  for(int i=0;i<data.length;i++)  checksum+=Integer.toString(data[i]);  System.out.println(checksum);        DataOutputStream cdout=new DataOutputStream(socket.getOutputStream());  cdout.writeUTF(checksum);  }  }  **Server.java ----------------------------------------------**  import java.io.\*;  import java.net.\*;  import java.util.Scanner;  public class server {  public static void main(String args[])throws IOException {  ServerSocket serversocket=new ServerSocket(4444);  Socket socket=serversocket.accept();  DataInputStream sdis=new DataInputStream(socket.getInputStream());  //Check for input CRC code  Scanner sc = new Scanner(System.in);  String message = sdis.readUTF();  System.out.print("Enter generator: ");  String generator = sc.nextLine();  int data[] = new int[message.length() + generator.length() - 1];  int divisor[] = new int[generator.length()];  for(int i=0;i<message.length();i++)  data[i] = Integer.parseInt(message.charAt(i)+"");  for(int i=0;i<generator.length();i++)  divisor[i] = Integer.parseInt(generator.charAt(i)+"");    //Calculation of remainder  for(int i=0;i<message.length();i++) {  if(data[i]==1)  for(int j=0;j<divisor.length;j++)  data[i+j] ^= divisor[j];  }  //Display validity of data  boolean valid = true;  for(int i=0;i<data.length;i++)  if(data[i]==1){  valid = false;  break;  }  if(valid==true)  System.out.println("Data stream is valid");  else  System.out.println("Data stream is invalid. CRC error occurred.");  }  } |
| **Output** |  |
| **Conclusion / Remarks** | Connection-oriented socket programming allows communication between client and server applications. It enables data exchange over a network using sockets. In this example, we demonstrated how to establish a connection, send a message from the client to the server, and receive a response from the server. The server application reverses the message and sends it back to the client. This demonstrates the bidirectional communication between the client and server using sockets. |
| **Exercises /**  **Challenges** | Handling Multiple Clients: Modifying the server code to handle multiple client connections concurrently by using threading or non-blocking I/O. **Error Handling**: Implementing error handling mechanisms to handle various exceptions that may occur during socket communication, such as connection errors or data transmission issues. |