1)

```
interface Shape {
    double calculateArea();
    double calculatePerimeter();
}
class Circle implements Shape {
    private double radius;

    public Circle(double radius) {
        this.radius = radius;
    }

    @Override
    public double calculateArea() {
        return Math.PI * radius * radius;
    }

    @Override
    public double calculatePerimeter() {
        return 2 * Math.PI * radius;
    }
}

class Triangle implements Shape {
    private double side1, side2, side3;

    public Triangle(double side1, double side2, double side3) {
        this.side1 = side1;
        this.side2 = side2;
        this.side3 = side3;
    }
```

```
    @Override
    public double calculateArea() {
        // Using Heron's formula to calculate the area of a triangle
        double s = (side1 + side2 + side3) / 2;
        return Math.sqrt(s * (s - side1) * (s - side2) * (s - side3));
    }

    @Override
    public double calculatePerimeter() {
        return side1 + side2 + side3;
    }
}

public class a1 {
    public static void main(String[] args) {
        Circle circle = new Circle(5);
        Triangle triangle = new Triangle(3, 4, 5);
        System.out.println("Circle:");
        System.out.println("Area: " + circle.calculateArea());
        System.out.println("Perimeter: " + circle.calculatePerimeter());

        System.out.println("\nTriangle:");
        System.out.println("Area: " + triangle.calculateArea());
        System.out.println("Perimeter: " + triangle.calculatePerimeter());
    }
}

2)
class Parent {
    public Parent() {
```

```
        System.out.println("Parent class constructor invoked.");
    }
}

class Child extends Parent {
    public Child() {
        super(); // Invoking parent class constructor
        System.out.println("Child class constructor invoked.");
    }
}
public class a2 {
    public static void main(String[] args) {
        Child child = new Child(); // Creating Child class object
    }
}
```

Key Points about Constructors:

1. Constructors are special methods in a class that are used to initialize objects of that class.
2. The name of the constructor is the same as the class name.
3. Constructors do not have a return type, not even **void**.
4. Constructors are automatically called when an object of the class is created using the **new** keyword.
5. If a class does not define any constructor, a default constructor (with no arguments) is automatically provided by the compiler.
6. Constructors can be overloaded, meaning that a class can have multiple constructors with different parameter lists.
7. The **super()** keyword is used to invoke the parent class constructor from a child class constructor. It must be the first statement in the child class constructor if used.
8. If the parent class does not have a no-argument constructor, the child class constructor must explicitly call one of the parent class constructors using **super()** with appropriate arguments.
9. Constructors can have access modifiers (e.g., public, private, protected), which determine the visibility of the constructor.

3)

```
import java.util.Scanner;

public class a3 {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter an integer: ");
        int number = scanner.nextInt();

        try {
            if (number < 0) {
                throw new IllegalArgumentException("Negative number not allowed");
            }
            System.out.println("Entered number: " + number);
        } catch (IllegalArgumentException e) {
            System.out.println("Exception: " + e.getMessage());
        }
    }
}
```

4)

```
import java.util.Scanner;

class BankAccount {
    private double balance;

    public BankAccount() {
        balance = 0.0;
    }

    public void deposit(double amount) {
        if (amount > 0) {
```

```
            balance += amount;
            System.out.println("Deposit successful. Current balance: " + balance);
        } else {
            System.out.println("Invalid amount. Deposit failed.");
        }
    }

    public void withdraw(double amount) {
        if (amount > 0 && amount <= balance) {
            balance -= amount;
            System.out.println("Withdrawal successful. Current balance: " + balance);
        } else {
            System.out.println("Insufficient funds or invalid amount. Withdrawal failed.");
        }
    }

    public double getBalance() {
        return balance;
    }
}

public class a4 {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        BankAccount account = new BankAccount();

        while (true) {
            System.out.println("\nBank Account Menu:");
            System.out.println("1. Deposit");
            System.out.println("2. Withdraw");
            System.out.println("3. Check Balance");
```

```
            System.out.println("4. Exit");
            System.out.print("Enter your choice: ");
            int choice = scanner.nextInt();

            switch (choice) {
                case 1:
                    System.out.print("Enter deposit amount: ");
                    double depositAmount = scanner.nextDouble();
                    account.deposit(depositAmount);
                    break;
                case 2:
                    System.out.print("Enter withdrawal amount: ");
                    double withdrawalAmount = scanner.nextDouble();
                    account.withdraw(withdrawalAmount);
                    break;
                case 3:
                    System.out.println("Current balance: " + account.getBalance());
                    break;
                case 4:
                    System.out.println("Exiting program. Thank you!");
                    System.exit(0);
                default:
                    System.out.println("Invalid choice. Please try again.");
            }
        }
    }
}
```

5)

Abstract class:

```
abstract class Animal {
    abstract void makeSound();
    void eat() {
        System.out.println("Animal is eating.");
    }
}

class Dog extends Animal {
    @Override
    void makeSound() {
        System.out.println("Dog is barking.");
    }
}

public class a5 {
    public static void main(String[] args) {
        Dog dog = new Dog();
        dog.makeSound();
        dog.eat();
    }
}
```

Interface:

```
interface Animal {
    void makeSound();
    void eat();
}

class Dog implements Animal {
    @Override
```

```
    public void makeSound() {
        System.out.println("Dog is barking.");
    }

    @Override
    public void eat() {
        System.out.println("Dog is eating.");
    }
}

public class a5 {
    public static void main(String[] args) {
        Dog dog = new Dog();
        dog.makeSound();
        dog.eat();
    }
}
```

Key Points:

1. Abstract Class:
    - An abstract class can have both abstract and non-abstract methods.
    - Abstract methods are declared without an implementation and must be overridden by the concrete subclasses.
    - An abstract class can have instance variables, constructors, and other concrete methods.
    - Abstract classes cannot be directly instantiated, but they can be used as a superclass for creating subclasses.
    - Subclasses extend a single abstract class.
2. Interface:
    - An interface defines a contract that classes implementing it must adhere to.
    - All methods in an interface are implicitly abstract and public.
    - Interfaces cannot have instance variables or constructors, but they can have constants (public static final variables).
    - A class can implement multiple interfaces.

- Interfaces provide a way for unrelated classes to achieve polymorphism through a common interface.

6)

```
import java.util.ArrayList;
import java.util.List;
public class StreamExample {
   public static void a6(String[] args) {
      List<Integer> numbers = new ArrayList<>();
      for (int i = 1; i <= 1000000; i++) {
         numbers.add(i);
      }
      long sum = numbers.stream()
            .filter(n -> n % 2 == 0) // Filter even numbers
            .mapToLong(n -> n) // Convert to long
            .sum(); // Calculate sum

      System.out.println("Sum of even numbers: " + sum);
   }
}
```

7)

```
import java.util.Arrays;
import java.util.Scanner;

public class a7 {
   public static int binarySearch(int[] arr, int target) {
      int left = 0;
      int right = arr.length - 1;

      while (left <= right) {
         int mid = left + (right - left) / 2;
```

```
            if (arr[mid] == target) {
                return mid;
            } else if (arr[mid] < target) {
                left = mid + 1;
            } else {
                right = mid - 1;
            }
        }

        return -1;
    }

    public static void main(String[] args) {
        int[] arr = {2, 4, 6, 8, 10, 12, 14, 16, 18, 20};
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter the target value: ");
        int target = scanner.nextInt();
        int index = binarySearch(arr, target);
        if (index != -1) {
            System.out.println("Target value found at index: " + index);
        } else {
            System.out.println("Target value not found.");
        }
    }
}
8)
class EvenThread extends Thread {
    @Override
    public void run() {
        for (int i = 2; i <= 10; i += 2) {
            System.out.println("Even Thread: " + i);
```

```
        }
    }
}

class OddThread extends Thread {
    @Override
    public void run() {
        for (int i = 1; i <= 9; i += 2) {
            System.out.println("Odd Thread: " + i);
        }
    }
}

public class a8 {
    public static void main(String[] args) {
        EvenThread evenThread = new EvenThread();
        OddThread oddThread = new OddThread();

        evenThread.start();
        oddThread.start();
    }
}
```

9)

```
import java.util.LinkedList;
import java.util.Queue;
import java.util.Random;

class Producer implements Runnable {
    private Queue<Integer> queue;
    private int maxSize;
    private Random random;
```

```
    public Producer(Queue<Integer> queue, int maxSize) {
        this.queue = queue;
        this.maxSize = maxSize;
        this.random = new Random();
    }

    @Override
    public void run() {
        while (true) {
            synchronized (queue) {
                while (queue.size() == maxSize) {
                    try {
                        queue.wait();
                    } catch (InterruptedException e) {
                        e.printStackTrace();
                    }
                }

                int number = random.nextInt(100);
                queue.offer(number);
                System.out.println("Producer produced: " + number);
                queue.notify();
            }
        }
    }
}

class Consumer implements Runnable {
    private Queue<Integer> queue;
```

```
    public Consumer(Queue<Integer> queue) {
        this.queue = queue;
    }

    @Override
    public void run() {
        while (true) {
            synchronized (queue) {
                while (queue.isEmpty()) {
                    try {
                        queue.wait();
                    } catch (InterruptedException e) {
                        e.printStackTrace();
                    }
                }

                int number = queue.poll();
                System.out.println("Consumer consumed: " + number);
                queue.notify();
                int sum = 0;
                for (int num : queue) {
                    sum += num;
                }
                System.out.println("Sum: " + sum);
            }
        }
    }
}

public class a9 {
    public static void main(String[] args) {
```

```
        Queue<Integer> queue = new LinkedList<>();
        int maxSize = 5;

        Thread producerThread = new Thread(new Producer(queue, maxSize));
        Thread consumerThread = new Thread(new Consumer(queue));

        producerThread.start();
        consumerThread.start();
    }
}
```

10)

```
import java.util.ArrayList;
import java.util.Collections;
import java.util.List;
import java.util.Scanner;

public class a10 {
    public static void main(String[] args) {
        List<Integer> numbers = new ArrayList<>();
        Scanner scanner = new Scanner(System.in);

        System.out.print("Enter the number of elements: ");
        int n = scanner.nextInt();

        System.out.println("Enter the elements:");
        for (int i = 0; i < n; i++) {
            int number = scanner.nextInt();
            numbers.add(number);
        }
```

```
        if (numbers.size() < 2) {
            System.out.println("Insufficient number of elements.");
        } else {
            int secondSmallest = findSecondSmallest(numbers);
            int secondLargest = findSecondLargest(numbers);

            System.out.println("Second Smallest: " + secondSmallest);
            System.out.println("Second Largest: " + secondLargest);
        }
    }

    public static int findSecondSmallest(List<Integer> numbers) {
        Collections.sort(numbers);
        return numbers.get(1);
    }

    public static int findSecondLargest(List<Integer> numbers) {
        Collections.sort(numbers, Collections.reverseOrder());
        return numbers.get(1);
    }
}
```

11)

```
package ineuron;
import java.sql.*;
public class Ineuron
{
public static void main(String[] args)
{
 try
 {
```

```
Class.forName("com.mysql.jdbc.Driver");
Connection con = DriverManager.getConnection("jdbc:mysql://localhost/souji","root","");
String sqlres = "select * from student";
PreparedStatement ps = con.prepareStatement(sqlres);
ResultSet rs = ps.executeQuery();
while(rs.next())
{
System.out.println("Student Roll Number : " + rs.getInt(1));
System.out.println("Student Name : " + rs.getString(2));
System.out.println("Student age : " + rs.getInt(3));
System.out.println("Student address : " + rs.getString(4));
System.out.println();
}
}
catch(Exception e)
{
System.out.println(e);
}
}
}
```

12)

```
package ineuron;
import java.io.*;
import java.sql.*;
import java.util.*;
public class ineuron12
{
public static void main(String[] args)
{
 try
```

```
 {
Scanner s = new Scanner(System.in);
 System.out.print("Enter Eno : ");
 int a = s.nextInt();
 System.out.print("Enter Ename : ");
 String ss = s.next();
 Class.forName("com.mysql.jdbc.Driver");
 Connection con = DriverManager.getConnection("jdbc:mysql://localhost/souji","root","");
 Statement st = con.createStatement();
 String sqltable = "create table employee (eno int, name varchar(20))";
 //st.executeUpdate(sqltable);
 String sqlinsert = "insert into employee values(4,'kumar')";
 //st.executeUpdate(sqlinsert);
 String insertdynamic = "insert into employee values("+a+", '"+ss+"')";
 //st.executeUpdate(insertdynamic);
 String sqlupdate = "update employee set name='kavya' where eno=4";
 //st.executeUpdate(sqlupdate);
 String sqldelete = "delete from employee where eno=4";
 st.executeUpdate(sqldelete);
 ResultSet rs = st.executeQuery("select * from employee");
while(rs.next())
 System.out.println(rs.getInt(1) + " " + rs.getString(2));
 con.close();
 }
 catch(Exception e)
 {
 System.out.println(e);
 }
 }
}
```

13)

```
import java.io.BufferedReader;
import java.io.FileReader;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.SQLException;

public class BatchUpdateExample {
    public static void main(String[] args) {
        String url = "jdbc:postgresql://localhost/souji";
        String username = "your_username";
        String password = "your_password";
        String filePath = "input_data.txt";
        try (Connection connection = DriverManager.getConnection(url, username, password)) {
            connection.setAutoCommit(false);
            String sql = "INSERT INTO your_table (column1, column2, column3) VALUES (?, ?, ?)";
            PreparedStatement statement = connection.prepareStatement(sql);
            BufferedReader reader = new BufferedReader(new FileReader(filePath));
            String line;
            while ((line = reader.readLine()) != null) {
                String[] data = line.split(",");
                String value1 = data[0].trim();
                String value2 = data[1].trim();
                String value3 = data[2].trim();

                statement.setString(1, value1);
                statement.setString(2, value2);
                statement.setString(3, value3);
                statement.addBatch();
            }
```

```
            reader.close();

            int[] updateCounts = statement.executeBatch();
            connection.commit();

            System.out.println("Batch update executed successfully.");
            System.out.println("Number of rows affected: " + updateCounts.length);
        } catch (SQLException e) {
            System.out.println("Error executing batch update: " + e.getMessage());
        } catch (Exception e) {
            System.out.println("Error: " + e.getMessage());
        }
    }
}
```

14)

Index.html

```
<!DOCTYPE html>
<html>
<head>
    <title>Welcome Form</title>
</head>
<body>
    <form action="WelcomeServlet" method="GET">
        <label for="name">Name:</label>
        <input type="text" id="name" name="name" required>
        <br><br>
        <input type="submit" value="Submit">
    </form>
</body>
</html>
```

WelcomeServlet.java

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.IOException;
import java.io.PrintWriter;

public class WelcomeServlet extends HttpServlet {
   public void doGet(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
      String name = request.getParameter("name");

      response.setContentType("text/html");
      PrintWriter out = response.getWriter();

      out.println("<html>");
      out.println("<head><title>Welcome</title></head>");
      out.println("<body>");
      out.println("<h1>Welcome, " + name + "!</h1>");
      out.println("</body>");
      out.println("</html>");
   }
}
```

15)

Index.html

```
<!DOCTYPE html>
<html>
<head>
   <title>Database Table Form</title>
</head>
<body>
```

```
    <form action="Servlet1" method="GET">
        <input type="submit" value="Retrieve Data">
    </form>
</body>
</html>
```

Servlet1.java

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.IOException;
import java.io.PrintWriter;
import java.sql.*;

public class Servlet1 extends HttpServlet {
    public void doGet(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();

        try {
            Class.forName("com.mysql.jdbc.Driver");
            Connection connection =
DriverManager.getConnection("jdbc:mysql://localhost/souji","root","");

            String query = "SELECT * FROM student";
            Statement statement = connection.createStatement();
            ResultSet resultSet = statement.executeQuery(query);

            out.println("<html>");
            out.println("<head><title>Student Table</title></head>");
            out.println("<body>");
            out.println("<table>");
```

```
        out.println("<tr><th>Column 1</th><th>Column 2</th><th>Column 3</th><th>Column 4</th></tr>");


        while (resultSet.next()) {
            int column1 = resultSet.getInt("sid");
            String column2 = resultSet.getString("sname");
            int column3 = resultSet.getInt("sage");
            String column4 = resultSet.getString("saddress");


            out.println("<tr><td>" + column1 + "</td><td>" + column2 + "</td><td>" + column3 + "</td><td>" + column4 +"</td></tr>");
        }


        out.println("</table>");
        out.println("</body>");
        out.println("</html>");


        resultSet.close();
        statement.close();
        connection.close();
      } catch (ClassNotFoundException e) {
        out.println("Error: MySQL JDBC driver not found.");
      } catch (SQLException e) {
        out.println("Error: " + e.getMessage());
      }
   }
}
```

21)

```
import org.springframework.data.repository.CrudRepository;
public interface StudentRepository extends CrudRepository<Student, Long>
```

```
{
}

@Service
public class StudentService {
private static final Logger log = LoggerFactory.getLogger(StudentService.class);
@Autowired
StudentRepository sr;
@PostConstruct
public void postConstruct() {
Student ob= new Student();
ob.setId(1L);
ob.setName("Kumar");
sr.save(ob);
//retreiving
log.info(“Student:” + sr.findAll());
}
}
```

22)

```
import org.springframework.data.repository.CrudRepository;
public interface StudentRepository extends CrudRepository<Student, Long>
{
}

public class StudentService {
private static final Logger log = LoggerFactory.getLogger(StudentService.class);
@Autowired
StudentRepository sr;
@PostConstruct
public void postConstruct() {
Student ob= new Student();
ob.getId(1L);
ob.getName("Kumar");
sr.save(ob);
//retreiving
log.info(“Student:” + sr.findAll());
}
}
```