1.=====

a.
```
public class DefaultValues {
    static byte byteVar;
    static short shortVar;
    static int intVar;
    static long longVar;
    static float floatVar;
    static double doubleVar;
    static char charVar;
    static boolean booleanVar;

    public static void main(String[] args) {
        System.out.println("Default Values of Primitive Data Types:");
        System.out.println("byte: " + byteVar);
        System.out.println("short: " + shortVar);
        System.out.println("int: " + intVar);
        System.out.println("long: " + longVar);
        System.out.println("float: " + floatVar);
        System.out.println("double: " + doubleVar);
        System.out.println("char: " + charVar); // Will display an empty character
        System.out.println("boolean: " + booleanVar);
    }
}
```

B.
```
import java.util.Scanner;

public class QuadraticEquation {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);

        System.out.print("Enter coefficient a: ");
        double a = sc.nextDouble();
        System.out.print("Enter coefficient b: ");
        double b = sc.nextDouble();
        System.out.print("Enter coefficient c: ");
        double c = sc.nextDouble();

        // Calculate discriminant
        double D = b * b - 4 * a * c;

        if (D > 0) {
            // Two real and distinct roots
            double root1 = (-b + Math.sqrt(D)) / (2 * a);
            double root2 = (-b - Math.sqrt(D)) / (2 * a);
            System.out.println("Roots are real and distinct: " + root1 + " and " + root2);
        } else if (D == 0) {
            // One real root (repeated)
            double root = -b / (2 * a);
            System.out.println("Roots are real and repeated: " + root);
        } else {
            // Complex roots
            double realPart = -b / (2 * a);
            double imaginaryPart = Math.sqrt(-D) / (2 * a);
            System.out.println("Roots are complex: " + realPart + " + " + imaginaryPart + "i and " +
                               realPart + " - " + imaginaryPart + "i");
        }
        sc.close();
    }
}
```

c.

```
import java.util.Scanner;

public class BikerRace {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        double[] speeds = new double[5];
        double sum = 0;

        // Input speeds of 5 racers
        for (int i = 0; i < 5; i++) {
            System.out.print("Enter speed of racer " + (i + 1) + ": ");
            speeds[i] = sc.nextDouble();
            sum += speeds[i];
        }

        // Calculate average speed
        double averageSpeed = sum / 5;
        System.out.println("Average Speed: " + averageSpeed);
        System.out.println("Racers qualifying with speed above average:");

        // Print qualifying racers
        for (int i = 0; i < 5; i++) {
            if (speeds[i] > averageSpeed) {
                System.out.println("Racer " + (i + 1) + " with speed " + speeds[i]);
            }
        }
        sc.close();
    }
}
```

2.====

a.

```
// Base class Shape
class Shape {
    // Method to be overridden by subclasses
    public void draw() {
        System.out.println("Drawing a shape");
    }

    public void erase() {
        System.out.println("Erasing a shape");
    }
}

// Subclass Circle
class Circle extends Shape {
    @Override
    public void draw() {
        System.out.println("Drawing a circle");
    }

    @Override
    public void erase() {
        System.out.println("Erasing a circle");
    }
}

// Subclass Triangle
class Triangle extends Shape {
    @Override
    public void draw() {
        System.out.println("Drawing a triangle");
```

```
    }

    @Override
    public void erase() {
        System.out.println("Erasing a triangle");
    }
}

// Subclass Square
class Square extends Shape {
    @Override
    public void draw() {
        System.out.println("Drawing a square");
    }

    @Override
    public void erase() {
        System.out.println("Erasing a square");
    }
}

// Main class to test the polymorphism
public class Main {
    public static void main(String[] args) {
        // Polymorphism - referring to different objects of subclasses
        Shape shape;

        shape = new Circle();
        shape.draw();
        shape.erase();

        shape = new Triangle();
        shape.draw();
        shape.erase();

        shape = new Square();
        shape.draw();
        shape.erase();
    }
}
```

B.

```
import java.util.Scanner;

class Room {
    // Attributes of the Room class
    int roomNo;
    String roomType;
    float roomArea;
    boolean hasAC;

    // Method to set the data for a room
    public void setData(int roomNo, String roomType, float roomArea, boolean hasAC) {
        this.roomNo = roomNo;
        this.roomType = roomType;
        this.roomArea = roomArea;
        this.hasAC = hasAC;
    }

    // Method to display room data
    public void displayData() {
        System.out.println("Room Number: " + roomNo);
        System.out.println("Room Type: " + roomType);
```

```
        System.out.println("Room Area: " + roomArea + " sq.ft.");
        System.out.println("AC Machine: " + (hasAC ? "Yes" : "No"));
    }
}

public class MainRoom {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        Room room = new Room();

        // Taking input from the user
        System.out.print("Enter Room Number: ");
        int roomNo = sc.nextInt();
        sc.nextLine();  // To consume the newline character

        System.out.print("Enter Room Type: ");
        String roomType = sc.nextLine();

        System.out.print("Enter Room Area (in sq.ft.): ");
        float roomArea = sc.nextFloat();

        System.out.print("Does the room have an AC (true/false): ");
        boolean hasAC = sc.nextBoolean();

        // Setting and displaying room data
        room.setData(roomNo, roomType, roomArea, hasAC);
        room.displayData();

        sc.close();
    }
}
```

3.====

```
A. // Interface A
interface A {
    void meth1();  // abstract method
    void meth2();  // abstract method
}

// Class implementing interface A
class MyClass implements A {
    @Override
    public void meth1() {
        System.out.println("Implementation of meth1");
    }

    @Override
    public void meth2() {
        System.out.println("Implementation of meth2");
    }
}

// Main class to test the implementation
public class MainA {
    public static void main(String[] args) {
        MyClass obj = new MyClass();
        obj.meth1();
        obj.meth2();
    }
}
```

B. // Interface Test
```
interface Test {
    int square(int n);  // abstract method
}

// Arithmetic class implementing Test interface
class Arithmetic implements Test {
    @Override
    public int square(int n) {
        return n * n;
    }
}

// Class ToTestInt to use Arithmetic class
class ToTestInt {
    public static void main(String[] args) {
        // Creating object of Arithmetic class
        Arithmetic arithmetic = new Arithmetic();

        // Testing the square method
        int number = 5;
        int result = arithmetic.square(number);
        System.out.println("The square of " + number + " is: " + result);
    }
}
```

C. // Outer class
```
class Outer {
    // Outer class display method
    public void display() {
        System.out.println("Display method of Outer class");
    }

    // Inner class
    class Inner {
        // Inner class display method
        public void display() {
            System.out.println("Display method of Inner class");
        }
    }
}

// Main class to test the Outer and Inner class
public class MainOuterInner {
    public static void main(String[] args) {
        // Creating object of Outer class
        Outer outer = new Outer();
        outer.display();  // Calling outer class display method

        // Creating object of Inner class
        Outer.Inner inner = outer.new Inner();
        inner.display();  // Calling inner class display method
    }
}
```

4.=====

A. // Thread that prints "Good Morning" every 1 second
```
class GoodMorningThread extends Thread {
    @Override
    public void run() {
        try {
            while (true) {
                System.out.println("Good Morning");
```

```
            Thread.sleep(1000);  // Sleep for 1 second
        }
    } catch (InterruptedException e) {
        System.out.println(e);
    }
  }
}

// Thread that prints "Hello" every 2 seconds
class HelloThread extends Thread {
  @Override
  public void run() {
    try {
      while (true) {
        System.out.println("Hello");
        Thread.sleep(2000);  // Sleep for 2 seconds
      }
    } catch (InterruptedException e) {
      System.out.println(e);
    }
  }
}

// Thread that prints "Welcome" every 3 seconds
class WelcomeThread extends Thread {
  @Override
  public void run() {
    try {
      while (true) {
        System.out.println("Welcome");
        Thread.sleep(3000);  // Sleep for 3 seconds
      }
    } catch (InterruptedException e) {
      System.out.println(e);
    }
  }
}

public class ThreadExample {
  public static void main(String[] args) {
    GoodMorningThread t1 = new GoodMorningThread();
    HelloThread t2 = new HelloThread();
    WelcomeThread t3 = new WelcomeThread();

    t1.start();
    t2.start();
    t3.start();
  }
}

Aa. // Runnable for "Good Morning" every 1 second
class GoodMorningRunnable implements Runnable {
  @Override
  public void run() {
    try {
      while (true) {
        System.out.println("Good Morning");
        Thread.sleep(1000);  // Sleep for 1 second
      }
    } catch (InterruptedException e) {
      System.out.println(e);
    }
```

```
    }
}

// Runnable for "Hello" every 2 seconds
class HelloRunnable implements Runnable {
    @Override
    public void run() {
        try {
            while (true) {
                System.out.println("Hello");
                Thread.sleep(2000);  // Sleep for 2 seconds
            }
        } catch (InterruptedException e) {
            System.out.println(e);
        }
    }
}

// Runnable for "Welcome" every 3 seconds
class WelcomeRunnable implements Runnable {
    @Override
    public void run() {
        try {
            while (true) {
                System.out.println("Welcome");
                Thread.sleep(3000);  // Sleep for 3 seconds
            }
        } catch (InterruptedException e) {
            System.out.println(e);
        }
    }
}

public class RunnableExample {
    public static void main(String[] args) {
        Thread t1 = new Thread(new GoodMorningRunnable());
        Thread t2 = new Thread(new HelloRunnable());
        Thread t3 = new Thread(new WelcomeRunnable());

        t1.start();
        t2.start();
        t3.start();
    }
}

b. class MyThread extends Thread {
    @Override
    public void run() {
        try {
            for (int i = 0; i < 5; i++) {
                System.out.println(Thread.currentThread().getName() + " is running");
                Thread.sleep(1000);  // Sleep for 1 second
            }
        } catch (InterruptedException e) {
            System.out.println(e);
        }
    }
}

public class ThreadAliveJoinExample {
    public static void main(String[] args) {
        MyThread t1 = new MyThread();
```

```
        MyThread t2 = new MyThread();

        // Start the threads
        t1.start();
        t2.start();

        // Check if t1 and t2 are alive
        System.out.println("Is t1 alive? " + t1.isAlive());
        System.out.println("Is t2 alive? " + t2.isAlive());

        try {
            // Use join to wait for t1 to finish before proceeding
            t1.join();
            System.out.println("t1 has finished");

            // Use join to wait for t2 to finish before proceeding
            t2.join();
            System.out.println("t2 has finished");
        } catch (InterruptedException e) {
            System.out.println(e);
        }

        // Check if t1 and t2 are alive after they finish
        System.out.println("Is t1 alive after join? " + t1.isAlive());
        System.out.println("Is t2 alive after join? " + t2.isAlive());
    }
}
```

5.===

a. File 1: `Emp.java` (Create a package `employee` and class `Emp`)

```
// Package declaration
package employee;

public class Emp {
    // Employee attributes
    String name;
    int empid;
    String category;
    double bpay, hra, da, pf, grosspay, incometax, allowance, npay;

    // Method to set employee data
    public void setEmployeeData(String name, int empid, String category, double bpay) {
        this.name = name;
        this.empid = empid;
        this.category = category;
        this.bpay = bpay;
    }

    // Method to calculate payroll components
    public void calculatePay() {
        hra = 0.10 * bpay; // HRA 10% of basic pay
        da = 0.05 * bpay;  // DA 5% of basic pay
        pf = 0.12 * bpay;  // PF 12% of basic pay
        allowance = 1000;  // Fixed allowance
        grosspay = bpay + hra + da + allowance;
        incometax = 0.10 * grosspay; // Income tax 10% of grosspay
        npay = grosspay - (pf + incometax); // Net Pay
    }

    // Method to display employee payroll details
    public void displayPayroll() {
```

```
        System.out.println("Employee ID: " + empid);
        System.out.println("Name: " + name);
        System.out.println("Category: " + category);
        System.out.println("Basic Pay: " + bpay);
        System.out.println("HRA: " + hra);
        System.out.println("DA: " + da);
        System.out.println("Allowance: " + allowance);
        System.out.println("Gross Pay: " + grosspay);
        System.out.println("Income Tax: " + incometax);
        System.out.println("PF: " + pf);
        System.out.println("Net Pay: " + npay);
    }
}
```

File 2: `Emppay.java` (Call the methods to perform calculations and print values)

```
// Importing the employee package
import employee.Emp;

public class Emppay {
    public static void main(String[] args) {
        // Creating an object of class Emp
        Emp e = new Emp();

        // Setting employee data (Example data)
        e.setEmployeeData("John Doe", 101, "Manager", 50000);

        // Calculating pay details
        e.calculatePay();

        // Displaying payroll details
        e.displayPayroll();
    }
}
```

b. File 1: Student.java (Create a package MCA and class Student)

```
// Package declaration
package MCA;

public class Student {
    // Student attributes
    String name;
    int rollNo;
    int[] marks = new int[5]; // Marks for 5 subjects

    // Parameterized constructor to accept student details
    public Student(String name, int rollNo, int[] marks) {
        this.name = name;
        this.rollNo = rollNo;
        for (int i = 0; i < marks.length; i++) {
            this.marks[i] = marks[i];
        }
    }

    // Method to calculate total marks
    public int calculateTotal() {
        int total = 0;
        for (int mark : marks) {
            total += mark;
        }
        return total;
    }

    // Method to calculate percentage
    public double calculatePercentage() {
        int total = calculateTotal();
```

```
        return (double) total / marks.length;
    }

    // Method to display student details
    public void display() {
        System.out.println("Roll Number: " + rollNo);
        System.out.println("Name: " + name);
        System.out.println("Marks: ");
        for (int i = 0; i < marks.length; i++) {
            System.out.println("Subject " + (i + 1) + ": " + marks[i]);
        }
        System.out.println("Total Marks: " + calculateTotal());
        System.out.println("Percentage: " + calculatePercentage() + "%");
    }
}
```

File 2: `MainStudent.java` (Call the methods to calculate total marks and percentage)

```
// Importing the MCA package
import MCA.Student;

public class MainStudent {
    public static void main(String[] args) {
        // Example student details
        String name = "Alice";
        int rollNo = 101;
        int[] marks = {85, 78, 92, 88, 90}; // Marks for 5 subjects

        // Creating an object of the Student class
        Student student = new Student(name, rollNo, marks);

        // Displaying student details
        student.display();
    }
}
```

6.===

A.

```
class LowPriorityThread extends Thread {
    @Override
    public void run() {
        for (int i = 1; i <= 5; i++) {
            System.out.println("Low Priority Thread: " + i);
            try {
                Thread.sleep(1000);  // Sleep to simulate work
            } catch (InterruptedException e) {
                System.out.println(e);
            }
        }
        System.out.println("Low Priority Thread completed.");
    }
}

class HighPriorityThread extends Thread {
    @Override
    public void run() {
        for (int i = 1; i <= 5; i++) {
            System.out.println("High Priority Thread: " + i);
            try {
                Thread.sleep(500);  // Sleep to simulate work
```

```
            } catch (InterruptedException e) {
                System.out.println(e);
            }
        }
        System.out.println("High Priority Thread completed.");
    }
}

public class ThreadPriorityExample {
    public static void main(String[] args) {
        // Create threads
        LowPriorityThread t1 = new LowPriorityThread();
        HighPriorityThread t2 = new HighPriorityThread();

        // Set thread priorities
        t1.setPriority(Thread.MIN_PRIORITY); // Lower priority
        t2.setPriority(Thread.MAX_PRIORITY); // Higher priority

        // Start threads
        t1.start();  // Low-priority thread starts first
        t2.start();  // High-priority thread starts next

        try {
            t1.join();  // Wait for low-priority thread to finish
            t2.join();  // Wait for high-priority thread to finish
        } catch (InterruptedException e) {
            System.out.println(e);
        }

        System.out.println("Main Thread exiting.");
    }
}
B.
class Queue {
    int item;
    boolean available = false;

    // Producer adds an item to the queue
    synchronized void produce(int value) {
        while (available) {
            try {
                wait(); // Wait until the item is consumed
            } catch (InterruptedException e) {
                System.out.println(e);
            }
        }
        item = value;
        available = true;
        System.out.println("Produced: " + item);
        notify(); // Notify consumer to consume the item
    }

    // Consumer consumes an item from the queue
    synchronized int consume() {
        while (!available) {
            try {
                wait(); // Wait until the item is produced
```

```
            } catch (InterruptedException e) {
                System.out.println(e);
            }
        }
        available = false;
        System.out.println("Consumed: " + item);
        notify(); // Notify producer to produce next item
        return item;
    }
}

// Producer class
class Producer extends Thread {
    Queue queue;

    Producer(Queue queue) {
        this.queue = queue;
    }

    @Override
    public void run() {
        int i = 0;
        while (true) {
            queue.produce(i++); // Produce items continuously
            try {
                Thread.sleep(1000); // Simulate production time
            } catch (InterruptedException e) {
                System.out.println(e);
            }
        }
    }
}

// Consumer class
class Consumer extends Thread {
    Queue queue;

    Consumer(Queue queue) {
        this.queue = queue;
    }

    @Override
    public void run() {
        while (true) {
            queue.consume(); // Consume items continuously
            try {
                Thread.sleep(1500); // Simulate consumption time
            } catch (InterruptedException e) {
                System.out.println(e);
            }
        }
    }
}

public class ProducerConsumerExample {
    public static void main(String[] args) {
        // Shared object between producer and consumer
```

```
        Queue queue = new Queue();

        // Creating Producer and Consumer threads
        Producer producer = new Producer(queue);
        Consumer consumer = new Consumer(queue);

        // Start the producer and consumer threads
        producer.start();
        consumer.start();
    }
}
```

7. ===

a.
```
import java.io.File;
import java.io.IOException;

public class FileCheck {
    public static void main(String[] args) {
        // Define the file path
        String filePath = "C:\\java\\abc.txt";
        File file = new File(filePath);

        // Create the file
        try {
            if (file.createNewFile()) {
                System.out.println("File created: " + file.getAbsolutePath());
            } else {
                System.out.println("File already exists: " + file.getAbsolutePath());
            }
        } catch (IOException e) {
            System.out.println("An error occurred while creating the file.");
            e.printStackTrace();
        }

        // Check if the file exists
        if (file.exists()) {
            System.out.println("File exists: " + file.getName());
            System.out.println("Is it a file? " + file.isFile());
            System.out.println("Is it a directory? " + file.isDirectory());
            System.out.println("Absolute path: " + file.getAbsolutePath());
        } else {
            System.out.println("File does not exist.");
        }
    }
}
```

B.
```
import java.io.File;

public class RenameAndDeleteFile {
    public static void main(String[] args) {
        // Check if file name is provided as command line argument
        if (args.length < 2) {
            System.out.println("Please provide the old file name and the new file name.");
            return;
        }

        String oldFileName = args[0]; // Original file name
```

```
        String newFileName = args[1]; // New file name

        // Create file objects
        File oldFile = new File(oldFileName);
        File newFile = new File(newFileName);

        // Rename the file
        if (oldFile.renameTo(newFile)) {
            System.out.println("File renamed successfully to: " + newFile.getAbsolutePath());

            // Delete the renamed file
            if (newFile.delete()) {
                System.out.println("File deleted successfully.");
            } else {
                System.out.println("Failed to delete the file.");
            }
        } else {
            System.out.println("Failed to rename the file.");
        }
    }
}
```

== java RenameAndDeleteFile C:\java\abc.txt C:\java\abc_renamed.txt

C.
```
import java.io.File;

public class CreateDirectory {
    public static void main(String[] args) {
        // Define the directory path
        String dirPath = "C:\\java\\newDirectory";
        File directory = new File(dirPath);

        // Create the directory
        if (directory.mkdir()) {
            System.out.println("Directory created: " + directory.getAbsolutePath());
        } else {
            System.out.println("Directory already exists or failed to create.");
        }

        // Check if the directory exists
        if (directory.exists() && directory.isDirectory()) {
            System.out.println("Directory exists: " + directory.getName());
            System.out.println("Absolute path: " + directory.getAbsolutePath());
        } else {
            System.out.println("Directory does not exist.");
        }
    }
}
```

8.===

A.
```
import java.net.InetAddress;
import java.net.UnknownHostException;
import java.util.Scanner;

public class HostnameToIPAddress {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter a host name: ");
```

```
        String hostname = scanner.nextLine();

        try {
            // Convert hostname to IP address
            InetAddress ipAddress = InetAddress.getByName(hostname);
            System.out.println("IP Address of " + hostname + " is: " + ipAddress.getHostAddress());
        } catch (UnknownHostException e) {
            System.out.println("Host not found: " + hostname);
        }

        scanner.close();
    }
}
```

B. Server:

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.PrintWriter;
import java.net.ServerSocket;
import java.net.Socket;

public class UppercaseServer {
    public static void main(String[] args) {
        int port = 12345; // Server will listen on this port

        try (ServerSocket serverSocket = new ServerSocket(port)) {
            System.out.println("Server is listening on port " + port);

            while (true) {
                Socket clientSocket = serverSocket.accept(); // Accept a client connection
                System.out.println("Client connected: " + clientSocket.getInetAddress());

                // Create input and output streams
                BufferedReader input = new BufferedReader(new InputStreamReader(clientSocket.getInputStream()));
                PrintWriter output = new PrintWriter(clientSocket.getOutputStream(), true);

                String message = input.readLine(); // Read message from client
                System.out.println("Received from client: " + message);

                // Convert the message to uppercase and send it back
                String response = message.toUpperCase();
                output.println(response);
                System.out.println("Sent to client: " + response);

                // Close the client connection
                clientSocket.close();
            }
        } catch (IOException e) {
            System.out.println("Server error: " + e.getMessage());
        }
    }
}
```

client:

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
```

```
import java.io.PrintWriter;
import java.net.Socket;
import java.util.Scanner;

public class UppercaseClient {
    public static void main(String[] args) {
        String serverAddress = "localhost"; // Server address (or IP)
        int port = 12345; // Server port

        try (Socket socket = new Socket(serverAddress, port)) {
            System.out.println("Connected to server");

            // Create input and output streams
            PrintWriter output = new PrintWriter(socket.getOutputStream(), true);
            BufferedReader input = new BufferedReader(new InputStreamReader(socket.getInputStream()));
            Scanner scanner = new Scanner(System.in);

            System.out.print("Enter a message to send to the server: ");
            String message = scanner.nextLine();

            // Send message to the server
            output.println(message);
            System.out.println("Sent to server: " + message);

            // Read the response from the server
            String response = input.readLine();
            System.out.println("Response from server: " + response);

            scanner.close();
        } catch (IOException e) {
            System.out.println("Client error: " + e.getMessage());
        }
    }
}
```

9.===

a.
```
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.util.Scanner;

// Base class for Course
abstract class Course {
    protected String name;
    protected double fee;

    public Course(String name, double fee) {
        this.name = name;
        this.fee = fee;
    }

    public String getName() {
        return name;
    }

    public double getFee() {
```

```
        return fee;
    }

    public abstract String getCourseType(); // Method to get course type
}

// Class for Classroom Course
class ClassroomCourse extends Course {
    public ClassroomCourse(String name, double fee) {
        super(name, fee);
    }

    @Override
    public String getCourseType() {
        return "Classroom Delivered";
    }
}

// Class for Online Course
class OnlineCourse extends Course {
    public OnlineCourse(String name, double fee) {
        super(name, fee);
    }

    @Override
    public String getCourseType() {
        return "Online Delivered";
    }
}

// Employee class
class Employee {
    private String name;
    private List<Course> enrolledCourses;

    public Employee(String name) {
        this.name = name;
        this.enrolledCourses = new ArrayList<>();
    }

    public String getName() {
        return name;
    }

    public void enrollCourse(Course course) {
        enrolledCourses.add(course);
    }

    public List<Course> getEnrolledCourses() {
        return enrolledCourses;
    }
}

// Course Coordinator class
class CourseCoordinator {
    private Map<String, Course> courses;
    private List<Employee> employees;
```

```
    public CourseCoordinator() {
        courses = new HashMap<>();
        employees = new ArrayList<>();
    }

    public void addCourse(Course course) {
        courses.put(course.getName(), course);
    }

    public void registerEmployee(Employee employee, String courseName) {
        Course course = courses.get(courseName);
        if (course != null) {
            employee.enrollCourse(course);
            System.out.println("Employee " + employee.getName() + " registered for " + courseName);
        } else {
            System.out.println("Course not found!");
        }
    }

    public void listRegisteredEmployees(String courseName) {
        System.out.println("Employees registered for " + courseName + ":");
        for (Employee employee : employees) {
            for (Course course : employee.getEnrolledCourses()) {
                if (course.getName().equals(courseName)) {
                    System.out.println(employee.getName());
                }
            }
        }
    }

    public void addEmployee(Employee employee) {
        employees.add(employee);
    }
}

public class CourseRegistrationSystem {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        CourseCoordinator coordinator = new CourseCoordinator();

        // Adding some courses
        coordinator.addCourse(new ClassroomCourse("Java Programming", 500.0));
        coordinator.addCourse(new OnlineCourse("Web Development", 300.0));

        // Creating employees
        Employee emp1 = new Employee("John Doe");
        Employee emp2 = new Employee("Jane Smith");
        coordinator.addEmployee(emp1);
        coordinator.addEmployee(emp2);

        // Registering employees for courses
        coordinator.registerEmployee(emp1, "Java Programming");
        coordinator.registerEmployee(emp2, "Web Development");

        // Listing employees registered for a specific course
        System.out.print("Enter course name to list registered employees: ");
```

```
        String courseName = scanner.nextLine();
        coordinator.listRegisteredEmployees(courseName);

        scanner.close();
    }
}

b. abstract class Worker {
    protected String name;
    protected double salaryRate;

    public Worker(String name, double salaryRate) {
        this.name = name;
        this.salaryRate = salaryRate;
    }

    public String getName() {
        return name;
    }

    public abstract double pay(int hours); // Abstract method to compute pay
}

// Class for DailyWorker
class DailyWorker extends Worker {
    public DailyWorker(String name, double dailyRate) {
        super(name, dailyRate);
    }

    @Override
    public double pay(int hours) {
        return salaryRate * hours; // Pay based on the number of days worked
    }
}

// Class for SalariedWorker
class SalariedWorker extends Worker {
    public SalariedWorker(String name, double salary) {
        super(name, salary);
    }

    @Override
    public double pay(int hours) {
        return salaryRate; // Pay is fixed for salaried workers
    }
}

public class WorkerPaymentSystem {
    public static void main(String[] args) {
        // Creating worker instances
        Worker dailyWorker = new DailyWorker("Alice", 100.0); // Daily rate
        Worker salariedWorker = new SalariedWorker("Bob", 800.0); // Monthly salary

        // Calculating pay for a week
        int dailyHours = 5; // Assume the daily worker works for 5 days
        System.out.println(dailyWorker.getName() + "'s weekly pay: $" + dailyWorker.pay(dailyHours));
```

```
        int salariedHours = 40; // Fixed hours for salaried worker
        System.out.println(salariedWorker.getName() + "'s weekly pay: $" + salariedWorker.pay(salariedHours));
    }
}
```

10===

a.
```
import java.applet.Applet;
import java.awt.*;
import java.awt.event.*;

public class PaintBrushApplet extends Applet implements MouseMotionListener {
    private Image image;
    private Graphics gImage;
    private int lastX, lastY;

    public void init() {
        setSize(800, 600);
        image = createImage(800, 600);
        gImage = image.getGraphics();
        gImage.setColor(Color.WHITE);
        gImage.fillRect(0, 0, 800, 600);
        addMouseMotionListener(this);
    }

    public void mouseDragged(MouseEvent e) {
        int x = e.getX();
        int y = e.getY();
        gImage.setColor(Color.BLACK);
        gImage.drawLine(lastX, lastY, x, y);
        lastX = x;
        lastY = y;
        repaint();
    }

    public void mouseMoved(MouseEvent e) {
        lastX = e.getX();
        lastY = e.getY();
    }

    public void paint(Graphics g) {
        g.drawImage(image, 0, 0, this);
    }
}
```

b.
```
import java.applet.Applet;
import java.awt.*;
import java.util.Calendar;

public class AnalogClockApplet extends Applet implements Runnable {
    private Thread thread;

    public void init() {
        setSize(400, 400);
        thread = new Thread(this);
        thread.start();
    }
```

```
    public void run() {
        while (true) {
            repaint();
            try {
                Thread.sleep(1000); // Refresh every second
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }

    public void paint(Graphics g) {
        Calendar calendar = Calendar.getInstance();
        int hour = calendar.get(Calendar.HOUR);
        int minute = calendar.get(Calendar.MINUTE);
        int second = calendar.get(Calendar.SECOND);

        int radius = 150;

        // Draw clock face
        g.setColor(Color.LIGHT_GRAY);
        g.fillOval(50, 50, 300, 300);
        g.setColor(Color.BLACK);
        g.drawOval(50, 50, 300, 300);

        // Draw hour hand
        double hourAngle = Math.toRadians((hour % 12 + minute / 60.0) * 30);
        int hourX = (int) (200 + radius * 0.5 * Math.cos(hourAngle - Math.PI / 2));
        int hourY = (int) (200 + radius * 0.5 * Math.sin(hourAngle - Math.PI / 2));
        g.drawLine(200, 200, hourX, hourY);

        // Draw minute hand
        double minuteAngle = Math.toRadians(minute * 6);
        int minuteX = (int) (200 + radius * 0.7 * Math.cos(minuteAngle - Math.PI / 2));
        int minuteY = (int) (200 + radius * 0.7 * Math.sin(minuteAngle - Math.PI / 2));
        g.drawLine(200, 200, minuteX, minuteY);

        // Draw second hand
        g.setColor(Color.RED);
        double secondAngle = Math.toRadians(second * 6);
        int secondX = (int) (200 + radius * 0.8 * Math.cos(secondAngle - Math.PI / 2));
        int secondY = (int) (200 + radius * 0.8 * Math.sin(secondAngle - Math.PI / 2));
        g.drawLine(200, 200, secondX, secondY);
    }
}
```

c.
```
import java.applet.Applet;
import java.awt.*;

public class ShapesApplet extends Applet {
    public void paint(Graphics g) {
        // Set color and draw a rectangle
        g.setColor(Color.BLUE);
        g.fillRect(50, 50, 100, 50); // Filled Rectangle

        // Set color and draw a circle
        g.setColor(Color.RED);
```

```
        g.fillOval(200, 50, 100, 100); // Filled Circle

        // Set color and draw a triangle
        g.setColor(Color.GREEN);
        int[] xPoints = {350, 400, 300};
        int[] yPoints = {50, 150, 150};
        g.fillPolygon(xPoints, yPoints, 3); // Filled Triangle

        // Set color and draw an arc
        g.setColor(Color.ORANGE);
        g.fillArc(50, 200, 200, 100, 0, 180); // Filled Arc
    }
}
```

== appletviewer PaintBrushApplet.java

11.===

A.
```
import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

public class Calculator extends JFrame implements ActionListener {
    private JTextField textField;
    private String operator;
    private double num1, num2;

    public Calculator() {
        // Frame setup
        setTitle("Calculator");
        setSize(400, 600);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setLayout(new BorderLayout());

        // Text field
        textField = new JTextField();
        textField.setEditable(false);
        add(textField, BorderLayout.NORTH);

        // Panel for buttons
        JPanel panel = new JPanel();
        panel.setLayout(new GridLayout(4, 4));

        // Button setup
        String[] buttons = {
            "7", "8", "9", "/",
            "4", "5", "6", "*",
            "1", "2", "3", "-",
            "C", "0", "=", "+"
        };

        for (String text : buttons) {
            JButton button = new JButton(text);
            button.addActionListener(this);
            panel.add(button);
        }
```

```
        add(panel, BorderLayout.CENTER);
        setVisible(true);
    }

    public void actionPerformed(ActionEvent e) {
        String command = e.getActionCommand();

        switch (command) {
            case "C":
                textField.setText("");
                break;
            case "=":
                num2 = Double.parseDouble(textField.getText());
                switch (operator) {
                    case "+":
                        textField.setText(String.valueOf(num1 + num2));
                        break;
                    case "-":
                        textField.setText(String.valueOf(num1 - num2));
                        break;
                    case "*":
                        textField.setText(String.valueOf(num1 * num2));
                        break;
                    case "/":
                        textField.setText(String.valueOf(num1 / num2));
                        break;
                }
                break;
            default:
                if (operator == null) {
                    num1 = Double.parseDouble(textField.getText());
                    operator = command;
                    textField.setText("");
                } else {
                    textField.setText(textField.getText() + command);
                }
                break;
        }
    }

    public static void main(String[] args) {
        new Calculator();
    }
}

b. import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.util.Calendar;

public class DigitalWatch extends JFrame implements ActionListener {
    private JLabel timeLabel;
    private Timer timer;

    public DigitalWatch() {
```

```
        // Frame setup
        setTitle("Digital Watch");
        setSize(300, 200);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setLayout(new FlowLayout());

        // Time label
        timeLabel = new JLabel();
        timeLabel.setFont(new Font("Arial", Font.BOLD, 48));
        add(timeLabel);

        // Timer setup
        timer = new Timer(1000, this);
        timer.start();

        setVisible(true);
    }

    public void actionPerformed(ActionEvent e) {
        // Get the current time
        Calendar calendar = Calendar.getInstance();
        int hour = calendar.get(Calendar.HOUR_OF_DAY);
        int minute = calendar.get(Calendar.MINUTE);
        int second = calendar.get(Calendar.SECOND);

        // Update the label
        timeLabel.setText(String.format("%02d:%02d:%02d", hour, minute, second));
    }

    public static void main(String[] args) {
        new DigitalWatch();
    }
}
```