Lab 03: Test Plan for 6 Programs

Suproteek Banerjee
CWID: *A20637525*

6. Problem Description (*ShippingCharges*)

The goal of this program is to calculate the total shipping charges for packages based on their weight and distance shipped.

The Fast Freight Shipping Company charges per 500 miles (not prorated), using the following rates:

Weight of Package (kg)	**Rate per 500 miles**
2 kg or less	\$1.10
$>$ 2 kg but $\leq$ 6 kg	\$2.20
$>$ 6 kg but $\leq$ 10 kg	\$3.70
$>$ 10 kg	\$4.80

If the distance is not a multiple of 500 miles, it is rounded up to the next 500.

Inputs

- **Weight:** Positive real number representing the package weight in kilograms.
- **Distance:** Positive integer representing miles shipped.

Processes / Operations

- Determine the number of 500-mile units:

$$\text{units} = \lceil \tfrac{\text{distance}}{500} \rceil$$

- Determine the rate based on weight range.
- Multiply the units by the rate to calculate total shipping charges.

Outputs

The program produces:

- The total shipping cost (in USD), based on package weight and distance.
- Example: For a 2kg package shipped 550 miles, the cost is

$$2 \times 1.10 = 2.20$$

Code Listing

```java
public class ShippingCharges {

    /*-------------------------------------------------------
      Instance Variable
    -------------------------------------------------------*/
    private double weight; // Weight of package in kilograms

    /*-------------------------------------------------------
      Constructor
    -------------------------------------------------------*/
    public ShippingCharges(double weight) {
        this.weight = weight;
    }

    /*-------------------------------------------------------
      Method: calculateCharges
      Purpose: Calculates total shipping charges based on weight
               and distance (rounded up to nearest 500 miles).
    -------------------------------------------------------*/
    public double calculateCharges(int distance) {
        // Round up miles to nearest 500
        int units = (int) Math.ceil(distance / 500.0);

        double ratePer500;

        if (weight <= 2) {
            ratePer500 = 1.10;
        } else if (weight <= 6) {
            ratePer500 = 2.20;
        } else if (weight <= 10) {
            ratePer500 = 3.70;
        } else {
            ratePer500 = 4.80;
        }

        return units * ratePer500;
    }

    /*-------------------------------------------------------
      Getters and Setters
    -------------------------------------------------------*/
    public double getWeight() {
        return weight;
    }

    public void setWeight(double weight) {
        this.weight = weight;
    }

    /*-------------------------------------------------------
      Main Method: For Testing
    -------------------------------------------------------*/
    public static void main(String[] args) {
        System.out.println(" ");
        System.out.println("==========================================");
        System.out.println("   Fast Freight Shipping Cost Calculator");
```

```
57          System.out.println("==========================================");
58          System.out.println(" ");
59          ShippingCharges pkg1 = new ShippingCharges(2);   // 2kg package
60          System.out.println("Package 1 (2 kg, 550 miles): $" + pkg1.
                calculateCharges(550));
61
62
63          ShippingCharges pkg2 = new ShippingCharges(8);   // 8kg package
64          System.out.println("Package 2 (8 kg, 1200 miles): $" + pkg2.
                calculateCharges(1200));
65
66
67          System.out.println(" ");
68          System.out.println("==========================================");
69          System.out.println("   End of Shipping Charges Calculation");
70          System.out.println("==========================================");
71      }
72  }
```

Listing 1: Java Program: ShippingCharges.java

Test Plan

Test Case	**Sample Data (Weight, Distance)**	**Expected Output (Charges)**
Lower boundary weight	(2 kg, 500 miles)	$1 \times 1.10 = 1.10$
Over boundary (requires rounding)	(2 kg, 550 miles)	$2 \times 1.10 = 2.20$
Medium weight, multiple of 500	(5 kg, 1000 miles)	$2 \times 2.20 = 4.40$
Higher weight, rounded distance	(8 kg, 1200 miles)	$3 \times 3.70 = 11.10$
Maximum weight test	(12 kg, 499 miles)	$1 \times 4.80 = 4.80$
Another boundary case	(10 kg, 1501 miles)	$4 \times 3.70 = 14.80$

7. Problem Description (*FatGram*)

The goal of this program is to calculate the percentage of calories in a food item that come from fat, based on user input.

- Each gram of fat contributes 9 calories.
- The percentage of calories from fat is calculated as:

$$\text{Percentage} = \frac{\text{Calories from Fat}}{\text{Total Calories}} \times 100$$

- If the calories from fat exceed the total calories, the program displays an error message.
- If the calories from fat are less than 30% of the total, the program indicates that the food is **low in fat**.

Inputs

- **Calories:** Total calories in the food item (positive double).
- **Fat Grams:** Number of fat grams in the food item (positive double).

Processes / Operations

1. Multiply fat grams by 9 to compute **Calories from Fat**.
2. Validate: Ensure **Calories from Fat** $\leq$ **Total Calories**.
3. Compute percentage of calories from fat:

$$\frac{\text{Calories from Fat}}{\text{Total Calories}} \times 100$$

4. If percentage < 30, display **Low Fat message**.

Outputs

The program produces:

- A formatted analysis report showing:
  - Total Calories
  - Calories from Fat
  - Fat Percentage
- An indication of whether the food is **low fat**.
- An error message if calories from fat exceed total calories.

Code Listing

```java
import java.util.Scanner;

public class FatGram {
    private double calories;
    private double fatGrams;

    public FatGram(double calories, double fatGrams) {
        this.calories = calories;
        this.fatGrams = fatGrams;
    }

    public double getCaloriesFromFat() {
        return fatGrams * 9;
    }

    public double getFatPercentage() {
        if (calories <= 0) {
            return 0;
        }
        return (getCaloriesFromFat() / calories) * 100;
    }

    public boolean isValid() {
        return getCaloriesFromFat() <= calories;
    }

    public boolean isLowFat() {
        return getFatPercentage() < 30;
    }

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

        System.out.println("==========================================");
        System.out.println("         FatGram - Nutrition Analyzer        ");
        System.out.println("==========================================");

        System.out.print("Enter the number of calories: ");
        double calories = input.nextDouble();

        System.out.print("Enter the number of fat grams: ");
        double fatGrams = input.nextDouble();

        FatGram food = new FatGram(calories, fatGrams);

        System.out.println("\n------------------------------------------");
        System.out.println("               Analysis Report               ");
        System.out.println("------------------------------------------");

        if (!food.isValid()) {
            System.out.println("Error: Calories from fat cannot exceed total
                calories.");
        } else {
            System.out.printf("Total Calories: %.2f cal\n", calories);
            System.out.printf("Fat Calories : %.2f cal\n", food.getCaloriesFromFat
                ());
```

```
            System.out.printf("Fat Percentage: %.2f%%\n", food.getFatPercentage()
                ;

            if (food.isLowFat()) {
                System.out.println("This food is low in fat.");
            } else {
                System.out.println("This food has a moderate/high fat content.");
            }
        }

        System.out.println("-----------------------------------------");
        System.out.println("         End of Nutrition Analysis        ");
        System.out.println("=========================================");

        input.close();
    }
}
```

Listing 2: Java Program: FatGram.java

Test Plan

Test Case	**Sample Data (Calories, Fat Grams)**	**Expected Output**
Valid Low Fat	(500, 10)	Calories from Fat = 90, Fat % = 18%, Low Fat message displayed
Valid High Fat	(400, 20)	Calories from Fat = 180, Fat % = 45%, High fat message displayed
Invalid Input	(200, 30)	Error: Calories from fat cannot exceed total calories
Boundary Case (exact 30%)	(300, 10)	Calories from Fat = 90, Fat % = 30.0%, Not low fat
Zero Fat	(250, 0)	Calories from Fat = 0, Fat % = 0%, Low Fat message displayed

8. Problem Description (*RunningRace*)

The goal of this program is to determine the finishing order of three runners in a race based on their completion times. The program stores each runner's name and race time (in minutes) and then outputs who placed 1st, 2nd, and 3rd.

Inputs

- **Name of Runner 1, 2, 3:** Strings representing runner names.
- **Time of Runner 1, 2, 3:** Positive double values representing the time (in minutes) taken to complete the race.

Processes / Operations

1. Compare the three times.
2. Determine:
    - The smallest time → **1st place**.
    - The median time → **2nd place**.
    - The largest time → **3rd place**.

Outputs

The program displays:

- The names of the 1st, 2nd, and 3rd place runners.
- A formatted race results report.

Code Listing

```
import java.util.Scanner;

public class RunningRace {

    private String runner1, runner2, runner3;
    private double time1, time2, time3;

    public RunningRace(String runner1, double time1,
                       String runner2, double time2,
                       String runner3, double time3) {
        this.runner1 = runner1;
        this.time1 = time1;
        this.runner2 = runner2;
        this.time2 = time2;
        this.runner3 = runner3;
        this.time3 = time3;
    }

    public String getFirstPlace() {
        if (time1 < time2 && time1 < time3) return runner1;
        else if (time2 < time1 && time2 < time3) return runner2;
```

```
        else return runner3;
    }

    public String getSecondPlace() {
        if ((time1 > time2 && time1 < time3) || (time1 < time2 && time1 > time3))
            return runner1;
        else if ((time2 > time1 && time2 < time3) || (time2 < time1 && time2 >
            time3))
            return runner2;
        else return runner3;
    }

    public String getThirdPlace() {
        if (time1 > time2 && time1 > time3) return runner1;
        else if (time2 > time1 && time2 > time3) return runner2;
        else return runner3;
    }

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

        System.out.println("=============================================");
        System.out.println("         Running the Race - Analyzer         ");
        System.out.println("=============================================");

        System.out.print("Enter name of runner 1: ");
        String r1 = input.nextLine();
        System.out.print("Enter time (in minutes) for " + r1 + ": ");
        double t1 = input.nextDouble();
        input.nextLine();

        System.out.print("Enter name of runner 2: ");
        String r2 = input.nextLine();
        System.out.print("Enter time (in minutes) for " + r2 + ": ");
        double t2 = input.nextDouble();
        input.nextLine();

        System.out.print("Enter name of runner 3: ");
        String r3 = input.nextLine();
        System.out.print("Enter time (in minutes) for " + r3 + ": ");
        double t3 = input.nextDouble();

        RunningRace race = new RunningRace(r1, t1, r2, t2, r3, t3);

        System.out.println("\n---------------------------------------------");
        System.out.println("                 Race Results                ");
        System.out.println("---------------------------------------------");
        System.out.println("1st Place: " + race.getFirstPlace());
        System.out.println("2nd Place: " + race.getSecondPlace());
        System.out.println("3rd Place: " + race.getThirdPlace());
        System.out.println("---------------------------------------------");
        System.out.println("             End of Race Analysis            ");
        System.out.println("=============================================");

        input.close();
    }
}
```

Listing 3: Java Program: RunningRace.java

Test Plan

Test Case	Sample Data (Name, Time)	Expected Output
All distinct times	(Alice, 12.5), (Bob, 10.8), (Charlie, 15.2)	1st: Bob, 2nd: Alice, 3rd: Charlie
Runner 1 fastest	(Alice, 9.5), (Bob, 11.2), (Charlie, 13.0)	1st: Alice, 2nd: Bob, 3rd: Charlie
Runner 3 fastest	(Alice, 14.0), (Bob, 13.5), (Charlie, 12.0)	1st: Charlie, 2nd: Bob, 3rd: Alice
Close times (different order)	(Alice, 10.0), (Bob, 10.5), (Charlie, 11.0)	1st: Alice, 2nd: Bob, 3rd: Charlie
Boundary case (two equal times)	(Alice, 10.0), (Bob, 10.0), (Charlie, 12.0)	Tie-handling not implemented; program may return Charlie as 3rd, but 1st/2nd can vary.

9. Problem Description (*SpeedOfSound*)

The goal of this program is to determine the time it takes for a sound wave to travel a specified distance through different media (Air, Water, Steel). The user selects the medium from a menu, enters the distance in feet, and the program calculates the travel time using predefined speeds of sound.

Inputs

- **Medium Choice:** User selects from the menu:
  - 1 = Air (1100 ft/s)
  - 2 = Water (4900 ft/s)
  - 3 = Steel (16400 ft/s)
- **Distance:** Positive real number representing the distance (in feet) that the sound wave will travel.

Processes / Operations

1. Display a menu with medium choices.
2. Validate the user's choice (must be 1, 2, or 3).
3. Prompt the user to enter the distance in feet.
4. Based on medium:

$$\text{Time} = \frac{\text{Distance}}{\text{Speed in Medium}}$$

   - Air: $t = \frac{d}{1100}$
   - Water: $t = \frac{d}{4900}$
   - Steel: $t = \frac{d}{16400}$

Outputs

The program displays:

- The medium selected.
- The distance traveled (in feet).
- The computed time (in seconds) to travel that distance in the chosen medium.

Code Listing

```
import java.util.Scanner;

public class SpeedOfSound {

    private double distance; // in feet

    public SpeedOfSound(double distance) {
        this.distance = distance;
    }

    public double getTimeInAir() {
        return distance / 1100.0;
    }

    public double getTimeInWater() {
        return distance / 4900.0;
    }

    public double getTimeInSteel() {
        return distance / 16400.0;
    }

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

        System.out.println("=============================================");
        System.out.println("         Speed of Sound - Calculator         ");
        System.out.println("=============================================");
        System.out.println("Select a medium for sound travel:");
        System.out.println("1. Air");
        System.out.println("2. Water");
        System.out.println("3. Steel");
        System.out.print("Enter your choice (1-3): ");
        int choice = input.nextInt();

        if (choice < 1 || choice > 3) {
            System.out.println("Error: Invalid selection!");
            input.close();
            return;
        }

        System.out.print("Enter the distance (in feet): ");
        double distance = input.nextDouble();

        SpeedOfSound sound = new SpeedOfSound(distance);

        System.out.println("\n---------------------------------------------");
        System.out.println("                Travel Report                ");
        System.out.println("---------------------------------------------");

        switch (choice) {
            case 1:
                System.out.printf("Medium: Air\nDistance: %.2f ft\nTime: %.6f sec\
                    n",
                                  distance, sound.getTimeInAir());
                break;
```

```
            case 2:
                System.out.printf("Medium: Water\nDistance: %.2f ft\nTime: %.6f
                    sec\n",
                                  distance, sound.getTimeInWater());
                break;
            case 3:
                System.out.printf("Medium: Steel\nDistance: %.2f ft\nTime: %.6f
                    sec\n",
                                  distance, sound.getTimeInSteel());
                break;
        }

        System.out.println("-----------------------------------------");
        System.out.println("           End of Speed Calculation          ");
        System.out.println("=========================================");

        input.close();
    }
}
```

Listing 4: Java Program: SpeedOfSound.java

Test Plan

Test Case	**Sample Data (Choice, Distance)**	**Expected Output**
Air, simple distance	(1, 1100)	Medium: Air, Distance: 1100 ft, Time: 1.000000 sec
Water, larger distance	(2, 9800)	Medium: Water, Distance: 9800 ft, Time: 2.000000 sec
Steel, very large distance	(3, 16400)	Medium: Steel, Distance: 16400 ft, Time: 1.000000 sec
Invalid menu choice	(4, 1000)	Error: Invalid selection!
Small distance in air	(1, 550)	Medium: Air, Distance: 550 ft, Time: 0.500000 sec
Boundary check water	(2, 4900)	Medium: Water, Distance: 4900 ft, Time: 1.000000 sec

10. Problem Description (*FreezingBoilingDemo*)

The goal of this program is to determine whether three substances (Ethyl Alcohol, Oxygen, and Water) will freeze or boil at a user-entered temperature in Fahrenheit. The program checks each substance's freezing and boiling thresholds and displays the results.

Inputs

- **Temperature:** A real number in Fahrenheit, entered by the user.

Processes / Operations

1. Read the temperature from user input.
2. Compare temperature against freezing and boiling points:
   - Ethyl Alcohol: Freeze ≤ -173, Boil ≥ 172
   - Oxygen: Freeze ≤ -362, Boil ≥ -306
   - Water: Freeze ≤ 32, Boil ≥ 212
3. Display which substances will freeze or boil at the given temperature.

Outputs

The program displays:

- The input temperature.
- A list of substances that will **freeze**.
- A list of substances that will **boil**.
- If none freeze/boil, the program shows *(None)*.

Code Listing

```
/* Checks freezing and boiling points for Ethyl Alcohol, Oxygen, Water */

import java.util.Scanner;

class FreezingBoilingPoints {
    private double temperature;

    public FreezingBoilingPoints(double temperature) {
        this.temperature = temperature;
    }
    public boolean isEthylFreezing() { return temperature <= -173; }
    public boolean isEthylBoiling()  { return temperature >= 172; }
    public boolean isOxygenFreezing(){ return temperature <= -362; }
    public boolean isOxygenBoiling() { return temperature >= -306; }
    public boolean isWaterFreezing() { return temperature <= 32; }
    public boolean isWaterBoiling()  { return temperature >= 212; }
}

public class FreezingBoilingDemo {
```

```
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter a temperature in Fahrenheit: ");
        double temp = sc.nextDouble();

        FreezingBoilingPoints fb = new FreezingBoilingPoints(temp);

        System.out.println("Substances that will FREEZE:");
        if (fb.isEthylFreezing()) System.out.println(" - Ethyl Alcohol");
        if (fb.isOxygenFreezing()) System.out.println(" - Oxygen");
        if (fb.isWaterFreezing()) System.out.println(" - Water");

        System.out.println("Substances that will BOIL:");
        if (fb.isEthylBoiling()) System.out.println(" - Ethyl Alcohol");
        if (fb.isOxygenBoiling()) System.out.println(" - Oxygen");
        if (fb.isWaterBoiling()) System.out.println(" - Water");

        sc.close();
    }
}
```

Listing 5: Java Program: FreezingBoilingDemo.java

Test Plan

Test Case	**Sample Data (Temp °F)**	**Expected Output**
Below all freezing points	-400	Freeze: Oxygen, Ethyl Alcohol, Water Boil: (None)
At Oxygen boiling point	-306	Freeze: (None) Boil: Oxygen
At Ethyl Alcohol freezing point	-173	Freeze: Ethyl Alcohol Boil: (None)
At Water freezing point	32	Freeze: Water Boil: (None)
Between freezing and boiling ranges	100	Freeze: (None) Boil: (None)
At Ethyl Alcohol boiling point	172	Freeze: (None) Boil: Ethyl Alcohol
At Water boiling point	212	Freeze: (None) Boil: Water
High temperature	300	Freeze: (None) Boil: Ethyl Alcohol, Water

11. Problem Description (*MobileServiceDemo*)

The goal of this program is to calculate a customer's monthly mobile phone bill based on the subscription package chosen and the number of minutes used. The provider offers three packages (A, B, and C), each with a base price and different rules for additional minutes.

Inputs

- **Package Type:** Character input (A, B, or C).
- **Minutes Used:** Integer input representing the total minutes.

Processes / Operations

1. Validate the package type.
2. For Package A:
    - Base cost = $39.99
    - 450 included minutes
    - Extra minutes = $0.45 each
3. For Package B:
    - Base cost = $59.99
    - 900 included minutes
    - Extra minutes = $0.40 each
4. For Package C:
    - Base cost = $69.99
    - Unlimited minutes
5. Calculate the total charges using the above rules.

Outputs

The program displays:

- Selected package
- Minutes used
- Total charges (formatted to two decimal places)

Code Listing

```
import java.util.Scanner;

class MobileServiceProvider {
    private char packageType;
    private int minutesUsed;

    public MobileServiceProvider(char packageType, int minutesUsed) {
        this.packageType = Character.toUpperCase(packageType);
        this.minutesUsed = minutesUsed;
    }

    public double calculateBill() {
        switch (packageType) {
            case 'A':
                double baseA = 39.99;
                int freeA = 450;
                double rateA = 0.45;
                return (minutesUsed > freeA)
                    ? baseA + (minutesUsed - freeA) * rateA : baseA;
            case 'B':
                double baseB = 59.99;
                int freeB = 900;
                double rateB = 0.40;
                return (minutesUsed > freeB)
                    ? baseB + (minutesUsed - freeB) * rateB : baseB;
            case 'C':
                return 69.99;
            default:
                System.out.println("Error: Invalid package type.");
                return 0.0;
        }
    }
}

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

        System.out.println("=========================================");
        System.out.println("       Mobile Service Billing System       ");
        System.out.println("=========================================");

        System.out.print("Enter your package (A, B, or C): ");
        char pkg = sc.next().charAt(0);

        System.out.print("Enter total minutes used: ");
        int minutes = sc.nextInt();

        MobileServiceProvider service = new MobileServiceProvider(pkg, minutes);
        double total = service.calculateBill();

        System.out.println("-----------------------------------------");
        System.out.println("Package Selected: " + Character.toUpperCase(pkg));
        System.out.println("Minutes Used    : " + minutes);
        System.out.printf("Total Charges   : $%.2f\n", total);
        System.out.println("=========================================");
```

```
57
58          sc.close();
59      }
60  }
```

Listing 6: Java Program: MobileServiceDemo.java

Test Plan

Test Case	**Sample Input**	**Expected Output**
Package A within limit	Package A, 300 minutes	Total Charges = $39.99
Package A over limit	Package A, 500 minutes	450 included, 50 extra @ $0.45 = $22.50 Total = $39.99 + $22.50 = $62.49
Package B within limit	Package B, 850 minutes	Total Charges = $59.99
Package B over limit	Package B, 950 minutes	900 included, 50 extra @ $0.40 = $20.00 Total = $59.99 + $20.00 = $79.99
Package C unlimited	Package C, 5000 minutes	Total Charges = $69.99
Invalid package	Package D, 300 minutes	Error message: "Invalid package type." Total = $0.00