Chapter 7 Reflection Logs

Clear Structure:

- The PiggyBank class handles the internal logic of managing coins, while the Savings class is responsible for interacting with the user. This separation of concerns helps maintain readability and modularity.

Menu System:

- The displayMenu() method lists options, and the program responds to user inputs in a straightforward way. The loop keeps the program running until the user decides to exit, providing an interactive experience.

Good Use of Methods:

- The program breaks functionality into methods making the code more modular and easier to test or modify.

Handling Coin Removal:

- The removeCoins method includes a check to ensure there are enough coins to remove. This prevents the user from accidentally removing more coins than they have.

```
package Mastery;
import java.util . Scanner;
public class Savings {
  static class PiggyBank {
     private Map<String, Integer> coins = new HashMap<>() {{
        put("pennies", 0);
        put("nickels", 0);
        put("dimes", 0);
        put("quarters", 0);
     }};
     public void addCoins(String coinType, int amount) {
        if (coins.containsKey(coinType)) {
           coins.put(coinType, coins.get(coinType) + amount);
        } else {
           System.out.println("Invalid coin type.");
        }
     }
     public void removeCoins(String coinType, int amount) {
        if (coins.containsKey(coinType)) {
           if (coins.get(coinType) >= amount) {
              coins.put(coinType, coins.get(coinType) - amount);
           } else {
              System.out.println("Not enough coins to remove.");
           }
        } else {
```

```
            System.out.println("Invalid coin type.");
        }
    }
    public double totalAmount() {
        return coins.get("pennies") * 0.01 +
                coins.get("nickels") * 0.05 +
                coins.get("dimes") * 0.10 +
                coins.get("quarters") * 0.25;
    }
}
private static void displayMenu() {
    System.out.println("\nMenu:");
    System.out.println("1: Show total in bank");
    System.out.println("2: Add a penny");
    System.out.println("3: Add a nickel");
    System.out.println("4: Add a dime");
    System.out.println("5: Add a quarter");
    System.out.println("6: Take money out of the bank");
    System.out.println("Enter 0 to quit");
    System.out.print("Enter your choice: ");
}
public static void main(String[] args) {
    PiggyBank piggyBank = new PiggyBank();
    Scanner scanner = new Scanner(System.in);
    while (true) {
        displayMenu();
        int choice = scanner.nextInt();
        switch (choice) {
            case 1:
                System.out.printf("Total amount in the bank: $%.2f%n", piggyBank.totalAmount());
                break;
            case 2:
                piggyBank.addCoins("pennies", 1);
                System.out.println("Added a penny.");
                break;
            case 3:
                piggyBank.addCoins("nickels", 1);
                System.out.println("Added a nickel.");
                break;
            case 4:
                piggyBank.addCoins("dimes", 1);
                System.out.println("Added a dime.");
                break;
            case 5:
                piggyBank.addCoins("quarters", 1);
                System.out.println("Added a quarter.");
                break;
            case 6:
                System.out.print("Enter coin type to remove (pennies, nickels, dimes, quarters): ");
                String coinTypeRemove = scanner.next().toLowerCase();
                System.out.print("Enter amount to remove: ");
                int amountRemove = scanner.nextInt();
                piggyBank.removeCoins(coinTypeRemove, amountRemove);
                break;
            case 0:
                System.out.println("Exiting the application.");
                scanner.close();
                return;
            default:
                System.out.println("Invalid choice. Please try again.");
        }
```

```
    }
  }
}
```

Savings

- Modular Design with **Food** Class:
    - The **Food** class is designed to encapsulate the properties of each food item. This follows the object-oriented principle of encapsulation, which makes it easier to manage food items and their attributes.
    - By using a separate class to represent food items. For example, if you wanted to add new food items, you can simply add more **Food** objects or extend the class to represent more food attributes.

- No Use of Object-Oriented Benefits:
    - The main logic in the **LunchOrder** class could benefit from more object-oriented principles. For example, the total cost is calculated directly using individual **Food** instances.

```
package Mastery;
import java.util.Scanner;
class Food {
  private double price;
  private int fat;
  private int carbs;
  private int fiber;
  public Food(double price, int fat, int carbs, int fiber) {
    this.price = price;
    this.fat = fat;
    this.carbs = carbs;
    this.fiber = fiber;
  }
  public double getPrice() {
    return price;
  }
  public int getFat() {
    return fat;
  }
  public int getCarbs() {
```

```
    return carbs;
  }
  public int getFiber() {
    return fiber;
  }
}
public class LunchOrder {
  public static void main(String[] args) {
    // Create Food objects with prices
    Food hamburger = new Food(1.85, 9, 33, 1);
    Food salad = new Food(2.00, 1, 11, 5);
    Food frenchFries = new Food(1.30, 11, 36, 4);
    Food soda = new Food(0.95, 0, 38, 0);
    Scanner scanner = new Scanner(System.in);
    // Prompt user for quantities
    System.out.print("Enter the number of hamburgers: ");
    int numHamburgers = scanner.nextInt();

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

    System.out.print("Enter the number of french fries: ");
    int numFrenchFries = scanner.nextInt();

    System.out.print("Enter the number of sodas: ");
    int numSodas = scanner.nextInt();
    // Calculate total
    double total = (numHamburgers * hamburger.getPrice()) +
            (numSalads * salad.getPrice()) +
            (numFrenchFries * frenchFries.getPrice()) +
            (numSodas * soda.getPrice());
    // Display total
    System.out.printf("Total cost of the order: $%.2f%n", total);


  }
}
```

LunchOrder

- Modular Design:
    - The `Num` class encapsulates the logic for extracting digits from a number, which keeps the main class (`DigitExtractor`) focused on handling user interaction and the menu.

- Case-insensitive User Choice:
    - The program is case-insensitive when it comes to the menu choices (`W`, `O`, `T`, `H`, `Q`), which is a nice touch for user convenience. By calling `toUpperCase()` on the input, the user can enter either lowercase or uppercase characters, improving usability.

- The menu is easy to follow, and the user is presented with clear options to choose from. The instructions are simple and intuitive.

- The program prints the results in a straightforward manner, making it clear what the user is interacting with.

- The program uses a `while (true)` loop to continually allow users to interact with it until they choose to quit by entering `"Q"`. The option to quit (`Q`) is clearly displayed and allows the program to exit gracefully.

```
package Mastery;
import java.util.Scanner;
class Num {
  private int number;
  public Num(int number) {
    this.number = number;
  }
  public int getWhole() {
    return number;
  }
  public int getOnes() {
    return Math.abs(number) % 10;
  }
  public int getTens() {
    return (Math.abs(number) / 10) % 10;
  }
  public int getHundreds() {
    return (Math.abs(number) / 100) % 10;
  }
}
public class DigitExtractor {
  public static void main(String[] args) {
    Scanner scanner = new Scanner(System.in);
    while (true) {
      System.out.print("Enter an integer: ");
      int userInput;
      try {
```

```
            userInput = Integer.parseInt(scanner.nextLine());
        } catch (NumberFormatException e) {
            System.out.println("Invalid input. Please enter a valid integer.");
            continue;
        }
        Num numObj = new Num(userInput);
        while (true) {
            System.out.println("\nInput W for whole number");
            System.out.println("Input O for ones place");
            System.out.println("Input T for tens");
            System.out.println("Input H for hundreds");
            System.out.println("Input Q to quit");
            System.out.print("Enter your choice: ");
            String choice = scanner.nextLine().trim().toUpperCase();
            switch (choice) {
                case "W":
                    System.out.println("Whole number: " + numObj.getWhole());
                    break;
                case "O":
                    System.out.println("Ones digit: " + numObj.getOnes());
                    break;
                case "T":
                    System.out.println("Tens digit: " + numObj.getTens());
                    break;
                case "H":
                    System.out.println("Hundreds digit: " + numObj.getHundreds());
                    break;
                case "Q":
                    System.out.println("Exiting the program.");
                    scanner.close();
                    return;
                default:
                    System.out.println("Invalid choice. Please try again.");
            }
        }
    }
}
```

DigitExtractor
