CIS007-3 Comparative Integrated Systems
Ruchi Bajracharya
2133112

Java Recycling Machine Documentation

Overview

This project simulates a recycling machine system that allows customers to return cans, bottles, and crates. The system checks the type of each item, registers the returned items, and prints a receipt showing the details of the deposited items, their values, and the total return sum. The system uses Object-Oriented Design (OOD) principles to define classes and their interactions.

Classes

1. CustomerPanel

- **Description**: Handles user interactions and processes deposited items.
- **Methods**:
    - itemReceived(int slot, int count): Receives items based on the slot number and count, adds them to the receipt basis.
    - printReceipt(): Prints the receipt using the deposit item receiver.

2. DepositItemReceiver

- **Description**: Classifies items and manages receipt creation.
- **Methods**:
    - classifyItem(int slot): Classifies items into Can, Bottle, or Crate based on the slot number.
    - createReceiptBasis(): Creates a new receipt basis.
    - printReceipt(ReceiptBasis receiptBasis): Prints the receipt using the ReceiptPrinter.

3. ReceiptBasis

- **Description**: Maintains a list of deposited items and computes the total sum.
- **Methods**:
    - addItem(DepositItem item): Adds a deposit item to the receipt basis.
    - computeSum(): Computes the total value of the deposited items.
    - getItems(): Returns the list of deposited items.

4. DepositItem

- **Description**: Abstract class representing a deposit item.
- **Attributes**:
    - number: The number of items.
    - value: The value of the item.
- **Constructors**:
    - DepositItem(int number, double value): Initializes the deposit item with a number and value.
- **Methods**:
    - getNumber(): Returns the number of items.
    - getValue(): Returns the value of the item.

5. ReceiptPrinter

- **Description**: Prints the receipt details.
- **Methods**:
    - print(ReceiptBasis receiptBasis): Prints the items deposited, their values, and the total return sum.

6. Can

- **Description**: Subclass of DepositItem representing a can.
- **Attributes**:
    - weight: The weight of the can.
    - size: The size of the can.
- **Constructors**:
    - Can(): Initializes the can with default values (number=1, value=0.10, weight=0.5, size="small").
- **Methods**:
    - getWeight(): Returns the weight of the can.
    - getSize(): Returns the size of the can.

7. Bottle

- **Description**: Subclass of DepositItem representing a bottle.
- **Attributes**:
    - weight: The weight of the bottle.
    - size: The size of the bottle.
- **Constructors**:

    - Bottle(): Initializes the bottle with default values (number=1, value=0.20, weight=1.0, size="medium").
- **Methods**:
    - getWeight(): Returns the weight of the bottle.
    - getSize(): Returns the size of the bottle.

8. Crate

- **Description**: Subclass of DepositItem representing a crate.
- **Attributes**:
    - weight: The weight of the crate.
    - size: The size of the crate.
- **Constructors**:
    - Crate(): Initializes the crate with default values (number=1, value=0.30, weight=2.0, size="large").
- **Methods**:
    - getWeight(): Returns the weight of the crate.
    - getSize(): Returns the size of the crate.

Main Class

- **Description**: Entry point of the application. Handles user input and processes deposit items.
- **Methods**:
    - main(String[] args): Main method that initializes the system and processes user input.

CIS007-3 Comparative Integrated Systems
Ruchi Bajracharya
2133112

Learnings:

It demonstrates the four main principles of Object-Oriented Programming (OOP): encapsulation, abstraction, inheritance, and polymorphism.

1. Encapsulation

Encapsulation is the concept of wrapping data (variables) and code (methods) together as a single unit, or class, and restricting access to some of the object's components. This is often achieved through the use of access modifiers like `private`, `protected`, and `public`.

- **Private Fields**: In the `Bottle`, `Can`, and `Crate` classes, the fields `weight` and `size` are private. This ensures that these fields cannot be accessed directly from outside these classes.

```
    private double weight;
    private String size;
```

- **Public Getters**: Access to these private fields is provided through public getter methods, which allow other classes to read these values without directly accessing the private fields.

```
    public double getWeight() {
        return weight;
    }

    public String getSize() {
        return size;
    }
```

2. Abstraction

Abstraction is the concept of hiding the complex implementation details and showing only the essential features of an object. It allows the user to interact with the object at a high level without needing to understand its internal workings.

- **Abstract Class**: The DepositItem class is abstract, meaning it cannot be instantiated directly. It provides a general template for its subclasses (Bottle, Can, Crate) with common properties and methods.

```
public abstract class DepositItem {
    private int number;
    private double value;

    public DepositItem(int number, double value) {
        this.number = number;
        this.value = value;
    }

    public int getNumber() {
        return number;
    }

    public double getValue() {
        return value;
    }
}
```

- **Class Methods**: The DepositItemReceiver class abstracts the process of creating DepositItem objects based on the slot number and provides a simple method for this.

```
public DepositItem classifyItem(int slot) {
        switch (slot) {
            case 1:
                return new Can();
            case 2:
                return new Bottle();
            case 3:
```

```
                return new Crate();
            default:
                    throw new IllegalArgumentException("Invalid
slot number");
        }
    }
```

3. Inheritance

Inheritance is a mechanism where one class (child) inherits the properties and methods of another class (parent). This promotes code reuse and establishes a relationship between the parent and child classes.

Class Inheritance: Bottle, Can, and Crate classes extend the DepositItem class, inheriting its fields and methods.

4. Polymorphism

Polymorphism allows objects of different classes to be treated as objects of a common superclass. It is particularly useful for methods that operate on objects of different classes in a uniform way.

Method Overriding: The DepositItem class has a method getValue(), and each subclass (Bottle, Can, Crate) inherits this method. Even though the method is inherited, it's used polymorphically in ReceiptBasis to calculate the total value, treating all DepositItem instances uniformly.

DepositItem Class

```
public abstract class DepositItem {
    private int number;
    private double value;

    public DepositItem(int number, double value) {
        this.number = number;
        this.value = value;
    }

    public int getNumber() {
```

```
        return number;
    }

    public double getValue() {
        return value;
    }
}
```

ReceiptBasis Class

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

public class ReceiptBasis {
    private List<DepositItem> items;

    public ReceiptBasis() {
        this.items = new ArrayList<>();
    }

    public void addItem(DepositItem item) {
        items.add(item);
    }

    public double computeSum() {
        double totalSum = 0;
        for (DepositItem item : items) {
            totalSum += item.getValue();
        }
        return totalSum;
    }

    public List<DepositItem> getItems() {
        return items;
    }
}
```

Dynamic Method Dispatch: The method classifyItem in DepositItemReceiver returns a DepositItem reference, but it can hold any subclass instance (Can, Bottle, Crate). This allows the code to interact with different types of DepositItem objects in a consistent way.

Conclusion

The system is designed to handle multiple types of deposit items (cans, bottles, and crates) and compute the total return value based on predefined values for each type. The main interaction occurs through the CustomerPanel class, which processes the items and prints a receipt. The DepositItemReceiver class classifies items and manages receipt creation, while the ReceiptBasis class maintains a list of items and computes the total return sum. Each deposit item (Can, Bottle, Crate) extends the abstract DepositItem class, which defines common attributes and methods for all deposit items. The ReceiptPrinter class handles the formatting and printing of the receipt. The Main class serves as the entry point for the application, handling user input and processing the deposited items accordingly.