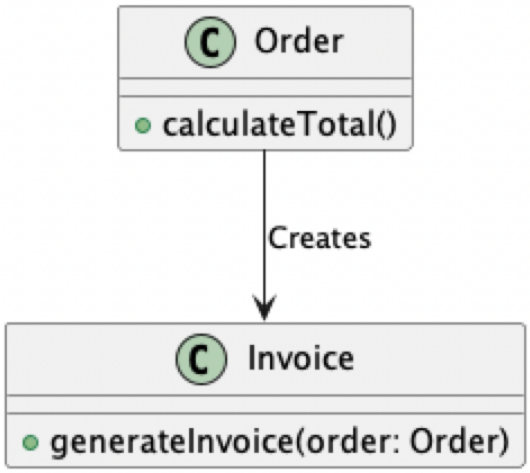
ENSF607
Assignment 4
Balkarn Gill
October 27th, 2023

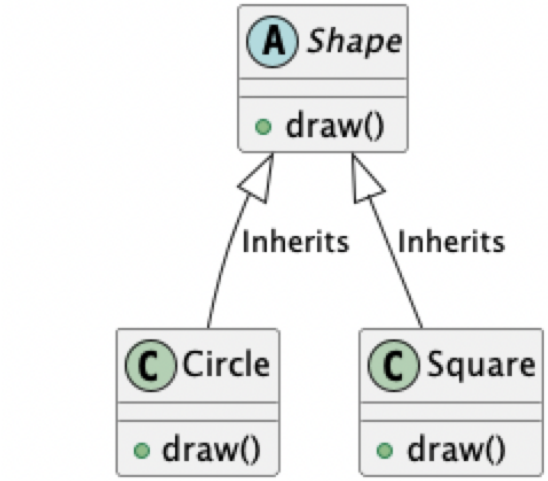
Exercise - 1: For each principle provide a class diagram (10 Marks)
Exercise - 2: For each principle provide the supporting code example (20 Marks)

1. Single Responsibility Principle (SRP):



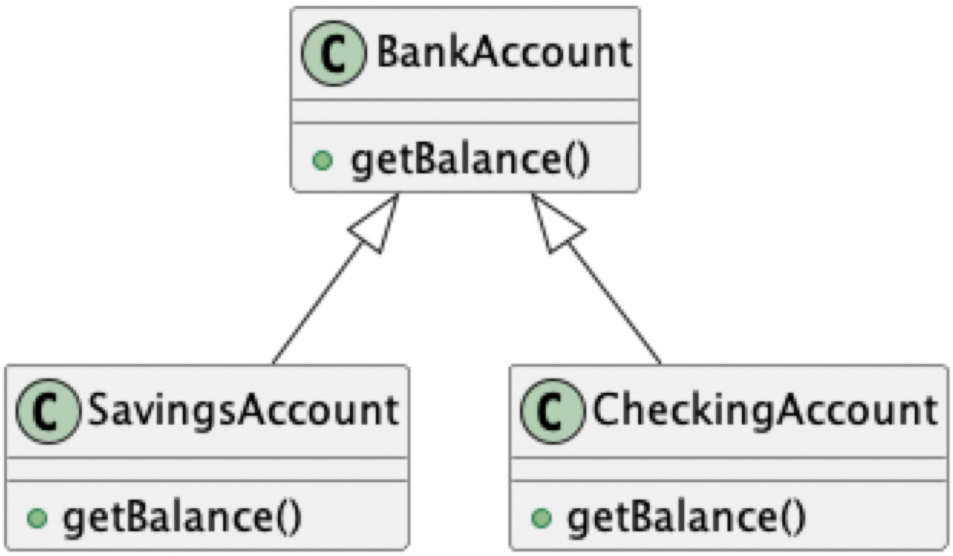

```java
class Order {
    private List<Item> items;
    private Customer customer;

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

    public double calculateTotal() {
        double total = 0;
        for (Item item : items) {
            total += item.getPrice();
        }
        return total;
    }
}

class Item {
    private String name;
    private double price;
    // Getters and setters

    public double getPrice() {
        return price;
    }
}

class Invoice {
    public void generateInvoice(Order order) {
        // Generate an invoice for the provided order
        Customer customer = order.getCustomer();
        // Include items, total, and customer information in the invoice
    }
}
```

2. Open/Closed Principle (OCP):



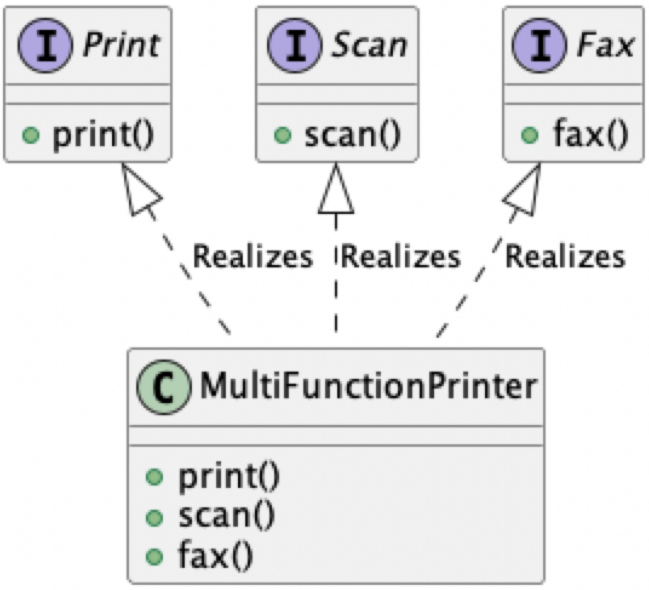

```java
abstract class Shape {
    public abstract void draw();
}

class Circle extends Shape {
    private double radius;

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

    @Override
    public void draw() {
        // Draw a circle with the given radius
    }
}

class Square extends Shape {
    private double sideLength;

    public Square(double sideLength) {
        this.sideLength = sideLength;
    }

    @Override
    public void draw() {
        // Draw a square with the given side length
    }
}
```

3. Liskov Substitution Principle (LSP):



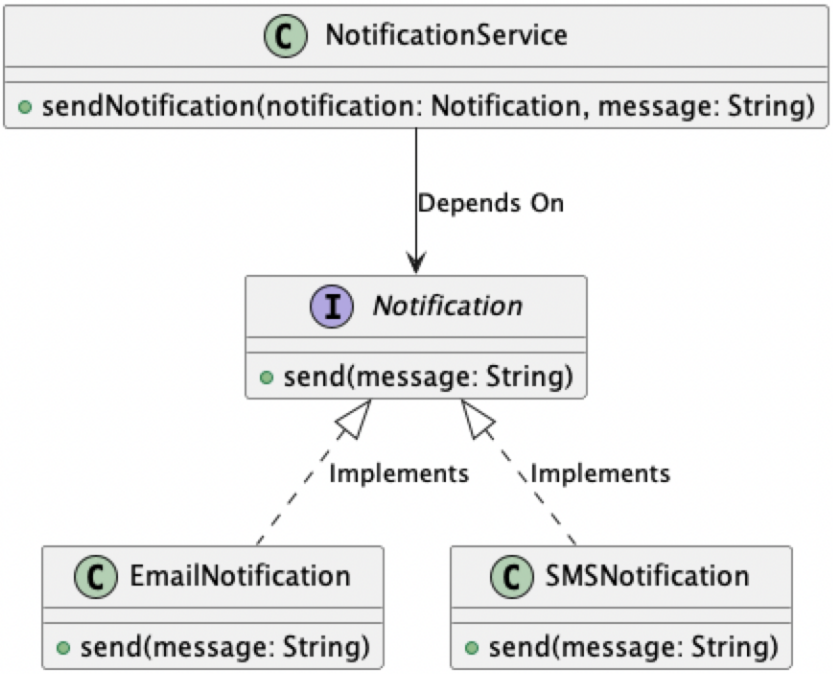

```
class BankAccount {
    protected double balance;

    public double getBalance() {
        return balance;
    }
}

class SavingsAccount extends BankAccount {
    private double interestRate;

    // Additional methods and properties specific to savings accounts
}

class CheckingAccount extends BankAccount {
    private double overdraftLimit;

    // Additional methods and properties specific to checking accounts
}
```

4. Interface Segregation Principle (ISP):




```java
interface Print {
    void print();
}

interface Scan {
    void scan();
}

interface Fax {
    void fax();
}

class MultiFunctionPrinter implements Print, Scan, Fax {
    @Override
    public void print() {
        // Implement print functionality
    }

    @Override
    public void scan() {
        // Implement scan functionality
    }

    @Override
    public void fax() {
        // Implement fax functionality
    }
}
```

5. Dependency Inversion Principle (DIP):




```java
interface Notification {
    void send(String message);
}

class EmailNotification implements Notification {
    @Override
    public void send(String message) {
        // Send an email
    }
}

class SMSNotification implements Notification {
    @Override
    public void send(String message) {
        // Send an SMS
    }
}

class NotificationService {
    public void sendNotification(Notification notification, String message) {
        // Use the provided notification method to send a message
        notification.send(message);
    }
}
```

Exercise - 2: For each principle discussed provide an appropriate use case. Discuss, why you would apply this principle for the described use case compared to the other four. (30 Marks)

1. Single Responsibility Principle (SRP):

Use Case: Logging and Reporting System

Why SRP: In a logging and reporting system, SRP is essential. It ensures that the Logger class has the sole responsibility of capturing log data, while the ReportGenerator class focuses on generating reports. Without SRP, combining both logging and reporting in a single class would lead to a complex and less maintainable codebase. SRP promotes a clear and focused responsibility for each class, enhancing maintainability, extensibility, and testability.

2. Open/Closed Principle (OCP):

Use Case: Plugin System

Why OCP: In this use case, OCP keeps the system open for extensions (adding new shapes) while remaining closed for modifications to the core system. Users can create new plugins (shapes) by extending the abstract class Shape without altering the core system. OCP is chosen in this context to strike a balance between extensibility and system stability. Compared to the other principles, OCP is the most suitable because it focuses on allowing extensions without changing existing code.

3. Liskov Substitution Principle (LSP):

Use Case: Banking System with Account Types

Why LSP: In this context, SavingsAccount and CheckingAccount can be substituted for BankAccount without breaking the system, demonstrating the Liskov Substitution Principle. Each derived class can provide additional methods and properties specific to its account type while still allowing users to work with them as if they were BankAccounts. This adherence to a consistent interface promotes code reliability and flexibility. LSP is chosen here because it ensures that the behavior of derived classes remains consistent with the base class, making it the most appropriate principle for this use case compared to the other four.

4. Interface Segregation Principle (ISP):

Use Case: User Permissions System

Why ISP: ISP segregates interfaces like Print, Scan, and Fax to ensure that each type of user only depends on the methods they require. In this use case, different user types may have distinct sets of permissions. By segregating interfaces and making them more focused, ISP

prevents code bloat and potential errors that could occur if all users were forced to implement methods they don't need. ISP is the most appropriate choice in this context compared to the other principles.

5. Dependency Inversion Principle (DIP):

Use Case: Payment Gateway Integration

Why DIP: DIP suggests depending on abstractions (e.g., interfaces) rather than concrete implementations. In this use case, payment gateways can change, and new ones can be added. DIP ensures flexibility and avoids tightly coupling the code to specific payment gateways. The UML diagram signifies that the NotificationService depends on the Notification interface, not concrete classes, enabling easy switching or addition of payment gateways without changing the core payment processing code. DIP is the best choice for its loose coupling and adaptability.