МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ
УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ
“БРЕСТСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ”
КАФЕДРА ИНТЕЛЛЕКТУАЛЬНЫХ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ

Отчёт

по лабораторной работе №5

Выполнила:
студентка группы ПО-9
Матюшик Е.П.

Проверил:
Крощенко А. А.

Брест 2024

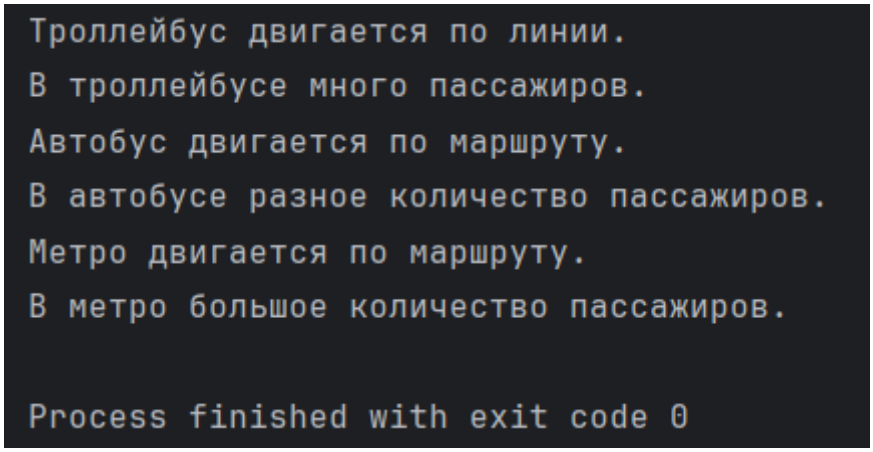
Цель работы: приобрести практические навыки в области объектно-ориентированного проектирования.

Задание 1

Вариант 13

Реализовать абстрактные классы или интерфейсы, а также наследование и полиморфизм для следующего класса: interface Транспортное Средство ← abstract class Общественный Транспорт ← class Троллейбус

Результат программы:



Код программы:

```
// Интерфейс Транспортное Средство
interface Transport {
   void move();
}

// Абстрактный класс Общественный Транспорт реализует интерфейс Transport
abstract class PublicTransport implements Transport {
   abstract void passengers();
}

// Класс Троллейбус наследует абстрактный класс Общественный Транспорт
class Trolleybus extends PublicTransport {
   @Override
   public void move() {
      System.out.println("Троллейбус двигается по линии.");
   }

   @Override
   void passengers() {
      System.out.println("В троллейбусе много пассажиров.");
   }
}

// Класс Автобус наследует абстрактный класс Общественный Транспорт
```

```
class Bus extends PublicTransport {
    @Override
    public void move() {
        System.out.println("Автобус двигается по маршруту.");
    }

    @Override
    void passengers() {
        System.out.println("В автобусе разное количество пассажиров.");
    }
}

// Класс Метро наследует абстрактный класс Общественный Транспорт
class Metro extends PublicTransport {
    @Override
    public void move() {
        System.out.println("Метро двигается по маршруту.");
    }

    @Override
    void passengers() {
        System.out.println("В метро большое количество пассажиров.");
    }
}

// Пример использования полиморфизма
public class Main1 {
    public static void main(String[] args) {
        // Создание экземпляра класса Троллейбус и вызов его методов
        Trolleybus trolleybus = new Trolleybus();
        trolleybus.move();
        trolleybus.passengers();

        // Создание экземпляра класса Автобус и вызов его методов
        Bus bus = new Bus();
        bus.move();
        bus.passengers();

        // Создание экземпляра класса Метро и вызов его методов
        Metro metro = new Metro();
        metro.move();
        metro.passengers();
```

```
        // Полиморфизм: создание массива Общественных Транспортов и вызов их методов
        PublicTransport[] transports = new PublicTransport[3];
        transports[0] = new Trolleybus();
        transports[1] = new Bus();
        transports[2] = new Metro();

        for (PublicTransport transport : transports) {
            transport.move();
            transport.passengers();
        }
    }
}
```

Задание 2

Вариант 5

Создать абстрактный класс Работник фирмы и подклассы Менеджер, Аналитик, Программист, Тестировщик, Дизайнер, Бухгалтер. Реализовать логику начисления зарплаты.

Результат программы:

```
Имя: John, Базовая зарплата: 2000.0
Конечная зарплата: 2500.0
Имя: Alice, Базовая зарплата: 1800.0
Конечная зарплата: 2300.0
Имя: Bob, Базовая зарплата: 2500.0
Конечная зарплата: 3500.0
```

Код программы:

```
// Абстрактный класс Работник фирмы
abstract class Employee {
    protected String name;
    protected double baseSalary;

    public Employee(String name, double baseSalary) {
        this.name = name;
        this.baseSalary = baseSalary;
    }

    // Абстрактный метод для начисления зарплаты
    abstract double calculateSalary();

    public String getName() {
        return name;
    }
```

```
    public double getBaseSalary() {
        return baseSalary;
    }

    // Метод для вывода входных данных
    public void displayInputData() {
        System.out.println("Имя: " + name + ", Базовая зарплата: " + baseSalary);
    }
}

// Подкласс Менеджер
class Manager extends Employee {
    private double bonus;

    public Manager(String name, double baseSalary, double bonus) {
        super(name, baseSalary);
        this.bonus = bonus;
    }

    @Override
    double calculateSalary() {
        return baseSalary + bonus;
    }
}

// Подкласс Аналитик
class Analyst extends Employee {
    private int completedProjects;
    private double bonusPerProject;

    public Analyst(String name, double baseSalary, int completedProjects, double
bonusPerProject) {
        super(name, baseSalary);
        this.completedProjects = completedProjects;
        this.bonusPerProject = bonusPerProject;
    }

    @Override
    double calculateSalary() {
        return baseSalary + (completedProjects * bonusPerProject);
    }
}
```

```
// Подкласс Программист
class Programmer extends Employee {
   private int linesOfCode;
   private double bonusPerLineOfCode;

   public Programmer(String name, double baseSalary, int linesOfCode, double
bonusPerLineOfCode) {
      super(name, baseSalary);
      this.linesOfCode = linesOfCode;
      this.bonusPerLineOfCode = bonusPerLineOfCode;
   }

   @Override
   double calculateSalary() {
      return baseSalary + (linesOfCode * bonusPerLineOfCode);
   }
}

// Пример использования
public class Main2 {
   public static void main(String[] args) {
      // Создаем массив объектов суперкласса и заполняем объектами подклассов
      Employee[] employees = new Employee[3];
      employees[0] = new Manager("John", 2000, 500); // Менеджер с базовой зарплатой
2000 и бонусом 500
      employees[1] = new Analyst("Alice", 1800, 5, 100); // Аналитик с базовой зарплатой
1800, завершенными проектами 5 и бонусом за проект 100
      employees[2] = new Programmer("Bob", 2500, 10000, 0.1); // Программист с базовой
зарплатой 2500, строками кода 10000 и бонусом за строку кода 0.1

      // Используем объекты подклассов для моделирования реальных ситуаций
      for (Employee employee : employees) {
         employee.displayInputData(); // Выводим входные данные
         System.out.println("Конечная зарплата: " + employee.calculateSalary());
      }
   }
}
```

Задание 3

Вариант

В задании 3 ЛР №4, где возможно, заменить объявления суперклассов объявлениями абстрактных классов или интерфейсов.

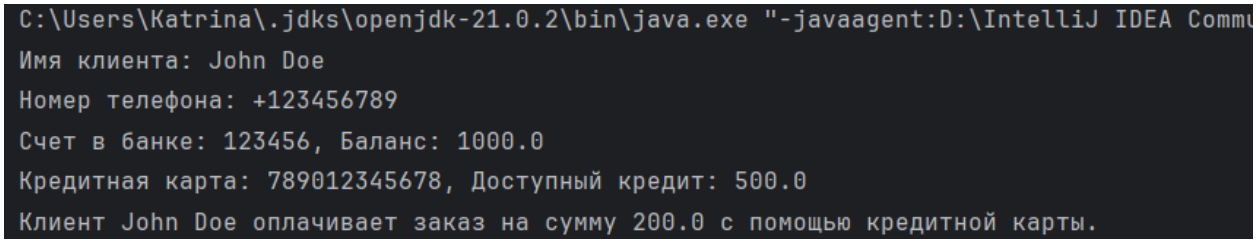
В обновлённом коде добавлены интерфейсы Account и Card, а также реализующие их классы BankAccount и CreditCard.

Интерфейс Account: определяет методы, которые должен реализовывать любой класс, представляющий банковский счет. Методы включают в себя различные операции. Позволяет абстрагировать операции с банковским счетом и имеет смысл, так как мы можем представить различные типы счетов с разными способами управления.

Интерфейс Card: определяет методы, которые должен реализовывать любой класс, представляющий кредитную карту. Методы включают в себя проверку превышения кредита, блокировку карты и получение информации о карте. Позволяет абстрагировать операции с кредитной картой и имеет смысл, так как различные банковские карты могут иметь разные правила и функциональность.

Классы BankAccount и CreditCard: реализуют соответствующие интерфейсы Account и Card. В этих классах реализованы методы, определенные в соответствующих интерфейсах. Реализация этих методов специфична для каждого типа счета и карты.

Результат программы:



Код программы:

```
// Интерфейс для банковского счета
interface Account {
    void topUp(double amount);
    void withdraw(double amount);
    void cancelAccount();
    double getBalance();
    String getAccountNumber();
}

// Интерфейс для кредитной карты
interface Card {
    boolean checkCreditExceed(double amount);
    void blockCard();
    String getCardNumber();
    double getAvailableCredit();
}

class Client {
    private String name;
    private String phoneNumber;
    private Account bankAccount;
    private Card creditCard;

    public Client(String name, String phoneNumber) {
        this.name = name;
```

```
        this.phoneNumber = phoneNumber;
    }

    public String getName() {
        return name;
    }

    public String getPhoneNumber() {
        return phoneNumber;
    }

    public void setBankAccount(Account bankAccount) {
        this.bankAccount = bankAccount;
    }

    public void setCreditCard(Card creditCard) {
        this.creditCard = creditCard;
    }

    public void payOrder(Order order) {
        System.out.println("Имя клиента: " + name);
        System.out.println("Номер телефона: " + phoneNumber);
        System.out.println("Счет в банке: " +
bankAccount.getAccountNumber() + ", Баланс: " +
bankAccount.getBalance());
        System.out.println("Кредитная карта: " +
creditCard.getCardNumber() + ", Доступный кредит: " +
creditCard.getAvailableCredit());
        System.out.println("Клиент " + name + " оплачивает заказ на
сумму " + order.getAmount() + " с помощью кредитной карты.");
    }

    public void makePaymentToAccount(Account recipientAccount, double
amount) {
        System.out.println("Имя клиента: " + name);
        System.out.println("Номер телефона: " + phoneNumber);
        System.out.println("Счет в банке: " +
bankAccount.getAccountNumber() + ", Баланс: " +
bankAccount.getBalance());
        System.out.println("Кредитная карта: " +
creditCard.getCardNumber() + ", Доступный кредит: " +
creditCard.getAvailableCredit());
        System.out.println("Клиент " + name + " делает платеж на
другой счет на сумму " + amount + " с помощью счета в банке.");
    }

    public void blockCreditCard() {
        if (creditCard != null) {
            creditCard.blockCard();
```

```
            System.out.println("Кредитная карта клиента " + name + "
была заблокирована.");
        }
    }

    public void cancelBankAccount() {
        if (bankAccount != null) {
            bankAccount.cancelAccount();
            System.out.println("Счет клиента " + name + " был
аннулирован.");
        }
    }
}

class BankAccount implements Account {
    private String accountNumber;
    private double balance;

    public BankAccount(String accountNumber, double balance) {
        this.accountNumber = accountNumber;
        this.balance = balance;
    }

    @Override
    public void topUp(double amount) {
        balance += amount;
    }

    @Override
    public void withdraw(double amount) {
        if (balance >= amount) {
            balance -= amount;
        } else {
            System.out.println("Недостаточно средств на счете");
        }
    }

    @Override
    public void cancelAccount() {
        // Логика аннулирования счета
    }

    @Override
    public double getBalance() {
        return balance;
    }

    @Override
    public String getAccountNumber() {
        return accountNumber;
```

```
    }
}

class CreditCard implements Card {
    private String cardNumber;
    private double creditLimit;
    private double availableCredit;
    private boolean blocked;

    public CreditCard(String cardNumber, double creditLimit) {
        this.cardNumber = cardNumber;
        this.creditLimit = creditLimit;
        this.availableCredit = creditLimit;
    }

    @Override
    public boolean checkCreditExceed(double amount) {
        return (availableCredit - amount) < 0;
    }

    @Override
    public void blockCard() {
        blocked = true;
    }

    @Override
    public String getCardNumber() {
        return cardNumber;
    }

    @Override
    public double getAvailableCredit() {
        return availableCredit;
    }
}

class Order {
    private String orderNumber;
    private double amount;

    public Order(String orderNumber, double amount) {
        this.orderNumber = orderNumber;
        this.amount = amount;
    }

    public double getAmount() {
        return amount;
    }
}
```

```
class Administrator {
    public void blockCardForOverdraft(Client client) {
        System.out.println("Администратор заблокировал карту клиента "
+ client.getName() + " за превышение кредита.");
    }
}

public class Main3 {
    public static void main(String[] args) {
        // Пример использования системы
        Client client = new Client("John Doe", "+123456789");
        Account bankAccount = new BankAccount("123456", 1000.0);
        Card creditCard = new CreditCard("789012345678", 500.0);

        client.setBankAccount(bankAccount);
        client.setCreditCard(creditCard);

        Order order = new Order("0001", 200.0);
        client.payOrder(order);

        Account recipientAccount = new BankAccount("654321", 0.0);
        client.makePaymentToAccount(recipientAccount, 100.0);

        Administrator administrator = new Administrator();
        administrator.blockCardForOverdraft(client);
    }
}
```

Вывод: в ходе выполнения лабораторной были приобретены практические навыки в области объектно-ориентированного проектирования.