Plan

2023-09-21

1. Repetition
2. OOP, Encapsulation, Polymorphism

---

1. Повторение
2. ООР, Инкапсуляция, Полиморфизм

Theory

▶ **English**

▼ **На русском**

объектно-ориентированное программирование (ООП)

Объектно-ориентированное программирование (ООП) - это парадигма программирования, которая базируется на концепции объектов. Она представляет собой методологию разработки программ, где центральными элементами являются объекты, которые могут иметь свойства (атрибуты) и методы (функции), а также взаимодействовать друг с другом.

Основные концепции ООП:

1. **Классы и объекты**:
    - **Класс** - это шаблон или чертеж, определяющий структуру и поведение объектов. Он описывает атрибуты (свойства) и методы (функции), которые будут доступны объектам этого класса.
    - **Объект** - это конкретный экземпляр класса, который создается на основе его описания. Объекты могут взаимодействовать друг с другом, вызывать методы и обмениваться данными.
2. **Инкапсуляция**:
    - Инкапсуляция представляет собой концепцию, при которой данные (атрибуты) и методы, которые работают с этими данными, объединены внутри класса. Таким образом, объект скрывает детали реализации и предоставляет только необходимый интерфейс для взаимодействия с внешним миром.

3. **Наследование**:
    - Наследование позволяет создавать новые классы на основе существующих классов. Это способствует повторному использованию кода и созданию иерархии классов, где дочерние классы наследуют свойства и методы родительских классов.
4. **Полиморфизм**:
    - Полиморфизм позволяет объектам разных классов реагировать на одинаковые методы вызова по-разному. Это упрощает общий интерфейс для работы с различными типами объектов.

Преимущества ООП:

- **Модульность**: Программа разбивается на небольшие, независимые объекты, что делает код более понятным и легко поддерживаемым.
- **Повторное использование кода**: Благодаря наследованию, можно повторно использовать существующий код для создания новых классов и объектов.
- **Упрощенное тестирование**: Классы и методы могут быть тестируемыми модулями, что облегчает проверку и отладку кода.
- **Увеличенная производительность разработки**: ООП способствует более эффективной организации работы над проектами и снижению времени разработки.

Объектно-ориентированное программирование широко используется в современном программировании и является мощным инструментом для разработки сложных программных систем.

Что такое объекты?

Мир, в котором мы существуем, насыщен разнообразными объектами. Если обратить внимание вокруг, можно увидеть, что нас окружают здания, природные образования, автомобили, мебель, техника, компьютеры. Все эти вещи можно рассматривать как объекты, и каждый из них обладает своим набором характеристик, функциональностью и предназначением.

Мы привыкли использовать объекты для выполнения разных задач в повседневной жизни. Например, для передвижения до места работы мы используем автомобиль, для приготовления пищи – кухонную утварь, а для отдыха – удобную мебель.

Этот объектно-ориентированный способ мышления также нашел свое место в программировании. Подход, который основывается на использовании объектов, называется объектно-ориентированным программированием (ООП).

Пример

Давайте рассмотрим пример. Представьте, что вы создали новую модель смартфона и планируете начать его массовое производство. Как разработчик, вы знаете, как он будет

функционировать, из каких компонентов состоит (корпус, микрофон, динамик, провода, кнопки и так далее). Вы знаете, как все эти компоненты соединяются друг с другом.

Однако вы не будете сами собирать каждый смартфон, у вас есть команда работников. Чтобы обеспечить однородность в производстве и избежать необходимости давать инструкции на каждом этапе, вам нужно создать подробное описание устройства смартфона. В объектно-ориентированном программировании, это описание называется классом, из которого создаются конкретные экземпляры - объекты.

Класс - это своего рода чертеж, описание объекта, который еще не был создан, но определяет его характеристики, методы и способ создания. **Объект**, в свою очередь, - это конкретный экземпляр класса, созданный на основе этого описания.

Инкапсуляция в Java:

Инкапсуляция - это концепция объектно-ориентированного программирования, которая позволяет скрыть внутренние детали реализации объекта и предоставить только необходимый интерфейс для взаимодействия с ним. В Java инкапсуляция достигается с помощью модификаторов доступа (private, protected, public) и методов получения (геттеров) и установки (сеттеров) для доступа к атрибутам объекта.

Пример инкапсуляции в Java:

```
public class Person {
    private String name; // Приватное поле

    public String getName() {
        return name; // Геттер для чтения значения name
    }

    public void setName(String newName) {
        name = newName; // Сеттер для установки значения name
    }
}

public class Main {
    public static void main(String[] args) {
        Person person = new Person();
        person.setName("John");
        System.out.println("Имя: " + person.getName());
    }
}
```

В этом примере атрибут `name` класса `Person` является приватным, и мы используем геттер и сеттер для доступа к этому атрибуту. Таким образом, мы скрываем детали реализации `name` и обеспечиваем контролируемый доступ к нему.

Статический полиморфизм (перегрузка методов) в Java:

Статический полиморфизм - это концепция, при которой один и тот же метод может иметь разные реализации, в зависимости от количества и типа его аргументов. Это достигается с помощью перегрузки методов в Java.

Пример статического полиморфизма (перегрузки методов) в Java:

```
public class Calculator {
    public int add(int a, int b) {
        return a + b;
    }

    public double add(double a, double b) {
        return a + b;
    }

    public String add(String a, String b) {
        return a + b;
    }
}

public class Main {
    public static void main(String[] args) {
        Calculator calculator = new Calculator();
        System.out.println("Сумма целых чисел: " + calculator.add(5, 10));
        System.out.println("Сумма дробных чисел: " + calculator.add(2.5, 3.
        System.out.println("Сумма строк: " + calculator.add("Hello", " Worl
    }
}
```

Здесь метод `add` перегружен для работы с разными типами данных (целыми числами, дробными числами и строками). В зависимости от переданных аргументов вызывается соответствующая версия метода `add`. Это называется статическим полиморфизмом или перегрузкой методов.

Homework

▶ **English**

▼ **На русском**

1 Конкурс по съеданию пиццы

У нас проводится конкурс по съеданию пиццы с участием группы из 5 до 8 человек. Каждый участник принимает участие в конкурсе и съедает случайное количество кусков пиццы, которое может быть от 0 до 6. Нам нужно провести анализ результатов конкурса и определить следующее:

1. **Количество съеденных кусков:** Необходимо определить, сколько кусков пиццы съел каждый участник.
2. **Победитель:** Определить, кто из участников съел наибольшее количество кусков пиццы и стал победителем конкурса.
3. **Второе и третье место:** Определить участников, занявших второе и третье место в конкурсе.
4. **Использованные пиццы:** Рассчитать общее количество пицц, которые были использованы для конкурса, учитывая, что каждая пицца была разрезана на 8 кусков.
5. **Не съеденные куски:** Определить, сколько кусков пиццы осталось несъеденными после завершения конкурса.
6. **Награды:** Вручить награды победителям с первого по третье место в соответствии с призами: первому месту - золотую медаль и сертификат на бесплатную пиццу, второму месту - серебряную медаль и сертификат на скидку 50% на следующую пиццу, третьему месту - бронзовую медаль и сертификат на скидку 25% на следующую пиццу.

Задача 2

Дано 6 участников конкурса по съеданию пиццы. Всего имеется 25 кусков пиццы. Участникам предлагается пицца по очереди, и каждый участник съедает 1 кусок пиццы за круг.

Задача состоит в том, чтобы определить:

- Кто из участников победил в конкурсе, то есть кто съел последний кусок пиццы.
- Сколько кругов потребовалось для завершения конкурса, при условии, что все участники поедают по одному куску пиццы за круг.

- Какой из участников съел предпоследний кусок пиццы.

Code

code/Pizza/src/Main.java

```
/**
 * @author Andrej Reutow
 * created on 29.09.2023
 */


import java.util.Random;

/**
 * Конкурс по съеданию пиццы
 * У нас проводится конкурс по съеданию пиццы с участием группы из 5 до 8 че
 * Каждый участник принимает участие в конкурсе и съедает случайное количест
 * Нам нужно провести анализ результатов конкурса и определить следующее:
 * <p>
 * Количество съеденных кусков: Необходимо определить, сколько кусков пиццы
 * <p>
 * Победитель: Определить, кто из участников съел наибольшее количество кус
 * <p>
 * Второе и третье место: Определить участников, занявших второе и третье ме
 * <p>
 * Использованные пиццы: Рассчитать общее количество пицц, которые были испо
 * что каждая пицца была разрезана на 8 кусков.
 * <p>
 * Не съеденные куски: Определить, сколько кусков пиццы осталось несъеденны
 * <p>
 * Награды: Вручить награды победителям с первого по третье место в соответ
 * первому месту - золотую медаль и сертификат на бесплатную пиццу,
 * второму месту - серебряную медаль и сертификат на скидку 50% на следующу
 * третьему месту - бронзовую медаль и сертификат на скидку 25% на следующу
 */

public class Main {
    public static void main(String[] args) {
        // 1. нужны участники (Person) - определить его поля
        // 2. определить от 5 до 8 участников (Person) конкурса - рандомно
        //        int persons = ....
        // 3. создать массив с количесвом участников из 2 пункта
```

```
        //    Person[] peoples = new Person[persons];
        // 4. заполнить массив участниками.
        //      4.1 перебираем массив, и в каждую ячейку устанавливаем Perso
        // 5. определить для каждого участника количесвто съеденых пиц
        //      5.1 при необходимости нужно добавить в класс Person поле ко
        //      5.2 создать setter для этого поля.
        //      5.3 установить каждому участнику (перебрав массив участнико
        // 6. написать метод, который распечатает количесвто съеденых куско
        // 7. ...
        Person[] peoples = new Person[8]; // желающие учавствовать в конкур
        peoples[0] = new Person(1, "Name1");
        peoples[1] = new Person(2, "Name2");
        peoples[2] = new Person(3, "Name3");
        peoples[3] = new Person(4, "Name4");
        peoples[4] = new Person(5, "Name5");
        peoples[5] = new Person(6, "Name6");
        peoples[6] = new Person(7, "Name7");
        peoples[7] = new Person(8, "Name8");

        // 2. определить от 5 до 8 участников (Person) конкурса - рандомно
        Random random = new Random();
//        int arraySize = random.nextInt(9 - 5) + 5; // генерирует случайно
        int arraySize = (int) (Math.random() * (8 - 5 + 1) + 5); // генерир

        // 3. создать массив с количесвом участников из 2 пункта
        Person[] participants = new Person[arraySize]; // список отобранных

        // 4. заполнить массив участниками.
        for (int i = 0; i < participants.length; i++) {
            // 4.1 перебираем массив, и в каждую ячейку устанавливаем Person
            participants[i] = peoples[i];
        }

        // 5.3 установить каждому участнику (перебрав массив участников) ра
        generateSlicePizza(participants);

        for (int i = 0; i < participants.length; i++) {
            participants[i].sayResult();
        }
    }

    /**
     * Устанавливает для каждого участника количесвто съеденых пиц. Использ
```

```
         *
         * @param peoples массив (список) участников
         */
        public static void generateSlicePizza(Person[] peoples) {
            Random random = new Random();
            for (int i = 0; i < peoples.length; i++) { // перебераю массив учас
                Person currentPerson = peoples[i]; // достаю участника из масси
                int slices = random.nextInt(7); // 0 - 6
                currentPerson.setPizzaSlices(slices); // устнавливаю количесвто
                // peoples[i].setPizzaSlices(random.nextInt(7));
            }
        }
    }
```

code/Pizza/src/Person.java

```
    /**
     * @author Andrej Reutow
     * created on 29.09.2023
     */

    // 1. нужны участники (Person) - определить его поля
    public class Person {

        private int id;
        private String name;
        // 5.1 при необходимости нужно добавить в класс Person поле которе хран
        private int pizzaSlices;

        public Person(int id, String name) {
            this.id = id;
            this.name = name;
        }

        public void eatSlice() {
            this.pizzaSlices++;
        }

        public void sayResult() {
            System.out.println("I'm " + this.name + ", I eat " + this.pizzaSlic
        }

        // 5.2
        public void setPizzaSlices(int pizzaSlices) {
```

```
            this.pizzaSlices = pizzaSlices;
        }
    }
```