МИНИСТЕРСТВО ЦИФРОВОГО РАЗВИТИЯ СВЯЗИ И МАССОВЫХ КОММУНИКАЦИЙ

Ордена Трудового Красного Знамени

Федеральное государственное бюджетное образовательное учреждение высшего образования

«Московский технический университет связи и информатики»

Кафедра «Математическая Кибернетика и Информационные технологии»

Дисциплина «Информационные технологии и программирование»

Лабораторная работа №2

«Введение в ООП Java»

Выполнила:

Студентка группы БВТ2303

Морозова Ольга

Москва
2024

Цель работы:

Изучение понятия ООП, основных его концепций и применение его на практике на языке программирования Java.

Ход работы:

Задание 1. Вариант 10.

1. Задание абстрактного класса.

Создаём базовый абстрактный класс Gadgets. В нём задаём поля (переменные класса) и определяем методы (хотя бы их основы), которые обязательны для всех дочерних классов (*это абстракция*). Обязательно добавляем конструкторы класса (с введёнными данными и по умолчанию), а также геттеры и сеттеры для каждого созданного поля класса. Также именно в этом классе создаём счётчик объектов, в котором будут учитываться все созданные объекты как этого класса, так и его дочерних классов.

Все методы класса имеют модификатор доступа public, что позволяет дочерним классам обращаться к этим методам и использовать их. Но все поля класса имеют модификатор доступа private, ведь это внутренняя информация класса, и к ней не должны иметь доступ напрямую. Именно для взаимодействия с переменными в данном случае и создаются геттеры и сеттеры (*это инкапсуляция*).

```java
package Lab2;

public abstract class Gadgets {  10 usages  3 inheritors
    private int battery;  6 usages
    private String brand;  3 usages
    private String owner;  3 usages
    private static int gadgetsCounter;  4 usages

    // Конструктор инициализации.
    public Gadgets (int battery, String brand, String owner) {  4 usages
        if ((battery >= 0) & (battery <= 100)) {
            this.battery = battery;
        } else {
            this.battery = 0;
        }
        this.brand = brand;
        this.owner = owner;
        gadgetsCounter++;
    }

    // Конструктор по умолчанию.
    public Gadgets () {  no usages
        this( battery: 0,  brand: "None",  owner: "None");
        gadgetsCounter++;
    }
```

```java
    public static int getGadgetsCounter() {  1 usage
        return gadgetsCounter;
    }

    public static void setGadgetsCounter(int gadgetsCounter) {  no usages
        Gadgets.gadgetsCounter = gadgetsCounter;
    }

    // Геттер для уровня заряда батареи гаджета.
    public int getBattery() {  3 usages
        return battery;
    }

    // Сеттер для уровня заряда батареи гаджета.
    public void setBattery(int battery) {  no usages
        if ((battery >= 0) & (battery <= 100)) {
            this.battery = battery;
        } else {
            this.battery = 0;
        }
    }

    // Геттер для названия производителя гаджета.
    public String getBrand() {  no usages
        return brand;
    }
```

```
54      // Сеттер для производителя гаджета.
55      public void setBrand(String brand) {  no usages
56          this.brand = brand;
57      }
58
59      // Геттер для имени владельца гаджета.
60      public String getOwner() {  no usages
61          return owner;
62      }
63
64      // Сеттер для имени владельца гаджета.
65      public void setOwner(String owner) {  no usages
66          this.owner = owner;
67      }
68
69      public abstract void showTime();  1 usage  3 implementations
70
71      public void checkBattery() {  1 usage  2 overrides
72          System.out.println(battery + "%");
73      }
74  }
```

2. Создаём дочерние классы.

Для создания дочерних классов для абстрактного класса Gadgets мы используем extends после имени класса и указываем родительский класс. С помощью super() мы используем конструктор родительского класса.

Дочерний класс Clocks.

Мы определяем абстрактный метод showTime(), который указывали в родительском классе. При этом нам не нужно заново прописывать метод chackBattery, т.к. его базовый функционал, прописанный в родительском классе, нас удовлетворяет. Это и есть *наследование*.

```java
package Lab2;

import java.time.LocalTime;

public class Clocks extends Gadgets {  1 usage
    private LocalTime time;  4 usages

    // Конструктор инициализации.
    public Clocks(int battery, String brand, String owner, LocalTime time) {  1 usage
        super(battery, brand, owner);
        this.time = time;
    }

    // Конструктор по умолчанию.
    public Clocks() {  1 usage
        this( battery: 0,  brand: "None",  owner: "None", LocalTime.now());
    }

    // Геттер для установленного времени на часах.
    public LocalTime getTime() {  no usages
        return time;
    }

    // Сеттер для установленного времени на часах.
    public void setTime(LocalTime time) {  no usages
        this.time = time;
    }

    public void showTime() {  1 usage
        System.out.println(this.time);
    }
}
```

Дочерний класс SmartPhones.

В данном случае мы переопределяем родительский класс checkBattery для дочернего, т.к. хотим, чтобы для данного класса он работал по-другому. Также мы добавляем уникальный метод call(), которого не было в родительском классе.

```java
package Lab2;

import java.time.LocalDate;
import java.time.LocalTime;

public class Smartphones extends Gadgets  {  3 usages
    private LocalTime time;  4 usages
    private LocalDate date;  4 usages

    // Конструктор инициализации.
    public Smartphones(int battery, String brand, String owner, LocalTime time, LocalDate date) {  1 usage
        super(battery, brand, owner);
        this.time = time;
        this.date = date;
    }

    // Конструктор по умолчанию.
    public Smartphones() {  1 usage
        this( battery: 0,  brand: "None",  owner: "None", LocalTime.now(), LocalDate.now());
    }

    // Геттер для установленного времени на смартфоне.
    public LocalTime getTime() {  no usages
        return time;
    }
```

```java
    // Сеттер для установленного времени на смартфоне.
    public void setTime(LocalTime time) {  no usages
        this.time = time;
    }

    // Геттер для установленной даты на смартфоне.
    public LocalDate getDate() {  no usages
        return date;
    }

    // Сеттер для установленной даты на смартфоне.
    public void setDate(LocalDate date) {  no usages
        this.date = date;
    }

    public void showTime() {  1 usage
        System.out.println(this.time);
        System.out.println(this.date);
    }

    @Override  1 usage
    public void checkBattery() {
        System.out.println("Заряд: " + getBattery() + "%");
    }
```

```java
    public void call(String number) {  1 usage
        System.out.println("Звоним абоненту " + number);
    }
}
```

Дочерний класс Laptops.

Как и в предыдущем случае мы переопределяем родительский класс checkBattery для дочернего и добавляем уникальный метод search(), которого не было в родительском классе.

```java
package Lab2;

import java.time.LocalDate;
import java.time.LocalTime;

public class Laptops  extends Gadgets {  3 usages
    private LocalTime time;  4 usages
    private LocalDate date;  4 usages

    // Конструктор инициализации.
    public Laptops(int battery, String brand, String owner, LocalTime time, LocalDate date) {  1 usage
        super(battery, brand, owner);
        this.time = time;
        this.date = date;
    }

    // Конструктор по умолчанию.
    public Laptops() {  1 usage
        this( battery: 0,  brand: "None",  owner: "None", LocalTime.now(), LocalDate.now());
    }

    // Геттер для установленного времени на ноутбуке.
    public LocalTime getTime() {  no usages
        return time;
    }
```

```java
    // Сеттер для установленного времени на ноутбуке.
    public void setTime(LocalTime time) {  no usages
        this.time = time;
    }

    // Геттер для установленной даты на ноутбуке.
    public LocalDate getDate() {  no usages
        return date;
    }

    // Сеттер для установленной даты на ноутбуке.
    public void setDate(LocalDate date) {  no usages
        this.date = date;
    }

    public void showTime() {  1 usage
        System.out.println("Время: " + this.time);
        System.out.println("Дата: " + this.date);
    }

    @Override  1 usage
    public void checkBattery() {
        System.out.println("Заряд: " + getBattery() + "%");
        System.out.println("Оставшееся время работы: " + getBattery() * 5 + "мин");
    }
```

```
    public void search(String query) {  1 usage
        System.out.println("Ищем в интернете слово: " + query);
    }
}
```

3. Создаём класс для работы с остальными.

В классе GadgetsWork мы, наконец, создаём метод main, в котором и будем создавать объекты созданных ранее классов, а также вызывать их методы.

Описывая методы для работы с объектами классов, мы создаём их из родительского абстрактного класса Gadgets, и его же задаём как тип данных в общих для всех методах. Это в итоге позволяет нам вызывать метод, переопределённый для каждого дочернего класса по-своему, используя лишь один метод в нашем текущем классе (*это полиморфизм*).

```
package Lab2;

import java.util.Scanner;

public class GadgetsWork {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        Gadgets clock = new Clocks();
        Gadgets phone = new Smartphones();
        Gadgets laptop = new Laptops();

        System.out.println("Зарегистрировано гаджетов: " + Gadgets.getGadgetsCounter());

        System.out.println("Часы:");
        showGadgetTime(clock);
        checkGadgetBattery(clock);

        System.out.println("Смартфон:");
        showGadgetTime(phone);
        checkGadgetBattery(phone);

        System.out.println("Ноутбук:");
        showGadgetTime(laptop);
        checkGadgetBattery(laptop);
```

```
        String number = scanner.next();
        callUser(number, (Smartphones) phone);

        String query = scanner.next();
        SearchQuery(query, (Laptops) laptop);
    }

    public static void showGadgetTime(Gadgets gadget) {  3 usages
        gadget.showTime();
    }

    public static void checkGadgetBattery(Gadgets gadget) {  3 usages
        gadget.checkBattery();
    }

    public static void callUser(String number, Smartphones phone) {  1 usage
        phone.call(number);
    }

    public static void SearchQuery(String query, Laptops laptop) {  1 usage
        laptop.search(query);
    }
}
```

4. Наконец, тестируем работу программы.

```
"C:\Program Files\Java\jdk-21\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2024.2.1\lib\idea_rt.jar=56335:C:\Program
Зарегистрировано гаджетов: 3
Часы:
21:21:04.352985800
0%
Смартфон:
21:21:04.352985800
2024-09-30
Заряд: 0%
Ноутбук:
Время: 21:21:04.353982600
Дата: 2024-09-30
Заряд: 0%
Оставшееся время работы: 0мин
89153363902
Звоним абоненту 89153363902
ООП
Ищем в интернете слово: ООП

Process finished with exit code 0
```

Вывод:

Мы изучили понятие ООП, основные его концепции, а также написали программу на языке программирования Java с использованием полученных знаний.

GitHub - https://github.com/MiniLynx13/ITaP_Lab2