Plan

2023-10-04

1. Homework Review
2. Object, equals(), toString()

---

1. Разбор домашнего задания
2. Object, equals(), toString()

Theory

▶ **English**

▼ **На русском**

Модификаторы доступа

Модификаторы доступа в Java определяют уровень доступа к переменным, методам и классам. Они нужны для обеспечения инкапсуляции.

В Java есть четыре модификатора доступа:

1. `public`: Доступен отовсюду.
2. `protected`: Доступен внутри пакета и для всех подклассов.
3. `default` (пакетный): Доступен только внутри пакета.
4. `private`: Доступен только внутри класса.

Вот как это выглядит:

Модификатор	Внутри класса	Внутри пакета	В подклассах	Вне пакета
`public`	Да	Да	Да	Да
`protected`	Да	Да	Да	Нет
`default`	Да	Да	Нет	Нет
`private`	Да	Нет	Нет	Нет

Примеры:

Для переменных

```
public int a;      // Доступен отовсюду
protected int b;   // Доступен внутри пакета и в подклассах
int c;             // Доступен внутри пакета (default)
private int d;     // Доступен только внутри класса
```

Для методов

```
public void method1() {}      // Доступен отовсюду
protected void method2() {}   // Доступен внутри пакета и в подклассах
void method3() {}             // Доступен внутри пакета (default)
private void method4() {}     // Доступен только внутри класса
```

Класс `Object` в Java

Класс `Object` является базовым классом для всех объектов в Java. Все классы в Java неявно наследуются от класса `Object`, если они явно не указывают другой родительский класс. Класс `Object` предоставляет ряд методов и функциональности, которые доступны для всех объектов в Java. Вот некоторые из ключевых методов класса `Object`:

Часто используемые методы класса Object

Класс Object предоставляет ряд встроенных методов, таких как `equals()`, `hashCode()`, `toString()`.

Метод `equals()`

Метод `equals()` используется для сравнения двух объектов на равенство. В классе `Object`, этот метод реализован так, что он сравнивает ссылки на объекты, а не их содержимое. Это означает, что два объекта будут считаться равными только в том случае, если они ссылаются на один и тот же объект в памяти. Вот пример:

```
Object obj1=new Object();
        Object obj2=new Object();

        boolean areEqual=obj1.equals(obj2); // false, так как obj1 и obj2 разные объекты
```

Переопределение методов

Чтобы более эффективно использовать методы `equals()`, `hashCode()` и `toString()` для пользовательских классов, их часто переопределяют. В этом случае, вы можете определить собственные правила сравнения, генерации хэш-кода и строкового представления объекта.

Метод `hashCode()`

Метод `hashCode()` возвращает целочисленное значение, называемое хэш-кодом, которое ассоциируется с объектом. По умолчанию, в классе `Object`, этот метод генерирует хэш-код, основанный на внутреннем адресе объекта в памяти. Хэш-коды используются, например, при работе с коллекциями, такими как хеш-таблицы. Важно, чтобы если два объекта равны (согласно методу `equals()`), их хэш-коды также были равны. Вот пример:

```
Object obj=new Object();
        int hashCode=obj.hashCode(); // возвращает хэш-код объекта
```

3. Метод `toString()`

Метод `toString()` возвращает строковое представление объекта. В классе `Object`, этот метод возвращает строку, содержащую имя класса и хеш-код объекта. Он часто переопределяется в пользовательских классах для предоставления более информативного описания объекта. Вот пример:

```
Object obj=new Object();
        String str=obj.toString(); // возвращает строковое представление объекта
```

Пример переопределения `equals()`

```
class MyClass {
    private int value;

    // Конструктор и другие методы класса

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;
        MyClass myClass = (MyClass) o;
        return value == myClass.value;
    }
}
```

Пример переопределения `hashCode()`

```
class MyClass {
    private int value;

    // Конструктор и другие методы класса

    @Override
    public int hashCode() {
        return Objects.hash(value);
    }
}
```

Пример переопределения `toString()`

```
class MyClass {
    private int value;

    // Конструктор и другие методы класса

    @Override
    public String toString() {
        return "MyClass{" +
                "value=" + value +
```

```
                '}';
    }
}
```

Важные моменты equals

- Если `a.equals(b)`, то `b.equals(a)`.
- Если `a.equals(b)` и b.`equals(c)`, то `a.equals(c)`.
- Повторные вызовы `a.equals(b)` возвращают одинаковый результат.
- `a.equals(null)` всегда возвращает `false`.
- Если `a.equals(b)`, то `a.hashCode() == b.hashCode()`.

Задачи для закрепления

1. Создайте класс `Person` с полями `name` и `age`. Переопределите методы `equals()`, `hashCode()`, и `toString()` для этого класса.

2. Создайте список объектов типа `Person` и проверьте, как работают методы `equals()`, `hashCode()` и `toString()` в вашей реализации класса `Person`.

Цель задания

Понять, как и зачем использовать метод `equals` для сравнения объектов в Java на примере моделирования работы банка и банкомата.

Описание процесса

- Пользователь регистрирует банковскую карту в банке, указывая свои имя и фамилию.
- Банк автоматически устанавливает уникальный номер карты на основе имени и фамилии владельца.
- После создания карты, пользователь может воспользоваться банкоматом для различных операций: проверка баланса, снятие и внесение денег.
- Банкомат при каждой операции обращается к банку для проверки, существует ли такая карта, сравнивая все поля карты: номер, баланс, имя и фамилия.

Задачи

1. **Создать класс `BankCard`**
    - Добавить поля для хранения `номера карты`, `баланса`, `имени` и `фамилии владельца`.
    - `номера карты` при создании не должен быть установлен.
    - Реализовать метод `equals`, который будет сравнивать карты по полям `баланс`, `имя` и `фамилия владельца`.
2. **Создать класс `Bank`**
    - Добавить массив (`BankCard[] cards`) для хранения объектов `BankCard`.
    - Реализовать метод `addCard`, который будет добавлять новую карту в массив. Добавлять можно только уникальные карты.

- Номер карты должен устанавливаться автоматически на основе имени и фамилии владельца.
- Реализовать метод `findCard`, который будет искать карту в массиве (`BankCard[] cards`) по объекту `BankCard`, используя метод `equals`.
- Реализовать метод `getCardNumber`, который будет искать карту в массиве (`BankCard[] cards`) по объекту `BankCard`, используя метод `equals` и возвращать номер карты.

3. **Создать класс `ATM`**

   - Добавить поле для хранения объекта `Bank`.
   - Реализовать методы для внесения (`deposit`) и снятия денег (`withdraw`). Эти методы должны использовать метод `findCard` из класса `Bank` для поиска соответствующей карты.
   - Добавить метод отображения текужего баланса текущей карты (`showBalance`).

4. **Создать класс `Main` с методом `main`**

   - Инициализировать объекты `Bank` и `ATM`.
   - Выполненть операций: регистрация карты, внесение, снятие денег, получение информации о текущем балансе и о нормере карты.

Дополнительные задачи

- Добавить проверку на максимальное количество карт в банке.
- Добавить интерфейс для работы с банкоматом и банком (`Scanner`)

Вопросы для обсуждения

- Почему важно переопределить метод `equals`?
- Какие могут быть проблемы, если не переопределить метод `equals`?

Homework

▶ **English**

▼ **На русском**

Задача 1:

Создайте класс `Car` с полями `make`, `model`, `year`, и `vin` (идентификационный номер автомобиля). Переопределите методы `equals()`, `hashCode()` и `toString()` для этого класса. Затем создайте несколько объектов класса Car и проверьте, как работают переопределенные методы.

1. Создайте несколько объектов класса Car с разными параметрами
2. Сравните объекты с использованием метода

Задача 2

Создайте класс `DataTime` для представления даты и времени, с полями `year`, `month`, `day`, `hour` и `minute`.

1. Переопределите метод `toString()` для класса `DateTime` так, чтобы он возвращал строку в формате "ГГГГ-ММ-ДД ЧЧ:ММ".
2. Создайте несколько объектов DateTime и запишите их в массив, с разными данными
3. Создайте ещё один объект DateTime, который будет являться целью для поиска.

Напишите код, который ищет первое совпадение объекта DateTime из пункта 3 в массиве объектов из пункта 2. Используйте метод equals() для сравнения объектов.

Выведите на экран результат поиска, показывая найденное совпадение или сообщение о его отсутствии. **Обратите внимание на формат вывода.**

Code

code/Lesson_21/src/to_string/Auto.java

```
package to_string;

/**
 * @author Andrej Reutow
 * created on 04.10.2023
 */
public class Auto extends Object{

    private String brand;
    private String model;

    public Auto(String brand, String model) {
        this.brand = brand;
        this.model = model;
    }

    @Override
    public String toString(){
        return "Bla bla bla!";
    }
}
```

code/Lesson_21/src/to_string/Person.java

```java
package to_string;

/**
 * @author Andrej Reutow
 * created on 04.10.2023
 */
public class Person {

    private String name;
    private String lastName;

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

    // name: Andrej, lastName: Reutow
    public void sing(){
        System.out.println("I am singing");
    }


    @Override
    public String toString() {
        return "Person{" +
                "name='" + this.name + '\'' +
                ", lastName='" + lastName + '\'' +
                '}';
    }
}
```

code/Lesson_21/src/to_string/Main.java

```java
package to_string;

/**
 * @author Andrej Reutow
 * created on 04.10.2023
 */
public class Main {

    public static void main(String[] args) {
        Auto auto = new Auto("VW", "Golf");
```

```
        auto.equals(null);

        Person person = new Person("Andrej", "Reutow");


        System.out.println(auto); // Auto@5474c6c (взята реализация метода
        //  System.out.println(auto) -> выведет: Bla bla bla!. Так как имен

        System.out.println(person); // name: Andrej, lastname: Reutow (Взят

        print(person);
        print(auto); // выведет: Bla bla bla!. Так как именно так переобпре

        Object personObj = new Person("Andrej", "Reutow");
        if (personObj instanceof Person) {
            ((Person) personObj).sing();

        }

        String string = "";
        string.toString();
        int a;
    }

    public static void print(Object obj) {
        String s = (obj == null) ? "null" : obj.toString();
        System.out.println(s);
    }
}
```

code/Lesson_21/src/equals/intro/IntroToEquals.java

```
package equals.intro;

import java.util.Scanner;

/**
 * @author Andrej Reutow
 * created on 04.10.2023
 */
public class IntroToEquals {

    public static void main(String[] args) {
        int a = 5;
```

```
        int b = 5;
        int c = 5;

        System.out.println(a == b); // true
        System.out.println(b == a); // true

        System.out.println(a == c); // true
        System.out.println(c == a); // true

        System.out.println(c == b); // true

        String str1 = "Hello"; // str1 ссылка - @713, значение "Hello" запи
        String str2 = "Hello"; // str2 ссылка - @713, т.к. в String pool уж

        System.out.println(str1 == str2); //true @713 == @713

        String str3 = new String("Hello"); // создаю новый объект (ключевое

        System.out.println(str1 == str3); // false // str1 ссылка - @713 ,
        System.out.println(str2 == str3); // false // str2 ссылка - @713 ,

//        System.out.println(str1.equals(str2)); // true
//        System.out.println(str1.equals(str3)); // true
//        System.out.println(str2.equals(str3)); // true


        Object someObject = getSomeObject();
//        System.out.println(someObject.equals(str3)); // false - т.к. это
        System.out.println(str3.equals(someObject)); // true
        System.out.println(str3 == someObject); // false - т.к. это ссылки
    }

    public static Object getSomeObject() {
        return new Scanner(System.in);
    }

//    public static Object getSomeObject() {
//        return new String("Hello");
//    }
}
```

code/Lesson_21/src/equals/auto2/Auto2.java

```java
package equals.auto2;

/**
 * @author Andrej Reutow
 * created on 04.10.2023
 */
public class Auto2 {

    private String brand;
    private String model;

    public Auto2(String brand, String model) {
        this.brand = brand;
        this.model = model;
    }


    @Override
    public boolean equals(Object o) {
        if (this == o) { // равны ли ссылки объектов
            return true;
        }
        if (!(o instanceof Auto2)) { // является ли сравниваемый объект (o)
            return false;
        }

        Auto2 auto = (Auto2) o; // приводим объект o к нужному типу, т.е. к

        boolean result = this.brand.equals(auto.brand) && this.model.equals
        return result;

//        return this.brand.equals(auto.brand) && this.model.equals(auto.mo
    }


    @Override
    public String toString() {
        return "Auto2: {" +
                " brand: " + this.brand +
                ", model: " + this.model +
                "}";
```

```
    }
}
```

code/Lesson_21/src/equals/auto2/Main.java

```
package equals.auto2;

import equals.auto2.Auto2;

/**
 * @author Andrej Reutow
 * created on 04.10.2023
 */
public class Main {

    public static void main(String[] args) {
        Auto2 Auto2VwPolo = new Auto2("VW", "POLO");
        Auto2 Auto2VwPolo2 = new Auto2("VW", "POLO");
        Auto2 Auto2MercC200 = new Auto2("Mercedes", "C200");

        System.out.println(Auto2VwPolo);
        System.out.println(Auto2VwPolo2);
        System.out.println(Auto2VwPolo.equals(Auto2VwPolo2)); // true

        System.out.println(Auto2MercC200);
        System.out.println(Auto2VwPolo.equals(Auto2MercC200)); // false

        System.out.println(Auto2VwPolo2.equals(Auto2VwPolo)); // true
        System.out.println(Auto2MercC200.equals(Auto2VwPolo)); // false
    }
}
```

code/Bank/src/ATM.java

```
/**
 * Класс для представления банкомата и выполнения операций с картами.
 */
public class ATM {
    private Bank bank;

    // Конструкторы и геттеры/сеттеры
```

```
    /**
     * Метод для внесения денег на карту.
     *
     * @param card   Карта, на которую вносятся деньги.
     * @param amount Сумма, которую нужно внести.
     */
    public void deposit(BankCard card, double amount) {
        // Реализация метода
    }

    /**
     * Метод для снятия денег с карты.
     *
     * @param card   Карта, с которой снимаются деньги.
     * @param amount Сумма, которую нужно снять.
     */
    public void withdraw(BankCard card, double amount) {
        this.bank.findCard(card);
        // Реализация метода
    }

    /**
     * Метод для отображения баланса карты.
     *
     * @param card Карта, баланс которой нужно отобразить.
     */
    public void showBalance(BankCard card) {
        // Реализация метода
    }
}
```

code/Bank/src/Bank.java

```
/**
 * Класс для представления банка и хранения карт.
 */
public class Bank {
    private BankCard[] cards;
    private int maxCards;
    private int numCards;

    public Bank(int maxCards) {
        this.maxCards = maxCards;
        this.cards = new BankCard[maxCards];
```

```
    }

    // Конструкторы и геттеры/сеттеры

    /**
     * Метод для добавления новой карты в банк.
     *
     * @param card Карта, которую нужно добавить.
     */
    public void addCard(BankCard card) {
        //AndreReutow
        String generatedCardNumber = card.getFirstName() + card.getLastName
        card.setCardNumber(generatedCardNumber);
        card.setBalance(0);
        // todo пройтись по всему массиву и найти пустую ячейку для установ
        // Если пустой ячейки нет, то указать пользователю, что наш банк бо
       // card[] = card;
    }

    /**
     * Метод для поиска карты в банке.
     *
     * @param card Карта, которую нужно найти.
     * @return Если карточка найдена true, в противном случае false
     */
    public boolean findCard(BankCard card) {
        // Реализация метода
        // todo пройтись по массиву и найти карточку. Если карточка найдена
        return false;
    }
}
```

code/Bank/src/BankCard.java

```
/**
 * Класс для представления банковской карты.
 */
public class BankCard {
    private String cardNumber;
    private double balance;
    private String firstName;
    private String lastName;

    public BankCard(String firstName, String lastName) {
```

```
        this.firstName = firstName;
        this.lastName = lastName;
    }


    // Конструкторы и геттеры/сеттеры


    public String getCardNumber() {
        return cardNumber;
    }

    public void setCardNumber(String cardNumber) {
        this.cardNumber = cardNumber;
    }

    public double getBalance() {
        return balance;
    }

    public void setBalance(double balance) {
        this.balance = balance;
    }

    public String getFirstName() {
        return firstName;
    }

    public void setFirstName(String firstName) {
        this.firstName = firstName;
    }

    public String getLastName() {
        return lastName;
    }

    public void setLastName(String lastName) {
        this.lastName = lastName;
    }

    /**
     * Метод для сравнения карт по балансу, имени и фамилии владельца.
     *
```

```
         * @param otherCard Карта, с которой сравниваем текущую карту.
         * @return true, если карты равны по балансу, имени и фамилии; в против
         */
        @Override
        public boolean equals(Object otherCard) {
            // Реализация метода
            return false;
        }
    }
```

code/Bank/src/Main.java

```
    /**
     * Главный класс для выполнения операций с банком и банкоматом.
     */
    public class Main {
        public static void main(String[] args) {
            Bank bank = new Bank(50);

            BankCard bankCard = new BankCard("Andre", "Reutow");
            bank.addCard(bankCard);

            ATM atm = new ATM();
            atm.withdraw(bankCard, 100);
            atm.deposit(bankCard, 100);
            atm.showBalance(bankCard);
        }
    }
```