Plan

2023-10-05

1. Homework Review
2. The static modifier
3. Running a Java application from the console
4. Application Arguments

---

1. Разбор домашнего задания
2. Модификатор static
3. Запуск Java-приложения из консоли
4. Аргументы приложения

Theory

▶ **English**

▼ **На русском**

Модификатор `static`

Модификатор `static` в Java применяется к переменным, методам и внутренним классам, чтобы они стали связанными с классом, а не с конкретными экземплярами класса. Вот как это работает:

1. Статические переменные (поля)

- Статические переменные принадлежат классу, а не экземплярам класса. Они общие для всех объектов этого класса.
- Пример:

```
class MyClass {
    static int count = 0; // Статическая переменная
}
```

2. Статические методы

- Статические методы тоже принадлежат классу, а не экземплярам. Они могут быть вызваны без создания объекта класса.
- Пример:

```
class MathUtils {
    static int add(int a, int b) {
        return a + b;
    }
}
```

3. Статические блоки инициализации

- Статические блоки инициализации выполняются при загрузке класса и могут использоваться для инициализации статических переменных.
- Пример:

```
class MyClass {
    static {
        // Этот блок выполняется при загрузке класса
        System.out.println("Класс MyClass загружен.");
    }
}
```

Модификатор `final` в Java

Модификатор `final` в Java может применяться к переменным, методам и классам. Он указывает на то, что элемент не может быть изменен после его инициализации (присвоения значения переменной, определения метода или создания класса). Вот как `final` применяется в различных контекстах:

1. `final` для переменных:

- **Статические константы:** Переменные, объявленные как `final`, могут быть использованы для создания статических констант, которые остаются неизменными после их определения.

```
public class MathConstants {
    static final double PI = 3.14159265359; // Статическая константа
 }
```

- **Локальные переменные:** Локальные переменные, объявленные как `final`, требуют, чтобы их значение было установлено только один раз и не могло быть изменено.

```
public void process(int value) {
    final int limit = 10; // Локальная константа
    // ...
 }
```

Статические переменные (поля):

1. **Общие данные для всех объектов класса:** Статические переменные используются, когда нужно хранить данные, которые должны быть общими для всех экземпляров класса. Например, счетчики объектов, общие настройки или константы.

```
class MyClass {
    static int count = 0; // Статическая переменная для подсчета экземпл
}
```

2. **Константы:** Статические переменные могут использоваться для хранения констант, которые не должны изменяться и могут использоваться без создания экземпляров класса.

```
class MathConstants {
    static final double PI = 3.14159265359; // Статическая константа
}
```

Статические методы:

1. **Утилитарные операции:** Статические методы часто используются для реализации утилитарных функций, которые не зависят от состояния объекта и могут быть вызваны без создания экземпляра класса.

```
class StringUtils {
    static boolean isEmpty(String str) {
        return str == null;
    }
}
```

3. **Математические операции:** В классах, связанных с математикой или физикой, статические методы могут предоставлять математические операции без создания объектов.

```
class MathUtils {
    static double calculateDistance(Point p1, Point p2) {
        // Вычисление расстояния между точками
        // ...
    }
}
```

Пример:

```
public class MathUtils {

    static int add(int a, int b) { // объявление статического метода с моди
        return a + b;
    }
}
```

```
public class Application {

    public static void main(String[] args) {
        int result = MathUtils.add(1, 3); // Вызов статического метода
        System.out.println(result); // 4
    }
}
```

Правила обращения к статическим методам и переменным в Java включают в себя следующие основные аспекты:

1. Доступ к статическим переменным:

- Статические переменные доступны из любого места в коде, где виден класс, к которому они принадлежат.

```
class MyClass {
    static int count = 0; // Статическая переменная
}
```

- Для доступа к статической переменной используется имя класса, за которым следует точка и имя переменной.

```
int myValue=MyClass.count; // Получение значения статической переменной
```

2. Изменение статических переменных:

- Статическую переменную можно изменять так же, как и любую другую переменную, но изменения будут видны для всех экземпляров класса и из любого места, где виден класс.

```
MyClass.count=10; // Изменение значения статической переменной
```

3. Доступ к статическим методам:

- Статические методы также доступны из любого места в коде, где виден класс и метод, к которому они принадлежат.

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

- Для **вызова статического метода** используется имя класса, за которым следует точка и имя метода.

```
int result=MathUtils.add(5,3); // Вызов статического метода
```

4. Ограничения статических элементов:

- Статические элементы **не имеют доступа к нестатическим** (обычным) элементам класса. То есть, они не могут использовать нестатические переменные или вызывать нестатические методы напрямую без создания объекта.

```
class MyClass {
    int value; // Обычная переменная

    static void doSomething() {
        // Нельзя обратиться к value напрямую
        // value = 10; // Это вызовет ошибку компиляции
    }
}
```

На заметку, важно!

- Важно помнить, что статические переменные и методы принадлежат классу, а не конкретным объектам, поэтому они доступны без создания экземпляров класса. Однако их использование должно быть оправданным и соответствовать требованиям конкретной задачи.

- Это означает, что изменения в статической переменной видны для всех экземпляров класса, и статические методы можно вызывать создания экземпляров класса. Они часто используются для реализации общих и универсальных функций, которые не требуют доступа к данным конкретных объектов.

- Статические элементы могут вызывать другие статические методы и обращаться к статическим переменным.

- Статические элементы **не имеют доступа к нестатическим** (обычным) элементам класса. То есть, они не могут использовать нестатические переменные или вызывать нестатические методы напрямую без создания объекта.

- Статические элементы доступны без создания экземпляра класса и могут быть вызваны из статического контекста, например, из другого статического метода или через имя класса.

- Обращение к статическим переменным и методам является удобным способом хранения общих данных и функций, которые не зависят от состояния объектов. Они часто используются для создания статических констант, утилитарных методов и фабричных методов.

2. `final` для методов:

- **Запрет переопределения:** Методы, объявленные как `final`, не могут быть переопределены в подклассах. Это полезно, когда вы хотите зафиксировать реализацию метода в классе и предотвратить ее изменение.

```
class Parent {
    final void doSomething() {
        // Этот метод нельзя переопределить в подклассах
    }
}
```

Использование модификатора `final` имеет несколько важных преимуществ:

- Гарантия неизменности: Позволяет гарантировать, что значение переменной, реализация метода или класс остаются неизменными.
- Оптимизация: Компилятор может выполнять оптимизации при работе с `final` элементами, так как он знает, что их значение не изменится.
- Защита от переопределения: Защищает методы от случайного или нежелательного переопределения.

Однако следует использовать модификатор `final` осторожно, так как он ограничивает изменение элементов, и это может привести к жестким ограничениям в коде. Выбор использования `final` должен зависеть от требований вашей программы и дизайна классов.

Запуск Java-приложения из консоли

Для одного класса:

1. Сохраните свой код в файл с расширением `.java`. Например, `MyApp.java`.

2. Откройте командную строку (консоль) и перейдите в каталог, где находится ваш файл `.java`.

3. Компилируйте класс с помощью команды `javac`, например:

```
javac MyApp.java
```

4. Запустите приложение с помощью команды `java`, указав имя класса с методом `public static void main(String[] args)`:

```
java MyApp
```

Приложение с пакетами

Приложение состоит из двух пакетов: `javac` и `javac.model`, а также двух классов: `MyApp` и `Auto`. Давайте рассмотрим, как его скомпилировать и запустить из командной строки.

Допустим, ваш проект имеет следующую структуру:

```
myproject/
    src/
        javac/
            MyApp.java
        javac/model/
            Auto.java
```

Вот как скомпилировать и запустить ваше приложение:

1. Откройте командную строку (консоль) и перейдите в корневую папку вашего проекта `myproject`.

2. Скомпилируйте ваше приложение с помощью команды `javac`, указав путь к корневой папке `src`:

```
javac -d . src/javac/*.java src/javac/model/*.java
```

3. После успешной компиляции, перейдите в папку `myproject`, где находятся скомпилированные файлы `.class`.

```
cd ..
```

4. Теперь вы можете запустить ваше приложение, указав полное имя класса `MyApp`:

```
java -cp . javac.MyApp
```

После выполнения этой команды, ваше приложение запустится, и вам будет предложено ввести марку и модель автомобиля. Затем оно выведет "Bla bla bla!", так как метод `toString()` в классе `Auto` всегда возвращает эту строку.

Убедитесь, что вы выполняете эти команды из корневой папки вашего проекта, и у вас должно получиться успешно скомпилировать и запустить ваше Java-приложение.

Аргументы приложения

В методе `public static void main(String[] args)`, параметр `args` представляет собой массив строк, который содержит аргументы, переданные при запуске приложения из командной строки. Пример:

```
public class MyApp {
    public static void main(String[] args) {
        for (int i = 0; i < args.length; i++) {
            String arg = args[i];
            System.out.println("Аргумент: " + arg);
        }
    }
}
```

При запуске приложения с аргументами:

```
java MyApp arg1 arg2 arg3
```

Вы увидите:

```
Аргумент: arg1
Аргумент: arg2
Аргумент: arg3
```

Это позволяет передавать параметры в приложение из командной строки и использовать их в вашей программе.

Homework

▶ **English**

▼ **На русском**

Задача 1

- Написать класс AppContants для хранения не изменяемых значений для приложения Bank.
- использовать переменные класса AppContants в приложении

Класс должен содержать переменные которые необходимы для приложения:

- количество карт, которые хранит банк `DEFAULT_CARDS_SIZE`
- добавить конструктор в класс Bank, и использовать для инициализации массива эту переменную
- сообщения об ошибках к примеру *"У вас не достаточно средств для выполнения этой операции"*, *"Банк больше не принимает новых клиентов"* и т.п.
- подумать какие еще значения можно записать в класс AppContants
- добавить для каждой новой переменной javadoc

java doc: `/** */`

```
/**
 * тут нужно описать что делает переменная или метод
 */
```

Задача 2

Расширить функционал приложения:

Написать статические методы для чтения ввода с консоли используя Scanner.

- определить отдельный пакет `menu` где будут храниться классы отвечающие за работу с меню
- создать класс `ConsoleInput` в пакете `menu` который будет содержать статические методы: `public static ...`
  - чтение и возврат целочисленого значение `public static int readInt()`
  - чтение и возврат строчного значение `public static String readLine()`
  - добавть другие методы, если необходимы
  - протестировать работу методов класса `ConsoleInput` в main.

Задача 3

Придумать пункты меню необходимые для нашего приложения Bank, учитывая только текущий функционал приложения.

- опубликовать свое предложение пунктов меню в нашей группе в слаке, с описанием каждого пункта
- предложение должно быть адекватным, т.е. вы должны иметь представление как это реализовать

Code

code/ClassWork_22/src/javac/MyApp.java

```
package javac;


import javac.model.Auto;

import java.util.Scanner;

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

    public static void main(String[] args) {
        System.out.println(args[0]);
        System.out.println(args[1]);

        Scanner scanner = new Scanner(System.in);

        System.out.println("Auto brand:");
        String brand = scanner.nextLine();
        System.out.println("Auto model:");
        String model = scanner.nextLine();

        Auto auto = new Auto(brand, model);

        System.out.println(auto);
    }

}
```

code/ClassWork_22/src/javac/model/Auto.java

```
package javac.model;

/**
 * @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_22/src/MyApp.java

```
import java.util.Arrays;

/**
 * @author Andrej Reutow
 * created on 05.10.2023
 */
public class MyApp {
    public static void main(String[] args) {

        System.out.println(Arrays.toString(args));
        if (args.length > 0) {
            System.out.println(args[0]);
        }
        if (args.length > 2) {
            System.out.println(args[1]);
        }

        System.out.println("Hello world!");
    }
}
```

code/Lesson_22/src/calculator/Calculator.java

```
package calculator;
```

```java
/**
 * @author Andrej Reutow
 * created on 05.10.2023
 */
public class Calculator { // utility class

    private static final double PI = 3.1415926;

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

    public static int subtract(int a, int b) {
        return a - b;
    }

    public static int multiply(int a, int b) {
        return a * b;
    }

    public static double circleArea(int radius) {
        // pi * (r * r)
        return PI * (radius * radius);
    }
}
```

code/Lesson_22/src/calculator/CalculatorApp.java

```java
package calculator;

/**
 * @author Andrej Reutow
 * created on 05.10.2023
 */
public class CalculatorApp {

    public static void main(String[] args) {
        System.out.println(Calculator.add(1, 2));       // 3
        System.out.println(Calculator.subtract(1, 3)); // -2
        System.out.println(Calculator.multiply(2, 3)); // 6

        System.out.println(Calculator.circleArea(6)); //
```

```
    }
}
```

code/Lesson_22/src/array_tool/ArrayTools.java

```
package array_tool;

import constants.AppConstants;

/**
 * @author Andrej Reutow
 * created on 05.10.2023
 */
public class ArrayTools {

    private ArrayTools() {
    }

    public static void printArray(Object[] array) {
        for (int i = 0; i < array.length; i++) {
            System.out.println(array[i].toString() + "\t");
        }
    }

    public static boolean findElement(Object target, Object[] array) {
        for (int i = 0; i < array.length; i++) {
            if (array[i].equals(target)) {
                return true;
            }
        }

        System.out.println(AppConstants.ELEMENT_NOT_FOUND_MSG);
        return false;
    }
}
```

code/Lesson_22/src/array_tool/ArrayToolApp.java

```
package array_tool;

import static_mod.Car;

/**
 * @author Andrej Reutow
```

```
 * created on 05.10.2023
 */
public class ArrayToolApp {

    public static void main(String[] args) {
        Car car1 = new Car("VW", "POLO", 1988, "von_001_2");
        Car car2 = new Car("VW", "Golf", 1988, "von_001_3");
        Car car3 = new Car("Audi", "TT", 2022, "von_001_5");
        Car car4 = new Car("VW", "POLO", 1988, "von_001_2");

        Object target = new Object();

        Car[] cars = {car1, car2, car3, car4};

        ArrayTools.printArray(cars);

        Integer[] ints = {1, 2, 3, 4, 5, 6};
        ArrayTools.printArray(ints);

        System.out.println(ArrayTools.findElement(car4, cars));
        System.out.println(ArrayTools.findElement(target, cars));
    }
}
```

code/Lesson_22/src/constants/AppConstants.java

```
package constants;

/**
 * @author Andrej Reutow
 * created on 05.10.2023
 */
public class AppConstants {

    public static final String ERROR_MESSAGE = "This is error message";

    public static final String ELEMENT_NOT_FOUND_MSG = "Element not exist";
}
```

code/Lesson_22/src/static_mod/Car.java

```java
package static_mod;

import java.util.Objects;
import java.util.Random;

/**
 * @author Andrej Reutow
 * created on 05.10.2023
 */

public class Car {

    private static int counter = 0;

    private int id;
    private String make;
    private String model;
    private int year;
    private String vin;

    static {
        System.out.println("Статичный блок");
        System.out.println("Создается объект n " + (counter + 1));
    }

    {
        Random random = new Random();
        this.id = random.nextInt(1000);
    }


    public Car(String make, String model, int year, String vin) {
        System.out.println("Конструктор класса Car");
        this.make = make;
        this.model = model;
        this.year = year;
        this.vin = vin;
//        this.id = ++Car.counter;
    }


    public static int getCounter() {
        return counter;
```

```
    }

    public int getId() {
        return id;
    }

    public String getMake() {
        return make;
    }

    public void setMake(String make) {
        this.make = make;
    }

    public String getModel() {
        return model;
    }

    public void setModel(String model) {
        this.model = model;
    }

    public int getYear() {
        return year;
    }

    public void setYear(int year) {
        this.year = year;
    }

    public String getVin() {
        return vin;
    }

    public void setVin(String vin) {
        this.vin = vin;
    }

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || !(o instanceof Car)) return false;
```

```
        Car car = (Car) o;

        if (year != car.year) return false;
        if (!Objects.equals(make, car.make)) return false;
        if (!Objects.equals(model, car.model)) return false;
        return Objects.equals(vin, car.vin);
    }

    @Override
    public int hashCode() {
        int result = make != null ? make.hashCode() : 0;
        result = 31 * result + (model != null ? model.hashCode() : 0);
        result = 31 * result + year;
        result = 31 * result + (vin != null ? vin.hashCode() : 0);
        return result;
    }

    @Override
    public String toString() {
        return "Car{" +
                "make='" + make + '\'' +
                ", model='" + model + '\'' +
                ", year=" + year +
                ", vin='" + vin + '\'' +
                '}';
    }
}
```

code/Lesson_22/src/static_mod/Main.java

```
package static_mod;

/**
 * @author Andrej Reutow
 * created on 05.10.2023
 */
public class Main {

    public static void main(String[] args) {
//        System.out.println(Car.getCounter()); // вызов статичного метода
        Car car1 = new Car("VW", "POLO", 1988, "von_001_2");

//        System.out.println(Car.getCounter()); // 2
        Car car2 = new Car("VW", "Golf", 1988, "von_001_3");
```

```
//        System.out.println(Car.getCounter()); // 3
        Car car3 = new Car("Audi", "TT", 2022, "von_001_5");

//        System.out.println(Car.getCounter()); // 4
        Car car4 = new Car("VW", "POLO", 1988, "von_001_2");

        System.out.println();
        System.out.println(car1.getId()); // 2
        System.out.println(car2.getId()); // 3
        System.out.println(car3.getId()); // 4
        System.out.println(car4.getId()); // 5
    }
}
```