

Инкапсуляция

НАШИ ПРАВИЛА



Включенная камера



Вопросы по поднятой руке



Не перебиваем друг друга



Все вопросы, не связанные с тематикой курса (орг-вопросы и т. д.), должны быть направлены куратору



Подготовьте свое рабочее окружение для возможной демонстрации экрана (закройте лишние соцсети и прочие приложения)

ЦЕЛЬ

Понять принципы инкапсуляции и необходимость ее использования

ПЛАН ЗАНЯТИЯ

■ ПРОБЛЕМА ОТКРЫТОСТИ
СОСТОЯНИЯ

■ ПРИВАТНЫЕ ПОЛЯ

■ ПРИВАТНЫЕ МЕТОДЫ

■ GETTERS, SETTERS

■ ПАКЕТЫ, ПУБЛИЧНЫЙ ДОСТУП,
УРОВНИ ДОСТУПА

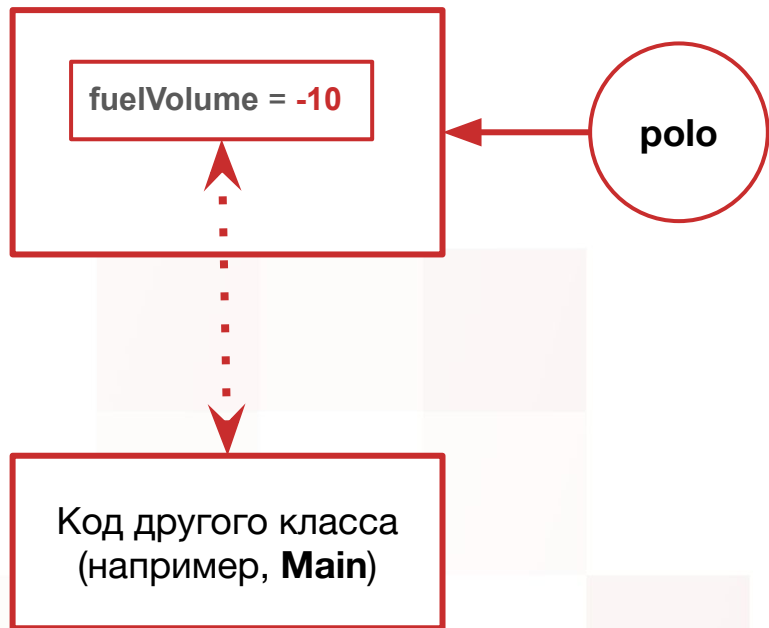
■ ИНКАПСУЛЯЦИЯ

ПРОБЛЕМА ОТКРЫТОСТИ СОСТОЯНИЯ


Сейчас внутреннее состояние объектов класса `Car` доступно извне. Следовательно, вызывающий код класса `Main` может изменять и читать значения полей `Car` напрямую:

```
class Car {  
    double fuelVolume = 0; // объем топлива  
    // ...  
}
```

```
Car polo = new Car(45, 6);  
polo.fuelVolume = -10;
```



СТАВИМ ПЛЮС, ЕСЛИ ПОНЯТНО



Открытые поля могут стать причиной
некорректного состояния объектов

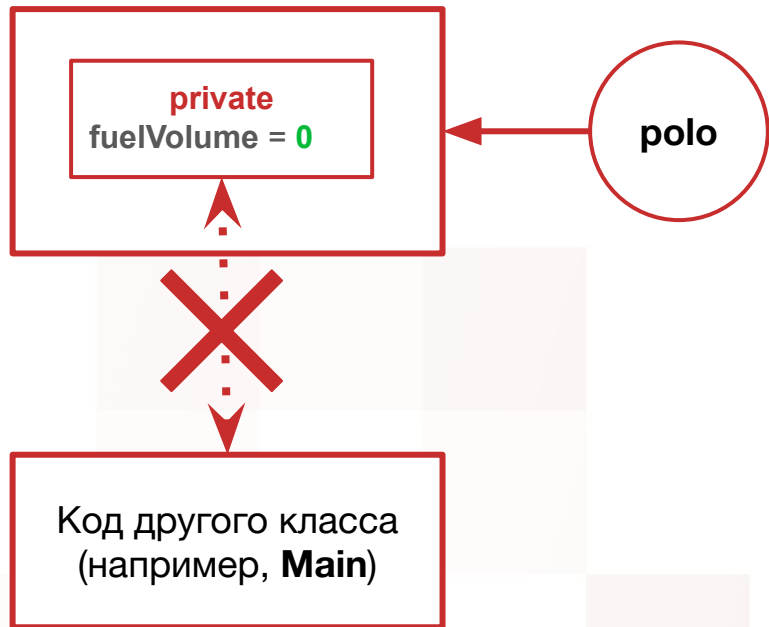
ПРИВАТНЫЕ ПОЛЯ

Модификатор доступа **private** позволяет закрыть поле внутри класса и сделать его недоступным для чтения и записи извне. Таким образом, данный модификатор обеспечивает **доступ только на уровне класса**.

```
class Car {  
    private double fuelVolume = 0;  
  
    // ...  
}
```

```
Car polo = new Car(45, 6);
```

```
// ошибка компиляции  
polo.fuelVolume = -10;
```



СТАВИМ ПЛЮС, ЕСЛИ ПОНЯТНО



Модификатор `private`

ПРИВАТНЫЕ МЕТОДЫ

Также данный модификатор можно применять и для методов, чтобы “спрятать” некоторые вспомогательные методы. Такие методы должны использоваться только внутри класса, где они описаны.

```
Car polo = new Car(45, 6);
```

```
// ошибка компиляции  
polo.calcFuel(5);
```

```
class Car {  
    private double fuelVolume = 0;  
  
    // ...  
  
    boolean go(double kilometers) {  
        double fuelNeeded =  
            calcFuel(kilometers);  
  
        // ...  
    }  
  
    // метод для расчета необходимого топлива  
    private double calcFuel(double kilometers) {  
        return (kilometers * fuelConsumption) / 100;  
    }  
}
```

СТАВИМ ПЛЮС, ЕСЛИ ПОНЯТНО



Создание и применение приватных методов, их назначение

GETTERS

Использование модификатора `private` закрывает также доступ к чтению поля. Для получения его значения следует реализовать **метод доступа `getter`**

```
Car polo = new Car(45, 6);  
polo.refuel(30);
```

```
// теперь можем получить значение  
System.out.println(polo.getFuelVolume());
```

```
class Car {  
    private double fuelVolume = 0;  
  
    // ...  
  
    double getFuelVolume() {  
        return fuelVolume;  
    }  
}
```

СТАВИМ ПЛЮС, ЕСЛИ ПОНЯТНО



Использование `get`-методов доступа

SETTERS

Аналогичным образом для изменения поля необходимо реализовать **setter** с дополнительными проверками корректности значений

```
Car polo = new Car(45, 6);  
polo.refuel(30);
```

```
// сообщение об ошибке  
polo.setFuelVolume(-10);
```

```
// установка значения  
polo.setFuelVolume(10);
```

```
class Car {  
    private double fuelVolume = 0;  
  
    // ...  
  
    void setFuelVolume(double fuelVolume) {  
        if (fuelVolume <= 0) {  
            System.err.println("Некорректное значение");  
        } else {  
            this.fuelVolume = fuelVolume;  
        }  
    }  
}
```

СТАВИМ ПЛЮС, ЕСЛИ ПОНЯТНО



Использование set-методов доступа

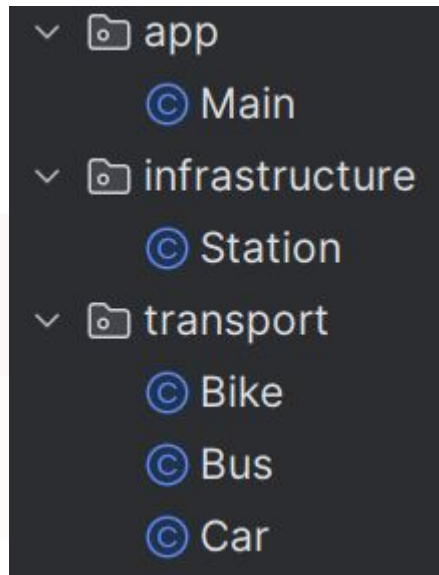
ПАКЕТЫ, ПУБЛИЧНЫЙ ДОСТУП

Пакеты представляют собой механизм для группировки связанных классов.

Если для поля, метода (их также называют **членами класса**) или конструктора **не указывается модификатор доступа**, то к этим элементам нет доступа для классов из другого пакета. Например, классы `Station` и `Main` не смогут обратиться к методу `refuel()` класса `Car`

При этом каждый из классов пакета `transport` имеет доступ к членам других классов и конструкторам этого пакета.

В случае, если мы хотим, чтобы элемент класса был доступен в любом пакете, необходимо определить для него **уровень доступа открытый** с помощью модификатора доступа **`public`**



СТАВИМ ПЛЮС, ЕСЛИ ПОНЯТНО

Пакеты

Отсутствие модификатора доступа

public модификатор

МОДИФИКАТОРЫ И УРОВНИ ДОСТУПА

МОДИФИКАТОР	КЛАСС	ПАКЕТ	ДЛЯ ВСЕХ
public	Да	Да	Да
<i>отсутствует</i>	Да	Да	Нет
private	Да	Нет	Нет

СТАВИМ ПЛЮС, ЕСЛИ ПОНЯТНО



Модификатор доступа



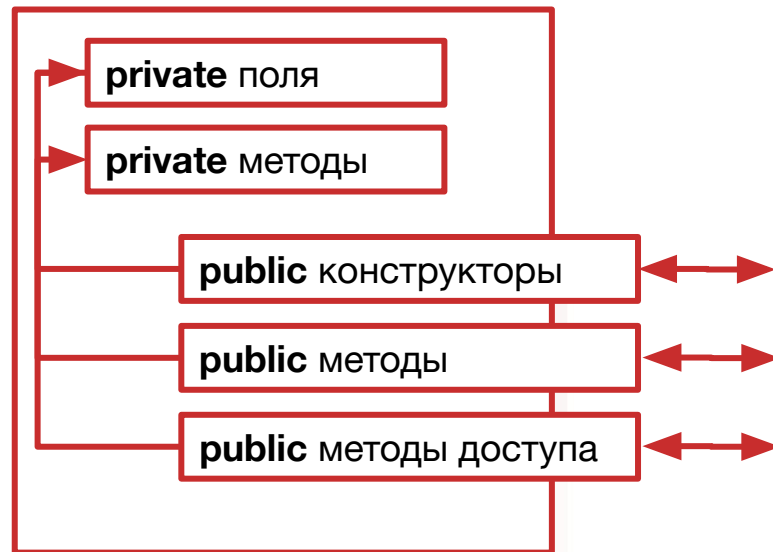
Уровень доступа

ИНКАПСУЛЯЦИЯ

Инкапсуляция - принцип ООП, который определяет по крайней мере два момента, относящихся к созданию и использованию классов.

Инкапсуляция позволяет объединить состояние (поля) и поведение (методы) в одной сущности - классе.

Инкапсуляция в Java позволяет защитить внутреннее состояние и предоставить к нему только контролируемый доступ за счет публичных методов (в том числе методов доступа) и конструкторов



СТАВИМ ПЛЮС, ЕСЛИ ПОНЯТНО



Структура класса с использованием
инкапсуляции

ПОИГРАЕМ ;)

■ Инкапсуляция

■ Модификаторы доступа и уровни доступа

■ Методы доступа

ДОМАШНЕЕ ЗАДАНИЕ

1. Указать модификаторы доступа для полей и внутренних методов вашей прошлой домашки (CurrencyConverter/Pokemon). Убедиться, что все связанные с сущностью методы помещены в класс.

Дополнительно: сделайте два пакета и распределите по этим пакетам свои классы.

2. По структуре справа создайте пакеты и классы в соответствующих файлах.
 - Классы должны быть все публичные (чтобы их можно было импортировать в другой пакет)
 - Задать поля и конструктор для каждого класса
 - Создайте вне пакета l17 файл L17Main.java и в нем запускайте программу
 - Все реализуемые классы должны обладать методом toString
 - Повторить для всех классов то, что сделано с Man классом (пример в чате)

```
l17
├── animal
│   ├── Cat.java
│   ├── Cow.java
│   └── Dog.java
├── people
│   ├── Man.java
│   └── Woman.java
└── transport
    ├── Bike.java
    ├── Bus.java
    └── Car.java
```



Ваша новая IT-профессия – Ваш новый уровень жизни

Программирование с нуля в
немецкой школе AIT TR GmbH