Занятие 2

ООП

Подпрограммы (методы)

Подпрограмма - именованная часть кода, выполняющая определенные действия

В Java называются “Метод”

Могут или возвращать значение, или не возвращать

Метод main

```
public static void main(String[] args) {
    System.out.println("Hello, Ylab!");
}
```

String[] args - параметры командной строки

Метод main

```
public class Main {

  public static void main(String[] args) {

    System.out.println("Hello, " + args[0]);

  }

}
```

```
> java Main World
Hello, World!
```

Методы

Методы могут принимать (а могут не принимать) значения, могут возвращать (а могут и не возвращать) значения

```
public static int sum(int x, int y){
    return x + y;
}
```

```
public static void printRand(){
    System.out.println(Math.random());
}
```

```
public static int sqr(int x) {
    return x * x;
}
```

```
public static void print(int x) {
    System.out.println(x);
}
```

Live Coding Section

Написать метод, который принимает массив целых чисел и возвращает true если массив является отсортированным по возрастанию, false - в противном случае

Написать метод, принимающий одно или несколько значений, и возвращающий true если переданные значения переданы в порядке возрастания

Класс и объект

Объект

Сильно упрощая, объект - это данные и методы работы с ними.

Данные обычно представляются переменными

Методы работы с данными - методы

Класс

Класс представляет собой “чертеж” объекта, который описывает его структуру.

Объект - это экземпляр класса

Пример:

“Автомобиль” - это класс

“Шестерка дяди Коли из соседнего подъезда” - это объект, экземпляр класса “Автомобиль”

Класс в Java

```
class Car {
    String name; // переменная
    double price; // переменная

    void printInfo() { // метод объекта
        System.out.println("Автомобиль " +
                           name + " стоит " + price);
    }

}
```

Создание экземпляра класса

Создание экземпляра класса (или создание объекта) выполняется при помощи ключевого слова **new** и имени класса. Значение в переменные класса можно сохранить обратившись через точку

```
Car myCar = new Car();
myCar.name = “Audi”;
myCar.price = 1000000.00;
myCar.printInfo();
```

```
Car otherCar = new Car();
otherCar.name = “Volvo”;
otherCar.price = 90000.00;
otherCar.printInfo();
```

Создание экземпляра класса. Конструктор

Когда необходимо добавить какую-либо логику при создании экземпляра, используется конструктор. Конструктор - как и метод, может принимать некоторые значения, но он ничего не возвращает.

У класс может быть несколько конструкторов. Пустой конструктор (без параметров) называется “конструктор по умолчанию”

Конструктор

```
class Car {
     String name; // переменная
     double price; // переменная

     Car(String name, double price) { // конструктор с двумя параметрами
          this.name = name;
          this.price = price;
     }

     void printInfo() { // метод объекта
          System.out.println("Автомобиль " +
                                   name + " стоит " + price);
     }

}

 Car car = new Car(“Lada”, 10000.00);
```

Пара слов о static

Ключевое слово **static** применяется к переменным класса и методам. Переменная класса или метод, не требуют создания экземпляра объекта для использования.

```
Math.PI; // переменная класса Math содержит значение числа pi
Math.sqrt(4.0); // статический (static) метод класса Math
Main.main(String[] args); // статический метод запуска Java
                          // программы (точка входа)
```

null

```
Car car = new Car(“Lada”, 10000.00);
```

Происходит следующее:
1. Создается объект в памяти
2. Ссылка на объект в памяти сохраняется в указатель car. Можно считать, что имя переменной является указателем на объект

null - это специальное значение переменной (указателя), означающее, что оно **не связано** ни с каким экземпляром объекта в памяти.

Если попытаться обратиться к переменной, которая не связана с объектом памяти, будет ошибка **NullPointerException**.

Такого поведения можно избежать, добавив проверку на null

```
if (car == null){}
```
или
```
if (car != null) {}
```

Принципы ООП

Принципы ООП

- **Инкапсуляция**. Принцип инкапсуляции говорит о том, что внутреннее устройство объектов должно быть скрыто, а использование организовано через специально выделенные методы
- **Наследование**. Механизм создания новых классов на базе существующих. Класс-наследник имеем все те же методы, что и класс-родитель + свои собственные
- **Полиморфизм**. Способность работать с объектами разных типов. Программа не требует информации, работает ли она с родительскими классами, или же с их наследниками. Программа лишь вызывает метод, а конкретная реализация выбирается исходя из класса объекта

Наследование в Java

Наследование классов организовано с помощью ключевого слова extends. **Наследоваться можно максимум от одного класса!**
В Java все объекты неявно наследуются от java.lang.Object

```
class Hero {
  String name;
}

class MovieHero extends Hero{
  String movie;
}
```

```
MovieHero movieHero = new MovieHero();
movieHero.name = "Neo";
movieHero.movie = "Matrix";
```

Наследование в Java

Для обращения к другим членам класса используется **this**

Для обращения к членам родительского класса **super**

Это применимо к переменным, методам, конструкторам

```
class Hero {
  String name;
}
```

```
class MovieHero extends Hero{
  String movie;

  void printInfo() {
    System.out.println("Герой " + super.name +
       " был в фильме " + this.movie);
  }
}
```

Инкапсуляция в Java

Для обеспечения инкапсуляции (сокрытия реализации) для переменных и методов используются модификаторы доступа:

- **public**. Доступен из любого места программы
- **private**. Доступен только из кода того же класса
- protected. Доступен из текущего класса, текущего пакета или из наследника
- default (‘’). Доступен из текущего пакета

public - для членов, используемых “снаружи”, private - для “внутренних” членов класса

Getter/Setter

```
class Student {
  private String name;

  public String getCourse() {
    return course;
  }

  public void setCourse(String course) {
    // дополнительная проверка
    this.course = course;
  }
}
```

Полиморфизм

```
class Shape {
  public void print() {
    System.out.println("Я не знаю кто я");
  }
}
class Triangle extends Shape {
  public void print() {
    System.out.println("Я треугольник");
  }
}
class Circle extends Shape {
  public void print() {
    System.out.println("Я круг");
  }
}
```

```
Shape shape = new Triangle();
shape.print();// Я треугольник
shape = new Circle();
shape.print();// Я круг
```

В данном примере примере говорят, что метод print является **переопределенным**

Полиморфизм

```
class Shape {
  public void print() {
    System.out.println("Я не знаю кто я");
  }

  public void print(String message) {
    System.out.println(message);
  }
}
```

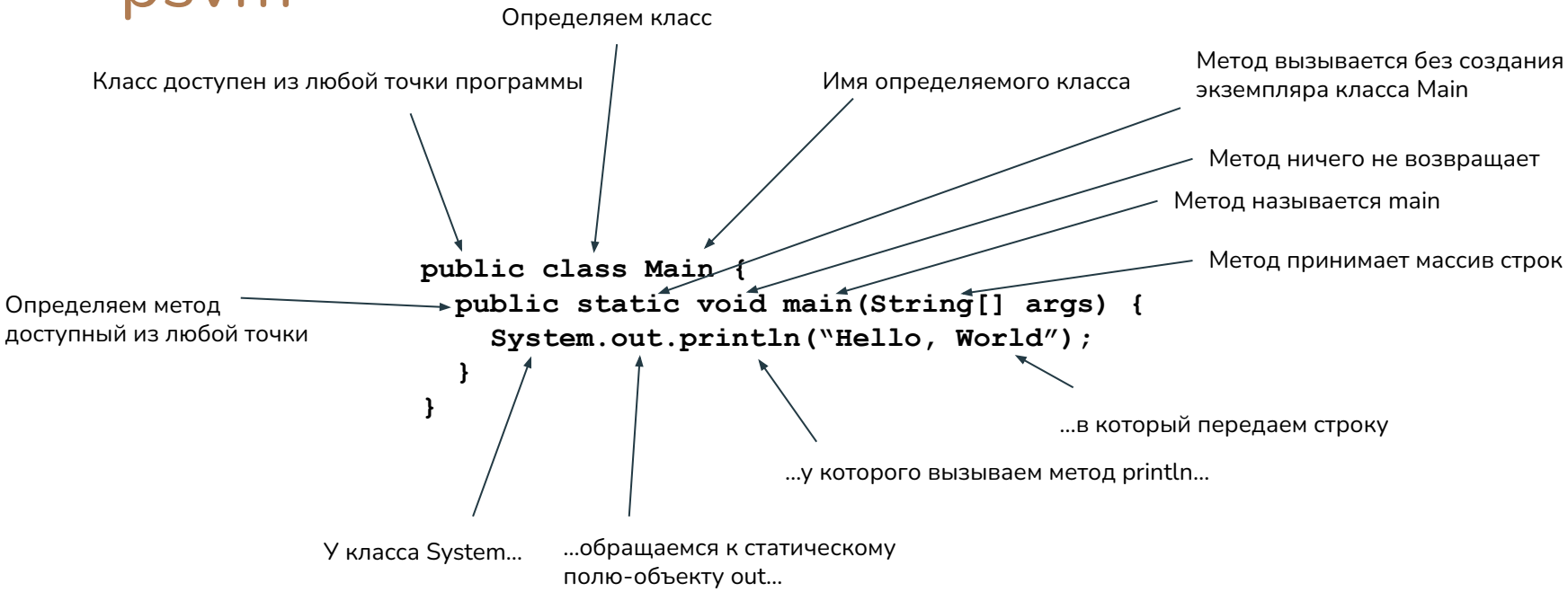
```
Shape shape = new Shape();
shape.print();// Я не знаю кто я
shape.print("Привет");// Привет
```

В данном примере примере говорят, что метод print является **перегруженным**

Переопределенный - это когда у наследника есть метод с таким же именем, количеством и типов входных аргументов, типом возвращаемого результата как и у родителя

Перегруженный - это когда в одном классе есть несколько методов с одним именем, но разным типов и количеством входных параметров

psvm


Определяем класс
Класс доступен из любой точки программы
Имя определяемого класса
Метод вызывается без создания экземпляра класса Main
Метод ничего не возвращает
Метод называется main
Метод принимает массив строк
public class Main {
public static void main(String[] args) {
System.out.println("Hello, World");
}
}
Определяем метод доступный из любой точки
...в который передаем строку
...у которого вызываем метод println...
У класса System...
...обращаемся к статическому полю-объекту out...

Интерфейсы и абстрактные классы

Интерфейсы и Абстрактные классы - это когда необходимо определить методы, которые должны быть в классе (их количество, имена и пр.). Отличия следующие:

- Интерфейс не содержит реализаций методов (кроме default), абстрактный класс может содержать все реализации
- При наследовании от абстрактного класса необходимо или объявлять абстрактный класс-наследник, или реализовать все абстрактные методы
- Для интерфейсов доступно множественное наследование (наследование от нескольких интерфейсов), для абстрактных классов нет
- ...

Интерфейсы

```
interface CanFight {
  void fight();
}

interface CanSwim {
  void swim();
}

interface CanFly {
  void fly();
}
```

```
class Superman implements CanFly, CanFight, CanSwim {
  // переопределить fight(), swim(), fly()
}

class OptimusPrime implements CanFly, CanFight {
  // переопределить fight(), fly()
}

class JohnWick implements CanFight, CanSwim {
  // переопределить fight(), swim()
}


CanSwim c = new JohnWick();
CanFly f = new Supermane();
```

Абстрактные классы

```
abstract class MovieHero {
  abstract String name();
  void introduce() {
    System.out.println("I am " + name());
  }
}

class Batman extends MovieHero {
  String name() {
    return "Batman";
  }
}

MovieHero hero = new Batman();
hero.introduce(); // I am Batman
```

Приведение типов

Иногда возникает необходимость привести один тип к другому, или, иначе говоря, изменить тип ссылки. В таком случае тип, к которому надо привести берется в скобки

```
abstract class MovieHero {
  abstract String name();
  void introduce() {
    System.out.println("I am " + name());
  }
}

class Batman extends MovieHero {
  String name() {
    return "Batman";
  }
  void ask() {
    System.out.println("Where's detonator?");
  }
}
```

```
MovieHero hero = new Batman();
hero.introduce(); // I am Batman
hero.ask(); // Ошибка! hero знает только о
            //методах класса MovieHero

Batman batman = (Batman) hero;
batman.ask();
```

Если приведение невозможно, будет ClassCastException.
Есть проверка

```
if (hero instanceof Batman) {...
```

Обзор домашнего задания

Рекомендации по ДЗ

1) Использовать общепринятое форматирование, лучше всего автоформатирование в IDE
2) Подбирать имена переменных и методов, отражающие смысл
3) KISS - не мудрить ради развлечения. Если есть желание сделать нетривиальное решение - сначала нужно сделать тривиально, а потом нетривиально, указав в комментарии для проверяющего, что текущее решение не основное, а экспериментальное
4) Внимательно смотреть, где вместо цепочки отдельно стоящих if лучше использовать if - else if - else
5) Не забывать закрывать ресурсы
6) Стараться избегать глубокой вложенности, разбивать длинные методы
7) Размышлять не только над тем как код будет работать, но над тем насколько легко он будет читаться