**Лабораторная работа № 5**

# Тема работы: «Перегрузка методов и передача аргументов»

**Цель работы:** закрепить навык перегрузки методов и конструкторов.

**Задание:** необходимо к классам из задания 1 лабораторной работы №4, добавить:

* не менее 5 перегруженных методов (для тех методов, где позволяет логика);
* не менее 3 методов с изменяемым числом входных параметров.

**Оснащение работы**

Задание по варианту, ЭВМ, среда разработки **IntelliJ IDEA**.

**Основные теоретические сведения**

**Перегрузка методов**

Метод - имя для действия: прыгнуть, мяукнуть, отформатировать диск.

Использование имён при написании кода упрощает её понимание и модификацию. Работа разработчика схожа с работой писателя - в обоих случаях требуется донести свою мысль до читателя/приложения.

Часто одно и то же слово имеет несколько разных значений - оно *перегружено*. Например, мы говорим "вымыть посуду" и "вымыть кота". Было бы глупо говорить "посудовымыть посуду" или "котовымыть кота", чтобы подчеркнуть разницу. Также и с методами. Можно создавать методы с одинаковыми именами, но с разным набором аргументов.

*Перегрузку (overloading) следует отличать от замещения (overriding) - иной реализации метода в подклассе первоначально определившего метод класса.*

Java, как и многие языки программирования, разрешает создавать внутри одно класса несколько методов с одним именем. Главное, чтобы у них различались параметры. Параметры могут различаться типами или количеством аргументов. Будьте внимательны, если вы зададите различные типы для возвращаемого значения, то этого будет недостаточно для создания перегруженной версии метода. Когда Java встречает вызов перегруженного метода, то выбирает ту версию, параметры которой соответствуют аргументам, использованным в вызове.

Создадим класс **Cat** с набором перегруженных методов:

class Cat {

void meow() {

// параметры отсутствуют

}

void meow(int count) {

// используется один параметр типа int

}

void meow(int count, int pause) {

// используются два параметра типа int

}

long meow(long time) {

// используется один параметр типа long return time;

}

double meow(double time) {

// используется один параметр типа double return time;

}

}

Вы можете вызвать любой метод из класса:

Cat kitty = new Cat(); kitty.meow(); kitty.meow(3); kitty.meow(3, 2); kitty.meow(4500.25);

Если присмотреться, то можно догадаться, какая именно версия метода вызывается в каждом конкретном случае.

Аналогично, перегрузка используется и для конструкторов.

**Методы с переменным числом параметров**

Можно создавать методы с переменным числом параметров. В этом случае используется многоточие.

public static double getMaxValue(Object... args) {

// код

}

По сути, создаѐтся массив типа **Object[]**.

Создадим метод, вычисляющий и возвращающий максимальное из произвольного количества значений.

public static double getMaxValue(double... values) { double largest = Double.MIN\_VALUE;

for(double v : values) { if (v > largest) {

largest = v;

}

}

return largest;

}

Использование метода.

double variable = getMaxValue(5.3, 39.6, -4);

**Метод toString()**

Каждый класс реализует метод **toString()**, так как он определён в классе **Object**. Но использовать метод по умолчанию не слишком удобно, так как не содержит полезной информации. Разработчики предпочитают переопределять данный метод под свои нужды. Сам метод имеет форму:

String toString()

Вам остаётся возвратить объект класса **String**, который будет содержать полезную информацию о вашем классе. Давайте возьмём для примера класс **Box**, созданный выше:

class Box {

int width; // ширина коробки int height; // высота коробки int depth; // глубина коробки

Box(int width, int height, int depth){ this.width = width;

this.height = height; this.depth = depth;

}

// вычисляем объѐм коробки String getVolume() {

return "Объѐм коробки: " + (width \* height \* depth);

+ depth;

}

public String toString() {

return "Коробочка для кота размером " + width + "x" + height + "x"

}

}

Теперь можете узнать о классе **Box**:

Box box = new Box(4, 5, 6); System.out.println(box.toString());

Метод очень часто используется при создании собственных классов и вам тоже придётся прибегать к этому способу.