**Лабораторная работа № 4**

**Тема работы: «Разработка методов и использование их в**

**программах»**

**Цель работы:** Закрепить навык создание методов и конструкторов, для эффективного решения задачи.

**Задание:** Необходимо реализовать классы из задания 1 лабораторной работы №3, и в каждом классе реализовать:

* не менее 4 полей (к каждому полю соответствуют нужные свойства);
* не менее 2 конструкторов (без параметров и со всеми параметрами);
* не менее 4 методов (с возвращающим значением и без возвращающего значения)

**Оснащение работы**

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

**Основные теоретические сведения**

Класс может содержать методы - один, два, три и больше в зависимости от сложности класса. Название метода всегда завершается двумя круглыми скобками, после которых идет блок кода, обрамлѐнный фигурными скобками. Например, метод **sayMeow()** класса **Cat** выводит мяуканье кошки. Внутри имени метода могут быть параметры, например, **sayMeow(3)** - кошка мяукает три раза. Параметров может быть несколько, тогда они разделяются запятыми.

Общая форма объявления метода выглядит следующим образом:

модификатор тип имяМетода(список\_параметров) {

// тело метода

}

Метод может не иметь параметров, в этом случае используются пустые скобки. Модификаторопределяет видимость метода (public, private). Если модификатор не указан, то считается, что метод имеет модификатор **private** в пределах своего пакета.

Методы могут вызывать другие методы.

Каждый метод начинается со строки объявления внутри круглых скобок, которую называют сигнатурой метода:

public static void sayMeow(int count) {

// здесь ваш код

}

Если рассмотреть данный метод, то можно сказать следующее. Ключевое слово public означает, что метод доступен для любого класса. Ключевое слово static означает, что нам не нужно создавать экземпляр (копию) объекта Cat в памяти. Ключевое слово void означает, что метод не возвращает никаких данных. Именем метода является слово перед круглыми скобками.

Если метод возвращает какие-то данные, то в теле метода используется оператор **return значение**, где **значение** - это возвращаемое значение. Тогда вместо **void** нужно указать возвращаемый тип.

Вспомним наш класс **Box**, в котором определены высота, ширина и глубина ящика. Зная эти величины, мы вычисляли объѐм коробки самостоятельно. Но мы можем расширить возможности класса, чтобы он сам мог вычислить объѐм внутри класса и предоставить нам готовые данные. Давайте добавим в класс новый метод:

class Box {

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

// вычисляем объѐм коробки String getVolume() {

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

}

}

Теперь пробуем вычислить объѐм коробки с помощью готового метода, который есть в классе:

Box catBox = new Box(); catBox.width = 400;

catBox.height = 200;

catBox.depth = 250;

mInfoTextView.setText(catBox.getVolume());

Мы уже не вычисляем объѐм вручную, за нас это сделает класс **Box**, у которого есть готовый метод для вычисления объѐмов.

Обращение к методу осуществляется как и к переменной через точку. Наличие круглых скобок позволяет различать метод от имени переменной. То есть, если вы увидите запись:

cat.getVolume; // это переменная cat.getVolume(); // это метод

Выше приведѐн немного искусственный пример, так как опытный программист никогда не назовѐт переменную именем **getVolume**. Существует рекомендация, что для методов в начале имени нужно использовать глагол и начинаться имя должно с маленькой буквы - переменные так называть не следует.

**Использование параметров**

Параметры позволяют работать с различными данными. Допустим, мы хотим вычислить площадь прямоугольника со сторонами 3 и 5 см.

Метод работает, но область его применения слишком ограничена. Мы сможем вычислять площадь только одного прямоугольника с жёстко заданными размерами. Но прямоугольники бывают разными и нам нужен универсальный метод. Добиться решения задачи можно с помощью параметров. Перепишем метод следующим образом:

int getSquare(int a, int b) { return a \* b;

}

Теперь мы можем вычислять площадь любого прямоугольника, зная его размеры. Возьмём тот же прямоугольник со сторонами 3 и 5 см и вычислим его площадь:

mInfoTextView.setText("Площадь прямоугольника: " + getSquare(3, 5));

В правильно написанных классах стараются избегать доступа к переменным класса напрямую. Следует использовать методы, которые позволяют избежать ошибок. В нашем классе **Box** использовались отдельные переменные **width, height, depth**.

Пробуем класс в действии:

Box catBox = new Box(); catBox.setDim(400, 200, 250); int vol = catBox.getVolume();

mInfoTextView.setText("Объѐм ящика: " + vol);

Теперь мы не обращаемся к отдельным переменным класса, а вызываем метод **setDim()** для установки размеров ящика, после чего можно вызвать метод **getVolume()** для вычисления объѐма. Естественно, вы можете реализовать свою логику в классе. Например, вы можете создать метод **getVolume()** с параметрами, которые отвечают за размеры ящика и возвращать сразу готовый результат.

**Класс Object**

В Java есть специальный суперкласс **Object** и все классы являются его подклассами. Поэтому ссылочная переменная класса **Object** может ссылаться на объект любого другого класса. Так как массивы являются тоже классами, то переменная класса **Object** может ссылаться и на любой массив.

// применимо к любому классу Object obj = new Cat("Barsik");

В таком виде объект обычно не используют. Чтобы с объектом что-то сделать, нужно выполнить приведение типов.

Cat cat = (Cat) obj;

У класса есть несколько важных методов.

* Object clone() - создаѐт новый объект, не отличающий от клонируемого
* boolean equals(Object obj) - определяет, равен ли один объект другому
* void finalize() - вызывается перед удалением неиспользуемого объекта
* Class<?> getClass() - получает класс объекта во время выполнения
* int hashCode() - возвращает хеш-код, связанный с вызывающим объектом
* void notify() - возобновляет выполнение потока, который ожидает вызывающего объекта
* void notifyAll() - возобновляет выполнение всех потоков, которые ожидают вызывающего объекта
* String toString() - возвращает строку, описывающий объект
* void wait() - ожидает другого потока выполнения
* void wait(long millis) - ожидает другого потока выполнения
* void wait(long millis, int nanos) - ожидает другого потока выполнения

Методы **getClass()**, **notify()**, **notifyAll()**, **wait()** являются финальными и их нельзя переопределять.

**Метод hashCode()**

Хеш-код - это целое число, генерируемое на основе конкретного объекта.

Его можно рассматривать как шифр с уникальным значением.

Для вычисления хеш-кода в классе **String** применяется следующий алгоритм.

int hash = 0;

for(int i = 0; i < length(); i++) hash = 31 \* hash + charAt(i);

У любого объекта имется хеш-код, определяемый по-умолчанию, который вычисляется по адресу памяти, занимаемой объектом.

Значение хеш-кода возвращает целочисленное значение, в том числе, и отрицательное.

Если в вашем классе переопределяется метод **equals()**, то следует переопределить и метод **hashCode()**.

**Метод toString()**

Очень важный метод, возвращающий значение объекта в виде символьной строки.

Очень часто при использовании метода **toString()** для получения описания объекта можно получить набор бессмысленных символов, например, [I@421199e8. На самом деле в них есть смысл, доступный специалистом. Он сразу может сказать, что мы имеем дело с одномерным массивом (одна квадратная скобка), который имеет тип **int** (символ I). Остальные символы тоже что-то означают, но вам знать это не обязательно.

Если же вам нужно научное объяснение, то метод работает по следующему алгоритму (из документации).

getClass().getName() + '@' + Integer.toHexString(hashCode())

Обычно принято переопределять метод, чтобы он выводил результат в читаемом виде.