# Лабораторная работа № 3

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

**программах»**

**Цель работы:** Закрепить навык создания и применения классов при решении различных задач.

З**адние:** Два задания обязательны для выполнения.

В первом задании необходимо произвести проектирование классов по теме вашего варианта. Нужно описать:

* не менее 3 классов (имя класса – существительное); 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 3 методов.

Также привести по одному примеру (объекту) каждого класса.

Во втором задании необходимо реализовать класс по условию варианта, заготовить несколько контрольных примеров. Организовать проверку существования необходимой фигуры.

Номер варианта соответствует вашему номеру по списку.

**Задание 1: проектирование Класса**

Вариант 1. Здравоохранение.

Вариант 2. Защита человека от воздействия окружающей среды. Безопасность.

Вариант 3. Метрология и измерения. Вариант 4. Машиностроение.

Вариант 5. Электроника.

Вариант 6. Телекоммуникации. Аудио- и видеотехника. Вариант 7. Информационные технологии.

Вариант 8. Офисное оборудование.

Вариант 9. Технология получения изображений. Вариант 10. Дорожно-транспортная техника.

Вариант 11. Железнодорожная техника.

Вариант 12. Судостроение и морские сооружения. Вариант 13. Авиационная и космическая техника. Вариант 14. Сельское хозяйство.

Вариант 15. Производство пищевых продуктов. Вариант 16. Химическая промышленность.

Вариант 17. Горное дело и полезные ископаемые.

Вариант 18. Добыча и переработка нефти, газа и смежные производства. Вариант 19. Металлургия.

Вариант 20. Гражданское строительство.

Вариант 21. Военная техника.

Вариант 22. Бытовая техника и торговое оборудование.

Вариант 23. Образование, среднее полное общее образование.

Вариант 24. Образование, высшее профессиональное образование.

Вариант 25. Библиотечное дело.

Вариант 26. Телекоммуникации.

Вариант 27. Компьютерные сети.

Вариант 28. Гражданская техника. Вариант 29. Космическая техника. Вариант 30. Химическое производство.

**Задание 2: создание Класса**

**Вариант 1.**

Класс: треугольник

Поля: три стороны

Операции:

увеличение/уменьшение размера сторон в заданное количество раз;

вычисление параметров;

вычисление площади;

определение значений углов.

**Вариант 2.**

Класс: треугольник

Поля: три стороны

Операции:

увеличение/уменьшение размера сторон в заданное количество процентов;

вычисление средней линии для любой из сторон;

определение вида треугольника по величине углов (Остроугольный, Тупоугольный, Прямоугольный);

определение значений уголов.

**Вариант 3.**

Класс: треугольник

Поля: две стороны и угол между ними

Операции:

увеличение/уменьшение размера угла на заданное количество процентов;

определение вида треугольника по числу равных сторон (Разносторонний, Равнобедренный, Равносторонний);

определение расстояния между центрами вписанной и описанной окружностей;

определение значений углов.

**Вариант 4.**

Класс: треугольник

Поля: две стороны и угол между ними

Операции:

увеличение/уменьшение размера угла (из полей) в заданное количество раз;

вычисление длины биссектрисы принадлежащей любому углу;

вычисление длин отрезков, на которые биссектриса делит любую сторону;

определение значений углов.

**Вариант 5.**

Класс: треугольник

Поля: сторона и два прилежащих к ней угла

Операции:

увеличение/уменьшение размера стороны (из полей) в заданное количество раз;

вычисление длины медианы, принадлежащей любой стороне;

определение подобен ли другой треугольник данному (указанному по индексу массива);

определение значений сторон.

**Вариант 6.**

Класс: треугольник

Поля: сторона и два прилежащих к ней угла

Операции:

увеличение/уменьшение значения любого угла (из полей) на заданное количество процентов;

вычисление длины высот, принадлежащей любой стороне;

определение значений сторон.

**Вариант 7.**

Класс: прямоугольный треугольник

Поля: две стороны

Операции:

увеличение/уменьшение размера любой стороны (из полей) на заданное количество процентов;

вычисление радиуса описанной окружности;

вычисление полупериметра;

определение значений сторон.

**Вариант 8.**

Класс: прямоугольный треугольник

Поля: сторона и угол

Операции:

увеличение/уменьшение размера любой стороны (из полей) на заданный процент;

вычисление радиуса вписанной окружности;

определение расстояния между центрами вписанной и описанной окружностей;

вычисление квадратного корня из площади;

определение значения сторон.

**Вариант 9.**

Класс: равнобедренный треугольник

Поля: основание и боковая сторона

Операции:

увеличение/уменьшение размера на определенный процент;

вычисление длины медианы, принадлежащей любой стороне;

вычисление периметра и площади;

определение значений углов.

**Вариант 10.**

Класс: равнобедренный треугольник

Поля: боковая сторона и угол при основании

Операции:

увеличение/уменьшение размера в заданное количество раз;

вычисление длины биссектрисы, принадлежащей любому углу;

вычисление длины высот, принадлежащей любой стороне;

определение значений сторон.

**Вариант 11.**

Класс: параллелограмм

Поля: две стороны и угол между ними

Операции:

увеличение/уменьшение размера любой из сторон (из полей) на определенный процент;

вычисление периметра и площади;

вычисление диагоналей;

вычисление высоты.

**Вариант 12.**

Класс: Параллелограмм

Поля: две стороны и диагональ (прилегающая к ним так, чтобы образовать треугольник)

Операции:

увеличение/уменьшение размера в заданное количество раз;

вычисление квадратного корня из периметра и площади;

вычисление диагонали и стороны;

вычисление высоты.

**Вариант 13.**

Класс: прямоугольник

Поля: две стороны

Операции:

увеличение/уменьшение размера любой из сторон на определенный процент;

вычисление периметра и площади;

вычисление диагонали.

**Вариант 14.**

Класс: квадрат

Поля: сторона

Операции:

увеличение/уменьшение размера на определенный процент;

вычисление периметра и площади;

вычисление диагонали.

**Вариант 15.**

Класс: ромб

Поля: сторона и диагональ (меньшая)

Операции:

увеличение/уменьшение размера на определенный процент;

вычисление периметра и площади;

вычисление диагонали.

* 1. **Оснащение работы**

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

* 1. **Основные теоретические сведения Классы**

Java позволяет создавать классы, которые представляют объекты из реального мира. Например, можно создать класс **Car** (автомобиль) или **Animal** (животное) и задать им различные свойства. Для класса **Car** логично создать такие свойства как двери, колѐса, лобовое стекло и т.д. Имея класс **Car**, можно создать новые классы Легковушки, Грузовики, Автобусы, которые будут иметь все свойства класса **Car**, а также свои собственные свойства. У класса **Animal** соответственно можно задать свойства Лапы, Хвост, а затем создать наш любимый класс **Cat**, у которого будет ещѐ дополнительное свойство Усы. Иными словами, классы могут наследовать свойства от других классов. Родительский класс называется суперклассом. Внутри классов могут быть объявлены поля и методы.

Для объявления класса служит ключевое слово **class**. Вспомним стандартную строчку кода из Android-проекта:public class MainActivity extends Activity {

// код внутри класса

}

Упрощённая общая форма для класса может иметь следующий вид:

class ИмяКласса {

тип переменная\_экземпляра1;

тип имяМетода(список параметров){

// тело метода

}

}

В Java принято начинать имена класса с большой буквы. В классе могут быть несколько переменных и методов. Переменные, определѐнные внутри класса (не метода), называются переменными экземпляра или полями (fields). Код пишется внутри класса. Методы и переменные внутри класса являются членами класса.

**Объекты**

Новый объект (или экземпляр) создаѐтся из существующего класса при помощи ключевого слова **new**:

Cat barsik = new Cat(); // создали кота из класса Cat

В большинстве случаев вы будете использовать такой способ. Пусть вас не удивляет, что приходится дважды использовать слово **Cat**, оно имеет разный смысл.

Слева от оператора присваивания **=** определяется имя переменной и его тип **Cat**. В правой части выражения происходит выделение памяти для нового экземпляра класса **Cat** и инициализируется экземпляр. Оператор присваивания присваивает переменной ссылку на только что созданный объект. Имена объектов не нужно начинать с большой буквы, как у класса. Так вы будете различать, где класс, а где экземпляр класса. Если имя экземпляра класса состоит из нескольких слов, то используется верблюжья нотация, когда все первые буквы слов, кроме первой, пишутся с большой - superBlackCat.

Если вы помните, при объявлении примитивных типов мы указывали нужный тип в самом начале.

int catAge;

Поэтому код **Cat barsik** также определяет его тип. Он не всегда может совпадать с именем класса.

Pet barsik = new Cat();

В этом примере используется тип класса домашних любимцев **Pet**, а обращаемся к классу котов **Cat**.

Теперь подробнее.

Простой пример создания класса **Box** (коробка для кота):

class Box {

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

}

При таком варианте Java автоматически присвоит переменным значения по умолчанию. Например, для **int** это будет значение 0. Но не всегда значения по умолчанию подойдут в вашем классе. Если вы создали переменную для описания количества лап у кота, то логично сразу присвоить значение 4. Поэтому считается хорошей практикой сразу присваивать нужные значения полям класса, не полагаясь на систему.

Вам нужно создать отдельный файл **Box.java**, в который следует вставить код, описанный выше. О том, как создавать новый файл для класса я не буду здесь расписывать.

Сам класс - это просто шаблон, заготовка. Чтобы ваше приложение могло использовать данный шаблон, нужно создать на его основе объект при помощи ключевого слова **new**:

Box catBox = new Box; // создали реальный объект с именем catBox на основе шаблона Box

Объект **catBox**, объявленный в коде вашей программы, сразу займѐт часть памяти на устройстве. При этом объект будет содержать собственные копии переменных экземпляра **width, height, depth**. Для доступа к этим переменным используется точка (.). Если мы хотим присвоить значение переменной **width**, то после создания объекта класса можете написать код:

catBox.width = 400; // ширина коробки для кота 400 миллиметров

Если мы хотим вычислить объём коробки, то нужно перемножить все

значения размеров коробки:

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

catBox.height = 200;

catBox.depth = 250;

int volume = catBox.width \* catBox.height \* catBox.depth;

mInfoTextView.setText("Объѐм коробки: " + volume);

Каждый объект содержит собственные копии переменных экземпляра. Вы можете создать несколько объектов на основе класса **Box** и присваивать разные значения для размеров коробки. При этом изменения переменных экземпляра одного объекта никак не влияют на переменные экземпляра другого объекта. Давайте объявим два объекта класса **Box**:

Box bigBox = new Box(); // большая коробка Box smallBox = new Box(); // маленькая коробка

int volume;

// присвоим значения переменным для большой коробки bigBox.width = 400;

bigBox.height = 200;

bigBox.depth = 250;

// присвоим значения переменным для маленькой коробки smallBox.width = 200;

smallBox.height = 100;

smallBox.depth = 150;

// вычисляем объѐм первой коробки

volume = bigBox.width \* bigBox.height \* bigBox.depth; mInfoTextView.setText("Объѐм большой коробки: " +

volume + "\n");

// вычисляем объѐм маленькой коробки

volume = smallBox.width \* smallBox.height \* smallBox.depth;

mInfoTextView.append("Объѐм маленькой коробки: " + volume);

Когда мы используем конструкцию типа **Box bigBox = new Box();**, то в

одной строке выполняем сразу два действия - объявляем переменную типа класса и резервируем память под объект. Можно разбить конструкцию на отдельные части:

BoxBox bigBox; // объявляем ссылку на объект

bigBox = new Box(); // резервируем память для объекта

Обычно такую конструкцию из двух строк кода не используют на практике, если нет особых причин.

Когда мы используем ключевое слово **new** и указываем имя класса, то после имени ставим круглые скобки, которые указывают на конструктор класса. О них поговорим позже.

**Ключевое слово final**

Поле может быть объявлено как **final** (финальное). Это позволяет предотвратить изменение содержимого переменной, по сути, это становится константой. Финальное поле должно быть инициализировано во время его первого объявления.

final int FILE\_OPEN = 1;

Теперь можно пользоваться переменной **FILE\_OPEN** так, как если бы она была константой, без риска изменения их значений. Принято записывать имена заглавными буквами.

Кроме полей, **final** можно использовать для параметров метода (препятствует изменению в пределах метода) и у локальных переменных (препятствует присвоению ей значения более одного раза).

Также слово **final** можно применять к методам, чтобы предотвратить его переопределение.

class Cat {

final void meow() { System.out.println("Мяу");

}

}

class Kittent extends Cat {

// Этот метод создать не получится void meow() {

System.out.println("Да хоть гав-гав, всѐ равно не заведѐтся");

}

}

Ещё один вариант использования ключевого слова **final** - предотвращение наследования класса. При этом неявно всего методы класса также становятся финальными. Поэтому нельзя одновременно объявить класс абстрактным и финальным, поскольку абстрактный класс является лишь шаблоном и только его подклассы реализуют методы.

final class Tail {

// ...

}

// Следующий класс недопустим class BigTail extends Tail {

// Ошибка! Класс Хвост нельзя переопределять.

}

**Ключевое слово instanceof - Проверка принадлежности к классу**

Иногда требуется проверить, к какому классу принадлежит объект. Это можно сделать при помощи ключевого слова **instanceof**. Это булев оператор, и выражение **foo instanceof Foo**истинно, если объект **foo** принадлежит классу **Foo** или его наследнику, или реализует интерфейс **Foo** (или, в общем виде, наследует класс, который реализует интерфейс, который наследует **Foo**).

Возьмѐм пример с рыбками, которые знакомы котам не понаслышке. Пусть у нас есть родительский класс **Fish** и у него есть унаследованные подклассы **SaltwaterFish** и **FreshwaterFish**. Мы можем протестировать, относится ли заданный объект к классу или подклассу по имени

SaltwaterFish nemo = new SaltwaterFish(); if(nemo instanceof Fish) {

// рыбка Немо относится к классу Fish

// это может быть класс Fish (родительский класс) или подкласс типа

// SaltwaterFish или FreshwaterFish.

if(nemo instanceof SaltwaterFish) {

// Немо - это морская рыбка!

}

}

Данная проверка удобна во многих случаях. В Android очень много классов, которые происходят от класса **View** - **TextView**, **CheckBox**, **Button**, имеющие свои собственные наборы свойств. И если имеется метод с параметром **View**, то при помощи **instanceof** можно разделить логику кода:

void checkforTextView(View view)

{

if(view instanceof TextView)

{

// Код для элемента TextView

} else {

// Для других элементов View

}

}

import - Импорт класса

Оператор import сообщает компилятору Java, где найти классы, на которые ссылается код. Любой сложный объект использует другие объекты для выполнения тех или иных функций, и оператор импорта позволяет сообщить о них компилятору Java. Оператор импорта обычно выглядит так:

import ClassNameToImport;

За ключевым словом **import** следуют класс, который нужно импортировать, и точка с запятой. Имя класса должно быть полным, то есть включать свой пакет. Чтобы импортировать все классы из пакета, после имени пакета можно поместить .\*.

ru.alexanderklimov.MyClass.Cat.sayMeow(); java.lang.System.out.println("Мяу");

**Статический импорт**

Существует ещѐ статический импорт, применяемый для импорта статических членов класса или интерфейса. Это позволяет сократить количество кода. Например, есть статические методы **Math.pow()**, **Math.sqrt()**. Для вычислений сложных формул с использованием математических методов, код становится перегружен. К примеру, вычислим гипотенузу.

hypot = Math.sqrt(Math.pow(side1, 2) + Math.pow(side2, 2));

В данном случае без указания класса не обойтись, так как методы

статические. Чтобы не набирать имена классов, их можно импортировать следующим образом:

import static java.lang.Math.sqrt; import static java.lang.Math.pow;

...

hypot = sqrt(pow(side1, 2) + pow(side2, 2));

После импорта уже нет необходимости указывать имя класса.

Второй допустимый вариант, позволяющий сделать видимыми все статические методы класса:

import static java.lang.Math.\*;

В этом случае вам не нужно импортировать отдельные методы. Но такой подход в Android не рекомендуется, так как требует больше памяти.

**Класс Class**

На первый взгляд, класс **Class** звучит как "масло масляное". Тем не менее, класс с таким именем существует и он очень полезен.

Иногда из программы нужно получить имя используемого класса. Для этого есть специальные методы **getClass().getName()** и другие родственные методы. Допустим, нам нужно узнать имя класса кнопки, на которую мы нажимаем в программе.

public void onClick(View view) {

String className = view.getClass().getName(); String simpleName =

view.getClass().getSimpleName();

String canonicalName = view.getClass().getCanonicalName();

if (canonicalName == null) { canonicalName = "null";

}

String s = "Имя класса: " + className + "\n" + "SimpleName: " + simpleName

+ "\n" + "CanonicalName: " + canonicalName + "\n";

System.out.println(s);

}

Если нужно узнать имя класса активности, то достаточно кода:

// подставьте имя вашей активности

String className = MainActivity.class.getName();

Если вам известно имя класса, то можете получить сам класс:

try {

// получим объект Class

Class<?> myClass = Class.forName("ru.alexanderklimov.test.MainActivity");

mInfoTextView.setText(myClass.getName()); // выводим в TextView

Intent intent = new Intent(this, myClass); startActivity(intent);

} catch (ClassNotFoundException e) {

// TODO Auto-generated catch block e.printStackTrace();

}

Метод **getSuperclass()** возвращает имя суперкласса. Остальные несколько десятков методов не столь популярны.