# Лабораторная работа № 7

# Тема работы: «Разработка программ с использованием наследования и переопределение методов»

**Цель работы:** отработать навык реализации наследования, и переопределения методов супер-класса.

**Задание:** необходимо реализовать в задания 1 лабораторной работы №5 логически продуманное наследование, при этом реализовать:

* цепочку не менее чем из 3 классов;
* в классах-потоках переопределить необходимые методы супер-классов;
* организовать наследование конструкторов супер-классов.

**Оснащение работы**

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

**Основные теоретические сведения Переопределение методов**

Если в иерархии классов имя и сигнатура типа метода подкласса совпадает с атрибутами метода суперкласса, то метод подкласса *переопределяет* метод суперкласса. Когда переопределённый метод вызывается из своего подкласса, он всегда будет ссылаться на версию этого метода, определѐнную подклассом. А версия метода из суперкласса будет скрыта.

Если нужно получить доступ к версии переопределённого метода, определённого в суперклассе, то используйте ключевое слово **super**.

Не путайте переопределение с перегрузкой. Переопределение метода выполняется только в том случае, если имена и сигнатуры типов двух методов идентичны. В противном случае два метода являются просто перегруженными.

В Java SE5 появилась запись @Override; она не является ключевым словом. Если вы собираетесь переопределить метод, используйте @Override, и компилятор выдаст сообщение об ошибке, если вместо переопределения будет случайно выполнена перегрузка.

Для закрепления материала создадим класс **Animal** с одним методом.

package ru.alexanderklimov.expresscourse; public class Animal {

String sleep(){

return "Животные иногда спят";

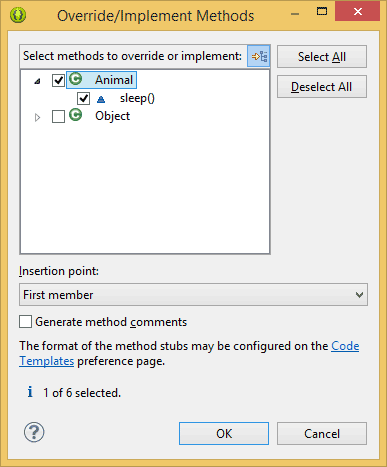
}

}

Теперь создадим класс **Cat**, наследующий от первого класса. public class Cat extends Animal {

}

Java знает, у родительского класса есть метод **sleep()**. Удостовериться можно следующим образом. Находясь в классе**Cat**, выберите в меню **Source | Override/Implement Methods...**. Появится диалоговое окно, где можно отметить флажком нужный метод.



В результате в класс будет добавлена заготовка:

@Override String sleep() {

// TODO Auto-generated method stub return super.sleep();

}

Попробуем вызвать данный метод в основном классе активности:

public void onClick(View v) { Cat barsik = new Cat();

mInfoTextView.setText(barsik.sleep());

}

Мы получим текст, который определѐн в суперклассе, хотя вызывали метод дочернего класса.

Но если мы хотим получить другой текст, совсем не обязательно придумывать новые методы. Достаточно закомментировать вызов метода из суперкласса и добавить свой вариант.

@Override String sleep() {

//return super.sleep(); return "Коты постоянно спят!";

}

Запускаем программу и нажимаем на кнопку. И получим уже другой ответ, более соответствующий описанию среднестатистического кота. Заметьте, что код для щелчка кнопки мы не меняем, но система сама разберѐтся, что выводить нужно текст не из суперкласса, а из дочернего класса.

Рассмотрим другой пример переопределения методов. Создадим суперкласс **Figure**, который будет содержать размеры фигуры, а также метод для вычисления площади. А затем создадим два других класса **Rectangle** и **Triangle**, у которых мы переопределим данный метод.

class Figure { double dim1; double dim2;

Figure(double a, double b) { dim1 = a;

dim2 = b;

}

double area() { System.out.printLn("Площадь фигуры");

return 0;

}

}

class Rectangle extends Figure { Rectangle(double a, double b) {

super(a, b);

}

// Переопределяем метод double area() {

System.out.println("Площадь прямоугольника"); return dim1 \* dim2;

}

}

class Triangle extends Figure { Triangle(double a, double b) {

super(a, b);

}

// переопределяем метод double area() {

System.out.println("Площадь треугольника"); return dim1 \* dim2 / 2;

}

}

// В главной активности

Figure figure = new Figure(10, 10); Rectangle rectangle = new Rectangle(8, 5);

|  |  |  |
| --- | --- | --- |
| Triangle triangle = new Triangle(10, | 6); |  |
| Figure fig; |  |
| fig = figure; mInfoTextView.setText("Площадь равна | " + | fig.area); |
| fig = rectangle; mInfoTextView.setText("Площадь равна | " + | fig.area); |
| fig = triangle; mInfoTextView.setText("Площадь равна | " + | fig.area); |