

Класс Object

Что объединяет все классы

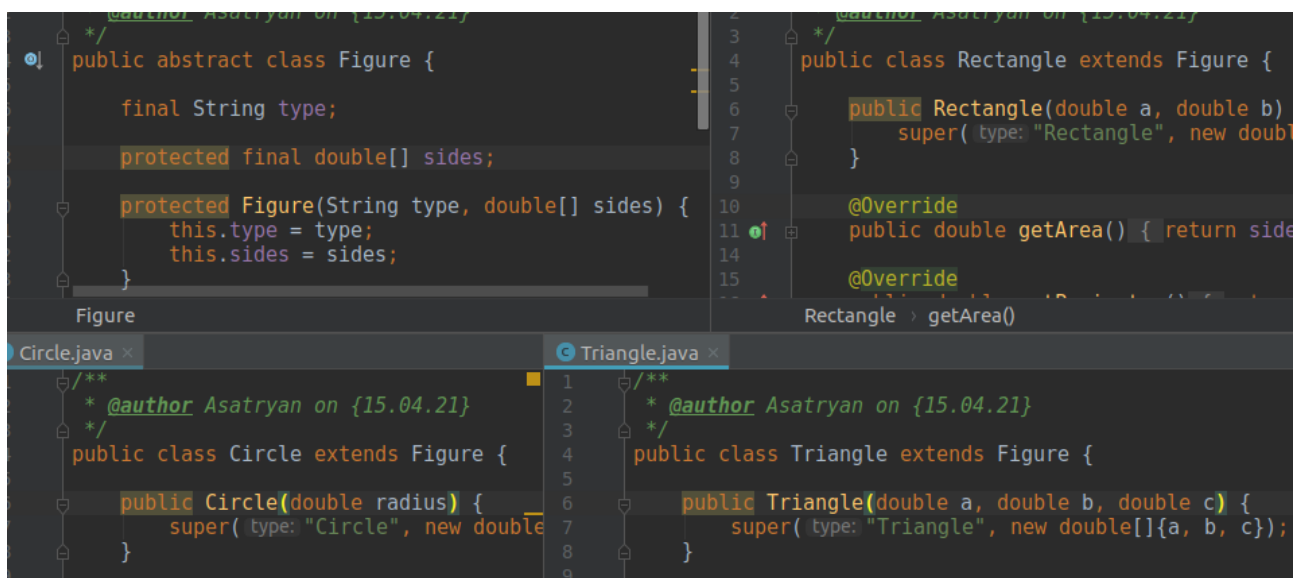
Содержание

1. Упрощаем код, метод getClass() у класса Object
2. Метод toString() у класса Object

1. Упрощаем код, метод getClass() у класса Object

Сразу заспойлерю, в этой лекции можно запутаться окончательно и потому вам стоит прочитать ее возможно не один раз. Здесь мы будем упрощать тот код, который написали в предыдущей лекции.

Итак, сначала мы посмотрим на вот что. У нашего абстрактного класса есть 3 наследника и поле тип. Давайте взглянем на него для начала. Вы заметили что оно совпадает с именем класса для всех случаев?



```
public abstract class Figure {
    final String type;
    protected final double[] sides;

    protected Figure(String type, double[] sides) {
        this.type = type;
        this.sides = sides;
    }
}

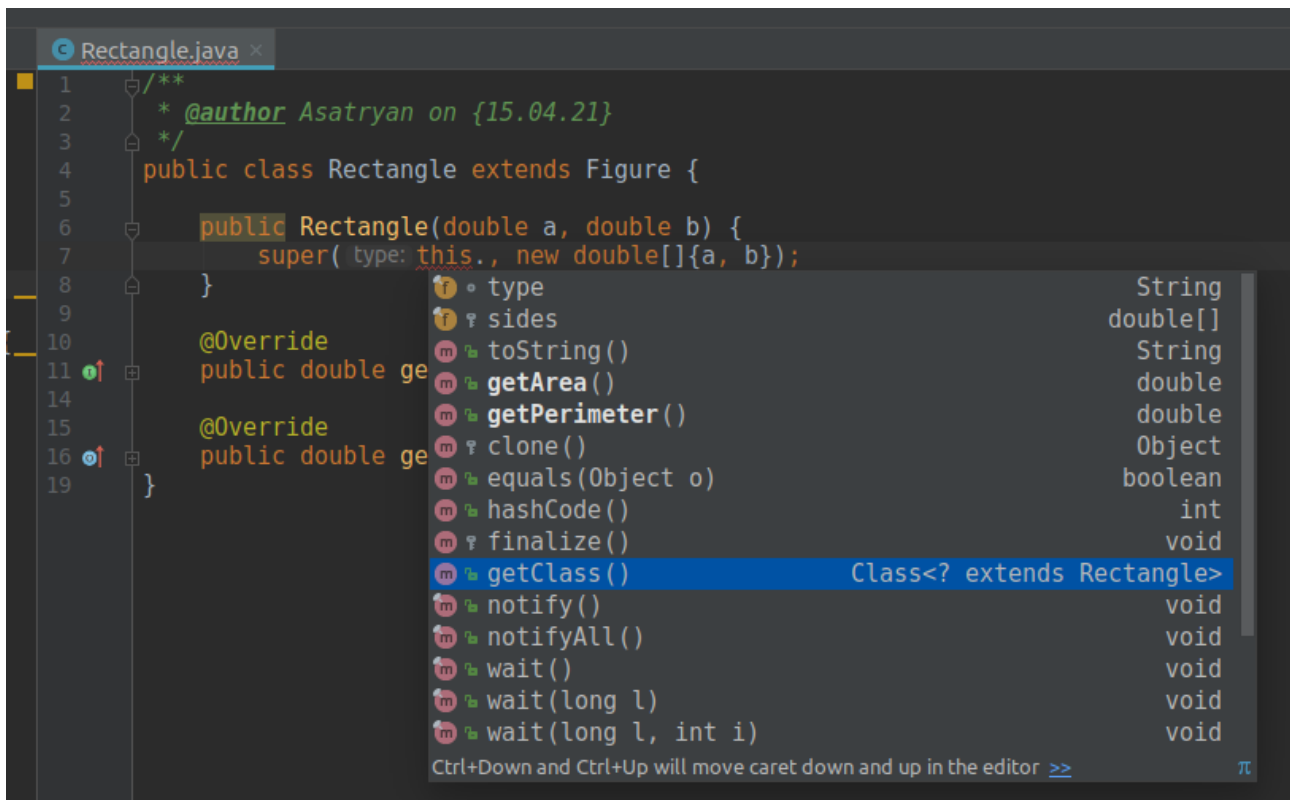
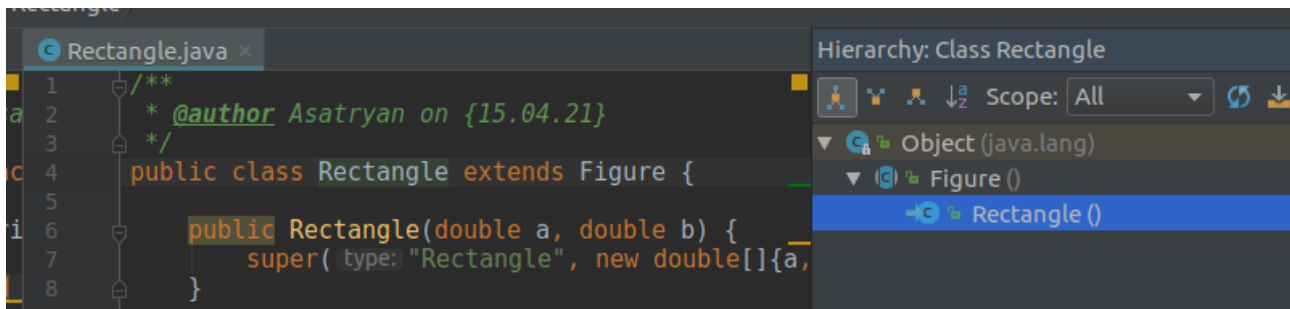
public class Circle extends Figure {
    public Circle(double radius) {
        super("Circle", new double[] {radius, radius});
    }
}

public class Rectangle extends Figure {
    public Rectangle(double a, double b) {
        super("Rectangle", new double[] {a, b});
    }
    @Override
    public double getArea() { return sides[0] * sides[1]; }
}

public class Triangle extends Figure {
    public Triangle(double a, double b, double c) {
        super("Triangle", new double[] {a, b, c});
    }
}
```

Было бы классно не писать для каждого класса тип геом.фигуры, а как-нибудь получить имя класса. Мы говорили что все в языке Джава это объекты. И было бы правильно предположить, что все классы в Джава наследники какого-нибудь класса Object. И это действительно так. Для этого ставим курсор на имени класса например прямоугольника и жмем Ctrl+N, где N значит иерархия. И видим следующее.

Действительно, наш класс прямоугольника наследуется от класса Figure, а тот в свою очередь является наследником класса Object. А значит у этого класса есть какие-то методы. Как их узнать? А легко. Мы же когда берем объект класса то получаем доступ к его публик методам через точку, верно? А к протектед методам сможем получить доступ через точку внутри класса. Пробуем.



Мы помним что можем обращаться к объекту класса через ключевое слово `this`. А значит к его методам через это слово. И как видим у нас есть некоторые методы класса `Object`. Среди них нас сейчас интересует метод `getClass`. Что же он делает? И здесь самое сложное. Кроме класса `Object` который является родителем любого класса в Джава есть еще другой класс, который называется... `Class`. Он дает доступ к информации относительно этого же класса. И кароче у него есть метод, который может вернуть имя этого же класса строкой. Смотрим.

И мы вызываем метод получения имени класса сразу у абстрактного класса, чтобы не вносить изменения в каждый наследник (в нашем случае это и невозможно, попробуйте сами). Теперь мы можем создавать сколько угодно классов наследников нашей фигуры и у всех у них будет тип который равен имени этого класса. Можете запустить код в мейне и проверить что все как и раньше. И это очень важно на самом деле. Вы можете дать возможность сделать так, чтобы тип (поле строкового вида) не совпадал с именем класса. Для этого просто делаем `protected` метод, который можно переопределить при необходимости. Например вот так.

```
public abstract class Figure {
    final String type;
    protected final double[] sides;

    protected Figure(double[] sides) {
        this.type = getClass().getSimpleName();
        this.sides = sides;
    }
}

public class Rectangle extends Figure {
    public Rectangle(double a, double b) {
        super(new double[]{a, b});
    }

    @Override
    public double getArea() { return sides[0] * sides[1]; }

    @Override
    public double getPerimeter() { return 2 * (sides[0] + sides[1]); }
}

Circle.java
/**
 * @author Asatryan on {15.04.21}
 */
public class Circle extends Figure {
    public Circle(double radius) {
        super(new double[]{radius, radius});
    }
}

Triangle.java
/**
 * @author Asatryan on {15.04.21}
 */
public class Triangle extends Figure {
    public Triangle(double a, double b, double c) {
        super(new double[]{a, b, c});
    }
}
```

```
public static void main(String[] args) {
    Figure[] figures = new Figure[3];
    figures[0] = new Circle(radius: 2);
    figures[1] = new Rectangle(a: 5, b: 6);
    figures[2] = new Triangle(a: 3, b: 4, c: 5);

    for (Figure figure : figures) {
        print(figure.type + " - area: " + figure.getArea() +
            ", perimeter: " + figure.getPerimeter());
    }
}

Main > main()

un: Main x
/usr/lib/jvm/java-1.8.0-openjdk-amd64/bin/java ...
Circle - area: 12.56, perimeter: 12.56
Rectangle - area: 30.0, perimeter: 22.0
Triangle - area: 6.0, perimeter: 12.0
```

Заметьте, что мы дали возможность наследникам только менять тип класса. Ведь поле все равно `final` типа и нельзя его перезаписать. И так как протектед метод вызывается в конструкторе мы не будем иметь ситуации с перезаписью поля извне после создания объекта.

И это не все возможные изменения которые мы рассмотрим в этой лекции. Теперь же давайте посмотрим на другие методы класса `Object` которые уже есть и попробуем использовать их.



```
public abstract class Figure {
    final String type;
    protected final double[] sides;

    protected Figure(double[] sides) {
        this.type = getType();
        this.sides = sides;
    }

    protected String getType() {
        return getClass().getSimpleName();
    }
}

public class Rectangle extends Figure {
    public Rectangle(double a, double b) {
        super(new double[]{a, b});
    }

    @Override
    protected String getType() {
        return "Прямоугольник";
    }

    @Override
    public double getArea() { return sides[0] * sides[1]; }
}

// Console Output:
Circle - area: 12.56, perimeter: 12.56
Прямоугольник - area: 30.0, perimeter: 22.0
Треугольник - area: 6.0, perimeter: 12.0
Process finished with exit code 0
```

2. Метод toString() у класса Object

Если вы помните, любой тип переменной к которой прибавили строку (даже пустую) становится как бы строкой. Но вы заметили, что у класса Object уже есть подобный метод? Метод toString(). Он позволяет получить строковое отображение объекта. Ок, если уже есть такой метод, почему бы его не использовать. Ведь посмотрите на наш мейн метод. Все что мы делаем это просто выводим в консоль информацию по объекту и для этого написали однотипный код для всех объектов. Почему бы его не скрыть внутри класса и к тому же сделать поле тип скрытым извне? К тому же еще и методы для получения площади и периметра тоже можно сделать protected. Посмотрим.

Итак, мы переопределили метод toString в самом классе Figure. Мы описали как представить наш класс в строковом виде и сделали это единожды для всех наследников. Периметр и площадь высчитываются у каждого наследника по-своему, тип тоже можно переопределить, но именно конструкцию под отображение в строке... Стоп. Мы же можем и это переопределить. Предположим я не хочу выдавать информацию по периметру для класса прямоугольник. Как же мне поступить? Ну если мы смогли переопределить метод toString у родительского класса Figure, то точно так же можем переопределить и у класса Rectangle. Так? Пробуем. И сразу же подумаем – а зачем нам тогда метод получения периметра если мы его не используем и он не виден снаружи? Сможем ли мы поменять ему видимость на public? Ведь в родительском объявлено protected. Пробуем!

```

public abstract class Figure {
    private final String type;
    protected final double[] sides;

    protected Figure(double[] sides) {
        this.type = getType();
        this.sides = sides;
    }

    protected String getType() {
        return getClass().getSimpleName();
    }

    protected abstract double getArea();

    protected double getPerimeter() {
        double perimeter = 0;
        for (double side : sides) {
            perimeter += side;
        }
        return perimeter;
    }

    @Override
    public String toString() {
        return getType() + " - area: " + getArea() +
            ", perimeter: " + getPerimeter();
    }
}

```

Да, как видите вы можете менять видимость метода у класса. Ведь вы переопределяете метод и конечно же можете менять не только реализацию, но и видимость. Аргументы метода и возвращаемый тип поменять не получится. В этом случае пишите свои методы.

```

public class Rectangle extends Figure {

    public Rectangle(double a, double b) {
        super(new double[]{a, b});
    }

    @Override
    protected String getType() {
        return "Прямоугольник";
    }

    @Override
    protected double getArea() {
        return sides[0] * sides[1];
    }

    @Override
    public double getPerimeter() {
        return 2 * super.getPerimeter();
    }

    @Override
    public String toString() {
        return getType() + " " + "площадь: " + getArea();
    }
}

```

И теперь к самому важному, зачем мы это все делали. Для того чтобы мой метод упростить. Давайте же посмотрим что с ним стало.

```

3  public static void main(String[] args) {
4      Figure[] figures = new Figure[3];
5      figures[0] = new Circle(radius: 2);
6      figures[1] = new Rectangle(a: 5, b: 6);
7      figures[2] = new Triangle(a: 3, b: 4, c: 5);
8
9      for (Figure figure : figures) {
10         print(figure.toString());
11     }
12 }

```

Main > main()

run: Main x

```

/usr/lib/jvm/java-1.8.0-openjdk-amd64/bin/java ..
Circle - area: 12.56, perimeter: 12.56
Прямоугольник площадь: 30.0
Triangle - area: 6.0, perimeter: 12.0
Process finished with exit code 0

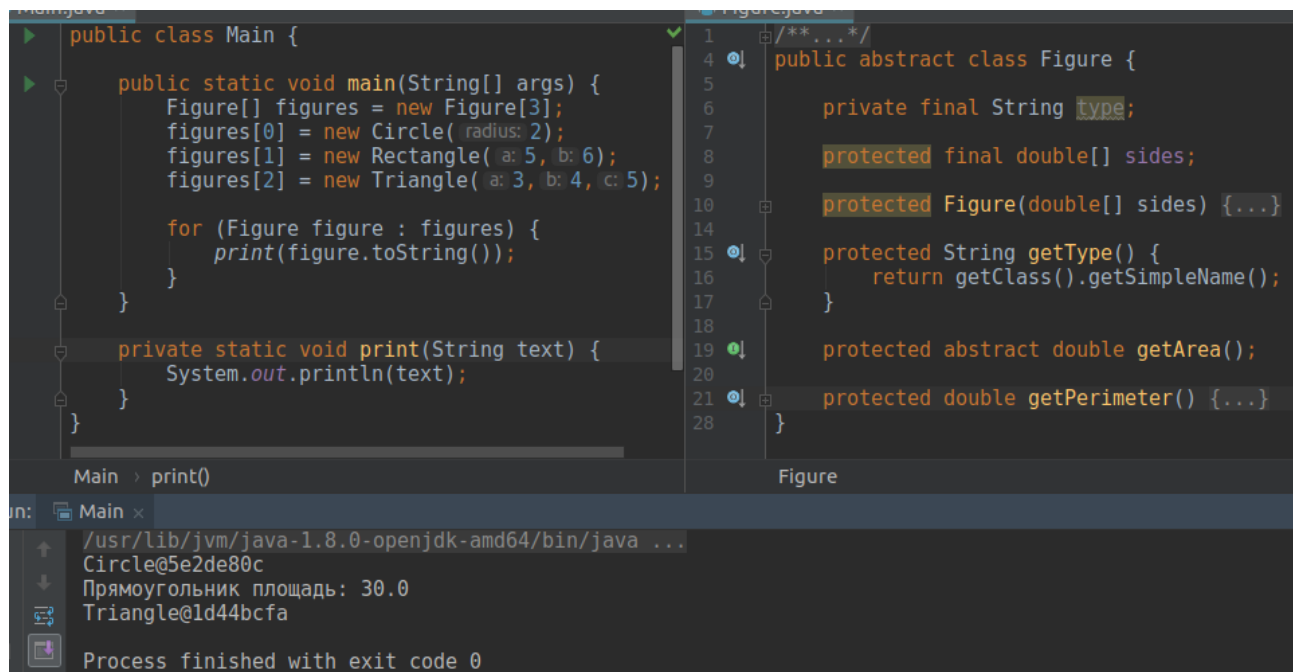
```

Как видите теперь нам достаточно у каждого элемента массива вызвать метод `toString` и каждый класс сам предоставит нам нужную информацию по объекту. Если же вам нужен более кастомизированный метод, то напишите его. Например метод типа

`getDescription(boolean includePerimeter, boolean includeArea)`

и просто проверяйте переданные аргументы и выдавайте строку. А для опиания по умолчанию можете использовать уже готовый метод `toString`.

И последний вопрос который нужно задать – а что будет если я вызову метод `toString` у объекта где нет переопределения этого метода? Ну что ж, давайте посмотрим на это.



```
public class Main {
    public static void main(String[] args) {
        Figure[] figures = new Figure[3];
        figures[0] = new Circle( radius: 2);
        figures[1] = new Rectangle( a: 5, b: 6);
        figures[2] = new Triangle( a: 3, b: 4, c: 5);

        for (Figure figure : figures) {
            print(figure.toString());
        }
    }

    private static void print(String text) {
        System.out.println(text);
    }
}

/**...*/
public abstract class Figure {
    private final String type;
    protected final double[] sides;
    protected Figure(double[] sides) {...}

    protected String getType() {
        return getClass().getSimpleName();
    }

    protected abstract double getArea();
    protected double getPerimeter() {...}
}
```

Output:

```
Circle@5e2de80c
Прямоугольник площадь: 30.0
Triangle@1d44bcfa
Process finished with exit code 0
```

Как видите у всех видов кроме прямоугольника вместо внятного описания какая-то абракадабра. Вот поэтому нужно переопределять метод `toString` чтобы иметь понимание с каким объектом мы имеем дело. Кстати, в этот метод можете передать какие-нибудь приватные поля.

В любом случае решать вам.

Для закрепления напишите такой код.

Абстрактный класс для животного и наследники типа Собака, Утка и Рыба. Для каждого вида сделать переопределяемый метод для движения и поле “кличка”. Написать внятное описание для каждого объекта и в цикле вывести в мейне. Можете усложнить себе задачу на свое усмотрение.