Switch

Еще пара слов об использовании

Содержание

1. Когда кейсы разные, но результат одинаков

Начнем с того, что решим одну задачу из предыдущей лекции. Предположим нам нужно отдать количество дней в месяца по строковому параметру. Мы можем написать следующий код.

```
private static int getDaysForMonth(String month) {
    switch (month) {
        case "january":
            return 31;
        case "february":
            return 28;
        default:
            throw new IllegalArgumentException("not recognized month " + month);
    }
}
```

Я не стал писать все 12 кейсов, думаю это и так понятно. И здесь у нас встанет вопрос – а откуда мы знаем, что в аргумент передадут имя месяца в нижнем регистре? А что если придет с заглавной буквы? Ну мы можем написать еще один кейс, например так

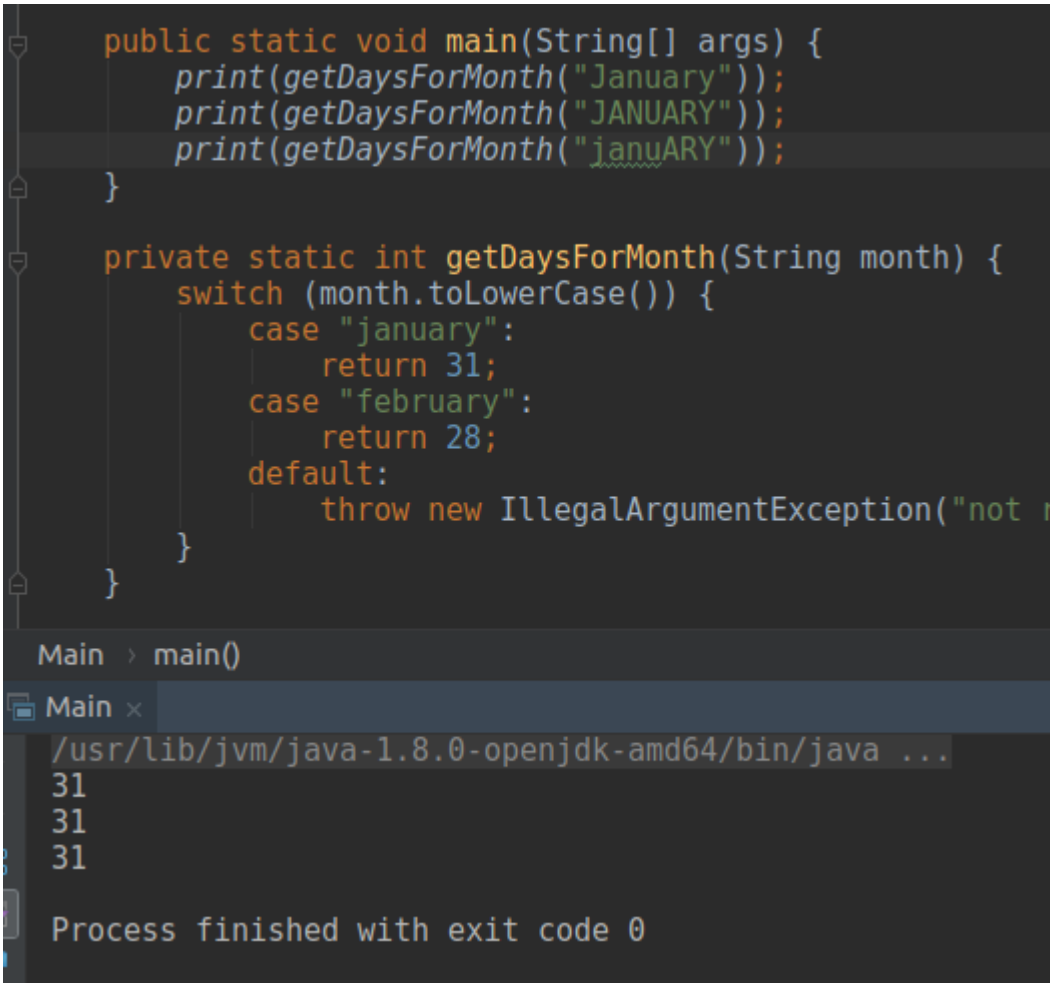
```
private static int getDaysForMonth(String month) {
    switch (month) {
        case "january":
            return 31;
        case "January":
            return 31;
        case "february":
            return 28;
        default:
            throw new IllegalArgumentException("not recognized month " + month);
    }
}
```

Эм, ну ок. И тут кто-то скажет, а что если вообще все в верхнем регистре пришло? Писать еще один кейс и возвращать 31?

Конечно же нет. Для строковых данных есть другой прием. Чтобы сравнивать было проще мы можем привести обе стороны сравнения в один регистр. Например так.

```
private static int getDaysForMonth(String month) {
    switch (month.toLowerCase()) {
        case "january":
            return 31;
        case "february":
            return 28;
        default:
            throw new IllegalArgumentException("not recognized month " + month);
    }
}
```

И здесь у нас аргумент меняется чтобы было проще сравнить. Можете проверить в мейн методе разные варианты.



Ладно, мы нашли способ сделать это. А что если название месяца пришло на кириллице? Не january, а например январь? У нас же нет гугл переводчика внутри кода, так? Ну да, на самом деле не самый удачный пример, но давайте на нем мы поймем как работает switch. Для этого давайте напишем вот как

```java
private static int getDaysForMonth(String month) {
    switch (month.toLowerCase()) {
        case "january":
        case "январь":
            return 31;
        case "february":
            return 28;
        default:
            throw new IllegalArgumentException("not
    }
}
```

У нас 2 кейса подряд и один результат. Это равнозначно коду если бы вы написали if (“january”.equals(month.toLowerCase()) || “январь”.equals(month.toLowerCase()))

Да, если у вас по разным кейсам один и тот же результат, то вы можете написать так.

2. Пишем 1 return в методе

А теперь рассмотрим такой вопрос: А что если мне не нужно сразу возвращать значение и вообще мы же решили что в 1 методе бестпрактис писать 1 return. Для наглядности представим что нам нужно вернуть описание типа “в месяце столько дней”. Пока давайте на английском чтобы было проще с окончаниями.

```java
public static void main(String[] args) {
    print(getDaysForMonth("January"));
}

private static String getDaysForMonth(String month) {
    int days;
    switch (month.toLowerCase()) {
        case "january":
            days = 31;
        case "february":
            days = 28;
        default:
            days = 0;
    }
    return month + " has " + days + " days.";
}
```

```
Main › getDaysForMonth()
un:  Main ×
/usr/lib/jvm/java-1.8.0-openjdk-amd64/bin/java ...
January has 0 days.

Process finished with exit code 0
```

Так, стоп. Мы должны были вернуть строку что в январе 31 день. А не 0. Что пошло не так?

Дело в том, что если даже в первом кейсе у вас совпадение, то компилятор идет вниз по другим кейсам и конечно же схватит дефолтный. Помните прерывание циклов? Та же история и здесь. Чтобы такого не происходило, нужно прерывать прорверку. Ведь вспомните как мы находили первый отрицательный член массива в цикле и прерывались.

Когда нам было бы удобно не писать break в switch? Когда вам например нужно при совпадении с другими кейсами добавить информации. Например вам нужен код который выполнится в дефолт для всех кейсов и чтобы его не писать 12 раз вы его пишете в дефолте и все. Т.е. свитч без брейков больше похож на if без else для последней ветки. Поэтому нужно запомнить нюанс с дефолтным кейсом и использовать по назначению. Обратите внимание, я нашел применение свитчу без брейков и не говорю вам обязательно писать брейки иначе все поломается. Нет, это не так. Для каждой конструкции языка джава можно найти применение и использовать в том или ином случае.

Ок, а что если мне нужно написать какой-нибудь код с проверкой, я смогу это сделать внутри кейса? Давайте попробуем.

```java
public static void main(String[] args) {
    print(getDaysForMonth("January"));
    print(getDaysForMonth("february"));
}

private static String getDaysForMonth(String month) {
    String result;
    int days = -1;
    switch (month.toLowerCase()) {
        case "january":
            days = 31;
        case "february":
            days = 28;
        default:
            if (days > 0) {
                result = month + " has " + days + " days.";
            } else {
                throw new IllegalArgumentException("invalid month " + month);
            }
    }
    return result;
}
```

Main › getDaysForMonth()

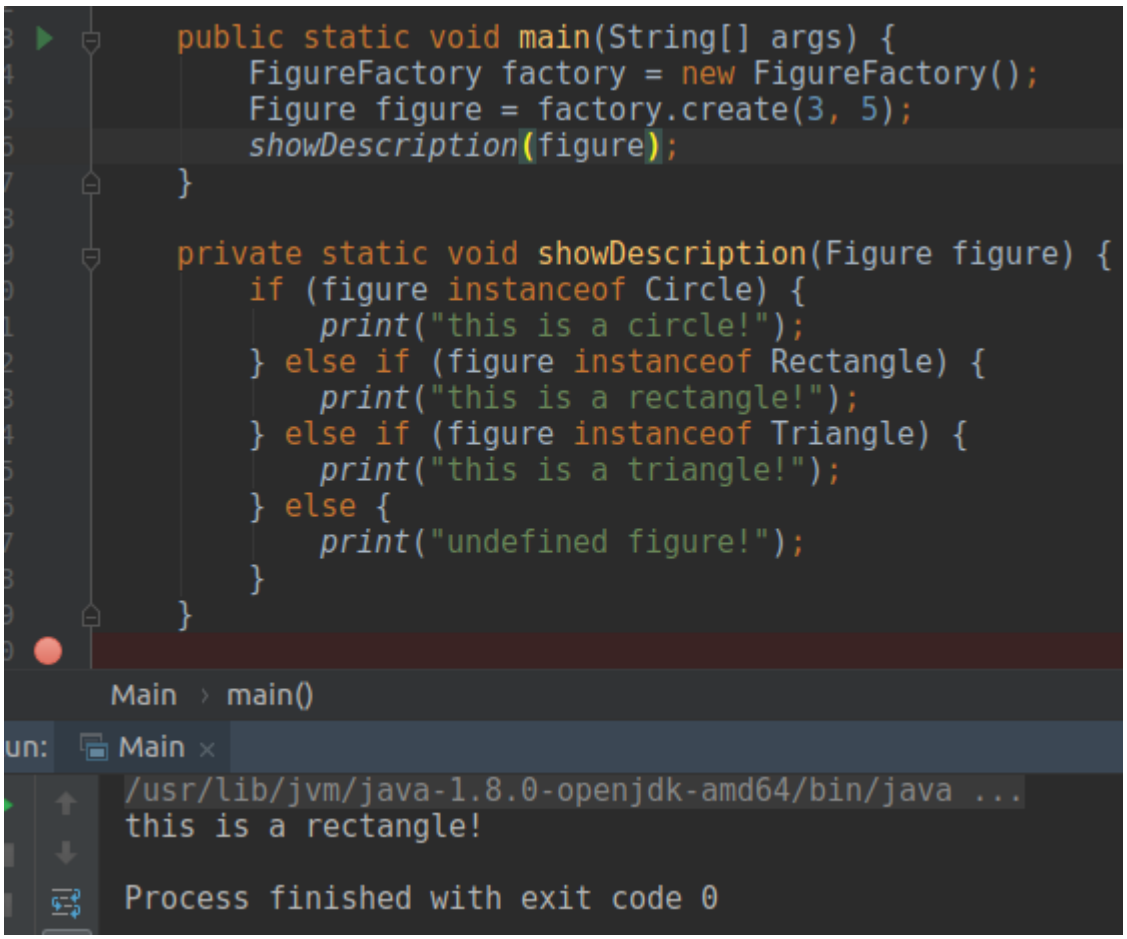
Main ×

```
/usr/lib/jvm/java-1.8.0-openjdk-amd64/bin/java ...
January has 28 days.
february has 28 days.

Process finished with exit code 0
```

Да, мы можем проверить в дефолтном кейсе заходили ли мы в другие кейсы и присвоили ли значение дням и тогда уже приписать какой-нибудь текст к этим дням и в конце метода у нас будет простой return result; Сделайте себе привычку возвращать в методе 1 раз в конце и ту переменную которую обьявили в начале метода. Тогда любой читатель увидит что происходит в методе.

3. Нечисловое и нестроковое представление констант

И рассмотрим еще одну интересную составляющую языка джава – enum. Представьте что мы должны написать метод, который по типу фигуры выведет что-то на экран, но мы не хотим писать это в классе фигуры или же не имеем доступа менять классы. Давайте вспомним свитч для фигур. Там мы смотрели на количество аргументов и создавали нужные обьекты – круг, прямоугольник или треугольник. А теперь у нас встанет вопрос – а что если у нас уже есть обьект, но метод принимает аргументом тип Figure и мы не знаем какого оно конкретного вида и нам нужно исходя из этого вывести в консоль строку. Здесь мы можем использовать хитрую проверку instanceOf. Давайте посмотрим.



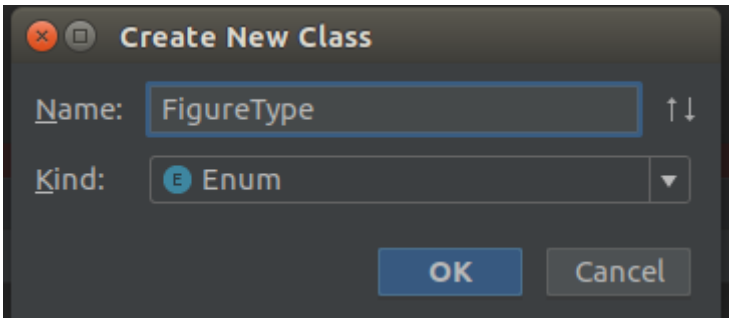
Так, мы создаем фабрику фигур. После чего порождаем обьект фигуры и мы не знаем какого оно типа. После чего отдаем его в метод определения описания. И с помощью instanceof узнаем его конечный тип класса. Сразу скажу – это не самое лучшее решение, оно долгое. Можете проверить в цикле миллион или 10 миллионов обьектов и измерить время.

И здесь кто-то скажет – подождите, мы же в классе фигуры написали поле тип. Там хранили строковую переменную которая имени класса, так может ее использовать? Ну да, можно. Тогда вам нужно будет написать свитч такой.

Стоп, что же произошло? А все просто – мы же дали возможность переопределить тип прямоугольника и там у нас тип написан на кириллице. Ок, даже если и не дать переопределить тип явно в наследнике. То у нас получается что могут возникнуть проблемы тогда, когда мы решим переименовать класс например круга. Ведь когда идея рефакторит имя класса она не может найти ваш метод где используется в нижнем регистре “circle”. Как же быть? Мы бы хотели иметь не строку, а нечто такое, которое бы подходило под тип идеально. И для этого в джава есть штука под названием Enum. Если у вас стоит задача просто перечислить типы классов например или просто список чего-либо, но вам не нужны строковые ресурсы или числовые, то вы всегда можете сделать вот так.

```
    public static void main(String[] args) {
        FigureFactory factory = new FigureFactory();
        Figure figure = factory.create(3, 5);
        showDescription(figure);
    }

    private static void showDescription(Figure figure) {
        switch (figure.getType().toLowerCase()) {
            case "circle":
                print("this is a circle!");
                break;
            case "rectangle":
                print("this is a rectangle!");
                break;
            case "triangle":
                print("this is a triangle!");
                break;
            default:
                print("undefined figure!");
        }
    }
```

```
Main › main()
Main
/usr/lib/jvm/java-1.8.0-openjdk-amd64/bin/java ...
undefined figure!

Process finished with exit code 0
```



И просто перечислим внутри все виды фигур.

```
public enum FigureType {
    CIRCLE,
    RECTANGLE,
    TRIANGLE
}
```

Теперь же давайте использовать этот тип – специальный тип констант (видите что они все написаны капсом?) для определения типа фигуры.

```java
public abstract class Figure {

    private final FigureType type;

    protected final double[] sides;

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

    public FigureType getType() {
        return type;
    }

    protected abstract double getArea();

    protected double getPerimeter() {...}
}
```

Вместо строкового типа у нас конкретный тип фигуры. Передаем его в конструкторе и даем паблик метод для тех, кто хочет знать какого типа фигура. Теперь, внесем правки в наследники

```java
public class Circle extends Figure {

    public Circle(double radius) {
        super(new double[]{radius}, FigureType.CIRCLE);
    }

    @Override
    public double getPerimeter() { return 2 * 3.14 * sides[0]; }

    @Override
    public double getArea() { return 3.14 * sides[0] * sides[0]; }
}
```

Заметьте, что мы передаем в супер т.е. конструктор родительского класса константу круга как будто она лежит в простом классе паблик статик константой.

Так же исправим для прямоугольника метод toString в котором мы использовали тип (хотя вы можете использовать тип иначе, но об этом в следующей лекции).

```java
public class Rectangle extends Figure {

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

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

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

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

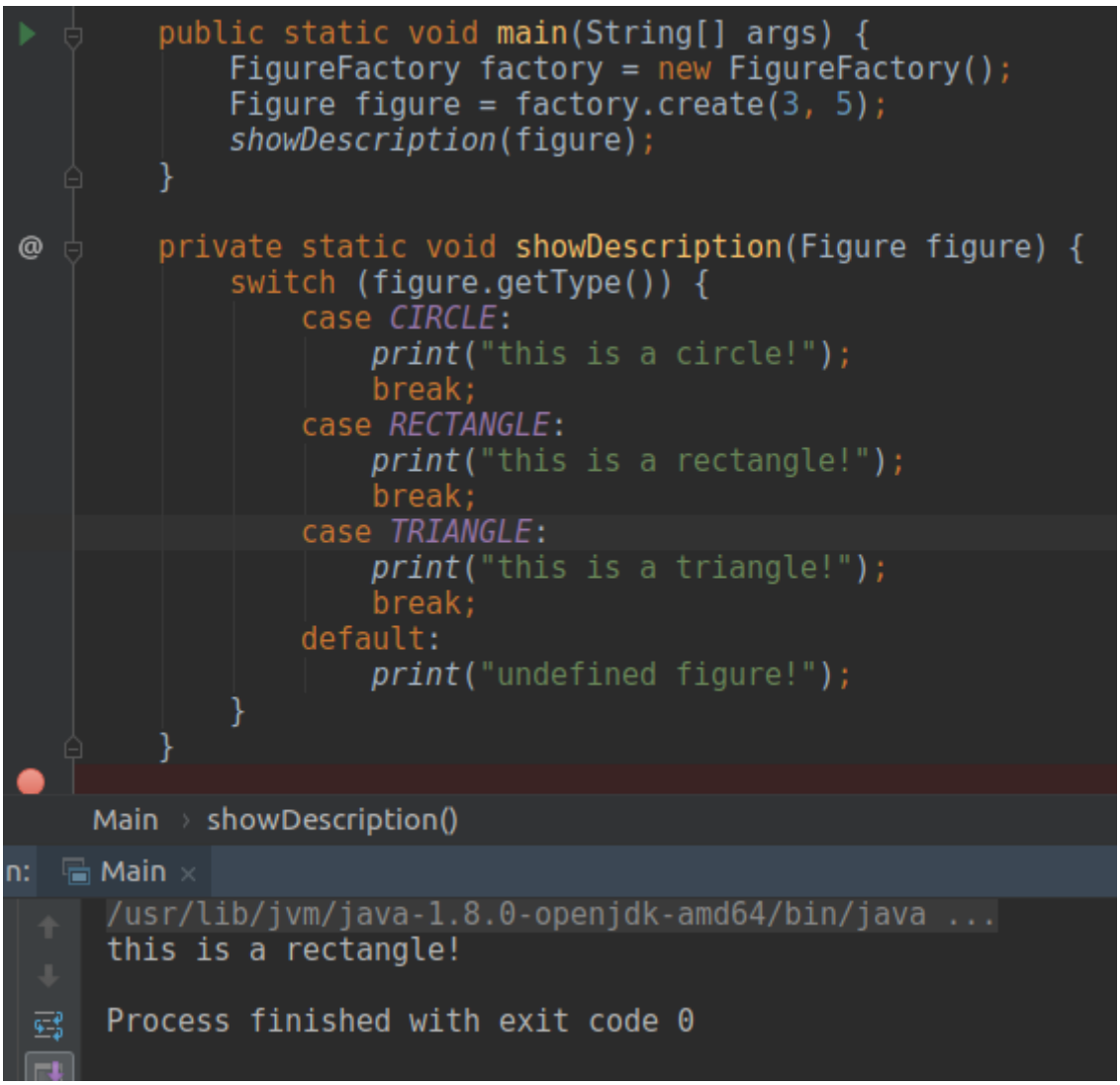
```java
public class Triangle extends Figure {

    public Triangle(double a, double b, double c) {
        super(new double[]{a, b, c}, FigureType.TRIANGLE);
    }

    @Override
    public double getArea() {
        double p = (sides[0] + sides[1] + sides[2]) / 2;
        return Math.sqrt(p * (p - sides[0]) * (p - sides[1]) * (p - sides[2]));
    }
}
```

И теперь уже вернемся в наш мейн класс. Для чего мы поменяли наши строки на енам – перечисление констант. А вот для чего. Во-первых не нужно использовать instanceof и во вторых не нужно беспокоиться что при рефакторинге имени класса у вас сломается другой метод в другом месте.

Теперь когда у вас появится новый тип фигуры вы внесете правки в енам класс типа и сможете отследить где они используются и добавить новый кейс в свитч где бы он ни был.

Это вполне себе часто используемое решение. Вопрос о том, насколько хорошо давать другим знать о классе такие вещи как тип иной и не будет рассмотрен. Это более серьезный вопрос который выходит за рамки лекций по джаве для новичков.

А пока попробуйте запомнить – если вам нужен список констант не числового и не стркового типа, а просто константы, то вам может подойти enum. И даже то, что я не рекомендую использовать их не должно вам помешать знать что это такое. А что использовать вместо этого больше относится к андроид. В простой джаве нет хорошего ответа (кроме как простых констант). Ведь если помните, простой примитив занимает меньше места в памяти чем целый обьект. А если вы заметили, мы создали класс enum class.



Если вы не писали задание из предыдущей лекции на тему разных видов работников, то сейчас самое время. Поменяйте абстрактный класс работника таким образом, чтобы мы знали о конкретном виде из другого класса. Напишите класс для подбора персонала и отсеивайте всех джунов, а всем кто выше джуна предлагайте высокую зарплату.

Если у вас возникают вопросы и что-то непонятно вы всегда можете задать мне вопрос в телеграм (@JohnnySC) или в группе (@easyCodeRuChat) прикрепленной к каналу (https://t.me/easyCodeRu) этих лекций.