Фабрика обьектов

Более удобный способ передать аргументы

Содержание

1. Фабрика обьектов

Итак, мы написали абстрактный класс для геом.фигуры, унаследовали 3 других класса и все равно мы порождаем обьекты руками. Хотелось бы задаться таким вопросом – а можно мы не будем каждый раз сами решать какой обьект создать – круг, прямоугольник или треугольник? Ведь в чем отличие этих обьектов? Давайте посмотрим внимательно. У абстрактного класса 1 поле вида массив сторон. И все наши обьекты в первую очередь различаются количеством сторон. Если так, то было бы неплохо написать метод, который бы создавал нужные обьекты. И вот вопрос – где написать этот метод? Мы не можем написать метод в пустоте. В джава нам нужен как минимум класс для этого чтобы туда записать метод. И методы бывают статик и нестатик. Мы можем написать статик метод и вызывать его откуда угодно, а можем сделать иначе. В любом случае нам нужен класс для этих целей. Класс который порождает другие обьекты принято называть в джава фабрикой(заводом). Ведь представьте для аналогии простой завод по созданию автомобилей. Каждый автомобиль не делается в гараже каком-нибудь, для этого нужен большой завод. Там уже внутри вся логика по которой собирается конкретная машина. А на выходе мы имеем разные реализации.

Давайте напишем фабрику для порождения обьектов геометрической фигуры на основе количества сторон.

```
public class FigureFactory {

    public Figure create(double[] sides) {
        if (sides.length == 1) {
            return new Circle(sides[0]);
        } else if (sides.length == 2) {
            return new Rectangle(sides[0], sides[1]);
        } else if (sides.length == 3) {
            return new Triangle(sides[0], sides[1], sides[2]);
        } else {
            throw new IllegalArgumentException("can't create object with arguments " + sides);
        }
    }
}
```

У класса 1 метод для порождения обьекта, он получает аргументом массив сторон и смотрит на его длину. Если в нем 1 число, то создать круг, если 2 то прямоугольник, а если 3 то треугольник. Во всех других случаях выкинуть ошибку. Ок, давайте тогда посмотрим как его использовать в мейн методе.

```
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);
```

Так было раньше. А теперь заменим явное создание обьектов обращением к фабрике.

```
public static void main(String[] args) {
    Figure[] figures = new Figure[3];
    FigureFactory factory = new FigureFactory();
    figures[0] = factory.create(new double[]{2});
    figures[1] = factory.create(new double[]{5,6});
    figures[2] = factory.create(new double[]{3,4,5});

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

Теперь так. Мы создали обьект фабрики и отдаем ему уже аргументы. Но стоп, стало хуже вроде как. Раньше да, мы писали сразу нужный обьект и передавали нужные данные в аргументы, а теперь какая-то фабрика за нас решает какой обьект создать и мало того, делает это не самым красивым способом. Насчет того, что теперь логика создания внесена в метод фабрики это так и нужно. Это нормально. А что насчет того что мы должны передать массив чисел, согласен. Неприятно писать лишний раз new double[] { } . Эх, было бы классно передать любое количество аргументов так же как мы передавали в конструкторы конечных классов наследников фигуры.

И у меня для вас есть хорошая новость! Давайте посмотрим на то, как может выглядеть метод создания в фабрике.

```
public Figure create(double... sides) {
    if (sides.length == 1) {
        return new Circle(sides[0]);
    } else if (sides.length == 2) {
        return new Rectangle(sides[0], sides[1]);
    } else if (sides.length == 3) {
        return new Triangle(sides[0], sides[1], sides[2]);
    } else {
        throw new IllegalArgumentException("can't create object
    }
```

Так, действительно, аргумент поменялся с массива на…. Что это такое? Но как видим весь код внутри метода не поменялся. Что за чертовщина? Если мы не знаем сколько придет аргументов в метод, то мы можем использовать мой любимый трюк. Троеточие значит что

могут прийти аргументы в любом количестве, но мы их примем как массив. Давайте посмотрим на мейн, как он теперь поменялся.

```
public static void main(String[] args) {
    Figure[] figures = new Figure[3];
    FigureFactory factory = new FigureFactory();
    figures[0] = factory.create(2);
    figures[1] = factory.create(5,6);
    figures[2] = factory.create(3,4,5);

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

Кстати, даже сама идея предложила убрать массив из аргумента. Как видите теперь стало намного лучше. Мы можем передать любое количество аргументов числового типа и все будет хорошо. То же самое работает и для обьектов. Давайте перепишем метод где в цикле выдается описание и уберем уже массив фигур.

```
public static void main(String[] args) {
    FigureFactory factory = new FigureFactory();
    print(
            factory.create(2),
            factory.create(5, 6),
            factory.create(3, 4, 5)
    );
}

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

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

Здесь важно понимать, что методы print разные. Один принимает аргументом любое количество фигур, а другой строку. И посмотрите как меньше стало кода в мейн методе. Нам теперь не нужно создавать массив фигур и туда класть обьекты. Можно сразу вот так передать в метод и там в цикле все выведется на экран.

2. Switch вместо if else if else if….

И давайте вернемся к нашему методу фабрики. Посмотрите на него. Мы каждый раз сравниваем с количеством аргументов длину массива. Вы не устали писать if else ? Я устал и для этого конкретного случая мы можем использовать другую конструкцию из языка Джава.

Ставим курсор на первый if и нажимаем Alt+Enter. Выбираем пункт Replace ‘if’ with ‘switch’ и смотрим что происходит

```
public Figure create(double... sides) {
    switch (sides.length) {
        case 1:
            return new Circle(sides[0]);
        case 2:
            return new Rectangle(sides[0], sides[1]);
        case 3:
            return new Triangle(sides[0], sides[1], sides[2]);
        default:
            throw new IllegalArgumentException("can't create object with arguments " + sides);
    }
}
```

В switch нужно передать какое-нибудь число или строку и мы сможем сравнить с какими-то значениями. Когда написано switch(length) case 1 это значит сравнить length со случаем равенства с единицей т.е. то же что было до этого. If (length == 1) И посмотрите внимательно на синтаксис, после ключевого слова case идет значение и двоеточие : после чего идет код. И в конце уже нас ждет ключевое слово default которое отработает в случае если ни один из вышеупомянутых кейсов не был отработан. Т.е. то же что и финальная ветка else в цепи.

Важно запомнить, что свитч работает только с числами и строками. И здесь может встать вопрос, всего 2 типа? Целое число и строка? А как же быть в других случаях? Нет конечно. Есть еще одна штука. И о ней я расскажу в следующей лекции.

А пока можете попробовать свитч со строкой

напишите метод который принимает строкой аргумент месяца и выдает количество дней в этом месяце (например за 2021 год)

Если хотите усложнить себе жизнь принимайте на 2 языках - английском и русском.

С этой задачей мы продолжим следующую лекцию.

И для закрепления фабрики :

Напишите фабрику разработчиков. У вас должен быть абстрактный работник с запрлатой и опытом и 3 уровня работников – Junior, Middle, Senior реализации. Написать фабрику в которой определить по опыту или зарплате какой нужно предоставить обьект работника. А еще можно сделать метод который принимает аргументом какое-то количество технологий которые должны знать работники и в зависимости от количества выбирается уровень работника.