

Копирование объектов

Привет!

В предыдущем видео мы с вами увидели, что объекты передаются по ссылкам. Изменяя объект, даже если мы его передали в какой-то метод, мы меняем его везде, поскольку он один, а разные переменные, через которые мы к нему обращаемся, — это всего лишь ссылки на этот же объект.

Как же быть, если мы хотим менять объект, но не хотим, чтобы он менялся везде? Для этого нам необходимо создать его копию, о чём сейчас мы и поговорим.

Вернёмся к классу `Product`. Если мы не хотим делать его иммутабельным и хотим сохранить в нём сеттеры, чтобы его можно было менять, но при этом хотим защитить его от изменений после добавления в заказ, мы можем просто копировать его при этом добавлении. Если мы поменяем исходный продукт, то копия, добавленная в этот заказ, меняться не будет.

```
public void addProduct(Product product) {  
    Product copy = new Product(product.getName(),  
    product.getPrice());  
    //add copy to order  
}
```

Я напомним, что если мы напишем...

```
public void addProduct(Product product) {  
    Product copy = product;  
    //add copy to order  
}
```

...то это будет не копия переданного в метод объекта, а копия ссылки на один и тот же объект в памяти. Возвращаем код обратно и оставим копирование.

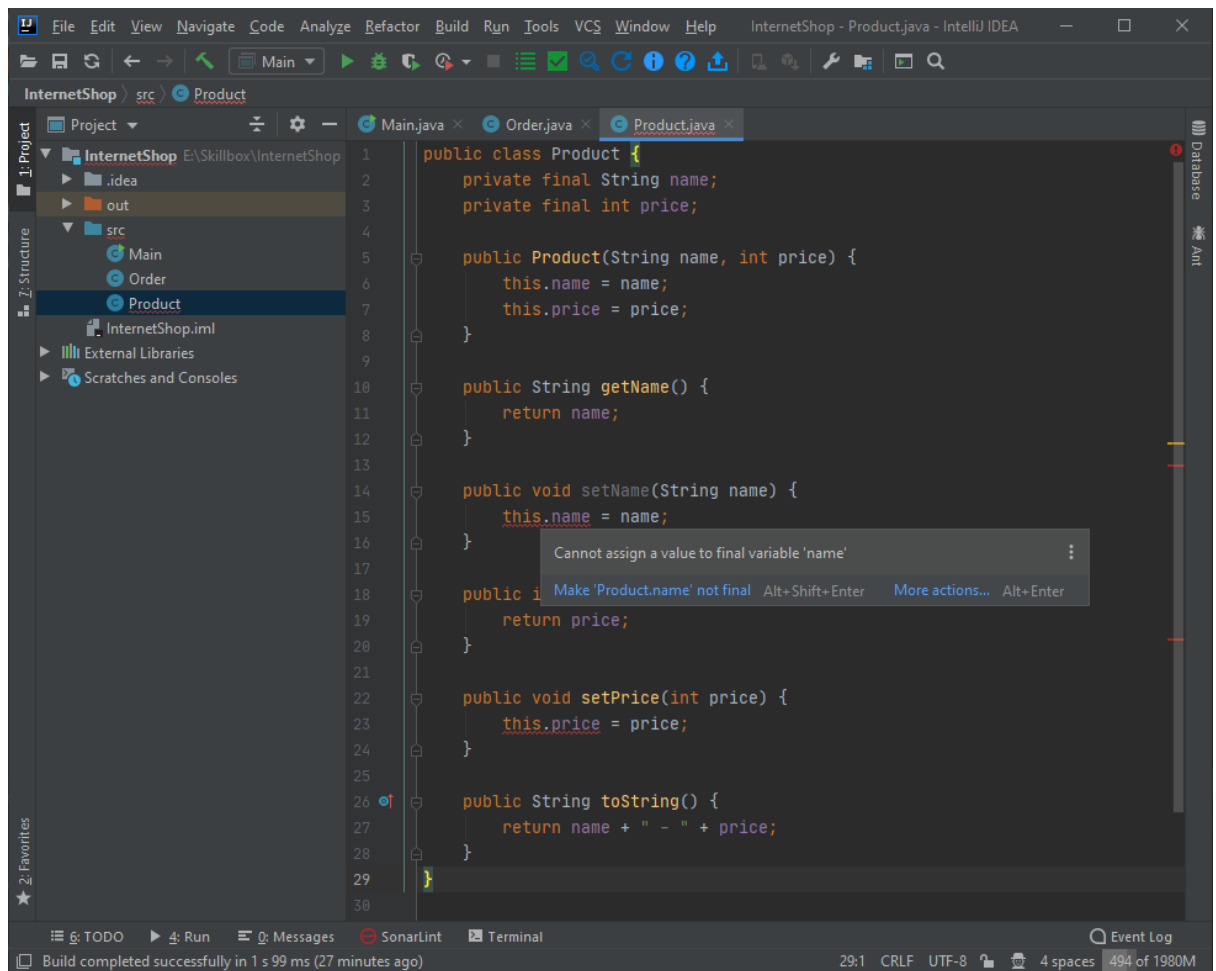
```
public void addProduct(Product product) {  
    Product copy = new Product(product.getName(),  
product.getPrice());  
    //add copy to order  
}
```

Но это не очень хороший способ, поскольку сам объект остаётся изменяемым. Если мы потом сделаем геттер, с помощью которого будем получать товары из заказа, то уже после их получения мы тоже сможем вносить в них изменения и снова придётся прописывать их копирование.

```
public Product getNextProduct() {  
    Product copy = new Product(...);  
}
```

Поэтому так делать не очень удобно, тем более что мы уже видели механизм, с помощью которого мы можем сделать объекты класса `Product` неизменяемыми, то есть иммутабельными.

Для этого мы все его переменные помечаем как `final`. Теперь то, что написано в сеттерах — изменение значений переменных, — сделать нельзя, поскольку переменные помечены как `final` (среда разработки тоже подсказывает об этом).



Что же делать? На самом деле мы можем сохранить сеттеры, но переписать их таким образом, чтобы в них создавалась копия исходного объекта.

```
public Product setName(String name) {
    return new Product(name, price);
}
```

В этом случае `name` у нас будет новый — из параметра метода, а `price` — старый, из текущего объекта. То же самое напишем и для цены.

```
public Product setPrice(int price) {
    return new Product(name, price);
}
```

Теперь мы можем менять оба свойства нашего объекта, но исходный объект будет оставаться неизменным. Фактически мы таким образом будем создавать копии исходного объекта с некоторыми изменениями и «зашьём» это копирование в сам класс, что сделает наш код проще.

Давайте посмотрим на ещё один пример. Представьте, что у вас есть компания, у которой есть название и банковские реквизиты (объект класса BankDetails).

```
public class Company {
    private final String name;
    private final BankDetails bankDetails;

    public Company(String name, BankDetails
bankDetails) {
        this.name = name;
        this.bankDetails = bankDetails;
    }

    public String getName() {
        return name;
    }

    public BankDetails getBankDetails() {
        return bankDetails;
    }

    public String toString() {
        return name + "\n" + bankDetails.toString();
    }
}
```

В этом классе есть пять полей, конструктор, сеттеры, геттеры и метод toString(), который возвращает содержимое этого класса в аккуратном формате.

```

public class BankDetails {
    private String billNumber;
    private String correspondenceBill;
    private String bikNumber;
    private String bankName;
    private String city;

    //getters + setters + constructor

    public String toString() {
        return "счёт: " + billNumber + "\n" +
            "к/с: " + correspondenceBill + "\n"
+
            "БИК: " + bikNumber + "\n" +
            "в " + bankName + " (" + city + ")";
    }
}

```

Это обычные классы с геттерами и сеттерами.

Создадим сначала объект класса BankDetails (реквизиты), зададим ему какие-нибудь параметры, создадим компанию с каким-нибудь названием и созданными реквизитами.

```

BankDetails details = new BankDetails();
bankDetails.setBillNumber("40702810500120002155");
Company company = new Company("Смарт-Экспресс",
details);
//some code

```

Теперь давайте поменяем банковские реквизиты (например, нам потребовалось для отдельного платежа изменить номер счёта в реквизитах) и распечатаем информацию о компании.

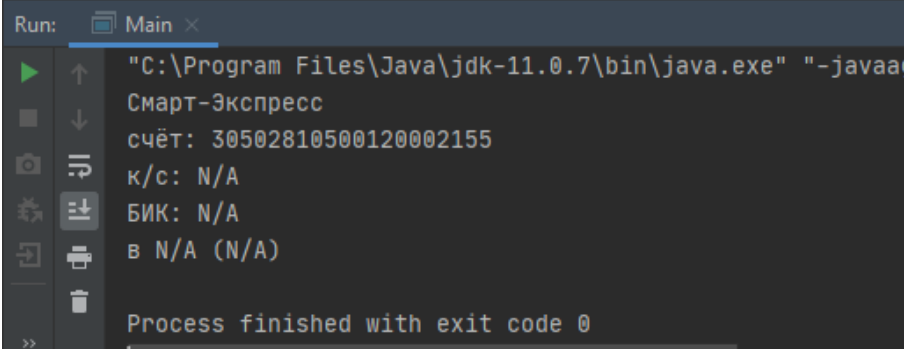
```

details.setBillNumber("30502810500120002155");
System.out.println(company);

```

Мы можем видеть, что внутри компании номер счёта тоже

поменялся.



```
Run: Main x
"C:\Program Files\Java\jdk-11.0.7\bin\java.exe" "-javaag
Смарт-Экспресс
счёт: 30502810500120002155
к/с: N/A
БИК: N/A
в N/A (N/A)
Process finished with exit code 0
```

Это происходит потому, что объект класса `BankDetails` в памяти один. Несмотря на то что мы его передали в конструктор, переменная `details` в конструкторе ссылается на этот же объект.

А мы хотели бы, чтобы реквизиты в компании не менялись никогда, тем более что эта переменная в классе `Company` уже помечена как `final`.

Как этого достичь? Во-первых, запретить изменения в классе `BankDetails` — сделать поля `final` внутри `BankDetails`, чтобы никто не мог их поменять.

```
private final String billNumber;
private final String correspondenceBill;
private final String bikNumber;
private final String bankName;
private final String city;
```

Сеттеры при этом перестанут работать. Затем создаём конструктор, который инициализирует все поля.

```
public BankDetails(String billNumber, String
correspondenceBill,
    String bikNumber, String bankName, String
city) {
    this.billNumber = billNumber;
    this.correspondenceBill = correspondenceBill;
    this.bikNumber = bikNumber;
```

```
this.bankName = bankName;  
this.city = city;  
}
```

Если всё же что-то меняется, в сеттере нужно создавать с помощью конструктора и возвращать новый объект класса BankDetails на основе данных из исходного, например:

```
public BankDetails setBillNumber(String billNumber)  
{  
    return new BankDetails(billNumber,  
        correspondenceBill, bikNumber, bankName, city);  
}
```

В данном случае billNumber будет взят из параметра метода, а остальные параметры конструктора — у текущего объекта класса BankDetails. Аналогично пишутся и все остальные сеттеры.

По сути, мы сделали объекты класса BankDetails неизменяемыми или, как их еще называют, **immutable** — **иммутабельными**. При любом изменении таких объектов исходные объекты не меняются, а создаются и возвращаются новые. И сделали мы это путём копирования тех свойств объектов, которые не менялись.

Разумеется, мы можем скопировать объект и полностью.

```
BankDetails copy = new BankDetails(  
    details.getBillNumber(),  
    details.getCorrespondenceBill(),  
    details.getBikNumber(),  
    details.getBankName(),  
    details.getCity()  
);
```

Итоги

Итак, мы рассмотрели примеры, когда необходимо копирование объектов, а также единственно верный способ такого копирования — путём создания нового объекта и присваивания его полям тех значений, которые есть у исходного объекта.

Важно, чтобы эти переменные:

- либо копировались по значению, как числа `int`;
- либо были `immutable`, как строки;
- либо сами были скопированы по отдельным полям, если это какие-то сложные неизменяемые объекты.