Enum

Привет!

Предыдущий материал был посвящён константам — специальным статическим переменным, которые можно задать заранее и нельзя будет изменить.

Этот материал познакомит вас с перечисляемыми типами, или enum, — от английского enumerate — «перечислять». Перечисляемые типы похожи на константы тем, что их нельзя изменить в процессе работы программы: они задаются лишь однажды при написании кода.

При разработке программ бывает нужно задать несколько значений одного типа. Например, в случае с уже знакомой коробкой передач это может быть тип этой коробки: автоматическая, ручная, роботизированная или вариативная. Как можно задать тип коробки?

Когда-то давно, когда в Java не было типа enum, делали так: создавали переменную «тип», обычно это было целое число — int:

```
public final int type;
```

и задавали константы с возможными значениями этого числа для каждого типа коробок передач:

```
public static final int TYPE_AUTOMATIC = 1;
public static final int TYPE_MANUAL = 2;
public static final int TYPE_ROBOT = 3;
public static final int TYPE_VARIATOR = 4;
```

а в конструктор передавали параметр:

```
public GearBox(int type) {
      this.type = type;
}
```

И создали бы геттер, чтобы всегда можно было узнать, какого типа созданная коробка передач.

То есть при создании новой коробки передач должны были определить её тип. Делалось это так:

```
GearBox gearBox = new GearBox(GearBox.TYPE_AUTOMATIC);
```

Это, кстати, одна из причин, почему константы стало принято делать публичными: чтобы их можно было использовать ровно в таких случаях, за пределами классов, в которых они заданы.

Но это неудобно, поскольку, во-первых, в конструктор можно передать какой-то другой int:

```
GearBox gearBox = new GearBox(75);
```

и становится совсем непонятно, что это за тип. А во-вторых, с этим int будет сложно работать: нужно помнить, какому значению какой int соответствует. И в этом случае по значению 75, как и по другим значениям, понять, каким типам коробок они соответствуют, будет практически невозможно. Эту информацию нужно будет где-то отдельно хранить.

Чтобы избежать всех этих сложностей, в Java появились так называемые перечисляемые типы, или enum, представляющие собой отдельные классы, в которых можно задать набор собственных значений, способных принимать переменные этих классов.

Создадим соответствующий enum:

```
enum GearBoxType {
      AUTOMATIC, MANUAL, ROBOT, VARIATOR
}
```

И теперь можно в конструктор передавать не int, а enum:

```
public GearBox(GearBoxType type) {
      this.type = type;
}
```

и из геттера тоже возвращать значение из этого enum:

```
public GearBox getType() {
      return type;
}
```

Передать в конструктор число или другое значение, не входящее в этот enum, теперь невозможно. Кроме того, если мы захотим узнать, какого типа у нас коробка передач, можно сделать так:

```
System.out.println(gearBox.getType());
```

Если у вас уже есть строка AUTOMATIC, её можно очень легко преобразовать в enum с помощью специального метода valueOf:

```
GearBoxType type = GearBoxType.valueOf("ROBOT");
GearBox gearBox = new GearBox(type);
```

Таким образом, enum — очень удобный тип, с помощью которого можно повысить качество, читабельность и поддерживаемость программного кода. Но это не единственный вариант, когда он может быть полезен. Посмотрим на ещё один пример использования enum.

Пусть это будет класс в том же интернет-магазине, отвечающий за создание задачи для службы доставки:

```
public class DeliveryOrder {
}
```

Представьте, что у вас есть метод или конструктор с несколькими параметрами. Пусть это будет конструктор, в который передаются три boolean-параметра:

```
public DeliveryOrder(boolean isPedestrian, boolean isFragile,
boolean isCold) {
}
```

isPedestrian — пеший или на автомобиле.

isFragile — хрупкий.

isCold — холодный, требует охлаждения.

И когда вы будете вызывать этот конструктор, вызов будет выглядеть так:

```
DeliveryOrder order = new DeliveryOrder(true, false, true);
```

В данном случае не очень понятно, что за параметры переданы. Если бы мы использовали enum для каждого из параметров, код выглядел бы проще. Создадим такие enum:

```
enum Pedestrian {
      YES,
      NO
}
```

```
enum Fragile {
```

```
      YES,
      NO
}

enum Cold {
      YES,
      NO
}
```

Поменяем в конструкторе boolean на enum:

```
public DeliveryOrder(Pedestrian value, Fragile value, Cold value)
{

}
```

И теперь вызов конструктора будет выглядеть гораздо понятнее:

```
DeliveryOrder order = new DeliveryOrder(Pedestrian.YES,
Fragile.NO, Cold.YES);
```

Кроме того, использование enum в данном случае поможет расширить диапазон значений: вместо двух YES и NO можно сделать три и большее количество значений. Например, сделать больше режимов доставки:

Переименовываем Pedestrian в DeliveryType, YES — в PEDESTRIAN, NO — в CAR. Добавляем BIKE.

```
enum DeliveryType {
           PEDESTRIAN,
           CAR,
           BIKE
}
```

Код стал значительно понятнее и удобнее.

Итоги

Вы познакомились с перечисляемыми типами — enum, которые позволяют создавать и использовать набор констант одного типа, повышая тем самым удобство и читабельность программного кода.

Глоссарий

enum