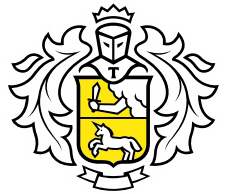


ООП 3

Николай Ивановский
Специалист по автоматизации CRM и интеграции.

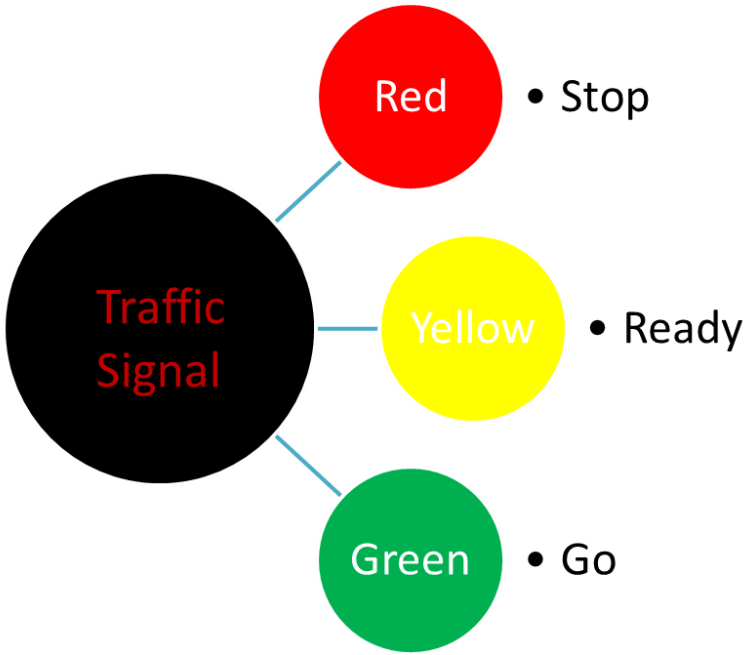
Tinkoff.ru

Перечисление

В простейшей форме перечисление - это список именованных констант.

Сигналы светофора:

- Красный
- Желтый
- Зеленый



Перечисление = enum

В java перечисления реализованы с помощью enum.

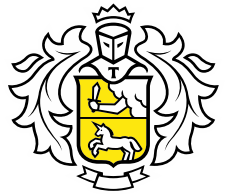
Enum – это специальный тип данных, который предоставляет заранее заданный набор значений на выбор.

```
public enum  TrafficSignal
{
    RED, YELLOW, GREEN;
}
```

```
public static void main(String[] args) {
 TrafficSignal.
}
```

```
valueOf(String name)                          TrafficSignal
values()                                     TrafficSignal[]
valueOf(Class<T> enumType, String name)                   T
GREEN                                         TrafficSignal
RED                                           TrafficSignal
YELLOW                                        TrafficSignal
```

Enum – это класс



На самом деле enum это обычный класс неявно унаследованный от `java.lang.Enum`

```
enum TrafficSignal { ... }  = class TrafficSignal extends java.lang.Enum { ... }
```

Внутренние ограничения языка не позволят нам использовать такой синтаксис, но обращаться с enum можно точно так же как с классом

Создать объект enum

```
TrafficSignal signal = TrafficSignal.GREEN;
```

Элементы перечисления – это статические экземпляры enum-класса

```
TrafficSignal.GREEN
TrafficSignal.YELLOW
```

То, что эти экземпляры статические, позволяет нам сравнивать их по ссылке

```
TrafficSignal signal = TrafficSignal.GREEN;
if (signal == TrafficSignal.GREEN) {
    System.out.println("Полный вперед");
}
```

Действия с enum

Все методы для работы с enum определены в `java.lang.Enum`

```
TrafficSignal signal = TrafficSignal.GREEN;
System.out.println("signal.name()=" + signal.name());
System.out.println("signal.toString()=" + signal.toString());
System.out.println("signal.ordinal()=" + signal.ordinal());
```

Результат:

```
signal.name()= GREEN
signal.toString()= GREEN
signal.ordinal()= 2
```

values() – метод возвращает массив элементов перечисления

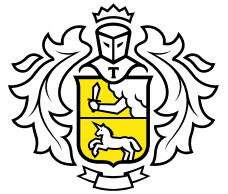
```
TrafficSignal signal = TrafficSignal.GREEN;
System.out.println(Arrays.toString(TrafficSignal.values()));
```

[RED, YELLOW, GREEN]

valueOf() – метод возврщает элемент enum по его имени

```
TrafficSignal signal = TrafficSignal.valueOf("GREEN")
```

Методы в enum?



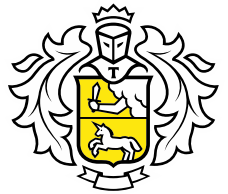
Внутри enum можно создавать методы.

```
public TrafficSignal getRedSignal(){
    return RED;
}
```

Они не статические, и вызываются через переменные класса enum

```
signal.getRedSignal()
```

Наследование в enum



Методы в enum можно переопределить в зависимости от экземпляра

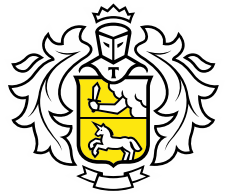
```
enum Direction {
    UP {
        public Direction opposite() { return DOWN; }
    },

    DOWN {
        public Direction opposite() { return UP; }
    };

    public abstract Direction opposite();

}
```

Обобщения



Обобщённое программирование — это такой подход к описанию данных и алгоритмов, который позволяет их использовать с различными типами данных без изменения их описания.

Давайте создадим класс, в котором нам нужно реализовать специфический вывод в консоль информации об объектах различного типа (с использованием фигурных скобок).

Реализация с помощью Object

```
class BoxPrinter {
    private Object val;

    public BoxPrinter(Object arg)
        val = arg;
    }

    public String toString() {
        return "{" + val + "}";
    }

    public Object getValue() {
        return val;
    }
}
```

```
BoxPrinter value1 = new BoxPrinter(new Integer(10));
        System.out.println(value1);
        Integer intValue1 = (Integer) value1.getValue();
        BoxPrinter value2 = new BoxPrinter("Hello world");
        System.out.println(value2);

        Integer intValue2 = (Integer) value2.getValue();
```

Здесь программист допустил ошибку, присваивая переменной типа Integer значение типа String.

Generics

Обобщения - это параметризованные типы. С их помощью можно объявлять классы, интерфейсы и методы, где тип данных указан в виде параметра.

```
class BoxPrinter<T> {
    private T val;

    public BoxPrinter(T arg) {
        val = arg;
    }

    public String toString() {
        return "{" + val + "}";

    }

    public T getValue() {
        return val;
    }
}
```

```
BoxPrinter<Integer> value1 = new BoxPrinter<Integer>(new Integer(10));
    System.out.println(value1);
    Integer intValue1 = value1.getValue();
    BoxPrinter<String> value2 = new BoxPrinter<String>("Hello world");
    System.out.println(value2);

    // Здесь повторяется ошибка предыдущего фрагмента кода
    Integer intValue2 = value2.getValue();
```

Этот пример даже не скомпилируется

Что же такое Generics

Определение класса с дженериком

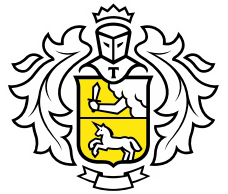
```
class BoxPrinter<T>
```

В треугольных скобках определяется тип, с который будет использоваться внутри класса

```
BoxPrinter <Integer>  value1
```

Когда мы создаем объект класса, мы сразу указываем с какии типом данных он будет работать. Тоесть все типы данных T будут заменены например на Integer

Несколько обобщенных типов в одном классе



В одном классе может быть неограниченное число объобщенных типов

```
class Pair<T1, T2> {
    T1 object1;
    T2 object2;

    Pair(T1 one, T2 two) {
        object1 = one;
        object2 = two;
    }

    public T1 getFirst() {
        return object1;
    }

    public T2 getSecond() {
        return object2;
    }
}
```

Алмазный синтаксис (Diamond syntax)

Алмазный синтаксис позволяет определить с какими именно типами данных будет работать класс

```
Pair<Integer, String> pair = new Pair<Integer, String>(6, "Apr");

Pair<Integer, String> pair = new Pair<String, String>(6, " Apr");//Ошибка
```

Однако нужно следить за совпадением указанных типов и передаваемых значений

Упрощенный синтаксис

```
Pair<Integer, String> pair = new Pair<>(6, " Apr");
```

Домашнее задание

Написать класс который будет реализовывать следующие методы для массива любого типа данных

- Вывести элемент массива в формате [<номер эдемента> = <вызов метода toString()>]
- Вывести массив поменяв два элемента местами
- Вывести перевернутый массив