Правда про enum-ы

(c) Effective Java

Юрий Литвинов
yurii.litvinov@gmail.com

27.02.2019г

Enum-ы

- Типобезопасны
- Автоматически получают пространство имен
- Возможны конструкторы и методы

```
public enum Apple { FUJI, PIPPIN, GRANNY_SMITH }
public enum Orange { NAVEL, TEMPLE, BLOOD }

enum Season {
    WINTER,
    SPRING,
    SUMMER,
    AUTUMN
}
```

Обещанная правда про Enum-ы

- Enum-ы — это классы
- Каждый элемент перечисления — это public static final поле
- Нет доступных конструкторов
  - Невозможно создать экземпляр, кроме элемента перечисления
  - Обобщение паттерна “Одиночка”
- Реализуют Comparable и Serializable

Методы

- Можно сравнивать с помощью ==
- name(), ordinal(), toString()

```
var season = Season.WINTER;
System.out.println(
    season.name() + ", " +
    season.toString() + ", " +
    season.ordinal()
);
```

WINTER, WINTER, 0

Статические методы

- valueOf()

```
String name = "WINTER";
Season season = Season.valueOf(name);

Season.valueOf(null); // NullPointerException
Season.valueOf("HOLIDAYS"); // IllegalArgumentException
```

- values()

```
System.out.println(Arrays.toString(Season.values()));
```

[WINTER, SPRING, SUMMER, AUTUMN]

- Автоматически добавляются компилятором

Поля

```
enum Type {
    INT(true),
    INTEGER(false),
    STRING(false);

    private final boolean primitive;

    Type(boolean primitive) { this.primitive = primitive; }

    public boolean isPrimitive() { return primitive; }
}
```

Методы

```
enum Direction {
  UP, DOWN;

  public Direction opposite() {
    switch (this) {
      case UP:
        return DOWN;
      case DOWN:
        return UP;
      throw new AssertionError("Unknown op: " + this);
    }
  }
}
```

Методы (constant-specific)

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

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

  public abstract Direction opposite();
}
```

Почему?

- Enum-ы — это иерархия классов
- Каждый элемент перечисления — это экземпляр своего наследника от базового класса-перечисления

Пример

```
enum Type {
    INT(true) {
        public Object parse(String string) { return Integer.valueOf(string); }
    },
    INTEGER(false) {
        public Object parse(String string) { return Integer.valueOf(string); }
    },
    STRING(false) {
        public Object parse(String string) { return string; }
    };

    private final boolean primitive;

    Type(boolean primitive) { this.primitive = primitive; }

    public boolean isPrimitive() { return primitive; }
    public abstract Object parse(String string);
}
```

fromString

```
// Implementing a fromString method on an enum type
private static final Map<String, Type> stringToEnum =
    Stream.of(values()).collect(
        toMap(Object::toString, e -> e));

// Returns Type for string, if any
public static Optional<Type> fromString(String symbol) {
    return Optional.ofNullable(stringToEnum.get(symbol));
}
```

- Сначала инициализируются элементы enum-а, затем остальные статические поля
  - Поэтому доступ к статическим полям из конструкторов элементов запрещён

Ещё пример

```
enum PayrollDay {
    MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY, SATURDAY, SUNDAY;

    private static final int HOURS_PER_SHIFT = 8;

    double pay(double hoursWorked, double payRate) {
        double basePay = hoursWorked * payRate;
        double overtimePay;
        switch (this) {
            case SATURDAY: case SUNDAY:
                overtimePay = hoursWorked * payRate / 2;
                break;
            default:
                overtimePay = hoursWorked <= HOURS_PER_SHIFT
                        ? 0 : (hoursWorked - HOURS_PER_SHIFT) * payRate / 2;
                break;
        }
        return basePay + overtimePay;
    }
}
```

Паттерн “Стратегия” на enum-ах

```
enum PayrollDay {
    MONDAY(PayType.WEEKDAY), TUESDAY(PayType.WEEKDAY),
    WEDNESDAY(PayType.WEEKDAY), THURSDAY(PayType.WEEKDAY),
    FRIDAY(PayType.WEEKDAY),
    SATURDAY(PayType.WEEKEND), SUNDAY(PayType.WEEKEND);

    private final PayType payType;

    PayrollDay(PayType payType) { this.payType = payType; }

    double pay(double hoursWorked, double payRate) {
        return payType.pay(hoursWorked, payRate);
    }
    ...
}
```

PayType

```
private enum PayType {
    WEEKDAY {
        double overtimePay(double hours, double payRate) {
            return hours <= HOURS_PER_SHIFT ? 0 :
                    (hours - HOURS_PER_SHIFT) * payRate / 2;
        }
    },

    WEEKEND {
        double overtimePay(double hours, double payRate) {
            return hours * payRate / 2;
        }
    };

    private static final int HOURS_PER_SHIFT = 8;

    abstract double overtimePay(double hrs, double payRate);

    double pay(double hoursWorked, double payRate) {
        double basePay = hoursWorked * payRate;
        return basePay + overtimePay(hoursWorked, payRate);
    }
}
```

Опасайтесь ordinal()

Неправильно:

```
public enum Ensemble {
    SOLO, DUET, TRIO, QUARTET, QUINTET,
    SEXTET, SEPTET, OCTET, NONET, DECTET;
    public int numberOfMusicians() { return ordinal() + 1; }
}
```

Правильно:

```
public enum Ensemble {
    SOLO(1), DUET(2), TRIO(3), QUARTET(4), QUINTET(5),
    SEXTET(6), SEPTET(7), OCTET(8), DOUBLE_QUARTET(8),
    NONET(9), DECTET(10), TRIPLE_QUARTET(12);

    private final int numberOfMusicians;
    Ensemble(int size) { this.numberOfMusicians = size; }
    public int numberOfMusicians() { return numberOfMusicians; }
}
```

Битовые поля

Как делали без enum-ов:

```
public class Text {
    public static final int STYLE_BOLD = 1 << 0;
    public static final int STYLE_ITALIC = 1 << 1;
    public static final int STYLE_UNDERLINE = 1 << 2;

    public void applyStyles(int styles) {
        // styles — побитовое "или"
        // text.applyStyles(STYLE_BOLD | STYLE_ITALIC);
    }
}
```

EnumSet

- Все возможности Set
- Внутри — long или long[]
  - Производительность сравнима с битовыми масками

```
public class Text {
    public enum Style { BOLD, ITALIC, UNDERLINE }

    public void applyStyles(Set<Style> styles) {
        // text.applyStyles(EnumSet.of(Style.BOLD, Style.ITALIC));
    }
}
```

EnumMap, пример

```
public class Plant {
    public enum LifeCycle { ANNUAL, PERENNIAL, BIENNIAL }

    private final String name;
    private final LifeCycle lifeCycle;

    public Plant(String name, LifeCycle lifeCycle) {
        this.name = name;
        this.lifeCycle = lifeCycle;
    }
}
```

EnumMap, пример без

```
var plantsByLifeCycle =
    (Set<Plant>[]) new Set[Plant.LifeCycle.values().length];
    // Indexed by Plant.LifeCycle.ordinal()

for (int i = 0; i < plantsByLifeCycle.length; i++) {
    plantsByLifeCycle[i] = new HashSet<>();
}

for (Plant p : garden) {
    plantsByLifeCycle[p.lifeCycle.ordinal()].add(p);
}
```

EnumMap, пример с

```
var plantsByLifeCycle =
    new EnumMap<Plant.LifeCycle, Set<Plant>>(Plant.LifeCycle.class);

for (Plant.LifeCycle lc : Plant.LifeCycle.values()) {
    plantsByLifeCycle.put(lc, new HashSet<>());
}

for (Plant p : garden) {
    plantsByLifeCycle.get(p.lifeCycle).add(p);
}
```

Или, через Stream API

```
System.out.println(Arrays.stream(garden)
    .collect(groupingBy(p -> p.lifeCycle,
        () -> new EnumMap<>(LifeCycle.class), toSet())));
```

EnumMap EnumMap-ов, пример без

```
public enum Phase {
    SOLID, LIQUID, GAS;

    public enum Transition {
        MELT, FREEZE, BOIL, CONDENSE, SUBLIME, DEPOSIT;

        private static final Transition[][] TRANSITIONS = {
            { null, MELT, SUBLIME },
            { FREEZE, null, BOIL },
            { DEPOSIT, CONDENSE, null }
        };

        public static Transition from(Phase src, Phase dst) {
            return TRANSITIONS[src.ordinal()][dst.ordinal()];
        }
    }
}
```

EnumMap EnumMap-ов, пример с

```
public enum Phase {
    SOLID, LIQUID, GAS;

    public enum Transition {
        MELT(SOLID, LIQUID), FREEZE(LIQUID, SOLID),
        BOIL(LIQUID, GAS), CONDENSE(GAS, LIQUID),
        SUBLIME(SOLID, GAS), DEPOSIT(GAS, SOLID);

        private final Phase from;
        private final Phase to;

        Transition(Phase from, Phase to) {
            this.from = from;
            this.to = to;
        }
        ...
    }
}
```

EnumMap EnumMap-ов, пример с (2)

```
public enum Transition {
    ...
    private static final Map<Phase, Map<Phase, Transition>> m =
      Stream.of(values()).collect(groupingBy(t -> t.from,
        () -> new EnumMap<>(Phase.class),
        toMap(t -> t.to, t -> t,
          (x, y) -> y, () -> new EnumMap<>(Phase.class))));

    public static Transition from(Phase from, Phase to) {
      return m.get(from).get(to);
    }
  }
}
```

Добавим новое состояние

```
public enum Phase {
    SOLID, LIQUID, GAS, PLASMA;

    public enum Transition {
        MELT(SOLID, LIQUID), FREEZE(LIQUID, SOLID),
        BOIL(LIQUID, GAS), CONDENSE(GAS, LIQUID),
        SUBLIME(SOLID, GAS), DEPOSIT(GAS, SOLID),
        IONIZE(GAS, PLASMA), DEIONIZE(PLASMA, GAS);

        // Дальше ничего не поменялось
        ...
    }
}
```