Lab: Enumerations and Annotations

Problems for exercises and homework for the "Java OOP Advanced" course @ SoftUni.

You can check your solutions here: https://judge.softuni.bg/Contests/524/Enumerations-and-Annotations-Lab .

Part I: Enumerations

1. Weekdays

Create **Enum** Weekday with the days from **Monday** through **Sunday**. Override **toString()**, which should return weekdays with a capital first letter, in format **"Monday"**, **"Tuesday"**, etc.

Create a class **WeeklyCalendar** that should have at least the methods:

- **void addEntry(String weekday, String notes)**
- **Iterable<WeeklyEntry> getWeeklySchedule()** - returns weekly entries sorted by day in **ascending order**

Create a class **WeeklyEntry** which should have constructor:

- **WeeklyEntry(String weekday, String notes)**

Override **toString()** of **WeeklyEntry** – "{**weekday**} - {**notes**}" (e.g. "Monday - sport", "Sunday - sleep")

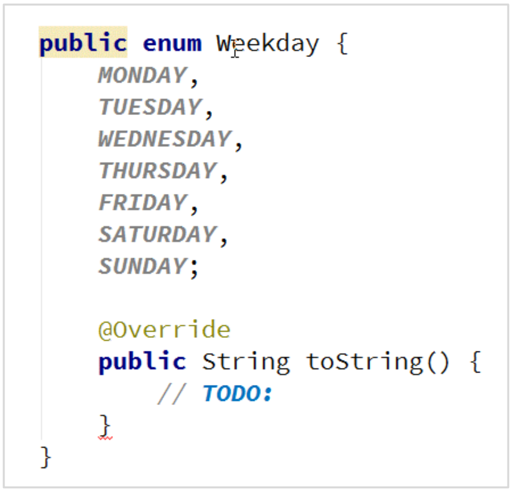
Examples

```java
public static void main(String[] args) {
    WeeklyCalendar wc = new WeeklyCalendar();
    wc.addEntry("Friday", "sleep");
    wc.addEntry("Monday", "sport");
    Iterable<WeeklyEntry> schedule = wc.getWeeklySchedule();
    for (WeeklyEntry weeklyEntry : schedule) {
        System.out.println(weeklyEntry);
    }
}
```

```
"C:\Program Files\Java\jdk1.8.0_91\bin\java" ...
Monday - sport
Friday - sleep
```

Hints

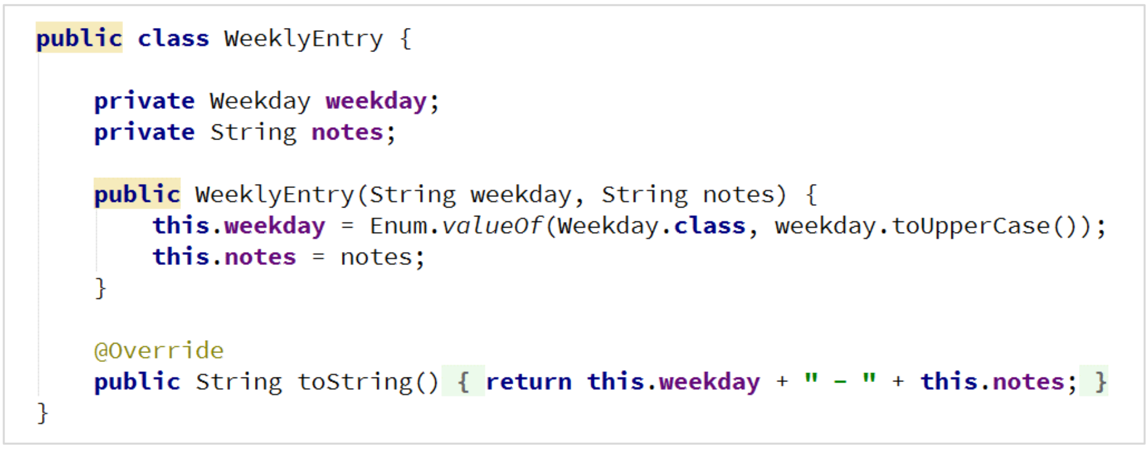
Create **enum** Weekday and override its **toString()** method:



```java
public enum Weekday {
    MONDAY,
    TUESDAY,
    WEDNESDAY,
    THURSDAY,
    FRIDAY,
    SATURDAY,
    SUNDAY;

    @Override
    public String toString() {
        // TODO:
    }
}
```

Create **`WeeklyEntry`** and think about a solution to compare entries:



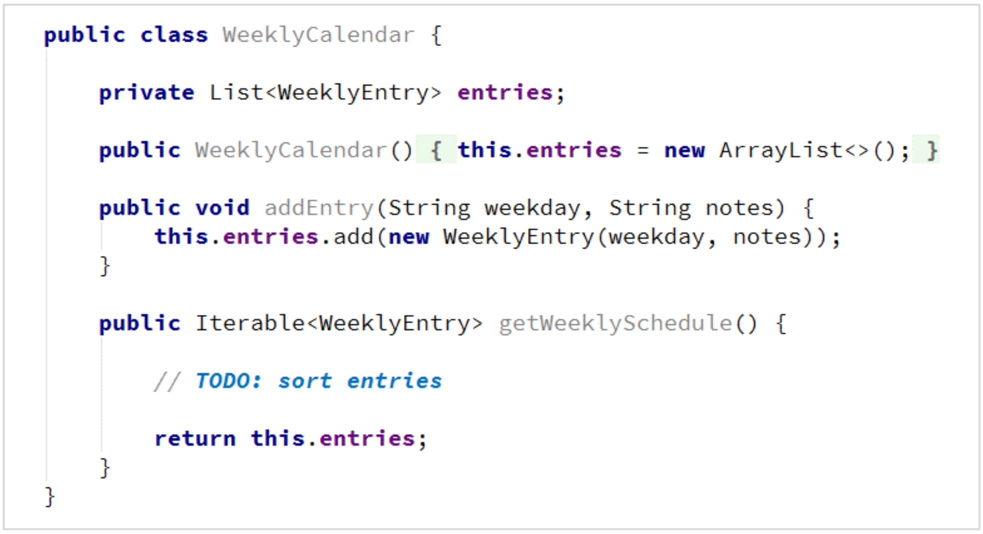
```java
public class WeeklyEntry {

    private Weekday weekday;
    private String notes;

    public WeeklyEntry(String weekday, String notes) {
        this.weekday = Enum.valueOf(Weekday.class, weekday.toUpperCase());
        this.notes = notes;
    }

    @Override
    public String toString() { return this.weekday + " - " + this.notes; }
}
```

Create the **`WeeklyCalendar`**:



```java
public class WeeklyCalendar {

    private List<WeeklyEntry> entries;

    public WeeklyCalendar() { this.entries = new ArrayList<>(); }

    public void addEntry(String weekday, String notes) {
        this.entries.add(new WeeklyEntry(weekday, notes));
    }

    public Iterable<WeeklyEntry> getWeeklySchedule() {

        // TODO: sort entries

        return this.entries;
    }
}
```

2. Warning Levels

Create a classes **`Logger`** and **`Message`**.

Create **enum `Importance`** with constants - Low, Normal, Medium, High.

The Logger should have a method that **receives a message**, but **records** only messages **above or equal to a given importance** level.

Create a method

- **`Iterable<Message> getMessages()`**

Examples

Input	Output

`HIGH`
`NORMAL: All systems running`
`HIGH: Leakage in core room`
`LOW: Food delivery`
`END`
	`HIGH: Leakage in core room`

`LOW`
`NORMAL: All systems running`
`HIGH: Leakage in core room`
`LOW: Food delivery`
`END`
	`NORMAL: All systems running`
	`HIGH: Leakage in core room`
	`LOW: Food delivery`

3. Coffee Machine

Create a class **`CoffeeMachine`**, with methods:

- **`void buyCoffee(String size, String type)`**
- **`void insertCoin(String coin)`**
- **`Iterable<Coffee> coffeesSold()`**

Create **`enum CoffeeType`** – Espresso, Latte, Irish

Create **`enum Coin`** – 1, 2, 5, 10, 20, 50 (constants should be named ONE, TWO, FIVE, etc.)

Create **`enum CoffeeSize`** that has **dosage** and **price** – Small (50 ml, 50 c), Normal (100 ml, 75 c), Double (200 ml, 100 c)

`CoffeeMachine` should **clear all coins after each successful coffee sold**.

Examples

Input	Output

`TEN`
`TWENTY`
`TWENTY`
`Small Espresso`
`END`
	(no output) Machine should have only one "Small Espresso" sold

`TEN`
`TWENTY`
`Small Espresso`
`TWENTY`
`Small Espresso`
`END`
	(no output) Machine should have only one "Small Espresso" sold
	
	Comment: first try - not enough coins

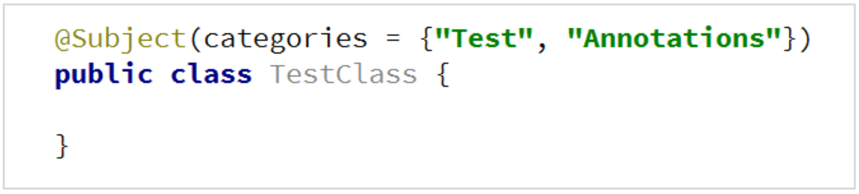
Part II: Annotations

4. Create Annotation

Create annotation **`Subject`** with a **`String[]`** element called **categories**, that:

- Should be available at runtime
- Can be placed only on types

Examples



```
@Subject(categories = {"Test", "Annotations"})
public class TestClass {

}
```

5. Coding Tracker

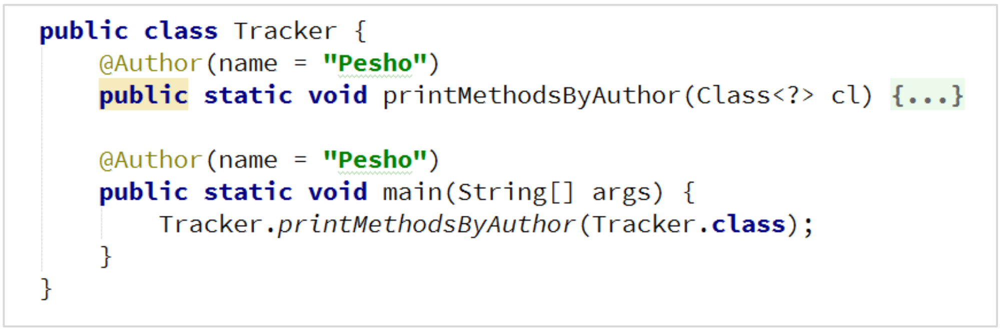
Create annotation **Author** with a **`String`** element called **name**, that:

- Should be available at runtime
- Can be placed only on methods

Create a class **Tracker** with a method:

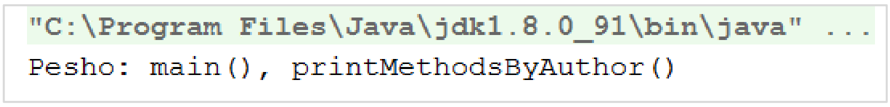
- **`static void printMethodsByAuthor()`**

Examples



```
public class Tracker {
    @Author(name = "Pesho")
    public static void printMethodsByAuthor(Class<?> cl) {...}

    @Author(name = "Pesho")
    public static void main(String[] args) {
        Tracker.printMethodsByAuthor(Tracker.class);
    }
}
```



```
"C:\Program Files\Java\jdk1.8.0_91\bin\java" ...
Pesho: main(), printMethodsByAuthor()
```