02

Muitos objetos e o Flyweight

Imagine que precisamos implementar um software musical. Um software que, dado uma sequência de notas musicais, ele deve tocar a música.

Como estamos trabalhando em uma linguagem orientada a objetos, as notas seriam representadas como classes: `Do`, `Re`, `Mi`, e assim por diante.

A criação de uma música ficaria algo como:

```
List<Nota> musica = Arrays.asList(
  new Do(), new Re(), new Mi(), new Fa(),
  new Fa(), new Fa(), new Sol(), new La() // ...
);
```

Apesar de bonito, o código acima é problemático. Imagine uma música de verdade, com milhares de notas musicais. Nesse caso, estaríamos criando milhares de dós, rés, mis e fás diferentes, consumindo demais a memória.

Mas repare que todos os dós são iguais; os rés, os mis, e assim por diante.

Uma solução para tal seria fazer então com que instanciássemos apenas uma instância de cada nota, e reutilizássemos essa mesma instância.

Vamos criar uma classe, parecida com uma Factory, que instanciará uma nota de cada, e sempre que alguém pedir uma nota, ela devolverá sempre a mesma instância:

```
public class NotasMusicais {

    private static Map<String, Nota> notas =
        new HashMap<String, Nota>();

    static {

        notas.put("do", new Do());
        notas.put("re", new Re());
        notas.put("mi", new Mi());
        notas.put("fa", new Fa());
        notas.put("sol", new Sol());
        notas.put("la", new La());
        notas.put("si", new Si());

    }
    public Nota pega(String nome) {
        return notas.get(nome);
    }
}
```

Uma `Nota`, no código acima é uma simples interface, com implementações para cada nota musical:

```
public interface Nota {
    String simbolo();
}

public class Do implements Nota {

    @Override
    public String simbolo() {
        return "C";
    }

}

// re, mi, fa, sol, la, si...
```

Veja só a classe `NotasMusicais`. Ela é uma fábrica, que tem um mapa, que guarda uma instância de cada nota.

Com isso em mãos, podemos compor uma música assim:

```
public class Programa {

    public static void main(String[] args) {

        NotasMusicais notas = new NotasMusicais();

        List<Nota> doReMiFa = Arrays.asList(
            notas.pega("do"),
            notas.pega("re"),
            notas.pega("mi"),
            notas.pega("fa"),
            notas.pega("fa"),
            notas.pega("fa"),

            notas.pega("do"),
            notas.pega("re"),
            notas.pega("do"),
            notas.pega("re"),
            notas.pega("re"),
            notas.pega("re"),

            notas.pega("do"),
            notas.pega("sol"),
            notas.pega("fa"),
            notas.pega("mi"),
            notas.pega("mi"),
            notas.pega("mi"),

            notas.pega("do"),
            notas.pega("re"),
            notas.pega("mi"),
            notas.pega("fa"),
            notas.pega("fa"),
            notas.pega("fa")
        );
```

```
    }
}
```

Veja então que nosso array contém muitas notas, mas estamos usando pouco a memória, já que reutilizamos a mesma instância das notas musicais. Muito prático!

Para finalizar a implementação, vamos tocar essa música. Faremos uso aqui do framework JFugue (http://www.jfugue.org/jfugue-5.0.9.jar), que nos ajuda a tocar notas musicais.

Veja a classe `Piano` abaixo que faz uso dela:

```
public class Piano {

    public void toca(List<Nota> musica) {
        Player player = new Player();

        StringBuilder musicaEmNotas = new StringBuilder();
        for(Nota nota : musica) {
            musicaEmNotas.append(nota.simbolo() + " ");
        }

        player.play(musicaEmNotas.toString());
    }
}
```

Agora, basta invocarmos o `Piano` no fim do nosso Programa e a musica tocará:

```
    Piano piano = new Piano();
    piano.toca(doReMiFa);
```

Sempre que temos uma quantidade grande de objetos similares a serem instanciados, uma boa solução é fazer cache dessas instâncias e reutilizá-la. Para isso, a implementação é geralmente fazer uso de uma fábrica, que controla as instâncias. Esse padrão é conhecido por **flyweight**.