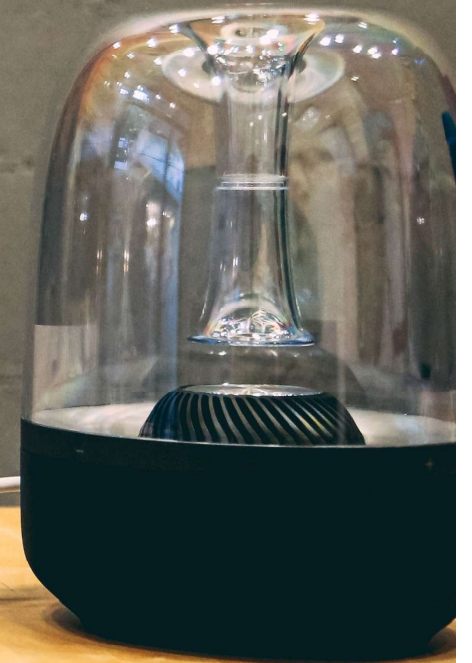


Programação Orientada Pelos Objectos

Imutabilidade final



2

Objectos imutáveis



Objectos imutáveis

3

- Um objecto imutável é um objecto cujo estado interno se mantém constante ao longo de todo o seu ciclo de vida, após a sua criação
 - São relativamente mais simples de entender e usar do que os objectos mutáveis
 - São muito úteis como “peças” na construção de outros objectos mais complexos

A `String` é um objecto imutável

4

- A API da `String` apenas nos oferece métodos read-only. Não inclui métodos que permitem alterar o estado interno do objecto!
 - Mesmo um método como o `replace()` não altera a `String` – devolve outra `String` como resultado

```
String name = "Lady Gaga";  
String newName = name.replace("Gaga", "Gogo");  
System.out.println(name);      // Lady Gaga  
System.out.println(newName);    // Lady Gogo
```

Afinal, o que é que o **final** nos proíbe de fazer a uma variável?

5

- O modificador **final** apenas nos proíbe de modificar o valor da variável
 - A String é um objecto imutável

```
final String name = "Shakira Isabel";  
name = "Carla Marti"; // Compile time error!
```

Afinal, o que é que o **final** nos proíbe de fazer a uma variável?

6

- O modificador **final** não nos impede de **modificar o estado interno do objecto** a que se refere usando a sua API pública!
 - O Array não é um objecto imutável!

```
final Array<String> strings = new ArrayClass<>();  
strings.insertLast("Casio"); //Ok. strings.size() == 1  
strings.insertLast("Rolex"); //Ok. strings.size() == 2  
strings.insertLast("Twingo"); //Ok. strings.size() == 3  
strings.insertLast("Rolls Royce"); // Ok. Got it...
```

Benefícios de um objecto imutável

7

- Como o seu estado não pode ser alterado, podemos partilhar o objecto de forma segura em múltiplas *threads*
 - Terá oportunidade de aprender mais sobre isto noutras unidades curriculares
- Pode usar livremente sem que outros objectos que o referenciem notem a diferença
 - **Os objectos imutáveis são livres de efeitos colaterais**

Objectos imutáveis

8

- Como criar um objecto imutável?
 - Declarar todos os seus membros como **final**
 - Inicializar todos os campos no construtor
 - Não disponibilizar métodos modificadores
 - Mas podem-se disponibilizar métodos de consulta
 - Declarar a classe como **final**, para os seus membros não poderem ser redefinidos
- Hmm... métodos e classes **final**?



Métodos “normais” vs. “finais”

10

- As classes, **concretas** ou **abstractas**, podem conter métodos normais, como os que temos usado, ou “finais”
- Um método diz-se **final** quando **não pode** ser redefinido numa subclasse – existente, ou que venha a ser criada no futuro
 - A tentativa de redefinir um método final origina um erro do compilador

A palavra reservada **final**

11

- Em geral, devemos declarar os métodos invocados pelos construtores como **final**
 - Isso impede que, por polimorfia, esses métodos tenham um comportamento inesperado
- Ocasionalmente, podemos querer definir uma **classe** como **final** para garantir que ela não pode ser redefinida
 - Útil, por exemplo, para garantir que os objectos dessa classe são imutáveis (**como acontece com as Strings**)

A palavra reservada **final**

12

- Recorde o modificador **final**, usado no contexto da definição de constantes
 - Tal como com os métodos, **final** indica que o elemento por ele afectado não pode ser alterado


```
class Chess {  
    private static final int WHITE = 0; // Peças brancas  
    private static final int BLACK = 1; // Peças pretas  
    //...  
    public final int getFirstPlayer() {  
        return Chess.WHITE;  
    }  
    //...  
}
```

- Isto aplica-se a métodos, membros de dados, parâmetros, e mesmo a classes

Implementação de um método `final`

13

```
public class Chess {  
    public static final int WHITE = 0; // Peças brancas  
    public static final int BLACK = 1; // Peças pretas  
    //...  
    /**  
     * Retorna o primeiro jogador a mover peças, num jogo de xadrez.  
     * De acordo com as regras, começam SEMPRE as brancas.  
     * @return Começam as brancas.  
     */  
    public final int getFirstPlayer() {  
        return Chess.WHITE;  
    }  
    //...  
}
```



- Não queremos deixar que um dia mais tarde alguém viole as regras do xadrez, redefinindo este método! Assim, **temos a certeza de que começam sempre as brancas, nesta classe ou em qualquer subclasse dela que venha a ser definida no futuro.**

Construir uma classe de um objecto imutável

14

```
class Money {
```

```
}
```

Declare as variáveis de instância como **final**

15

```
class Money {  
    private final double amount;  
    private final Currency currency;  
  
}
```

Inicialize **todas** as variáveis de instância no construtor

16

```
class Money {  
    private final double amount;  
    private final Currency currency;  
  
    public Money(double amount, Currency currency) {  
        this.amount = amount;  
        this.currency = currency;  
    }  
  
}
```


Ofereça apenas métodos de consulta na API

17

```
class Money {  
    private final double amount;  
    private final Currency currency;  
  
    public Money(double amount, Currency currency) {  
        this.amount = amount;  
        this.currency = currency;  
    }  
    public Currency getCurrency() {  
        return currency;  
    }  
    public double getAmount() {  
        return amount;  
    }  
}
```

IMUTÁVEL!!!

18

```
class Money {  
    private final double amount;  
    private final Currency currency;  
  
    public Money(double amount, Currency currency) {  
        this.amount = amount;  
        this.currency = currency;  
    }  
    public Currency getCurrency() {  
        return currency;  
    }  
    public double getAmount() {  
        return amount;  
    }  
}
```

Se todos os objectos **Money** são imutáveis,
Se todos os objectos **SpecialMoney** são **Money**,
Se afinal os **SpecialMoney** são mutáveis...

19

```
class Money { // A minha classe de objectos imutáveis
    ...
}
```

```
class SpecialMoney extends Money { // Ooops
    // Uhuuu, aqui vou mudar o estado como quiser!
}
```

Repare que a parte **Money** the **SpecialMoney** continua imutável. Mas isto pode ser um problema, se estava **mesmo** a contar que **Money** fosse imutável. O que fazer?

Declare a classe e métodos como **final**

20

```
final class Money {  
    private final double amount;  
    private final Currency currency;  
  
    public Money(double amount, Currency currency) {  
        this.amount = amount;  
        this.currency = currency;  
    }  
    public final Currency getCurrency() {  
        return currency;  
    }  
    public final double getAmount() {  
        return amount;  
    }  
}
```

A palavra reservada **final**

21

- O modificador **final**, exprime sempre limitações à modificação, mas aquilo a que se aplica varia com o contexto em que é usado
 - Nas constantes, **final** refere-se ao **valor** da “variável”
 - Nos métodos, **final** refere-se à possibilidade de redefinir o método
 - Nas classes, **final** refere-se à herança, i.e., à possibilidade de criar uma subclasse
- No caso geral, o Java **não** fornece um mecanismo que garanta que um **objecto** é constante
 - Os programadores é que têm de garantir que uma classe é **imutável**, ao não fornecer operações que alteram o estado