02

Quando o cast é necessário

Às vezes, temos referências de um tipo mas sabemos que lá há um objeto de outro tipo, um mais específico:

```
public class Teste{
    public static void main(String...args){
        Object[] objetos = new Object[100];

        String s = "certificacao";
        objetos[0] = s;

        String recuperada = objetos[0];
    }
}
```

COPIAR CÓDIGO

O código acima não compila:

```
Teste.java:3: incompatible types
found   : java.lang.Object
required: java.lang.String
                String recuperada = objetos[0];
                                    ^
1 error
```

COPIAR CÓDIGO

Temos um array de referências para `Object`. Nem todo `Object` é uma `String`, então o compilador não vai deixar você fazer essa conversão. Lembre-se que, em geral, o compilador não conhece os *valores* das variáveis, apenas seus tipos.

Vamos precisar *moldar* a referência para que o código compile:

```
String recuperada = (String) objetos[0];
```

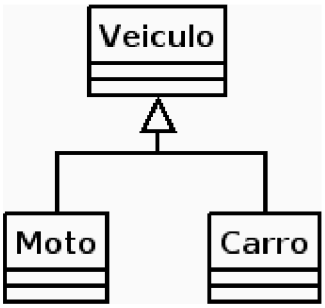
COPIAR CÓDIGO

A partir de agora, esse código compila. Mas será que roda? Durante a execução, o **casting** vai ver se aquele objeto é mesmo compatível com o tipo `String` (no nosso caso é). Se não fosse, ele lançaria uma `ClassCastException` (exceção *unchecked*).

Considere as classes:

```
class Veiculo {}
class Moto extends Veiculo {}
class Carro extends Veiculo {}
```

COPIAR CÓDIGO



E o código:

```
Veiculo v = new Carro();
Moto m = v;
```

COPIAR CÓDIGO

Na primeira linha, usamos polimorfismo para chamar um `Carro` de `Veiculo` (**é um**). Na segunda linha, o que o compilador sabe é que `v` é do tipo `Veiculo`. E *nem todo* `Veiculo` *é uma* `Moto`. Por isso, essa linha não compila.

Mas existem *alguns* `Veiculo` que são `Moto`. Então, o compilador deixa que façamos o casting:

```
Veiculo v = new Carro();
Moto m = (Moto) v;
```

COPIAR CÓDIGO

Com isso, o código compila, mas repare que, em tempo de execução, `v` aponta para um objeto do tipo `Carro`. Quando o código for executado, haverá um erro de execução: `ClassCastException`. `Carro` **não** *é uma* `Moto`.

Cuidado que, se o *casting* for totalmente impossível, o compilador já acusará erro:

```
Carro c = new Carro();
Moto m = (Moto) c;
```

COPIAR CÓDIGO

Um `Carro` **nunca** poderá ser uma `Moto`. Então, nem com casting isso compila.

É importante lembrar que quando não precisamos de casting, ele é opcional, portanto todas as linhas a seguir funcionam com ou sem casting:

```
String guilherme = "guilherme";
String nome = guilherme;
String nome2 = (String) guilherme;
Object nome3 = guilherme;
Object nome4 = (String) guilherme;
Object nome5 = (Object) guilherme;
```

COPIAR CÓDIGO

Regra geral!

Se você está subindo na hierarquia de classes, a autopromoção vai fazer tudo sozinho; e se você estiver descendo, vai precisar de casting. Se não houver um caminho possível, não compila nem com casting.

Na prova, faça sempre os diagramas de hierarquia de tipos que fica extremamente fácil resolver esses castings.

Casting com interfaces

Dado o código a seguir:

```
Carro c = new Carro();
Moto m = (Moto) c;
```

COPIAR CÓDIGO

Quando dizemos que ele não compila, é porque um `Carro` nunca pode ser uma `Moto`. Mas como o compilador sabe que isso é impossível mesmo? Existe alguma chance de algum objeto de qualquer tipo ser, ao mesmo tempo, `Carro` e `Moto`?

```
class X extends Moto, Carro { // não compila!

}
```

COPIAR CÓDIGO

A única maneira de isso acontecer seria se Java suportasse herança múltipla; aí escreveríamos uma classe que herdasse de `Carro` e `Moto` ao mesmo tempo. Como Java não tem herança múltipla, isso realmente é impossível de acontecer.

Mas e quando fazemos casting com interfaces envolvidas? Apesar de não existir herança múltipla, podemos implementar múltiplas interfaces! Fazer casting para interfaces sempre é possível e vai compilar (há apenas uma exceção a essa regra).

Pegue uma interface *qualquer*, por exemplo `Runnable`. O código a seguir compila:

```
Carro c = new Carro();
Runnable r = (Runnable) c;
```

COPIAR CÓDIGO

Um `Carro` *pode ser* um `Runnable`? Sabemos que a classe `Carro` propriamente não implementa essa interface. Mas existe a possibilidade de existir algum objeto em Java que seja, ao mesmo tempo, `Carro` e `Runnable`?

A resposta é sim! E se tivéssemos uma classe `CarroRodavel`?

```
class CarroRodavel extends Carro implements Runnable { ... }
```

COPIAR CÓDIGO

O compilador não sabe o valor da variável `c` nesse exemplo. Ele não sabe que na verdade é uma instância de `Carro` e não de `CarroRodavel`. Ele sabe apenas que é do tipo `Carro` e, pela simples possibilidade de existir um objeto que seja `Carro` e `Runnable`, ele deixa o código compilar.

Mas repare que a classe `CarroRodavel` não existe no diagrama original. Mesmo assim, o código compila! Apenas com a *possibilidade* de existir uma classe dessa, o compilador já aceita aquele casting, mesmo que uma classe dessas não exista na prática.

Claro que o objeto é do tipo `Carro`, que não implementa `Runnable` e, em tempo de execução, vai ocorrer uma `ClassCastingException`.

E final

Dizemos que o código anterior compila porque há a possibilidade de uma classe como `CarroRodavel` existir algum dia. Mas será que sempre há essa possibilidade mesmo?

Se a classe `Carro` for `final`, é impossível existir uma classe filha dela. E como a própria `Carro` não implementa `Runnable`, nesse caso, será impossível fazer o casting para `Runnable` (o próprio compilador já acusa erro).

Dica

Muitos exercícios são sobre casting de referência. Uma dica é seguir o que é possível, impossível e óbvio.

Se é óbvio que o casting funciona, isso é, se a conversão é sempre verdade, a autopromoção faz sozinha.

Se o casting é possível, mas nem sempre é verdade, o casting compila, mas pode lançar erro em tempo de execução.

Se o casting é impossível, isto é, ele nunca pode dar certo, o código não vai compilar nem com casting.

Em alguns livros, você encontra tabelas complicadas e grandes que o "ajudam" a decidir se o casting compila e roda, mas é muito mais fácil seguir pela lógica.

instanceof

O operador `instanceof` (`a instanceof Classe`) devolve `true` caso a referência `a` aponte para um objeto compatível (*assignable*, atribuível) ao tipo `Classe` .

```
Object c = new Carro();
boolean b1 = c instanceof Carro; // true
boolean b2 = c instanceof Moto; // false
```

COPIAR CÓDIGO

O `instanceof` não compila se a referência em questão for obviamente incompatível, por exemplo:

```
        String s = "a";
        boolean b  = s instanceof java.util.List; // não
compila
```

COPIAR CÓDIGO

Detalhe

`instanceof` é um operador que deve ser usado com extremo cuidado no dia a dia. Em muitos casos, ele indica a fraca modelagem de um sistema, com blocos que parecem "*switchs*" e poderiam ser trocados por polimorfismo.