02

Use classes abstradas e interface.

Classes e métodos podem ser abstratos.

Uma classe abstrata pode não ter nenhum método abstrato:

```
// compila
abstract class SemMetodos {
}
```

COPIAR CÓDIGO

Se uma classe tem um método que é abstrato, ela deve ser declarada como abstrata, ou não compilará.

```
// não compila, se tem método abstrato, tem que implementar
class ComMetodoAbstrato {
    public abstract void executa();
}
```

COPIAR CÓDIGO

Uma classe abstrata não pode ser instanciada diretamente:

```
abstract class X{

}
```

```
public class Teste{
    public static void main(String[] args) {
        X x = new X();
    }
}
```

COPIAR CÓDIGO

A classe `Teste` não compila! Classes abstratas não podem ser instanciadas diretamente:

```
Teste.java:7: X is abstract; cannot be instantiated
        X x = new X();
                  ^
1 error
```

COPIAR CÓDIGO

Um método abstrato é um método sem corpo, somente com a definição. Uma classe que tem um ou mais métodos abstratos precisa ser declarada como abstrata.

Não importa se o método foi escrito diretamente ou foi herdado:

```
abstract class Veiculo {
    public abstract void liga();
}

// compila pois implementou
class Moto extends Veiculo {
    public void liga() {

    }
```

```
}

// compila pois a classe é abstrata, com método herdado
// abstrato ainda
abstract class QuatroRodas extends Veiculo {
}

// não compila pois a classe não é abstrata,
// com método herdado abstrato ainda
class SemRodas extends Veiculo {
}
```

COPIAR CÓDIGO

Um método abstrato tem de ser reescrito ou herdado pelas suas filhas concretas.

Agora veja o exemplo a seguir:

```
abstract class Veiculo {
    public abstract void liga();
}

class Moto extends Veiculo {
    public void liga() {

    }
}
```

COPIAR CÓDIGO

O método `liga` foi implementado na classe filha, então ela pode ser concreta. Basta pensar que métodos abstratos herdados são *responsabilidades herdadas*: você não poderá ser um objeto concreto enquanto tiver responsabilidades a serem tratadas.

Quando herdamos de uma classe abstrata que possui um método abstrato, temos que escolher: ou implementamos o método, ou somos abstratos também e *passamos adiante* a responsabilidade. Note que a classe pode implementar o método e mesmo assim também ser abstrata.

```
abstract class Veiculo {
    public abstract void liga();
}

abstract class Moto extends Veiculo {

}

class MotoEspecial extends Moto {
    public void liga() {

    }
}

// compila: decidi implementar mas mesmo assim
// manter a classe abstrata
abstract class QuatroRodar extends Veiculo {
    public void liga() {
    }
}
```

COPIAR CÓDIGO

Código de uma classe abstrata pode acessar o código da classe concreta, uma vez que ele só será executado quando o objeto for criado:

```
abstract class X{
```

```
    void x() {
        System.out.println(y());
    }
    abstract String y();
}
class Y extends X {
    String y() {
        return "codigo";
    }
}
public class Teste {
    public static void main(String[] args) {
        new Y().x(); // imprime código
    }
}
```

COPIAR CÓDIGO

Interfaces

Uma interface declara métodos que deverão ser implementados pelas classes concretas que queiram ser consideradas como tal. Por padrão, são todos métodos públicos e abstratos.

```
interface Veiculo {
    void ligar();
    // public abstract! Você pode escrever, mas é por padrão
    // isso.
    public abstract int pegaMarcha();
}
```

COPIAR CÓDIGO

Quando você implementa a interface em uma classe concreta, é preciso implementar todos os métodos. Similarmente, ao herdar uma classe abstrata, a classe concreta deve implementar todos os métodos que não foram implementados ainda:

```
// compila, todos os métodos implementados
class Carro implements Veiculo {
    public void ligar() {
    }
    public int pegaMarcha() {
        return 0;
    }
}
// não compila, onde está o pegaMarcha?
class Moto implements Veiculo {
    public void ligar() {
    }
}
// não compila, o método pegaMarcha definiu escopo default,
// quando deveria definir public
class Triciclo implements Veiculo {
    public void ligar() {
    }
    int pegaMarcha() {
        return 0;
    }
}
```

COPIAR CÓDIGO

Valem as mesmas regras de quando você herda de uma classe abstrata: ou você tem todos os métodos reescritos, e aí pode declará-la como concreta, ou então você precisa declará-la como abstrata.

```
// compila, pois a classe é abstrata
abstract class Moto implements Veiculo {
    public void ligar() {
    }
}
```

COPIAR CÓDIGO

Uma classe pode implementar diversas interfaces:

```
abstract class MinhaClasse implements Serializable, Runnable {
}
```

COPIAR CÓDIGO

Justamente por isso, a prova vê como um bom uso de interfaces quando você quer herdar de dois lugares mas a herança de classes não permite. Para a prova, essa razão é suficiente, mas na prática existe uma diferença grande entre composição (herança de interfaces não envolve herdar comportamento e variáveis membro) e herdar comportamento e variáveis membro de uma classe mãe. Como a implementação de uma interface nos obriga a escrever todos os métodos, estamos compondo nossa classe de diversas interfaces.

Lembre-se que uma interface pode herdar de outra, inclusive de diversas interfaces:

```
interface A extends Runnable {}
interface B extends Serializable {}
interface C extends Runnable, Serializable {}
```

COPIAR CÓDIGO

Note que uma interface nunca implementa outra interface:

```
interface A implements Runnable {} // não compila
```

COPIAR CÓDIGO

Você pode declarar variáveis em uma interface, todas elas serão `public final static` , isto é, constantes.

```
interface X {
    int i = 5;
    // você até pode escrever public static final, mas é sempre
    // assim
}
```

COPIAR CÓDIGO

Uma interface, por sua vez, pode estender outra interface, herdando suas responsabilidades e constantes. Uma interface pode estender mais de uma interface!

```
interface X extends Runnable, Comparable { }
```

COPIAR CÓDIGO