Métodos Default (padrão) em Interfaces

Instituto Federal do Espirito Santo – Ifes
Campus Colatina – Programação Orientada a Objetos I
Professor: Giovany Frossard Teixeira

Método default

- Um método default é um método de uma interface que possui corpo, ou seja, possui código, não sendo portanto só uma assinatura.
- O código do método default só é utilizado quando a classe concreta que poderia implementar o método não o faz.
- É importante notar que, ao colocar um método como default não há mais a obrigatoriedade de implementação por parte da classe concreta. Isso é interessante porque podemos não ter a necessidade de implementar tal comportamento, por outro lado pode ser ruim pois caso haja a necessidade e o programador tenha esquecido de implementar, o compilador não irá avisar nada a ele.

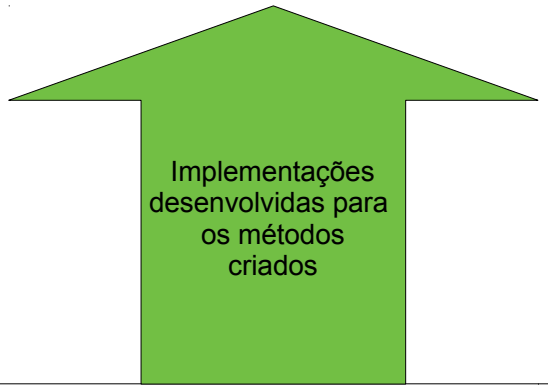
Exemplo

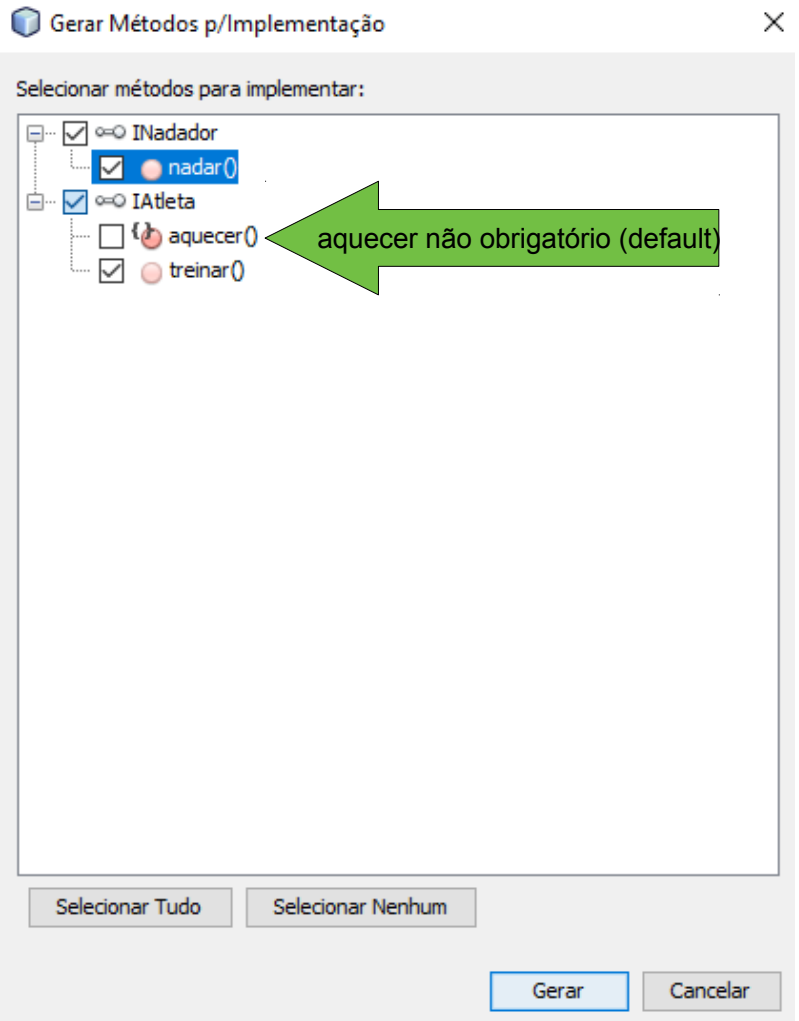
```
public interface IAtleta {

    public default void aquecer(){

        System.out.println("Estou
        aquecendo");

    }


    public void treinar();

}
```

```
public interface INadador
extends IAtleta {

    public void nadar();

}
```

Exemplo

```
public class Nadador implements INadador {

    @Override

    public void nadar() {

        System.out.println("Estou nadando");

    }

    @Override

    public void treinar() {

        System.out.println("Estou treinando como um nadador");

    }

}
```





Mudando o exemplo – herança múltipla

```
public interface IAtleta {

    public default void aquecer(){

        System.out.println("Estou aquecendo");

    }

    public void treinar();

}
```

```
public interface INadador extends IAtleta {

    public default void aquecer(){

        System.out.println("Estou aquecendo como um
nadador");

    }

    public void nadar();

}
```

```
public interface ICiclista extends IAtleta {

    public default void aquecer(){

        System.out.println("Estou aquecendo como um
ciclista");

    }

    public void pedalar();

}
```

- Agora suponhamos a criação da classe BiAtleta que seria o Atleta que faz ciclismo e natação.
- Em outras palavras BiAtleta implementaria tanto INadador quanto ICiclista.
- Nesse caso esse BiAtleta deveria aquecer como um Atleta comum (IAtleta), como um nadador (INadador) ou como um ciclista (INadador) ?

Mudando o exemplo herança múltipla

```
/**
class BiAtleta inherits unrelated defaults for aquecer() from types INadador and ICiclista
----
(Alt-Enter mostra dicas)
public class BiAtleta implements INadador, ICiclista{

    @Override
    public void nadar() {

    }

    @Override
    public void treinar() {
    }

    @Override
    public void pedalar() {
    }

}
```

- Java não tem suporte a herança múltipla de classes mas permite que uma classe implemente diversas interfaces. O problema é que se essas interfaces tiverem o mesmo método, há conflito (tanto INadador quanto ICiclista possuem o método aquecer()).
- Nesse contexto, somos obrigados a implementar o método aquecer na classe BiAtleta podendo inclusive chamar o código de uma das interfaces. Ex:

```
@Override

public void aquecer(){    INadador.super.aquecer();    }
```

- É importate notar que Java nos obriga a implementar o método aquecer porque ele não tem como saber o comportamento que queremos (INadador ou ICiclista).
- Entretanto se retirarmos o código de aquecer de uma das duas interfaces (INadador ou ICiclista) Java assume o método da interface mais próxima. No próximo slide apresentaremos essa situação.

E agora ?

```java
public class BiAtleta implements INadador, ICiclista{
    /*
    @Override
    public void aquecer(){
        INadador.super.aquecer();
    }
    */
    @Override
    public void nadar() {}

    @Override
    public void treinar() {}

    @Override
    public void pedalar() {}
}
```

```java
public interface ICiclista extends IAtleta {
  /*  public default void aquecer(){
        System.out.println("Estou aquecendo como um ciclista");
    }
*/
    public void pedalar();
}
```

```java
public interface INadador extends IAtleta {
    @Override
    public default void aquecer(){
        System.out.println("Estou aquecendo como um nadador");
    }
    public void nadar();
}
```

```java
public interface IAtleta {
    public default void aquecer(){
        System.out.println("Estou aquecendo");
    }
    public void treinar();
}
```

- Como um BiAtleta deve aquecer? Como um IAtleta (notar que IClilista herda de IAtleta)? Como INadador (que redefiniu o comportamento de IAtleta)?
- Nesse caso o Java assume a classe de implementação direta (não obtida por herança, ou seja, INadador). No próximo slide veremos o resultado.

Resultado

```
public class UsandoMetodoDefault {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        BiAtleta atleta = new BiAtleta();
        atleta.aquecer();
    }

}
```

usandometododefault.UsandoMetodoDefault > main >

a - UsandoMetodoDefault (run) ×

```
run:
Estou aquecendo como um nadador
CONSTRUÍDO COM SUCESSO (tempo total: 0 segundos)
```

Dúvidas?