PROGRAMAÇÃO ORIENTADA A OBJETOS

Interface

Prof. Marcello Thiry <marcello.thiry@gmail.com>

Revendo o conceito de Interface

- Nós já vimos um conceito de Interface
  - A **interface** define o conjunto de operações visíveis (públicas) de uma classe
    - Define quais operações podem ser **invocadas** nos objetos de uma determinada classe
    - Este conceito também é chamado de “**protocolo**”

Revendo o conceito de Interface

- Até agora, a interface estava definida na própria classe
- Porém, passaremos a considerar **Interface** como um mecanismo a parte
  - Uma Interface é uma especificação de comportamento (**ou contrato**) que implementadores concordam em implementar

Revendo o conceito de Interface

- Uma Interface é similar a uma classe abstrata, onde todas as operações são também abstratas:
  - Não pode ser instanciada, pois não possui implementação própria
  - Funciona como um contrato
    - A classe que implementa a Interface deve garantir que todas as operações da Interface tenham um método (o contrato deve ser respeitado)

Interface x Classe Abstrata

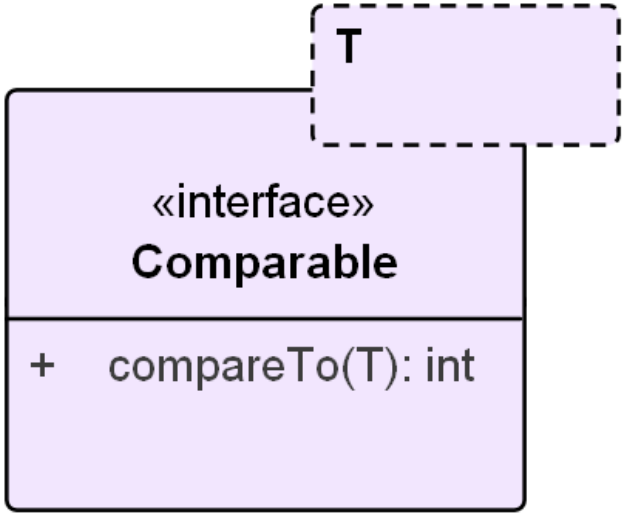
- **Interface**
  - Quando queremos que uma determinada classe seja capaz de realizar alguma operação
    - Independentemente de sua hierarquia de generalização
    - A API Java usa, muitas vezes, o sufixo –able (apto a)
    - Serializable, Runnable, Cloneable, Comparable, ...

Interface x Classe Abstrata

- Há diferença semântica entre “**ser um**” (generalização) e “**ser capaz de**” (interface)

- Uma **interface** é uma especificação de comportamento (ou contrato) que programadores concordam em implementar

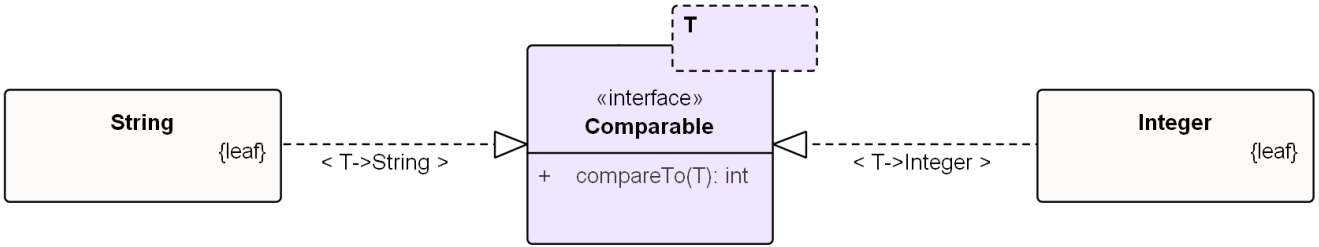
Exemplo de Interface




```
public interface Comparable<T> {

    public int compareTo(T o);
}
```

O método não é especificado aqui, mas delegado para as classes que implementarem a interface

Como implementar




```
public final class Integer extends Number implements Comparable<Integer> {
```

Integer "herda" Number e "implementa" Comparable

Contrato aceito

marcello.thiry@gmail.com

```java
/**
 * Compares two {@code Integer} objects numerically.
 *
 * @param   anotherInteger   the {@code Integer} to be compared.
 * @return  the value {@code 0} if this {@code Integer} is
 *          equal to the argument {@code Integer}; a value less than
 *          {@code 0} if this {@code Integer} is numerically less
 *          than the argument {@code Integer}; and a value greater
 *          than {@code 0} if this {@code Integer} is numerically
 *           greater than the argument {@code Integer} (signed
 *           comparison).
 * @since   1.2
 */
public int compareTo(Integer anotherInteger) {
    return compare(this.value, anotherInteger.value);
}
```

Método implementado na classe Integer

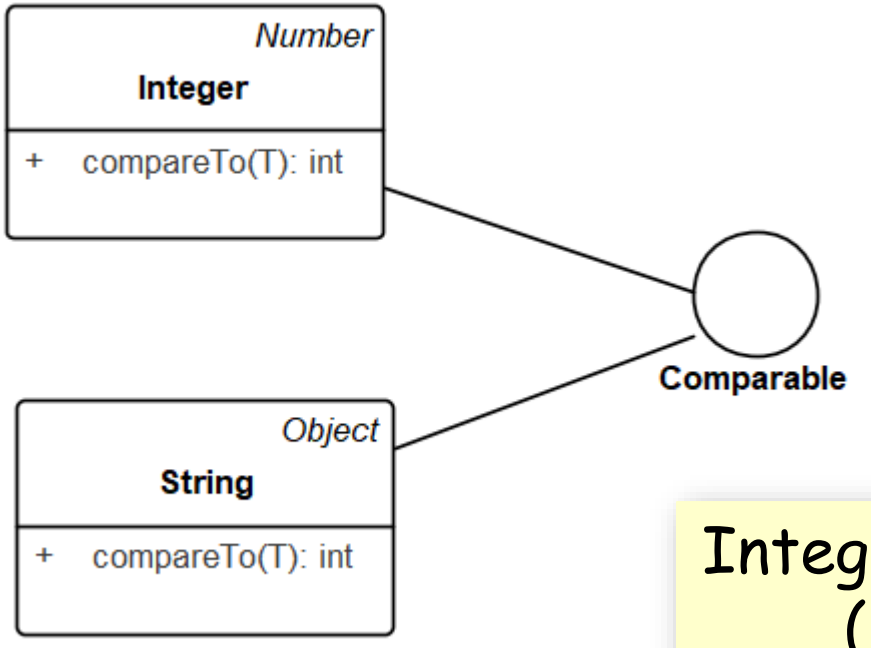
```java
/**
 * Compares two {@code int} values numerically.
 * The value returned is identical to what would be returned by:
 * <pre>
 *    Integer.valueOf(x).compareTo(Integer.valueOf(y))
 * </pre>
 *
 * @param  x the first {@code int} to compare
 * @param  y the second {@code int} to compare
 * @return the value {@code 0} if {@code x == y};
 *         a value less than {@code 0} if {@code x < y}; and
 *         a value greater than {@code 0} if {@code x > y}
 * @since 1.7
 */
public static int compare(int x, int y) {
    return (x < y) ? -1 : ((x == y) ? 0 : 1);
}
```

```java
public class ComparableClass implements Comparable<ComparableClass> {

    @Override
    public int compareTo(ComparableClass o) {
        throw new UnsupportedOperationException("Not supported yet.");
    }

}
```

Não seria necessário, pois não há herança

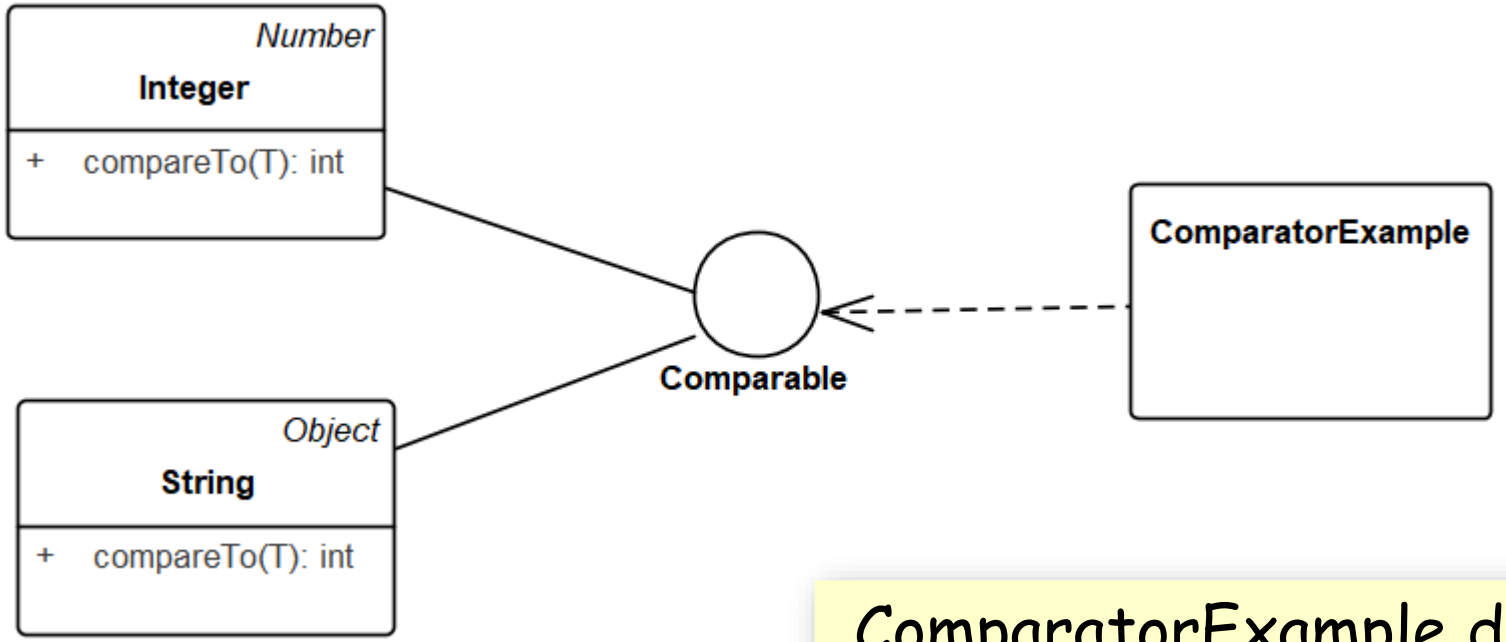
Mas, a partir do Java 6, @Override passou a ser utilizado para indicar também implementação

Notação Círculo (Lollipop)



Integer e String realizam (implementam) a interface Comparable

Modelando o uso



ComparatorExample depende (usa) da interface Comparable

Considerações sobre Interfaces

- Não têm construtores, uma vez que não podem ser instanciadas
- Uma mesma interface pode estender uma ou mais interfaces
- Uma mesma classe pode realizar (implementar) várias interfaces

Considerações sobre Interfaces

- Classes que realizam (implementam) uma mesma interface não precisam compartilhar uma estrutura comum
- Uma interface especifica, usualmente, uma parte limitada do comportamento de uma classe
- A realização (implementação) de uma ou mais classes não exclui a possibilidade de generalização (herança)

Considerações sobre Interfaces

- Reduz a dependência (acoplamento) entre classes, aumentando sua reusabilidade

- Oferece uma alternativa simplificada para a implementação de herança múltipla

Interfaces a partir do Java 8


```
package br.univali.kob.poo1.metodosdefault;

public interface InterfaceA {

    default void metodoDefault(){
        System.out.println("Método default da Interface A");
    }

}
```

Implementando a Interface...


```
package br.univali.kob.poo1.metodosdefault;

public class ExemploImplementacao implements InterfaceA {

}
```

Note que não implementamos o método ()

Como usar...


```
ExemploImplementacao ex = new ExemploImplementacao();
ex.metodoDefault();
```

Herança múltipla?


```
package br.univali.kob.poo1.metodosdefault;

public interface InterfaceB {

    default void metodoDefaultB(){
        System.out.println("Método default da Interface B");
    }

}
```

Herança múltipla?


```
package br.univali.kob.poo1.metodosdefault;

public class HerancaMultipla implements InterfaceA, InterfaceB {

}
```

```
HerancaMultipla hm = new HerancaMultipla();
hm.metodoDefault();
hm.metodoDefaultB();
```

Limitações

- Não resolve o problema do diamante
  - A solução deve ser implementada manualmente

```
public class HerancaMultiplaDiamante implements InterfaceA, InterfaceC {
    @Override
    public void metodoDefault(){
        InterfaceC.super.metodoDefault();
    }
}
```

- Herança somente de métodos
  - Interfaces não possuem atributos

PROGRAMAÇÃO ORIENTADA A OBJETOS

Interface

Prof. Marcello Thiry <marcello.thiry@gmail.com>