Visão Geral de *Generics* e *Collections*

Prof. Jeferson Souza, MSc. (thejefecomp)

jeferson.souza@udesc.br

UDESC
UNIVERSIDADE
DO ESTADO DE
SANTA CATARINA

JOINVILLE
CENTRO DE CIÊNCIAS
TECNOLÓGICAS

Introdução aos Generics e as Collections

Olha lá professor, o que são os famosos Generics e Collections?

Ao que tudo indica, até este exato momento, o conhecimento a respeito dos *Generics* e das *Collections* é muito incipiente, a considerar a probabilidade de não se ter tido consciência de suas utilizações. Os famosos *Generics* e *Collections* são fundamentais na Linguagem Java, a constituirem formas de representação de tipos de maneira generalizada (i.e. *Generics*), e fornecimento de diferentes estruturas de dados e algoritmos associados às referidas estruturas (i.e. *Collections*).

Introdução aos Generics e às Collections

Prazer, Generics, somos os shapeshifters da Linguagem Java

Sua definição é generalizada, a compor uma característica importante da Linguagem Java na especificação de modelos e escrita de código-fonte independente do tipo de dado utilizado. Os *Generics* foram introduzidos na linguagem Java a partir da versão 1.5 (Java 5 para os mais íntimos), e o seu principal objetivo é permitir a referida generalização de uma forma distinta da alcançada por meio de uma hierarquia de Classes ou *Interfaces* tradicional, a ser o compilador responsável pela automatização.

Introdução aos Generics e às Collections

Prazer, Collections, as "caixinhas" (i.e. tupperwares) da Linguagem Java

As coleções especificam formas distintas de armazenar e manipular dados, a representar implementações de algoritmos e estruturas de dados disponibilizadas pela Linguagem Java para o desenvolvimento de programas.

Generics

Como funcionam os Generics?

Ao entrar-se nos detalhes que compõe o domínio dos tipos genéricos da Linguagem Java, fica-se frente a frente com um conceito aplicado no domínio das linguagens de programação chamado de **Type Erasure** [GoslingEtAl, 2021]. De forma simplificada, **Type Erasure** pode ser definido pela operação de substituição de tipos necessária para o bom andamento do programa, a representar a troca de um tipo por outro de acordo com o contexto e domínio associado à referida troca. No caso dos *Generics*, o uso de **Type Erasure** permite a troca de um tipo de dados genérico por um outro tipo de dados concreto (que pode ser inclusive o *Object*) no momento da compilação do programa.

Generics

Mas se os tipos são substituídos, não é mais fácil usar diretamente o tipo concreto?

No momento que tem-se a necessidade de especificar um dado comportamento que pode ser aplicado a um conjunto inimaginável de elementos, o uso dos *Generics*, alcançado pela automatização fornecida pelo compilador da Linguagem Java, torna-se obrigatório. As *Collections* são o principal exemplo de uso de *Generics* na Linguagem Java. Todas as estruturas de dados disponíveis podem ser parametrizadas com qualquer tipo definido na Linguagem, ou diretamente pelo programa desenvolvido.

Generics - Convenção de Nomes

Como definir o nome de um tipo de dado genérico?

Teoricamente, um tipo de dado genérico da Linguagem Java pode receber qualquer nome que seja do agrado do projetista do modelo e/ou do desenvolvedor do código-fonte, desde que o nome não viole nenhuma restrição imposta pela linguagem. Entretanto, existem algumas convenções de nomes seguidas pela Linguagem Java que tornam mais simples a identificação e utilização dos *Generics*. Vamos a elas!

Generics - Convenção de Nomes

Convenção de nomes para tipos genéricos [Boyarsky&Selikoff, 2016]

- **E** para elemento;
- **K** para uma chave em um mapa;
- **V** para um valor em um mapa;
- **N** para um número;
- **T** para um tipo de dado genérico;
- **S,U,V**, e assim por diante, para múltiplos tipos de dados genéricos;

Generics

Para generalizar basta parametrizar...

Para definir qualquer elemento que use um tipo de dado genérico, basta parametrizar o referido elemento. *Interfaces*, Classes, e Métodos podem ser parametrizados para que permitam o uso dos *Generics*. Essa parametrização é realizada diretamente no código-fonte, por meio do uso do operador *diamond* na definição da *Interface*, Classe, ou Método onde o tipo de dado genérico vai ser especificado pela primeira vez.

Generics - Exemplo de Definição de Interface Genérica

Definição de Interface Genérica

```
public interface Controle<T>{
    boolean ligar(T equipamento);
    boolean desligar(T equipamento);
    ...
}
```

Tá, então o T é o tipo genérico?

Exatamente! O *T* é o tipo genérico utilizado na especificação da *Interface Controle*. Como o tipo *T* foi definido diretamente na especificação da *Interface* Controle, pode-se utilizá-lo livremente dentro do escopo da referida *Interface*.

Generics - Exemplo de Concretização de Interface Genérica

Concretizar Interface Genérica com tipo Concreto

```
public class ControleImpl implements Controle<ArCondicionado>{
  public boolean ligar(ArCondicionado equipamento){
    ...
  }
  public boolean desligar(ArCondicionado equipamento){
  ...
  }
}
```

A especificação do tipo concreto só precisa ser fornecida na implementação da *Interface*

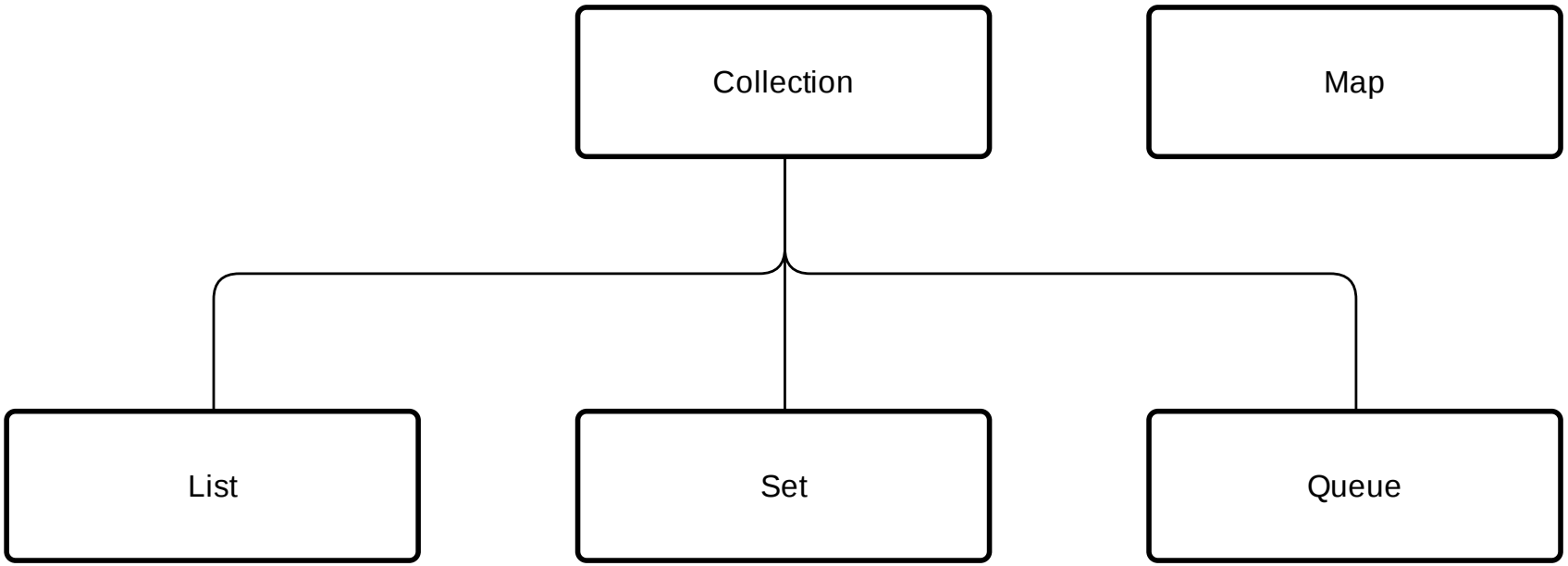
Generics - Exemplo de Concretização de Interface Genérica

Concretizar Interface Genérica com tipo Genérico

```
public class ControleImpl<T> implements Controle<T>{
  public boolean ligar(T equipamento){
    ...
  }
  public boolean desligar(T equipamento){
  ...
  }
}
```

A definição do tipo genérico precisa ser estendida para a especificação da classe *ControleImpl*.

Generics - As Interfaces Genéricas das Collections



As Interfaces Genéricas da Linguagem Java presentes nas *Collections* [Boyarsky&Selikoff, 2016]

Generics - Exemplo de Extensão de Interface Genérica

Olha a List ai gente!

```
public interface List<E> extends Collection<E>{
   ...
}
```

Eeeee lá! Então o E é o tipo genérico de List?

Exatamente! O *E* é o tipo genérico utilizado na especificação da *Interface List*. Percebe-se que o tipo *E* é oriundo da *Interface* Collection, e como não recebeu um tipo concreto em *List*, sua definição precisa ser estendida para *List*.

Generics - Exemplo de Definição de Classe Genérica

Definição de Classe Genérica

```
public class Aquecedor<E>{
  public boolean aquece(E entidade, Float
temperaturaAlvo){
    ...
  }
  public void arrefece(E entidade){
    ...
  }
}
```

A definição do tipo genérico está presente diretamente na especificação da classe *Aquecedor*.

Generics - Exemplo de Definição de Método Genérico

Definição de Método Genérica

public <T> boolean realizarComputacao(T tipo){...}

No caso da definição de métodos com *Generics*, o tipo genérico é definido juntamente com a especificação do método, a indicar um tipo que é independente da especificação da classe onde o método é declarado.

Bibliografia

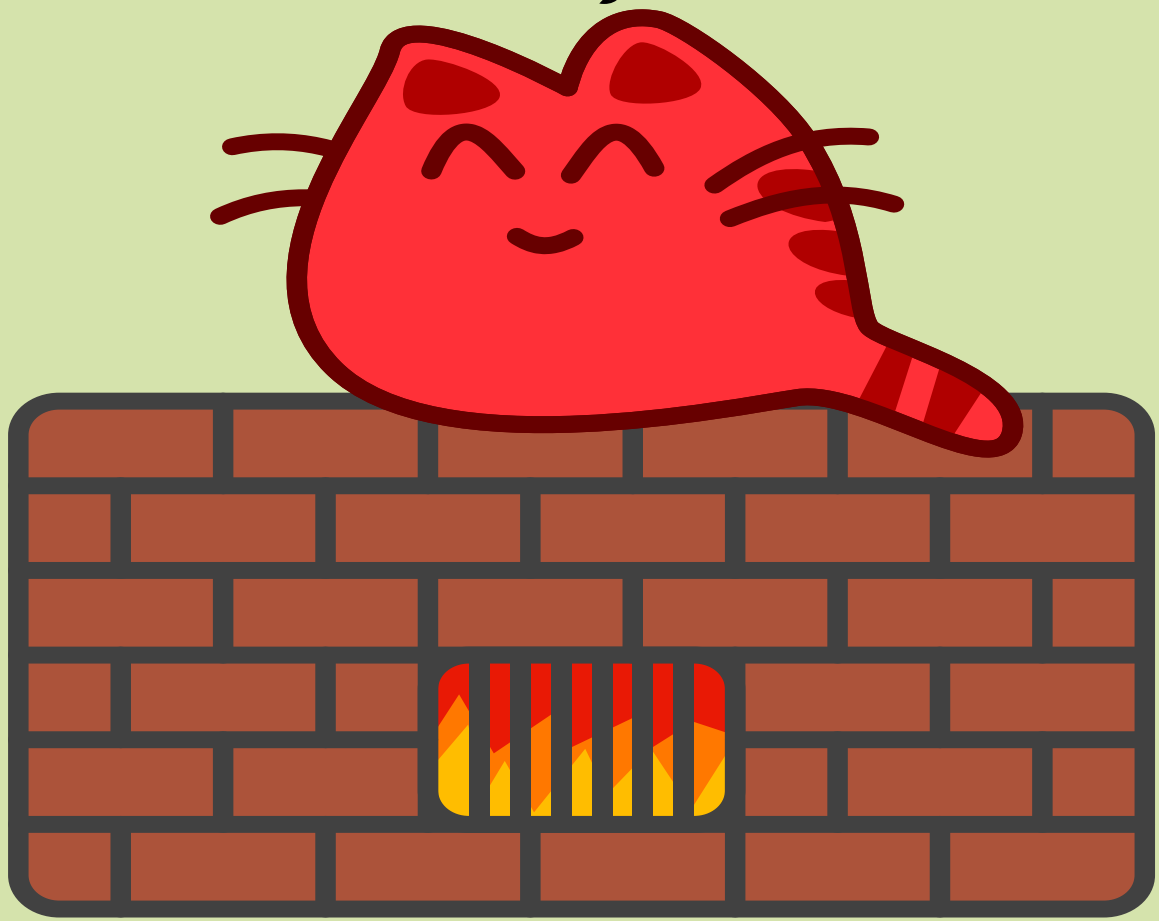
GOSLING, J.; JOY, B.; STEELE, G.; BRACHA, G.; BUCKLEY, A.; SMITH, D.; BIERMAN, G. **"The Java Language Specification: Java SE 16 Edition"**. Oracle. 2021. Disponível em: `https://docs.oracle.com/javase/specs/jls/se16/jls16.pdf`. Acesso em: 12 Jul. 2021.

BOYARSKY, J. and SELIKOFF, S. **"Oracle Certified Associate Java SE 8 Programmer I: Study Guide"**. Sybex. Indianapolis, Indiana, USA. 2015.

Bibliografia (Continuação)

BOYARSKY, J. and SELIKOFF, S. **“Oracle Certified Associate Java SE 8 Programmer II: Study Guide”**. Sybex. Indianapolis, Indiana, USA. 2016.

That's it folks!



Thank you for your attention!