Polimorfismo paramétrico

Prof. Hugo de Paula

PONTIFÍCIA UNIVERSIDADE CATÓLICA DE MINAS GERAIS
Departamento de Ciência da Computação

Sumário

Polimorfismo universal paramétrico: *Generics*

- Exemplo de Coleções no *Java Collections Framework v1.4*:

```
ArrayList listaDeProfessores = new ArrayList();

listaDeProfessores.add("Hugo de Paula");
     // Adiciona um professor na lista
listaDeProfessores.add("Zé da Silva");
     // Adiciona um professor na lista

String prof1 = (String) listaDeProfessores.get(0);
     // Recupera o 1o professor da lista

Professor prof2 = (Professor) listaDeProfessores.get(1);
     // Essa última linha irá compilar, uma vez que a função
     // get(n) retorna Object, mas irá produzir erro de execução.
```

Polimorfismo paramétrico: Tipos como parâmetros

Listas com tipos parametrizados

List<Tipo> variavel = **new** ArrayList<Tipo>();

- A classe ArrayList aceita um tipo como parâmetro:
  - Tipo passado como parâmetro usando < >.
  - Por compatibilidade: Versão antiga funciona, mas produz *warnings*.

```
ArrayList<String> listaDeProfessores = new ArrayList();
    // Define tipo da lista como parâmetro

listaDeProfessores.add("Hugo de Paula");
listaDeProfessores.add("Zé da Silva");

String prof1 = listaDeProfessores.get(0);
    // Sem type casting
Professor prof2 = (Professor) listaDeProfessores.get(1);
    // Produz erro de compilação (erro de tipo)
```

Generics: Classes

Classes parametrizadas

public class Nome<Tipo> { }

ou

public class Nome<Tipo1, Tipo2, ..., TipoN> { }

- Um tipo deve ser passado como parâmetro para <Tipo> no momento da construção do objeto.
- o resto da sua classe pode ser implementada baseada nesse nome de tipo.
  - Convenção de nomenclatura – usar apenas um caractere:

    T para Tipo, E para Elemento, N para Número, K para Chave – *Key*, e V para Valor.

Generics: variáveis e arranjos

- Não é possível se construir objetos ou arranjos com tipos parametrizados.

```
public class Shulambs<T> {
  private T campo;                  // ok
  private T[] arranjo;              // ok

  public Shulambs(T param) {
        campo = new T();            // erro
        arranjo = new T[10];        // erro
  }
}
```

Generics: variáveis e arranjos

- Pode-se criar variáveis e passar parâmetros.
- Pode-se fazer *type casting* de arranjos a partir de Object[].

```
public class ShulambsFixed<T> {
    private T campo;
    private T[] arranjo;

    @SuppressWarnings("unchecked")
    public ShulambsFixed(T param) {
        campo = param;
        arranjo = (T[]) new Object[10];
    }
}
```

Generics: comparação de objetos

- Generics usam semântica de referência.
- Deve-se comparar objetos de tipos parametrizados usando o método T.equals(T).

```
public class ArrayList<E> {
  ...
  public int indexOf(E value) {
    for (int i = 0; i < size; i++) {
    // if (elementData[i] == value) {
       if (elementData[i].equals(value)) {
          return i;
       }
    }
    return -1;
  }
}
```

Generics: interfaces

```
// Representa uma lista de valores
public interface List<E> {
    public void add(E value);
    public void add(int index, E value);
    public E get(int index);
    public int indexOf(E value);
    public boolean isEmpty();
    public void remove(int index);
    public void set(int index, E value);
    public int size();
}

public class ArrayList<E> implements List<E> { ... }

public class LinkedList<E> implements List<E> { ... }
```

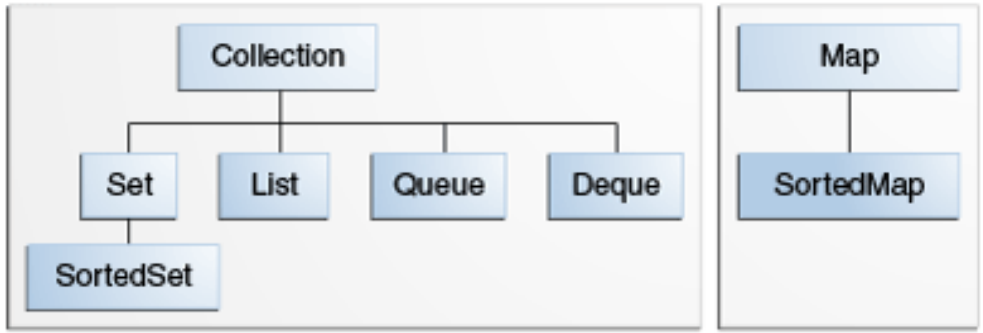
Collections

Collections

Uma coleção é um objeto que agrupo múltiplos objetos, como um *container*.

Java Collections Framework

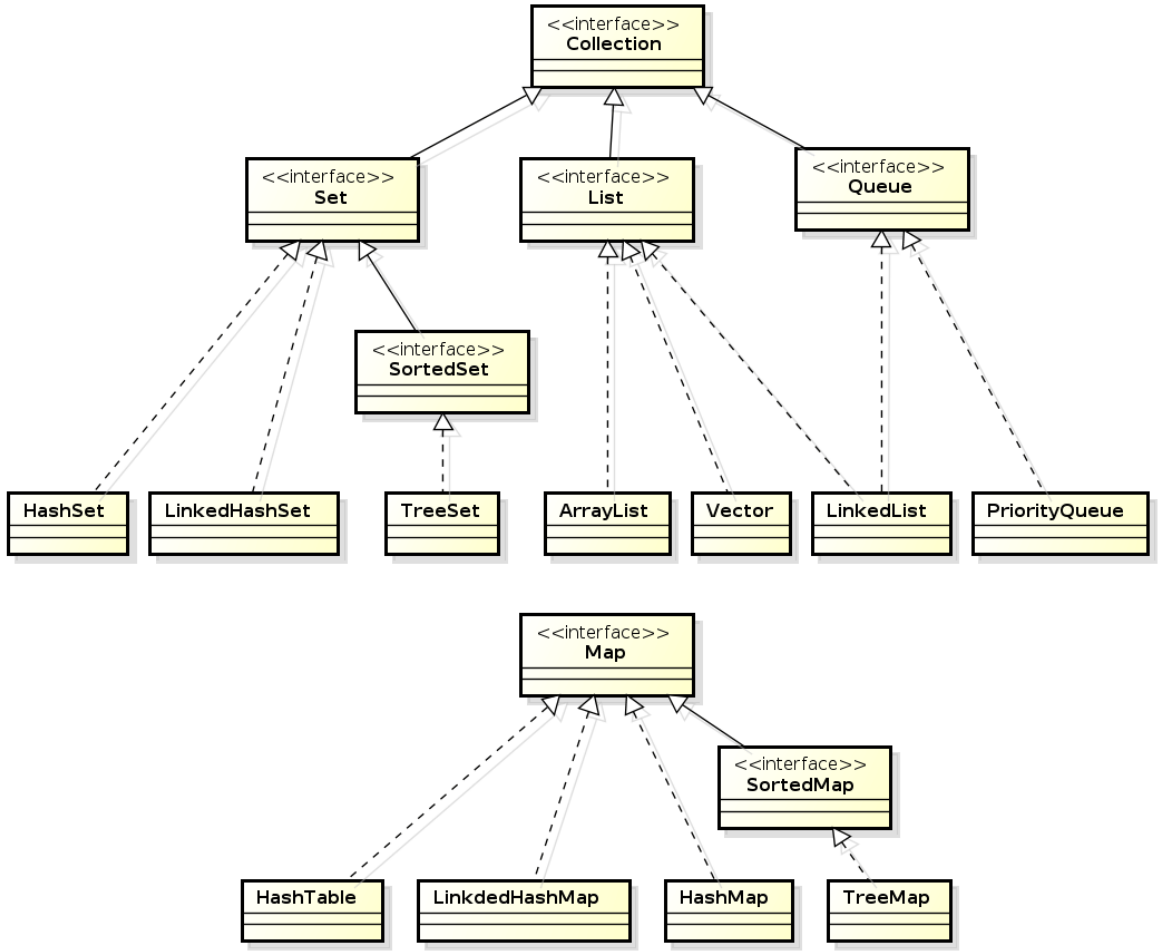
É uma arquitetura unificada para representação e manipulação de coleções, independentes de implementação.



[1]

[1] Oracle. *Collections: The Java Tutorial*. 2016.

Java Collections Framework



Sintaxe

- Exemplo de criação de coleção:

```
List<String> list = new ArrayList<String>();
```

- Processamento:
  - for-each

```
for (Object o: list ) System.out.println(o);
```

  - Iterador

```
Iterator<?> it = list.iterator();
while(it.hasNext())
if (!cond(it.next()))
it.remove();
```

Interface Collection<E>

- Inclusão / remoção:

```
boolean add(E elemento)
boolean remove(E elemento)
```

- Consultas:

```
int size()
boolean isEmpty()
boolean contains(Object elemento)
Iterator<E> iterator()
Object[] toArray()
```

- Operações com grupos:

```
boolean containsAll(Collection<?> coleção)
boolean addAll(Collection<? extends E> coleção)
void clear()
void removeAll(Collection<?> coleção)
void retainAll(Collection<?> coleção)
```

Interface Set<E> – conjuntos de elementos

- Conjuntos: não aceitam elementos duplicados.
- Principais métodos:
  - Herdados da interface Collection.
- Implementações:
  - Classe HashSet – acesso mais rápido.
  - Classe TreeSet – acesso ordenado.
  - Classe LinkedHashSet – mais versátil.
- Considerações:
  - Dois conjuntos são iguais se contiverem os mesmos elementos (determinado através dos métodos equals() e hashCode()).

Interface SortedSet<E> – conjunto ordenado

- Principais métodos:

```
E first()
E last()

SortedSet<E> headSet(E aoElemento)
SortedSet<E> subSet(E doElemento, E aoElemento)
SortedSet<E> tailSet(E doElemento)

Comparator<? super E> comparator()
```

Exemplo de conjuntos

```
public class ExemploSet {
    public static void main( String[] args ) {

        Set<String> conjunto = new HashSet<String>();

        conjunto.add( "Bernardo" );
        conjunto.add( "Carolina" );
        conjunto.add( "Felipe" );
        conjunto.add( "Carolina" );
        conjunto.add( "Ana" );

        System.out.println( conjunto );

        Set<String> conjuntoOrdenado =
        new TreeSet<String>( conjunto );

        System.out.println( conjuntoOrdenado );
    }
}
```

Exemplo de Problema: Bacteria

- Cientistas criaram uma bactéria que come lixo. Ela pesa 10g e consome metade do seu peso a cada dia. O peso não se altera, ou seja, o lixo é todo metabolizado.
- A cada dia, todas as bactérias existentes são clonadas. Cada nova bactéria dura apenas 5 dias e depois morre.
- Modele a classe Bacteria. Deve haver métodos para:
  - Retornar quanto lixo ela come (que é um valor fixo);
  - Criar uma nova bactéria (clonando a atual);
  - Simular a passagem de um dia.
- A seguir, implemente uma simulação: crie uma bactéria e simule a passagem de 10 dias, exibindo a quantidade de bactérias existentes e o total de lixo consumido. Utilize os métodos da classe Bacteria.

Solução parcial: Bacteria

```
public class Bacteria implements Cloneable {
    private double peso;
    private int diasRestantes;

    public Bacteria() {
        diasRestantes = 5;
        peso = 10;
    }
    public void passaDia() {
        diasRestantes--;
    }
    public boolean morreu() {
        return (diasRestantes <= 0);
    }
    @Override
    public Bacteria clone() throws CloneNotSupportedException {
        return new Bacteria();
    }
}
```

Solução parcial: Bacteria

```
public static void main(String[] args) {
    Collection<Bacteria> colonia = new HashSet<Bacteria>();
    Collection<Bacteria> novas = new HashSet<Bacteria>();
    colonia.add(new Bacteria());
    for (int i = 0; i < 10; i++) {
        for (Bacteria o : colonia) {
            o.passaDia();
            try {
                novas.add((Bacteria) o.clone());
            } catch (Exception e) {
                System.out.println("Bactéria não clonável");
            }
        }
        colonia.addAll(novas);
        novas.clear();
    }
    System.out.println(colonia.size());
}
```

Interface List<E> – listas

- Coleção indexada com possibilidade de chaves duplicadas
- Principais métodos:

```
void add( int índice, E elemento )
boolean add( E elemento )
boolean addAll( int índice, Collection<? extends E> coleção )

E get( int índice )
E set( int índice, E element )
int indexOf( Object elemento )
int lastIndexOf( Object elemento )

E remove( int índice )
List<E> subList( int índiceInicial, int índiceFinal )
```

Implementações de listas

- Classe ArrayList<E> – semelhante a vetores dinâmicos.
  - Implementa os métodos da interface.
- Classe LinkedList<E> – manipulação sequencial de elementos (filas, pilhas, deques, etc.).
  - Implementa métodos adicionais, além dos da interface:

```
void addFirst( E elemento )
void addLast( E elemento )
E getFirst()
E getLast()
Object removeFirst()
Object removeLast()
```

Exemplo de listas

```
public class ExemploDeListas {
    public static void main(String[] args) {
        List<String> lista = new ArrayList<String>();
        lista.add("Bernardo");      lista.add("Carolina");
        lista.add("Felipe");        lista.add("Carolina");
        lista.add("Clara");
        System.out.println(lista);
        System.out.println("2: " + lista.get(2));
        LinkedList<String> fila = new LinkedList<String>();
        fila.addFirst("Bernardo");  fila.addFirst("Carolina");
        fila.addFirst("Felipe");    fila.addFirst("Elizabeth");
        fila.addFirst("Clara");
        System.out.println(fila);
        fila.removeLast();
        fila.removeLast();
        System.out.println(fila);
    }
}
```

Interface Map<K,V> – mapeamentos

- Associações de chaves (*K – Keys*) e valores (*V – Values*).
- Principais métodos para alteração:

```
V put( K chave, V valor )
V remove( K chave )
void putAll( Map<? extends K, ? extends V> mapeamento )
void clear()
```

- Principais métodos para consulta:

```
V get( K chave )
boolean containsKey( Object chave )
boolean containsValue( Object valor )
int size()
boolean isEmpty()
```

- Principais métodos para grupos:

```
Set<K> keySet()
Collection<V> values()
Set<Map.Entry<K,V>> entrySet()
```

Map.Entry<K,V> – elementos de mapeamentos

- Representa um par chave-valor.
- Principais métodos:

```
boolean equals( Object objeto )
K getKey();
V getValue();
V setValue( V valor );
```

- Implementações de mapeamentos:
  - **class** HashMap<K,V> – agilidade, permite `nulls`.
  - **class** TreeMap<K,V> – ordenação (árvore balanceada).
  - **class** LinkedHashMap<K,V> – ordem de iteração previsível.

SortedMap<K,V> – mapeamentos ordenados

- Implementado pelo TreeMap<K, V>.
- Principais métodos:

```
Comparator<? super K> comparator();
SortedMap<K,V> headMap( K ateChave );
SortedMap<K,V> subMap( K daChave, K ateChave );
SortedMap<K,V> tailMap( K daChave );
K firstKey();
K lastKey();
```

Exemplo de mapeamentos

```
public class ExemploMap {
    public static void main(String[] args) {
        Map<String, Integer> mapH = new HashMap<String, Integer>();
        Integer UM = new Integer(1);
        for (int i = 0, n = args.length; i < n; i++) {
            String chave = args[i];
            Integer frequencia = mapH.get(chave);
            if (frequencia == null) {
                frequencia = UM;
            } else {
                int valor = frequencia.intValue();
                frequencia = new Integer(valor + 1);
            }
            mapH.put(chave, frequencia);
        }
        System.out.println(mapH);
        Map<String, Integer> mapOrd = new TreeMap<String, Integer>(map
        System.out.println(mapOrd);
    }
}
```