Programação Orientada Pelos Objectos

Tipos genéricos

2

Programação genérica

Programação genérica

- Em geral, a programação genérica tem como objectivo a criação de código fonte que possa ser utilizado com diferentes tipos de dados
- Torna o código fonte mais estável pois permite que eventuais erros possam ainda ser detectados na fase de compilação e não posteriormente, durante a execução
- Mecanismos a adoptar
  - Herança
  - **Variáveis de tipo**

Tipos genéricos

- Um tipo genérico é um tipo referenciado, classe ou interface, que usa na sua definição uma ou mais variáveis "de tipo" – os parâmetros de tipo
  - Quando instanciado, o tipo genérico é substituído por um tipo concreto, ou seja, após instanciação de um tipo genérico, obtemos um tipo parametrizado concreto
- Convenção: uma letra maiúscula para os parâmetros
  - de tipo genérico `T, U`
  - de colecção `E`

Consideremos o seguinte exemplo


```
public class Box {
  private Object object;

  public void add(Object object) {
    this.object = object;
  }
  public Object get() {
    return object;
  }
}
```

- Abordagem à la Java 2:
  - Os métodos recebem e devolvem `Object`
  - Pode ser usado qualquer tipo de argumento, desde que este não seja primitivo
- Não é possível restringir o tipo dos elementos

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

  Box animalBox = new Box();
  // Só seria aceitável colocar objectos
  // do tipo Animal nesta caixa!
  Animal snoopy = new DogClass("Snoopy");
  animalBox.add(snoopy);

  Animal snuupy = (DogClass) animalBox.get();
  System.out.println(snuupy);
}
```

Erro em tempo de execução


```
public class Box {
  private Object object;

  public void add(Object object) {
    this.object = object;
  }
  public Object get() {
    return object;
  }
}
```

- Imagine-se que, por algum motivo, um programador modifica o código e passa uma String como argumento
- Qual é o resultado?
  - **Erro em run-time**
    (o problema dos *downcasts*)

```
public static void main(String[] args) {
  Box animalBox = new Box();
  // Só seria aceitável colocar objectos
  // do tipo Animal nesta caixa!
  animalBox.add("String qualquer como argumento");
  Animal snuupy = (DogClass) animalBox.get();
  System.out.println(snuupy);
}
```

```
Exception in thread "main" java.lang.ClassCastException: java.lang.String cannot be cast to poo.DogClass
        at poo.GenericsTester.taskB(GenericsTester.java:33)
        at poo.GenericsTester.main(GenericsTester.java:9)
```

Mesmo exemplo mas com tipo genérico


```
/**
* Versão da classe Box com tipo
genérico
**/
public class Box<T> {
  private T t; // T -> "Type"

  public void add(T t) {
    this.t = t;
  }
  public T get() {
    return t;
  }
}
```

```
public static void main(String[] args) {
  Box<Animal> animalBox = new Box<Animal>();

  Animal snoopy = new DogClass("Snoopy");
  animalBox.add(snoopy);

  Animal snuupy = animalBox.get();
  System.out.println(snuupy);
}
```

- A variável de tipo **T** pode ser usada no interior da classe **Box**
- **T** é uma variável especial – pode ser tipo de classe, de interface, de outra variável de tipo, exceptuando tipos primitivos
- **T** também se designa por parâmetro de tipo formal da classe **Box**

Erro em tempo de compilação


```
/**
* Versão da classe Box com tipo genérico
**/
public class Box<T> {
  private T t; // T -> "Type"

  public void add(T t) {
    this.t = t;
  }
  public T get() {
    return t;
```

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

    Box<Animal> animalBox = new Box<Animal>();

    animalBox.add("String qualquer como argumento");

    Animal snuupy = animalBox.get();; // no cast!
    System.out.println(snuupy);
}

}
```

The method add(Animal) in the type Box<Animal> is not applicable for the arguments (String)

Note-se que variáveis de tipo não são tipos, nem **T** faz parte da classe **Box**. Na fase de compilação, toda a informação genérica é retirada, ficando apenas **Box.class**

É por isso que não podemos, por exemplo, criar instâncias de T, i.e., usar T em expressões com **new**.

Uso de vectores estáticos com tipos genéricos

- E se quisermos instanciar um tipo genérico?
  Ou seja, criar um novo objecto usando **new**?
- No caso geral, não podemos: os tipos genéricos do Java são como as interfaces Java
  - Da mesma maneira que não podemos instanciar uma interface, também não podemos instanciar T
- Na maioria dos contextos, a instanciação de tipos genéricos não é permitida nem possível. Mas podemos contornar esta limitação na criação de vectores estáticos:

```
@SuppressWarnings("unchecked")
public void clear() {
    elems = (E[]) new Object[capacity];
    counter = 0;
}
```

Uso de vectores estáticos com tipos genéricos

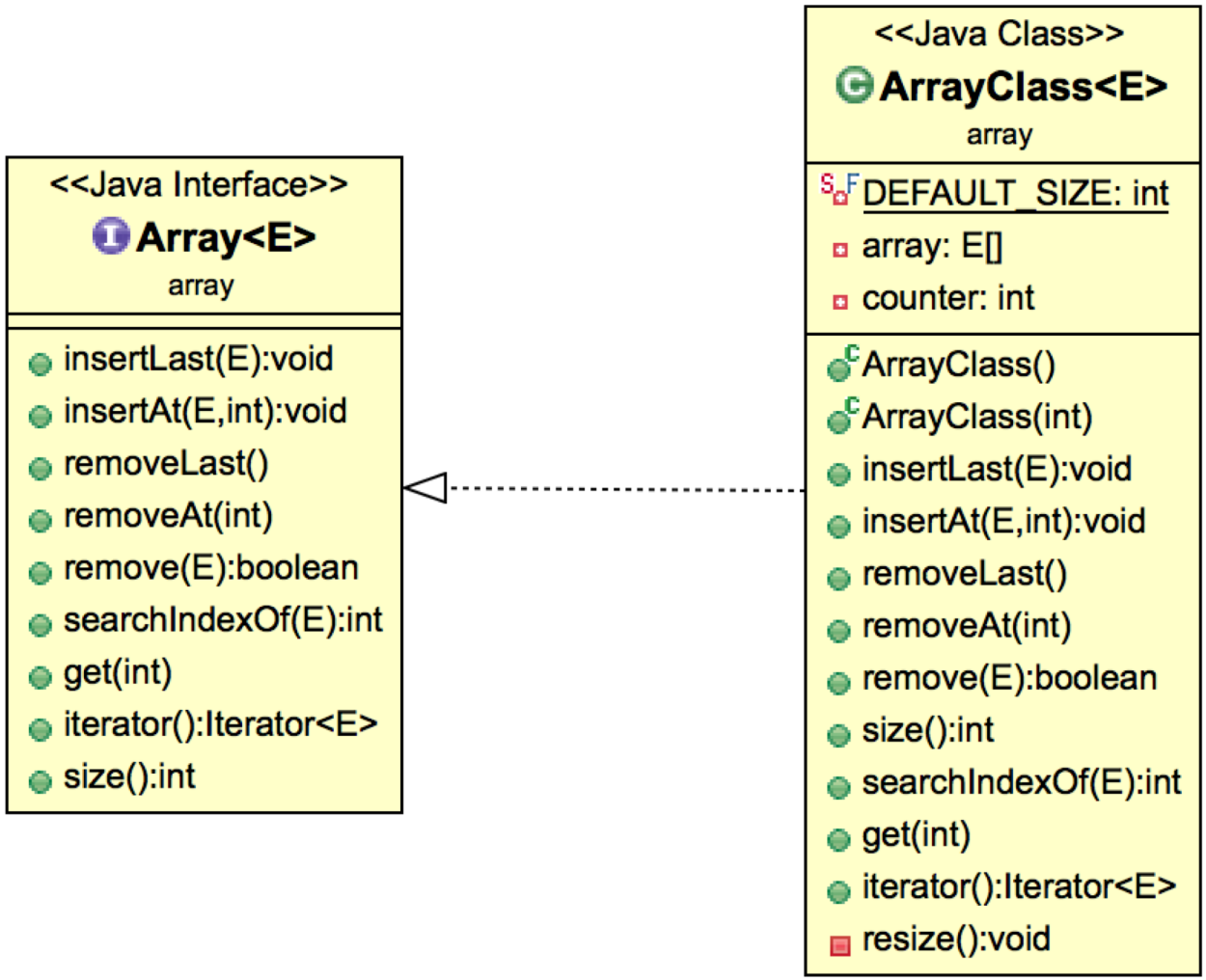
- Um tipo genérico T é na realidade um *alias* para **`java.lang.Object`**
- A diferença entre T e **`Object`** é que com T o compilador possui mais informação sobre os tipos reais, quando está a compilar o programa
- Porém, essa informação adicional é descartada. Quando o programa corre, só resta **`Object`**

```
@SuppressWarnings("unchecked")
public void clear() {
    elems = (E[]) new Object[capacity];
    counter = 0;
}
```

11

Array genérico

Array genérico



Interface do array genérico


```
public interface Array<E>{
  /**
   * Insere o elemento <code>e</code> na ultima posicao do vector
   * @param e elemento a inserir no vector
   */
  void insertLast(E e);

  /**
   * Insere o elemento <code>e</code> na posicao <code>pos</code> do vecto
   * @param e elemento a inserir no vector
   * @param pos posicao do vector a inserir o elemento
   * @pre pos < size()
   */
  void insertAt(E e, int pos);

  /**
   * Remove o ultimo elemento do vector
   * @pre size() > 0
   */
  void removeLast();
```

Interface do array genérico


```
/**
 * @param pos posicao do elemento a remover do vector
 * @pre pos < size()
 */
void removeAt(int pos);


/**
 * @param e elemento a procurar do vector
 * @return a posicao do elemento no vector,
 *                      <code>-1</code> caso o elemento nao exista
 */
int searchIndexOf(E e);
```

Interface do array genérico


```
    /**
     * Procura a posicao do elemento <code>e</code> no vector
     * @param pos posicao do vector do elemento a devolver
     * @return o elemento na posicao <code>pos</code>,
     * @pre pos < size()
     */
    E get(int pos);

    /**
     * Devolve o numero de elementos no vector
     * @return o numero de elementos no vector
     */
    int size();

    /**
     * Devolve um iterador para os contactos
     * @return iterador para os contactos
     */
    Iterator<E> iterator();
}
```

Array genérico


```
public class ArrayClass<E> implements Array<E> {
    private static final int SIZE = 50;
    private E[] elems;
    private int counter;

    @SuppressWarnings("unchecked")
    public ArrayClass() {
        elems = (E[]) new Object[SIZE];
        counter = 0;
    }

    public void insertLast(E e) {
        if (counter == elems.length) resize();
        elems[counter++] = e;
    }
    public void removeLast() {
        elems[--counter] = null;
    }
```

Array genérico


```
public void insertAt(E e, int pos) {
    for(int i = counter-1; i >= pos; i--)
        elems[i+1] = elems[i];
    elems[pos] = e;
    counter++;
}

public void removeAt(int pos) {
    for(int i = pos; i< counter -1; i++)
        elems[i] = elems[i+1];
    elems[--counter] = null;
}

public boolean search(E e) {
     return indexOf(e) != -1;
}
```

Array genérico


```
public int searchIndexOf(E e) {
      int i = 0;
      int result = -1;
      boolean found = false;
      while (i<counter && !found)
            if (elems[i].equals(e))
                  found = true;
            else
                  i++;
      if (found) result = i;
      return result;
}

public E get(int pos) {
      return elems[pos];
}
```

Array genérico


```
    public int size() {
        return counter;
    }

    public Iterator<E> iterator() {
        return new ArrayIterator<E>(elems, counter);
    }

    @SuppressWarnings("unchecked")
    private void resize() {
        E tmp[] = (E[]) new Object[2*elems.length];
        for (int i=0;i<counter; i++)
            tmp[i] = elems[i];
        elems = tmp;
    }
}
```

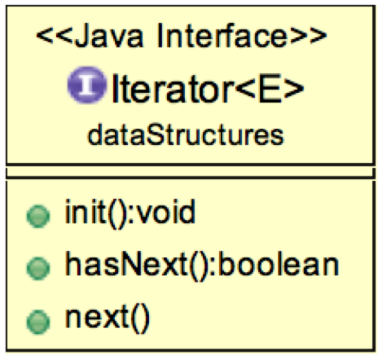
Iterador genérico

Iterator genérico


```
public interface Iterator<E> {
   /**
    * Vai para o inicio da coleccao de objectos
    */
   void init();

   /**
    * Devolve <code>true</code> se existirem mais objectos a visitar,
    * ou <code>false</code>, caso contrario
    * @return se existem mais objectos a visitar
    */
   boolean hasNext();

   /**
    * Devolve o proximo objecto
    * @return proximo objecto
    * @pre hasNext()
    */
   E next();
}
```



Iterator genérico


```
public class ArrayIterator<E> implements Iterator<E>{
    private E[] elems;
    private int counter;
    private int current;

    public ArrayIterator(E[] elems, int counter) {
         this.elems = elems;
         this.counter = counter;
         init();
    }

    public void init() { current = 0; }

    public boolean hasNext() { return current < counter;}

    public E next() {
         return elems[current++];
    }
}
```