


Laboratório de Engenharia de Software

INF1636 – PROGRAMAÇÃO ORIENTADA A OBJETOS

Departamento de Informática – PUC-Rio

Ivan Mathias Filho
ivan@inf.puc-rio.br


Programa – Capítulo 14

Laboratório de Engenharia de Software

- Generics

© LES/PUC-Rio

Programa – Capítulo 14




Laboratório de Engenharia de Software

- Generics

© LES/PUC-Rio

Generics – introdução



Laboratório de Engenharia de Software

- Até a versão 1.4 do Java (JDK 1.4) as classes de coleção (Vector, Hashtable , Array e etc) manipulavam objetos do tipo **Object**;
- Objetos de quaisquer classes podiam ser inseridos nessas coleções, uma vez que qualquer classe é descendente de **Object**;
- Quando um objeto era retirado de uma coleção como essas tornava-se necessário descobrir o seu tipo, para que uma conversão (*cast*) segura pudesse ser realizada.

© LES/PUC-Rio

Casting de um elemento (1)



```
import java.util.Vector;

public class Ex {
    public static void main(String[] args) {
        Vector ls=new Vector();

        ls.add("INF1381");
        ls.add("INF1318");
        ls.add("INF1337");

        lista(ls);
    }
}
```

Casting de um elemento (2)



```
public static void lista(Vector v) {
    String s;
    Object o;

    for(int i=0;i<v.size();i++) {
        o=v.get(i);
        if(o instanceof String) {
            s=(String)o;
            System.out.println(s);
        }
    }
}
```

Generics – motivação (1)



- O uso de conversão de tipo não é uma estratégia segura, pois apenas em tempo de execução é possível saber o tipo de um elemento de uma coleção;
- No exemplo anterior, se `v.get(i)` não fosse um objeto do tipo `String` uma exceção seria levantada;
- Para evitar problemas na conversão de tipo, o operador `instanceof` foi usado para testar o tipo do elemento `v.get(i)` antes da conversão.

Generics – motivação (2)



- Para tornar a programação mais segura, foi introduzido o conceito de *generics* a partir da versão 1.5 de Java;
- Com o uso dos *generics* é possível definir coleções de objetos de tipos específicos;
- Dessa forma, não é necessário realizar conversões quando elementos de uma coleção são recuperados;
- A verificação de tipos passou a ser feita em tempo de compilação.

Generics - declaração



- O primeiro passo para criar uma coleção de um determinado tipo é declarar a coleção e o tipo desejado entre chaves angulares.

```
Vector<String> ls;
```

- O segundo passo é criar um objeto que represente uma coleção de objetos do tipo desejado.

```
ls=new Vector<String>();
```

© LES/PUC-Rio

Iteração sobre a coleção



- Dessa forma, não é mais necessário realizar conversões de tipo na recuperação dos elementos de uma coleção.

```
public static void lista(Vector<String> v) {  
    String s;  
  
    for(int i=0;i<v.size();i++)  
        System.out.println(v.get(i));  
}
```

© LES/PUC-Rio

Exemplo completo



```
import java.util.Vector;

public class Ex {
    public static void main(String[] args) {
        Vector<String> ls=new Vector<String>();

        ls.add("INF1381");
        ls.add("INF1318");
        ls.add("INF1337");

        lista(ls);
    }

    public static void lista(Vector<String> v) {
        for(int i=0;i<v.size();i++)
            System.out.println(v.get(i));
    }
}
```

© LES/PUC-Rio

Exemplo – classe No



- O exemplo anterior usou uma classe já existente, definida em uma das bibliotecas nativas de Java (`java.util.Vector`);
- No exemplo a seguir criaremos a nossa própria classe genérica, utilizando, para tal, a classe `No`, definida para representar um nó de uma lista encadeada;
- A classe `No` original foi concebida de modo a poder ser empregada em listas de quaisquer tipos de objetos;
- Como ela não utiliza *generics*, foi necessário declarar o elemento da lista como sendo um `Object`.

© LES/PUC-Rio

A classe No original



```
public class No {
    private Object elem;
    private No prox;

    public No(Object e, No p) {
        elem=e;
        prox=p;
    }

    public Object getElem() {
        return elem;
    }

    public No getProx() {
        return prox;
    }

    public void setProx(No o) {
        prox=o;
    }
}
```

© LES/PUC-Rio

A nova classe No



- A nova definição utiliza um parâmetro para criar uma classe No que possa ser empregada na criação de listas genéricas.

```
public class No <X> {
    private X elem;
    private No<X> prox;

    public No(X e, No<X> p)
    {...}

    public X getElem()
    {...}

    public No<X> getProx()
    {...}

    public void setProx(No<X> o)
    {...}
}
```

© LES/PUC-Rio

Exemplo completo



```
public class Ex {  
    public static void main(String[] args) {  
        No<String> header=new No<String>("INF1337",null);  
  
        header=new No<String>("INF1318",header);  
        header=new No<String>("INF1381",header);  
  
        for(No<String> n=header;n!=null;n=n.getProx())  
            System.out.println(n.getElem());  
    }  
}
```