Tipos Genéricos 2

Maurício Linhares

Conteúdo da Aula

- Erasure (apagamento) e tradução de código;
- Usando curingas (?) nos parâmetros;
- Arrays e tipos genéricos
- Limite inferior utilizando “super”;
- Composição de limites genéricos;
- A classe genérica Class<T>
- Tipos brutos;

Tradução de código genérico

- Genéricos são implementados a nível de compilador;
- Não são uma construção da máquina virtual e não existem em bytecode;
- Classes genéricas são sempre únicas e não uma classe para cada tipo declarado;

Exemplo de tradução

```
List<String> lista = new ArrayList<String>();
lista.add( "Maurício" );
System.out.println( lista.iterator().next() );

//Traduzindo

List lista = new ArrayList();
lista.add( "Maurício" );
System.out.printf( (String) lista.iterator().next() );
```

Quando o código genérico é compilado

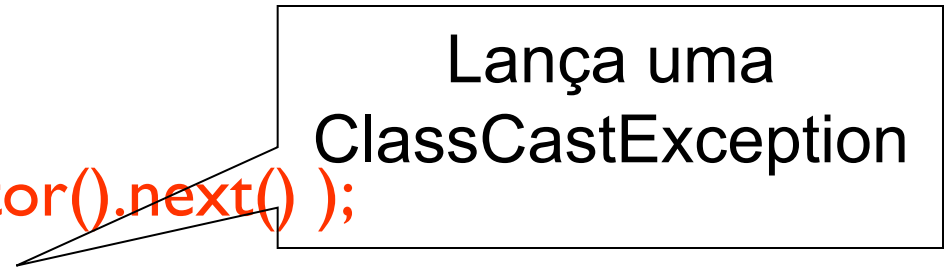
- Toda a informação de tipos genéricos é retirada (apagada);
- Todos os parâmetros são definidos pelo limite superior (normalmente Object);
- Sempre que o código não estiver com os tipos corretos uma operação de "cast" será adicionada;
- A coerência dos tipos nunca está em risco;

Exemplo de uso incorreto de genéricos

```
List<String> lista = new ArrayList<String>();
List listaBruta = lista;

listaBruta.add(1);
lista.add("Meu Nome");

System.out.printf( lista.iterator().next() );
```



Exemplo de classes genéricas como classes únicas

```
ArrayList<String> listaString = new ArrayList<String>();
ArrayList lista = new ArrayList();


System.out.println
(
   listaString.getClass().equals( lista.getClass() )
 );
```

Curingas como parâmetros genéricos

- Utilizados quando não for necessário usar o tipo parametrizado dentro do método;
- Podem ter limites superiores ou inferiores;
- São definidos por um sinal de interrogação (?);
- Coleções definidas com curingas não podem receber objetos;

Exemplo de declaração de método com curingas

```
public static void printGenerico
                                    ( Collection <?> colecao )
{

   for (Object object : colecao) {
         System.out.println( object );
   }

}
```

Desvantagens do uso de curingas

- Como não é declarado um parâmetro de método, o curinga (?) não pode ser referenciado dentro do corpo do método;
- Não é possível adicionar objetos em coleções declaradas com curingas;
- Se não houver uma declaração de limite superior, só é possível chamar métodos da classe Object no curinga;

Arrays e tipos genéricos

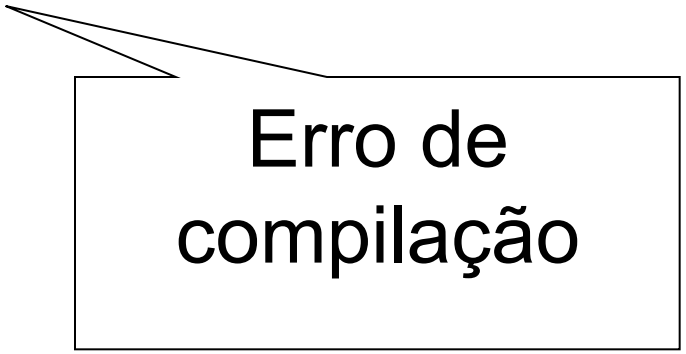
- Arrays não podem ser definidos com parâmetros de tipo, pois os tipos genéricos não existem no código de máquina gerado;
- Arrays só podem conter tipos parametrizados (genéricos) se o parâmetro for um curinga sem limite;

Exemplo do uso incorreto de arrays

```
public <T> T[] criarArray( Collection<T> col) {

        return new T[ col.size() ];

}
```



Arrays compostos de tipos genéricos

```
List<?>[] listas = new ArrayList<?>[10];

List<String> nomes = new ArrayList<String>();
nomes.add( "Maurício" );

listas[0] = nomes;

System.out.println( "O nome é " + listas[0].get(0) );
```

Limites inferiores em genéricos

- São utilizados quando os tipos aplicáveis são supertipos do tipo definido;
- Comumente utilizados em algoritmos de ordenação, comparação e classificação de objetos;
- Definidos pela sintaxe <(E|?) super T>;

Exemplo de uso de limites inferiores – classe TreeSet <E>

- Coleção ordenada de objetos;
- Não permite objetos duplicados;
- Não é thread-safe;

A interface Comparator <T>

- Define um algoritmo para a ordenação completa entre um grupo de objetos;
- Define um único método, "compare()", que recebe dois objetos do mesmo tipo e retorna um número negativo se o primeiro for menor que o segundo, 0 se os dois forem iguais e um número positivo se o primeiro for maior que o segundo;

Construtor de TreeSet <E>

TreeSet(Comparator<? super E> c)

A ordenação é feita por um Comparator, que pode ser do tipo <E> ou de qualquer um dos seus supertipos.

Limites inferiores em TreeSet<E>

Em um TreeSet<String> podem ser utilizados dois objetos Comparator:

Comparator<String>

Comparator<Object>

Pois um String é comparável tanto como String como pela sua superclasse, Object.

Composição em limites genéricos

- É possível utilizar múltiplos limites em uma declaração de limite superior ("extends");
- Os limites são declarados um após o outro separados por um &;
- A primeira declaração de limite vai ser utilizada como tipo da variável após o apagamento;

Exemplo de composição de limites genéricos

```
private <T extends Object & Comparable> void
   printObject( Collection<T>  objetos ) {

        for (T objeto : objetos) {
                System.out.println( objeto );
        }

}
```

A classe Class<T>

- Tornou-se uma classe genérica no Java 5;
- Aumenta a segurança ao lidar com código reflexivo (que usa reflexão);
- Pode ser utilizada na passagem de parâmetros genéricos;

Novas definições de classe

String.class -> Class<String>

Number.class -> Class<Number>

Date.class -> Class<Number>

Exemplo de uso da classe Class parametrizada

```
Class<Date> dataClass = Date.class;
Date dataAtual = dataClass.newInstance();

System.out.println( dataAtual );

Class data = dataClass;
Date dataAtualizada = (Date) data.newInstance();

System.out.println( dataAtualizada );
```

Usando a classe Class<T> para carregamento dinâmico

```
Serializable ser = Class.forName
        ( "Calendario" )      .asSubclass( Serializable.class )
                .newInstance();

System.out.println( ser.getClass() );
```

Utilizando o objeto Class<T> em métodos genéricos

```
public <T> T createObject( Class<T> classe ) throws
   Exception {

        return classe.newInstance();

}
```

Tipos Brutos – Raw Types

- São classes genéricas declaradas sem nenhum parâmetro de tipo;
- Garantem a retrocompatibilidade da linguagem;
- Todos os parâmetros de tipo são "trocados" por Object;
- Não faz nenhum teste nem validação nos parâmetros;

Exemplo de declaração de tipos brutos

```
List lista = new ArrayList();

lista.add( "Maurício" );
lista.add( new Date() );

System.out.println(lista.iterator().next());
```

Nenhuma declaracao de tipo

Interoperabilidade com código legado

```
public class Calendario {
    private List datas = new ArrayList();

    public List getDatas() {return this.datas;}
    public void setDatas( List lista ) { this.datas = lista; }

    public void imprimirDatas() {
            out.println( "As datas são:" );
            for (Object data : datas) {
                    out.println( data );
            }
    }
}
```

Interoperabilidade com código legado

```
Calendario calendario = new Calendario();
List<Date> datas = new ArrayList<Date>();

datas.add( new Date() );
datas.add( new Date() );

calendario.setDatas(datas);
calendario.imprimirDatas();
```

Compatibilidade X Segurança

Manter a compatibilidade com o código legado é importante, mas em muitos momentos é perdida a segurança dos tipos genéricos, portanto os tipos brutos devem ser utilizados apenas em último caso.

Revisão – Erasure (apagamento) e tradução

- O que é apagamento?
- Quem faz o apagamento?
- Qual é o tipo que toma o lugar do tipo "apagado"?
- Quem faz a tradução do código "apagado" para código Java comum?

Revisão – Curingas como parâmetros genéricos

- Como se declara um curinga em um parâmetro genérico?
- Onde os curingas devem ser utilizados?
- É possível declarar uma coleção com um curinga?
- É possível adicionar um item a uma coleção de curinga?

Revisão – Curingas como parâmetros genéricos

- Curingas podem ter limites?
- Quando não há limite, qual é o tipo tido como limite em um curinga?
- Qual tipo de container de objetos só pode conter objetos parametrizados com curingas?

Revisão – limite inferior em genéricos

- Qual a palavra utilizada para definir um limite inferior em genéricos?

- Como funciona a declaração genérica de limite inferior?

Revisão – composição de limites genéricos

- Qual a sintaxe utilizada na composição de limites?
- Qual vai ser o tipo da variável após o apagamento do código?

Revisão – a classe Class<T>

- Como se declaram os tipos de classe usando a classse Class<T>?
- Onde essa funcionalidade pode ser utilizada?

Revisão – Tipos Brutos

- O que são tipos brutos?
- Onde eles são utilizados?
- Eles são seguros?

Laboratório - Apagamento

- Desenvolver código genérico que adicione um conjunto de objetos String em um List<String>;
- Refazer o exercício anterior traduzindo todo o código para chamadas comuns;
- Desenvolver código genérico que adicione um conjunto de objetos quaisquer em List<String> e depois tente retirálos;

Laboratório – Uso de curingas

- Desenvolver código que imprima objetos Date (e suas subclasses) utilizando objetos DateFormat (e suas subclasses) passados também como parâmetro usando curingas;

Laboratório – limites inferiores

- Desenvolver dois objetos Comparator, um para a classe Object e outro para a classe Number e os utilizar como ordenadores em um TreeSet de objetos Integer;

Laboratório – Composição de limites

- Compor um limite de método que só permita objetos que herdem da classe Number e implementem Serializable;
- Compor um limite de método que só permita objetos que implementem Serializable e Cloneable;

Laboratório – classe Class<T>

- Desenvolver um método genérico que crie objetos do tipo <T>;
- Desenvolver um método genérico que carregue uma classe que implemente a interface Comparable, crie uma instância do objeto e imprima o nome da classe completo no console;

Laboratório – tipos brutos

- Desenvolver código que utilize as coleções do Java sem utilizar tipos genéricos, identificando os métodos que mostram "avisos" quando compilados usando "//" no fim da linha;
- Reescrever todo o código utilizando genéricos;

Mais informações

- JavaDoc do Java SE 1.5 - http://java.sun.com/j2se/1.5.0/docs/api/
- DEITEL, H. M.; DEITEL, P. J.; **Java Como Programar 6ª Edição**. Editora Campus, 2005.
- Grupo de Usuários Java – http://www.guj.com.br/