NGÔN NGỮ LẬP TRÌNH JAVA

Nội dung

Cơ bản về ngôn ngữ lập trình Java

- Lập trình hướng đối tượng
- Biến, từ khoá, kiểu dữ liệu
- Biểu thức, các cấu trúc điều khiển
- Dữ liệu kiểu mảng

Các khía cạnh nâng cao của lập trình hướng đối tượng

- Thiết kế lớp
- Thiết kế lớp nâng cao
- Xử lý ngoại lệ
- Generics

Generics – Tham số hoá kiểu dữ liệu

Nội dung

1. Giới thiệu về Generics
2. Generic classes
3. Generic methods
4. Wildcard

1. Giới thiệu về Generics

- Để truyền đối số vào một phương thức:
  - Đặt các đối số vào trong ngoặc nhọn ();
  - Truyền các đối số cho phương thức
- Khái niệm Generics:
  - Thay vì truyền các đối số,truyền kiểu dữ liểu
  - Tham số hoá kiểu dữ liệu
  - Trừu tượng hoá các kiểu dl khi sử dụng trong các collections (vd JCF)
  - <E> : khai báo kiểu generics

Vấn đề của Pre-JDK 1.5: Chuyển kiểu ko an toàn

```
// Pre-JDK 1.5
import java.util.*;
public class ArrayListWithoutGenericsTest {
   public static void main(String[] args) {
      List strLst = new ArrayList();  // List and ArrayList holds Objects
      strLst.add("alpha");            // String upcast to Object implicitly
      strLst.add("beta");
      strLst.add("charlie");
      Iterator iter = strLst.iterator();
      while (iter.hasNext()) {
         String str = (String)iter.next(); // need to explicitly downcast Object back to String
         System.out.println(str);
      }
      strLst.add(new Integer(1234));       // Compiler/runtime cannot detect this error
      String str = (String)strLst.get(3);  // compile ok, but runtime ClassCastException
   }
}
```

- ArrayList trong pre-JDK 1.5 được thiết kế để lữu giữ java.lang.Object
- Bất kỳ lớp con nào của Object có thể được thay thế cho Object.
- Lớp Object là gốc của mọi lớp => bất kỳ đối tượng nào đều có thể được lưu trữ trong ArrayList

2. Generic classes

- Khái niệm generics cho phép trừu tượng hoá kiểu dữ liệu đối số
- LTV có thể:
  - Thiết kế một lớp với kiểu generic
  - Cung cấp thông tin cụ thể về kiểu khi khởi tạo các đối tượng của lớp
- Compiler có thể kiểm tra kiểu lúc biên dịch và đảm bảo rằng không tồn tại các lỗi chuyển kiểu khi thực thi ứng dụng

Ví dụ: Cài đặt MyGenericArrayList với generics

```
// A dynamically allocated array with generics
public class MyGenericArrayList<E> {
   private int size;     // number of elements
   private Object[] elements;

   public MyGenericArrayList() {  // constructor
      elements = new Object[10];  // allocate initial capacity of 10
      size = 0;
   }

   public void add(E e) {
      if (size < elements.length) {
         elements[size] = e;
      } else {
         // allocate a larger array and add the element, omitted
      }
      ++size;
   }

   public E get(int index) {
      if (index >= size)
         throw new IndexOutOfBoundsException("Index: " + index + ", Size: " + size);
      return (E)elements[index];
   }

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

E is replaced with Object
Check e is of type E when invoked

E is replaced with Object
Insert downcast operator E<E> for the return type.

Ví dụ: Cài đặt MyGenericArrayList

```
public class MyGenericArrayListTest {
   public static void main(String[] args) {
      // type safe to hold a list of Strings
      MyGenericArrayList<String> strLst = new MyGenericArrayList<String>();

      strLst.add("alpha");   // compiler checks if argument is of type String
      strLst.add("beta");

      for (int i = 0; i < strLst.size(); ++i) {
         String str = strLst.get(i);   // compiler inserts the downcasting operator (String)
         System.out.println(str);
      }

      strLst.add(new Integer(1234));  // compiler detected argument is NOT String, issues compilation error
   }
}
```

- **Lỗi chuyển kiểu được xác định ngay khi biên dịch!**
- **Compiler đảm bảo các phương thức vận hành đúng với kiểu dữ liệu truyền vào**

Ví dụ: Cài đặt ArrayList trong JCF

<E> : tham số kiểu dữ liệu hình thức

```
public class ArrayList<E> implements List<E> .... {
   // Constructor
   public ArraList() { ..... }

   // Public methods
   public boolean add(E e) { ..... }
   public void add(int index, E element) { ..... }
   public boolean addAll(int index, Collection<? extends E> c)
   public abstract E get(int index) { ..... }
   public E remove(int index)
   .......
}
```

```
ArrayList<Integer> lst1 = new ArrayList<Integer>(); // E substituted with Integer
lst1.add(0, new Integer(88));
lst1.get(0);

ArrayList<String> lst2 = new ArrayList<String>();   // E substituted with String
lst2.add(0, "Hello");
lst2.get(0);
```

Quy ước tên tham số kiểu hình thức

- <E> : một phần tử của tập hợp
- <T> : kiểu
- <K,V> : key (khoá) & value (giá trị)

3. Generic method

- Tương tự generic classes, generic method có thể được định nghĩa với generic types.

```
public static <E> void ArrayToArrayList(E[] a, ArrayList<E> lst) {
   for (E e : a) lst.add(e);
}
```

4. Wildcard

- Unbounded wildcard <?>
  - ? : any unknown type

```
public static void printList(List<?> lst) {
  for (Object o : lst) System.out.println(o);
}
```

- Upperbound wildcard <? extends type>
  - <? extends type> : type & its sub-types

```
public static void printList(List<? extends Number> lst) {
  for (Object o : lst) System.out.println(o);
}
```

- Lowerbound wildcard <? super type>
  - <? super type> : type & its super-types