Môn: Lập trình Hướng đối tượng (Object Oriented Programming)

Chương 6. Lập trình Generics

Nội dung

6.1. Khái niệm về Generics
6.2. Mục đích của Generics
6.3. Generics ở mức Lớp
6.4. Generics ở mức phương thức
6.5. Sử dụng Wildcards trong Generics
6.6. Generics và xử lý ngoại lệ (Exception)
6.7. Thừa kế và Generics

6.1. Khái niệm về Generics

- Các kiểu và method generic là các tính năng mới của Java 5.
- Một ưu điểm mà người ta thường nhắc tới chính là dùng generic có thể hạn chế được các lỗi trong ép kiểu.
- Collections Framework, một trong những gói được cài đặt generic nhiều nhất ở Java 5.
- Ví dụ: kiểu java.util.List<E> là một kiểu generic: một danh sách chứa các phần tử của một kiểu nào đó được thể hiện bởi nơi giữ chỗ E. Kiểu này có một method tên add(), định nghĩa nhận một đối số kiểu E, và một method tên get(), định nghĩa để trả lại một giá trị kiểu E.

6.1. Khái niệm về Generics (tt)

- Khi xác định các kiểu thực sự cho biến kiểu (hay các biến), tạo một kiểu tham số hoá chẳng hạn List<String>.
  - Lý do để xác định thông tin về kiểu bổ sung này nhằm giúp trình biên dịch có thể cung cấp việc kiểm tra kiểu chặt chẽ vào thời điểm biên dịch, tăng sự an toàn kiểu cho chương trình.
  - Việc kiểm tra kiểu này ngăn chặn việc thêm 1 đối tượng khác String vào List. Ngoài ra, còn cho phép trình biên dịch ép kiểu giúp. Trình biên dịch biết rằng method get( ) của một List<String> trả lại một đối tượng kiểu String.

6.2. Mục đích của Generics

- Phương pháp chỉ ra kiểu của các "Đối tượng" mà một Lớp có thể "chấp nhận"
- Phát hiện sớm các kiểu dữ liệu không phù hợp tại thời điểm biên dịch chương trình.
- Cho phép tham số là kiểu dữ liệu
- Tham số khác nhau nhưng vẫn dùng chung mã lệnh

6.2. Mục đích của Generics (tt)

- Không có generics, việc sử dụng các tập hợp collection đòi hỏi lập trình viên phải nhớ kiểu phần tử của mỗi collection.
- Khi tạo một collection trong Java 1.4, người lập trình cần biết kiểu của các object sẽ lưu trong collection đó, nhưng trình biên dịch không biết kiểu dữ liệu nào → phải cẩn thận trong việc thêm các phần tử có kiểu tương ứng.
- Khi truy vấn các phần tử từ một collection, người lập trình phải viết rõ ràng việc ép kiểu để chuyển các phần tử từ Object về kiểu thực của chúng.

→ Các kiểu generic giải quyết vấn đề an toàn kiểu.

6.2. Mục đích của Generics (tt)

- Vấn đề:

```
6       public static void main(String[] args)
7       {
8           ArrayList list = new ArrayList();
9
10          // you do this
11          list.add(new Integer(10));
12          list.add(new Integer(15));
13
14          // your colleague do this
15          list.add(new String("10"));
16
17          // I'd like to find element has an integer value of 15?
18          for (int i = 0; i < list.size(); i++)
19          {
20              Integer e = (Integer) list.get(i);
21              if (e.intValue() == 15)
22              {
23                  System.out.println("Get it at " + i);
24              }
25          }
26
27          System.out.println("Thank you!");
28      }
```

```
Exception in thread "main" java.lang.ClassCastException: java.lang.String
        at BoxDemo1.main(BoxDemo1.java:20)
Get it at 1
```

6.2. Mục đích của Generics (tt)

- Giải pháp:

```
6       public static void main(String[] args)
7       {
8           ArrayList list = new ArrayList();
9
10          // you do this
11          list.add(new Integer(10));
12          list.add(new Integer(15));
13
14          // your colleague do this
15          list.add(new String("10"));
16
17          // I'd like to find element has an integer value of 15?
18          for (int i = 0; i < list.size(); i++)
19          {
20              Object e = list.get(i);
21              if (e instanceof Integer)
22              {
23                  Integer item = (Integer) e;
24                  if (item.intValue() == 15)
25                  {
26                      System.out.println("Get it at: " + i);
27                  }
28              }
29          }
30
31          System.out.println("Thank you!");
32      }
```

Output

```
Get it at: 1
Thank you!
```

6.2. Mục đích của Generics (tt)

- Xem xét 2 ví dụ sau:

```
public class DemoNonGenerics
{
    public static void main(String[] args)
    {
       // This list is intended to hold only strings.
       // The compiler doesn't know that so we have to remember ourselves.
       List wordlist = new ArrayList();

       // Oops! We added a String[] instead of a String.
       // The compiler doesn't know that this is an error.
       wordlist.add(args);

       // Since List can hold arbitrary objects
       // Object.  Since the list is intended t
       // return value to String but get a Clas
       // the error above.
       String word = (String)wordlist.get(0);
    }
}
```

```
class DemoGenenics
{
    public static void main(String[] args)
    {
       // This list can only hold String objects
       List<String> wordlist = new ArrayList<String>();

       // args is a String[], not String, so the compiler won't let us do this
       wordlist.add(args);  // Compilation error!

       // We can do this, though.
       // Notice the use of the new for/in looping statement
       for(String arg : args) wordlist.add(arg);

       // No cast is required.  List<String>.get() returns a String.
       String word = wordlist.get(0);
    }
}
```

6.3. Generics ở mức Lớp

- Lớp Generic là một cơ chế để chỉ rõ mối quan hệ giữa Lớp và kiểu dữ liệu liên quan đến nó (type parameter).
- "Các Tham số kiểu" sẽ được xác định tại thời điểm đối tượng của Lớp được tạo
- Quy ước về tên của Tham số kiểu (Type Parameter Naming Conventions)
  - Viết hoa, dùng một chữ cái.
    - E – Element
    - K – Key
    - N – Number
    - **T – Type**
    - V – Value

6.3. Generics ở mức Lớp (tt)

- Tạo lớp Generic

```
public class NumberList<T>
{
    private T obj;

    public void add(T value)
    {
        this.obj = value;
    }

    public T get()
    {
        return obj;
    }
}
```

```
public void testNumberList_String()
{
    NumberList<String> list = new NumberList<String>();
    list.add("Hello");

    System.out.println(list.get());
}

public void testNumberList_Integer()
{
    NumberList<Integer> list = new NumberList<Integer>();
    list.add(new Integer(10));
    // list.add("10"); //Error

    System.out.println(list.get().intValue());
}
```

6.3. Generics ở mức Lớp (tt)

- Đa tham số kiểu cho một lớp

```
interface Pair<K, V>
{
    public K getKey();
    public V getValue();
}

public class OrderedPair<K, V> implements Pair<K, V>
{
    private K key;
    private V value;

    public OrderedPair(K key, V value)
    {
        this.key = key;
        this.value = value;
    }
    public K getKey(){ return key; }
    public V getValue() { return value; }
}
```

- Sử dụng

```
public static void main(String[] args)
{
    Pair<String, Integer> p1 = new OrderedPair<String, Integer>("Even", 8);
    Pair<String, String> p2 = new OrderedPair<String, String>("hello", "world");
}
```

6.4. Generics ở mức phương thức

- Generic ở mức phương thức là phạm vi của kiểu dữ liệu giới hạn trong một phương thức.
- Cú pháp:
  - Các “tham số kiểu” được khai báo trong phạm vi của phương thức.
  - Tham số kiểu phải được chỉ rõ trước kiểu dữ liệu trả về của phương thức và đặt trong cặp dấu <>.

- Có thể dùng tham số kiểu cho:
  - Các tham số của phương thức
  - Dữ liệu trả về
  - Biến cục bộ

6.4. Generics ở mức phương thức (tt)

```
public class GenericMethods
{
    public static <E> void printArray(E[] inputArray)
    {
        for (E element : inputArray)
        {
            System.out.printf("%s ", element);
        }
    }
}
```

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

    Double[] doubleArray = { 1.1, 2.2, 3.3, 4.4, 5.5, 6.6, 7.7 };
    Character[] characterArray = { 'H', 'E', 'L', 'L', 'O' };

    System.out.println("\nArray doubleArray contains:");
    printArray(doubleArray);

    System.out.println("");

    System.out.println("\nArray characterArray contains:");
    printArray(characterArray);
}
```

Output

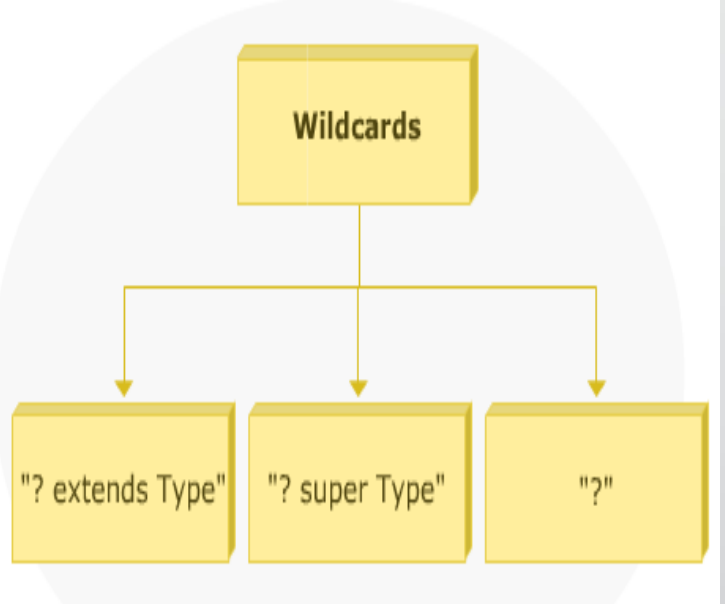
```
Array doubleArray contains:
1.1 2.2 3.3 4.4 5.5 6.6 7.7
Array characterArray contains:
H E L L O
```

6.5. Sử dụng Wildcards trong Generics

- Trong lập trình generic, ký tự "?" đại diện cho kiểu chưa biết
- Wildcards được dùng cho vài tình huống:
  - kiểu tham số,
  - kiểu trường,
  - kiểu biến cục bộ,
  - kiểu trả về.

6.5. Sử dụng Wildcards trong Generics (tt)

- "?"
  - Đại diện cho một kiểu chưa xác định.
- "? extends Type"
  - Đại diện cho một kiểu là lớp con của lớp được chỉ ra hoặc chính nó.
  - e.g. List <? extends Number>
- "? super Type"
  - Đại diện cho một kiểu là lớp cha của lớp được chỉ ra hoặc chính nó.
  - e.g. List <? super Number>



6.5. Sử dụng Wildcards trong Generics (tt)

- "?" bounded type/unbounded type

```
List<?> list = null;

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

- ? extends type" upper bounded wildcard

```
List<? extends Date> dateList = null;

dateList = new ArrayList<Date>();
dateList = new ArrayList<MyDate>(); // class MyDate extends Da
```

- "? super type" lower bounded wildcard

```
List<? super MyDate> dateList1 = null;

dateList1 = new ArrayList<Date>();
dateList1 = new ArrayList<MyDate>(); // class MyDate extends Date
```

6.5. Sử dụng Wildcards trong Generics (tt)

- Ví dụ: ?

```
public class Box<T>
{
    private T t;
    public void add(T t)
    {
        this.t = t;
    }
    public T get()
    {
        return t;
    }
    public <U extends Number> void inspect(U u)
    {
        System.out.println("T: " + t.getClass().getName());
        System.out.println("U: " + u.getClass().getName());
    }
    public static void main(String[] args)
    {
        Box<Integer> integerBox = new Box<Integer>();
        integerBox.add(new Integer(10));
        integerBox.inspect(10);
```

6.5. Sử dụng Wildcards trong Generics (tt)

- Ví dụ ? extends Type

- *? Là kiểu Number hoặc kiểu con của Number*

```
import java.util.ArrayList;
import java.util.List;
public class UpperBoundedDemo
{
    public static void main(String[] args)
    {
        List<Integer> ints = new ArrayList<>();
        ints.add(3);
        ints.add(5);
        ints.add(10);
        double sum = sum(ints);
        System.out.println("Sum of ints = "+sum);
    }

    public static double sum(List<? extends Number> list)
    {
        double sum = 0;
        for(Number n : list)
        {
            sum += n.doubleValue();
        }
        return sum;
    }
}
```

6.5. Sử dụng Wildcards trong Generics (tt)

- Ví dụ ? supper Type

- Ký tự "?" được dùng để kết với kiểu của lớp cha

```
public static void addNumbers(List<? super Integer>
list)
{
   for (int i = 1; i <= 10; i++)
   {
      list.add(i);
   }
}
```

6.6. Generics và xử lý ngoại lệ (Exception)

- Tham số kiểu cũng được dùng trong việc đưa ra (throw) các ngoại lệ.

```
public interface Command<X extends Exception>
{
    public void doit(String arg) throws X;
}
```

```
public class ExTest implements Command<IOException>
{
    public void doit(String filename) throws IOException {
        FileWriter fw = new FileWriter(filename);
        fw.write("hello world");
        fw.close();
    }
}
```

6.7. Thừa kế và Generics

- Một Lớp có thể thừa kế từ một Lớp Generic, và chỉ rõ kiểu của Generic, nếu không lớp con này phải khai báo như một lớp Generic

```
public class MyClass2<T> extends MyClass1<T>
{
    //OK
}
```

```
public class MyClass2<T> extends T
{
    //ERROR
}
```

- Một "Lớp" chỉ được hiện thực một trường hợp cụ thể "Giao tiếp generic" (Generic Interface)

```
class MyList implements MyCollection<Integer>
{
    // OK
}

class MyList implements MyCollection<Integer>, MyCollection<Double>
{
    // ERROR
}
```

6.7. Thừa kế và Generics (tt)

```
public class GenericClass<T>
{
    private T obj;

    public GenericClass(T obj)
    {
        this.obj = obj;
    }

    public void addT(T value)
    {
        this.obj = value;
    }

    public T getT()
    {
        return obj;
    }
}
```

```
public class InheritanceGenericClass<T, V> extends GenericClass<T>
{
    private V valObj;

    public InheritanceGenericClass(V vObj, T tObj)
    {
        super(tObj);
        this.valObj = vObj;
    }

    public void addV(V value)
    {
        this.valObj = value;
    }

    public V getV()
    {
        return this.valObj;
    }
}
```

```
public void testInheritanceGenericClass()
{
    InheritanceGenericClass<String, Integer> in;
    in = new InheritanceGenericClass<String, Integer>("Hello", 87);

    System.out.println(in.getT());
    System.out.println(in.getV());
}
```

```
Hello
87
```