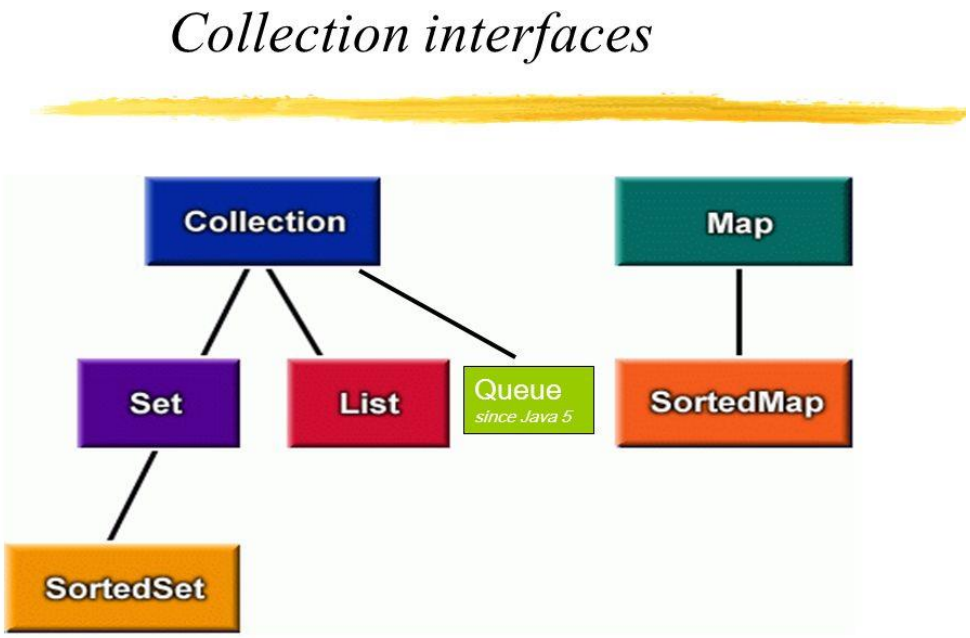
DEVMASTER.VN

Bài 7
Generic – Collections

Mục tiêu

- Giới thiệu Generics
- Collections
- List và ArrayList
- Set và SortedSet
- Map và HashMap



4

Giới thiệu Generics

- Java cung cấp **nhiều kiểu dữ liệu** để lập trình.
- Nhu cầu xây dựng phương thức, chức năng, lớp có **tính chất tương tự nhau** nhưng trên các kiểu dữ liệu khác nhau.
- **Generics** giúp cho lập trình viên có thể chỉ định loại đối tượng mà một lớp có thể làm việc thông qua tham số truyền vào tại thời điểm khai báo.
- Có thể sử dụng với **các loại kiểu** đối số.

Giới thiệu Generics

Ưu điểm:

- Generics cho phép **linh hoạt** và ràng buộc động.
- Có thể nhóm các chức năng/đối tượng **tương tự**.
- Mã nguồn đơn giản.

Nhược điểm:

- Không thể tạo **contructor**.
- **Biến cục bộ** không thể khai báo với khóa vào kiểu giá trị khác biệt cho mỗi tình huống khác nhau.

Giới thiệu Generics

Generic method (phương thức): khai báo một phương thức độc lập có khả năng nhận các tham số với **kiểu dữ liệu** khác nhau.

- Tham số của phương thức có **kiểu** cần khai báo đặt trong cặp dấu ngoặc móc (**<** và **>**).
- Có thể khai báo **nhiều tham số** phân cách bởi dấu **,**.
- **Phần thân viết tương tự** như phương thức truyền thống.

Generic method

Code: tạo class thông thường

```
12  public class DemoGeneric {
13
14      /**
15       * @param args the command line arguments
16       */
17      public static void main(String[] args) {
18          // TODO code application logic here
19      }
20
21  }
22
```

Generic method

Code: viết phương thức với E là kiểu dữ liệu đại diện

```
14      // Phương thức printArray
15      // in dữ liệu từ mảng với bất kỳ dữ liệu nào
16      public static < E> void printArray(E[] inputArray) {
17          // Display array elements
18          for (E element : inputArray) {
19              System.out.printf("%s ", element);
20          }
21          System.out.println();
22      }
```

Generic method

Code: từ hàm main gọi tới phương thức generic trên và truyền tham số

```
27      public static void main(String[] args) {
28          // Tạo mảng Integer, Double và Character
29          Integer[] intArray = {1, 2, 3, 4, 5};
30          Double[] doubleArray = {1.1, 2.2, 3.3, 4.4};
31          Character[] charArray = {'H', 'E', 'L', 'L', 'O'};
32
33          System.out.println("Array integerArray contains:");
34          printArray(intArray);   // Truyền vào mảng Integer
35
36          System.out.println("\nArray doubleArray contains:");
37          printArray(doubleArray);   // Truyền vào mảng Double
38
39          System.out.println("\nArray characterArray contains:");
40          printArray(charArray);   // Truyền vào mảng Character
41      }
```

Generic method

Kết quả chạy mã nguồn trên:

Output

```
Array integerArray contains:
1 2 3 4 5

Array doubleArray contains:
1.1 2.2 3.3 4.4

Array characterArray contains:
H E L L O
```

Giới thiệu Generics

- **Generic Class**(lớp): khai báo tương tự như class thông thường ngoại trừ theo sau tên lớp là kiểu dữ liệu đại diện.
- Kiểu dữ liệu đại diện (E, T,….) cần đặt trong cặp dấu ngoặc móc (**<** và **>**).
- Tương tự Generic Method, Generic Class cũng có thể khai báo nhiều kiểu tham số (phân cách bằng dấu **,**).

Generic class

Code: tạo class như sau

```
public class DemoGenericClass<T> {

    private T bienKieuT;

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        // TODO code application logic here
    }

}
```

Generic class

Code: viết phương thức get/set

```
public class DemoGenericClass<T> {

    private T bienKieuT;

    public void add(T t) {
        this.bienKieuT = t;
    }

    public T get() {
        return bienKieuT;
    }

}
```

Generic class

Code: viết thân hàm main

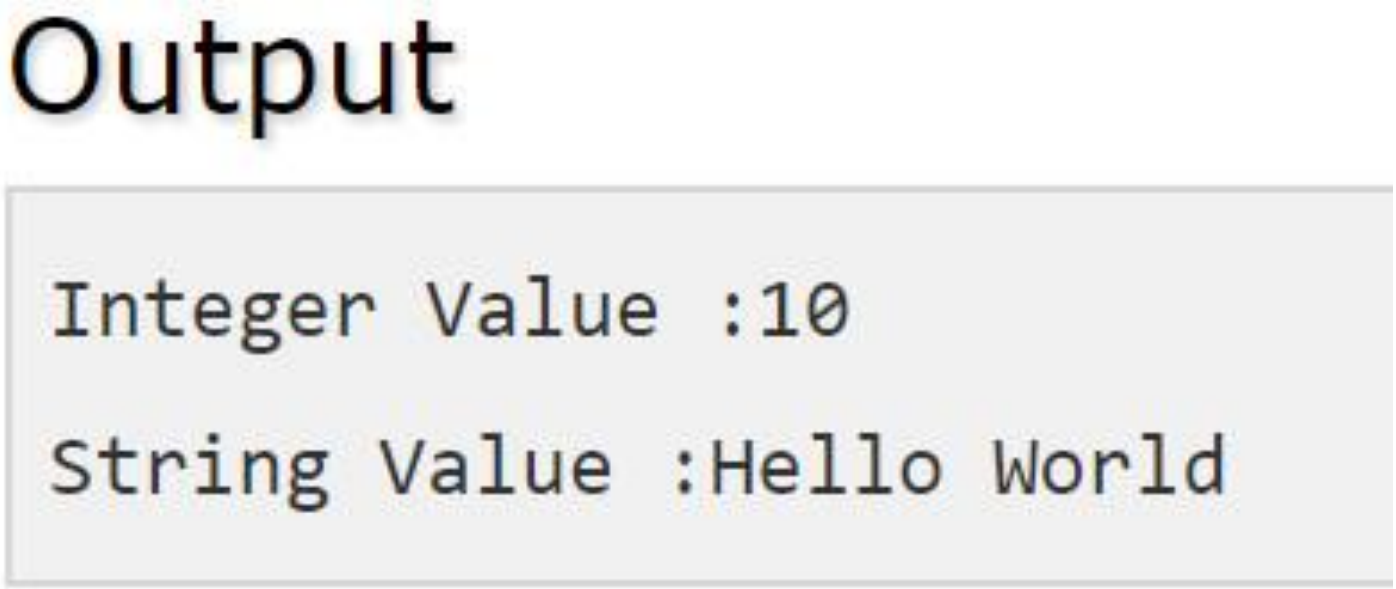
```
public static void main(String[] args) {
    DemoGenericClass<Integer> integerDemoGenericClass = new DemoGenericClass<Integer>();
    DemoGenericClass<String> stringDemoGenericClass = new DemoGenericClass<String>();

    integerDemoGenericClass.add(new Integer(10));
    stringDemoGenericClass.add(new String("Hello World"));

    System.out.printf("Integer Value :%d\n\n", integerDemoGenericClass.get());
    System.out.printf("String Value :%s\n", stringDemoGenericClass.get());
}
```

Generic class

Code: kết quả



Collections

- **Collection framework** bao gồm tập hợp các interface phục vụ thao tác tới dữ liệu dạng mảng.
- Nó được **đóng gói triển khai** với mục đích chung, cho phép thích ứng với các dạng mảng có dữ liệu khác nhau.
- Các interface, class của collection được đặt trong package: **java.util**.
- Giúp cho việc **nhóm nhiều phần tử** vào một đơn vị duy nhất.
- Làm cho việc lưu trữ, truy xuất, thao tác, truyền dữ liệu trở nên **đơn giản** hơn so với dùng mảng.

Collections

Collection **interface**

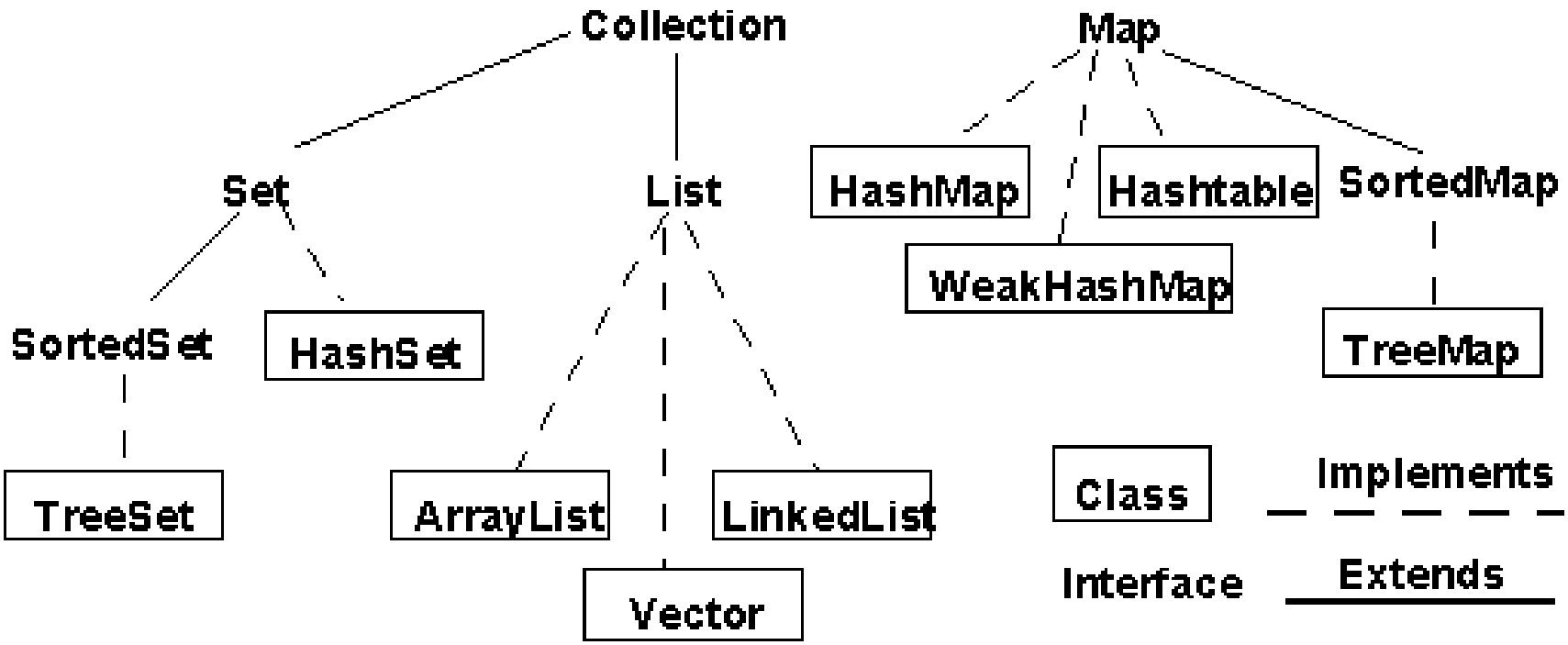
- List
- Set
- Queue

Collection **class**

- Hashset
- LinkedHashset
- TreeSet

Collections

Sơ đồ Collection



Collections

Để truy cập vào **Collection framework** có thể theo ½ cách sau:

- Sử dụng cấu trúc for-each.

Code Snippet

```
for (Object obj : collection)
System.out.println(obj);
```

- Sử dụng interface Iterator.

```
public interface Iterator<E> {
    boolean hasNext();
    E next();
    void remove(); //optional
}
```

List và ArrayList

- **Interface List** là một phần trong Collection.
- Nó cho phép thêm các đối tượng (**kể cả trùng lặp**) vào danh sách.
- List cho phép thêm phần tử vào **vị trí chỉ định**.
- List sử dụng **chỉ mục (index)** để xác định vị trí của phần tử (bắt đầu từ 0).
- List là tuần tự nên có thể truy cập bằng **iterator**.

List và ArrayList

Các phương thức trong List

- `add(int index, E element)`
- `addAll(int index, Collection<? extends E> c)`
- `get(int index)`
- `set(int index, E element)`
- `remove(int index)`
- `subList(int start, int end)`
- `indexOf(Object o)`
- `lastIndexOf(Object o)`

List và ArrayList

- Lớp **ArrayList** thực thi interface List.
- ArrayList là một mảng các phần tử với **kích thước có thể thay đổi**.
- Phần tử trong ArrayList có thể là **null**.
- ArrayList phù hợp với việc **truy cập ngẫu nhiên** vào phần tử bất kỳ.

List và ArrayList

ArrayList có các constructor như sau:

- `ArrayList()`
- `ArrayList(Collection <? extends E> c)`
- `ArrayList(int initialCapacity)`

Code:

Code Snippet

```
    . . .
List<String> listObj = new ArrayList<String> ();
System.out.println("The size is : " + listObj.size());
for (int ctr=1; ctr <= 10; ctr++)
{
        listObj.add("Value is : " + new Integer(ctr));
}
. . .
```

ArrayList có các phương thức:

- `add(E obj)`
- `trimToSize()`
- `ensureCapacity(int minCap)`
- `clear()`
- `contains(Object obj)`
- `size()`

Code Snippet

```
. . .
List<String> listObj = new ArrayList<String> ();
System.out.println("The size is : " + listObj.size());
for (int ctr=1; ctr <= 10; ctr++)
{
        listObj.add("Value is : " + new Integer(ctr));
}
listObj.set(5, "Hello World");
System.out.println("Value is: " +(String)listObj.get(5));
. . .
```

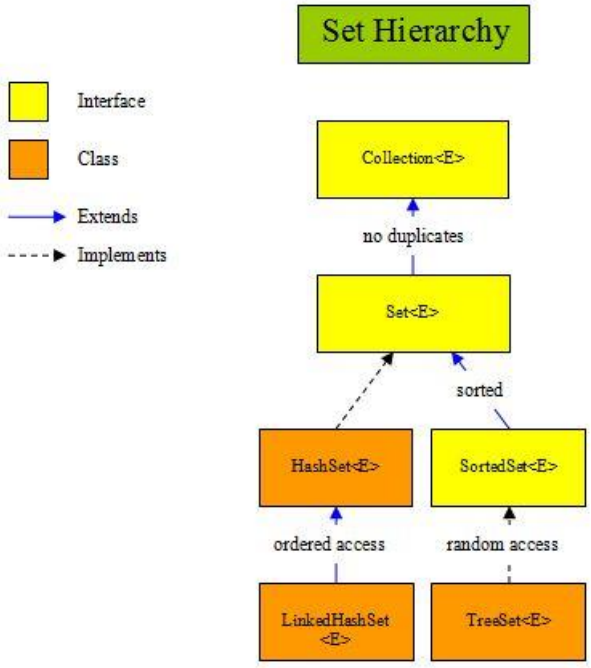
Set và SortedSet

- **Interface Set** tạo ra một danh sách các đối tượng không có thứ tự.
- Set không chứa **dữ liệu trùng lặp**.
- **Kế thừa đầy đủ** các phương thức từ interface Collection.
- Về cơ bản **Set tương tự List** ngoại trừ phương thức thêm phần tử **add()** không chấp nhận giá trị trùng lặp.

Set và SortedSet

Interface Set có các phương thức:

- `containsAll(Collection<?> obj)`
- `addAll(Collection<? extends E> obj)`
- `retainAll(Collection<?> obj)`
- `removeAll(Collection<?> obj)`



Set và SortedSet

- **Interface SortedSet** kế thừa interface Set và nó thực hiện sắp xếp thứ tự phần tử theo tăng dần.
- **Sắp xếp** có thể thực hiện tự động hoặc sử dụng **Comparator** khi tạo SortedSet.
- SortedSet được sử dụng khi muốn tạo ra danh sách các **phần tử không trùng lặp được sắp xếp**.

Map và HashMap

- **Map** là đối tượng lưu trữ dữ liệu dưới dạng mối quan hệ **KHÓA** và **GIÁ TRỊ**.
- Mỗi Khóa (key) sẽ nối với chỉ một giá trị (value) xác định.
- Khóa **không được trùng lặp** – phải là duy nhất.
- Map **không kế thừa** interface Collection.

Map và HashMap

Collection API có 3 cách tiếp cận với Map thông qua:

- `HashMap`
- `TreeMap`
- `LinkedHashMap`

Các phương thức:

- `put(K key, V value)`
- `get(Object key)`
- `containsKey(Object key)`
- `containsValue(Object value)`
- `size()`
- `values()`

Map và HashMap

Lớp **HashMap** thực thi interface Map và kế thừa tất cả phương thức của Map.

Constructor của HashMap:

- `HashMap()`
- `HashMap(int initialCapacity)`
- `HashMap(int initialCapacity, float loadFactor)`
- `HashMap(Map<? extends K,? extends V> m)`

Map và HashMap

Code:

Code Snippet

```
. . .
class EmployeeData
{
        public EmployeeData(String nm)
        {
                name = nm;
                salary = 5600;
        }
        public String toString()
        {
              return “[name=” + name + “, salary=” + salary + “]”;
```

Map và HashMap

Code:

```
public String toString()
{
return “[name=” + name + “, salary=” + salary + “]”;
}
. . .
}
public class MapTest
{
public static void main(String[] args)
{
Map<String, EmployeeData> staffObj = new HashMap<String,
EmployeeData>();
staffObj.put(“101”, new EmployeeData(“Anna John”));
```

Map và HashMap

Code:

```
        staffObj.put(“102”, new EmployeeData(“Harry Hacker”));
        staffObj.put(“103”, new EmployeeData(“Joby Martin”));
        System.out.println(staffObj);
        staffObj.remove(“103”);
        staffObj.put(“106”, new EmployeeData(“Joby Martin”));
        System.out.println(staffObj.get(“106”));
        System.out.println(staffObj);
        . . .
        }
}
```

Tóm tắt bài học

- ✓ **Generic** là cách thức lập trình Java cung cấp giúp việc xây dựng phương thức và lớp có thể nhận kiểu dữ liệu mà sẽ được định nghĩa trong quá trình thực thi.
- ✓ Generic được dùng để xây dựng Collection giúp tăng tính **mềm dẻo linh hoạt**.
- ✓ **List** là interface cho phép lưu danh sách các phần tử.
- ✓ **Set** là interface cho phép lưu danh sách các phần tử không trùng lặp
- ✓ **Map** là interface lưu trữ dữ liệu dưới dạng cặp Khóa-Giá trị

Thank for watching!