Lập trình ứng dụng Java

Generics & Collection

Nội dung

- ❏ Generics
- ❏ Collections

Nội dung

- **Generics**
- Collections

Generics

- **Generic** nghĩa là "**tổng quát**" hàm ý đưa ra những cách làm chung nhất cho nhiều vấn đề.
- Tình huống ví dụ cần xây dựng 1 Ngăn xếp (***Stack***) chứa kiểu dữ liệu ***int*** với các phương thức cơ bản như: ***push***, ***pop***, ***check***,… Sau đó lại cần xây dựng cũng ***Stack*** với các phương thức như vậy nhưng dữ liệu lại là ***String***
- Giải pháp cho vấn đề trên?

❑ Với int

```
public class IntStack {
   private int[] _data = new int[100];
   private int _curIdx = -1;

   public void Push(int value) {
      _data[++_curIdx] = value;
   }

   public int Pop() {
      return _data[_curIdx--];
   }
}
```

❑ Với String

```
public class StringStack {
   private String[] _data = new String[100];
   private int _curIdx = -1;

   public void Push(String value) {
      _data[++_curIdx] = value;
   }

   public String Pop() {
      return _data[_curIdx--];
   }
}
```

Giải pháp

- Cách đầu tiên thường làm: copy lại và thay đổi kiểu dữ liệu trong code.
  - Nếu có yêu cầu đổi kiểu dữ liệu sang ***float***, ***double***,…???
- Cách tốt hơn khi đã biết đến kiểu dữ liệu cơ sở trong Java là ***Object***.
  - Khi sử dụng kiểu dữ liệu ***Object*** thì ***Stack*** đã chấp nhận được mọi kiểu dữ liệu?

- Cài đặt với Object

```
public class ObjectStack {
   private Object[] _data = new Object[100];
   private int _curIdx = -1;

   public void Push(Object value) {
      _data[++_curIdx] = value;
   }

   public Object Pop() {
      return _data[_curIdx--];
   }
}
```

Vấn đề

- Vấn đề của **ObjectStack** kể trên là dễ gây ra lỗi ***runtime*** (tức là chỉ giải quyết tốt về mặt syntax)

```
public static void main(String[] args) {
    // TODO code application logic here
    ObjectStack s = new ObjectStack();
    s.Push(0);
    s.Push("test");
    int x = 8 + (int)s.Pop();
}
```

- Java cung cấp giải pháp cho vấn đề này thông qua khái niệm **Generic**.

Generic Types

- **Generic** là một khái niệm được đưa vào Java từ phiên bản 5
- Trong **Java**, **Generic** được giới thiệu là khái niệm về các kiểu tham số và được dùng để thiết kế ***class*** và ***method*** nhằm để trì hoãn chỉ ra kiểu dữ liệu cho đến khi chúng được khai báo hoặc khởi tạo.
- Một trong những điểm nổi bật của kiểu **Generic** là cho phép kiểm tra cú pháp trong lúc biên dịch. Có thể sử dụng nhiều kiểu dữ liệu khác nhau với cùng 1 đoạn code (tương tự như khái niệm **Template** trong **C++**)

Generic Class

- Lớp Generic là một cơ chế để chỉ rỏ mối quan hệ giữa Lớp và kiểu dữ liệu liên quan đến nó (type parameter).
- Lớp có thể có nhiều tham số. Ví dụ:

```
public class CBox<T1,T2,…,Tn> {…}
```

- Quy ước về tên tham số kiểu (Type Parameter Naming Conventions)
  - E – Element
  - K – Key
  - N – Number
  - **T – Type**
  - V – Value

Tham số T

- Bằng cách sử dụng tham số **T** là tham số chung, cho phép tạo 1 ***class*** duy nhất và sử dụng cho nhiều ***type*** khác nhau.

```
public class GenericStack<T> {
   private T[] _data = (T[]) new Object[100];
   private int _curIdx = -1;

   public void Push(T value) {
      _data[++_curIdx] = value;
   }

   public T Pop() {
      return _data[_curIdx--];
   }
}
```

```
public static void main(String[] args) {
   GenericStack<String> sS = new GenericStack<>();
   sS.Push("value 1");
   sS.Push(1); //error
}
```

Generic Methods

- Là các hàm thông dụng (cách hành xử chung cho các ***type*** khác nhau) nên cài đặt theo kiểu **Generic** (sử dụng các tham số đầu vào hay đầu ra là generic)
- 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:
  - Dữ liệu trả về
  - Các tham số của phương thức
  - Biến cục bộ

Ví dụ

```
static <T> void PrintArray(T[] arr){
    for (T element : arr)
        System.out.println(element);
}
```

```
public static void main(String[] args) {
    Integer[] arrInt = { 2, 5, 7, 8 };
    Character[] arrChar = { 'l', 't', 'j', 'v' };
    System.out.println("Print array:");
    PrintArray(arrInt);
    PrintArray(arrChar);
}
```

Khởi tạo đối tượng Generic

- Ví dụ

```
public class MyGeneric<T> {
    T t = new T(); //error
}
```

- Việc khởi tạo một đối tượng generic như trên là không được phép, vì <T> không hề tồn tại ở thời điểm chạy của **Java**. Nó chỉ có ý nghĩa với trình biên dịch kiểm soát code syntax. Mọi kiểu <T> đều như nhau nó được hiểu là **Object** tại thời điểm chạy của **Java**.
- Muốn khởi tạo đối tượng generic **<T>** bạn cần cung cấp cho Java đối tượng **Class<T>**, Java sẽ tạo đối tượng **<T>** tại thời điểm runtime bằng **Java Reflection**.

```
public class MyGeneric<T> {
   T t;

   public MyGeneric(Class<T> tClass)
         throws IllegalAccessException, InstantiationException,
         NoSuchMethodException, InvocationTargetException {
      t = tClass.getDeclaredConstructor().newInstance();
   }
}
```

```
public static void main(String[] args) throws Exception {
   MyGeneric<String> mg = new MyGeneric<String>(String.class);
   mg.t = "abc";
   System.out.println(mg.t);
}
```

Giới hạn tham số kiểu

```
static <T extends Number> double Calculate(T x, T y){
    return (x.doubleValue() + y.doubleValue());
}
```

```
public static void main(String[] args) throws Exception {
    double x = 8;
    int y = 10;
    String z = "test";
    System.out.println(Calculate(x, y));
    System.out.println(Calculate(x, z));
}
```

- Có thể giới hạn với nhiều kiểu
- Sử dụng “&”
- Nếu kiểu giới hạn thuộc kiểu class thì phải đặt trước
- Ví dụ:

Class A { /* ... */ }

interface B { /* ... */ }

interface C { /* ... */ }

class D <T extends A & B & C> { /* ... */ }

Generics với wildcard

- Sử dụng ký tự ? để khai báo tham số kiểu đại diện

```
public static void main(String[] args) throws Exception {
    GenericStack<?> gs = new GenericStack<String>();
    GenericStack<? super String> gsN = new GenericStack<String>();
    GenericStack<? extends Number> gsN = new GenericStack<String>();
}
```

- Khi sử dụng ký tự đại diện thì không sử dụng được phương thức có sử dụng tham số kiểu

```
public static void main(String[] args) throws Exception {
    GenericStack<?> gs = new GenericStack<String>();
    gs.Push("test"); //error
}
```

Ưu điểm của Generics

- Kiểu dữ liệu an toàn.
- Kiểm soát kiểu dữ liệu chặt chẽ trong biên dịch.
- Hạn chế việc ép kiểu không an toàn.
- Giúp dễ dàng hiện thực khung thuật toán, dễ dàng thay đổi, an toàn dữ liệu và dễ đọc.

Hạn chế khi sử dụng Generics

- ❏ Không thể sử dụng Primitive type.
- ❏ Không thể tạo đối tượng trực tiếp.
- ❏ Không thể sử dụng static với đối tượng Generics.
- ❏ Không thể sử dụng *instanceof*

Nội dung

- ❏ *Generics*
- ❏ **Collections**

Mảng – Ví dụ

```
public class Car {

}
```

```
Car[] cars1;
```
null

```
Car[] cars2 = new Car[5];
```
null	null	null	null	null

```
for(int i=0; i<5;i++){
    cars2[i]=new Car();
    // Xử lý khác
}
```

Mảng – Nhận xét

- Phải cho biết trước số lượng phần tử trong mảng
- Ko thể thay đổi kích thước về sau (mở rộng)
  - Khai báo mảng mới cars3
  - Copy dữ liệu qua mảng mới cars3
  - Cấp vùng nhớ mới cho cars2
  - Copy dữ liệu từ cars3 qua cars2
- Thêm phần tử x vào vị trí k
- Xóa phần tử x

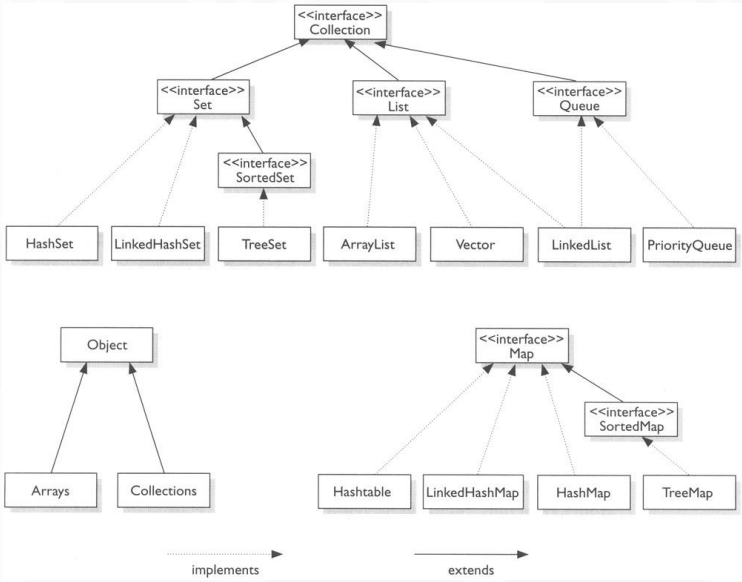
Collections

- Collection là đối tượng có khả năng chứa các đối tượng khác.
- Các thao tác thông thường trên collection
  - Khởi tạo collection
  - Thêm/Xoá đối tượng vào/khỏi collection
  - Kiểm tra một đối tượng có ở trong collection không
  - Lấy một đối tượng từ collection
  - Duyệt các đối tượng trong collection
  - Xoá toàn bộ collection

Collections Framework

- Một số lợi ích của Collections Framework:
  - Giảm thời gian lập trình
  - Tăng cường hiệu năng chương trình
  - Dễ mở rộng các collection mới
  - Khuyến khích việc sử dụng lại mã chương trình

Tổng quan Java Collections



- Collections Framework bao gồm:
  - Interfaces: Là các giao tiếp thể hiện tính chất của các kiểu collection khác nhau như List, Set, Map.
  - Implementations: Là các lớp collection có sẵn được cài đặt các collection interfaces.
  - Algorithms: Là các phương thức tĩnh để xử lý trên collection, ví dụ: sắp xếp danh sách, tìm phần tử lớn nhất...

Interface Collection

- Là thành phần cơ sở
- Không có lớp con được cài đặt sẵn, chỉ có interface con thông qua Set, List, Queue
- Cung cấp các thao tác chính trên collection:
  - boolean add(Object element);
  - boolean remove(Object element);
  - boolean contains(Object element);
  - int size();
  - boolean isEmpty();

Interface List

- List kế thừa từ Collection
- Cung cấp thêm các phương thức để xử lý collection kiểu danh sách (chứa các phần tử được xếp theo chỉ số).
  - Phần tử có thể trùng
- Một số phương thức của List
  - Object get(int index);
  - Object set(int index, Object o);
  - void add(int index, Object o);
  - Object remove(int index);
  - int indexOf(Object o);
  - int lastIndexOf(Object o);

Interface Set

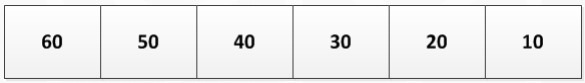
- Set kế thừa từ Collection
- Hỗ trợ các thao tác xử lý trên collection kiểu tập hợp (toán học)
  - Phần tử KHÔNG THỂ trùng
- Một số method riêng:
  - set1.containsAll(set2) // set2 là 1 **subset** của set1
  - set1.addAll(set2); // phép **hội**
  - set1.retainAll(set2); // phép **giao**
  - set1.removeAll(set2); // phép **trừ**

Interface SortedSet

- SortedSet kế thừa từ Set
- Nó hỗ trợ thao tác trên tập hợp các phần tử có thể so sánh được.
- Các đối tượng đưa vào trong một SortedSet phải cài đặt giao tiếp Comparable hoặc phải truyền vào lớp cài đặt SortedSet một Comparator
- Một số phương thức của SortedSet:
  - Object first();
  - Object last();
  - SortedSet headSet(Object end); // <= end
  - SortedSet tailSet(Object start); // >=start
  - SortedSet subSet(Object start, Object end);

Interface Queue

- Kế thừa interface Collection
- Hoạt động theo cơ chế FIFO
- Một số method riêng:
  - boolean offer(Object obj); // Thêm vào queue
  - Object poll(); // Xóa khỏi queue, trả về phần tử đầu
  - Object peek(); // Lấy phần tử đầu, ko xóa



Interface Map

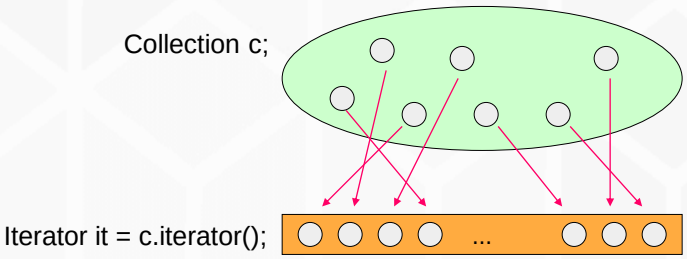
- Ko kế thừa interface Collection
- Quản lý các phần tử theo cơ chế key-value
  - Các Key không trùng nhau
- Một số method thông dụng:
  - Object put(Object key, Object value);
  - Object get(Object key);
  - Object remove(Object key);
  - boolean containsKey(Object key);
  - boolean containsValue(Object value);
  - Set keySet(); // Trả về các key
  - Collection values(); // Trả về các value
  - Set entrySet(); // Trả về các cặp key-value

Interface SortedMap

- Kế thừa interface Map
- Giống như SortedSet, các đối tượng key đưa vào trong SortedMap phải cài đặt giao tiếp Comparable hoặc lớp cài đặt SortedMap phải nhận một Comparator trên đối tượng key.
- Một số method riêng:
  - Object firstKey( );
  - Object lastKey( );
  - SortedMap headMap(Object end);
  - SortedMap tailMap(Object start);
  - SortedMap subMap(Object start, Object end);

Iterator Interface

- Các phần tử trong collection có thể được duyệt thông qua **Iterator (bộ duyệt)**.
- Các lớp cài đặt Collection cung cấp phương thức trả về iterator trên các phần tử của chúng.

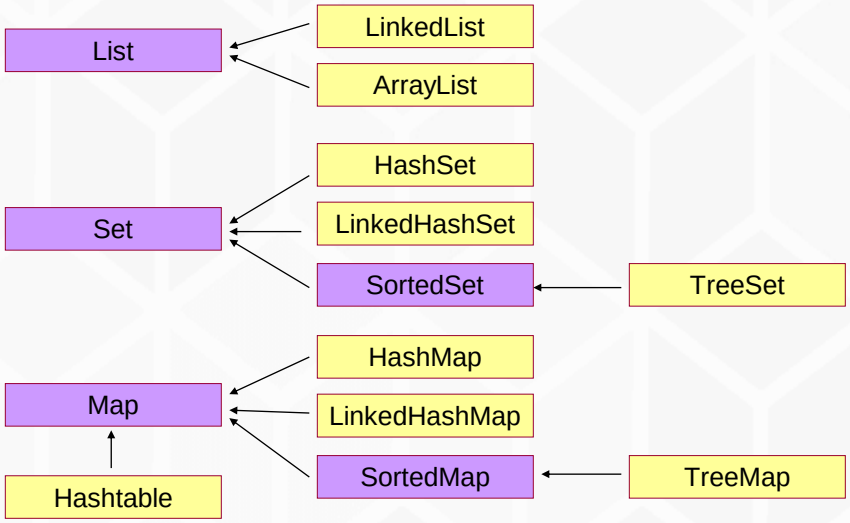


Duyệt Collection

- Iterator cho phép duyệt tuần tự một collection.
- Các phương thức của Iterator:
  - boolean hasNext();
  - Object next();
  - void remove();
- Ví dụ:

```
public static void main(String[] args) {
    ArrayList al = new ArrayList();
    Iterator it = al.iterator();
    while(it.hasNext()){
        Object obj = it.next();
        System.out.println(obj);
    }
}
```

Implementations



Ví dụ: SinhVien Set

```
public class SinhVien implements Comparable{
    private String mssv;
    private double diem;

    public SinhVien(String ms, double d) {
        mssv = ms;
        diem = d;
    }
    public double getDiem() {
        return diem;
    }
    public String toString() {
        return "(" + mssv + ", " + diem + ")";
    }
    @Override
    public int compareTo(Object o) {
        SinhVien sv2 = (SinhVien) o;
        return mssv.compareTo(sv2.mssv);
    }
}
```

❑ Áp dụng collection sorted

```
public static void main(String[] args) {
    SortedSet ssSV = new TreeSet();
    ssSV.add(new SinhVien("SV0001", 8.5));
    ssSV.add(new SinhVien("SV1201", 5.5));
    ssSV.add(new SinhVien("SV0401", 6.0));
    ssSV.add(new SinhVien("SV6001", 4.0));
    ssSV.add(new SinhVien("SV0041", 7.5));

    System.out.println(ssSV);
}
```

Bài tập

- Xem file bài tập