Lập trình JAVA cơ bản

Tuần 4

Giảng viên: Trần Đức Minh

Nội dung bài giảng

- Generics
- Package
- File .jar
- Final
- Map
- Set
- Bảng băm và phương thức hashCode()

Generics

- **Generics** (tham số hóa kiểu dữ liệu) là một khái niệm được đưa vào từ Java phiên bản 5. Generics cho phép ta **tạo và sử dụng lớp, interface, phương thức với nhiều kiểu dữ liệu khác nhau**.
- Nói cách khác, generics là cách thức lập trình tổng quát cho phép **một lớp hoạt động với nhiều kiểu dữ liệu khác nhau**.

Generics Class

- Ví dụ: Sử dụng lớp **GenericClass** với thuộc tính có các kiểu dữ liệu khác nhau

```
class GenericClass<T> {
    private T object;

    public T getObject() {
        return object;
    }

    public void setObject(T object) {
        this.object = object;
    }
}

public class MainClass {
    public static void main(String[] args) {
        GenericClass<String> genericClassString = new GenericClass<String>();
        genericClassString.setObject("Hello");
        System.out.println(genericClassString.getObject());

        GenericClass<Integer> genericClassInteger = new GenericClass<Integer>();
        genericClassInteger.setObject(89);
        System.out.println(genericClassInteger.getObject());
    }
}
```

Generics Interface

- Ví dụ: Cách sử dụng **Generics Interface**

```
interface GenericInterface<T> {
    void insert(T object);
    void delete(T object);
}

class GenericInterfaceImpl<T> implements GenericInterface<T> {
    public void insert(T object) {
        //...
    }

    public void delete(T object) {
        //...
    }
}
```

Generics method

- Ví dụ:

```java
import java.util.ArrayList;
import java.util.Collection;

public class MainClass {
    public static <T> int count(Collection<T> collection, T itemToCount) {
        int count = 0;
        for (T item : collection) {
            if (itemToCount.equals(item)) {
                count++;
            }
        }
        return count;
    }

    public static void main(String[] args) {
        ArrayList<Integer> arrayList = new ArrayList<Integer>();
        arrayList.add(3);
        arrayList.add(1);
        arrayList.add(2);
        arrayList.add(3);
        System.out.println(count(arrayList, itemToCount: 3));
    }
}
```

Quy ước đặt tên tham số

Ký tự	Ý nghĩa
E	Element - Phần tử
K	Key - Khóa
V	Value - Giá trị
T	Type - Kiểu dữ liệu
N	Number - Số

Quy ước đặt tên tham số

- Ví dụ:

```java
class KeyValuePair<K, V> {
    private K key;
    private V value;

    public KeyValuePair(K key, V value) {
        this.key = key;
        this.value = value;
    }

    public K getKey() {
        return key;
    }

    public void setKey(K key) {
        this.key = key;
    }

    public V getValue() {
        return value;
    }

    public void setValue(V value) {
        this.value = value;
    }
}
```

```java
public class MainClass {
    public static void main(String[] args) {
        KeyValuePair<String, Integer> entry = new KeyValuePair<String, Integer>("A21123", 123456789);
        String name = entry.getKey();
        Integer id = entry.getValue();
        System.out.println("Name = " + name + ", Id = " + id); // Name = gpcoder, Id = 123456789
    }
}
```

Kế thừa lớp Generics

- Khi một lớp kế thừa từ một lớp Generics, nó có thể **chỉ định rõ kiểu cho tham số** Generics; nó cũng có thể giữ nguyên hoặc thêm các tham số Generics.
- Ví dụ 1:

```
public class ContactEntry extends KeyValuePair<String, Integer> {
    public ContactEntry(String key, Integer value) {
        super(key, value);
    }
}
```

- Ví dụ 2:

```
public class ContactEntry2<V> extends KeyValuePair<String, V> {
    public ContactEntry2(String key, V value) {
        super(key, value);
    }
}
```

Kế thừa lớp Generics

- Ví dụ 3:

```
public class ContactEntry3<K, V> extends KeyValuePair<K, V> {
    public ContactEntry3(K key, V value) {
        super(key, value);
    }
}
```

- Ví dụ 4:

```
class ContactEntry4<K, V, T> extends KeyValuePair<K, V> {
    private T obj;

    public ContactEntry4(K key, V value, T obj) {
        super(key, value);
        this.obj = obj;
    }

    public T getObj() {
        return obj;
    }

    public void setObj(T obj) {
        this.obj = obj;
    }
}
```

Các ký tự đại diện Generics

- Ký tự đại diện **<?>**
  - Chấp nhận tất cả các loại đối số (chứa mọi kiểu đối tượng)
  - Ví dụ:
    - KeyValuePair<String, ?> pair = new ContactEntry3<String, Integer>("A21122", 9);
    - KeyValuePair<String, ?> pair = new ContactEntry3<String, String>("A21122", “Java”);

Các ký tự đại diện Generics

- Ký tự đại diện **<? extends type>**
  - Chấp nhận bất kỳ đối tượng nào miễn là đối tượng này **kế thừa từ type** hoặc là đối tượng của **type**.
  - Ví dụ: Danh sách chỉ chứa kiểu dữ liệu Number hoặc kiểu con của Number.
    - List<? extends Number> list = new ArrayList<Long>();
    - ~~List<? extends Number> list = new ArrayList<String>();~~

Các ký tự đại diện Generics

- Ký tự đại diện **<? super type>**
  - Chấp nhận bất kỳ đối tượng nào miễn là đối tượng này là **cha của type** hoặc đối tượng của **type**.
  - Ví dụ: Danh sách chỉ chứa kiểu dữ liệu Number hoặc kiểu con của Number.
    - ArrayList<? super Integer> cmp = new ArrayList<Number>();
    - ~~ArrayList<? super String> cmp = new ArrayList<Integer>();~~

Package

- Java sử dụng **package** nhằm tránh sự xung đột trong điều khiển truy cập lớp.
- Một package chứa một tập hợp các **lớp**, **interface** và các **package con**.
- Các lớp được định nghĩa trong pakage nào thì phần đầu file của lớp đó phải có dòng cú pháp sau:

  package <tên gói>
- Cấu trúc của gói được thể hiện trên đĩa vật lý **theo dạng thư mục**.
- Ví dụ:

```
package com.example.JavaThangLong;


public class MainClass {
}
```

Package

- Lợi ích của việc dùng package
  - Cung cấp bảo vệ truy cập cho các lớp.
  - Khắc phục được việc trùng tên giữa các lớp.
  - Dùng để phân loại các lớp và interface.
- Sử dụng package

  import <tên gói>
- Ví dụ:

  import com.example.JavaThangLong.MainClass;

  import com.example.JavaThangLong.*;

Phạm vi truy cập mặc định của class

- Một class A **không khai báo phạm vi** thì class đó được coi là có **phạm vi truy cập mặc định** (default access modifier) tức là các class **thuộc cùng package** với class A mới được phép truy cập đến class A.
- Ví dụ: Đoạn chương trình sau có lỗi

```
package com.libraries;

class MyClass {
    private int value;

    public MyClass() {
        this.value = 0;
    }
}
```

```
package com.example.lophocjava;

import com.libraries.MyClass;

public class MainClass {
    public static void main(String args[]) {
        MyClass myClass = new MyClass();
    }
}
```

File .jar

- **File .jar** (**J**ava **AR**chive) có định dạng theo **chuẩn nén ZIP** dùng để gộp nhiều file về thành chung một file, đồng thời nén dung lượng để giảm kích thước.
- Tác dụng của file .jar
  - Dùng để **lưu trữ, xử lý** các file âm thanh, hình ảnh và các file .class trong Java.
  - Thường được **sử dụng như các khối module hay thư viện** để tích hợp thêm vào ứng dụng hoặc các tiện ích khác.

File jar

- Ví dụ:
  - Xem phần hỗ trợ kỹ thuật
    - Cách tạo file .jar từ Intellij
    - Cách thêm thư viện mở rộng từ file .jar

Biến final

- Biến được sử dụng với từ khóa final sẽ được thể hiện ở các dạng sau
  - Dùng để **khai báo hằng số**

    private final int MAX = 100;
  - **Thuộc tính trống** (không được khởi tạo khi khai báo)

```
class HinhTron {
    private final double PI;
    public HinhTron() {
        PI = tinhSoPI(); // Buộc phải được gán giá trị ở tất cả cấu tử
    }
    private double tinhSoPI() {
    …
    }
}
```

Phương thức final

- Phương thức khi được sử dụng với từ khóa final thì nó **không thể nạp chồng** được.
- class XeMay {

```
    public final void run() {
        System.out.println("running");
    }
}
```

- class XeHonDa extends XeMay {

```
    ~~public void run()~~ {
        System.out.println("Bền máy");
    }
}
```

Lớp final

- Lớp khi được sử dụng với từ khóa final thì nó **không thể được kế thừa**.
- final class XeMay {

```
    public void run() {
        System.out.println("running");
    }
}
```

- ~~class XeHonDa extends XeMay~~ {

```
    public void run() {
        System.out.println("Bền máy");
    }
}
```

Interface Map

- **Map** là một interface được thiết kế để lưu trữ một tập hợp các cặp **key - value** (khóa - giá trị). Trong đó các **value** có thể giống nhau nhưng **key** bắt buộc phải khác nhau. Dựa vào **key**, ta có thể xác định các **values** tương ứng với nó.
- interface Map nằm trong gói **java.util.Map**

Key	Value
K1	AAA,BBB,CCC
K2	AAA,BBB
K3	AAA,DDD
K4	AAA,2,01/01/2015
K5	3,ZZZ,5623

Interface Map

- Chúng ta sử dụng Map khi muốn truy xuất, cập nhật hay tìm kiếm thông tin các phần tử thông qua khóa của phần tử đó.
- Ví dụ:
  - Một Map bao gồm thông tin của sinh viên và điểm các môn học của sinh viên đó. **Key** sẽ là mã của sinh viên và **Values** sẽ là danh sách điểm các môn học.
- Địa chỉ tra cứu
  - https://docs.oracle.com/javase/8/docs/api/java/util/Map.html

Lớp Map

- Các lớp được dùng để triển khai interface Map
  - **HashMap**: Thứ tự các phần tử không dựa theo thứ tự lúc thêm vào.
  - **LinkedHashMap**: Thứ tự các phần tử dựa theo thứ tự lúc thêm vào.
  - **TreeMap**: Thứ tự các phần tử được sắp xếp theo chiều tăng dần của các **Keys**.
- Ví dụ:
  - Map<Integer, String> hashMap = new HashMap<>();
  - Map<String, String> linkedHashMap = new LinkedHashMap<>();
  - Map<String, Integer> treeMap = new TreeMap<>();

Interface Map

- Đưa dữ liệu vào Map

  hashMap.**put**(15, “Lập trình Java”);

- Xóa một phần tử

  hashMap.**remove**(15);

- Thay thế một phần tử

  hashMap.**replace**(15, “Java cơ bản”);

- Sao chép một Map vào một Map khác

  Map<Integer, String> treeMap = new TreeMap<>();

  treeMap.**putAll**(hashMap);

Interface Map

- **Lấy toàn bộ dữ liệu của Map**
  - Cách 1:

```
for (Map.Entry<Interger, String> entry : mapObject.entrySet()) {
    System.out.println(entry.getKey() + " : " + entry.getValue());
}
```

  - Cách 2:

```
hashMap.forEach((keyInt, valueStr) -> {
    System.out.println(keyInt + " : " + valueStr);
});
```

- **Lấy toàn bộ keys**

```
for (Integer key : hashMap.keySet()) {
    System.out.println("Key = " + key);
}
```

- **Lấy toàn bộ values**

```
for (String value : hashMap.values()) {
    System.out.println("Value = " + value);
}
```

Ví dụ 1

- Xây dựng lớp SinhVien chứa các thuộc tính tên và điểm.
- Xây dựng lớp DanhSachSinhVien có thuộc tính listSinhVien là một đối tượng có kiểu dữ liệu là **Map**, trong đó:
  - key: là mã sinh viên
  - value là một đối tượng của lớp SinhVien
- Thực hiện các thao tác của lớp DanhSachSinhVien như sau:
  - Thêm một sinh viên
  - Xóa một sinh viên theo mã.
  - Sửa thông tin sinh viên dựa trên mã.
  - Liệt kê danh sách sinh viên.

Set interface

- **Set** là một interface được thiết kế để lưu trữ một danh sách các phần tử. Trong đó giá trị các phần tử phải khác nhau.
- interface Set nằm trong gói **java.util.Set**
- Địa chỉ tra cứu
  - https://docs.oracle.com/javase/7/docs/api/java/util/Set.html

Set interface

- Các lớp được dùng để triển khai interface Set
  - **HashSet**: Thứ tự các phần tử không dựa theo thứ tự lúc thêm vào.
  - **TreeSet**: Thứ tự các phần tử được sắp xếp theo chiều tăng dần.
- Ví dụ:
  - Set<Integer> hashSet = new HashSet<>();
  - Set<String> treeSet = new TreeSet<>();

Set interface

- Đưa dữ liệu vào Set

```
if (!hashSet.contains(28)) {
    hashSet.add(28);
}
```

- Xóa phần tử

```
hashSet.remove(28);
if (!hashSet.isEmpty()) {
    hashSet.clear();
}
```

- Đếm số phần tử trong Set
  - hashSet.**size**();
- Kiểm tra một Set có phải là tập con của Set khác hay không
  - hashSet.**containsAll**(subHashSet);

Set interface

- Hợp của hai tập hợp

```
Integer[] arraySet1 = {2, 10, 4, 8, 5};
Integer[] arraySet2 = {1, 6, 0};

List<Integer> list1 = Arrays.asList(arraySet1);
List<Integer> list2 = Arrays.asList(arraySet2);

Set<Integer> setInteger1 = new HashSet<>(list1);
Set<Integer> setInteger2 = new HashSet<>(list2);

setInteger1.addAll(setInteger2);
System.out.println(setInteger1);
```

Set interface

- Giao của hai tập hợp

```
Integer[] arraySet1 = {2, 10, 4, 8, 5};
Integer[] arraySet2 = {8, 12, 4};

List<Integer> list1 = Arrays.asList(arraySet1);
List<Integer> list2 = Arrays.asList(arraySet2);

Set<Integer> setInteger1 = new HashSet<>(list1);
Set<Integer> setInteger2 = new HashSet<>(list2);
// Loại bỏ các phần tử có trong setInteger1 nhưng không có trong
// setInteger2
setInteger1.retainAll(setInteger2);
System.out.println(setInteger1);
```

Set interface

- Hiệu của hai tập hợp

```
Integer[] arraySet1 = {2, 10, 4, 8, 5};
Integer[] arraySet2 = {8, 12, 4};

List<Integer> list1 = Arrays.asList(arraySet1);
List<Integer> list2 = Arrays.asList(arraySet2);

Set<Integer> setInteger1 = new HashSet<>(list1);
Set<Integer> setInteger2 = new HashSet<>(list2);
// Loại bỏ các phần tử có trong setInteger1 và cũng có trong
// setInteger2
setInteger1.removeAll(setInteger2);
System.out.println(setInteger1);
```

Set interface

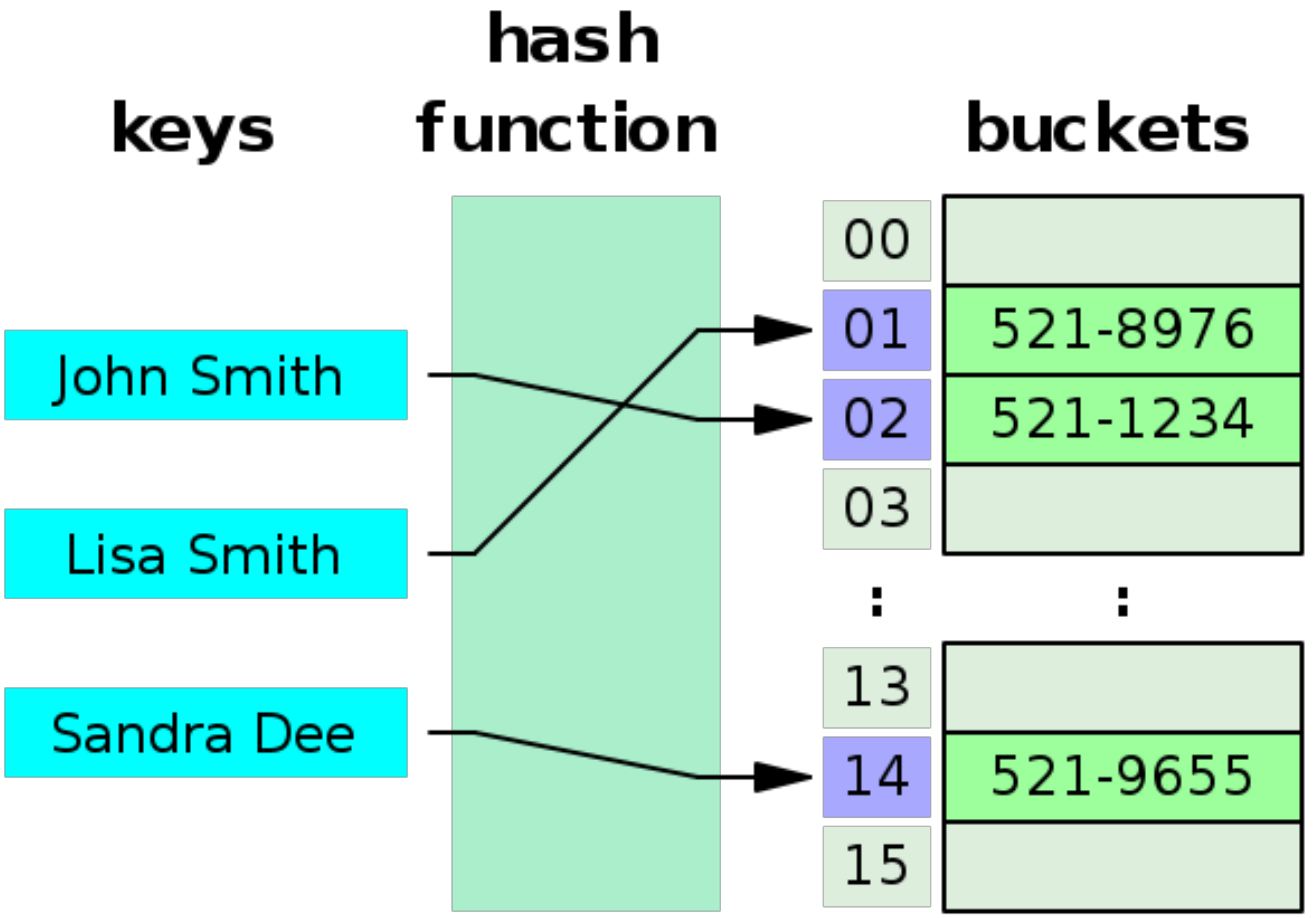
- Duyệt các phần tử trong Set với vòng lặp for cải tiến

```
for (Integer valueSet : hashSet) {
    System.out.println(valueSet);
}
```

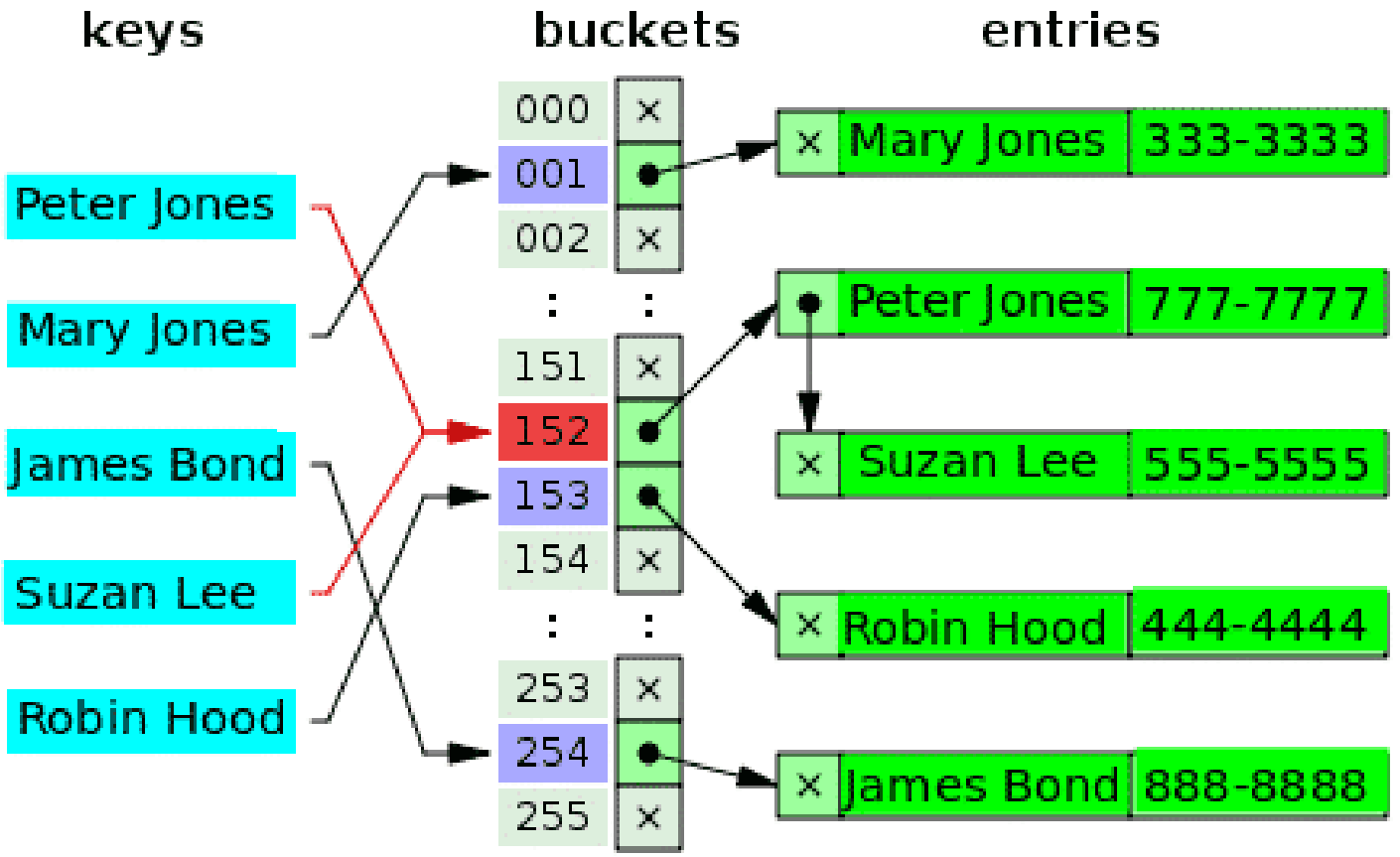
- Duyệt sử dụng Iterator

```
Iterator<Integer> iterator = hashSet.iterator();
while (iterator.hasNext()) {
    System.out.println(iterator.next());
}
```

Bảng băm


keys
hash function
buckets
John Smith
Lisa Smith
Sandra Dee
00
01
02
03
13
14
15
521-8976
521-1234
521-9655

Bảng băm


keys
buckets
entries
Peter Jones
Mary Jones
James Bond
Suzan Lee
Robin Hood
000
001
002
151
152
153
154
253
254
255
Mary Jones
333-3333
Peter Jones
777-7777
Suzan Lee
555-5555
Robin Hood
444-4444
James Bond
888-8888

Ví dụ 2

- Xây dựng lớp SinhVien chứa các thuộc tính mã sinh viên, tên và điểm.
- Xây dựng lớp DanhSachSinhVien có thuộc tính listSinhVien là một đối tượng có kiểu dữ liệu là **Set**
- Thực hiện các thao tác của lớp DanhSachSinhVien như sau:
  - Thêm một sinh viên
    - Không cho phép thêm thông tin sinh viên mới nếu mã sinh viên đã tồn tại trong danh sách.
  - Xóa một sinh viên theo mã.
  - Sửa thông tin sinh viên dựa trên mã.
  - Liệt kê danh sách sinh viên.

Ví dụ 2

- Chú ý nạp chồng 2 phương thức sau trong lớp SinhVien để hỗ trợ so sánh:
  - **public boolean equals(Object)**
    - So sánh đối tượng hiện tại với đối tượng nằm ở tham số.
  - **public int hashCode()**
    - Đây là phương thức dùng để phân biệt các đối tượng.
    - Hàm sẽ tạo ra một **mã băm** để hỗ trợ **phân biệt các đối tượng với nhau**. Mỗi đối tượng sẽ có một mã băm riêng.

Hết Tuần 4

Cảm ơn các bạn đã chú ý lắng nghe !!!