Lập trình JAVA cơ bản

Tuần 3

Giảng viên: Trần Đức Minh

Nội dung bài giảng

- Tính trừu tượng
- Interface
- Lớp trừu tượng
- Mở rộng Interface với phương thức default
- Inner Class
- Java Collections
  - Lớp ArrayList
  - Lớp LinkedList
  - Stack
  - Queue
  - Phương thức equals()

Tính trừu tượng

- **Tính trừu tượng** (abstract) là một đặc tính mà ẩn các cài đặt hay triển khai chi tiết và chỉ hiển thị tính năng đối với thành phần sử dụng.
- Tính trừu tượng giúp ta tập trung vào lớp thay vì quan tâm đến cách mà nó cài đặt thế nào.
- Trong Java có 2 cách để trừu tượng hóa:
  - Sử dụng **interface**.
  - Sử dụng **lớp abstract**.

Phương thức trừu tượng

- Một phương thức được **khai báo là abstract** và **không được cài đặt** thì đó là **phương thức trừu tượng** (abstract method).

  <phạm vi truy cập> **abstract** <giá trị trả về> <Tên phương thức>(<các đối số>);

- Ví dụ:
  - public **abstract** void printContent();

Interface

- **Interface** chứa một tập hợp các **phương thức trừu tượng**.
- Khi khai báo các phương thức bên trong Interface, ta **không cần dùng từ khóa abstract** bởi mặc nhiên chúng được coi là các phương thức trừu tượng.
- Khi một lớp **thực hiện** (implements) một interface thì lớp đó **bắt buộc phải nạp chồng toàn bộ các phương thức trừu tượng trong interface**.
- Một lớp có thể thực hiện nhiều interface cùng một lúc.
- Khai báo:

  interface <tên interface> {

  <các phương thức>

  }
- Ví dụ:

  interface ITinhLuong { … }

  public class NhanVien **implements** ITinhLuong { … }

Interface

- Phân tích đoạn code và chương trình sau:
  - Sử dụng Interface như đối số của phương thức

```
interface IHello {
    void showHello();
}

interface IGoodbye{
    void showGoodbye();
}

class Meeting implements IHello, IGoodbye {
    public void showHello() {
        System.out.println("Hello");
    }

    public void showGoodbye() {
        System.out.println("Goodbye");
    }
}
```

Chỉ gọi được phương thức showHello()

```
public class MainClass {
    public static void start(IHello hello) {
        hello.showHello();
    }

    public static void main(String[] args) {
        Meeting meeting = new Meeting();
        start(meeting);
    }
}
```

Interface

- Phân tích đoạn code và chương trình sau:
  - Khởi tạo một đối tượng cho Interface
    - Ta **không được phép tạo một đối tượng trực tiếp đến interface**.

```
interface IHello {
    void showHello();
}

interface IGoodbye{
    void showGoodbye();
}

class Meeting implements IHello, IGoodbye {
    public void showHello() {
        System.out.println("Hello");
    }

    public void showGoodbye() {
        System.out.println("Goodbye");
    }
}
```

Chỉ gọi được phương thức showGoodbye()

```
public class MainClass {
    public static void main(String[] args) {
        IGoodbye goodbye = new Meeting();
        goodbye.showGoodbye();
    }
}
```

Interface

- Interface được phép kế thừa

```
interface IGreeting {
    void showGreeting();
}

interface IHello extends IGreeting {
    void showHello();
}
```

  - Các lớp **implements từ IHello** sẽ phải nạp chồng cả hai phương thức **showGreeing()** và **showHello()**

Lớp trừu tượng

<phạm vi truy cập> **abstract class** <Tên lớp> {

…

}

- Lớp trừu tượng có thể chứa các phương thức **abstract** hoặc **không abstract**.
- Ta **không được phép tạo một đối tượng trực tiếp từ lớp abstract**.
- Một lớp kế thừa từ lớp trừu tượng không cần phải cài đặt cho các phương thức không phải abstract nhưng **bắt buộc phải nạp chồng các phương thức abstract** (trừ trường hợp lớp kế thừa cũng là một lớp trừu tượng)

Lớp trừu tượng

```
public abstract class Shape {
    private String color = “Red”;
    public abstract void draw();
    public String getColor() {
        return color;
    }
}

public class Rectangle extends Shape {
    public void draw() {
        System.out.println(“Draw “ + super.getColor() + “ rectangle”);
    }
}

public class MainClass {
    public static void main(String[] args) {
        Shape rect = new Rectangle();
        rect.draw();
    }
}
```

Sự khác nhau giữa lớp abstract và Interface

Lớp abstract	Interface
Lớp abstract có thể chứa cả phương thức abstract lẫn không phải phương thức abstract.	Interface chỉ chứa phương thức abstract.
Lớp abstract không hỗ trợ đa kế thừa.	Có thể sử dụng interface để hỗ trợ đa kế thừa.
Lớp abstract có thể chứa các biến final, non-final, static và non-static.	Interface chỉ chứa các biến static và final.
Lớp abstract có thể chứa nội dung cài đặt cho interface.	Interface không thể chứa nội dung cài đặt cho phương thức của lớp abstract.
Lớp abstract không phải trừu tượng hoàn toàn.	Về bản chất Interface là trừu tượng hoàn toàn.

Ta nên áp dụng lớp trừu tượng và Interface trong các bài toán có sử dụng tính đa hình.

Mở rộng interface

- Từ Java 8 trở đi bắt đầu thêm khái niệm **phương thức mặc định** (default method) và **phương thức tĩnh** (static method) đối với **interface**.
  - Đây là cách thức nhằm bổ sung hay cải tiến một interface đã tồn tại mà không làm ảnh hưởng đến những lớp đã implements interface trước đó.

Mở rộng interface

- Đặt vấn đề
  - Nếu một interface được implements đến nhiều lớp, thì khi ta thêm một phương thức trừu tượng đến interface này, ta cũng sẽ phải nạp chồng lên phương thức trừu tượng đó ở bên trong tất cả các lớp implements đến interface. Điều này gây ra vấn đề rất phức tạp khi số lượng lớp sử dụng interface lớn.
- **Phương thức mặc định**
  - Phương thức mặc định được sinh ra để giải quyết vấn đề này, giúp ta **bổ sung** những phát sinh của interface mà **không làm ảnh hưởng** đến các lớp implements đến interface.

Mở rộng interface

- Phương thức mặc định
  - Phương thức mặc định **bắt buộc** phải **triển khai code ngay bên trong interface**.
  - Để tạo phương thức mặc định ta sử dụng từ khóa **default**

```java
public interface MyInterface {
    int getValue();

    default int calculateValue() {
        return getValue() + 5;
    }
}
```

```java
public class MainClass implements MyInterface {
    @Override
    public int getValue() {
        return 8;
    }

    public static void main(String args[]) {
        MyInterface myInterface = new MainClass();
        System.out.println("Gia tri tinh toan la: " + myInterface.calculateValue());
    }
}
```

Mở rộng interface

- Phương thức mặc định
  - Nếu một lớp implements đến các interface **chứa các phương thức mặc định giống nhau** thì chương trình sẽ **sinh ra lỗi** do trình biên dịch không biết sẽ lấy phương thức mặc định ở interface nào để thực hiện.

```
public interface MyInterface {
    int getValue();

    default int calculateValue() {
        return getValue() + 5;
    }
}
```

```
public interface YourInterface {
    int getValue();

    default int calculateValue() {
        return getValue() * 15;
    }
}
```

Mở rộng interface

- Phương thức mặc định
  - Để khắc phục vấn đề **trùng tên phương thức mặc định của các interface**, ta thực hiện **nạp chồng lên phương thức mặc định** ở bên trong lớp implements đến các interface, trong đó nội dung được thực hiện theo một trong 2 cách sau:
    - **Cách 1**: Tạo ra nội dung mới hoàn toàn

```
public class MainClass implements MyInterface, YourInterface {
    @Override
    public int getValue() {
        return 8;
    }

    @Override
    public int calculateValue() {
        return getValue() + 50;
    }

    public static void main(String args[]) {
        MainClass mainClass = new MainClass();
        System.out.println(mainClass.calculateValue());
    }
}
```

Mở rộng interface

- Phương thức mặc định
  - Để khắc phục vấn đề **trùng tên phương thức mặc định của các interface**, ta thực hiện **nạp chồng lên phương thức mặc định** ở bên trong lớp implements đến các interface, trong đó nội dung được thực hiện theo một trong 2 cách sau:
    - **Cách 2**: Sử dụng từ khóa **super** để gọi đến phương thức mặc định của Interface mà ta muốn thực thi.

```
public class MainClass implements MyInterface, YourInterface {
    @Override
    public int getValue() {
        return 8;
    }

    @Override
    public int calculateValue() {
        return YourInterface.super.calculateValue();
    }

    public static void main(String args[]) {
        MainClass mainClass = new MainClass();
        System.out.println(mainClass.calculateValue());
    }
}
```

Mở rộng interface

- **Phương thức tĩnh** bên trong interface cũng giống với phương thức mặc định. Tuy nhiên chúng **không được phép nạp chồng** bên trong lớp implements đến interface.

```java
public interface MyInterface {
    int getValue();

    static int calculateValue() {
        return 5;
    }
}
```

```java
public class MainClass implements MyInterface {
    @Override
    public int getValue() { return 8; }

    @Override
    public int calculateValue() {
        return getValue() + 1;
    }

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

Inner class

- **Inner class** là một lớp được khai báo **bên trong một lớp** hoặc **một interface** hoặc **một phương thức**.
- Inner class được sử dụng để nhóm các lớp và các interface có cùng một xu hướng logic nào đó với nhau nhằm giúp code dễ đọc và dễ bảo trì hơn.
- Inner class có thể truy cập đến tất cả các thành viên của lớp bên ngoài bao gồm cả các thành viên có phạm vi **private**.
- Có 3 dạng Inner class chính:
  - Member inner class
  - Inner class vô danh
  - Local Inner class

Inner class

- **Member inner class**: Là một lớp được tạo ra ở bên trong một lớp nhưng nằm bên ngoài các phương thức.

```java
public class OuterClass {
    private int intValue = 68;

    class InnerClass {
        public void showData() {
            System.out.println("Integer value is: " + intValue);
        }
    }

    public static void main(String[] args) {
        OuterClass outerClass = new OuterClass();
        OuterClass.InnerClass innerClass = outerClass.new InnerClass();
        innerClass.showData();
    }
}
```

Inner class

- **Inner class vô danh**: có thể được tạo ra bằng lớp hoặc interface rồi nạp chồng lên phương thức bên trong lớp hoặc interface.
- Ví dụ trường hợp với lớp:

```
class InnerClass {
    public int intValue = 68;

    public void showData() {
        System.out.println(intValue);
    }
}

public class OuterClass {
    public static void main(String[] args) {
        InnerClass innerClass = new InnerClass() {
            public void showData() {
                System.out.println("Integer value is: " + intValue);
            }
        };
        innerClass.showData();
    }
}
```

Inner class

- Ví dụ trường hợp với interface:
  - Ý nghĩa: Tạo ra một Inner class vô danh implements đến interface IInnerClass. Do đó Inner class vô danh này bắt buộc phải nạp chồng lên phương thức showData().

```
interface IInnerClass {
    public static int intValue = 68;
    public void showData();
}

public class OuterClass {
    public static void main(String[] args) {
        IInnerClass innerClass = new IInnerClass() {
            public void showData() {
                System.out.println("Integer value is: " + intValue);
            }
        };
        innerClass.showData();
    }
}
```

Inner class

- **Local inner class**: Là một lớp được tạo **bên trong một phương thức**.
  - Nếu ta muốn gọi phương thức của một lớp được khai báo bên trong một phương thức, ta phải tạo ra đối tượng của lớp này bên trong phương thức chứa nó.

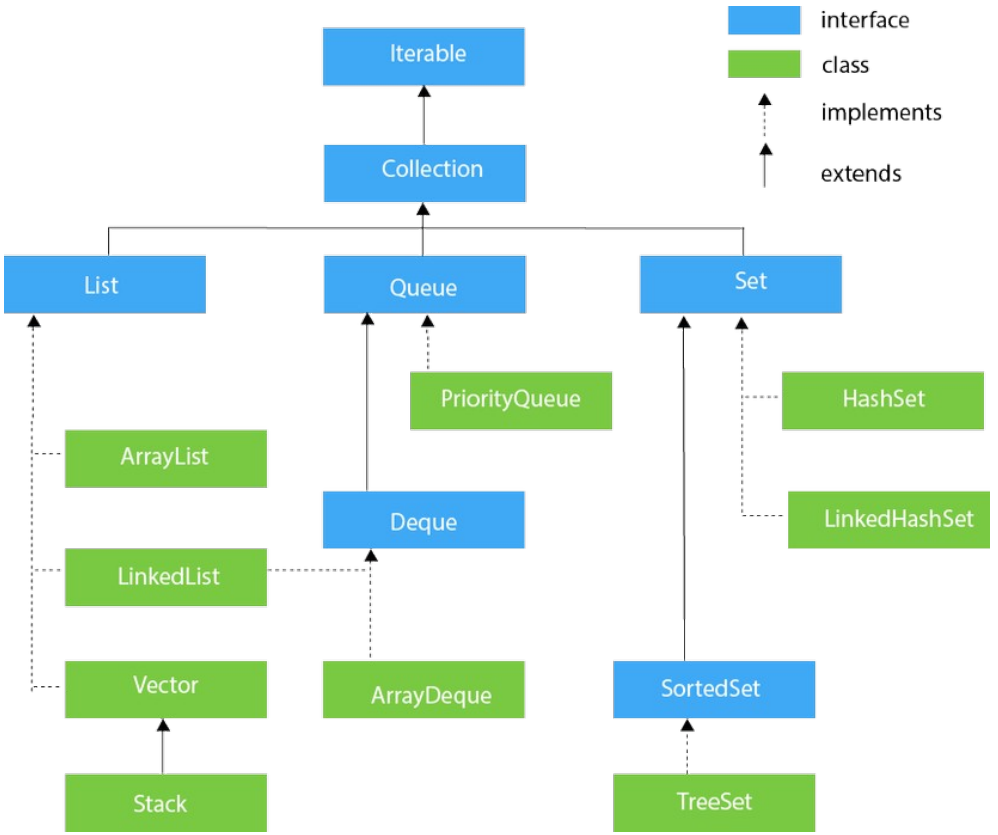
```
class OuterClass {
    private int intValue = 68;
    public void display() {
        int localValue = 99;

        class InnerClass {
            public void showData() {
                System.out.println("Instance value is " + intValue + " and local value is " + localValue);
            }
        }
        InnerClass innerClass = new InnerClass();
        innerClass.showData();
    }

    public static void main(String[] args) {
        OuterClass outerClass = new OuterClass();
        outerClass.display();
    }
}
```

Java Collections

- **Java Collections** là một framework cung cấp một kiến trúc để **lưu trữ và thao tác trên một nhóm các đối tượng**.
- Java Collections thực hiện hầu hết **các thao tác liên quan đến dữ liệu** như tìm kiếm, sắp xếp, chèn, xóa, …

Cấu trúc phân cấp của Java Collection

- Gói **java.util** là nơi chứa **toàn bộ các lớp và interface** liên quan đến Collection.



Lớp ArrayList

- Lớp **ArrayList** kế thừa từ lớp **AbstractList** và thực hiện (implements) **List interface**. ArrayList lưu trữ các phần tử như một mảng động, tức là số lượng phần tử của ArrayList có thể co dãn chứ không cố định như Mảng.
- Lớp ArrayList nằm trong gói **java.util.ArrayList**
- Từ phiên bản JDK 1.5 trở lại đây các lớp dạng **Collection** trong Java là **Generic** (tức là ta cần xác định rõ kiểu của các phần tử)
- Địa chỉ tra cứu:
  - https://docs.oracle.com/javase/7/docs/api/java/util/ArrayList.html

Lớp ArrayList

- Một số phương thức cần quan tâm
  - **boolean add(E element)**: Thêm một phần tử vào cuối danh sách.
  - **void add(int index, E element)**: Chèn một phần tử vào vị trí có chỉ số là **index**.
  - **E remove(int index)**: Lấy ra và xóa phần tử tại vị trí **index**.
  - **void clear()**: Xóa toàn bộ các phần tử trong danh sách.
  - **boolean isEmpty()**: Trả về true nếu danh sách rỗng.
  - **int size()**: Trả về sống lượng phần tử bên trong danh sách.
  - **E get(int index)**: Trả về phần tử ở vị trí **index**.
  - **E set(int index, E element)**: Thay thế phần tử tại vị trí **index** bằng phần tử **element**. Hàm trả về phần tử nằm ở vị trí **index** trước khi thay thế.

Lớp ArrayList

- **ArrayList**<kiểu dữ liệu> <tên ArrayList> = **new ArrayList**<kiểu dữ liệu>();
- Ví dụ:

```
ArrayList<String> arrListTen = new ArrayList<String>();
ArrayList<SinhVien> arrListSinhVien = new ArrayList<SinhVien>(35);
```

- Hiển thị các phần tử:

```
System.out.println(arrListTen);
```

- Duyệt các phần tử với vòng lặp **for cơ bản**:

```
for(int i = 0; i < arrListSinhVien.size(); i++) {
    System.out.println(arrListSinhVien.get(i).getHoVaTen());
}
```

- Duyệt các phần tử với vòng lặp **for cải tiến**:

```
for(SinhVien sinhvien : arrListSinhVien) {
    System.out.println(sinhvien.getHoVaTen());
}
```

Lớp ArrayList

- Duyệt các phần tử với **Iterator** (là một interface trong Java nằm trong gói **java.util.Iterator**):

```
Iterator<SinhVien> iterator = arrListSinhVien.iterator();
while(iterator.hasNext()) {
    System.out.println(((SinhVien) iterator.next()).getHoVaTen());
}
```

- Duyệt các phần tử với **ListIterator** (là một interface trong Java nằm trong gói **java.util.ListIterator**):

```
ListIterator<SinhVien> iterator = arrListSinhVien.listIterator();
while(iterator.hasNext()) {
    System.out.println(((SinhVien) iterator.next()).getHoVaTen());
}
```

Lớp ArrayList

- Các kiểu dữ liệu nguyên thủy không được phép sử dụng với ArrayList mà thay vào đó ta phải sử dụng các **Wrapper class** để thay thế.
- Ví dụ:

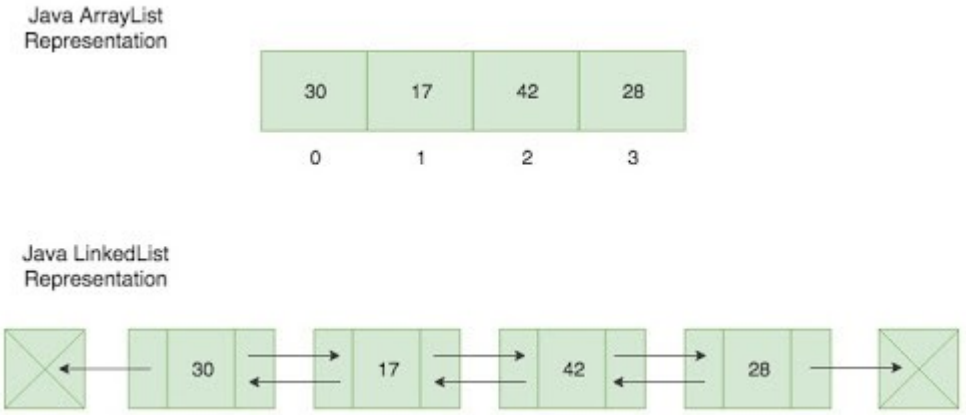
```
ArrayList<int> arr = new ArrayList<int>();
ArrayList<Integer> arr = new ArrayList<Integer>();
for (int i = 1; i <= 10; i++) {
    arr.add(Integer.valueOf(i * 2));
}

for (Integer intNumber : arr) {
    System.out.println(intNumber.intValue());
}
```

Ví dụ 1

- Sử dụng ArrayList thực hiện công việc sau:
  - Xây dựng lớp **Contact** chứa hai thuộc tính là số điện thoại và họ tên.
  - Xây dựng lớp **DanhBaDienThoai** có thuộc tính là một danh sách các Contact có kiểu là ArrayList
    - Xây dựng các thuộc tính thêm, bớt, sửa các contact trong danh bạ điện thoại.

Lớp LinkedList

- Lớp **LinkedList** kế thừa từ lớp **AbstractSequentialList** và thực hiện (implements) List interface. LinkedList lưu trữ các phần tử có thể không liên tục bên trong bộ nhớ và số lượng phần tử của LinkedList có thể co dãn chứ không cố định như Mảng.
- Lớp LinkedList nằm trong gói **java.util.LinkedList**



- Địa chỉ tra cứu:

  https://docs.oracle.com/javase/7/docs/api/java/util/LinkedList.html

Lớp LinkedList

- Một số phương thức cần quan tâm
  - **boolean add(E element)**: Thêm một phần tử vào cuối danh sách.
  - **void add(int index, E element)**: Chèn một phần tử vào vị trí có chỉ số là **index**.
  - **E remove()**: Lấy ra và xóa phần tử nằm ở đầu danh sách.
  - **E remove(int index)**: Lấy ra và xóa phần tử tại vị trí **index**.
  - **void clear()**: Xóa toàn bộ các phần tử trong danh sách.
  - **boolean isEmpty()**: Trả về true nếu danh sách rỗng.
  - **int size()**: Trả về sống lượng phần tử bên trong danh sách.
  - **E get(int index)**: Trả về phần tử ở vị trí **index**.
  - **E set(int index, E element)**: Thay thế phần tử tại vị trí **index** bằng phần tử **element**. Hàm trả về phần tử nằm ở vị trí **index** trước khi thay thế.

Lớp LinkedList

- **LinkedList**<kiểu dữ liệu> <tên LinkedList> = **new LinkedList**<kiểu dữ liệu>();
- Ví dụ:
  - LinkedList<Double> linkedListSoThuc = new LinkedList<Double>();
  - LinkedList<SinhVien> linkedListSinhVien = new LinkedList<SinhVien>();
- Các phương pháp duyệt phần tử trong LinkedList cũng giống như duyệt phần tử trong ArrayList đã trình bày.
- Các kiểu dữ liệu nguyên thủy không được phép sử dụng với LinkedList mà thay vào đó ta phải sử dụng các **Wrapper class** để thay thế.
  - ~~LinkedList<double> arrDouble = new LinkedList<double>()~~;
- Chuyển từ LinkedList sang mảng

```
Object[] objects = linkedListSinhVien.toArray();
for(int i = 0; i < objects.length; i++) {
    System.out.println(((SinhVien) objects[i]).getHoVaTen());
}
```

Sự khác nhau giữa ArrayList và LinkedList

ArrayList	LinkedList
ArrayList sử dụng **mảng động** để lưu trữ các phần tử.	LinkedList sử dụng **danh sách liên kết doubly** để lưu trữ các phần tử.
Thao tác ArrayList sẽ **chậm** do phải dồn mảng mỗi khi thực hiện thêm, bớt phần tử.	Thao tác LinkedList sẽ thực hiện **nhanh** hơn do không cần dồn danh sách mỗi khi thêm, bớt phần tử.
Sử dụng ArrayList sẽ tốt hơn trong việc lưu trữ và truy xuất dữ liệu. Do đó ta nên sử dụng ArrayList đối với dữ liệu **ít phải thao tác** trên đó.	LinkedList sẽ thực hiện tốt hơn khi thao tác với dữ liệu. Do đó ta nên sử dụng LinkedList đối với dữ liệu cần phải **thao tác nhiều** trên đó.

ArrayList và LinkedList

- Chuyển đổi từ ArrayList sang LinkedList:
  - ArrayList<SinhVien> arrayListSinhVien = new ArrayList<SinhVien>();

    ……
  - LinkedList<SinhVien> linkedListSinhVien = new LinkedList<SinhVien>(arrayListSinhVien);
- Chuyển đổi từ LinkedList sang ArrayList:
  - LinkedList<SinhVien> linkedListSinhVien = new LinkedList<SinhVien>();

    …….
  - ArrayList<SinhVien> arrayListSinhVien = new ArrayList<SinhVien>(linkedListSinhVien);

Ví dụ 2

- Thực hiện bài tập ở Ví dụ 1, nhưng thay thế việc sử dụng ArrayList bằng LinkedList.

Phương thức equals()

- Phương thức equals() là phương thức được định nghĩa sẵn bên trong lớp **java.lang.Object,** do đó tất cả các lớp bên trong Java đều có chứa phương thức này.
  - Được dùng để so sánh dữ liệu của 2 đối tượng với nhau (**trả về giá trị true nếu bằng nhau**)
  - **Khác với toán tử so sánh ==**
    - Toán tử so sánh == dùng để so sánh tham chiếu (địa chỉ bộ nhớ) của 2 đối tượng
    - Ví dụ:

```
String s1 = new String("Day ky tu");
String s2 = new String("Day ky tu");

System.out.println("s1 == s2: " + (s1 == s2));  // false
System.out.println("s1.equals(s2): " + (s1.equals(s2)));   // true
```

Phương thức equals()

- Đối với lớp do người dùng tự định nghĩa, ta cần phải nạp chồng phương thức equals() để các đối tượng của lớp có thể **so sánh với nhau theo một tiêu chí nào đó**.
- Ví dụ:

```
public class Student {
    private String id;
    private String name;
    // Hai đối tượng bằng nhau khi trùng cả id và name
    public boolean equals(Object obj) {
        if (obj instanceof Student) {
            Student another = (Student) obj;
            if (this.id.equals(another.id) && this.name.equals(another.name))
                return true;
        }
        return false;
    }
}
```

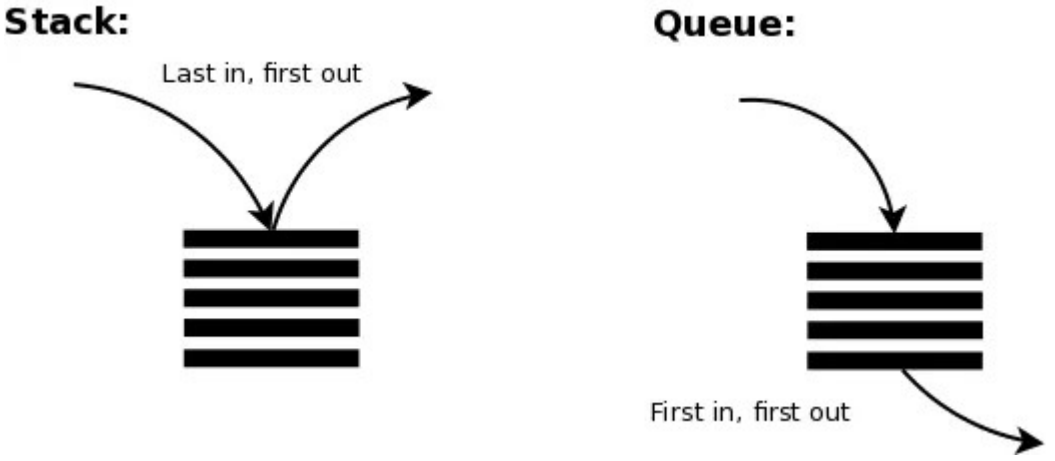
Phương thức equals()

- Áp dụng khi làm việc với Collection Framework
  - Một số phương thức của các lớp thuộc Collection Framework có thể chứa lời gọi đến phương thức equals() ở bên trong nó.
  - Ví dụ:

```
Student student1 = new Student("123", "Minh");
Student student2 = new Student("456", "Nam");

List<Student> listStudents = new ArrayList<>();
listStudents.add(student1);
listStudents.add(student2);

Student searchStudent1 = new Student("123", “Minh”);
Student searchStudent2 = new Student("789", “Linh”);
// Phương thức contains() có chứa phương thức equals() ở bên trong để so sánh
System.out.println("Search student 1: " + listStudents.contains(searchStudent1));  // true
System.out.println("Search student 2: " + listStudents.contains(searchStudent2));  // false
```

Lớp LinkedList

- Lớp LinkedList ngoài việc được sử dụng như List (danh sách), nó còn có thể sử dụng như **Stack** (ngăn xếp) hoặc **Queue** (hàng đợi) do có chứa một số phương thức liên quan hỗ trợ cho việc xây dựng Stack và Queue.



Xây dựng Stack với LinkedList

- Một số phương thức hỗ trợ cho việc xây dựng Stack
  - **void push(E)**: Thêm một phần tử vào đỉnh Stack.
  - **E pop()**: Lấy giá trị phần tử từ đỉnh Stack rồi loại bỏ nó.
  - **E peek()**: Lấy giá trị phần tử từ đỉnh Stack nhưng không loại bỏ phần tử đó.
  - **boolean isEmpty()**: Kiểm tra Stack có rỗng không.

Xây dựng Stack với LinkedList

- Ví dụ: Hãy cho biết thông tin in ra bởi đoạn chương trình sau:

```
LinkedList<Integer> stackList = new LinkedList<Integer>();

stackList.push( e: 5);
stackList.push( e: 6);
stackList.push( e: 7);

int topValue = stackList.pop();
System.out.println("Value in the top of Stack: " + topValue);

topValue = stackList.peek();
System.out.println("Value in the top of Stack: " + topValue);

for(int value : stackList) {
    System.out.println(value);
}
```

Xây dựng Queue với LinkedList

- Một số phương thức hỗ trợ cho việc xây dựng Queue
  - **boolean add(E element)**: Thêm một phần tử vào Queue.
  - **E poll()**: Lấy giá trị phần tử từ Queue rồi loại bỏ nó.
  - **E peek()**: Lấy giá trị phần tử từ Queue nhưng không loại bỏ phần tử đó.
  - **boolean isEmpty()**: Kiểm tra Queue có rỗng không.

Xây dựng Queue với LinkedList

- Ví dụ: Hãy cho biết thông tin in ra bởi đoạn chương trình sau:

```
LinkedList<Integer> queueList = new LinkedList<Integer>();

queueList.add(5);
queueList.add(6);
queueList.add(7);

int value = queueList.poll();
System.out.println("Value in the output of Queue: " + value);

value = queueList.peek();
System.out.println("Value in the output of Queue: " + value);

for(int v : queueList) {
    System.out.println(v);
}
```

Hết Tuần 3

Cảm ơn các bạn đã chú ý lắng nghe !!!