Java Core

1. **Java.util**
   1. **List**

**List** là một interface trong java. Nó chứa các phương thức để chèn và xóa các phần tử dựa trên chỉ số index.

Khởi tạo:

List<TYPE> NAME = **new** ArrayList<TYPE>();

Ví dụ:

List<String> newList = **new** ArrayList<String>();

**import** java.util.ArrayList;

**import** java.util.List;

**public** **class** Demo\_List {

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

List<String> newList = **new** ArrayList<String>();

newList.add("A");

newList.add("C");

newList.add(1,"B");

List<String> secondList = **new** ArrayList<String>();

secondList.add("D");

newList.addAll(2,secondList);

newList.forEach(newLists->

{

System.***out***.println(newLists);

});

}

}

* + 1. **Các thủ tục hỗ trợ:**

|  |  |
| --- | --- |
| **Method** | **Description** |
| void add(int index, E element) | Chèn phần tử được chỉ định vào vị trí đã chỉ định trong danh sách. |
| boolean add(E e) | Nối phần tử được chỉ định vào cuối danh sách. |
| boolean  addAll(Collection<? extends E> c) | Nối tất cả các thành phần trong Collection đã chỉ định vào cuối danh sách. |
| boolean  addAll(int index, Collection<? extends E> c) | Nối thêm tất cả các thành phần trong bộ sưu tập đã chỉ định, bắt đầu từ vị trí đã chỉ định của danh sách. |
| void clear() | Sử dụng để loại bỏ tất cả các yếu tố khỏi danh sách này. |
| boolean equals(Object o) | Sử dụng để so sánh đối tượng được chỉ định với các thành phần của danh sách. |
| int hashcode() | Sử dụng để trả về giá trị mã băm cho một danh sách. |
| E get(int index) | Sử dụng để lấy phần tử từ vị trí cụ thể của danh sách. |
| boolean isEmpty() | Trả về true nếu danh sách trống, ngược lại là false. |
| int lastIndexOf(Object o) | Trả về chỉ mục trong danh sách này về lần xuất hiện cuối cùng của phần tử đã chỉ định hoặc -1 nếu danh sách không chứa phần tử này. |
| Object[] toArray() | Trả về một mảng chứa tất cả các phần tử trong danh sách này theo đúng thứ tự. |
| T[] toArray(T[] a) | Trả về một mảng chứa tất cả các phần tử trong danh sách này theo đúng thứ tự. |
| boolean contains(Object o) | Trả về true nếu danh sách chứa phần tử đã chỉ định. |
| boolean containsAll(Collection<?> c) | Trả về true nếu danh sách chứa tất cả phần tử đã chỉ định |
| int indexOf(Object o) | Trả về chỉ mục trong danh sách này về lần xuất hiện đầu tiên của phần tử đã chỉ định hoặc -1 nếu Danh sách không chứa phần tử này. |
| E remove(int index) | loại bỏ phần tử có mặt tại vị trí đã chỉ định trong danh sách. |
| boolean remove(Object o) | Loại bỏ sự xuất hiện đầu tiên của phần tử được chỉ định. |
| boolean removeAll(Collection<?> c) | Loại bỏ tất cả các yếu tố khỏi danh sách. |
| void replaceAll(UnaryOperator operator) | Thay thế tất cả các yếu tố từ danh sách bằng các yếu tố được chỉ định. |
| void retainAll(Collection<?> c) | Giữ lại tất cả các yếu tố trong danh sách có trong bộ sưu tập được chỉ định. |
| E set(int index, E element) | Thay thế phần tử được chỉ định trong danh sách, có mặt tại vị trí đã chỉ định. |
| void sort(Comparator<? super E> c) | Sắp xếp các thành phần của danh sách trên cơ sở so sánh được chỉ định. |
| Spliterator spliterator() | Tạo spliterator trên các phần tử trong danh sách. |
| List<E> subList(int fromIndex, int toIndex) | Tìm nạp tất cả các yếu tố nằm trong phạm vi nhất định. |
| int size() | Trả về số lượng phần tử có trong danh sách. |

* 1. **Map, HashMap, HashTable:**
     1. **Map:**

Trong java, map được sử dụng để lưu trữ và truy xuất dữ liệu theo cặp key và value. Mỗi cặp key và value được gọi là mục nhập (entry). Map trong java chỉ chứa các giá trị key duy nhất. Map rất hữu ích nếu bạn phải tìm kiếm, cập nhật hoặc xóa các phần tử trên dựa vào các key.

Map<K, V> Name = **new** HashMap<K, V>();

Map<Integer, String> newMap = **new** HashMap<Integer, String>();

**import** java.util.\*;

**import** java.util.Map.Entry;

**public** **class** Demo\_Map {

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

// Create new map

Map<Integer, String> newMap = **new** HashMap<Integer, String>();

// Put element to map at 0

newMap.put(0, "First Element");

Set<Entry<Integer, String>> set=newMap.entrySet();

Iterator<Entry<Integer, String>> itr=set.iterator();

**while**(itr.hasNext()){

Entry<Integer, String> entry=itr.next();

System.***out***.println(entry.getKey()+" "+entry.getValue());

}

}

}

**Các thủ tục hỗ trợ:**

|  |  |
| --- | --- |
| **Method** | **Description** |
| V put(Object key, Object value) | Được sử dụng để chèn một mục trong bản đồ. |
| void putAll(Map map) | Được sử dụng để chèn Map được chỉ định trong Map. |
| V putIfAbsent(K key, V value) | Chỉ chèn giá trị được chỉ định bằng khóa được chỉ định trong Map nếu chưa được chỉ định. |
| V remove(Object key) | Được sử dụng để xóa một mục nhập cho khóa được chỉ định. |
| boolean remove(Object key, Object value) | Loại bỏ các giá trị được chỉ định bằng các khóa được chỉ định liên quan khỏi Map. |
| Set keySet() | Trả về khung nhìn Set chứa tất cả các phím. |
| Set<Map.Entry<K,V>> entrySet() | Trả về khung nhìn Set chứa tất cả các khóa và giá trị. |
| void clear() | Được sử dụng để thiết lập lại Map. |
| boolean containsKey(Object key) | Phương thức này trả về true nếu một số khóa bằng với khóa tồn tại trong Map, một số khác trả về false. |
| boolean equals(Object o) | Được sử dụng để so sánh Đối tượng được chỉ định với Map. |
| void forEach(BiConsumer<? super K,? super V> action) | Thực hiện hành động đã cho cho từng mục trong Map cho đến khi tất cả các mục đã được xử lý hoặc hành động đưa ra một ngoại lệ. |
| V get(Object key) | Phương thức này trả về đối tượng chứa giá trị được liên kết với khóa. |
| V getOrDefault(Object key, V defaultValue) | Trả về giá trị mà khóa được chỉ định được ánh xạ hoặc defaultValue nếu Map không chứa ánh xạ cho khóa. |
| int hashCode() | Trả về giá trị mã băm cho Map |
| boolean isEmpty() | Phương thức này trả về true nếu Map trống; trả về false nếu chứa ít nhất một khóa. |
| V replace(K key, V value) | Thay thế giá trị được chỉ định cho một khóa được chỉ định. |
| boolean replace(K key, V oldValue, V newValue) | Thay thế giá trị cũ bằng giá trị mới cho một khóa được chỉ định. |
| void replaceAll(BiFunction<? super K,? super V,? extends V> function) | Thay thế giá trị của mỗi mục nhập bằng kết quả của việc gọi hàm đã cho trên mục đó cho đến khi tất cả các mục đã được xử lý hoặc hàm ném ngoại lệ. |
| Collection values() | Trả về chế độ xem bộ sưu tập các giá trị có trong Map. |
| int size() | Phương pháp này trả về số lượng mục trong Map. |

* + 1. **HashMap**

Những điểm quan trọng về lớp HashMap trong java là:

* HashMap lưu trữ dữ liệu dưới dạng cặp key và value.
* Nó chứa các key duy nhất.
* Nó có thể có 1 key là null và nhiều giá trị null.
* Nó duy trì các phần tử KHÔNG theo thứ tự.

Sự khác nhau giữa HashSet và HashMap

* Câu trả lời là: HashSet chỉ chứa giá trị (value) trong khi HashMap chứa cặp key và value.

HashMap<K, V> Name = **new** HashMap<K, V>();

HashMap<Integer, String> Name = **new** HashMap<Integer, String>();

import java.util.HashMap;

import java.util.Set;

public class HashMapExample1 {

    public static void main(String args[]) {

        // init hashMap

        HashMap<Integer, String> hashMap = new HashMap<Integer, String>();

        // add elements to hashMap

        hashMap.put(1, "Java");

        hashMap.put(3, "C++");

        hashMap.put(2, "PHP");

        hashMap.put(4, "Python");

        // remove element

        hashMap.remove(2);

    }

}

**Duyệt HashMap:**

Có 2 cách để lặp 1 map: Giả sử có [HashMap](https://howtocodevn.wordpress.com/2015/11/04/java-map/) sau

|  |
| --- |
| HashMap<String, String> mapDatas = new HashMap<>();  mapDatas.put("first", “A”);  mapDatas.put("second", "B"); |

Lấy ra tập key rồi duyệt dựa trên nó:

|  |
| --- |
| for (String key : mapDatas.keySet()) {      System.out.println("key: " + key + " value: " + mapDatas.get(key));  } |

Sử dụng EntrySet và vòng lặp for hoặc [iterator](https://howtocodevn.wordpress.com/2015/11/03/iterator-va-listiterator/)

|  |
| --- |
| Set<Map.Entry<String, String>> entrySet = mapDatas.entrySet();  for (Entry entry : entrySet) {      System.out.println("key: " + entry.getKey() + " value: " + entry.getValue());  } |

* + 1. **HashTable**

Những điểm quan trọng về lớp Hashtable trong java là:

* Hashtable là một mảng của list. Mỗi list được biết đến như một xô chứa các phần tử. Ví trí của một xô được xác định bằng việc gọi phương thức hashcode(). Hashtable cũng lưu trữ dữ liệu dưới dạng cặp key và value.
* Nó chứa các key duy nhất.
* Nó KHÔNG thể có bất kỳ key hoặc giá trị nào là null.
* Nó được đồng bộ.

Sự khác nhau giữa HashMap và Hashtable:

|  |  |
| --- | --- |
| **HashMap** | **Hashtable** |
| 1) HashMap **cho phép một key là null và nhiều giá trị null**. | Hashtable **không cho phép bất kỳ key hoặc giá trị null**. |
| 2) HashMap **không đồng bộ**. | Hashtable là **đồng bộ**. |
| 3) HashMap là một lớp mới được giới thiệu trong JDK 1.2. | Hashtable là một [**lớp legacy**](https://viettuts.vn/interview/list-cau-hoi-phong-van-java-collection). |
| 4) HashMap **nhanh**. | Hashtable **chậm**. |
| 5) Chúng ta có thể làm cho HashMap đồng bộ bằng cách gọi phương thức: Map m = Collections.synchronizedMap(hashMap); | Hashtable được đồng bộ nội bộ và không thể hủy đồng bộ hóa. |
| 6) HashMap được **duyệt bởi Iterator**. | Hashtable được **duyệt bởi Enumerator và Iterator**. |
| 7) Iterator trong HashMap là **fail-fast**. | Enumerator trong Hashtable là **không fail-fast**. |
| 8) HashMap kế thừa lớp **AbstractMap**. | Hashtable kế thừa lớp **Dictionary**. |

Hashtable<K, V> hashtable = **new** Hashtable<K, V>();

Hashtable<Integer, String> hashtable = **new** Hashtable<Integer, String>();

import java.util.Hashtable;

public class HashTableExample1 {

    public static void main(String args[]) {

        // init hashtable

        Hashtable<Integer, String> hashtable = new Hashtable<Integer, String>();

        // add elements to hashtable

        hashtable.put(3, "C++");

        hashtable.put(1, "Java");

        hashtable.put(2, "PHP");

        hashtable.put(4, "Python");

        // remove element

        hashtable.remove(2);

    }

}

* 1. **Set:**
     1. **Khởi tạo:**

Lớp HashSet trong java được sử dụng để tạo một bộ sưu tập sử dụng bảng băm để lưu trữ. Nó kế thừa lớp AbstractSet và implements giao diện Set.

Các điểm quan trọng về lớp HashSet trong java là:

* HashSet chỉ chứa các phần tử duy nhất.
* HashSet lưu trữ các phần tử bằng cách sử dụng một cơ chế được gọi là băm.

Sự khác nhau giữa List và Set trong java

* Câu trả lời là **List** có thể chứa các phần tử trùng lặp, trong khi **Set** chỉ chứa các phần tử duy nhất.

Set<E> Name = **new** HashSet<E>();

Set<String> hash\_Set = **new** HashSet<String>();

**import** java.util.\*;

**public** **class** Demo\_Map {

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

Set<String> hash\_Set = **new** HashSet<String>();

hash\_Set.add("Geeks");

hash\_Set.add("For");

hash\_Set.add("Geeks");

hash\_Set.add("Example");

hash\_Set.add("Set");

System.***out***.print("Set output without the duplicates");

System.***out***.println(hash\_Set);

System.***out***.print("Sorted Set after passing into TreeSet");

Set<String> tree\_Set = **new** TreeSet<String>(hash\_Set);

System.***out***.println(tree\_Set);

}

}

* 1. **Date**
* Lớp java.util.Date trong java biểu diễn ngày và giờ (date và time). Nó cung cấp các Constructor và phương thức để xử lý date và time trong java.
* Lớp java.util.Date implements các giao diện Serializable, Cloneable and Comparable<Date>. Nó được kế thừa bởi các lớp java.sql.Date, java.sql.Time và java.sql.Timestamp.
* Sau khi lớp Calendar ra đời, hầu hết các Constructor và phương thức của lớp java.util.Date đã bị khuyến cáo không nên dùng nữa (@deprecated).

|  |  |  |
| --- | --- | --- |
| **No.** | **Constructor** | **Mô tả** |
| 1) | Date() | Tạo một đối tượng Date đại diện cho ngày tháng và thời gian hiện tại. |
| 2) | Date(long milliseconds) | CTạo một đối tượng Date trong một mili giây cho trước kể từ ngày 1 tháng 1 năm 1970, 00:00:00 GMT. |

package vn.viettuts.date;

import java.util.Date;

public class DateExample1 {

    public static void main(String[] args) {

        long miliSeconds = System.currentTimeMillis();

        // cach 1

        Date date1 = new Date();

        System.out.println(date1);

        // cach 2

        Date date2 = new Date(miliSeconds);

        System.out.println(date2);

    }

}

|  |  |  |
| --- | --- | --- |
| **No.** | **Phương thức** | **Mô tả** |
| 1) | boolean after(Date date) | kiểm tra nếu ngày hiện tại là sau ngày đã cho. |
| 2) | boolean before(Date date) | kiểm tra nếu ngày hiện tại là trước ngày đã cho. |
| 3) | Object clone() | Trả về đối tượng nhân bản của ngày hiện tại. |
| 4) | int compareTo(Date date) | So sánh ngày hiện tại với ngày đã cho. |
| 5) | boolean equals(Date date) | So sánh ngày hiện tại với ngày đã cho. |
| 6) | static Date from(Instant instant) | Trả về một thể hiện của đối tượng Date từ ngày Instant. |
| 7) | long getTime() | Trả về thời gian được đại diện bởi đối tượng date. |
| 8) | int hashCode() | Trả về giá trị mã băm cho đối tượng ngày này. |
| 9) | void setTime(long time) | Thay đổi ngày hiện tại và thời gian cho thời gian đã cho. |
| 10) | Instant toInstant() | Chuyển đổi date hiện tại thành đối tượng Instant. |
| 11) | String toString() | Chuyển đổi date hiện tại thành chuỗi String. |

* 1. **Collections**

**Collection trong java** là một root interface trong hệ thống cấp bậc Collection. Java Collection cung cấp nhiều interface (Set, List, Queue, Deque vv) và các lớp (ArrayList, Vector, LinkedList, PriorityQueue, HashSet, LinkedHashSet, TreeSet vv).

* 1. **Vectors**

Lớp Vector trong Java triển khai một mảng động. Nó tương tự như ArrayList, nhưng với hai điểm khác biệt:

Vector được đồng bộ.

Vector chứa các phương thức legacy mà không là một phần của Collection Framework.

* 1. **Properties**

Lớp **Properties** trong java được sử dụng để tạo ra đối tượng chứa cặp khóa (key) và giá trị (value) như một chuỗi. Lớp **java.util.Properties** là một lớp con của [Hashtable](https://gpcoder.com/2709-hashtable-trong-java/).

Lớp **Properties**có thể được sử dụng để lấy giá trị (key) dựa trên khóa (key) của thuộc tính. Lớp Properties cung cấp các phương thức lấy dữ liệu từ các file .properties và lưu trữ dữ liệu vào file .properties. Hơn nữa, nó có thể được sử dụng để có được properties của hệ thống.

**File Properties không cần biên dịch lại, nếu thông tin được thay đổi từ file .properties**: Nếu có bất kỳ thông tin nào được thay đổi từ file .properties, bạn không cần phải biên dịch lại lớp java. Nó được sử dụng để lưu trữ thông tin mà sẽ được thay đổi thường xuyên.

**Một số quy định nội dung của file .properties:**

* Mỗi cặp khóa – giá trị (key-value) được viết trên một dòng, phân cách bằng dấu =.
* Sử dụng dấu # để thêm ghi chú cho các cặp khóa – giá trị (key-value).
* Có thể thêm trùng khóa, tuy nhiên lớp Properties chỉ lấy giá trị của khóa cuối cùng, do lớp Properties sử dụng map để lưu dữ liệu nên cặp khóa cuối cùng được giữ lại.
* Các dòng bắt đầu bằng các ký tự ! hoặc # bị bỏ qua. Dòng trắng cũng bị bỏ qua.
* Tên key không được chứa khoảng trắng ở giữa. Ví dụ: key 1 là không đúng, phải đặt là key1.
* Có thể có thêm khoảng trắng ở 2 đầu của khóa và giá trị: tuy nhiên lớp Properties sẽ tự động cắt khoảng trắng ở 2 đầu của khóa, cắt khoảng trắng đầu của giá trị, khoảng trắng ở cuối giá trị vẫn giữ nguyên. Ví dụ: name=gpcoder và name = gpcoder là giống nhau.
* Giá trị có thể kéo dài một vài dòng nếu mỗi dòng được kết thúc bởi dấu gạch chéo ngược (‘\‘). Ví dụ:

targetCities=\

    Detroit,\

    Chicago,\

    Los Angeles

Điều này tương đương với targetCities = Detroit, Chicago, Los Angeles (khoảng trắng ở đầu dòng được bỏ qua).

* Ký tự \n, \r, và \t có thể được sử dụng.
* Ký tự dấu gạch chéo \ ngược của anh ta phải được thay thế bằng dấu gạch chéo kếp (\\). Ví dụ: path=c:\\docs\\doc1
* Ký tự UNICODE có thể được nhập như trong một chương trình Java, sử dụng tiền tố \u. Ví dụ: \u002c.

**Các phương thức của lớp Properties trong java**

Lớp Properties kế thừa từ lớp [Hashtable](https://gpcoder.com/2709-hashtable-trong-java/). Lớp Hashtable cài đặt [Map Interface](https://gpcoder.com/2632-map-interface-trong-java/) nên bạn có thể sử dụng các phương thức đã được định nghĩa của lớp [HashTable](https://gpcoder.com/2709-hashtable-trong-java/) và [Map Interface](https://gpcoder.com/2632-map-interface-trong-java/).

Ngoài ra, lớp Properties còn cung cấp các phương thức riêng của nó như dưới đây:

|  |  |
| --- | --- |
| **Phương thức** | **Mô tả** |
| public void load(Reader r) | Tải dữ liệu từ đối tượng Reader. |
| public void load(InputStream is) | Tải dữ liệu từ đối tượng InputStream. |
| public String getProperty(String key) | Trả về giá trị dựa trên key. |
| public void setProperty(String key,String value) | Gán giá trị (value) dựa vào khóa (key). |
| public void store(Writer w, String comment) | Lưu các thuộc tính trong đối tượng Writer. |
| public void store(OutputStream os, String comment) | Lưu các thuộc tính trong đối tượng OutputStream. |
| storeToXML(OutputStream os, String comment) | Lưu các thuộc tính trong đối tượng Writer để tạo tài liệu xml. |
| public void storeToXML(Writer w, String comment, String encoding) | Lưu các thuộc tính trong đối tượng Writer để tạo ra tài liệu xml với mã hoá được chỉ định. |

**VD:**

**config.properties**

siteName=GP Coder

siteUrl=gpcoder.com

**PropertiesExample1.java**

import java.io.FileReader;

import java.util.Properties;

public class PropertiesExample1 {

    public static void main(String[] args) throws Exception {

        // create reader object

        FileReader reader = new FileReader("config.properties");

        // crate properties object

        Properties properties = new Properties();

        properties.load(reader);

        // show file info

        System.out.println(properties.getProperty("siteName"));

        System.out.println(properties.getProperty("siteUrl"));

    }

}

* 1. **ResourceBundle**

ResourceBundle là một class trong Java dùng để lấy dữ liệu / văn bản /string theo các ngôn ngữ khác nhau. Với mỗi ngôn ngữ ta có một file .properties riêng để chứa message, label… cho ngôn ngữ đó. Và ResourceBundle thực hiện đọc các file .properties theo ngôn ngữ truyền vào.

File chứa message/label cho từng ngôn ngữ sẽ có định dạng: **baseName\_locale.properties**.

* **baseName** chính là bắt đầu tên file, các file ngôn ngữ đều phải có chung baseName.
* **locale** chính là ngôn ngữ.

**messages\_en.properties**

hello=Hello

**messages\_vi.properties**

hello=Xin Chào

**Demo\_ ResourceBundle.java**

**import** java.util.Locale;

**import** java.util.ResourceBundle;

**public** **class** Demo\_ ResourceBundle {

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

Locale localeEn = **new** Locale("en");

ResourceBundle labels = ResourceBundle.*getBundle*("messages", localeEn);

System.***out***.println("Xin chào trong tiếng anh: "+labels.getString("hello"));

Locale localeVi = **new** Locale("vi");

labels = ResourceBundle.*getBundle*("messages", localeVi);

System.***out***.println("Xin chào trong tiếng việt: "+labels.getString("hello"));

}

}

* 1. **Calendar**

Lớp **Calendar trong java** là một lớp trừu tượng (abstract) cung cấp phương thức chuyển đổi ngày giữa một thời điểm cụ thể theo thời gian và một tập hợp các trường của calendar như MONTH, YEAR, HOUR, ... Nó kế thừa lớp Object và implements giao diện Comparable.

import java.util.Calendar;

public class CalendarExample1 {

    public static void main(String[] args) {

        Calendar calendar = Calendar.getInstance();

        System.out.println("Ngay gio hien tai: " + calendar.getTime());

        calendar.add(Calendar.DATE, -15);

        System.out.println("15 ngay sau: " + calendar.getTime());

        calendar.add(Calendar.MONTH, 4);

        System.out.println("4 thang sau: " + calendar.getTime());

        calendar.add(Calendar.YEAR, 2);

        System.out.println("2 nam sau: " + calendar.getTime());

    }

}

|  |  |
| --- | --- |
| **Phương thức** | **Mô tả** |
| abstract void add(int field, int amount) | Nó được sử dụng để thêm hoặc trừ số lượng thời gian nhất định vào trường calendar đã cho, dựa trên các quy tắc của calendar. |
| int get(int field) | Nó được sử dụng để trả lại giá trị của trường calendar đã cho. |
| static Calendar getInstance() | Nó được dùng để lấy calendar sử dụng time zone và locale mặc định. |
| abstract int getMaximum(int field) | Nó được sử dụng để trả về giá trị MAX cho trường calendar đã cho của thể hiện Calendar hiện tại. |
| abstract int getMinimum(int field) | Nó được sử dụng để trả về giá trị MIN cho trường calendar đã cho của thể hiện Calendar hiện tại. |
| void set(int field, int value) | Nó được sử dụng để thiết lập trường cho trước vói giá trị đã cho. |
| void setTime(Date date) | Nó được sử dụng để thiết lập time của Calendar với Date đã cho. |
| Date getTime() | Nó được sử dụng để trả về đối tượng Date biểu diễn giá trị time của Calendar. |

* 1. **Enum**

Enum trong java là một kiểu dữ liệu đặc biệt của Java được sử dụng để định nghĩa các tập hợp các hằng số. Cụ thể hơn, Java enum là một kiểu đặc biệt của lớp trong java. Một enum có thể chứa các trường, phương thức và Constructor.

Nó có thể được sử dụng để định nghĩa các ngày trong tuần (SUNDAY, MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY, SATURDAY), các mùa trong năm (SPRING, SUMMER, FALL, WINTER), ...

Enum trong java có thể được định nghĩa bên trong hoặc bên ngoài một lớp, vì nó tương tự như lớp trong java.

public class EnumExample1 {

    // define enum

    enum Season {

        SPRING, SUMMER, FALL, WINTER;

    }

    public static void main(String[] args) {

        Season season = Season.WINTER;

        System.out.println(season);

    }

}

// define enum

enum Season {

    SPRING, SUMMER, FALL, WINTER;

}

public class EnumExample2 {

    public static void main(String[] args) {

        Season season = Season.WINTER;

        System.out.println(season);

    }

}

public class EnumExample4 {

    // define enum Season

    public enum Season {

        SPRING, SUMMER, FALL, WINTER

    }

    public static void main(String[] args) {

        // traverse enum Season

        for (Season s : Season.values()) {

            System.out.println(s);

        }

    }

}

* 1. **Iterator**

Iterator trong Java là một interface được sử dụng để thay thế Enumerations trong Java Collection Framework. Bạn có thể sử dụng interator để:

* Duyệt các phần tử từ đầu đến cuối của một collection.
* Iterator cho phép xóa phần tử khi lặp một collection.

import java.util.ArrayList;

import java.util.Iterator;

import java.util.ListIterator;

public class IteratorDemo {

    public static void main(String args[]) {

        ArrayList listLanguages = new ArrayList();

        // them phan tu vao array list

        listLanguages.add("Java");

        listLanguages.add("Python");

        listLanguages.add("PHP");

        listLanguages.add(".NET");

        listLanguages.add("C");

        listLanguages.add("C++");

        // su dung iterator de hien thi noi dung cua listLanguages

        System.out.println("Danh sach ngon ngu lap trinh: ");

        Iterator itr = listLanguages.iterator();

        while (itr.hasNext()) {

            Object element = itr.next();

            System.out.println(element);

        }

    }

}

* 1. **FileInputStream, FileOutputStream**

**Lớp FileInputStream trong java** đọc được các byte từ một input file. Nó được sử dụng để đọc dữ liệu theo định dạng byte (các byte stream) như dữ liệu hình ảnh, âm thanh, video vv. Bạn cũng có thể đọc các dữ liệu có định dạng ký tự. Tuy nhiên, để đọc các dòng ký tự (các character stream), bạn nên sử dụng lớp FileReader.

import java.io.FileInputStream;

import java.io.IOException;

public class FileInputStreamExample1 {

    public static void main(String args[]) throws IOException {

        FileInputStream fin = null;

        try {

            fin = new FileInputStream("D:\\testout.txt");

            int i = fin.read();

            System.out.print((char) i);

            fin.close();

        } catch (Exception e) {

            System.out.println(e);

        } finally {

            fin.close();

        }

    }

}

**Java FileOutputStream** là một output stream được sử dụng để ghi dữ liệu vào một file theo định dạng byte (byte stream).

Sử dụng lớp **FileOutputStream** trong java, nếu bạn phải ghi các giá trị nguyên thủy vào một file. Bạn có thể ghi dữ liệu theo định dạng byte hoặc định dạng ký tự thông qua lớp FileOutputStream. Tuy nhiên, đối với các dữ liệu được ghi theo ký tự, sử dụng FileWriter thích hợp hơn FileOutStream.

import java.io.FileOutputStream;

import java.io.IOException;

public class FileOutputStreamExample {

    public static void main(String args[]) throws IOException {

        FileOutputStream fout = null;

        try {

            fout = new FileOutputStream("D:\\testout.txt");

            fout.write(65);

            fout.close();

            System.out.println("success...");

        } catch (Exception e) {

            System.out.println(e);

        } finally {

            // close file output stream

            fout.close();

        }

    }

}

* 1. **BufferedInputStream, BufferedOutputStream**

**Lớp BufferedOutputStream** trong java được sử dụng để đệm một output stream. Trong nội bộ của lớp này sử dụng bộ đệm để lưu trữ dữ liệu. Vì vậy, nó giúp hiệu suất ghi dữ liệu nhanh.

Để thêm bộ đệm vào một đối tượng OutputStream, sử dụng lớp BufferedOutputStream.

import java.io.BufferedOutputStream;

import java.io.FileOutputStream;

import java.io.IOException;

public class BufferedOutputStreamExample {

    public static void main(String args[]) throws IOException {

        FileOutputStream fout = null;

        BufferedOutputStream bout = null;

        try {

            fout = new FileOutputStream("D:\\testout.txt");

            bout = new BufferedOutputStream(fout);

            String s = "Welcome to java.";

            byte b[] = s.getBytes();

            bout.write(b);

            bout.flush();

        } catch (IOException ex) {

            ex.printStackTrace();

        } finally {

            bout.close();

            fout.close();

        }

        System.out.println("success!");

    }

}

**Lớp BufferedInputStream** trong java được sử dụng để đọc thông tin từ stream (buffered stream). Trong nội bộ của lớp này sử dụng cơ chế đệm để làm cho hiệu suất đọc nhanh hơn.

Những điểm quan trọng về BufferedInputStream là:

* Khi các byte từ stream được bỏ qua hoặc đọc, bộ đệm nội bộ sẽ tự động nạp lại từ input steam chứa, nhiều byte tại một thời điểm.
* Khi một BufferedInputStream được tạo ra, một mảng đệm nội bộ sẽ được tạo ra.

import java.io.BufferedInputStream;

import java.io.FileInputStream;

import java.io.IOException;

public class BufferedInputStreamExample {

public static void main(String args[]) throws IOException {

FileInputStream fin = null;

BufferedInputStream bin = null;

try {

fin = new FileInputStream("D:\\testout.txt");

bin = new BufferedInputStream(fin);

int i;

while ((i = bin.read()) != -1) {

System.out.print((char) i);

}

} catch (IOException e) {

System.out.println(e);

} finally {

bin.close();

fin.close();

}

}

}

* 1. **java.io.File**
     1. **Tạo file:**

**import** java.io.File;

**import** java.io.IOException;

**public** **class** File\_Demo {

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

**try** {

File file = **new** File("E:/myfile.txt");

**if** (file.createNewFile())

System.***out***.println("Success!");

**else**

System.***out***.println("Error, file already exists.");

} **catch** (IOException ioe) {

ioe.printStackTrace();

}

}

}

* + 1. **Xóa File**