Chương 4.
Xử lý ngoại lệ
& Các lớp tiện ích của Java

Nội dung

- Xử lý ngoại lệ
- Các lớp tiện ích của Java
- Java Collection Framework

4.1. Xử lý ngoại lệ (EXCEPTION HANDLING)

Giới thiệu

- Ngoại lệ (exception) là viết tắt của cụm từ "sự kiện đặc biệt".
- Một ngoại lệ là một sự kiện, xảy ra trong thời gian thực hiện của một chương trình, làm gián đoạn dòng chảy bình thường của một chương trình.
- Khi một lỗi xảy ra trên một phương thức, phương thức này tạo ra một đối tượng và đưa nó vào Runtime System. Đối tượng đó được gọi là đối tượng ngoại lệ, nó chứa tất cả các thông tin về lỗi và trạng thái của chương trình khi xảy ra lỗi. Sau đó Runtime System sẽ xử lý lỗi này.
- Tạo một đối tượng ngoại lệ và đưa nó vào Runtime System được gọi là ném ra một ngoại lệ.
- Một lập trình viên cần thấy trước các điều kiện và quan tâm đến chúng bằng cách viết thông báo lỗi thích hợp để đưa ra khi chương trình gặp lỗi.
- Quá trình xử lý ngoại lệ được gọi là bắt ngoại lệ, nếu Runtime System không xử lý được ngoại lệ thì chương trình sẽ kết thúc.

Có 3 loại ngoại lệ

- ***Checked Exception:*** *là loại ngoại lệ xẩy ra trong thời gian dịch chương trình. Loại ngoại lệ này không thể được bỏ qua trong quá trình biên dịch, bắt buộc phải xử lý nó.*

  Ví dụ: IOException, FileNotFoundException,...

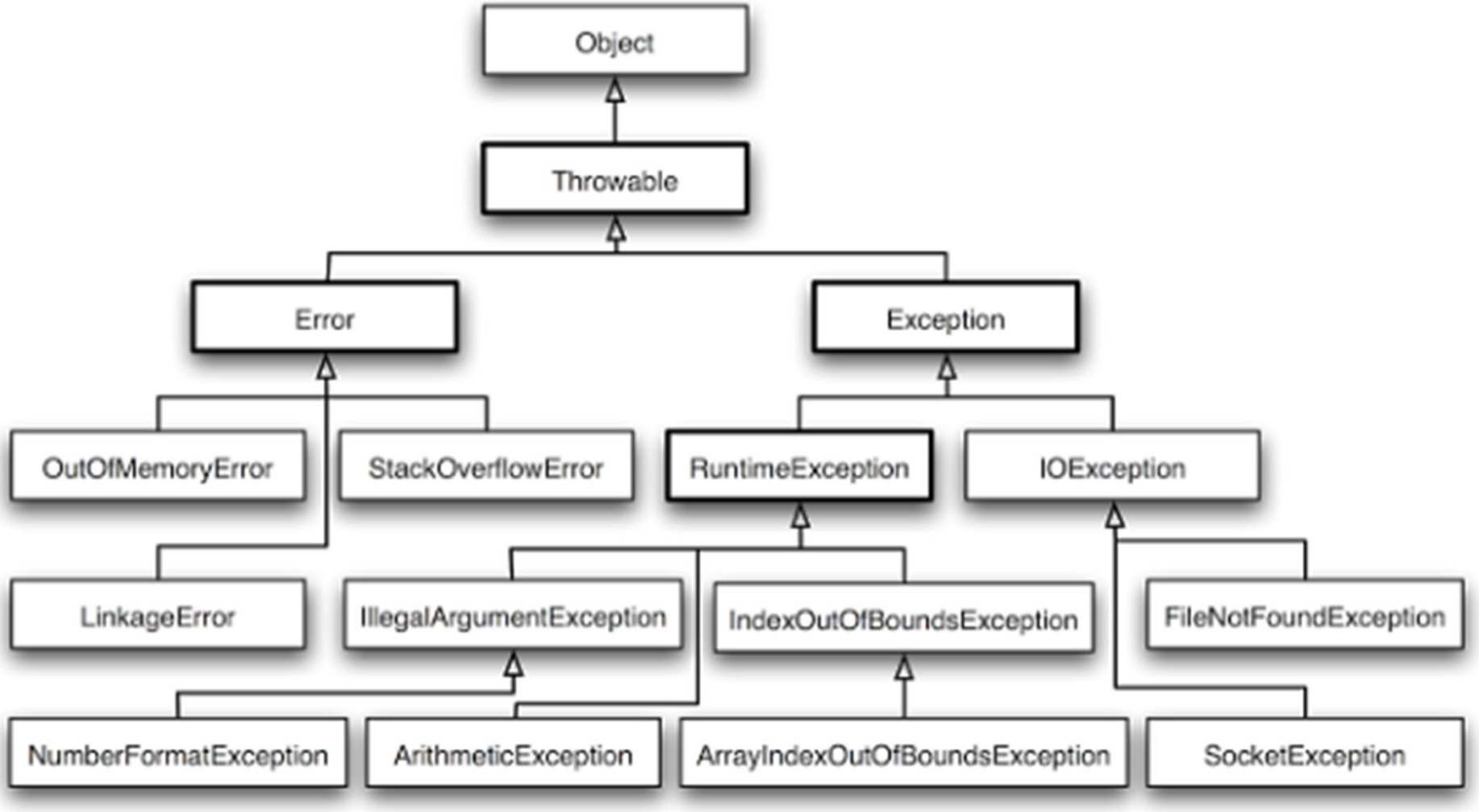
- ***Unchecked Exception:*** *là loại ngoại lệ xảy ra tại thời điểm thực thi chương trình. Đó là lỗi kỹ thuật lập trình, lỗi logic của chương trình... Loại ngoại lệ này được bỏ qua trong quá trình biên dịch, và không bắt buộc phải xử lý nó khi lập trình.*

  Ví dụ: NumberFormatException, ArithmeticException, ...

- ***error:*** *là những vấn đề nghiêm trọng liên quan đến môi trường thực thi của ứng dụng, hệ thống. Nó thường làm chết chương trình.*

  Ví dụ: OutOfMemoryError, StackOverflowError,...

Hệ thống cấp bậc của các lớp ngoại lệ (Mô hình Exception)



Một số phương thức của lớp Exception

- **public String getMessage()**

  Trả về thông báo lỗi cụ thể về ngoại lệ đã xảy ra.

- **public Throwable getCause()**

  Trả về nguyên nhân xẩy ra ngoại lệ

- **public String toString()**

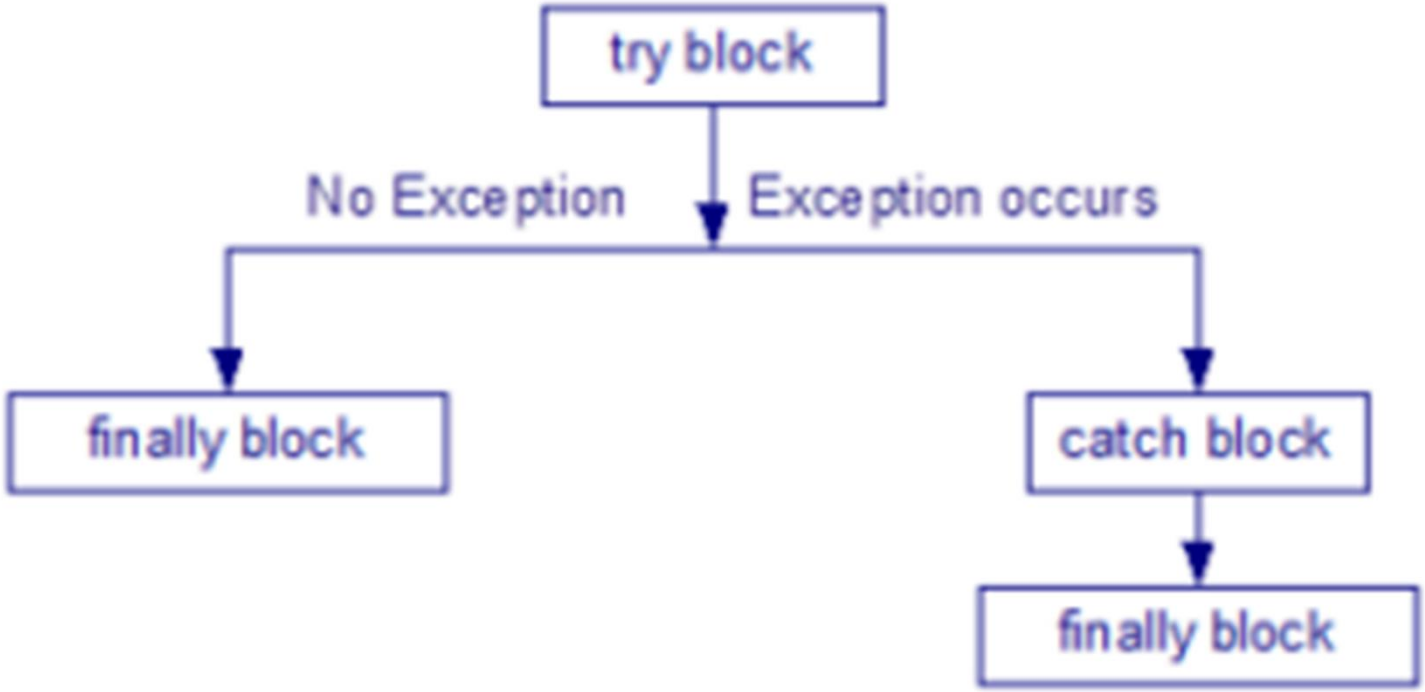
  Trả về tên của lớp và kết hợp với kết quả từ phương thức getMessage()

Một số lỗi ngoại lệ

- **Exception**: *Lớp nền của run-time*
- **NullPointerException:** *Một đối tượng không tồn tại*
- **ClassNotFoundException:** *Không tìm thấy Class*
- **FileNotFoundException**: *Không tìm thấy file*
- **ArrayIndexOutOfBoundsException:** *Vượt quá chỉ mục của mảng*
- **ArithmeticException:** *Lỗi thực thi một phép toán*
- **NumberFormatException:** *Định dạng số không đúng*
- **IOException:** *Lỗi nhập xuất*
- **EOFException**: *Kết thúc một tập tin*

Xử lý ngoại lệ trong Java

- Để bắt lỗi ngoại lệ ta sử dụng các khối **try-catch** và **finally**.
- Có 2 cách xử lý lỗi ngoại lệ trong Java:
  1) Trực tiếp bằng **try-catch**
  2) Gián tiếp bằng **throws**.



Sử dụng khối try-catch để xử lý ngoại lệ

Cú pháp:

```
try {
        //khối lệnh có thể phát sinh ngoại lệ
} catch (tên_ngoại_lệ   e) {
        //khối lệnh xử lý khi có ngoại lệ
}
```

Ý nghĩa: xử lý trực tiếp tại khối lệnh có thể phát sinh lỗi và chương trình có thể chạy tiếp những lệnh tiếp theo. Hoặc là bắt lỗi để chuyển thành một lỗi khác và đưa ra ngoài xử lý.

Ví dụ: chia cho 0 trong trường hợp không/có xử lý ngoại lệ.

Sử dụng khối try-catch để xử lý ngoại lệ

Trường hợp dùng **try** có nhiều **catch**: trong trường hợp có thể có nhiều ngoại lệ xẩy ra ta sẽ dùng nhiều **catch** để xử lý các ngoại lệ đó.

Cú pháp:

```
try {
        //khối lệnh có thể phát sinh ngoại lệ
} catch (tên_ngoại_lệ   e1) {
        //khối lệnh xử lý khi có ngoại lệ e1
} catch (tên_ngoại_lệ e2) {
        //khối lệnh xử lý khi có ngoại lệ e2
}...
```

Chú ý:

- ✓ *Tại một thời điểm, chỉ một ngoại lệ được xuất hiện và tại một thời điểm chỉ có một khối catch được thực thi.*
- ✓ *Tất cả khối catch phải được sắp xếp từ cụ thể nhất tới chung nhất.*

Khối finally

Khối **finally** là một khối được sử dụng để thực thi các phần code quan trọng như đóng kết nối, đóng stream,… Khối **finally** luôn luôn được thực thi dù cho ngoại lệ có được xử lý hay không. Khối **finally** phải được theo sau bởi khối **try** hoặc khối **catch**.

- ***Qui tắc**: Với một khối try thì có thể không có hoặc có nhiều khối catch, nhưng chỉ có một khối finally.*
- ***Ghi chú**: Khối finally sẽ không được thực thi nếu chương trình thoát ra (bởi gây ra một lỗi nghiêm trọng làm ngừng tiến trình).*

Từ khóa Throw

Từ khóa **throw** trong Java được sử dụng để ném tường minh một ngoại lệ. Chúng ta có thể ném hoặc Checked Exception hoặc Unchecked Exception trong Java bởi từ khóa **throw**.

Từ khóa **throw** được sử dụng chủ yếu để ném các Custom Exception (là các ngoại lệ được định nghĩa bởi lập trình viên).

Cú pháp:

```
throw exception;
```

Ví dụ:

```
throw new ArithmeticException("Device by zero");
```

Từ khóa Throws

Từ khóa **throws** trong Java được sử dụng để khai báo một ngoại lệ. Nó cung cấp một thông tin tới lập trình viên rằng có thể xuất hiện một ngoại lệ, từ đó để xử lý ngoại lệ nhằm duy trì luồng chuẩn của chương trình.

Cú pháp:

```
Kiểu Tên_phương_thức() throws Tên_lớp_exception{
        //phần code của phương thức
}
```

Từ khóa Throws

Qui tắc: Nếu bạn đang gọi một phương thức mà khai báo một ngoại lệ, bạn phải hoặc bắt hoặc khai báo ngoại lệ đó.

Có hai trường hợp:

- ✓ Trường hợp 1: bắt ngoại lệ (sử dụng **try-catch** để xử lý ngoại lệ đó)
- ✓ Trường hợp 2: khai báo ngoại lệ (xác định từ khóa **throws** với phương thức đó)

Custom Exception

- **Custom Exception** là ngoại lệ do người lập trình tự định nghĩa hay tự tạo riêng cho mình.
  Custom Exception trong Java được sử dụng để tùy biến ngoại lệ theo yêu cầu của người dùng. Với sự giúp đỡ của loại ngoại lệ này, ta có thể có kiểu và thông điệp ngoại lệ riêng cho mình.
- Cách định nghĩa:

```
class Tên_Ngoại_lệ extends Exception {
        //xử lý ngoại lệ
}
```

Ví dụ:

Xử lý ngoại lệ & Ghi đè phương thức

- **Nếu phương thức của lớp cha không khai báo một ngoại lệ**:

  Qui tắc 1: phương thức ghi đè của lớp con không thể khai báo Checked Exception.

  Qui tắc 2: phương thức ghi đè của lớp con không thể khai báo Checked Exception nhưng có thể khai báo Unchecked Exception.

- **Nếu phương thức lớp cha khai báo một ngoại lệ**: phương thức ghi đè của lớp con có thể khai báo cùng ngoại lệ đó hoặc không khai báo ngoại lệ nào, nhưng không thể khai báo ngoại lệ cha.

4.2. Các lớp tiện ích

Các lớp tiện ích trong gói java.lang

- Gói **java.lang** chứa đựng nhiều lớp cơ bản và thường được sử dụng trong Java:
  - Các lớp xâu: String & String Buffer
  - Lớp Math
  - Các lớp bao bọc kiểu nguyên thủy:
    - *Boolean*
    - *Byte*
    - *Character*
    - *Double*
    - *Float*
    - *Integer*
    - *Long*
    - *Short*

java.lang.String

- **Các xâu kiểu String là các đối tượng không thể thay đổi được:**

```
String s = “abc”;
String s2 = s;
s = s.concat(“def”);
```

- **Một vài phương thức quan trọng:**

```
public char charAt(int index)
public String concat(String s)
public boolean equals(Object anObject)
public boolean equalsIgnoreCase(String s)
public int length()
public String replace(char old, char new)
public String substring(int begin)
public String substring(int begin, int end)
public String toLowerCase()
public String toUpperCase()
public String trim()
```

java.lang.StringBuffer

- Lớp StringBuffer được sử dụng khi có nhiều thay đổi đến xâu đối với các ký tự.
- Một vài phương thức quan trọng:

  public synchronized StringBuffer **append(String s)**

  public synchronized StringBuffer **insert(int offset, String s)**

  public synchronized StringBuffer **reverse()**

  public String **toString()**

java.lang.Math

- Lớp **Math** là lớp được sử dụng để thực hiện các phép toán cơ bản của toán học.
- Tất cả các phương thức của lớp Math là *static*.
- Không thể kế thừa từ lớp Math.
- Lớp Math chứa các hằng xấp xỉ như *pi và e*:

  public final static double **Math.PI**

  public final static double **Math.E**

- Các phương thức: **abs(x), ceil(d), floor(d), max(x, y), min(x, y), random(), round(x), sin(x), cos(x), tan(x), sqrt(x),…**

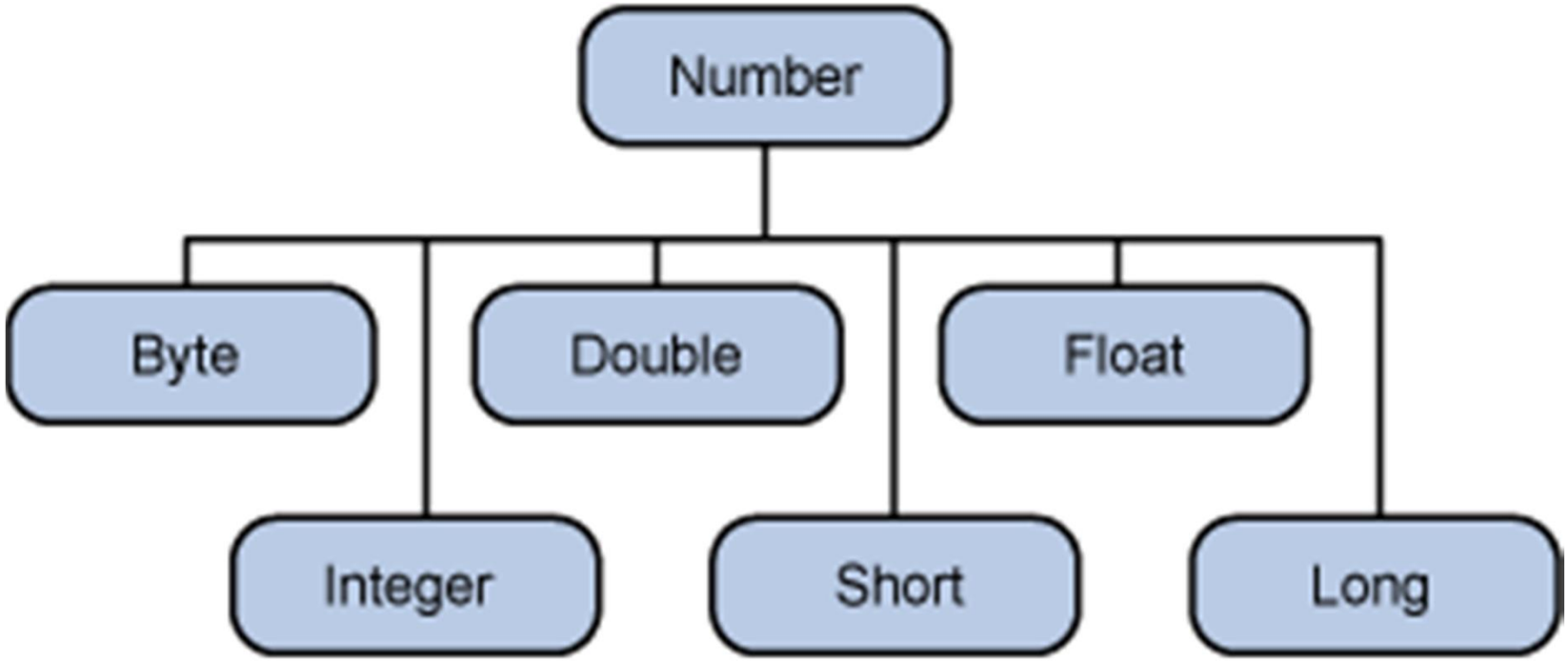
Các lớp bao bọc kiểu nguyên thủy

- Lớp Wrapper (bao bọc) trong Java cung cấp kỹ thuật để chuyển đổi kiểu gốc (primitive) thành đối tượng và từ đối tượng thành kiểu gốc.
- Có các lớp bao bọc đối với mọi lớp nguyên thủy trong Java.

  Ví dụ: lớp bao bọc của *int* là Integer, *float* là Float

Các lớp bao bọc (tiếp)

- Tất cả các lớp bao bọc kiểu số là lớp con của lớp trừu tượng Number:



Java.lang – Wrappers (cont.)

- Khởi tạo các đối tượng Wrapper:
  - Các hàm khởi tạo Wrapper:

```
Integer i1 = new Integer(42);
Integer i2 = new Integer("42");
```

  - Phương thức valueOf()

```
Integer i3 = Integer.valueOf("101011", 2);
Float f1 = Float.valueOf("3.14f");
```

Java.lang – Wrappers (cont.)

- Sử dụng các phương thức chuyển đổi:
  - **xxxValue**() => chuyển giá trị số của lớp bao bọc thành giá trị nguyên thủy
  - **parseXxx**() and **valueOf**() => chuyển xâu thành số, ném ra một ngoại lệ NumberFormatException nếu xâu không đúng định dạng.
  - **toString**()

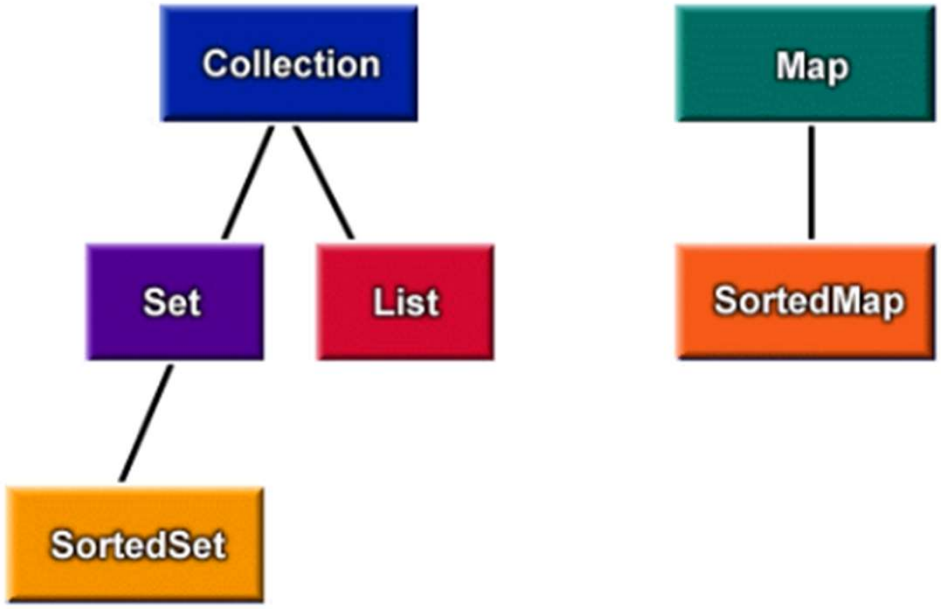
Phương thức equals() - Lớp Java.lang

- equals() chỉ được dùng để so sánh các đối tượng.
- equals() trả về true hoặc false.
- Lớp StringBuffer không ghi đè equals().
- Các lớp String và wrapper là final và có ghi đè equals().

4.3. JAVA COLLECTIONS FRAMEWORK

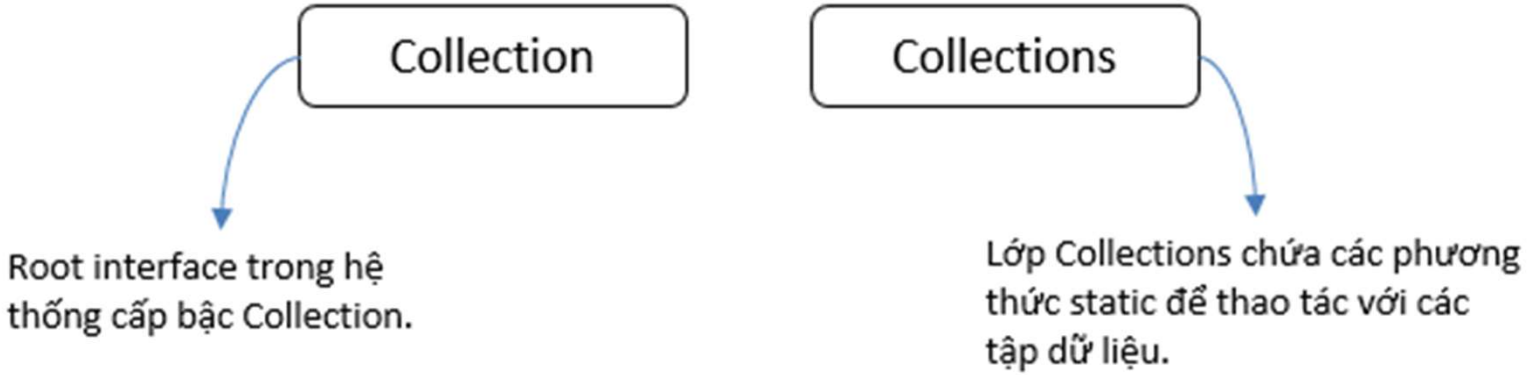
Collections Framework

- Khái niệm Collection được dùng để biểu diễn một nhóm các đối tượng.
- Java collections framework là một tập hợp các giao diện và các lớp để làm việc với các nhóm đối tượng.
- Java Collections Framework cung cấp:
  - **Interfaces**
  - **Implementations**: triển khai các giao diện.
  - **Algorithms**: các thuật toán.

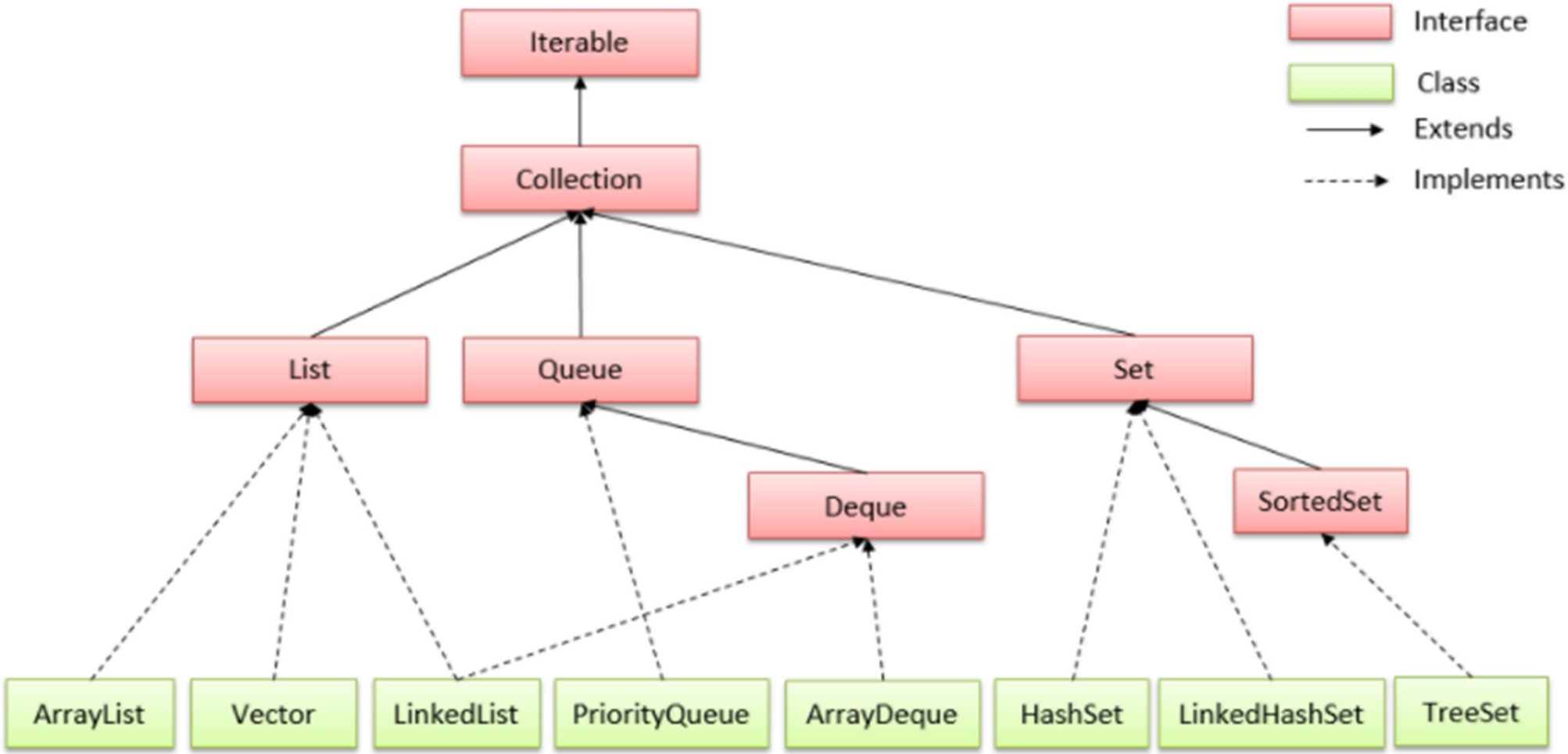


Collection vs Collections

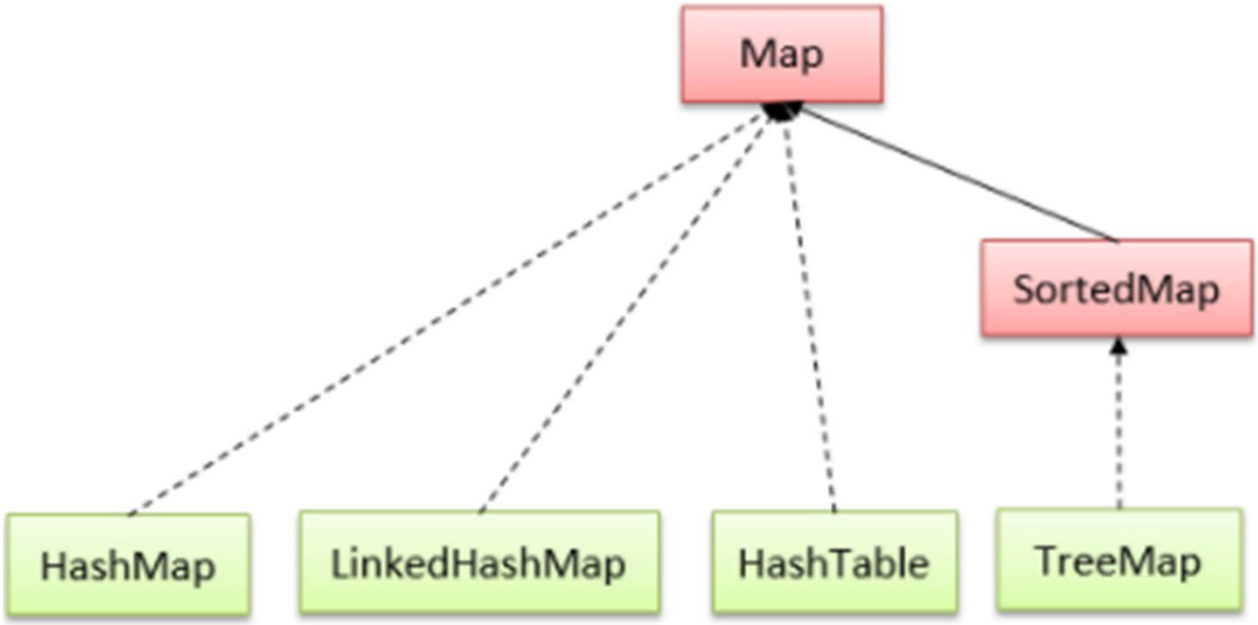
- **Collections** trong java là một khuôn khổ cung cấp một kiến trúc để lưu trữ và thao tác tới nhóm các đối tượng. Tất cả các hoạt động mà bạn thực hiện trên một dữ liệu như tìm kiếm, phân loại, chèn, xóa,… có thể được thực hiện bởi Java Collections.
- **Collection** trong java là một root interface trong hệ thống cấp bậc Collection. Java Collection cung cấp nhiều interface (Set, List, Queue, Deque,…) và các lớp (ArrayList, Vector, LinkedList, PriorityQueue, HashSet, LinkedHashSet, TreeSet,…).



Hệ thống cấp bậc Collection trong java



Hệ thống cấp bậc Collection trong java



Các interface chính của Collection

- **Set:** là một collection không thể chứa 2 giá trị trùng lặp.
- **List:** là một collection có thứ tự. List có thể chứa các phần tử trùng lặp và có thể truy cập chúng bằng chỉ số vị trí.
- **Queue (hàng đợi):** là một collection được sử dụng để chứa nhiều phần tử trước khi xử lý. Bên cạnh các thao tác cơ bản của collection, Queue cung cấp các thao tác bổ sung như chèn, lấy ra và kiểm tra. Queue có thể được sử dụng như là FIFO.
- **Deque:** là một collection được sử dụng để chứa nhiều phần tử trước khi xử lý. Ngoài các thao tác cơ bản của collection, một Deque cung cấp các thao tác bổ sung như chèn, lấy ra và kiểm tra. Deque có thể được sử dụng như là FIFO và LIFO.
- **Map:** là một collection mà mỗi đối tượng ánh xạ mỗi key tương ứng với một giá trị. Map không thể chứa giá trị trùng lặp. Mỗi key có thể ánh xạ đến nhiều nhất một giá trị.

Iterator interface

- **Iterator interface**

  Iterator cung cấp phương tiện duyệt các phần tử từ đầu đến cuối của một collection.

- **Các phương thức của Iterator interface**

Phương thức	Mô tả
public boolean **hasNext()**	Trả về true nếu iterator còn phần tử kế tiếp phần tử đang duyệt.
public object **next()**	Trả về phần tử hiện tại và di chuyển con trỏ tới phần tử tiếp theo.

- **Duyệt các phần tử của collection**

  Có 2 cách để duyệt các phần tử của collection trong java.

  1. Sử dụng Iterator interface.
  2. Sử dụng vòng lặp for-each.

Ví dụ

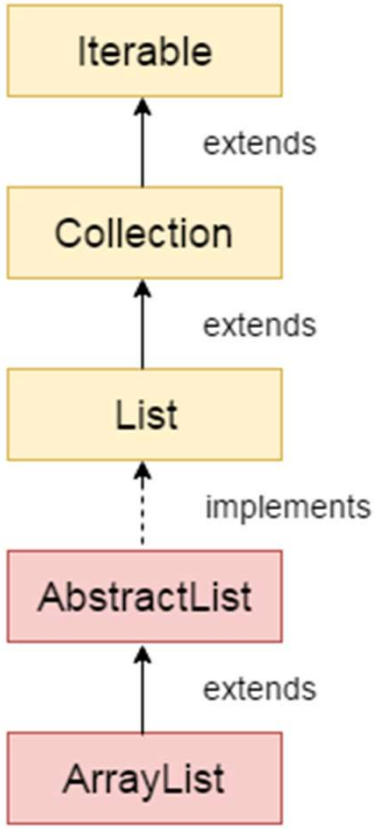
```
ArrayList<String> arrList = new ArrayList<String>();

arrList.add("Java");
arrList.add("C++");
arrList.add("PHP");
arrList.add("Java");

Iterator<String> itr = arrList.iterator();
while (itr.hasNext()) {
    System.out.print(itr.next() + ", ");
}

System.out.println();
for (String obj : arrList) {
    System.out.print(obj + ", ");
}
```

Một số phương thức của Collection

Phương thức	Mô tả
public boolean **add(Object element)**	- Chèn một phần tử vào collection.
public boolean **addAll(Collection c)**	- Chèn các phần tử collection được chỉ định vào collection gọi phương thức này.
public boolean **remove(Object element)**	- Xóa phần tử từ collection.
public boolean **removeAll(Collection c)**	- Xóa tất cả các phần tử của collection được chỉ định từ collection gọi phương thức này.
public int **size()**	- Trả lại tổng số các phần tử trong collection.
public boolean **contains(Object element)**	- Tìm kiếm phần tử.
public Iterator **iterator()**	- Trả về một iterator.
public Object[] **toArray()**	- Chuyển đổi collection thành mảng (array).
public boolean **isEmpty()**	- Kiểm tra nếu collection trống.
public boolean **equals(Object element)**	- So sánh 2 collection.

Lớp ArrayList trong java

- **Lớp ArrayList trong java** được sử dụng như một mảng động để lưu trữ các phần tử. Nó kế thừa lớp AbstractList và List interface.



- **Những đặc điểm của ArrayList**:
  - Lớp ArrayList có thể chứa các phần tử trùng lặp.
  - Lớp ArrayList duy trì thứ tự của phần tử được thêm vào.
  - Lớp ArrayList là không đồng bộ (non-synchronized).
  - Lớp ArrayList cho phép truy cập ngẫu nhiên vì nó lưu dữ liệu theo chỉ mục.
  - Lớp ArrayList thao tác chậm vì cần nhiều sự dịch chuyển nếu bất kỳ phần tử nào bị xoá khỏi danh sách.

Lớp ArrayList trong java

- Các phương thức khởi tạo

Constructor	Mô tả
ArrayList()	Khởi tạo một ArrayList trống.
ArrayList(Collection c)	Khởi tạo một ArrayList với các phần tử của collection c.
ArrayList(int capacity)	Khởi tạo một ArrayList có kích thước ban đầu được chỉ định.

```
ArrayList<String> arrList = new ArrayList<String>();
ArrayList<Integer> arrList1 = new ArrayList<Integer>(5);
ArrayList<String> arrList2 = new ArrayList<String>(arrList);
```

Một số phương thức lớp ArrayList

Phương thức	Mô tả
boolean **add(Object o)**	Thêm phần tử được chỉ định vào cuối danh sách.
boolean **addAll(int index, Collection c)**	Chèn tất cả các phần tử trong collection được chỉ định vào 1 ArrayList kể từ vị trí index.
void **clear()**	Xóa tất cả các phần tử khỏi danh sách.
int **indexOf(Object o)**	Trả về chỉ số đầu tiên của phần tử được chỉ định, hoặc -1 nếu danh sách không chứa phần tử này.
int **lastIndexOf(Object o)**	Trả về chỉ số cuối cùng của phần tử được 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 theo đúng thứ tự.
Object[] **toArray(Object[] a)**	Trả về một mảng chứa tất cả các phần tử trong danh sách này theo đúng thứ tự.
Object **clone()**	Trả về một bản sao của ArrayList.

So sánh giữa Array với ArrayList

Array	ArrayList
1) Kích thước **cố định**.	Kích thước có thể **thay đổi được**.
2) Có thể lưu trữ dữ liệu kiểu **nguyên thủy** và **đối tượng**.	Chỉ có thể lưu trữ dữ liệu kiểu **đối tượng**.
3) Tốc độ lưu trữ và thao tác **nhanh hơn**.	Tốc độ lưu trữ và thao tác **chậm hơn**.
4) Chỉ có thuộc tính **length**.	Có nhiều phương thức để thao tác với dữ liệu.

Chuyển đổi Array ⇆ ArrayList

- ArrayList → Array: sử dụng phương thức **toArray**

```
ArrayList<String> arrayList = new ArrayList<String>();
arrayList.add(...);
String[] array = arrayList.toArray(new String[arrayList.size()]);
```

- Array → ArrayList: sử dụng phương thức **asList** của lớp **Arrays**

```
List<String> arrayList = new ArrayList<String>();
arrayList = Arrays.asList(array);
```

Lớp Vector trong java

- Lớp Vector tương tự như ArrayList nhưng được đồng bộ.

- Các phương thức khởi tạo

Constructor	Mô tả
Vector()	Khởi tạo một vector trống.
Vector(Collection c)	Khởi tạo một vector với các phần tử của collection c.
Vector(int size)	Khởi tạo một vector có kích thước ban đầu được chỉ định.
Vector(int size, int incr)	Khởi tạo một vector có kích thước ban đầu được chỉ định, với giá trị tăng kích thước incr

Một số phương thức lớp Vector

- void addElement(Object element)
- int capacity()
- int size()
- boolean contains(Object element)
- Object elementAt(int index)
- Object firstElement()
- Object lastElement()
- Object get(int index)
- boolean isEmpty()
- boolean removeElement(Object element)
- boolean removeAll(Collection c)
- void setElementAt(Object element, int index)

So sánh giữa ArrayList với Vector

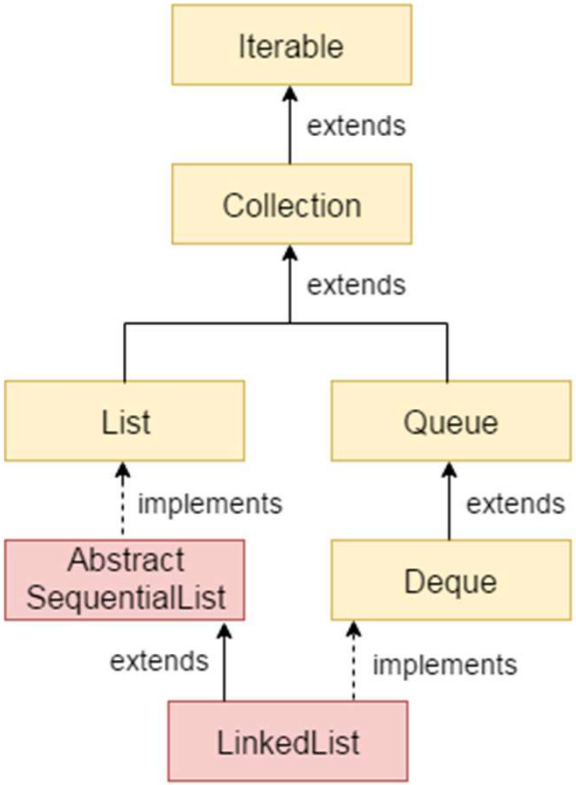
ArrayList	Vector
1) ArrayList là không đồng bộ	Vector là đồng bộ
2) ArrayList tăng 50% kích thước hiện tại nếu số phần tử vượt quá khả năng chứa của nó.	Vector tăng 100% nghĩa là tăng gấp đôi kích thước hiện tại nếu số phần tử vượt quá khả năng chứa của nó.
4) ArrayList là **nhanh hơn** vì nó là không đồng bộ.	Vector là **chậm hơn** ví nó là đồng bộ. Tức là, trong môi trường đa luồng, các thread giữ nó ở trong trạng thái runnable hoặc non-runnable cho đến khi thread hiện tại giải phóng đối tượng đó.
5) ArrayList sử dụng Iterator để duyệt các phần tử.	Vector sử dụng Enumeration và Iterator để duyệt các phần tử.

Lớp LinkedList trong java

❑ **Lớp LinkedList trong java** sử dụng cấu trúc danh sách liên kết đôi để lưu trữ các phần tử.

❑ **Những đặc điểm của LinkedList**:

- Lớp LinkedList có thể chứa các phần tử trùng lặp.
- Lớp LinkedList duy trì thứ tự của phần tử được thêm vào.
- Lớp LinkedList là không đồng bộ (non-synchronized).
- Lớp LinkedList thao tác nhanh vì không cần phải dịch chuyển nếu bất kỳ phần tử nào bị xoá.
- Lớp LinkedList có thể được sử dụng như list (danh sách), stack (ngăn xếp) hoặc queue (hàng đợi).



Lớp LinkedList trong java

- Các phương thức khởi tạo

Constructor	Mô tả
LinkedList()	Khởi tạo một LinkedList trống.
LinkedList(Collection c)	Khởi tạo một LinkedList chứa các phần tử của collection được chỉ định.

```
LinkedList<String> linkedList1 = new LinkedList<String>();
LinkedList<String> linkedList2 = new LinkedList<>(linkedList1);
```

Một số phương thức lớp LinkedList

Phương thức	Mô tả
boolean **add(Object o)**	Thêm phần tử đã cho vào cuối.
void **add(int index, Object element)**	Chèn element tại index.
void **addFirst(Object o)**	Chèn phần tử đã cho vào đầu tiên.
void **addLast(Object o)**	Thêm phần tử đã cho vào cuối cùng.
int **size()**	Trả về số phần tử.
boolean **contains(Object o)**	Trả về true nếu list này chứa phần tử đã cho.
boolean **remove(Object o)**	Gỡ bỏ phần tử tại vị trí đã cho.
Object **getFirst()**	Trả về phần tử đầu tiên.
Object **getLast()**	Trả về phần tử cuối cùng.
int **indexOf(Object o)**	Trả về chỉ số đầu tiên của phần tử đã cho, hoặc -1 nếu List không chứa phần tử này.
int **lastIndexOf(Object o)**	Trả về chỉ số cuối cùng của phần tử đã cho, hoặc -1 nếu List không chứa phần tử này.

Lớp HashSet trong java

- **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.
- **Những đặc điểm của HashSet**:
  - 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**

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

Lớp HashSet trong java

- Các phương thức khởi tạo
  - **HashSet()**
  - **HashSet(Collection c)**
  - **HashSet(int capacity)**

```
HashSet<String> set0 = new HashSet<String>();
HashSet<String> set1 = new HashSet<String>(set0);
HashSet<String> set2 = new HashSet<String>(10);
```

Một số phương thức lớp HashSet

- boolean add(Object o)
- void clear()
- boolean contains(Object o)
- boolean isEmpty()
- boolean remove(Object o)
- Object clone()
- Iterator iterator()
- int size()

Lớp HashMap trong java

❑ **Những đặc điểm của HashMap**:

- HashMap lưu trữ dữ liệu dưới dạng cặp < **key, 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 bổ sung.

❑ **Sự khác nhau giữa HashSet và HashMap**

HashSet chỉ chứa giá trị (value) trong khi HashMap chứa cặp key và value.

Lớp HashMap trong java

- Các phương thức khởi tạo cơ bản
  - **HashMap()**
  - **HashMap(Map m)**
  - **HashSet(int capacity)**

```
HashMap<Integer, String> hashMap0 = new HashMap<Integer, String>();
HashMap<Integer, String> hashMap1 = new HashMap<>(hashMap0);
HashMap<Integer, String> hashMap2 = new HashMap<Integer, String>(10);
```

Một số phương thức lớp HashMap

- Object put(Object key, Object value)
- void clear()
- boolean containsKey(Object key)
- boolean containsValue(Object value)
- boolean isEmpty()
- Object clone()
- Set entrySet()
- Set keySet()
- int size()
- Collection values()

Lớp Hashtable trong java

❑ **Những đặc điểm của Hashtable**:

- Tương tự như HashMap, nhưng nó được đồng bộ.
- Hashtable lưu trữ dữ liệu dưới dạng cặp < **key, value** >.
- Nó chứa các **key duy nhất**.
- **Nó KHÔNG chứa 1 key hay giá trị null nào**.
- Nó duy trì các phần tử KHÔNG theo thứ tự bổ sung.

Lớp Hashtable trong java

- Các phương thức khởi tạo cơ bản
  - **Hashtable()**
  - **Hashtable(Map m)**
  - **Hashtable(int capacity)**

```
Map<Integer, String> map = new HashMap<>();

Hashtable<Integer, String> hashTable0 = new Hashtable<Integer, String>();
Hashtable<Integer, String> hashTable1 = new Hashtable<Integer, String>(10);
Hashtable<Integer, String> hashTable2 = new Hashtable<Integer, String>(map);
```

Một số phương thức lớp Hashtable

- Object put(Object key, Object value)
- void clear()
- boolean contains(Object value)
- boolean containsValue(Object value)
- boolean containsKey(Object key)
- boolean isEmpty()
- Object get(Object key)
- Object remove(Object key)
- int size()

So sánh giữa HashMap với 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**.
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**.
8) HashMap kế thừa lớp **AbstractMap**.	Hashtable kế thừa lớp **Dictionary**.

Q&A