**Java Core**

[1. Java.util 3](#_Toc476476847)

[1.1. Iterator (Java.util.Iterator) 3](#_Toc476476848)

[1.2. Collections Framework (Nền tảng các tập hợp) 4](#_Toc476476849)

[1.3. List (Interface List<E>) 4](#_Toc476476850)

[1.3.1. ArrayList (java.util.ArrayList) 5](#_Toc476476852)

[1.3.2. LinkedList (java.util.LinkedList) 6](#_Toc476476856)

[1.3.3. Vector (java.util.Vector) 7](#_Toc476476857)

[1.3.4. Stack (java.util.Stack) 8](#_Toc476476858)

[1.4. Map (Interface Map<K, V>) 9](#_Toc476476859)

[1.4.1. HashMap 10](#_Toc476476860)

[1.4.2. HashTable 11](#_Toc476476861)

[1.4.3. So sánh giữa HashTable và HashMap 12](#_Toc476476862)

[1.4.4. LinkedHashMap 12](#_Toc476476863)

[1.4.5. StortedMap 13](#_Toc476476864)

[1.4.6. TreeMap 13](#_Toc476476865)

[1.5. Set (Interface Set<E>) 14](#_Toc476476866)

[1.5.1. HashSet 15](#_Toc476476867)

[1.5.2. TreeSet 15](#_Toc476476868)

[1.5.3. EnumSet 16](#_Toc476476869)

[1.5.4. LinkedHashSet 16](#_Toc476476870)

[1.6. Sự khác nhau giữa các Collection 17](#_Toc476476871)

[1.6.1. List, Set và Map 17](#_Toc476476872)

[1.7. Date (java.util.Date) 17](#_Toc476476873)

[1.8. Calendar (java.util.Calendar) 18](#_Toc476476874)

[1.9. Enum 20](#_Toc476476875)

[1.10. Properties (Class Proterties) 20](#_Toc476476876)

[1.11. Rescource Bundle 21](#_Toc476476877)

[2. Java.io 21](#_Toc476476878)

[2.1. Stream 22](#_Toc476476879)

[2.2. File (Java.IO.File) 22](#_Toc476476880)

[2.3. FileInputStream 24](#_Toc476476881)

[2.4. FileOutputStream 25](#_Toc476476882)

[2.5. Buffered IO 25](#_Toc476476883)

[2.5.1. Buffered Input 26](#_Toc476476884)

[2.5.2. Buffered Ouput 26](#_Toc476476885)

[2.6. PrintWriter (Java.IO.PrintWriter) 26](#_Toc476476886)

[2.7. Interface (java.IO.Interface) 27](#_Toc476476887)

[2.7.1. DataInput Interface 27](#_Toc476476888)

[2.7.2. Serializable Interface 27](#_Toc476476889)

[2.8. Lớp Exeption thường gặp 27](#_Toc476476890)

[3. Java.lang 28](#_Toc476476891)

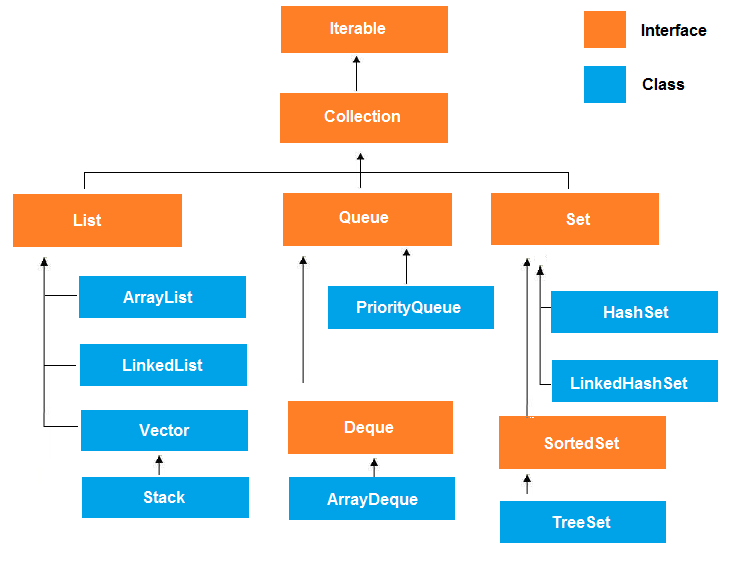
[3.1. StringBuffer 28](#_Toc476476892)

[3.2. StringBuilder 29](#_Toc476476893)

[3.3. So sánh String, StringBuffer, StringBuilder 30](#_Toc476476894)

[TÀI LIỆU THAM KHẢO 31](#_Toc476476895)

1. Java.util

Java Until là gói chứa các collections framework, cung cấp nhiều lớp và nhiều giao diện tiện ích thường xuyên trong tất cả các loại ứng dụng như: Date, Dictionary, BitSet, Hashtable, Properties, Random, Stack, String Tokenizer, Vector.

Iterator (Java.util.Iterator)

**Iterator** giống như một máy lặp để lấy dữ liệu, cách truy cập lần lượt từ phần tử này đến phần tử khác.

**Nhóm Collection** cũng có thể truy cập theo kiểu lần lượt bằng cách gọi method iterator() để lấy được đối tượng Iterator.

* **java.util.Collection** mở rộng từ interface java.lang.Iterable (có thể lặp được) do đó nó thừa kế phương thức public Iterator<E> iterator(), bộ lặp Iterator sử dụng để duyệt trên các phần tử của Collection.

Link Demo:

<https://github.com/ThuHien25/JavaTraining/tree/master/JavaCore/DemoJavaCore/src/JavaUtil>

Collections Framework (Nền tảng các tập hợp)

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

|  |  |  |
| --- | --- | --- |
| **java.util.Queue** | **java.util.List** | **java.util.Set** |
| Cho phép chứa các phần tử trùng lặp | Cho phép chứa các phần tử trùng lặp | Không cho phép chứa các phần tử trùng lặp |
| Không cho phép chứa các phần tử null | Cho phép chứa nhiều phần tử null | Tùy theo class thi hành Set hỗ trợ chứa phần tử null hay không .Nếu có hỗ trợ thì chỉ chứa nhiều nhất 1 phần tử null nếu có. |

List (Interface List<E>)

**List** là một interface dùng chứa **danh sách liên tục,** nó có đầy đủ các tính năng của Collection đồng thời có thêm một số tính chất đặc biệt:

* Cho phép phần tử trùng lặp.
* Cho phép **nhiều** phần tử **null.**
* Là một tập hợp có tuần tự.
* Một số method trong List

|  |  |  |
| --- | --- | --- |
| **STT** | **Method** | **Mô tả** |
|  | Int size(); | Trả về số các phần tử trong List |
|  | boolean isEmpty(); | Trả về true nếu List này không chứa phần tử nào |
|  | boolean contains  (Object o); | Kiểm tra xem List có chứa Object o không. |
|  | Iterator<E> iterator(); | Trả về một iterator có các phần tử trong List có trình tự hợp lý |
|  | Object[] toArray();  <T> T[] toArray(T[] a); | Chuyển List thành mảng các đối tượng |
|  | void add(E e) | Thêm một phần tử vào List |
|  | void add  (int index, E element) | Thêm một phần tử vào List tại vị trí index |
|  | Object remove(Object o) | Xóa một phần tử khỏi List |
|  | void clear() | Xóa tất cả các phần tử trong List |
|  | Object get(int index) | Lấy ra phần tử của List tại vị trí index |
|  | Object set  (int index, E element) | Thay thế phần tử tại vị trí index trong List với phần tử mới |
|  | lastIndexOf(Object o) | Trả về index cuối cùng của object trong List |
|  | int indexOf(Object obj) | Trả về vị trí đầu tiên của object trong List |

Những class hiện thực List

* + 1. ArrayList (java.util.ArrayList)
* **Array** có những nhược điểm:
  + Phải xác định trước số lượng phần tử trong mảng.
  + Nếu khai báo kích thước mảng quá nhỏ thì sẽ dẫn đến thiếu bộ nhớ.
  + Nếu khai báo quá lớn thì lại lãng phí bộ nhớ.
  + Các phần tử trong mảng phải có cùng kiểu dữ liệu.
* **ArrayList** đã khắc phục được những nhược điểm này. ArrayList là một kiểu mảng động, nếu các phần tử thêm vào vượt quá kích cỡ mảng, mảng sẽ tự động tăng kích cỡ.
* **Đặc điểm:** 
  + Quản lý một dãy các đối tượng.
  + Có thể tăng, giảm kích thước theo nhu cầu.
* **Constructor**

|  |  |  |
| --- | --- | --- |
| **STT** | **Method** | **Mô tả** |
|  | ArrayList() | Tạo 1 ArrayList rỗng. |
|  | ArrayList  (Collection c) | Tạo một ArrayList mà được khởi tạo với các phần tử của collection c. |
|  | ArrayList  (int capacity) | Tạo một ArrayList với kích thước của mảng được xác định, kích thước tự động tăng khi thêm các phần tử. |

* **Một số method trong ArrayList**

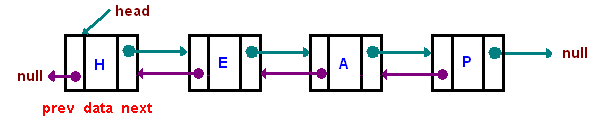
|  |  |  |
| --- | --- | --- |
| **STT** | **Method** | **Mô tả** |
|  | int Size() | Trả về số lượng phần tử có trong danh sách |
|  | void add(int index, Object o) | Thêm một phần tử mới vào vị trí index. |
|  | void add(Object o) | Thêm một phần tử mới vào vị trí cuối cùng của danh sách |
|  | boolean addAll(Collection c) | Thêm tất cả các phần tử của Collection c vào ArrayList |
|  | boolean addAll(int index, Collection c) | Thêm tất cả các phần tử của Collection C vào ArrayList bắt đầu từ vị trí index |
|  | void clear() | Xóa toàn bộ các phần tử có trong trong danh sách. |
|  | Object clone() | Tạo một ArrayList mới như ArrayList đã cho |
|  | Object remove  (int index) | Xóa một phần tử tại vị trí index |
|  | boolean contains(Object o) | Trả về true nếu object nằm trong ArrayList |
|  | Object get(int index) | Trả về phần tử ở vị trí **index trong danh sách.** |
|  | int indexOf(Object o) | Trả về vị trí index của Object **o** được tìm thấy **đầu tiên** trong danh sách. Trả về **-1** nếu không tìm thấy. |
|  | int lastIndexOf  (Object o) | Trả về vị trí index **cuối cùng** mà Object o xuất hiện trong danh sách, trả về **-1** nếu không tìm thấy. |
|  | Object set(int index, Object o) | Sửa giá trị tại ví trí index. |
|  | Object[] toArray() | Trả vể một mảng bao gồm các đối tượng có trong danh sách với thứ tự đúng như trên ArrayList. |

* Link đọc thêm:

<https://www.tutorialspoint.com/java/util/java_util_arraylist.htm>

* Link Demo:

<https://github.com/ThuHien25/JavaTraining/tree/master/JavaCore/DemoJavaCore/src/JavaUtil>

* + 1. LinkedList (java.util.LinkedList)

**LinkedList:** là danh sách liên kết 2 chiều. Hỗ trợ thao tác trên đầu và cuối danh sách. LinkedList giúp tiết kiệm bộ nhớ so với mảng trong các bài toán xử lý danh sách.

* + Truy nhập trên LinkedList luôn phải tuần tự.
  + Có thể sử dụng như một **list, stack hay queue**

**ArrayList** tìm kiếm phần tử **nhanh** hơn **LinkedList.**

**Constructor:**

LinkedList()

LinkedList(Collection c)

**Một số method đặc trưng trong LinkedList**

|  |  |  |
| --- | --- | --- |
| **STT** | **Method** | **Mô tả** |
|  | E getFirst() | Trả về phần tử ở vị trí đầu tiên trong danh sách. |
|  | E getLast() | Trả về phần tử ở vị trí cuối cùng trong danh sách. |
|  | E addFirst() | Thêm phần tử vào đầu danh sách |
|  | E addLast() | Thêm phần tử vào cuối danh sách |
|  | E removeFirst() | Xóa phần tử đầu tiên trong danh sách |
|  | E removeLast() | Xóa phần tử cuối cùng trong danh sách |

* Link đọc thêm:

<https://www.tutorialspoint.com/java/util/java_util_linkedlist.htm>

* Link Demo:

<https://github.com/ThuHien25/JavaTraining/tree/master/JavaCore/DemoJavaCore/src/JavaUtil>

* + 1. Vector (java.util.Vector)

Class **Vector** có tính năng tương tự như **ArrayList**. Khác biệt là các method của Vector được **đồng bộ hóa** vì vậy sử dụng tốt trong các ứng dụng đa luồng *(Multiple Thread).*

**Constructor**

Vector() // Mặc định và có kích thước khởi tạo là 10

Vector(int size) //size là dung lượng của vector

Vector(int size, int incr) // tự động tăng dung lượng thêm “resize” nếu số phần tử vượt qua dung lượng hiện tại

Vector(Collection c)

**Một số method đặc trưng trong Vector**

|  |  |  |
| --- | --- | --- |
| **STT** | **Method** | **Mô tả** |
|  | void addElement  (E Object) | Thêm phần tử được chỉ định vào lớp Vector. |
|  | int capacity() | Trả về dung lượng hiện thời của lớp Vector. |
|  | void copyInto  (Object[] anArray) | Sao chép các phần tử của lớp Vector vào mảng được chỉ định. |
|  | E elementAt(int index) | Lấy phần tử vị trí được chỉ định. |
|  | void ensureCapacity  (int minCapacity) | Đảm bảo rằng lớp Vector có thể lưu trữ ít nhất dung lượng tối thiểu được chỉ định. |
|  | E firstElement() | Trả về phần tử đầu tiên trong lớp Vector. |
|  | void insertElementAt  (E Object, int index) | Chèn đối tượng được chỉ định tại vị trí được chỉ định. |
|  | boolean isEmpty() | Trả về True nếu lớp Vector không có phần tử. |
|  | E lastElement() | Trả về phần tử cuối cùng trong lớp Vector. |
|  | void removeAllElements() | Xoá tất cả các phần tử từ lớp Vector. |
|  | boolean removeElement  (Object obj) | Xoá đối tượng được chỉ định từ lớp Vector. |
|  | void removeElementAt  (int index) | Xoá đối tượng tại chỉ mục được chỉ định. |
|  | setElementAt  (Object, int) | Thay thế đối tượng tại chỉ mục được chỉ định với đối tượng được chỉ định. |
|  | void setSize  (int newSize) | Thiết lập kích thước của lớp Vector thành kích thước mới được chỉ định. |
|  | void trimToSize() | Định lại kích thước của lớp Vector để di chuyển dung lượng thừa trong nó. |

* Link đọc thêm:

<https://www.tutorialspoint.com/java/util/java_util_vector.htm>

* Link Demo:

<https://github.com/ThuHien25/JavaTraining/tree/master/JavaCore/DemoJavaCore/src/JavaUtil>

* + 1. Stack (java.util.Stack)

**Stack** là một cấu trúc dữ liệu lưu trữ nhiều phần tử dữ liệu. Stack hoạt động theo cơ chế vào sau ra trước **Last In/First Out (LIFO).**

Lớp Stack bao gồm tất cả phương thức được định nghĩa bởi lớp Vector, và một số phương thức khác của riêng nó.

**Một số method đặc trưng trong Stack**

|  |  |  |
| --- | --- | --- |
| **STT** | **Method** | **Mô tả** |
|  | E Push(E item) | Thêm 1 phần tử vào đỉnh Stack |
|  | E Pop() | Lấy 1 phần tử từ đỉnh Stack |
|  | E Peek() | Trả về phần tử đầu tiên của Stack mà không loại bỏ nó ra khỏi Stack |
|  | Boolean Empty() | Kiểm tra Stack có rỗng ko? |
|  | int Search  (Object o) | Tìm kiếm phần tử trong Stack. Trả về vị trí phần tử trong Stack tính từ đỉnh stack nếu ko thấy trả về -1 |

* Link đọc thêm:

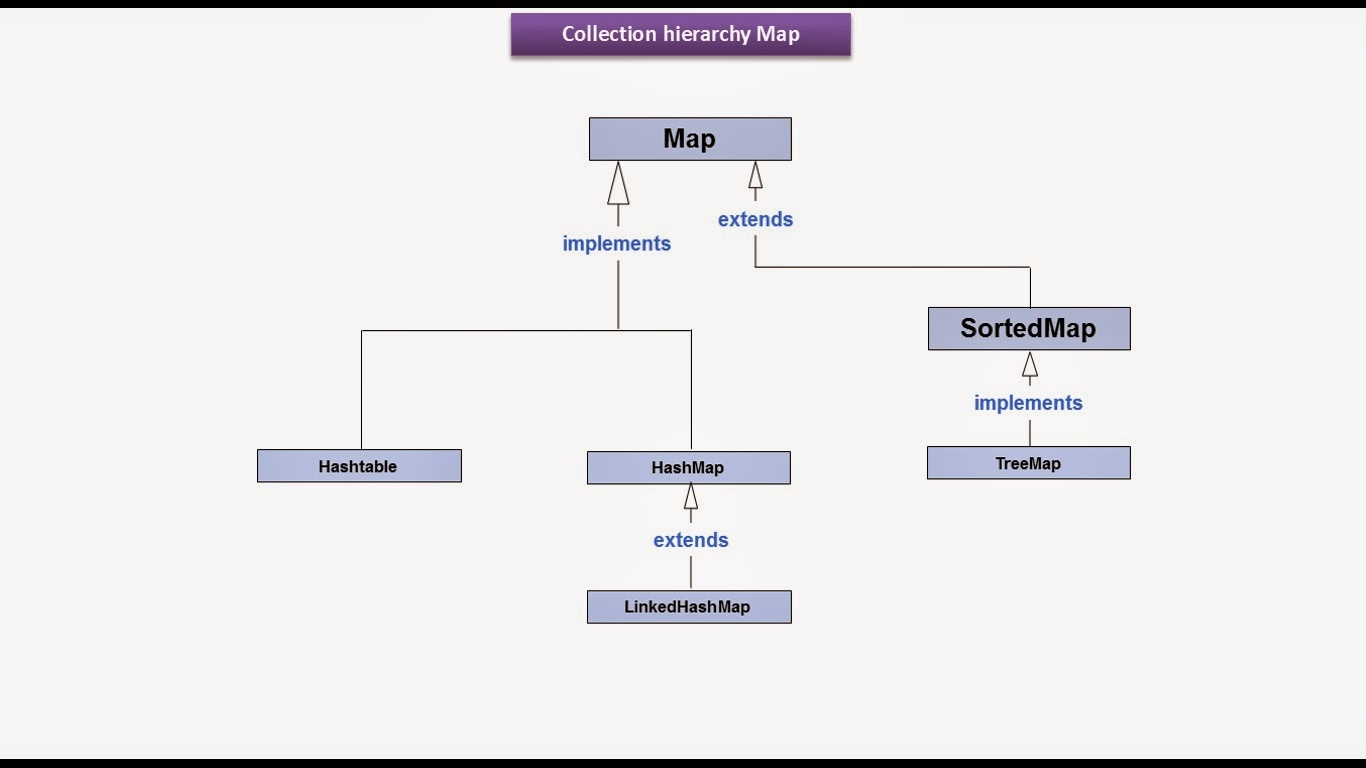
<https://www.tutorialspoint.com/java/util/java_util_stack.htm>

* Link Demo:

<https://github.com/ThuHien25/JavaTraining/tree/master/JavaCore/DemoJavaCore/src/JavaUtil>

Map (Interface Map<K, V>)

**Map** lưu trữ các cặp **key/value**:

* + Các cặp key/value chứa trong Map **không trùng nhau**.
  + **Nếu biết key có thể lấy ra giá trị value trong Map ứng với key này

**Một số phương thức trong Map**

|  |  |  |
| --- | --- | --- |
| **STT** | **Method** | **Mô tả** |
|  | clear( ) | Xóa bỏ tất cả các key/value trong map |
|  | set keySet() | Trả về các key |
|  | Collection values() | Trả về các values |
|  | set entrySet() | Trả về các cặp key-values |
|  | Int size() | Trả về số các cặp key/value của map |
|  | boolean containsValue  (Object value) | Trả về true nếu một hoặc nhiều key của map chứa value |
|  | boolean containsKey  (Object k) | Trả về true nếu k là một key của map |
|  | boolean equals  (Object obj) | So sánh điểm chung giữa obj với map. Nếu giống thì trả về true |
|  | Object get  (Object k) | Trả về value có key là k |
|  | int hashCode( ) | Trả về hash code cho Map đang gọi |
|  | boolean isEmpty( ) | Trả về true nếu map rỗng |
|  | Object put(Object k, Object v) | Ghi đè giá trị v lên vị trí có key là k. Nếu key không tồn tại thì sẽ trả về Null |
|  | putAll(Map m) | Đặt tất cả các entry của m vào map |
|  | Object remove  (Object k) | Xóa key k |

Những class hiện thực Map

* + 1. HashMap

Lớp **HashMap** sử dụng *hashtable (bảng băm)* để triển khai **Map Interface.** Nó cho phép chúng ta lưu giá trị theo cặp key/value, trong đó key là duy nhất.

* + Thứ tự của các đối tượng trong HashMap sẽ bị thay đổi theo thời gian.
  + HashMap cho phép null values và null key.

**Constructor**

HashMap() //mặc định capacity: 16, load factor: 0.75

HashMap(Map m) //tạo ra 1 HashMap mới từ 1 Map khác

HashMap(int capacity) // capacity: dung lượng của HashMap

HashMap(int capacity, float loadFactor) // loadFactor là hệ số tải

**Method** của HashMap implements từ Map interface và không có method riêng.

* Link đọc thêm:

<https://www.tutorialspoint.com/java/util/java_util_hashmap.htm>

* Link Demo:

<https://github.com/ThuHien25/JavaTraining/tree/master/JavaCore/DemoJavaCore/src/JavaUtil>

* + 1. HashTable

Lớp **HashTable** implements từ Map interface. Có 2 giá trị là **key/value**. Tương tự như HashMap nhưng được đồng bộ.

* + Trong HashTable, key và value không được null.
  + Key được sắp xếp theo thứ tự giảm dần.

**Constructor**

Hashtable()

Hashtable(Map m)

Hashtable(int size)

Hashtable(int size, float fillRatio)

**Một số phương thức trong HashTable**

|  |  |  |
| --- | --- | --- |
| **STT** | **Method** | **Mô tả** |
|  | clear() | Xóa bỏ tất cả các key/value trong map |
|  | Object clone() | Tạo một bản sao của map |
|  | boolean containsValue  (Object value) | Trả về true nếu một hoặc nhiều key của map chứa value |
|  | boolean containsKey  (Object k) | Trả về true nếu k là một key của map |
|  | Enumeration elements() | Trả về một bản liệt kê các value trong map |
|  | Enumeration<V> keys() | Trả về một bản liệt kê các key được chứa trong hash table |
|  | Object get(Object k) | Trả về value có key là k |
|  | Object put(Object k, Object v) | Ghi đè giá trị v lên vị trí có key là k. Nếu key không tồn tại thì sẽ trả về Null |
|  | Object remove(Object k) | Xóa key k |
|  | void rehash() | Tăng kích cỡ của hash table và làm lại mới tất cả các key của nó |
|  | String toString() | Trả về một chuỗi tương đương với hashtable |

* Link đọc thêm:

<https://www.tutorialspoint.com/java/util/java_util_hashtable.htm>

* Link Demo:

<https://github.com/ThuHien25/JavaTraining/tree/master/JavaCore/DemoJavaCore/src/JavaUtil>

* + 1. So sánh giữa HashTable và HashMap

|  |  |  |
| --- | --- | --- |
|  | **HashTable** | **HashMap** |
| **Khả năng đồng bộ** | HashTable đồng bộ **(synchronized)**. HashTable chỉ cho phép tối đa một thread truy cập và xử lý dữ liệu tại một thời điểm. | HashMap không đồng bộ **(non-synchronized).** HashMap có thể cho phép có nhiều hơn một thread cùng truy cập và xử lý dữ liệu tại một thời điểm. |
| **Key/Value** | HashTable **không** **cho** phép key/value **NULL**. | HashMap **cho** phép key/value **NULL**. |

* + 1. LinkedHashMap

**LinkedHashMap** cũng tương tự như **HashMap** nhưng có sử dụng danh sách liên kết, các item được liên kết với nhau và có thứ tự. Tuy nhên, LinkedHashMap truy cập item **chậm hơn** so mới HashMap.

**Constructor**

LinkedHashMap()

LinkedHashMap(int capacity)

LinkedHashMap(int capacity, float loadFactor)

LinkedHashMap(int capacity, float loadFactor, boolean accessOder)

LinkedHashMap(Map m)

**Một số phương thức trong** **LinkedHashMap**

|  |  |  |
| --- | --- | --- |
| **STT** | **Method** | **Mô tả** |
|  | void clear | Xóa bỏ tất cả các key/value trong map. |
|  | boolean containsValue  (Object value) | Trả về true nếu một hoặc nhiều key của map chứa value. |
|  | Object get(Object k) | Trả về value có key là k. |
|  | [protected boolean removeEldestEntry(eldest)](https://www.tutorialspoint.com/java/util/linkedhashmap_removeeldestentry.htm) | Trả về true nếu map nên gỡ bỏ các entry cũ nhất của nó. |

* Link đọc thêm:

https://www.tutorialspoint.com/java/util/java\_util\_linkedhashmap.htm

* Link Demo:

<https://github.com/ThuHien25/JavaTraining/tree/master/JavaCore/DemoJavaCore/src/JavaUtil>

* + 1. StortedMap

**Interface SortedMap** là interface con của **Map**. Nó đảm bảo rằng các cặp key/value được sắp xếp theo thứ tự **tăng dần theo key.**

Chỉ có một class trong gói java.util thi hành interface SortedMap, đó là **TreeMap.**

* + 1. TreeMap

**Constructor**

TreeMap()

TreeMap(Map m)

TreeMap(Comparator comp)

TreeMap(SortedMap m)

**Một số phương thức trong** **TreeMap**

|  |  |  |
| --- | --- | --- |
| **STT** | **Method** | **Mô tả** |
|  | Comparator comparator() | Trả về bộ so sánh (**Comparator**) sử dụng để sắp xếp. Nếu thứ tự tự nhiên được sử dụng cho map (keyimplements **Comparable**), **null**   sẽ được trả về. |
|  | Object firstKey() | Trả về key đầu tiên trong map. |
|  | SortedMap headMap  (Object end) | Trả về một SortedMap con, với các cặp **key/value** mà **key** được đánh giá nhỏ hơn hoặc bằng **end** |
|  | Object lastKey() | Trả về key cuối cùng. |
|  | SortedMap subMap  (Object start, Object end) | Trả về một SortedMap con, chứa các cặp **key/value** mà key được đánh giá lớn hơn hoặc bằng **start** và nhỏ hơn hoặc bằng **end**. |
|  | SortedMap tailMap  (Object start) | Trả về một **SortedMap** con, chứa các cặp **key/value** mà key của nó được đánh giá lớn hơn hoặc bằng **start**. |
|  | ceilingEntry(K key) | Trả về một key/value. Nếu key không tồn tại thì trả về null |
|  | ceilingKey(K key) | Trả về key |
|  | descendingKeySet() | Trả về list key |
|  | descendingMap() | Trả về một map |
|  | floorEntry(K key) | Trả về key và value của key hoặc null nếu không chứa key |
|  | K floorKey(K key) | Trả về key hoặc null nếu không chứa key |
|  | lowerEntry(K key) | Trả về một key và value hoặc null hếu như Map không chứa key |
|  | pollFirstEntry() | Lấy và xóa đi phần tử đầu tiên của map hoặc null nếu như map rỗng |
|  | pollLastEntry() | Lấy và xóa đi phần tử cuối cùng của map hoặc null nếu như map rỗng |

* Link đọc thêm:

<https://www.tutorialspoint.com/java/util/java_util_treemap.htm>

* Link Demo:

<https://github.com/ThuHien25/JavaTraining/tree/master/JavaCore/DemoJavaCore/src/JavaUtil>

Set (Interface Set<E>)

**Set** là một Interface con của Collection, nó có đầy đủ các tính năng của Collection, và có thêm một số tính năng:

* Các phần tử trong set là không trùng lặp
* Cho phép chứa tối đa 1 phần tử null trong Set
* Nếu thêm 1 phần tử đã tồn tại vào Set thì Set vẫn chỉ chứa 1 phần tử mà thôi.

**Một số phương thức trong Set**

|  |  |  |
| --- | --- | --- |
| **STT** | **Method** | **Mô tả** |
|  | add(E e) | Thêm một phần tử vào Collection |
|  | clear() | Xóa tất cả các phần tử của Collection |
|  | contains(Object o) | Trả về true nếu như Collection chứa object đó |
|  | iterator() | Trả về một iterator khác chứa các phần tử của Set |
|  | remove(Object o) | Xóa object khỏi Set |
|  | size() | Trả về dung lượng của Set |
|  | isEmpty() | Trả về true nếu Collection không có phần tử nào |
|  | toArray(T[] a) | Trả về một mảng T chứa tất cả các phần tử của Set |
|  | equals(Object o) | So sánh các đối tượng trong Set. |
|  | retainAll(Collection<?> c) | Chỉ giữ lại phần tử trong Set mà collection chứa |
|  | toArray() | Trả về một mảng chứa các phần tử của Set |
|  | addAll(Collection<? extends E> c) | Thêm tất cả các phần tử của collection vào Set |

Tóm tắt một cách ngắn gọn, nếu ưu tiên về **tốc độ** sử dụng **HashSet,** nếu cần một Set được **sắp xếp** chọn **TreeSet,** còn nếu muốn đọc một Set **theo thứ tự** mà các phần tử được insert vào thì dùng **LinkedHashSet.**

Những class hiện thực Set

* + 1. HashSet

Là 1 class implement Set Interface, mà các phần tử được lưu trữ dưới dạng **bảng băm (hash table).**

* + Các item trong HashSet sẽ không trùng lặp, không được sắp xếp trật tự.
  + Mỗi phần tử được lưu trữ và truy xuất thông qua các mã băm của nó.

Đây cũng là loại Set có tốc độ cao nhất.

Hầu hết các Method trong HashSet được cung cấp bởi các superclasses là AbstractCollection và AbstractSet

* Link đọc thêm:

<https://www.tutorialspoint.com/java/util/java_util_hashset.htm>

* Link Demo:

<https://github.com/ThuHien25/JavaTraining/tree/master/JavaCore/DemoJavaCore/src/JavaUtil>

* + 1. **TreeSet**

Là 1 class thực thi giao diện **SortedSet Interface,** trong đó các phần tử trong set đã được sắp xếp theo thứ tự **tăng dần.**

Thời gian truy cập và thu nhận dữ liệu là khá nhanh, vì thế TreeSet một lựa chọn tuyệt vời khi **lưu giữ một lượng lớn thông tin đã xếp thứ tự** mà phải được tìm kiếm một cách nhanh chóng.

**Một số phương thức đặc trưng của SortedSet**

|  |  |  |
| --- | --- | --- |
| **STT** | **Method** | **Mô tả** |
|  | **comparator()** | Sắp xếp các phần tử trong Set. |
|  | **first()** | Phần tử đầu tiên (nhỏ nhất theo thứ tự sắp xếp) của Set hiện tại |
|  | **last()** | Phần tử cuối cùng (lớn nhất theo thứ tự sắp xếp) của Set hiện tại |
|  | **headSet(E toElement)** | Các phần tử nằm trước (nhỏ hơn) toelement trong Set hiện tại theo thứ tự sắp xếp |
|  | **tailSet(E fromElement)** | Các phần tử nằm sau (lớn hơn) fromelement trong Set hiện tại  theo thứ tự sắp xếp |
|  | **subSet(E fromElement, E toElement)** | Các phần tử bắt đầu từ (lớn hơn hoặc bằng) fromelement và đứng trước (nhỏ hơn) toelement trong sortedset hiện tại |

* Link đọc thêm:

<https://www.tutorialspoint.com/java/util/java_util_treeset.htm>

* Link Demo:

<https://github.com/ThuHien25/JavaTraining/tree/master/JavaCore/DemoJavaCore/src/JavaUtil>

* + 1. EnumSet

**EnumSet** cũng là 1 class thực thi giao diện Set Interface, tuy nhiên thay vì việc các phần tử trong Set là các đối tượng của một lớp nào đó thì các phần tử trong EnumSet lại các giá trị Enum.

**Constructor**

Set<myEnum> s = Collections.synchronizedSet(EnumSet.noneOf(MyEnum.class));

* Link đọc thêm:

https://www.tutorialspoint.com/java/util/java\_util\_enumset.htm

* Link Demo:

<https://github.com/ThuHien25/JavaTraining/tree/master/JavaCore/DemoJavaCore/src/JavaUtil>

* + 1. LinkedHashSet

**LinkedHashSet** là sự kết hợp giữ bảng băm (Hash table) và danh sách liên kết (linked list) để thực thi giao diện Set Interface. Tương tự HashSet nhưng có kèm theo danh sách liên kết 2 chiều.

* Link đọc thêm:

<https://www.tutorialspoint.com/java/util/java_util_linkedhashset.htm>

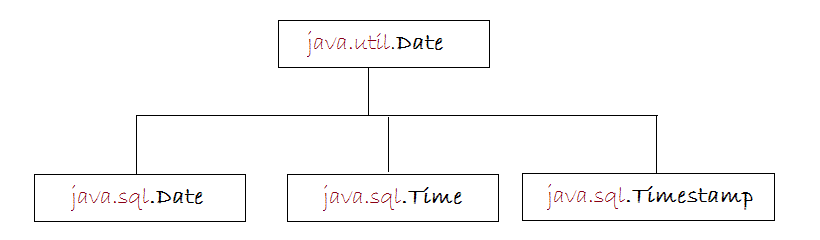
Sự khác nhau giữa các Collection

* + 1. List, Set và Map

**List** cung cấp method để chèn và xóa các item tại một điểm bất kì trong danh sách, các item được truy cập và tìm kiếm các phần tử trong danh sách theo chỉ mục (index). Danh sách có thể chứa các item **trùng nhau.**

**Set** hỗ trợ các thao tác xử lý trên collection kiểu tập hợp, các phần tử trong Set **không trùng nhau.**

**Map** cung cấp các thao tác xử lý trên các bảng ánh xạ, các phần tử được lưu trữ theo khóa và 2 khóa **không được trùng nhau.**

Date (java.util.Date)

**java.util.Date** mô tả cho ngày tháng năm và thời gian. Tiếc là hầu hết các method của nó đã bị lỗi thời, khuyến cáo là không nên sử dụng các method đó, tuy nhiên class Date vẫn được sử dụng rộng rãi.

**Constructor**

//Tạo một đối tượng Date mô tả thời điểm hiện tại

Date date1 = new Date()

//Tạo một đối tượng Date theo thời gian là millisec tính từ 1-1-1970

Date date1 = new Date(long millisec)

**Một số phương thức đặc trưng của Date**

|  |  |  |
| --- | --- | --- |
| **STT** | **Method** | **Mô tả** |
|  | boolean after(Date when) | Kiểm tra nếu ngày when sau ngày hiện tại |
|  | boolean before(Date when) | Kiểm tra nếu ngày when trước ngày hiện tại |
|  | Object clone() | Trả về một bản sao |
|  | int compareTo(Date anotherDate) | So sánh hai ngày khác nhau |
|  | boolean equals(Object obj) | So sánh hai ngày tương đồng |
|  | getDate()  getDay()  getHours()  getMinutes()  getMonth()  getSeconds()  getTime()  getTimezoneOffset() | Get giá trị |
|  | setYear()  setDate()  setDay()  setHours()  setMinutes()  setMonth()  setSeconds()  setTime()  setTimezoneOffset()  setYear() | Set giá trị |

* Link đọc thêm:
  + <https://www.tutorialspoint.com/java/util/java_util_date.htm>
  + <http://o7planning.org/vi/10245/huong-dan-su-dung-date-time-trong-java#a24894>
* Link Demo:

<https://github.com/ThuHien25/JavaTraining/tree/master/JavaCore/DemoJavaCore/src/JavaUtil>

Calendar (java.util.Calendar)

**Class Calendar** mô tả bộ Lịch. Nó có các method toán học về thời gian, chẳng hạn thêm ngày, bớt ngày,...

**Calendar** là một class trừu tượng. Không thể khởi tạo nó từ cấu tử (Constructor). Tuy nhiên có 2 method tĩnh để tạo ra đối tượng Calendar.

Public static Calendar getInstance();

Public static Calendar getInstance(TimeZone zone);

Public static Calendar getInstance(LocaleZone aLocale);

Public static Calendar getInstance(TimeZone zone, LocaleZone aLocale);

**Ví dụ:**

**Một số phương thức trong Calendar**

|  |  |  |
| --- | --- | --- |
| **STT** | **Method get(int)** | **Return Values** |
|  | get(Calendar.DAY\_OF\_WEEK) | 1 (Calendar.SUNDAY) tới 7 (Calendar.SATURDAY). |
|  | get(Calendar.YEAR) | Năm (year) |
|  | get(Calendar.MONTH) | 0 (Calendar.JANUARY) tới 11 (Calendar.DECEMBER). |
|  | get(Calendar.DAY\_OF\_MONTH) | 1 tới 31 |
|  | get(Calendar.DATE) | 1 tới 31 |
|  | get(Calendar.HOUR\_OF\_DAY) | 0 tới 23 |
|  | get(Calendar.MINUTE) | 0 tới 59 |
|  | get(Calendar.SECOND) | 0 tới 59 |
|  | get(Calendar.MILLISECOND) | 0 tới 999 |
|  | get(Calendar.HOUR) | 0 tới 11, được sử dụng cùng với Calendar.AM\_PM. |
|  | get(Calendar.AM\_PM) | 0 (Calendar.AM) hoặc 1 (Calendar.PM). |
|  | get(Calendar.DAY\_OF\_WEEK\_IN\_MONTH) | DAY\_OF\_MONTH 1 tới 7 luôn luôn tương ứng với DAY\_OF\_WEEK\_IN\_MONTH 1;  8 tới 14 tương ứng với DAY\_OF\_WEEK\_IN\_MONTH 2, ... |
|  | get(Calendar.DAY\_OF\_  YEAR) | 1 tới 366 |
|  | get(Calendar.ZONE\_OFFSET) | Giá trị GMT của múi giờ. |
|  | get(Calendar.ERA) | Biểu thị AD (GregorianCalendar.AD), BC (GregorianCalendar.BC). |
|  | Date getTime() | Trả về đối tượng Date dựa trên giá trị của Calendar. |
|  | void add(int field, int amount) | Thêm hoặc trừ một khoảng thời gian chỉ định (amount) theo trường (field). |
|  | void roll(int calendarField, boolean up) | Cộng hoặc trừ (lên/xuống) duy nhất một đơn vị thời gian cho bởi trường (field) và không làm ảnh hưởng tới các trường khác. |
|  | void roll(int calendarField, int amount) | Thêm một lượng thời gian trên trường chỉ định (calendarField) và không làm ảnh hưởng tới các trường khác |

* Link đọc thêm:
  + <https://www.tutorialspoint.com/java/util/java_util_calendar.htm>
  + <http://o7planning.org/vi/10245/huong-dan-su-dung-date-time-trong-java#a24894>
* Link Demo:

<https://github.com/ThuHien25/JavaTraining/tree/master/JavaCore/DemoJavaCore/src/JavaUtil>

Enum

**Enum** trong Java là một từ khóa, một tính năng được sử dụng để đại diện cho số cố định. Một kiểu **enum** là một kiểu dữ liệu đặc biệt cho phép cho một biến là một tập hợp các hằng số **được xác định trước.**

**Ví dụ:** ngày trong tuần, hành tinh trong hệ mặt trời,…

Link đọc thêm:

<http://o7planning.org/vi/10199/huong-dan-su-dung-java-num#a18700>

Link Demo:

<https://github.com/ThuHien25/JavaTraining/tree/master/JavaCore/DemoJavaCore/src/JavaUtil>

Properties (Class Proterties)

Lớp **Properties** trong Java là lớp phụ của lớp **Hashtable.** Nó được sử dụng để duy trì các danh sách của các value trong đó key là một String và value cũng là một String.

Lớp Properties được sử dụng bởi nhiều lớp Java khác. Ví dụ, nó là kiểu đối tượng được trả về bởi **System.getProperties()** khi đạt được các value môi trường.

**Một số phương thức trong Properties**

|  |  |  |
| --- | --- | --- |
| **STT** | **Method** | **Mô tả** |
|  | void load(Reader r) | Load dữ liệu từ đối tượng Reader |
|  | void load(InputStream is) | Load các dữ liệu từ đối tượng dòng dữ liệu vào |
|  | String getProperty(String key) | Trả về value mà liên kết với key. |
|  | String getProperty(String key, String defaultProperty) | Trả về value mà liên kết với key. defaultPropperty được trả về nếu key hoặc không có trong danh sách hoặc không có trong property list mặc định |
|  | void list(PrintStream streamOut) | Gửi property list tới output stream mà liên kết tới streamOut |
|  | void list(PrintWriter streamOut) | Gửi property list tới output stream mà liên kết tới streamOut |
|  | void load(InputStream streamIn) throws IOException | Nhập một property list từ input stream mà liên kết tới streamIn |
|  | Enumeration propertyNames( ) | Trả về một bản liệt kê các key. Nó cũng bao gồm các key được tìm thấy trong property list mặc định |
|  | Object setProperty(String key, String value) | Liên kết value với key. Trả về value trước mà liên kết với key, hoặc trả về null nếu không có liên kết nào tồn tại |
|  | void store(OutputStream streamOut, String description) | Sau khi ghi chuỗi được xác định bởi description, property list được ghi tới output stream mà liên kết tới streamOut |

* Link đọc thêm:

<https://www.tutorialspoint.com/java/util/java_util_properties.htm>

* Link Demo:

<https://github.com/ThuHien25/JavaTraining/tree/master/JavaCore/DemoJavaCore/src/JavaUtil>

Rescource Bundle

**Rescource Bundle** dùng để chuyển ngôn ngữ, giúp ứng dụng thể hiện được tính quốc tế hóa (đa ngôn ngữ cho 1 ứng dụng). Nói cách khác, chúng ta có thể cung cấp một cơ chế toàn cầu hóa các thông điệp.

* Link đọc thêm:
  + <https://www.tutorialspoint.com/java/util/java_util_resourcebundle.htm>
  + <http://www.javatpoint.com/ResourceBundle-class>
* Link Demo:

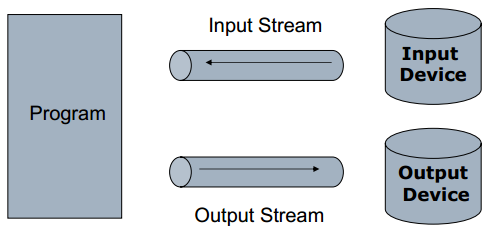
<https://github.com/ThuHien25/JavaTraining/tree/master/JavaCore/DemoJavaCore/src/JavaUtil>

1. Java.io

Cung cấp cho hệ thống input và output thông qua các dòng dữ liệu và hệ thống tập tin.

Stream

Luồng là một “dòng chảy” của dữ liệu được gắn với các thiết bị vào ra. Việc xử lý vào ra thông qua luồng giúp cho lập trình viên không phải quan tâm đến bản chất của thiết bị vào ra. Hai loại luồng:

* **InputStream:** Gắn với các thiết bị nhập *(bàn phím, máy scan, file...)*
* **OutputStream:** Gắn với các thiết bị xuất *(màn hình, máy in, file...)*

Chương trình đọc trên luồng nhập để lấy dữ liệu từ thiết bị nhập, ghi vào luồng xuất để đưa dữ liệu ra thiết bị xuất.

**Các luồng cơ bản:**

* Luồng nhị phân:
  + InputStream: Dòng dữ liệu nhị phân để **đọc**
  + OutputStream: Dòng dữ liệu nhị phân để **ghi**
* Luồng văn bản:
  + Reader: Dòng dữ liệu văn bản để **đọc**
  + Writer: Dòng dữ liệu văn bản để **ghi**
* Các lớp luồng nằm trong gói java.io

File (Java.IO.File)

Lớp File trong Java biểu diễn tên file hoặc thư mục trên file hệ thống của hệ điều hành. Lớp này được sử dụng để tạo các file và thư mục, tìm kiếm các file, xóa các file, …

**Constructor:**

* File(File parent, String child)
* File(String pathname)
* File(String parent, String child)
* File(URI uri)

**Một số method của File**

|  |  |  |
| --- | --- | --- |
| **STT** | **Method** | **Mô tả** |
|  | String getName() | Trả về tên của file hoặc thư mục đã được biểu thị bởi pathname trừu tượng này |
|  | String getParent() | Lấy thư mục cha |
|  | String getPath() | Biến đổi pathname trừu tượng này thành một chuỗi pathname |
|  | boolean canRead() | Kiểm tra có ứng dụng có thể đọc file hay không. |
|  | boolean canWrite() | Kiểm tra có ứng dụng có thể sửa đổi file hay không. |
|  | boolean exists() | Kiểm tra sự tồn tại của file |
|  | boolean isDirectory() | Kiểm tra xem file có phải là thư mục |
|  | boolean isFile() | Kiểm tra có hay không file được biểu thị bởi pathname trừu tượng này là một normal file. Một file là normal nếu nó không là một thư mục và ngoài ra, thỏa mãn các tiêu chuẩn khác phụ thuộc vào hệ thống. Bất kỳ một non-directory file được tạo bởi một ứng dụng Java được bảo đảm là một normal file. |
|  | long lastModified() | Lấy ngày sửa file gần nhất |
|  | long length() | Lấy cỡ file (Byte) |
|  | boolean createNewFile() | Tạo một file mới. |
|  | boolean delete() | Xóa file hoặc thư mục |
|  | void deleteOnExit() | Xóa file hoặc thư mục khi kết thúc chương trình |
|  | String[] list() | Lấy nội dung của thư mục |
|  | File[] listFiles() | Trả về một mảng các pathname |
|  | File[] listFiles(FileFilter filter) | Trả về một mảng các pathname biểu thị các file và thư mục trong thư mục được biểu thị bởi pathname này mà thỏa mãn filter đã cho |
|  | boolean mkdir() | Tạo thư mục |
|  | boolean mkdirs() | Tạo thư mục bao gồm tất cả các thư mục cha không tồn tại |
|  | boolean renameTo(File dest) | Đổi tên file |
|  | boolean setReadOnly() | Thiết lập trạng thái chỉ cho đọc |
|  | int compareTo(File pathname) | So sánh 2 pathname. |

* Link đọc thêm:

<https://www.tutorialspoint.com/java/io/java_io_file.htm>

* Link Demo:

<https://github.com/ThuHien25/JavaTraining/tree/master/JavaCore/DemoJavaCore/src/JavaIO>

FileInputStream

Class FileInputStream dùng để **đọc dữ liệu** của 1 file.

**Constructor**

* FileInputStream(File file)
* FileInputStream(String name)

**Một số method của FileInputStream**

|  |  |  |
| --- | --- | --- |
| **STT** | **Method** | **Mô tả** |
|  | **int available()** | Trả về một ước tính số byte còn lại có thể được đọc (hoặc bỏ qua) |
|  | void close() | Đóng luồng (Stream) |
|  | protected void finalize() | Xóa bỏ tất cả các kết nối đến file. |
|  | FileChannel getChannel() | Trả về đối tượng fileChannel duy nhất |
|  | FileDescriptor getFD() | Trả về mô tả file |
|  | int read() | Đọc dữ liệu byte từ input stream |
|  | int read(byte[] b) | Đọc một số byte dữ liệu từ luồng và lưu vào mảng byte |
|  | int read(byte[] b, int off, int len) | Đọc len byte dữ liệu từ luồng và lưu vào mảng byte bắt đầu tại vị trí off |
|  | long skip(long n) | Bỏ qua n byte dữ liệu đầu vào |

* Link đọc thêm:

<https://www.tutorialspoint.com/java/io/java_io_fileinputstream.htm>

* Link Demo:

<https://github.com/ThuHien25/JavaTraining/tree/master/JavaCore/DemoJavaCore/src/JavaIO>

FileOutputStream

Class FileOuputStream dùng để **ghi dữ liệu** của 1 file.

**Constructor:**

* FileOutputStream(File file)
* FileOutputStream(File file, boolean append)
* FileOutputStream(FileDescriptor fdObj)
* FileOutputStream(String name)

**Một số method của FileOutputStream**

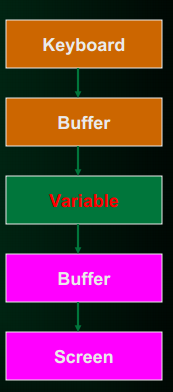
|  |  |  |
| --- | --- | --- |
| **STT** | **Method** | **Mô tả** |
|  | void close() | Đóng stream |
|  | protected void finalize() | Xóa bỏ tất cả các kết nối đến file. |
|  | FileChannel getChannel() | Trả về đối tượng fileChannel duy nhất |
|  | FileDescriptor getFD() | Trả về mô tả file |
|  | void write(byte[] b) | Ghi dữ liệu tương ứng mảng b |
|  | void write(byte[] b, int off, int len) | Dùng để ghi mảng byte xuống luồng bắt đầu từ vị trí off của mảng |
|  | void write(int b) | Ghi giá trị b xác định theo dạng byte |

* Link đọc thêm:

<https://www.tutorialspoint.com/java/io/java_io_fileoutputstream.htm>

* Link Demo:

<https://github.com/ThuHien25/JavaTraining/tree/master/JavaCore/DemoJavaCore/src/JavaIO>

Buffered IO

**Buffered IO** nhập/xuất thông qua bộ đệm (1 vùng nhớ trung gian) giúp giảm bớt số lần đọc ghi dữ liệu trên thiết bị vào ra, tăng tốc độ nhập/xuất.

**Buffered Input:** Dữ liệu nhập được đệm lại không đi vào biến ngay lập tức. **Ví dụ:** Nhập 1 số chỉ kết thúc khi gõ enter.

**Buffered Ouput:** Dữ liệu xuất chỉ được xuất thực sự khi bộ đệm đầy hoặc khi gặp một tác vụ buộc xuất tường minh (flush).

* + 1. Buffered Input

**Constructor**

* BufferedInputStream(InputStream in):
* BufferedInputStream(InputStream in, int size)
* Link đọc thêm:

https://www.tutorialspoint.com/java/io/java\_io\_bufferedinputstream.htm

* Link Demo:

<https://github.com/ThuHien25/JavaTraining/tree/master/JavaCore/DemoJavaCore/src/JavaIO>

* + 1. Buffered Ouput

**Constructor**

* BufferedOutputStream(OutputStream out)
* BufferedOutputStream(OutputStream out, int size)
* Link đọc thêm:

<https://www.tutorialspoint.com/java/io/java_io_bufferedoutputstream.htm>

* Link Demo:

<https://github.com/ThuHien25/JavaTraining/tree/master/JavaCore/DemoJavaCore/src/JavaIO>

****PrintWriter (Java.IO.PrintWriter)****

**PrintWriter** được dùng trong việc ghi dữ liệu, trợ giúp cho việc in ấn dữ liệu cơ bản.

**Một số method của PrintWriter**

|  |  |  |
| --- | --- | --- |
| **STT** | **Method** | **Mô tả** |
|  | **PrintWriter append(char c)** | Được sử dụng để phụ thêm chuỗi đã cho với chuỗi này |
|  | **boolean checkError()** | Dùng để kiểm tra lỗi |
|  | **protected void clearError()** | Dùng để xoá sạch các lỗi |
|  | **void close()** | Dùng để đóng stream lại |
|  | **void flush()** | Dùng để đẩy dữ liệu đi |
|  | **void println(String x)** | Dùng để ghi một chuỗi và xuống dòng |
|  | **void write(String s)** | Dùng để ghi một chuỗi |

* Link đọc thêm:

<https://www.tutorialspoint.com/java/io/java_io_printwriter.htm>

* Link Demo:

<https://github.com/ThuHien25/JavaTraining/tree/master/JavaCore/DemoJavaCore/src/JavaIO>

****Interface (java.IO.Interface)****

* + 1. ****DataInput Interface****

**DataInput Interface** được dùng để đọc các byte nhị phân từ 1 dòng byte vật lý (InputStream). Chuyển đổi chuỗi dạng UTF-8 có sửa đổi thành dữ liệu dạng String.

**Lớp DataInputStream** hiện thực DataInput Interface.

* + 1. ****Serializable Interface****

Serialize có nghĩa là sắp theo thứ tự. Khi ta muốn lưu một đối tượng xuống tập tin trên đĩa để lưu trữ, ta phải định ra trình tự lưu trữ của dữ liệu trong đối tượng. Khi cần tái tạo lại đối tượng từ thông tin trên tập tin đã lưu trữ, ta sẽ nạp đúng theo trình tự đã định trước đó. Đây được gọi quá trình Serialize.

Serialize là tiến trình biến đổi trạng thái của đối tượng theo một định dạng có thể được lưu trữ hay dịch chuyển. Lớp Serializable giúp Java quản lí và ràng buộc được các đối tượng cả khi đọc và khi ghi xuống tập tin.

Để Serialize một object ta cần đảm bảo rằng class của object đó cài đặt **giao diện java.io.Serializable**. Đây là một giao diện **trống, ko có method** nào cần cài đặt.

* Link đọc thêm:

<http://laptrinh.vn/d/4157-serializable-va-deserialize-la-gi.html>

Lớp Exeption thường gặp

|  |  |  |
| --- | --- | --- |
| **STT** | **Exeption** | **Mô tả** |
|  | FileNotFoundException | Đường dẫn không đúng |
|  | IOException | Nghĩa là một số ngoại lệ đã xảy ra |
|  | UnsupportedEncoding  Exception | Không hỗ trợ định dạng chữ viết |

* Link đọc thêm:

<http://www.vanthuong.com/2016/09/exception-va-cach-xu-ly.html>

1. Java.lang

**Gói Java.lang** là gói cơ bản của ngôn ngữ Java *(language).* Chứa các lớp quan trọng nhất của ngôn ngữ Java.

* Các kiểu dữ liệu cơ bản như Character, Integer, Float, Double,…
* Các lớp có chứa hàm tính toán.
* Các lớp có liên quan đến chuỗi
* Các lớp làm nhiệm vụ xử lý lỗi và các lớp nhập xuất chuẩn.

StringBuffer

**Lớp StringBuffer** trong Java được sử dụng để tạo chuỗi có thể sửa đổi (chuỗi dạng mutable). Lớp StringBuffer là giống như lớp String trong Java, ngoại trừ ở điểm là nó là có thể sửa đổi.

**Constructor**

* StringBuffer(): tạo một bộ đệm chuỗi trống với dung lượng capacity ban đầu là 16.
* StringBuffer(String str): tạo một bộ đệm chuỗi với chuỗi đã xác định.
* StringBuffer(int capacity): tạo một bộ đệm chuỗi trống với dung lượng capacity đã cho.

**Một số method của StringBuffer**

|  |  |  |
| --- | --- | --- |
| **STT** | **Exeption** | **Mô tả** |
|  | append(String s) | Nối thêm một chuỗi hoặc một mảng ký tự vào cuối cùng của đối tượng stringbuffer |
|  | insert(int offset, String s) | Offset là vị trí chèn. Tham số thứ 2 có thể là string, char, int, float được chèn vào. Vị trí chèn sẽ lớn hơn hay bằng 0, và nhỏ hơn hay bằng chiều dài của đối tượng stringbuffer. Bất kỳ đối số nào, điều được chuyển sang chuỗi và sau đó mới được chèn vào. |
|  | replace(int startIndex, int endIndex, String str) | Thay thế chuỗi từ chỉ mục ban đầu startindex và chỉ mục kết thúc endindex đã cho. |
|  | delete(int startIndex, int endIndex) | Xóa chuỗi từ chỉ mục startindex và endindex đã cho. |
|  | deleteCharAt(int index) | Xóa ký tư tại vị trí index |
|  | reverse() | Đảo ngược chuỗi |
|  | capacity() | Trả về dung lượng capacity hiện tại. |
|  | ensureCapacity(int minimumCapacity) | Bảo đảm rằng capacity ít nhất bằng với minimum đã cho. |
|  | charAt(int index) | Trả về một ký tự trong đối tượng stringbuffer tại vị trí được chỉ định. |
|  | setCharAt(int index, char value) | Thay thế ký tự trong một stringbuffer bằng một  Ký tự khác tại một vị trí được chỉ định. |
|  | setLength() | Thiết lập chiều dài của đối tượng stringbuffer. Nếu chiều dài được chỉ định nhỏ hơn chiều dài dữ liệu hiện tại của nó, thì các ký tự thừa sẽ bị cắt bớt. Ngược lại, các ký tự null được thêm vào phần cuối của stringbuffer nếu chiểu dài chỉ định nhiều hơn chiều dài dữ liệu. |
|  | substring(int beginIndex) | Trả về chuỗi con từ chỉ mục bắt đầu beginindex đã cho. |
|  | substring(int beginIndex, int endIndex) | Trả về chuỗi con từ chỉ mục bắt đầu beginindex đến endindex đã cho |

* Link đọc thêm:

<https://www.tutorialspoint.com/java/lang/java_lang_stringbuffer.htm>

* Link Demo:

<https://github.com/ThuHien25/JavaTraining/tree/master/JavaCore/DemoJavaCore/src/JavaLang>

StringBuilder

Lớp StringBuilder trong Java được sử dụng để tạo chuỗi có thể thay đổi (chuỗi dạng mutable). Lớp StringBuilder là giống như lớp StringBuffer ngoại trừ rằng nó là không đồng bộ.

**Constructor**

* StringBuilder(): tạo một Builder trống với dung lượng capacity ban đầu là 16.
* StringBuilder(String str): tạo một Builder với chuỗi đã xác định.
* StringBuilder(int capacity): tạo một Builder trống với dung lượng capacity đã cho.

Phương thức của StringBuilder tương tự StringBuffer

* Link đọc thêm:

<https://www.tutorialspoint.com/java/lang/java_lang_stringbuilder.htm>

* Link Demo:

<https://github.com/ThuHien25/JavaTraining/tree/master/JavaCore/DemoJavaCore/src/JavaLang>

So sánh String, StringBuffer, StringBuilder

|  |  |  |
| --- | --- | --- |
| **STT** | **String** | **StringBuffer, StringBuilder** |
|  | Không thể thay đổi giá trị (**immutable**) | Có thể thay đổi giá trị (**mutable**) |
|  | Chậm và tốn nhiều bộ nhớ hơn trong việc nối chuỗi | Nhanh và tốn ít bộ nhớ hơn trong việc nối chuỗi |
|  | String ghi đè phương thức equals của lớp Object, nên có thể so sánh nội dung của hai string khác nhau | StringBuffer không ghi đè phương thức equals của lớp Object |

|  |  |  |
| --- | --- | --- |
| **STT** | **StringBuilder** | **StringBuffer** |
|  | StringBuilder là không đồng bộ (non-synchronized), các phương thức không xử lý đa luồng. | Lớp StringBuffer là đồng bộ (synchronized), các phương thức của nó đều là “thread safe” (thích hợp với xử lý đa luồng – multithread) |
|  | StringBuilder hiệu quả hơn StringBuffer | StringBuffer kém hiệu quả hơn StringBuilder |

* Link đọc thêm:

<https://daynhauhoc.com/t/nhung-diem-khac-nhau-cua-string-stringbuilder-va-stringbuffer-trong-java/12407/2>

**TÀI LIỆU THAM KHẢO**

1. <https://www.tutorialspoint.com/java/io/index.htm>
2. <http://www.javatpoint.com/java-tutorial>
3. <http://v1study.com/java-gioi-thieu-a573.html>
4. <http://kenhlaptrinh.net/gioi-thieu-kenh-lap-trinh/>