MỤC LỤC

I. Java.util

1. List và các thư viện đi kèm theo List

List là một interface con của Collection 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

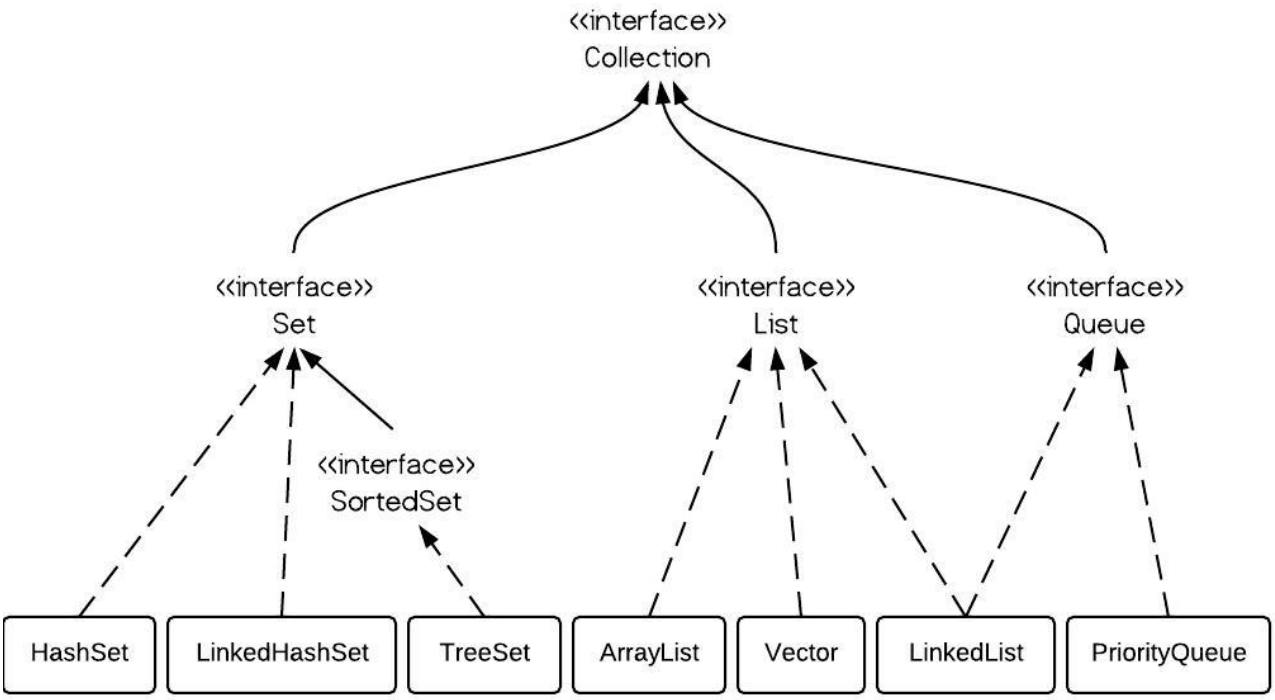
Cho phép 0 hoặc nhiều phần tử null.

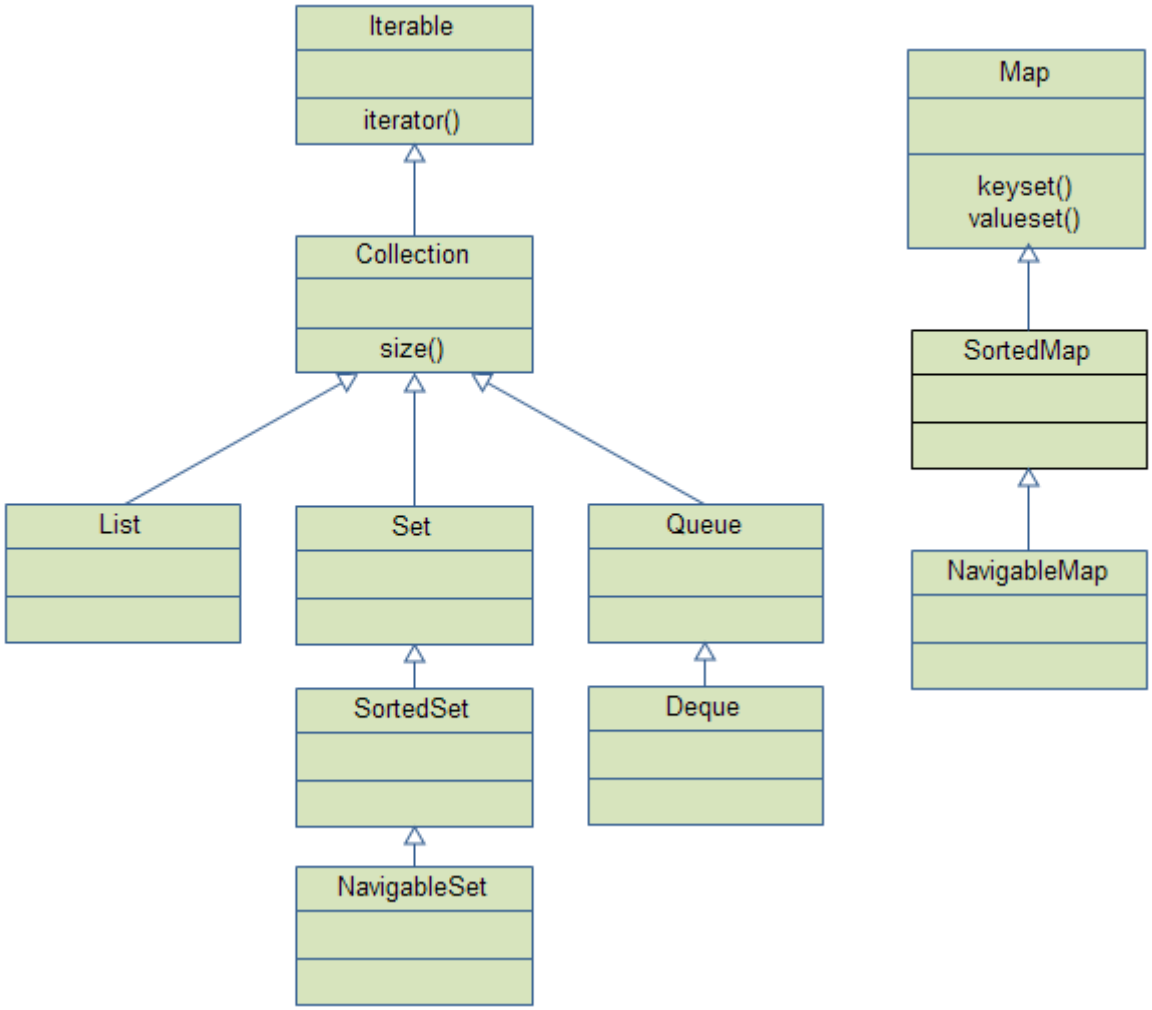
Là một tập hợp có tuần tự.

Các phần tử có thể được chèn hoặc truy cập thông qua vị trí của chúng trong danh sách.

Các thư viện đi kèm List: ArrayList, LinkedList, Stack, ArrayDeque, Vector,…

Để hiểu và sử dụng được Java Collections API một cách hiểu quả, ta hãy nhìn mô hình dưới đây để có cái nhìn tổng quát





1.1. ArrayList Trong Java

Việc sử dụng mảng (array) có một vài nhược điểm mà các lập trình viên thường gặp phải như sau:

Nếu khai báo kích thước mảng quá nhỏ thì sẽ dẫn đến thiếu bộ nhớ khi lưu trữ. Nếu khai báo lớn thì thừa bộ nhớ (quá lãng phí). Các phần tử trong mảng lại có cùng kiểu dữ liệu.

ArrayList đã tránh những nhược điểm này. ArrayList là một mảng động.

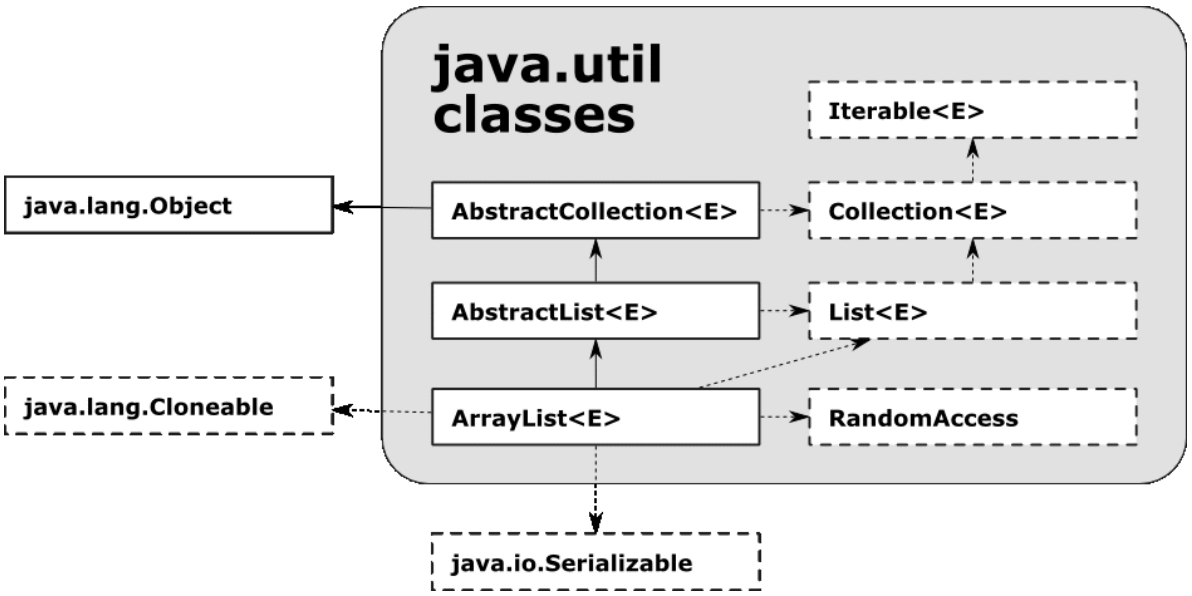
ArrayList trong Java kế thừa AbstractList và thực thi List Interface.

ArrayList được ưa chuộng trong Java vì chức năng và tính linh hoạt mà nó cung cấp. Hầu hết các lập trình viên đều chọn ArrayList thay thế cho cách dùng Array truyền thống.

Kiểu dữ liệu mảng truyền thống Array bị giới hạn với kích thước xác định. Nếu một số phần tử trong mảng bị xóa đi thì bộ nhớ chứa các phần tử đó không được thu hồi. Ngược lại kích thước của ArrayList có thể tăng hoặc giảm tùy thích.

Các mảng truyền thống có độ dài cố định. Sau khi các mảng được tạo, chúng không thể tăng hoặc giảm kích cỡ.

ArrayList được tạo với một kích cỡ ban đầu. Khi kích cỡ này bị vượt, collection tự động được tăng. Khi các đối tượng bị gỡ bỏ, ArrayList có thể bị giảm kích cỡ.



Một số phương thức thường được sử dụng trong ArrayList:

1) add(Object o): Phương thức thêm phần tử vào cuối mảng
2) add(int index, Object o): Phương thức thêm phần tử vào chỉ mục định trước
3) remove(Object o): Xóa phần tử trong mảng
4) remove(int index): Xóa phần tử theo chỉ mục
5) set(int index, Object o): Phương thức cập nhật phần tử trong mảng, nó thay thế phần tử ở chỉ mục đã cho bằng phần tử mới.
6) int indexOf(Object o): Phương thức lấy chỉ mục của một phần tử, nếu kết quả trả về -1 thì phần tử không tồn tại trong mảng.
7) Object get(int index): Phương thức lấy phần tử ở chỉ mục đã cho trong mảng.
8) int size(): Phương thức xác định kích thước mảng.
9) boolean contains(Object o): Phương thức kiểm tra phần tử có tồn tại trong mảng hay không.
10) clear(): Phương thức xóa tất cả các phần tử trong mảng.

Ví dụ:

```
package arrayListExample;

import java.util.ArrayList;
import java.util.Iterator;

public class ArrayListExamples {

     public static void main(String[] args) {
//         Creating an empty array list
           ArrayList<String> list = new ArrayList<String>();

//         Adding items to array list
           list.add("Student A");
           list.add("Student B");
           list.add("Student C");
           list.add(3, "Student D");
           list.add("Student E");

//         Show the contents of the array list
           System.out.println("The arraylist contains: " + list);

//         checking index of an item
           int position = list.indexOf("Student D");
           System.out.println("The index of Student D is: " +
position);

//         checking list isEmpty
           boolean check = list.isEmpty();
           System.out.println("Checking if the arraylist is empty:
" + check);

//         getting the size of the list
           int size = list.size();
           System.out.println("The size of the list is: " + size);

//         checking if an element is included to the list
           boolean element = list.contains("Student F");
           System.out.println("ArrayList contains Student F: " +
element);
```

```
//        getting the element in a specific position
          String student = list.get(0);
          System.out.println("The student is at the index 0 is: "
+ student);

//        loop using index and size list
          System.out.println("Loop using index and size list");
          for (int i = 0; i < list.size(); i++) {
               System.out.println("Index: " + i + " - Student: "
+ list.get(i));
          }

//        using foreach loop
          System.out.println("Using foreach loop");
          for (String str : list) {
               System.out.println("Student is: " + str);
          }

//        using iterator
          System.out.println("Using iterator");
          for (Iterator<String> iterator = list.iterator();
iterator.hasNext();) {
               System.out.println("Student is: " +
iterator.next());
          }

//        replacing an element
          list.set(2, "New Student");
          System.out.println("The arraylist after replacement is:
" + list);

//        removing students
          list.remove(0);
          list.remove("Student D");
          System.out.println("Contents of arraylist: " + list);

     }
}
```

Kết quả:

```
The arraylist contains:
[Student A, Student B, Student C, Student D, Student E]
The index of Student D is: 3
Checking if the arraylist is empty: false
The size of the list is: 5
ArrayList contains Student F: false
The student is at the index 0 is: Student A
Loop using index and size list
Index: 0 - Student: Student A
Index: 1 - Student: Student B
Index: 2 - Student: Student C
Index: 3 - Student: Student D
Index: 4 - Student: Student E
Using foreach loop
Student is: Student A
Student is: Student B
Student is: Student C
Student is: Student D
Student is: Student E
Using iterator
Student is: Student A
Student is: Student B
Student is: Student C
Student is: Student D
Student is: Student E
The arraylist after replacement is:
[Student A, Student B, New Student, Student D, Student E]
Contents of arraylist: [Student B, New Student, Student E]
```

1.2. LinkedList trong Java

Lớp LinkedList trong Java kế thừa lớp AbstractSequentialList và thực thi List Interface. Nó cung cấp một cấu trúc dữ liệu linked-list (danh sách liên kết).

LinkedList có cơ chế giống như 1 sợi dây xích, phần tử 0 móc vào phần tử 1, rồi 1 móc vào 2, 2 móc vào 3,… không phải khai báo thừa ra như ArrayList nên bộ nhớ tốn ít hơn.

Thời gian gọi tới 1 phần tử thứ i thì chậm dần đều từ ngoài vào trong: tức là phần tử ngoài cùng của sợi dây xích thì gọi tới rất nhanh, nhưng càng vào giữa thì càng chậm dần

Thời gian xóa thì nhanh hơn so với ArrayList, nó chỉ mất chủ yếu là thời gian tìm tới phần tử thứ i thôi, còn sau đó chỉ việc tách nó ra khỏi sợi dây xích và nối 2 đầu dây lại

Thêm phần tử mới cũng nhanh hơn ArrayList, chỉ việc móc thêm vào đuôi cái xích là xong

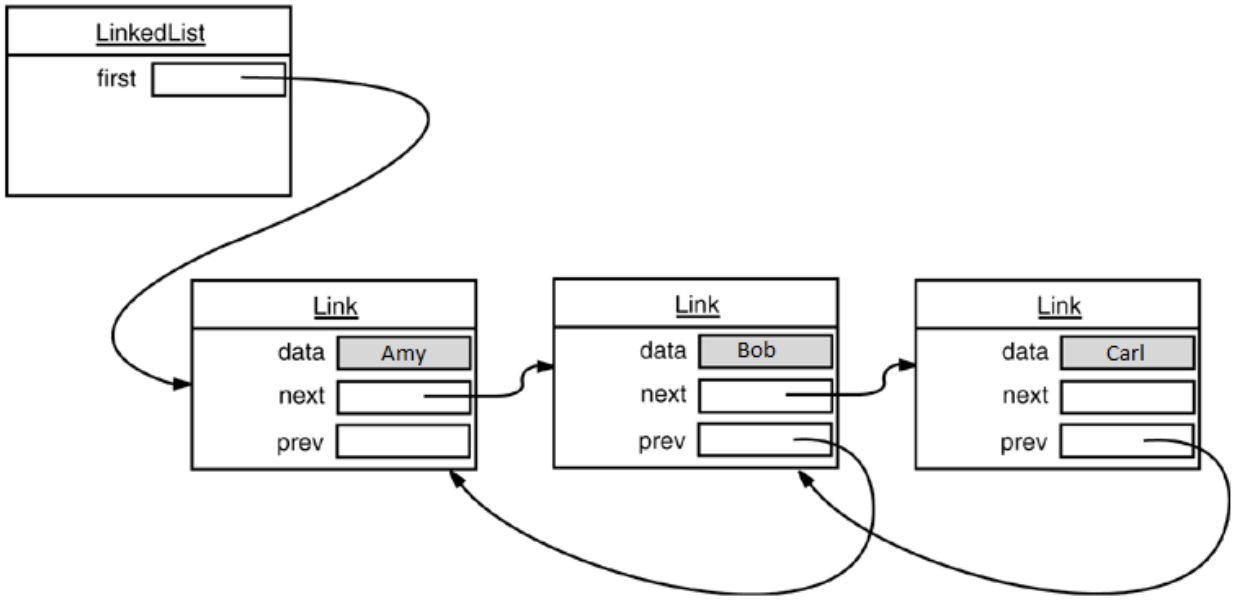
Ngoài các method như ArrayList, LinkedList còn có thêm các method:

1. addFirst()
2. addLast()
3. getFirst()
4. getLast()
5. removeFirst()
6. removeLast()
7. peek() = getFirst()
8. poll() = getFirst() + removeFirst()

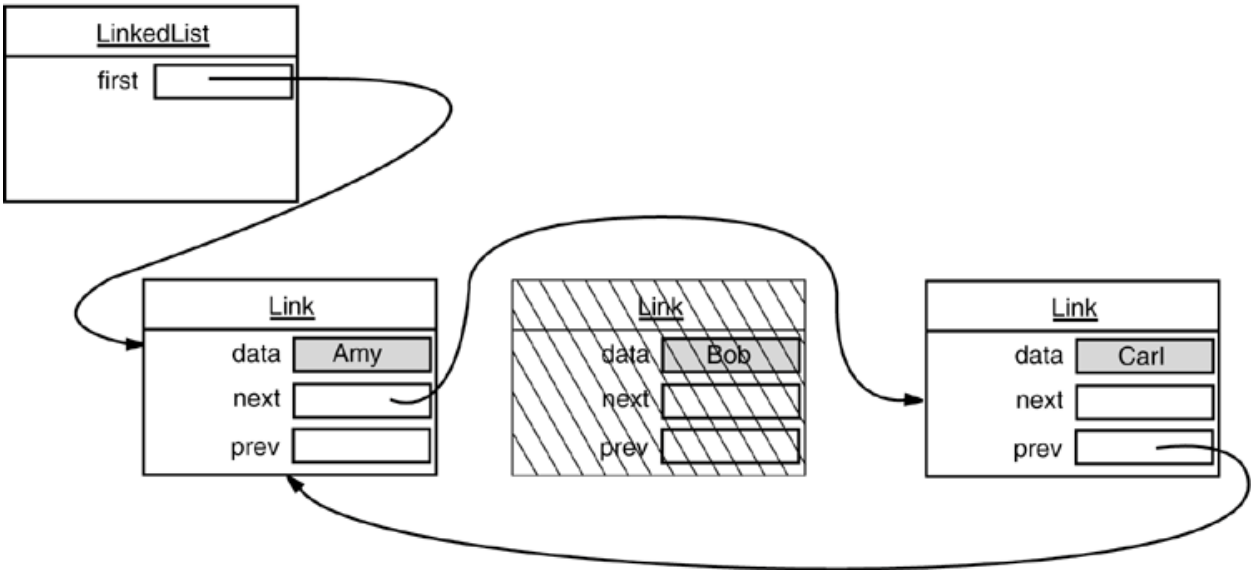
Khi nào dùng ArrayList, khi nào dùng LinkedList?

Nếu chương trình thường xuyên phải truy nhập trực tiếp dữ liệu trong list à sử dụng ArrayList

Nếu chương trình thường xuyên phải thêm hoặc xóa 1 phần tử trong list(đặc biệt là thêm hoặc xóa ở giữa) thì nên sử dụng LinkedList



Khi có một phần tử bất kì bị remove: Tôi remove **Bob** nhé:



Ví dụ:

```
package linkedListinJava;

import java.util.LinkedList;
import java.util.ListIterator;

public class LinkedListExample {

	public static void main(String[] args) {
//		create a LinkedList
		LinkedList<String> list = new LinkedList<String>();

//		add elements to the linked list
		list.add("Tuesday");
		list.add("Wednesday");
		list.add("Thursday");
		list.add("Friday");

//		create a list iterator
		ListIterator<String> listIterator =
list.listIterator();
		System.out.println("-----LinkedList-----");

		while (listIterator.hasNext()) {
			System.out.println(listIterator.next());
		}

//	add elements in the beginning and in the end of the list
		list.addFirst("Monday");
		list.addLast("Saturday");
		System.out.println("-----After adding-----");

		listIterator = list.listIterator();
		while (listIterator.hasNext()) {
			System.out.println(listIterator.next());
		}

		System.out.println("Check if list contains the Friday:
" + list.contains("Friday"));
```

```
            System.out.println("Position of Wednesday is: " +
list.indexOf("Wednesday"));

            System.out.println("Get element in 6th position: " +
list.get(5));

            int size = list.size();
            System.out.println("The size of list is: " + size);

            System.out.println("-----Reversing list-----");
            listIterator = list.listIterator(size);
            while(listIterator.hasPrevious()) {
                 System.out.println(listIterator.previous());
            }

//          remove element from list
            list.remove("Thursday");
            list.removeFirst();
            list.removeLast();
            System.out.println("---LinkedList after removing---");
            listIterator = list.listIterator();
            while (listIterator.hasNext()) {
                 System.out.println(listIterator.next());
            }
      }
}
```

Kết quả:

```
-----LinkedList-----
Tuesday
Wednesday
Thursday
Friday
-----After adding-----
Monday
Tuesday
Wednesday
Thursday
Friday
Saturday
Check if list contains the Friday: true
```

```
Position of Wednesday is: 2
Get element in 6th position: Saturday
The size of list is: 6
-----Reversing list-----
Saturday
Friday
Thursday
Wednesday
Tuesday
Monday
-----LinkedList after removing-----
Tuesday
Wednesday
Friday
```

1.3. Vector trong Java

Lớp vector tương tự như ArrayList và nó cũng thực thi các mảng động. Lớp vector lưu mảng các đối tượng và kích thước của mảng này có thể tăng hoặc giảm. Chúng ta có thể truy cập các phần tử của Vector bằng cách sử dụng vị trí của phần tử đó.

Sự khác nhau giữa Vector và ArrayList là các phương thức của Vector dùng cơ chế đồng bộ (synchronised) và thread-safe. Cho nên Vector thường chạy chậm, các phương thức của ArrayList chạy nhanh hơn.

Các phương thức cơ bản của lớp này:

addElement(Object obj) phương thức này thêm phần tử vào vị trí cuối cùng của Vector và kích thước của Vector sẽ tăng lên 1.

insertElementAt(Object obj, int index) chèn một phần tử tại vị trí xác định.

setElementAt(Object obj, int index) đặt giá trị cho một phần tử tại một vị trí

toArray() trả về một mảng chứa tất cả các phần tử trong Vector.

elementAt(int pos) lấy một đối tượng được lưu trữ xác định bởi vị trí.

removeElement(Object obj) xóa một phần tử xuất hiện đầu tiên trong Vector có giá trị bằng với Object.

setSize(int newSize) thiết lập kích cỡ của Vector này

int size() trả về số phần tử trong Vector này

int lastIndexOf(Object elem, int index) tìm kiếm ngược về sau cho đối tượng đã cho, bắt đầu từ index đã xác định, và trả về một chỉ mục

Ví dụ:

```
package vecTorExample;

import java.util.Enumeration;
import java.util.Vector;

public class VectorExample {

	public static void main(String[] args) {
//		Create a Vector and populate it with elements
		Vector<Object> vector = new Vector<Object>();
		vector.addElement(new String("Task 1"));
		vector.addElement(new String("Task 2"));
		vector.addElement(new String("Task 3"));
		vector.addElement("Task 4");
		vector.add("Task 5");

		System.out.println(vector);
		System.out.println("Capacity: " + vector.capacity());
		System.out.println("Vector Size: " + vector.size());

//		Clone another vector
		System.out.println(vector.clone());

//		Replacing an element at the specified index of Vector
		vector.set(1, "Finish");
		System.out.println(vector);

//		checking if an element is included to the list
		boolean check = vector.contains("Task 3");
		System.out.println("Vector contains Task 3: " + check);

		Enumeration<Object> enumeration = vector.elements();

		System.out.println("-----Vector Elements-----");
		while (enumeration.hasMoreElements()) {
			System.out.println(enumeration.nextElement());
		}

//		Remove the specific object from the Vector
		vector.remove("Task 4");
```

```
            System.out.println("After removing: " + vector);

            Object object = vector.elementAt(3);
            System.out.println("Position 3 is: " + object);

        }

}
```

Kết quả:

```
[Task 1, Task 2, Task 3, Task 4, Task 5]
Capacity: 10
Vector Size: 5
[Task 1, Task 2, Task 3, Task 4, Task 5]
[Task 1, Finish, Task 3, Task 4, Task 5]
Vector contains Task 3: true
-----Vector Elements-----
Task 1
Finish
Task 3
Task 4
Task 5
After removing: [Task 1, Finish, Task 3, Task 5]
Position 3 is: Task 5
```

1.4. Calendar trong Java

Sơ lược về các bộ lịch:

Calendar là class mô phỏng một hệ thống Lịch.

Gregorian Calendar:Đây chính là Dương Lịch, còn gọi lịch Thiên chúa giáo, là lịch quốc tế. Nó được sử dụng rộng rãi nhất được đặt tên theo Đức Giáo Hoàng Gregory XIII, người đã giới thiệu nó vào năm 1582.

Buddhist Calendar: Đây là một bộ lịch phật giáo, thường được sử dụng tại một số nước Đông Nam Á trước kia như Thái Lan, Lào, Campuchia, cũng như Sri Lanka. Hiện nay lịch này được sử dụng trong các lễ hội phật giáo. Và không còn quốc gia nào sử dụng lịch này một cách chính thức, các quốc gia này đã đổi sang sử dụng **Gregorian Calendar**.

Japanese Imperial Calendar: Đây là bộ lịch truyền thống của Nhật bản, hiện nay Nhật bản đã chuyển sang sử dụng dương lịch (**Gregorian Calendar**), tuy nhiên bộ lịch truyền thống vẫn được sử dụng một cách không chính thức.

Phương thức get(int)	Giá trị trả về
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).

Ví dụ:

```
package calendarExample;

import java.util.Calendar;
import java.util.TimeZone;

public class CalendarExample {

     public static void main(String[] args) {
//         Create calendar instance
           Calendar calendar = Calendar.getInstance();
           int year = calendar.get(Calendar.YEAR);

//         get time date
           int month = calendar.get(Calendar.MONTH);
           int day = calendar.get(Calendar.DAY_OF_MONTH);
           int hour = calendar.get(Calendar.HOUR_OF_DAY);
           int minute = calendar.get(Calendar.MINUTE);
           int second = calendar.get(Calendar.SECOND);
```

```
            int millis = calendar.get(Calendar.MILLISECOND);
            int hour12 = calendar.get(Calendar.AM_PM);
            int GMT = calendar.get(Calendar.ZONE_OFFSET);

//          get current TimeZone
            TimeZone timeZone = calendar.getTimeZone();

            System.out.println("Year: " + year);
           System.out.println("Month: " + (month+1));
           System.out.println("Day: " + day);
           System.out.println("Hour: " + hour);
           System.out.println("Minute: " + minute);
           System.out.println("Second: " + second);
           System.out.println("Milli Second: " + millis);
           System.out.println("AM/ PM: " + hour12);
           System.out.println("GMT: " + GMT);
           System.out.println("Current week of this month: " +
calendar.get(Calendar.WEEK_OF_MONTH));
           System.out.println("Current week of this year: " +
calendar.get(Calendar.WEEK_OF_YEAR));
           System.out.println("Current TimeZone is : " +
timeZone.getDisplayName());
           System.out.println(hour + ":" + minute + ":" + second +
":" + millis + "," + day + "/" + (month+1) + "/" + year);

     }
}
```

Kết quả:

```
Year: 2016
Month: 2
Day: 16
Hour: 17
Minute: 6
Second: 8
Milli Second: 817
AM/ PM: 1
GMT: 25200000
Current week of this month: 3
Current week of this year: 8
Current TimeZone is : Indo china Time
17:6:8:817,16/2/2016
```

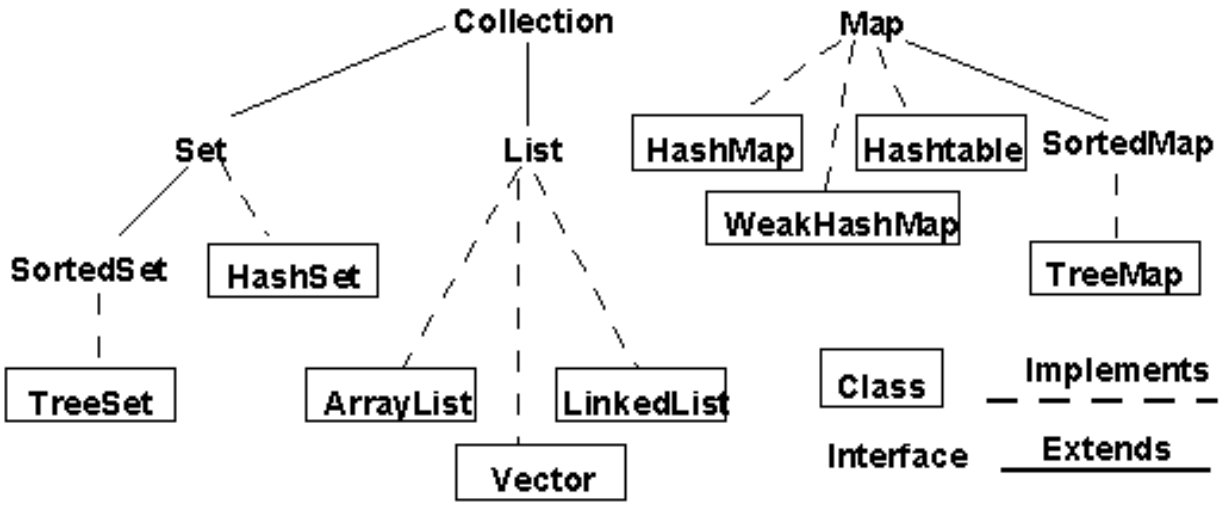
2. Map và các thư viện đi kèm theo Map

Map là một tập dữ liệu được lưu dưới dạng key-value.

Một Map không thể chứa những key trùng nhau, nhưng mỗi key thì có thể được ánh xạ đến nhiều hơn một giá trị. Nghĩa là giá trị các phần tử trong Map thì có thể trùng nhau nhưng khóa thì không được trùng.

Map interface được định nghĩa gồm những method phục vụ những hoạt động cơ bản (như put, get, remove, containsKey, containsValue, size, empty), phục vụ việc thao tác hàng loạt (như putAll, clear), và phục vụ việc xem dữ liệu trong tập (như keySet, entrySet, values).

Các thư viện đi kèm: HashMap, TreeMap, HashTable.



2.1. HashMap

HashMap còn được gọi là mảng băm chứa cặp khóa - giá trị và được ký hiệu HashMap <Key, Value> hoặc HashMap <K, V>

HashMap là một implement của Map interface trong Java. Các phương thức của HashMap không đồng bộ (unsynchronized), không thread safe và cho phép khóa và giá trị có thể nhận giá trị null.

Kiểu đối tượng của giá trị key/value của HashMap phải đồng nhất. Trong trường hợp không xác định kiểu dữ liệu thì java sẽ xem như là kiểu Object – là cha của tất cả đối tượng khác trong Java.

Danh sách phương thức của HashMap.

1. **void clear():** Loại bỏ tất cả các cặp khóa và giá trị ra khỏi HashMap.
2. **Object clone():** Trả về một bản copy tất cả các cặp khóa và giá trị, thường được sử dụng để sao chép sang một HashMap khác.
3. **boolean containsKey(Object key):** trả về TRUE nếu trong HashMap có chứa key, trả về FALSE nếu trong HashMap không chứa key.
4. **boolean containsValue(Object value):** trả về TRUE nếu trong HashMap có chứa value, trả về FALSE nếu trong HashMap không chứa value.
5. **Value get(Object key):** Trả về giá trị được ánh xạ bởi key tương ứng.
6. **boolean isEmpty():** Thực hiện kiểm tra HashMap có rỗng hay không.
7. **Set keySet():** Trả về Set - danh sách tất cả các key được lấy từ HashMap.
8. **Value put(Key k, Value v):** Chèn thêm value vào trong HashMap với key.
9. **int size():** Trả về số lượng phần tử có trong HashMap.
10. **Collection values():** Trả về danh sách tất cả các giá trị có trong HashMap.
11. **Value remove(Object key):** Trả về cặp khóa giá trị tương ứng với key
12. **void putAll(Map m):** Copy tất cả các giá trị của HashMap vào Map

Ví dụ:

```
package hashMapExample;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.Iterator;
import java.util.Map;
import java.util.Map.Entry;
import java.util.Set;

public class HashMapExample {

	public static void main(String[] args) {
//		create a hashmap
		HashMap<String, Integer> laptop = new HashMap<>();
//		hashMap with multiple values
		HashMap<String, ArrayList<String>> lapColor =
new HashMap<String, ArrayList<String>>();
```

```
//        Add some laptops, put elements to the hashMap
          laptop.put("Lenovo", 5);
          laptop.put("Dell", 7);
          laptop.put("Vaio", 3);
          laptop.put("Asus", 9);

//        take size of hashmap
          System.out.println("Total Lap: " + laptop.size());

//        check if hashmap contains values 5
          System.out.println("Hashmap contains 5: "
+ laptop.containsValue(5));

//        check if hashmap contains key Acer
          System.out.println("Hasmap contains Acer: "
+ laptop.containsKey("Acer"));

//        Using the keySet method, take a value of a specific key
          for (String key: laptop.keySet()) {
               System.out.println(key + " have: "
+ laptop.get(key));
          }

          String searchKey = "Lenovo";
          if (laptop.containsKey(searchKey)) {
               System.out.println("We found "
+ laptop.get(searchKey) + " " + searchKey);
          }


//        Create an arrayList lapColor
          ArrayList<String> firstList = new ArrayList<String>();
          firstList.add("Blue");
          firstList.add("Black");
          firstList.add("Brown");

//        Create second arraylist lapColor
          ArrayList<String> secondList = new ArrayList<String>();
          secondList.add("Pink");
          secondList.add("Purple");

//        Put values into hashmap
```

```
          lapColor.put("B Color", firstList);
          lapColor.put("P Color", secondList);

          Set<Entry<String, ArrayList<String>>> setMap =
lapColor.entrySet();

          Iterator<Entry<String, ArrayList<String>>> iteratorMap
= setMap.iterator();

          System.out.println("Hasmap with multiValues: ");
          while (iteratorMap.hasNext()) {
               Map.Entry<String, ArrayList<String>> entry =
                         (Map.Entry<String, ArrayList<String>>)
iteratorMap.next();
               String key = entry.getKey();
               ArrayList<String> values = entry.getValue();
               System.out.println("Key = '" + key + "' has
values: " + values);
          }
     }
}
```

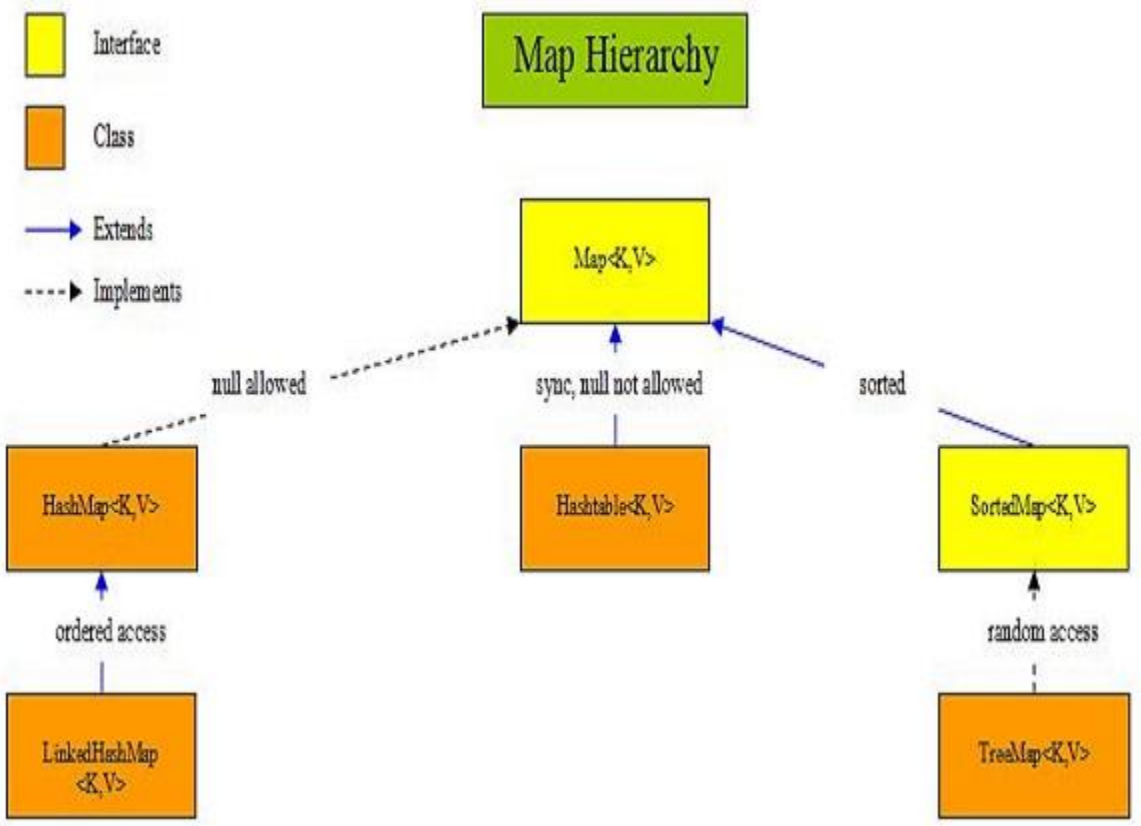
Kết quả:

```
Total Lap: 4
Hashmap contains 5: true
Hasmap contains Acer: false
Lenovo have: 5
Dell have: 7
Vaio have: 3
Asus have: 9
We found 5 Lenovo
Hasmap with multiValues:
Key = 'P Color' has values: [Pink, Purple]
Key = 'B Color' has values: [Blue, Black, Brown]
```

2.2. HashTable

Lớp Hashtable cũng tương tự như HashMap, nhưng nó được đồng bộ và không cho phép key và values có thể nhận giá trị null.

Lớp này cũng được định nghĩa trong gói java.util. Hashtable được sử dụng để ánh xạ khoá (key) đến giá trị (value). Ví dụ, nó có thể được sử dụng để ánh xạ các tên đến tuổi, những người lập trình đến những dự án, chức danh công việc đến lương, và cứ như vậy.



Hashtable mở rộng kích thước khi các phần tử được thêm vào. Khi một Hashtable mới, bạn có thể chỉ định dung lượng ban đầu và yếu tố nạp (load factor).

Điều này sẽ làm cho hashtable tăng kích thước lên, bất cứ lúc nào việc thêm vào một phần tử mới làm vượt qua giới hạn hiện hành của Hashtable. Giới hạn của Hashtable là dung lượng nhân lên bởi các yếu tố được nạp.

Ví dụ: một bảng băm với dung lượng 100, và một yếu tố nạp là 0.75 sẽ có một giới hạn là 75 phần tử.

Các phương thức khởi tạo cho Hashtable gồm:

Hashtable(int) (Xây dựng một bảng mới với dung lượng ban đầu được chỉ định.)
Hashtable(int, float) (Xây dựng một lớp mới với dung lượng ban đầu được chỉ định và yếu tố nạp.)
Hashtable() (Xây dựng một lớp mới bằng cách sử dụng giá trị mặc định cho dung lượng ban đầu và yếu tố nạp.)

Hashtable hash1 = new Hashtable(500,0.80);

Trong trường hợp này, Bảng băm "hash1" sẽ lưu trữ 500 phần tử. Khi bảng băm lưu trữ vừa đầy 80% (một yếu tố nạp vào của 0.80), kích thước tối đa của nó sẽ được tăng lên.

Mỗi phần tử trong một hashtable bao gồm một khoá và một giá trị. Các phần tử được thêm vào bảng băm bằng cách sử dụng phương thức put(), và được truy lục bằng cách sử dụng phương thức get(). Các phần tử có thể được xoá từ một bảng băm với phương thức remove(). Các phương thức contains() và containsKey() có thể được sử dụng để tra cứu một giá trị hoặc một khoá trong bảng băm.

Một vài phương thức của Hashtable:

clear() (Xoá tất cả các phần tử từ bảng băm.)
clone() (Tạo một bản sao của Hashtable.)
contains(Object) (Trả về True nếu bảng băm chứa các đối tượng được chỉ định.)
containsKey(Object) (Trả về True nếu bảng băm chứa khoá được chỉ định.)
elements() (Trả về một tập hợp phần tử của bảng băm.)
get(Object key) (Trả về đối tượng có khoá được chỉ định.)

Ví dụ:

```
package hashTableExample;

import java.util.Hashtable;
import java.util.Iterator;
import java.util.Set;

public class HashTableExample {

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

```
//        Declare and instantiate the Hashtable.
          Hashtable<String, Integer> product = new Hashtable<>();

          String productName;
//        Insert some elements for hashtable
          product.put("Softdrink", 30);
          product.put("Beer", 40);
          product.put("Candy", 25);
          product.put("Cake", 33);

//        Show all balances in hashtable.
//        First, get a set view of the keys.
          Set<String> keys = product.keySet();

//        Get an Iterator.
          Iterator<String> iterator = keys.iterator();

//        Iterate.
          while (iterator.hasNext()) {
               productName = iterator.next();
               System.out.printf("%s: %d\n", productName,
product.get(productName));
          }

//        Get size of this hashtable
          System.out.println("Hashtable Size is: " +
product.size());

//        Check existing by boolean containsKey(Object key)
          boolean existKey = product.containsKey("Softdrink");
          boolean existValue = product.containsValue("100");
          System.out.println("Softdrink exists in Hashtable? : "
+ existKey);
          System.out.println("Candy has 100 boxes true or false:
" + existValue);

//        Insert more 20 cans into Beer
          int beerCan = product.get("Beer");
          product.put("Beer", beerCan + 20);

          System.out.println("Total of Beer Can is: " +
product.get("Beer") + " cans");
```

```
            System.out.println("Before removing Cake from
Hashtable: " + product.clone());
//          Remove a key value pair from Hashtable.
            product.remove("Cake");
            System.out.println("After removing Cake Hashtable
contains: " + product);
      }

}
```

Kết quả:

```
Candy: 25
Beer: 40
Softdrink: 30
Cake: 33
Hashtable Size is: 4
Softdrink exists in Hashtable? : true
Candy has 100 boxes true or false: false
Total of Beer Can is: 60 cans
Before removing Cake from Hashtable:
{Candy=25, Beer=60, Softdrink=30, Cake=33}
After removing Cake Hashtable contains:
{Candy=25, Beer=60, Softdrink=30}
```

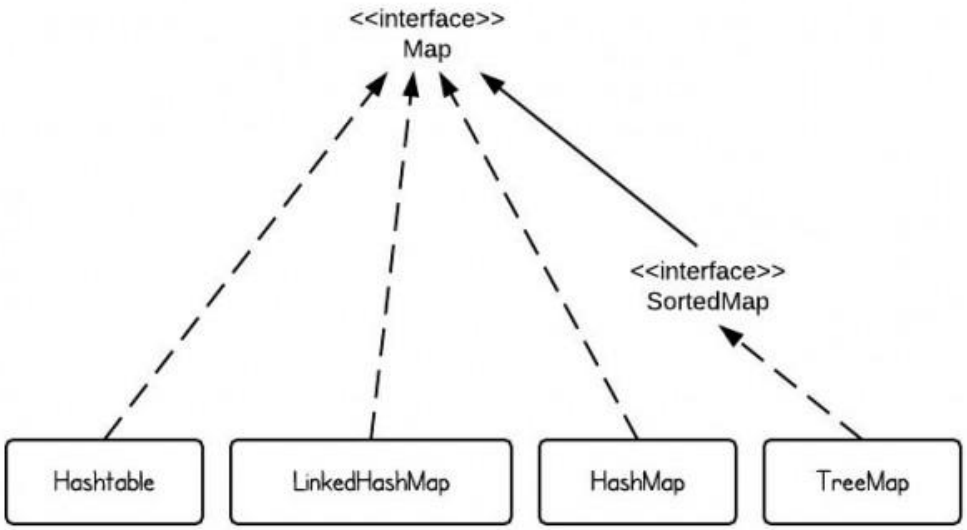
2.3. TreeMap

Lớp TreeMap trong Java thực thi Map Interface bởi sử dụng một tree.

Một TreeMap cung cấp các phương thức hiệu quả để lưu giữ các cặp key/value trong thứ tự được sắp xếp, và cho phép thu hồi nhanh chóng.

Như vậy Treemap lưu trữ các phần tử theo cấu trúc cây, các phần tử sắp xếp dựa trên giá trị của khóa

Không giống một Hashmap, các phần tử của Treemap được xếp thứ tự theo thứ tự key tăng dần hoặc bằng phương thức Comparator để thực hiện việc Sorted map



Ví dụ:

```
package treeMapExample;

import java.util.Iterator;
import java.util.Map;
import java.util.Map.Entry;
import java.util.Set;
import java.util.TreeMap;

public class TreeMapExample {

	public static void main(String[] args) {
//		Declare and instantiate TreeMap
		TreeMap<Integer, String> treeMap = new TreeMap<>();
```

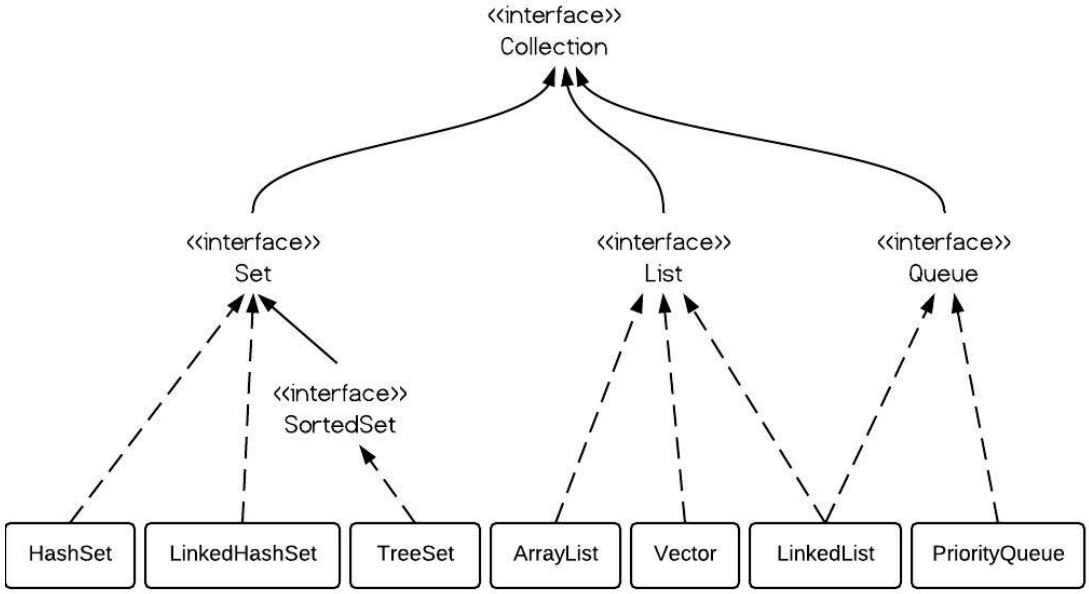
```
//        Adding elements to TreeMap
          treeMap.put(11, "Tuan Chien");
          treeMap.put(14, "Kim Tam");
          treeMap.put(28, "Hoang Dong");
          treeMap.put(19, "Duc Tran");
          treeMap.put(45, "Nhu Phung");

//        Using Iterator
          Set<Entry<Integer,String>> setTreeMap =
treeMap.entrySet();

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

          while (iterator.hasNext()) {
               Map.Entry<Integer, String> entryMap =
(Map.Entry<Integer, String>)iterator.next();

                System.out.print("Key is: "+ entryMap.getKey() +
" & Value is: ");
               System.out.println(entryMap.getValue());
          }

//        Get keys of TreeMap
          Set<Integer> keySet = treeMap.keySet();
          System.out.println("TreeMap keys: " + keySet);

//        Get Size of TreeMap
          System.out.println("Size of TreeMap : " +
treeMap.size());

//        Removing a element
          System.out.println(treeMap);
          treeMap.remove(28);
          System.out.println(treeMap);

     }

}
```

Output:

```
Key is: 11 & Value is: Tuan Chien
Key is: 14 & Value is: Kim Tam
Key is: 19 & Value is: Duc Tran
Key is: 28 & Value is: Hoang Dong
Key is: 45 & Value is: Nhu Phung
TreeMap keys: [11, 14, 19, 28, 45]
Size of TreeMap : 5
{11=Tuan Chien, 14=Kim Tam, 19=Duc Tran, 28=Hoang Dong, 45=Nhu
Phung}
{11=Tuan Chien, 14=Kim Tam, 19=Duc Tran, 45=Nhu Phung}
```

3. Set Interface

Set interface mở rộng từ Collection interface.
Trong Set không cho phép việc trùng lắp các phần tử, mọi phần tử trong Set phải là duy nhất.
Bạn có thể add một phần tử trùng lắp vào Set một cách đơn giản nhưng kết quả trả về sẽ tự động remove phần tử đó
Ta sẽ tìm hiểu thêm về một số lớp có liên quan trong Set: HashSet, TreeSet, LinkedHashSet



HashSet	TreeSet
Implement từ interface SET.	Implement từ interface SortedSet và SortedSet implement interface SET
Extends từ AbstractSet	
HashSet cho phép element null	TreeSet ngược lại, không đồng ý giá trị null
Default chưa được sắp xếp. Sắp xếp dựa trên việc implement Comparable or implement Comparator.	Defaul được sắp xếp
No-Duplicate các phần tử nghĩa là không có các phần tử trùng nhau	No-Duplicate các phần tử
So sánh các phần tử để phân biệt duplicate dựa trên Hashcode – Equal	So sánh và sắp xếp dựa trên Comparator or Comparable
HashSet cho hiệu suất performance nhanh hơn TreeSet	

3.1. HashSet

3.2. TreeSet

TreeSet cũng tương tự như HashSet ngoại trừ việc TreeSet thì quan tâm tới việc sắp xếp các phần tử theo thứ tự tăng dần trong khi HashSet thì không.

TreeSet không cho phép các phần tử null.

```
package treeSetExample;

import java.util.Iterator;
import java.util.TreeSet;

public class TreeSetExample {

     public static void main(String[] args) {

          TreeSet<Integer> treeSet = new TreeSet<Integer>();

          treeSet.add(23);
          treeSet.add(12);
```

```
            treeSet.add(43);
            treeSet.add(34);
            treeSet.add(54);
            treeSet.add(42);

            Iterator<Integer> iteratorSet = treeSet.iterator();
            System.out.printf("TreeSet data: ");

            while (iteratorSet.hasNext()) {
                 System.out.printf(iteratorSet.next() + " ");
            }

//          Check empty or not
            if (treeSet.isEmpty()) {
                 System.out.println("TreeSet is empty");
            } else {
                 System.out.println("\nTreeSet size is: " +
treeSet.size());
            }

//          Getting first element from TreeSet
            System.out.println("First Element: " +
treeSet.first());

//          Getting last element from TreeSet
            System.out.println("Last Element: " + treeSet.last());

//          Remove element by value
            if (treeSet.remove(42)) {
                 System.out.println("Data is removed from tree
set");
            } else {
                 System.out.println("Data doesn't exist!");
            }
            System.out.print("Now the tree set contain: ");
            iteratorSet = treeSet.iterator();

            while (iteratorSet.hasNext()) {
                 System.out.print(iteratorSet.next() + " ");
            }
```

```
            System.out.println("\nNow the size of tree set: " +
treeSet.size());

//          Remove all
            treeSet.clear();
            if (treeSet.isEmpty()) {
                 System.out.print("Tree Set is empty.");
            } else {
                 System.out.println("Tree Set size: " +
treeSet.size());
            }
       }

}
```

Kết quả:

```
TreeSet data: 12 23 34 42 43 54
TreeSet size is: 6
First Element: 12
Last Element: 54
Data is removed from tree set
Now the tree set contain: 12 23 34 43 54
Now the size of tree set: 5
Tree Set is empty.
```

4. Enum

Enum là một kiểu dữ liệu đặc biệt, về cơ bản enum là một tập các hằng số. Đây là một ví dụ về enum:

```
public enum Planets {
        Mercury,
        Venus,
        Earth,
        Mars,
        Jupiter,
        Saturn,
        Uranus,
        Neptune;
    }
```

Ở đây Planets là một kiểu enum, là một tập chứa 8 hằng số Mercury, Venus, Earth, Mars, Jupiter, Saturn, Uranus, Neptune

Gán giá trị cho kiểu enum.

```
Planets planets = Planets.Earth;
```

Biến planets có kiểu là Planets, có thể nhận bất kỳ giá trị nào trong 8 giá trị (Mercury, Venus, Earth, Mars, Jupiter, Saturn, Uranus, Neptune). Trong ví dụ trên dir nhận giá trị Earth.

Ví dụ sử dụng enum trong mệnh đề if-else.

```
package enumExample;

public class EnumExample {

    public enum Planets {
        Mercury,
        Venus,
        Earth,
        Mars,
        Jupiter,
        Saturn,
        Uranus,
        Neptune;
    }
```

```
    public static void main(String[] args) {
//        Assigning values
        Planets planets = Planets.Earth;

        if (planets == Planets.Mercury) {
            System.out.println("This planet is: Mercury");
        } else if (planets == Planets.Venus) {
            System.out.println("This planet is: Venus");
        } else if (planets == Planets.Earth) {
            System.out.println("This planet is: Earth");
        } else if (planets == Planets.Mars) {
            System.out.println("This planet is: Mars");
        } else if (planets == Planets.Jupiter) {
            System.out.println("This planet is: Jupiter");
        } else if (planets == Planets.Saturn) {
            System.out.println("This planet is: Saturn");
        } else if (planets == Planets.Uranus) {
            System.out.println("This planet is: Uranus");
        } else {
            System.out.println("This planet is: Neptune");
        }

    }

}
```

Kết quả: This planet is: Earth

II. Java.Lang

Java.lang: Cung cấp các hàm thao tác trên các kiểu dữ liệu cơ bản, xử lí lỗi và ngoại lệ, xử lí vào ra trên các thiết bị chuẩn như bàn phím và màn hình. Trong đó có các lớp quan trọng khác như String, StringBuffer, StringBuilder

Khi làm việc với các dữ liệu văn bản, Java cung cấp cho bạn 3 class String, StringBuffer và StringBuilder. Nếu làm việc với các dữ liệu lớn bạn nên sử dụng StringBuffer hoặc StringBuilder để đạt hiệu năng nhanh nhất. Về cơ bản 3 class này có nhiều điểm giống nhau.

1. String Class

Lớp String là lớp immutable và final, với đặc điểm này, một đối tượng kiểu String là không thể thay đổi được. Nếu thực hiện thay đổi một đối tượng String thì kết quả sẽ sinh ra một đối tượng mới vì thế sinh ra nhiều đối tượng rác, làm tốn bộ nhớ.

- Các chuỗi được đặt trong dấu nháy kép, như: “abcd”, “xyz”… và các chuỗi này được tạo trong String pools.
- Khi so sánh hai chuỗi bấy kỳ dùng toán tử “==” thì kết quả trả về luôn là true vì chúng đều là thể hiện của một lớp String. Lưu ý, so sánh giữa hai đối tượng luôn luôn dùng phương thức equals().
- Toán tử “+” dùng để nối hai chuỗi
- Lớp String override hai phương thức equals() và hashcode() và hai chuỗi được xem là giống nhau nếu chúng có cùng số lượng ký tự, thứ tự các ký tự trong hai chuỗi giống nhau, cùng chữ in(in hoa hoặc in thường).
- Phương thức toString() trả về một chuỗi của một đối tượng kiểu String.

```
package stringClassExample;

public class StringClassExample {

	public static void main(String[] args) {

		String a = "You will see output";
		String b = "You will see output";

//		It is important to use equals for string comparison.
```

```
            if (a.equals(b)) {
                System.out.println("a and b are equal");
            } else {
                System.out.println("Not equal");
            }

//          Should never use '==' to compare any two Objects
            String c = new String("You will see output");
            if (a == c) {
                System.out.println("a and c are equal");
            } else {
                System.out.println("a and c are NOT equal\n");
            }

//          Case sensitive comparison
            String upString = "MY LAPTOP IS LENOVO";
            String lowString = "my laptop is lenovo";

            if (upString.equals(lowString)) {
                System.out.println("UpString and LowString are
equal");
            } else {
                System.out.println("UpString and LowString are NOT
equal by using equal");
            }

//          Using equalsIngoreCase
            if (upString.equalsIgnoreCase(lowString)) {
                System.out.println("UpString and LowString are
equal by using equalsIngoreCase");
            } else {
                System.out.println("UpString and LowString are NOT
equal");
            }

//          Concatenation using + operator
            String firstString = "I am a";
            String secondString = "Dev";
            secondString = firstString + " " + secondString;
            System.out.println("Result: " + secondString);
        }
}
```

```
Output:
a and b are equal
a and c are NOT equal

UpString and LowString are NOT equal by using equal
UpString and LowString are equal by using equalsIngoreCase
Result: I am a Dev
```

Nhược điểm của lớp String trong Java

Đặc điểm immutable là điểm mạnh nhất của lớp String nhưng cũng là điểm yếu nhất của nó nếu chúng ta dùng String không đúng. Những thao tác: nối chuỗi, trích xuất chuỗi(substring()), lowercase, uppercase…đều tạo ra chuỗi mới và đối tượng rác trong bộ nhớ heap.

Nếu bạn lạm dụng các đối tượng String trong khi lập trình bạn sẽ thấy nó sinh ra rất nhiều đối tượng rác không được quản lý dẫn đến hiệu suất chương trình thấp.

Giải pháp cho vấn đề này là sử dụng các lớp StringBuffer hoặc StringBuilder để thay thế String

2. StringBuffer Class

StringBuffer là có thể thay đổi (mutable). Nó có thể thay đổi về độ dài và nội dung. StringBuffers là luồng an toàn (thread-safe), có nghĩa là nó đã đồng bộ các method để kiểm soát truy cập, tại một thời điểm chỉ có một thread có thể truy cập vào **StringBuffer**. Vì vậy, các đối tượng**StringBuffer** thường an toàn để sử dụng trong một môi trường đa luồng (multiple thread) có thể cố gắng để truy cập vào cùng một đối tượng**StringBuffer** cùng một lúc.

Ví dụ:

```
package stringBufferExample;

public class StringBufferClass {

	public static void main(String[] args) {

		StringBuffer stringBuffer = new StringBuffer();

//		Append the string representation of the argument to the
end of the buffer.
```

```
            stringBuffer.append("Happy New Year!...");
            System.out.println(stringBuffer.toString());

//          Delete the specified substring by providing
//          the start and the end of the sequence.
            stringBuffer.delete(15, 18);
            System.out.println(stringBuffer.toString());

//          Delete just one char by providing its position.
            stringBuffer.deleteCharAt(14);
            System.out.println(stringBuffer.toString());

//          Insert a string in a specified place inside the buffer.
            stringBuffer.insert(14, " 2016");
            System.out.println(stringBuffer.toString());
      }

}
```

3. StringBuilder Class

StringBuilder Class tương tự như StringBuffer Class tuy nhiên nó không đồng bộ, không phải là luồng an toàn (thread-safe). Bởi không được đồng bộ, hiệu suất của **StringBuilder** là tốt hơn so với **StringBuffer**. Vì vậy, nếu bạn đang làm việc trong một môi trường đơn luồng, sử dụng **StringBuilder** thay vì **StringBuffer** sẽ có hiệu suất tốt hơn.

Ví dụ:

```
package stringBuilderExample;

public class StringBuilderClass {

      public static void main(String[] args) {

            StringBuilder stringBuilder = new StringBuilder();
            stringBuilder.append("I am Tuan Chien");
            System.out.println("StringBuilder just appends a
String: " + stringBuilder);

//          Append a character
            char charInText = '!';
```

```
            stringBuilder.append(charInText);
            System.out.println("StringBuilder continues to add a
char: " + stringBuilder);

//          Insert my family name
            stringBuilder.insert(4, " Vu");
            System.out.println("StringBuilder after inserting: " +
stringBuilder);

//          StringBulder with a initialized capacity
            StringBuilder secondStringBuilder = new
StringBuilder();
            secondStringBuilder.append(123456789);
            System.out.println("StringBuilder with length: "
            + secondStringBuilder.length()
            + " and capacity: " + secondStringBuilder.capacity()
            + " appends an int: " + secondStringBuilder);

//          Delete 456
            secondStringBuilder.delete(3, 6);
            System.out.println("StringBuilder after deleting: " +
secondStringBuilder);

      }
}
```

Output:

```
StringBuilder just appends a String: I am Tuan Chien
StringBuilder continues to add a char: I am Tuan Chien!
StringBuilder after inserting: I am Vu Tuan Chien!
StringBuilder with length:
9 and capacity: 16 appends an int: 123456789
StringBuilder after deleting: 123789
```

4. So Sánh String, StringBuilder và StringBuffer

- Mọi thao tác(ví dụ như cắt chuỗi, cộng chuỗi, in hoa, in thường, ...) trên String sẽ tạo ra một class mới còn StringBuffer thì không.
- Method StringBuffer.append() được sử dụng để String concatenation trong Java.
- StringBuffer và String trong Java là không cùng phân cấp. Nó có nghĩa là bạn không thể cast từ String sang StringBuffer và ngược lại.
- String được lưu trữ trong String Pool, StringBuffer được lưu vào heap.
- Thực hiện nối chuỗi, String sẽ sử dụng toán phép +, StringBuffer sử dụng methodStringBuffer.append() để làm việc này.
- Về hiệu năng thực thi nối chuỗi, StringBuffer có điểm tối ưu cao hơn so với String. Có nghĩa là StringBuffer thực hiện nhanh hơn so với String.
- StringBuilder và StringBuffer là giống nhau, nó chỉ khác biệt tình huống sử dụng có liên quan tới đa luồng (Multi Thread).
- Nếu sử lý văn bản sử dụng nhiều luồng (Thread) bạn nên sử dụng StringBuffer để tránh tranh chấp giữa các luồng.
- Nếu sử lý văn bản sử dụng 1 luồng (Thread) nên sử dụng StringBuilder.
- Nếu so sánh về tốc độ sử lý StringBuilder là tốt nhất, sau đó StringBuffer và cuối cùng mới là String.

StringBuffer	StringBuilder
Thread safe	Not thread safe
Synchronized	Not synchronized
Slower than StringBuilder	Faster than StringBuffer

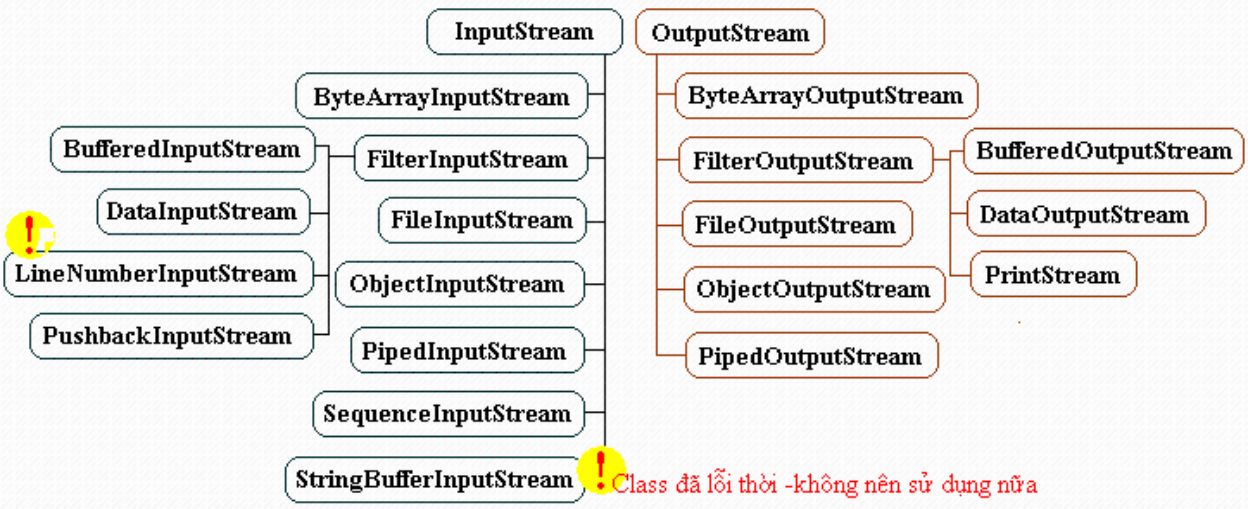
Ví dụ:

```
package compareJavaLang;

public class StringvsStringBuffervsStringBuilder {

	public static void main(String[] args) {

		long startTime;
		String text = "Compare";
		StringBuffer stringBuffer = new StringBuffer();
		StringBuilder stringBuilder = new StringBuilder();
		System.out.println(text);
//		Using String
		startTime = System.currentTimeMillis();
		for (int i = 0; i < 10000; i++) {
			text += "JavaLang";
		}
		System.out.println("Time using String: " +
(System.currentTimeMillis() - startTime) + "ms.");

//		 Using StringBuffer
		startTime = System.currentTimeMillis();
		for (int i = 0; i < 10000; i++) {
			stringBuffer.append("JavaLang");
		}
		System.out.println("Time using StringBuffer: " +
(System.currentTimeMillis() - startTime) + "ms.");

//		 Using StringBuilder
		startTime = System.currentTimeMillis();
		for (int i = 0; i < 10000; i++) {
			stringBuilder.append("JavaLang");
		}
		System.out.println("Time using StringBuilder: " +
(System.currentTimeMillis() - startTime) + "ms.");
	}
}
```

Output:

```
		Compare
		Time using String: 582ms.
		Time using StringBuffer: 2ms.
		Time using StringBuilder: 1ms.
```

III. Java.IO



Java định nghĩa hai kiểu luồng: byte và ký tự

- Luồng byte (hay luồng dựa trên byte) hỗ trợ việc xuất nhập dữ liệu trên byte, thường được dùng khi đọc ghi dữ liệu nhị phân.
- Luồng ký tự được thiết kế hỗ trợ việc xuất nhập dữ liệu kiểu ký tự (Unicode).
- Trong một vài trường hợp luồng ký tự sử dụng hiệu quả hơn luồng byte, nhưng ở mức hệ thống thì tất cả những xuất nhập đều phải qui về byte.
- Luồng ký tự hỗ trợ hiệu quả chỉ đối với việc quản lý, xử lý các ký tự.

Tóm tắt về xử lý file: Khi xử lý đọc và ghi dữ liệu file:

- Nên dùng DataInputStream và DataOutputStream để nhập/xuất các dữ liệu kiểu sơ cấp (int, float...)
- Nên dùng ObjectInputStream và ObjectOutputStream để nhập/xuất các đối tượng(object).
- Nên kết hợp luồng file và luồng đọc/ghi ký tự để nhập xuất các file ký tự Unicode.
- Nên dùng RandomAccessFile nếu muốn đọc/ghi ngẫu nhiên trên file.
- Dùng lớp File để lấy thông tin về file

1. FileInputStream

Lớp này cho phép đọc đầu vào từ một tập tin dưới dạng một luồng. Các đối tượng của lớp này được tạo ra nhờ dùng một tập tin String, File, hoặc một đối tượng FileDescriptor làm một đối số. Lớp này chồng lên các phương thức của lớp InputStream.

Ví dụ:

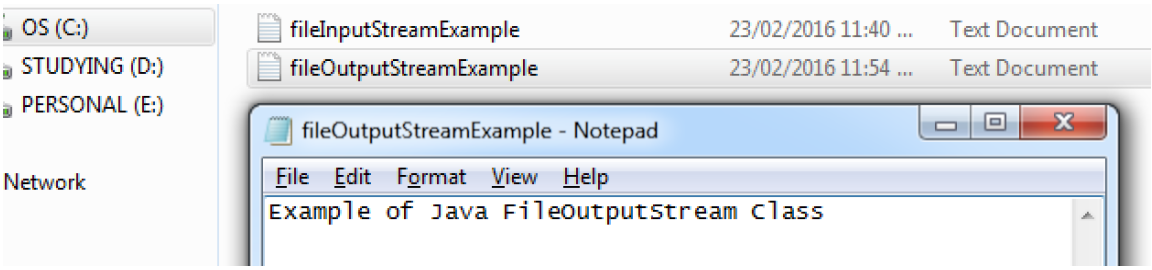
```
package fileInputStream;

import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.IOException;

public class FileInputStreamExample {

	public static final String INPUT_FILE =
"C:\\fileInputStreamExample.txt";

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

		FileInputStream fileInputStream = null;
		try {
			fileInputStream = new FileInputStream(INPUT_FILE);
			System.out.println("Available: " +
fileInputStream.available());

//			Read a single byte
			int result = fileInputStream.read();
			System.out.println("Read byte: " + result);

			int content;
			while ((content = fileInputStream.read()) != -1) {
				// convert to char and display it
				System.out.print((char) content);
			}
		} catch (FileNotFoundException e) {
			System.out.println("File Not Found");
		} catch (IOException e1) {
			System.out.println("Error when read file");
		} finally {
```

```
//              close file
                if (fileInputStream != null) {
                     fileInputStream.close();
                }
           }
     }
}
```

2. FileOutputStream

Lớp này cho phép ghi kết xuất ra một luồng tập tin. Các đối tượng của lớp này cũng tạo ra sử dụng các đối tượng chuỗi tên tập tin, tập tin, FileDesciptor làm tham số.

```
package fileOutputStream;

import java.io.FileOutputStream;

public class FileOutputStreamExample {

     public static final String OUTPUT_FILE =
"C:\\fileOutputStreamExample.txt";

     public static void main(String[] args) {
          try {
                FileOutputStream fileOutputStream = new
FileOutputStream(OUTPUT_FILE);
                String content = "Example of Java FileOutputStream
Class";

//              converting string into byte array
                byte[] bytes = content.getBytes();
                fileOutputStream.write(bytes);
                fileOutputStream.close();
                System.out.println("Success...");
           } catch (Exception e) {
                System.out.println("Can not write");
           }
     }
}
```



3. BufferedInputStream

Việc truy xuất tập tin trực tiếp trên đĩa có thể gây ra chậm chạp, do mỗi lần ta thực hiện thao tác ghi hay đọc một ký tự từ tập tin, chương trình lại phải dò tìm trên đĩa. Có một cơ chế khác nhanh hơn là sử dụng bộ đệm.

Thường thì việc sử dụng bộ đệm chủ yếu dành cho các thao tác đối với tập tin, nhưng ta vẫn có thể dùng nó với bất kỳ luồng con nào khác được dẫn xuất từ lớp InputStream bởi BufferedInputStream là một lớp luồng trung gian

Ví dụ:

```
package bufferedInputStream;

import java.io.BufferedInputStream;
import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;

public class BufferedInputStreamExample {

	public static final String INPUT_FILE =
"C:\\BufferedInputStream.txt";

	public static void main(String[] args) {
		File file = new File(INPUT_FILE);
		FileInputStream fileInputStream = null;
		BufferedInputStream bufferedInputStream = null;
		try {
			fileInputStream = new FileInputStream(file);
			bufferedInputStream = new
BufferedInputStream(fileInputStream);

			//process each character at a time
			while (bufferedInputStream.available() > 0) {
```

```
            System.out.print((char)bufferedInputStream.read());
                        }
                    } catch (IOException e) {
                        e.printStackTrace();
                        } finally {
                            try {
                                fileInputStream.close();
                                bufferedInputStream.close();
                                } catch (IOException ex) {
                                    ex.printStackTrace();
                                    }
                            }
                }
}
```

4. BufferedOutputStream

Ngược với quá trình đọc dữ liệu từ tập tin lên, quá trình ghi dữ liệu xuống tập tin cũng vậy, nếu ta cứ ghi liên tục từng ký tự một xuống tập tin vậy lý trên đĩa thì sẽ tỏ ra chậm chạp. Java cung cấp cơ chế ghi vào bộ đệm, ta chỉ việc kết nối tập tin với luồng BufferedOutputStream.

Tuy nhiên, không phải cứ ghi vào bộ đệm là dữ liệu được an toàn nằm trên tập tin mà phải mất một thời gian. Do đó nếu muốn chắc chắn thì phải gọi thêm phương thức flush().

```
package bufferedOutputStream;

import java.io.BufferedOutputStream;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;

public class BufferedOutputStreamExample {

    public static final String OUTPUT_FILE =
"C:\\BufferedOutputStream.txt";
```

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

          BufferedOutputStream bufferedOutputStream = null;
          FileOutputStream fileOutputStream = null;
          try {
//             Create FileOutputStream from
BufferedOutputStream.txt
               fileOutputStream = new
FileOutputStream(OUTPUT_FILE);

//             Create BufferedOutputStream for FileOutputStream
               bufferedOutputStream = new
BufferedOutputStream(fileOutputStream);

               String contents = "This is my Buffered Output
Stream Example";
               bufferedOutputStream.write(contents.getBytes());
               System.out.println("Success");

          } catch (FileNotFoundException e) {
               System.out.println("File not found");
          } catch (IOException ioException) {
               System.out.println("Error when writing file");
          } finally {
               if (bufferedOutputStream != null) {
                    try {
                         bufferedOutputStream.flush();
                         bufferedOutputStream.close();
                    } catch (Exception e) {
                         System.out.println("Error when closing
streams");
                    }
               }
          }

     }
}
```