Java Class

Kiến thức về các Class trong Java

- Java String class - Lớp String trong Java
- Java StringBuffer class - Lớp StringBuffer trong Java
- Java StringBuilder class - Lớp StringBuilder trong Java
- Java Math class - Lớp java.lang.Math trong Java
- Java Runtime class - Lớp Runtime trong Java
- Java System class - Lớp System trong Java
- Java Class class - Lớp Class trong Java
- Java Object class - Lớp Object trong Java
- Java Hashtable class - Lớp Hashtable trong Java
- Java Random class - Lớp Random trong Java
- Java Vector class - Lớp Vector trong Java
- Java StringTokenizer class - Lớp StringTokenizer trong Java

Java String class - Lớp String trong Java

1. Khái niệm

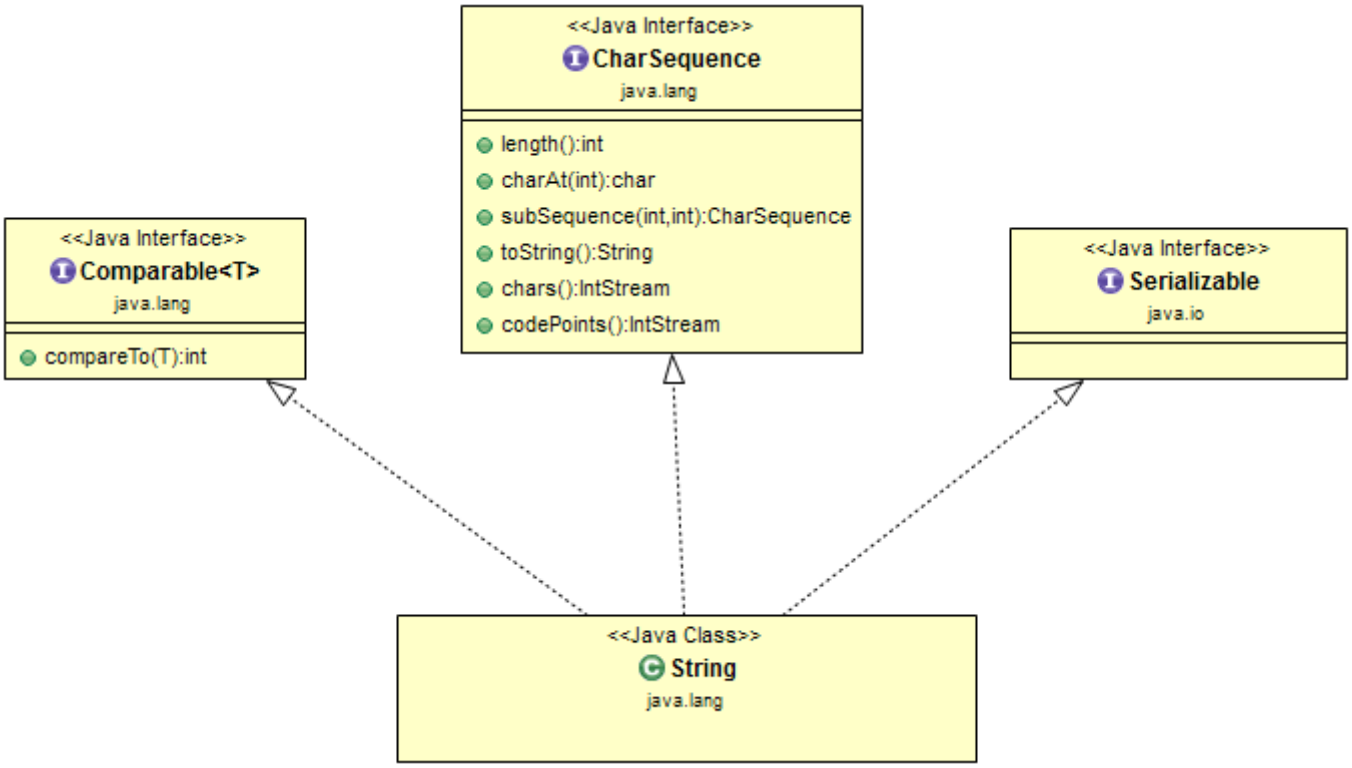
String là một chuỗi các ký tự. Lớp String cung cấp hàng loạt các phương thức để thao tác với các chuỗi. Nó cung cấp các phương thức khởi tạo (constructor) khác nhau.

Package: java.lang

```
new String(byte[] byte_arr);
new String(byte[] byte_arr, Charset char_set)
new String(byte[] byte_arr, String char_set_name)
new String(byte[] byte_arr, int start_index, int length)
new String(byte[] byte_arr, int start_index, int length, Charset char_set)
new String(byte[] byte_arr, int start_index, int length, String char_set_name)
new String(char[] char_arr)
new String(char[] char_array, int start_index, int count)
new String(int[] uni_code_points, int offset, int count)
new String(StringBuffer s_buffer)
new String(StringBuilder s_builder)
```

Lớp String thuộc gói `java.lang`

Lớp java.lang.String được implements từ các interface Serializable, Comparable and CharSequence.



Ví dụ:

```
String str1 = new String( );
//str1 chứa một dòng tr□ng.
String str2 = new String("Hello World");
//str2 chứa dòng "Hello World"
char ch[] = {'A','B','C','D','E'};
String str3 = new String(ch);
//str3 chứa "ABCDE"
String str4 = new String(ch,0,2);
//str4 chứa "AB" vì 0- tính từ ký tự b□t đ□u,  2- là s□ lượng ký tự k□ từ ký tự b□t đ□u.
```

2. String pool

Một chương trình Java có thể chứa nhiều chuỗi bằng chữ. "String Pool" đại diện cho tất cả các chữ được tạo trong chương trình. Mỗi khi một chuỗi bằng chữ được tạo, String Pool tìm kiếm để nhìn thấy nếu chuỗi bằng chữ tồn tại. Nếu nó tồn tại, một thể hiện mới được gán đến một chuỗi mới.

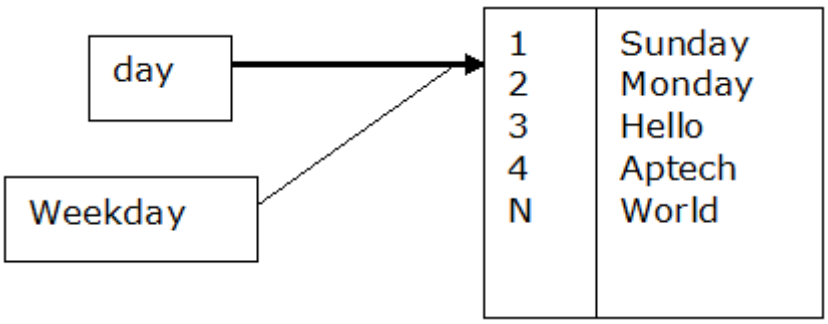
Ví dụ:

```
String day = "Monday";
String weekday = "Monday";
```

Ở đây, một thể hiện cho biến "day", biến đó có giá trị là "Monday", được tạo trong String Pool. Khi chuỗi bằng chữ "weekday" được tạo, việc lưu giữ các giá trị giống nhau như của biến "day", một thể

hiện đang tồn tại được gán đến biến "weekday". Vì cả hai biến "day" và "weekday" cũng đều nhằm chỉ vào chuỗi tương tự trong String Pool.

Hình ảnh sau minh hoạ khái niệm của "String Pool".



Theo ví dụ trên, chỉ có một đối tượng chuỗi “Monday” được tạo ra. Biến `day` và `weekday` được tham chiếu đến đối tượng chuỗi “Monday”.

3. Các phương thức lớp String

Trong phần này, chúng ta sẽ xem xét các phương thức của lớp String.

+ charAt()

Phương thức này trả về một ký tự tại một vị trí đặc biệt trong một chuỗi.

Ví dụ:

```
String name = new String("Java Language");
char ch = name.charAt(5);
```

Biến "ch" chứa giá trị "L", từ đó vị trí các số bắt đầu từ 0.

+ startsWith()

Phương thức này trả về giá trị kiểu logic (Boolean), phụ thuộc vào chuỗi có bắt đầu với một giá trị đặc biệt không.

Ví dụ:

```
String strname = "Java Language";
boolean flag = strname.startsWith("Java");
```

Biến "flag" chứa giá trị true.

+ endsWith()

Phương thức này trả về một giá trị kiểu logic (boolean), có chăng phụ thuộc vào chuỗi kết thúc với một giá trị đặc biệt.

Ví dụ:

```
String strname = "Java Language";
boolean flag = strname.endsWith("Java");
```

Biến "flag" chứa giá trị false.

+ copyValueOf()

Phương thức này trả về một chuỗi được rút ra từ một mảng ký tự được truyền như một đối số. Phương thức này cũng lấy hai tham số nguyên. Tham số đầu tiên chỉ định vị trí từ nơi các ký tự phải được rút ra, và tham số thứ hai chỉ định số ký tự được rút ra từ mảng.

Ví dụ:

```
char name[] = {'L','a','n','g','u','a','g','e'};
String subname = String .copyValueOf(name,5,2);
```

Bây giờ biến "subname" chứa chuỗi "ag".

+ toCharArray()

Phương thức này lấy một chuỗi, và chuyển nó vào một mảng ký tự. Ví dụ:

```
String text = new String("Hello World");
Char textArray[] = text.toCharArray();
```

+ indexOf()

Phương thức này trả về thứ tự của một ký tự đặc biệt, hoặc một chuỗi trong phạm vi một chuỗi. Các câu lệnh sau biểu diễn các cách khác nhau của việc sử dụng hàm.

```
String day = new String("Sunday");
int index1 = day.indexOf('n');
//chứa 2

int index2 = day.indexOf('z',2);
//chứa –1 nếu "z" không tìm thấy tại vị trí 2.

int index3 = day.indexOf("Sun");
//chứa mục 0 của mẫu
```

+ toUpperCase()

Phương thức này trả về chữ hoa của chuỗi thông qua hàm.

```
String lower = new String("good morning");
System.out.println("Uppercase: "+lower.toUpperCase( ));
```

+ toLowerCase()

Phương thức này trả về chữ thường của chuỗi thông qua hàm.

```
String upper = new String("LAPTRINH.VN");
System.out.println("Lowercase: "+upper.toLowerCase( ));
```

+ trim()

Phương thức này cắt bỏ khoảng trắng trong đối tượng String. Hãy thử đoạn mã sau để thấy sự khác nhau trước và sau khi cắt bỏ khoảng trắng.

```
String space = new String(" Spaces ");
System.ut.println(spaces);
System.out.println(spaces.trim()); //Sau khi c□t b□ kho□ng tr□ng
```

+ equals()

Phương thức này so sánh nội dung của hai đối tượng chuỗi.

```
String name1 = "Laptrinh.vn", name2 = "LAPTRINH.VN";
boolean flag = name1.equals(name2);
```

Biến "flag" chứa giá trị false.

4. Các phương thức của lớp String trong Java

SN	Methods	Description
1	char charAt(int index)	Trả về một ký tự tại vị trí có chỉ số được chỉ định.
2	int compareTo(Object o)	So sánh một String với một Object khác.
3	int compareTo(String anotherString)	So sánh hai chuỗi theo từ điển. (Phân biệt chữ hoa chữ thường)
4	int compareToIgnoreCase(String str)	So sánh hai chuỗi theo từ điển. (Không phân biệt chữ hoa chữ thường)
5	String concat(String str)	Nối chuỗi được chỉ định đến cuối của chuỗi này.

6	boolean contentEquals(StringBuffer sb)	Trả về true nếu và chỉ nếu chuỗi này đại diện cho cùng một chuỗi ký tự như là StringBuffer quy định.
7	static String copyValueOf(char[] data)	Trả về một chuỗi đại diện cho chuỗi ký tự trong mảng quy định.
8	static String copyValueOf(char[] data, int offset, int count)	Trả về một chuỗi đại diện cho chuỗi ký tự trong mảng quy định.
9	boolean endsWith(String suffix)	Kiểm tra nếu chuỗi này kết thúc với hậu tố quy định.
10	boolean equals(Object anObject)	So sánh với một đối tượng
11	boolean equalsIgnoreCase(String anotherString)	So sánh với một String khác, không phân biệt chữ hoa chữ thường.
12	byte[] getBytes()	Mã hóa chuỗi này thành một chuỗi các byte bằng cách sử dụng bảng mã mặc định của flatform (nền tảng), lưu trữ kết quả vào một mảng byte mới.
13	byte[] getBytes(String charsetName)	Mã hóa chuỗi này thành một chuỗi các byte bằng cách sử dụng bảng mã cho trước, lưu trữ kết quả vào một mảng byte mới.
14	void getChars(int srcBegin, int srcEnd, char[] dst, int dstBegin)	Copy các ký tự từ chuỗi này vào mảng ký tự đích.
15	int hashCode()	Trả về một mã "hash code" cho chuỗi này.
16	int indexOf(int ch)	Trả về chỉ số trong chuỗi này xuất hiện đầu tiên của ký tự cụ thể.
17	int indexOf(int ch, int fromIndex)	Trả về chỉ số trong chuỗi này xuất hiện đầu tiên của ký tự được chỉ định, bắt đầu tìm kiếm từ chỉ số cụ thể đến cuối.
18	int indexOf(String str)	Trả về chỉ số trong chuỗi này xuất hiện đầu tiên của chuỗi quy định.
19	int indexOf(String str, int fromIndex)	Trả về chỉ số trong chuỗi này xuất hiện đầu tiên của chuỗi quy định, bắt đầu từ chỉ số xác định.
20	String intern()	Returns a canonical representation for the string object.
21	int lastIndexOf(int ch)	Trả về chỉ số trong chuỗi này về sự xuất hiện cuối cùng của ký tự cụ thể.
22	int lastIndexOf(int ch, int fromIndex)	Trả về chỉ số trong chuỗi này về sự xuất hiện cuối cùng của ký tự được chỉ định, tìm kiếm lùi lại bắt đầu từ chỉ số xác định.
23	int lastIndexOf(String str)	Trả về chỉ số trong chuỗi này xảy ra cuối cùng bên phải của chuỗi quy định.
24	int lastIndexOf(String str, int fromIndex)	Trả về chỉ số trong chuỗi này về sự xuất hiện cuối cùng của chuỗi xác định, tìm kiếm lùi lại bắt đầu từ chỉ số xác định.
25	int length()	Trả về độ dài chuỗi.
26	boolean matches(String regex)	Kiểm tra chuỗi này khớp với biểu thức chính quy chỉ định hay không.

27	boolean regionMatches(boolean ignoreCase, int toffset, String other, int ooffset, int len)	Kiểm tra chuỗi có một phần giống nhau.
28	boolean regionMatches(int toffset, String other, int ooffset, int len)	Kiểm tra chuỗi có một phần giống nhau.
29	String replace(char oldChar, char newChar)	Trả về một chuỗi mới từ thay thế tất cả các lần xuất hiện của ký tự oldChar trong chuỗi này với ký tự newChar.
30	String replaceAll(String regex, String replacement)	Thay thế tất cả các chuỗi con của chuỗi này khớp với biểu thức chính quy bởi String mới replacement
31	String replaceFirst(String regex, String replacement)	Thay thế chuỗi con đầu tiên của chuỗi này khớp với biểu thức chính quy bởi một String mới replacement
32	String[] split(String regex)	Tách chuỗi này thành các chuỗi con, tại các chỗ khớp với biểu thức chính quy cho trước.
33	String[] split(String regex, int limit)	Tách chuỗi này thành các chuỗi con, tại các chỗ khớp với biểu thức chính quy cho trước. Tối đa limit chuỗi con.
34	boolean startsWith(String prefix)	Kiểm tra nếu chuỗi này bắt đầu với tiền tố quy định.
35	boolean startsWith(String prefix, int toffset)	Kiểm tra nếu chuỗi này bắt đầu với tiền tố quy định bắt đầu một chỉ số xác định.
36	CharSequence subSequence(int beginIndex, int endIndex)	Trả về một chuỗi ký tự mới là một dãy con của dãy này.
37	String substring(int beginIndex)	Trả về một chuỗi ký tự mới là một dãy con của dãy này. Từ chỉ số cho trước tới cuối
38	String substring(int beginIndex, int endIndex)	Trả về một chuỗi ký tự mới là một dãy con của dãy này. Từ chỉ số bắt đầu cho tới chỉ số cuối.
39	char[] toCharArray()	Chuyển chuỗi này thành mảng ký tự.
40	String toLowerCase()	Chuyển tất cả các ký tự của chuỗi này sang chữ thường, sử dụng miền địa phương mặc định (default locale)
41	String toLowerCase(Locale locale)	Chuyển tất cả các ký tự của chuỗi này sang chữ thường, sử dụng miền địa phương (locale) cho trước.
42	String toString()	Trả về String này.
43	String toUpperCase()	Chuyển tất cả các ký tự của chuỗi này sang chữ hoa, sử dụng miền địa phương mặc định (default locale)
44	String toUpperCase(Locale locale)	Chuyển tất cả các ký tự của chuỗi này sang chữ hoa, sử dụng miền địa phương (locale) cho trước.
45	String trim()	Trả về một String mới, sau khi loại bỏ các ký tự trắng (whitespace) bên trái và bên phải.
46	static String valueOf(primitive data type x)	Returns the string representation of the passed data type argument.

5. Tính đối tượng và vừa có tính nguyên thủy của String

- Tính nguyên thủy:

Bạn có thể tạo một string literal (chuỗi chữ), string literal được lưu trữ trong ngăn xếp (stack), đòi hỏi không gian lưu trữ ít hơn khi thao tác.

```
String literal = "Hello World";
```

Bạn có thể sử dụng toán tử + để nối 2 string, toán tử này vốn quen thuộc và sử dụng cho các kiểu dữ liệu nguyên thủy int, float, double.

Các string literal được chứa trong một bể chứa (String pool). Như vậy hai string literal có nội dung giống nhau sử dụng chung một vùng bộ nhớ trên stack, điều này giúp tiết kiệm bộ nhớ.

- Tính đối tượng:

Vì String là một class, vì vậy nó có thể được tạo ra thông qua toán tử new.

```
String object = new String("Hello World");
```

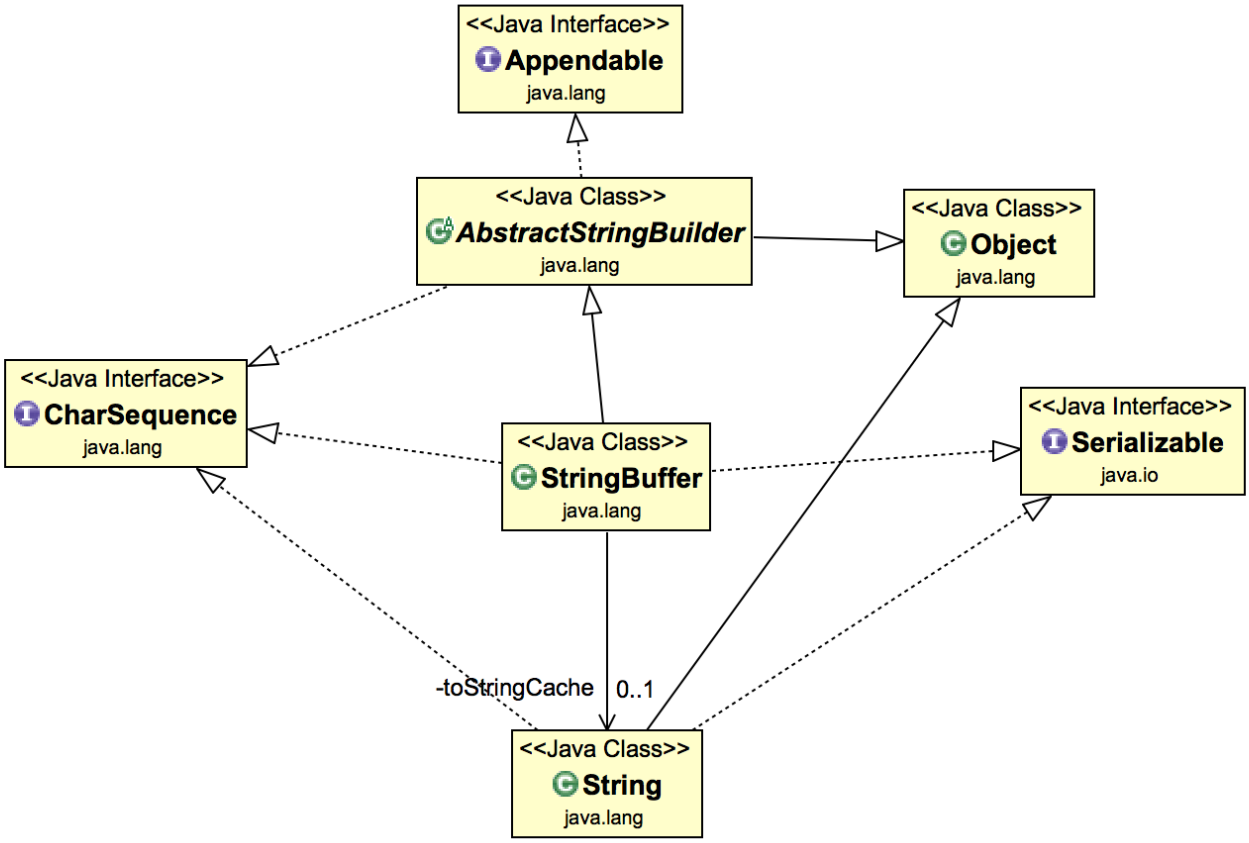
Các đối tượng String được lưu trữ trên Heap, yêu cầu quản lý bộ nhớ phức tạp và tốn không gian lưu trữ. Hai đối tượng String có nội dung giống nhau lưu trữ trên 2 vùng bộ nhớ khác nhau của Heap.

Java StringBuffer class - Lớp StringBuffer trong Java

1. Khái niệm

Lớp StringBuffer cung cấp các phương thức khác nhau để thao tác một đối tượng dạng chuỗi. Lớp StringBuffer được sử dụng để tạo chuỗi có thể thay đổi (mutable). Lớp StringBuffer trong java tương tự như lớp String ngoại trừ nó có thể thay đổi.

Package: java.lang



2. Khởi tạo đối tượng lớp StringBuffer

- StringBuffer(): Tạo ra một bộ đệm chuỗi với dung lượng ban đầu là 16.
- StringBuffer(String str): Tạo ra một bộ đệm chuỗi với chuỗi cụ thể.
- StringBuffer(int capacity): Tạo ra một bộ đệm chuỗi với dung lượng được chỉ định như độ dài chuỗi.

Ví dụ:

```
class StringBufferCons {

    public static void main(String args[]) {
        StringBuffer s1 = new StringBuffer();
        StringBuffer s2 = new StringBuffer(20);
        StringBuffer s3 = new StringBuffer("StringBuffer");

        System.out.println("s3 = " + s3);
        System.out.println(s2.length()); //chứa 0
        System.out.println(s3.length()); //chứa 12
        System.out.println(s1.capacity()); //chứa 16
        System.out.println(s2.capacity()); //chứa 20
        System.out.println(s3.capacity()); //chứa 28
    }
}
```

`length()` và `capacity()` của đối tượng StringBuffer là hoàn toàn khác nhau. Phương thức `length()` đề cập đến số các ký tự mà đối tượng đưa ra, trong khi `capacity()` trả về tổng dung lượng mặc định của một đối tượng (16), và số các ký tự trong đối tượng StringBuffer.

3. Các phương thức lớp StringBuffer

- public synchronized StringBuffer **append**(String s): được sử dụng để nối thêm các chuỗi được chỉ định với chuỗi này. Các phương thức append() được nạp chồng như append(char), append(boolean), append(int), append(float), append(double), ...
- public synchronized StringBuffer **insert**(int offset, String s): được sử dụng để chèn chuỗi chỉ định với chuỗi này tại vị trí quy định. Các phương thức insert() được nạp chồng như insert(int, char), insert(int, boolean), insert(int, int), insert(int, float), insert(int, double), ...
- public synchronized StringBuffer **replace**(int startIndex, int endIndex, String str): được sử dụng để thay thế chuỗi từ vị trị startIndex đến endIndex bằng chuỗi str.
- public synchronized StringBuffer **delete**(int startIndex, int endIndex): được sử dụng để xóa chuỗi từ vị trí startIndex đến endIndex.
- public synchronized StringBuffer **reverse**(): được sử dụng để đảo ngược chuỗi.
- public int **capacity**(): được sử dụng để trả về dung lượng hiện tại.
- public void **ensureCapacity**(int minimumCapacity): được sử dụng để đảm bảo dung lượng ít nhất bằng mức tối thiểu nhất định.
- public char **charAt**(int index): được sử dụng trả về ký tự tại vị trí quy định.
- public int **length**(): được sử dụng trả về chiều dài của chuỗi nghĩa là tổng số ký tự.
- public String **substring**(int beginIndex): được sử dụng trả về chuỗi con bắt đầu từ vị trí được chỉ định.
- public String **substring**(int beginIndex, int endIndex): được sử dụng trả về chuỗi con với vị trí bắt đầu và vị trí kết thúc được chỉ định.

+ append()

Phương thức này nối thêm một chuỗi hoặc một mảng ký tự tại vị trí cuối cùng của một đối tượng StringBuffer. Ví dụ:

```
StringBuffer s1 = new StringBuffer("Good");
s1.append("evening");
```

Giá trị trong s1 bây giờ là "goodevening".

+ insert()

Phương thức này lấy hai tham số. Tham số đầu tiên là vị trí chèn. Tham số thứ hai có thể là một chuỗi, một ký tự (char), một giá trị nguyên (int), hay một giá trị số thực (float) được chèn vào. Vị trí chèn sẽ lớn hơn hay bằng đến 0, và nhỏ hơn hay bằng chiều dài của đối tượng Stringbuffer. Bất kỳ đối số nào, trừ ký tự hoặc chuỗi, được chuyển vào biểu mẫu chuỗi, và sau đó được chèn vào. Ví dụ:

```
StringBuffer str = new StringBuffer("Java sion");
str.insert(1,'b');
```

Biến "str" chứa chuỗi "Java sion".

+ charAt()

Phương thức này trả về một giá trị ký tự trong đối tượng StringBuffer tại vị trí được chỉ định. Ví dụ:

```
StringBuffer str = new StringBuffer("James Gosling");
char letter = str.charAt(6); //chứa "G"
```

+ setCharAt()

Phương thức này được sử dụng để thay thế ký tự trong một StringBuffer với những cái khác tại một vị trí được chỉ định.

```
StringBuffer name = new StringBuffer("Java");
name.setCharAt(2,'v');
```

Biến "name" chứa "Java".

+ setLength()

Phương thức này thiết lập chiều dài của đối tượng StringBuffer. Nếu chiều dài được chỉ định nhỏ hơn chiều dài nguyên thuỷ của bộ nhớ trung gian, thì các ký tự thừa sẽ bị cắt bớt. Nếu chiều dài chỉ định nhiều hơn chiều dài nguyên thủy của bộ nhớ đệm, các ký tự null được thêm vào tại vị trí cuối cùng của bộ nhớ đệm.

```
StringBuffer str = new StringBuffer(10);
str.setLength(str.legth() +10);
```

+ getChars()

Phương thức này được sử dụng để trích ra các ký tự từ đối tượng StringBuffer, và sao chép chúng vào một mảng. Phương thức getChars() lấy bốn tham số sau:

- Mục bắt đầu: vị trí bắt đầu, từ nơi mà ký tự được lấy vào.
- Mục kết thúc: vị trí kết thúc
- Mảng: Mảng đích, nơi mà các ký tự được sao chép.
- Nơi gởi tới mục bắt đầu: Các ký tự được sao chép trong mảng đích từ vị trí này.

Ví dụ:

```
StringBuffer str = new StringBuffer("Leopard");
char ch[] = new char[10];
str.getChars(3,6,ch,0);
```

Bây giờ biến "ch" chứa "par"

+ reverse()

Phương thức này đảo ngược nội dung của một đối tượng StringBuffer, và trả về một đối tượng StringBuffer. Ví dụ:

```
StringBuffer str = new StringBuffer("devil");
StringBuffer strrev = str.reverse();
```

Biến "strrev" chứa "lived".

+ replace()

Phương thức replace() của lớp StringBuffer thay thế chuỗi bằng chuỗi khác từ vị trị bắt đầu và kết thúc được quy định.

```
public class StringBufferExam3 {
    public static void main(String args[]) {
        StringBuffer sb = new StringBuffer("Hello");
        sb.replace(1, 3, "Java");
        System.out.println(sb); //in HJavalo
    }
}
```

+ delete()

Phương thức replace() của lớp StringBuffer xóa chuỗi từ vị trị bắt đầu và kết thúc được quy định.

```
public class StringBufferExam4 {
    public static void main(String args[]) {
        StringBuffer sb = new StringBuffer("Hello");
        sb.delete(1, 3);
        System.out.println(sb); //in Hlo
    }
}
```

+ capacity()

Phương thức capacity() của lớp StringBuffer trả về dung lượng của bộ nhớ đệm. Dung lượng mặc định của bộ nhớ đệm là 16. Nếu số lượng ký tự của chuỗi tăng lên thì dung lượng được tính theo công thức (dung lượng cũ*2)+2. Ví dụ: Nếu dung lượng hiện tại là 16, nó sẽ tăng lên (16*2)+2=34.

```
public class StringBufferExam6 {
    public static void main(String args[]) {
        StringBuffer sb = new StringBuffer();
        System.out.println(sb.capacity()); //mặc định là 16
        sb.append("Hello");
        System.out.println(sb.capacity()); //đ□n đây v□n là 16
        sb.append("java is my favourite language");
        System.out.println(sb.capacity()); //đ□n đây là (16*2)+2=34 i.e (dung lượng cũ*2)+2
    }
}
```

+ ensureCapacity()

Phương thức ensureCapacity() của lớp StringBuffer đảm bảo rằng dung lượng đã cho là tối thiểu với dung lượng hiện tại. Nếu nó lớn hơn dung lượng hiện tại, dung lượng hiện tại được tăng theo công thức (dung lượng cũ*2)+2. Ví dụ, dung lượng hiện tại là 16, nó sẽ tăng lên là (16*2)+2=34.

```
public class StringBufferExam7 {
    public static void main(String args[]) {
        StringBuffer sb = new StringBuffer();
        System.out.println(sb.capacity()); //mặc định là 16
        sb.append("Hello");
        System.out.println(sb.capacity()); //đ□n đây là 16
        sb.append("java is my favourite language");
```

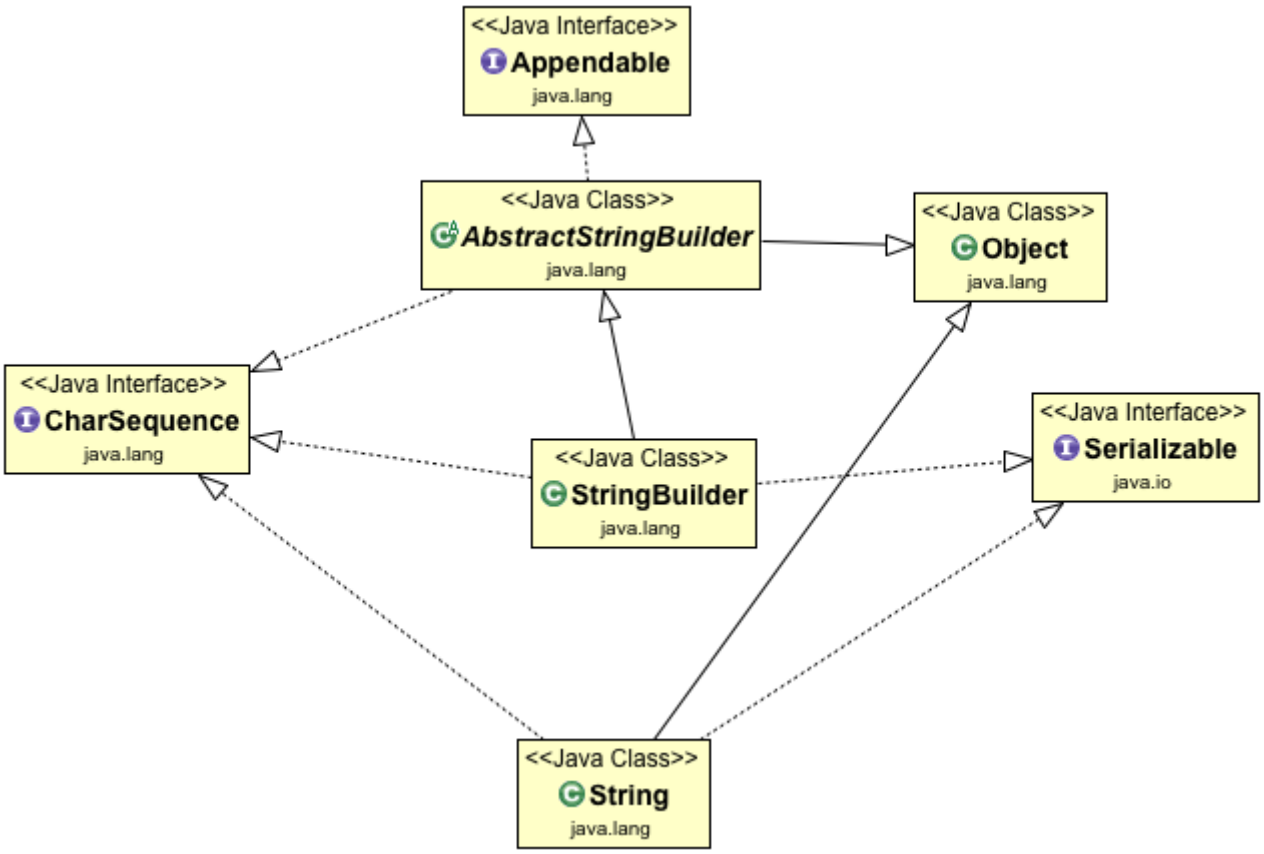
```
        System.out.println(sb.capacity()); //đến đây là (16*2)+2=34 i.e (dung lượng cũ*2)+2
        sb.ensureCapacity(10); //đến đây không có sự thay đổi
        System.out.println(sb.capacity()); //đến đây là 34
        sb.ensureCapacity(50); //đến đây là (34*2)+2
        System.out.println(sb.capacity()); //đến đây là 70
    }
}
```

Java StringBuilder class - Lớp StringBuilder trong Java

1. Khái niệm

Trong java, lớp StringBuilder được sử dụng để tạo chuỗi có thể thay đổi (mutable). Lớp StringBuilder trong java tương tự như lớp StringBuffer ngoại trừ nó có các phương thức không đồng bộ (non-synchronized).

Package: java.lang



2. Khởi tạo đối tượng lớp StringBuilder

- StringBuilder(): Tạo ra một Builder chuỗi với dung lượng ban đầu là 16.
- StringBuilder(String str): Tạo ra một Builder chuỗi với chuỗi cụ thể.
- StringBuilder(int capacity): Tạo ra một Builder chuỗi với dung lượng được chỉ định như độ dài chuỗi.
- StringBuilder(CharSequence cs): Tạo ra một Builder chuỗi với chuỗi cụ thể.

3. Các phương thức của lớp StringBuilder

- public StringBuilder **append**(String s): được sử dụng để nối thêm các chuỗi được chỉ định với chuỗi này. Các phương thức append() được nạp chồng như append(char), append(boolean), append(int), append(float), append(double), ...
- public StringBuilder **insert**(int offset, String s): được sử dụng để chèn chuỗi chỉ định với chuỗi này tại vị trí quy định. Các phương thức insert() được nạp chồng như insert(int, char), insert(int, boolean), insert(int, int), insert(int, float), insert(int, double), ...
- public StringBuilder **replace**(int startIndex, int endIndex, String str): được sử dụng để thay thế chuỗi từ vị trị startIndex đến endIndex bằng chuỗi str.
- public StringBuilder **delete**(int startIndex, int endIndex): được sử dụng để xóa chuỗi từ vị trí startIndex đến endIndex.
- public StringBuilder **reverse**(): được sử dụng để đảo ngược chuỗi.
- public int capacity(): được sử dụng để trả về dung lượng hiện tại.
- public void **ensureCapacity**(int minimumCapacity): được sử dụng để đảm bảo dung lượng ít nhất bằng mức tối thiểu nhất định.
- public char **charAt**(int index): được sử dụng trả về ký tự tại vị trí quy định.
- public int length(): được sử dụng trả về chiều dài của chuỗi nghĩa là tổng số ký tự.
- public String **substring**(int beginIndex): được sử dụng trả về chuỗi con bắt đầu từ vị trí được chỉ định.
- public String **substring**(int beginIndex, int endIndex): được sử dụng trả về chuỗi con với vị trí bắt đầu và vị trí kết thúc được chỉ định.

+ append()

Phương thức append() của lớp StringBuilder nối thêm tham số vào cuối chuỗi.

```
public class StringBuilderExam1 {

    public static void main(String args[]) {
        StringBuilder sb = new StringBuilder("Hello ");
        sb.append("Java");
        System.out.println(sb);
    }
}
```

+ insert()

Phương thức insert() của lớp StringBuilder chèn chuỗi vào chuỗi này từ vị trí quy định.

```
public class StringBuilderExam2 {

    public static void main(String args[]) {
        StringBuilder sb = new StringBuilder("Hello ");
        sb.insert(1, "Java");
        System.out.println(sb);
```

```
    }
}
```

+ replace()

Phương thức replace() của lớp StringBuilder thay thế chuỗi bằng chuỗi khác từ vị trị bắt đầu và kết thúc được quy định.

```
public class StringBuilderExam3 {

    public static void main(String args[]) {
        StringBuilder sb = new StringBuilder("Hello");
        sb.replace(1, 3, "Java");
        System.out.println(sb);
    }
}
```

+ delete()

StringBuilder delete() được sử dụng để xóa chuỗi từ vị trí startIndex đến endIndex

```
StringBuilder sb = new StringBuilder("JournalABC");
sb.delete(7,14);
System.out.println(sb);// prints Journal
```

+ reverse()

Phương thức reverse() được sử dụng để đảo ngược chuỗi.

```
StringBuilder sb = new StringBuilder("lived");
sb.reverse();
System.out.println(sb);// prints devil
```

+ capacity()

Phương thức capacity() của lớp StringBuilder trả về dung lượng của bộ nhớ đệm. Dung lượng mặc định của bộ nhớ đệm là 16. Nếu số lượng ký tự của chuỗi tăng lên thì dung lượng được tính theo công thức (dung lượng cũ*2)+2. Ví dụ: Nếu dung lượng hiện tại là 16, nó sẽ tăng lên (16*2)+2=34.

```
StringBuilder sb=new StringBuilder();
System.out.println(sb.capacity()); // default value 16
sb.append("Java");
System.out.println(sb.capacity()); // still 16
```

```
sb.append("Hello StringBuilder Class!");
System.out.println(sb.capacity()); // (16*2)+2
```

+ ensureCapacity()

Phương thức ensureCapacity() của lớp StringBuilder đảm bảo rằng dung lượng đã cho là tối thiểu với dung lượng hiện tại. Nếu nó lớn hơn dung lượng hiện tại, dung lượng hiện tại được tăng theo công thức (dung lượng cũ*2)+2. Ví dụ, dung lượng hiện tại là 16, nó sẽ tăng lên là (16*2)+2=34.

```
public class StringBuilderExample {

    public static void main(String[] args) {

        StringBuilder sbObj = new StringBuilder();
        System.out.println(sbObj.capacity()); //default 16

        sbObj.append("Java StringBuilder Class!");
        System.out.println(sbObj.capacity()); // capacity 34

        sbObj.ensureCapacity(12); // no change
        System.out.println(sbObj.capacity()); //still 34

        sbObj.ensureCapacity(60); // (34*2)+2 = 70
        System.out.println(sbObj.capacity()); //70
    }
}
```

Java Math class - Lớp java.lang.Math trong Java

Lớp này chứa các phương thức tĩnh để thực hiện các thao tác toán học.

Package: java.lang

Các phương thức của Math class như sau:

abs()

Phương thức này trả về giá trị tuyệt đối của một số. Đối số được truyền đến nó có thể là kiểu int, float, double, hoặc long. Kiểu dữ kiệu byte và short được chuyển thành kiểu int nếu chúng được truyền tới như là một đối số. Ví dụ:

```
int num = -1;
Math.abs(num) //trả về 1.
```

ceil()

Phương thức này tìm thấy số nguyên lớn hơn hoặc bằng đối số được truyền đến ngay tức thời.

floor()

Phương thức này trả về số nguyên nhỏ hơn hoặc bằng đối số được truyền vào ngay tức thời.

```
System.out.println(Math.ceil(8.02)); //trả về 0.9
System.out.println(Math.ceil(-1.3)); //trả về -1.0
System.out.println(Math.ceil(100)); //trả về 100.0
System.out.println(Math.floor(-5.6)); //trả về -6.0
System.out.println(Math.floor(201.1)); //trả về 201
System.out.println(Math.floor(100)); //trả về 100
```

max()

Phương thức này tìm giá trị lớn nhất trong hai giá trị được truyền vào. Các đối số được

truyền vào có thể là kiểu int, long, double, và float.

min()

Phương thức này tìm giá trị nhỏ nhất trong hai giá trị được truyền vào. Các đối số được truyền vào có thể là kiểu int, long, double và float.

round()

Phương thức này làm tròn đối số có dấu phẩy động. Ví dụ, câu lệnh Math.round(34.5) trả về 35.

random()

Phương thức này trả về một số ngẫu nhiên giữa 0.0 và 1.0 của kiểu double.

sqrt()

Phương thức này trả về bình phương của một số. Ví dụ, câu lệnh Math.sqrt(144) trả về 12.0.

sin()

Phương thức này trả về sine của một số, nếu góc được truyền đến bằng radian. Ví dụ: Math.sin(Math.PI/2) trả về 1.0, giá trị của sin 45.

Pi/2 radians = 90 độ. Giá trị của "pi" bắt nguồn từ hằng số được định nghĩa trong lớp "Math.PI".

cos()

Phương thức này trả về cos của một số, nếu góc được truyền đến bằng radian.

tan()

Phương thức này trả về tan của một số, nếu góc được truyền đến bằng radian.

Java Runtime class - Lớp Runtime trong Java

Lớp Runtime được gói gọn trong môi trường Runtime. Lớp này được sử dụng cho việc quản lý bộ nhớ, và việc thực thi của các quá trình xử lý gia tăng. Mỗi chương trình Java có một thể hiện đơn của lớp này, để cho phép ứng dụng giao tiếp với môi trường. Nó không thể được khởi tạo, khi mà một ứng dụng không thể tạo ra một minh dụ của riêng mình thuộc lớp này. Tuy nhiên, chúng ta có thể tạo ra một minh dụ hiện hành trong lúc thực hiện chương trình từ việc dùng phương thức `Runtime().garbage`

Package: *java.lang*

Bây giờ, chúng ta biết rằng việc thu gom các dữ liệu không thích hợp trong Java là một tiến trình tự động, và chạy một cách định kỳ. Để kích hoạt một cách thủ công bộ thu thập dữ liệu không thích hợp, ta gọi phương thức gc() trên minh dụ thời gian thời gian thực hiện hành. Để quyết định chi tiết cấp phát bộ nhớ, sử dụng các phương thức `totalMemory()` và `freeMemory()`.

```
Runtime r = Runtime.getRunTime(); .....
.....
long freemem = r.freeMemory();
long totalmem = r.totalMemory(); r.gc();
```

Bảng sau biểu diễn một vài phương thức được sử dụng chung của lớp này:

STT	Method	Purpose
1	exit(int)	Dừng việc thực thi, và trả về giá trị của đoạn mã đến hệ điều hành. Việc dừng thông thường với giá trị 0; giá trị khác 0 cho biết việc dừng khác thường.
2	freeMemory()	Quyết định số lượng sẵn có của bộ nhớ trống đến hệ thống thời gian chạy của Java trong giới hạn của các byte
3	getRuntime()	Trả về thể hiện thời gian chạy hiện hành.
4	gc()	Gọi những bộ phận thu thập dữ liệu vô nghĩa.

5	totalMemory()	Để quyết định tổng số lượng bộ nhớ sẵn có của chương trình.
6	exec(String)	Thực thi một chương trình phân cách của tên được gọi.

Ví dụ:

```
class RuntimeDemo {
  public static void main(String args[]) {
    Runtime r = Runtime.getRuntime();
    Process p = null;
    try {
      p = r.exec("calc.exe");
    }
    catch(Exception e) {
      System.out.println("Error executing calculator");
    }
  }
}
```

Bạn có thể đạt được minh dụ thời Runtime hiện hành thông qua phương thức Runtime.getRuntime().

Sau đó, bạn có thể tham chiếu đến chương trình thi hành calc.exe, và lưu trữ trong một đối tượng của tiến trình.

Java System class - Lớp System trong Java

Lớp System cung cấp các điều kiện thuận lợi như là, xuất, nhập chuẩn và các luồng lỗi. Nó cũng cung cấp một giá trị trung bình để các thuộc tính truy cập được kết hợp với hệ thống thời gian chạy của Java, và các thuộc tính môi trường khác nhau như là, phiên bản, đường dẫn, hay các dịch vụ... Các trường của lớp này là in, out, và err, các trường này tiêu biểu cho xuất, nhập và lỗi chuẩn tương ứng.

Package: *java.lang*

STT	Method	Desc
1	exit(int)	Dừng việc thực thi, và trả về giá trị của đoạn mã. 0 cho biết có thể thoát ra một cách bình thường.
2	gc()	Thực hiện trình thu gom rác *Garbage Collection*
3	getProperties()	Trả về thuộc tính được kết hợp với hệ thống thời gian chạy của Java.
4	setProperties()	Thiết lập các đặc tính hệ thống hiện hành.
5	currentTimeMillis()	Trả về thời gian hiện tại trong mili giây (ms), được đo lường lúc nửa đêm vào tháng giêng năm 1970.
6	arraycopy(Object, int, Object, int, int)	Sao chép một mảng.

Lớp System không thể khai báo để tạo các đối tượng.

Đoạn mã trong chương trình sau truy lục và hiển thị một vài các thuộc tính môi trường liên quan đến Java.

```
class SystemDemo {
  public static void main(String args[]) {
    System.out.println(System.getProperty("java.class.path"));
    System.out.println(System.getProperty("java.home"));
    System.out.println(System.getProperty("java.class.version"));
```

```
    System.out.println(System.getProperty("java.specification.vendor"));
    System.out.println(System.getProperty("java.specification.version"));
    System.out.println(System.getProperty("java.vendor"));
    System.out.println(System.getProperty("java.vendor.url"));
    System.out.println(System.getProperty("java.version"));
    System.out.println(System.getProperty("java.vm.name"));
  }
}
```

Mỗi thuộc tính mà được yêu cầu để được in, được cung cấp như một tham số chuỗi đến phương thức `System.getProperty()`. Phương thức này lần lượt sẽ trả về thông tin có liên quan đến phương thức `System.out.println()`.

Java Class class - Lớp Class trong Java

Các thể hiện của lớp Class đại diện cho các lớp và giao diện (interface) trong một ứng dụng Java đang chạy. Điều này cho phép chúng ta truy cập thông tin về đối tượng trong suốt thời gian chạy.

Package: *java.lang*

Chúng ta có thể lấy một đối tượng của lớp này, hoặc một minh dụ bằng một trong ba cách sau:

Sử dụng phương thức getChar() trong một đối tượng.

```
interface A {
  final int id = 1;
  final String name = "Diana";
}
class B implements A {
  int deptno;
}
class ClassDemo {
  public static void main(String args[]) {
    A a = new B();
    B b = new B();
    Class x;
    x = a.getClass();
    System.out.println("a is object of type: " + x.getName());
    x = b.getClass();
    System.out.println("b is object of type: " + x.getName());
    x = x.getSuperclass();
    System.out.println(x.getName() + " is the superclass of b.");
  }
}
```

Java Object class - Lớp Object trong Java

Lớp Object là một lớp cha của tất cả các lớp. Dù là một lớp do người dùng định nghĩa không mở rộng bất kỳ một lớp nào khác, theo mặc định nó mở rộng lớp đối tượng.

Package: java.lang

STT	Method	Desc
	equals(Object)	So sánh thể hiện đối tượng hiện tại với đối tượng đã cho, và kiểm tra nếu chúng bằng nhau.
	finalize()	Mặc định hình thức của phương thức cuối cùng. Thông thường bị phủ bởi lớp con.
	notify()	Thông báo dòng (thread) mà hiện thời trong trạng thái đang chờ trên màn hình của đối tượng này.
	notifyAll()	Thông báo tất cả các dòng (thread) hiện hành trong trạng thái chờ trên màn hình của đối tượng này.
	toString()	Trả về một chuỗi đại diện cho đối tượng.
	wait()	Tạo ra dòng hiện hành để nhập vào trạng thái đang chờ.

Trong chương trình sau, chúng ta không khai báo bất kỳ lớp hoặc gói nào. Bây giờ, chúng ta có thể tạo bằng cách sử dụng phương thức equals(). Bởi vì, theo mặc định lớp ObjectDemo mở rộng lớp Object.

```
class ObjectDemo {
  public static void main(String args[]) {
    if (args[0].equals(“Aptech”));
    System.out.println(“Yes, Aptech is the right choice ! ”);
  }
}
```

Java Hashtable class - Lớp Hashtable trong Java

Lớp Hashtable mở rộng lớp trừu tượng Dictionary, lớp này cũng được định nghĩa trong gói java.util. Hashtable được sử dụng để ánh xạ các khoá đến các giá trị. Ví dụ, nó có thể được sử dụng để ánh xạ các tên đến tuổi, những người lập trình đến những dự án, các tiêu đề công việc đến các lương, và cứ tiếp tục như vậy.

Packge: *java.util*

Hashtable mở rộng kích thước khi các phần tử được thêm vào. Khi đó việc tạo một bảng băm mới, bạn có thể chỉ định một dung lượng ban đầu và các yếu tố nạp vào. Điều này sẽ làm cho hashtable tăng kích thước lên, bất cứ lúc nào việc thêm vào một phần tử mới sẽ làm thay đổi giới hạn của hashtable cũ. Giới hạn của bảng băm là dung lượng được nhân lên bởi các yếu tố được nạp vào.

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

Các phương thức xây dựng cho bảng băm được biểu diễn trong bảng sau:

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

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 .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 được tóm tắt trong bảng sau:

STT	Method	Desc
1	clear()	Xoá tất cả các phần tử từ bảng băm.
2	clone()	Tạo một bảng sao của Hashtable.
3	contains(Object)	Trả về True nếu bảng băm chứa các đối tượng được chỉ định.
4	containsKey(Object)	Trả về True nếu bảng băm chứa khoá được chỉ định.
5	elements()	Trả về một bảng liệt kê các yếu tố trong bảng băm.
6	get(Object key)	Truy xuất phần tử với khoá được chỉ định.
7	isEmpty()	Trả về true nếu bảng băm trống.
8	keys()	Trả về một bảng liệt kê các khoá trong bảng băm.
9	put(Object, Object)	Thêm một phần tử mới vào bảng băm bằng cách sử dụng khoá và giá trị được chỉ định.
10	rehash()	Thay đổi bảng băm thành một bảng băm lớn hơn.
11	remove(Object key)	Xoá một đối tượng được cho bởi khoá được chỉ định.
12	size()	Trả về số phần tử trong bảng băm.
13	toString()	Trả về đại diện chuỗi được định dạng cho bảng băm.

Chương trình sau sử dụng lớp Hashtable. Trong chương trình này, tên của các tập ảnh là các khoá, và các năm là các phần tử

- `contains` được sử dụng để tra cứu phần tử nguyên 1969, để thấy có danh sách chứa bất kỳ các tập ảnh từ 1969.
- `containsKey` được sử dụng để tìm kiếm cho khoá "Animals", để nhìn thấy nếu tập ảnh đó tạo nên danh sách.
- `get()` được sử dụng để truy lục tập ảnh "Wish You Were Here" có trong bảng băm không. Phương thức get() trả về phần tử kết hợp với khoá, cả hai tên và năm được hiển thị tại điểm này.

```
import java.util. * ;
public class HashTableImplementer {
  public static void main(String args[]) {
    //tạo một b□ng băm mới
```

```
    Hashtable ht = new Hashtable();
    //thêm các tập nh t t nh t c a Pink Floyd
    ht.put("Pulse", new Integer(1995));
    ht.put("Dark Side of the Moon", new Integer(1973));
    ht.put("Wish You Were Here", new Integer(1975));
    ht.put("Animals", new Integer(1997));
    ht.put("Ummagumma", new Integer(1969));
    //Hi n thị b ng băm
    System.out.println("Initailly: " + ht.toString());
    //ki m tra cho b t kỳ tập nh nào từ 1969
    if (ht.contains(new Integer(1969)))
      System.out.println("An album from 1969 exists");
    //ki m tra cho tập nh các con thú
    if (ht.containsKey("Animals"));
    System.out.println("Animals was found");
    //Tìm ra
    Integer year = (Integer) ht.get("Wish You Were Here");
    System.out.println("Wish you Were Here was released in " + year.toString());
    //Xoá một tập nh
    System.out.println("Removing Ummagumma\r\n");
    ht.remove(“Ummagumma”);
    //Di chuy n thông qua một b ng liệt kê c a t t c các khoá trong b ng.
    System.out.println("Remaining: \r\n");
    for (Enumeration enum = ht.keys(); enum.hasMoreElements();)
      System.out.println((String) enum.nextElement());
  }
}
```

Java Random class - Lớp Random trong Java

Lớp này đại diện một bộ tạo số ngẫu nhiên (pseudo-random).

Package: *java.util*

STT	Method	Desc
1	random()	Tạo ra một bộ tạo số ngẫu nhiên mới
2	random(long)	Tạo ra một bộ tạo số ngẫu nhiên mới dựa trên giá trị khởi tạo được chỉ định.
3	nextDouble()	Trả về một giá trị kiểu double kế tiếp giữa 0.0D đến 1.0D từ bộ tạo số ngẫu nhiên.
4	nextFloat()	Trả về một giá trị kiểu float kế tiếp giữa 0.0F và 1.0F từ bộ tạo số ngẫu nhiên.
5	nextGaussian()	Trả về kiểu double được phân phối Gaussian kế tiếp từ bộ tạo số ngẫu nhiên. Tạo ra các giá trị Gaussian sẽ có một giá trị trung bình của 0, và một độ lệch tiêu chuẩn của 1.0.
6	nextInt()	Trả về giá trị kiểu Integer kế tiếp từ một bộ tạo số ngẫu nhiên.
7	nextLong()	Trả về giá trị kiểu long kế tiếp từ một bộ tạo số ngẫu nhiên.
8	setSeed(long)	Thiết lập giá trị khởi tạo từ bộ tạo số ngẫu nhiên.

Java Vector class - Lớp Vector trong Java

Một trong các vấn đề với một mảng là chúng ta phải biết nó lớn như thế nào khi chúng ta tạo nó. Nó thì không thể xác định kích thước của mảng trước khi tạo nó.

Lớp Vector của Java giải quyết vấn đề này. Nó cung cấp một dạng mảng với kích thước ban đầu, mảng này có thể tăng thêm khi nhiều phần tử được thêm vào. Một lớp Vector lưu trữ các item của kiểu Object, nó có thể dùng để lưu trữ các thể hiện của bất kỳ lớp Java nào. Một lớp Vector đơn lẻ có thể lưu trữ các phần tử khác nhau, các phần tử khác nhau này là thể hiện của các lớp khác nhau.

Package: *java.util*

Tại bất kỳ thời điểm, một lớp Vector có dung lượng để lưu trữ một số nào đó của các phần tử. Khi một lớp Vector biết về dung lượng của nó, thì dung lượng của nó được gia tăng bởi một số lượng riêng cho Vector đó. Lớp Vector cung cấp ba phương thức xây dựng khác nhau mà có thể chúng ta chỉ định dung lượng khởi tạo, và tăng số lượng của một Vector, khi nó được tạo ra.

Các phương thức khởi tạo của lớp Vector được tóm tắt trong bảng sau:

STT	Constructor	Desc
1	Vector(int)	Tạo ra một lớp Vector mới với dung lượng ban đầu được chỉ định.
2	Vector(int, int)	Tạo ra một lớp Vector mới với dung lượng ban đầu được chỉ định, và tăng số lượng.
3	Vector()	Tạo ra một lớp Vector mới với dung lượng khởi tạo mặc định, và tăng số lượng.

Một mục (item) được thêm vào một lớp Vector bằng cách sử dụng hàm addElement(). Tương tự, một phần tử có thể được thay thế bằng cách sử dụng hàm setElementAt(). Một lớp Vector có thể tìm kiếm bằng cách sử dụng phương thức contains(), phương thức này trông có vẻ dễ dàng cho một lần xuất hiện của một đối tượng (Object). Phương thức elements() thì hữu dụng bởi vì nó trả về một bảng liệt kê của các đối tượng được lưu trữ trong lớp Vector.

Các phương thức này và các phương thức thành viên khác của lớp Vector được tóm tắt trong bảng dưới đây:

STT	Method	Desc
1	addElement(Object)	Chèn các phần tử được chỉ định vào lớp Vector.
2	capacity()	Trả về số phần tử mà sẽ vừa đủ cho phần được cấp phát hiện thời của lớp Vector.
3	clone()	Tạo bản sao của vector, nhưng không phải là các phần tử của nó.
4	contains(Object)	Trả về True nếu lớp Vector chứa đối tượng được chỉ định.
5	copyInto(Object [])	Sao chép các phần tử của lớp Vector vào mảng được chỉ định.
6	elementAt(int)	Truy lục phần tử được cấp phát tại chỉ mục được chỉ định.
7	elements()	Trả về một bảng liệt kê của các phần tử trong lớp Vector.
8	ensureCapacity(int)	Chắc chắn rằng lớp Vector có thể lưu trữ ít nhất dunglượng tối thiểu được chỉ định.
9	firstElement()	Trả về phần tử đầu tiên trong lớp Vector.
10	indexOf(Object)	Tìm kiếm lớp Vector, và trả về chỉ mục zero cơ bản cho khớp với đối tượng đầu tiên.
11	indexOf(Object, int)	Tìm kiếm lớp Vector đang bắt đầu tại số chỉ mục được chỉ định, và trả về chỉ mục zero cơ bản cho khớp với đối tượng kế tiếp.
12	insertElementAt(Object, int)	Thêm các đối tượng được chỉ định tại chỉ mục được chỉ định.
13	isEmpty()	Trả về True nếu lớp Vector không có phần tử.
14	lastElement()	Trả về phần tử cuối cùng trong lớp Vector.
15	lastIndexOf(Object)	Tìm kiếm lóp Vector, và trả về chỉ mục zero cơ bản cho khớp với đối tượng cuối cùng.
16	lastIndexOf(Object, int)	Tìm kiếm lớp Vector đang bắt đầu tại số chỉ mục được chỉ định, và trả về chỉ mục zero cơ bản cho khớp với đối tượng trước.
17	removeAllElements()	Xoá tất cả các phần tử từ lớp Vector.

18	removeElement(Object)	Xoá đối tượng được chỉ định từ lớp Vector.
19	removeElementAt(int)	Xoá đối tượng tại chỉ mục được chỉ định.
20	setElementAt(Object, int)	Thay thế đối tượng tại chỉ mục được chỉ định với đối tượng được chỉ định.
21	setSize(int)	Thiết lập kích thước của lớp Vector thành kích thước mới được chỉ định.
22	setSize(int)	Thiết lập kích thước của lớp Vector thành kích thước mới được chỉ định.
23	size()	Trả về số của các phần tử hiện thời trong lớp Vector.
24	toString()	Trả về một đại diện chuỗi được định dạng nội dung của lớp Vector.
25	trimToSize()	Định lại kích thước của lớp Vector để di chuyển dung lượng thừa trong nó.

Chương trình sau tạo ra một lớp Vector "vect". Nó chứa 6 phần tử: "Numbers In Words", "One", "Two", "Three", "Four", "Five". Phương thức removeElement() được sử dụng để xoá các phần tử từ "vect".

```
import java.util. * ;
public class VectorImplementation {
  public static void main(String args[]) {
    Vector vect = new Vector();
    vect.addElement("One");
    vect.addElement("Two");
    vect.addElement("Three");
    vect.addElement("Four");
    vect.addElement("Five");
    vect.insertElementAt("Numbers In Words", 0);
    vect.insertElementAt("Four", 4);
    System.out.println("Size: " + vect.size());
    System.out.println("Vector");
    for (int i = 0; i < vect.size(); i++) {
      System.out.println(vect.elementAt(i) + ", ");
    }
    vect.removeElement("Five");
    System.out.println("");
    System.out.println("Size: " + vect.size());
    System.out.println("Vector");
```

```
    for (int i = 0; i < vect.size(); i++) {
      System.out.print(vect.elementAt(i) + ", ");
    }
  }
}
```

Java StringTokenizer class - Lớp StringTokenizer trong Java

Lớp `StringTokenizer` được sử dụng để tách một chuỗi thành các phân tử token của nó.

Package: *java.util*

Ví dụ: Mỗi từ trong một câu có thể coi như là một token.

Lớp StringTokenizer có thể chỉ định một bộ dấu phân tách token. Dấu phân cách (khoảng trắng) là ký tự phân tách mặc định, tuy nhiên, chúng ta có thể sử dụng tập các toán tử toán học (+, *, /, và -) trong khi phân tách một biểu thức. Các ký tự phân tách có thể chỉ định khi một đối tượng StringTokenizer mới được khởi tạo.

Các phương thức khởi tạo

#	Phương thức	Mục đích
1	StringTokenizer(String)	Tạo ra một lớp StringTokenizer mới dựa trên chuỗi chỉ định được thông báo.
2	StringTokenizer	Tạo ra một lớp StringTokenizer mới dựa trên (String, String) chuỗi chỉ định được thông báo, và một tập các dấu phân cách.
3	StringTokenizer(String, String, Boolean)	Tạo ra một lớp StringTokenizer dựa trên chuỗi chỉ định được thông báo, một tập các dấu phân cách, và một cờ hiệu cho biết nếu các dấu phân cách sẽ được trả về như các token.

Các phương thức của lớp StringTokenizer

#	Phương thức	Mục đích
1	countTokens()	Trả về số các token còn lại.

2	hasMoreElements()	Trả về True nếu nhiều phần tử đang được đánh dấu trong chuỗi. Nó thì giống hệt như hasMoreTokens.
3	hasMoreTokens()	Trả về True nếu nhiều tokens đang được đánh dấu trong chuỗi. Nó thì giống hệt như hasMoreElements.
4	nextElement()	Trả về phần tử kế tiếp trong chuỗi. Nó thì giống như nextToken.
5	nextToken()	Trả về Token kế tiếp trong chuỗi. Nó thì giống như nextElement.
6	nextToken()	Thay đổi bộ dấu phân cách đến chuỗi được chỉ định, và sau đó trả về token kế tiếp trong chuỗi.

Ví dụ:

```
import java.util. * ;
public class StringTokenizerImplementer {
  public static void main(String args[]) {
    // đặt một biểu thức toán học trong một chuỗi và tạo một tokenizer cho chuỗi đó. String
mathExpr = “4*3+2/4”;
    StringTokenizer st1 = new StringTokenizer(mathExpr, ” * +/-“, true);
/ / trong khi cócác token trái, hãy hiển thịmỗi token System.out.println(“Tokens f mathExpr: “);
    while (st1.hasMoreTokens()) System.out.println(st1.nextToken());
    //tạo một chuỗi của các trường được phân cách bởi dấu phẩy và tạo một tokenizer cho chuỗi.
    String commas = “field1, field2, field3, and field4”;
    StringTokenizer st2 = new StringTokenizer(commas, ”, ”, false); //trong khi có các token
trái, hãy hiển thị mỗi token. System.out.println(“Comma-delimited tokens : “);
    while (st2.hasMoreTokens()) System.out.println(st2.nextToken());
  }
}
```