

Lesson 08

Java String Libraries



Java String

❖ Phân loại String.

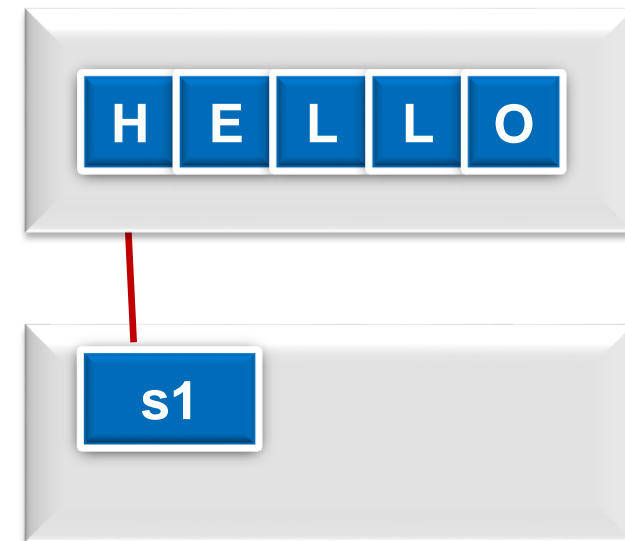
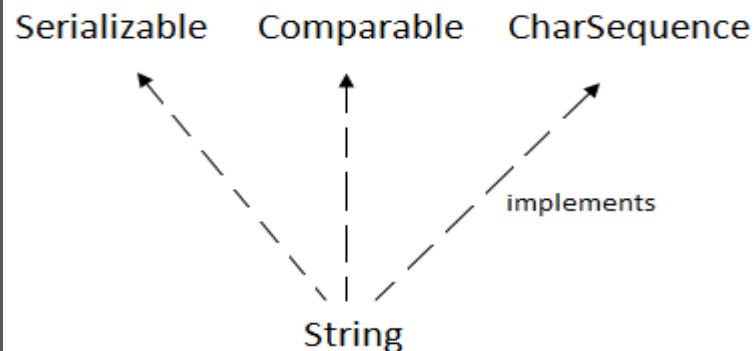
- Immutable
 - String literal
 - String object
- Mutable
 - StringBuilder
 - StringBuffer

String literal vs object

❖ Khai báo chuỗi

- String: là chuỗi, tập các ký tự
- Trong Java, String là lớp quản lý dữ liệu văn bản và là class với các phương thức đi kèm
- Khai báo: Có 2 cách

```
String s1 = new String("Hello");  
String s2 = "Hello";
```

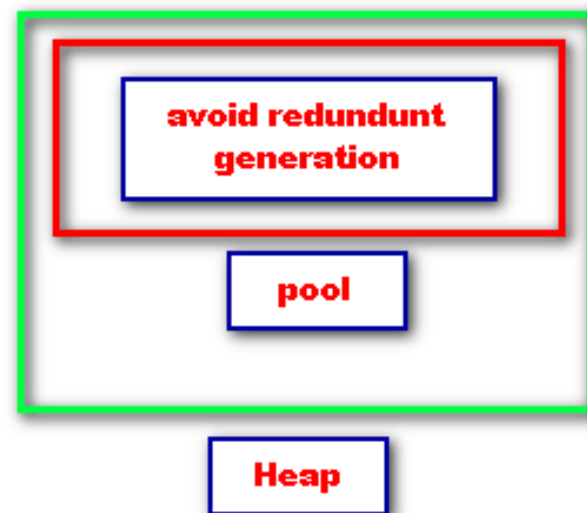


String literal vs object

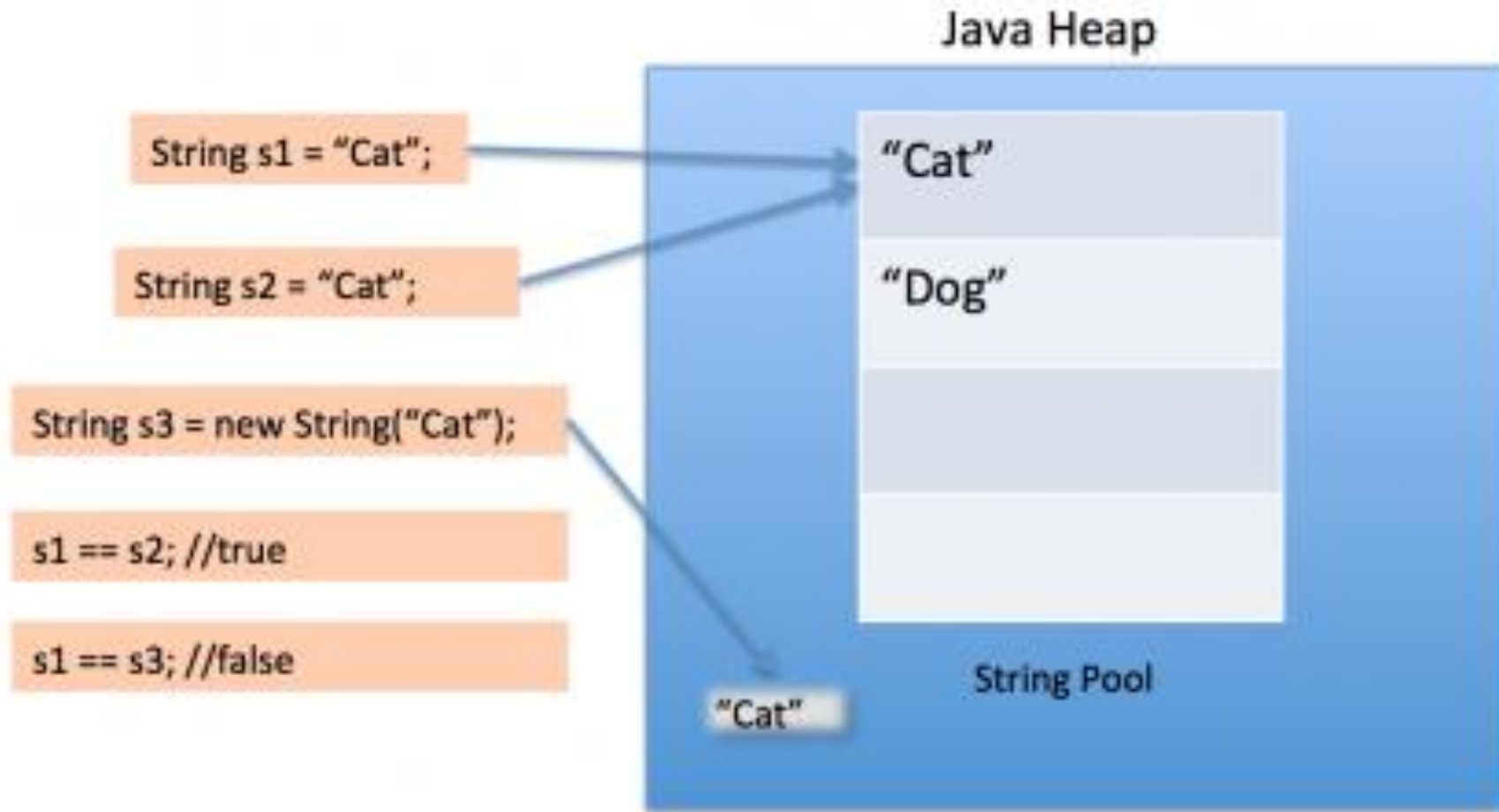
❖ Khai báo chuỗi

```
String s = "text";  
String x = new String("text");
```

```
/**  
 * Giống nhau:  
 *   +> Cả 2 đều là tham chiếu  
 *   +> Thuộc kiểu dữ liệu đối tượng là String  
 * Khác nhau:  
 *   +> new String("text"):  
 *       . Luôn luôn tạo [new] mới một tham chiếu, địa chỉ  
 *       . Không quan tâm giá trị đã tồn tại hay chưa  
 *       . Lưu giá trị tại Heap  
 *   +> s = "text"  
 *       . Tăng khả năng reuse các thể hiện [instance] từ vùng nhớ String constant pool  
 *       . Tiết kiệm bộ nhớ  
 * **/
```




What is a constant pool





String constant pool

- ❖ String Pool is a pool of Strings stored in Heap Memory. Because String is **immutable** in Java and it's implementation of String **interning concept**
- ❖ String pool helps in **saving a lot of space** for Java Runtime although it **takes more time to create** the String.

- ❖ When we use **double quotes – string literal** to create a String, it first looks for String with same value in the String pool, if found it just returns the reference else it creates a new String in the pool and then returns the reference.
- ❖ However using **new operator – string object**, we force String class to create a new String object in heap space. We can use **intern()** method to put it into the pool or refer to other String objects from string pool having same value.



Object or Literal

```
203     ....public void test() {  
204         ....final String l1 = "cat";  
205         ....final String l2 = "dog";  
206  
207         ....final String o1 = new String("tiger");  
208         ....final String o2 = new String("mouse");  
...
```

'new String("mouse")' is redundant [less...](#) (Ctrl+F1)

Reports any attempt to instantiate a new **String** object by copying an existing string. Constructing new **String** objects in this way is rarely necessary, and may cause performance problems if done often enough.

```
212     ....}
```




So sánh chuỗi

❖ Tham chiếu (So sánh địa chỉ - Ít khi được sử dụng)

- Sử dụng toán tử ==

❖ Nội dung

- CompareTo
- Equals

So sánh chuỗi

❖ Toán tử ==

- Dùng để so sánh giá trị của
 - Kiểu dữ liệu nguyên thủy
 - Địa chỉ của đối tượng
- Kiểm tra 2 tham chiếu có cùng trỏ đến một ô nhớ hay không

❖ Phương thức equals

- Dùng để so sánh giá trị của 2 đối tượng
- Phương thức equals() chỉ áp dụng cho kiểu đối tượng, không áp dụng cho kiểu nguyên thủy.

❖ Phương thức compareTo()

- Trả về giá trị kiểu int
 - = 0
 - != 0

Ví dụ

1. Cùng giá trị → true

```
new String("test").equals("test")
```

2. Khác object, instance, reference → false

```
new String("test") == "test"
```

3. Khác reference → false

```
new String("test") == new String("test")
```

4. Cùng tham chiếu – reference → true

```
"test" == "test" : String constant pool
```

5. Test

```
Objects.equals("test", new String("test")) // --> true
```

```
Objects.equals(null, "test") // --> false
```



So sánh chuỗi

❖ Ex01

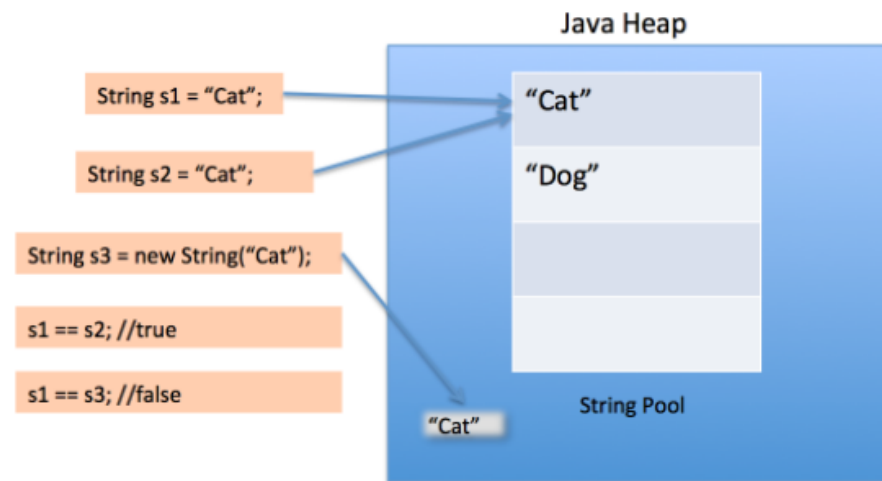
```
// Original String => String pool
String s1 = "bkit-java";
/**
    Vấn đề = và += : đang tạo 1 thể hiện mới
    có dạng new String ("literal") => tạo 1 địa chỉ mới
**/
String test = "bkit";
System.out.println("1: " + ((test+="-java")==s1)); // false
```

❖ Ex02

```
// Original String => String pool
String s1 = "bkit-java"; // POOL
/**
    Vấn đề = và += : đang tạo 1 thể hiện mới
    có dạng new String ("literal") => tạo 1 địa chỉ mới
**/
String test = "bkit";
test += "-java"; // <=> test = test + java <=> test = new String(test+java) // HEAP
// Có thể same hashCode same address memory nhưng khác vùng nhớ => FALSE
System.out.println("1: " + (test==s1)); // false
```

❖ Ex03

```
System.out.println("1: " + (test.intern()==s1.intern())); // true
// Đưa về String pool để so sánh
```





Libraries

❖ Các phương thức xử lý trên kiểu String

Vấn đề:

Cần có các giá trị để phục vụ cho việc hiển thị và tính toán

Ví dụ:

- Tính chiều dài của chuỗi s
- Nối chuỗi s1 vào chuỗi s
- Lấy một ký tự tại vị trí index(3) trong chuỗi s
- Duyệt từng phần tử trong chuỗi
- Tìm vị trí – chỉ số xuất hiện đầu tiên, cuối cùng của ký tự “a” trong chuỗi s

Giải quyết:

Sử dụng hàm chuỗi trong thư viện hàm của Java



Libraries

❖ Các phương thức xử lý trên kiểu String

Vấn đề:

Cần có các giá trị để phục vụ cho việc hiển thị và tính toán

Ví dụ:

- Kiểm tra chuỗi s1 có phải là chuỗi bắt đầu || kết thúc trong chuỗi s không.
- Thay thế chuỗi s1 bằng chuỗi s2 trong chuỗi s
- Loại bỏ các khoảng trắng thừa của chuỗi s3
- Tạo chuỗi con của chuỗi s bắt đầu từ vị trí số 2
- Xác định chuỗi s2 có tồn tại trong chuỗi s1 hay không

Giải quyết:

Sử dụng hàm chuỗi trong thư viện hàm của Java



String with split method

- ❖ Có 2 instances của split() method trong java string
 - String split(String regex)
 - String split(String regex, int limit)
 - Regex: biểu thức chính quy được áp dụng trong string
 - **PatternSyntaxException** nếu pattern không tồn tại
 - Limit: giới hạn số lượng chuỗi sau khi split

```
String s1 = "xin chào các bạn";  
  
System.out.println("-----");  
for (String w : s1.split(" ", 3))  
    System.out.println(w);  
}
```

String parsing

❖ Parsing string with split

- Chia một string vào trong **tokens** dựa trên các **delimiters** đã cho
- Tokens: one piece of information, a "word"
- Delimiters: one (or more) characters used to separate tokens
- Example:

```
String phrase = "Java is one of the best languages";  
String delims = "[ ]+";  
String [] tokens = phrase.split(delims);  
  
/* "[delim_characters]+" */
```




Xử lý chuỗi split.

❖ Convert từ String thành Stream với Pattern split

```
String phrase = "Join Beta, Anne Hora, Steffen Baka, Join Son";  
String delims = ",";  
String [] tokens = Pattern.compile(delims).split(phrase);
```

Ques 01. List the name has first name is Join.

```
String result = Pattern.compile(", ")  
    .splitAsStream(phrase)  
    .filter(obj -> obj.contains("Join"))  
    .collect(Collectors.joining("\n"))
```

StringTokenizer

❖ Cho phép chương trình break a string vào trong tokens

```
StringTokenizer st = new StringTokenizer(charSq, delims);  
while (st.hasMoreTokens()) {  
    System.out.println(st.nextToken());  
}
```

```
String[] result = "this is a test".split("\\s");  
for (int x=0; x<result.length; x++){  
    System.out.println(result[x]);  
}
```

String Mutable

❖ Khi làm việc với văn bản Java cung cấp 3 classes:

- String (Object & Literal): Immutable
 - StringBuffer: Mutable multi-thread
 - StringBuilder: Mutable single-thread
-
- **Immutable String:** cannot change the original string value in heap
 - **Mutable String:** could change the original string value in heap

StringBuilder	StringBuffer
Single-Thread, no synchronized, Unsafe	Multi-Thread, safe synchronized, thread safe
	Avoidable conflict threads
Single-Thread => faster	Faster than String
Mutable: chuỗi có thể thay đổi giá trị ở HEAP	



Thread - Single

```
public class Ex01 {  
    public static void main(String[] args) {  
        // Immutable: String(Literal, Object)  
        String i = "Box"; // constant pool - heap  
        JvmUtils.hash("i1", i);  
  
        i = i.concat(" Layout");  
        JvmUtils.hash("i2", i);  
  
        System.out.println("Value i: " + i);  
  
        // Mutable: StringBuilder, StringBuffer  
        StringBuffer m = new StringBuffer("Singleton") ; // normal - heap  
        JvmUtils.hash("m1", m);  
        m.append(" Pattern")  
            .append(" JAVA")  
            .reverse();  
        JvmUtils.hash("m2", m);  
  
        System.out.println("Value m: " + m);  
    }  
}
```



Thread - Mutiple

```
public class Ex02 {  
    public static void main(String[] args) {  
        // Thread  
        // default: single thread - thread main  
        System.out.println("Ex02#main - start");  
  
        System.out.println(Thread.currentThread().getName() + " is running ...");  
  
        Thread t0 = new Thread(new Task(), "thread-0");  
        t0.start();  
  
        try {  
            TimeUnit.SECONDS.sleep(2);  
        } catch (InterruptedException e) {  
            e.printStackTrace();  
        }  
  
        System.out.println("Ex02#main - end");  
    }  
  
    // thread execute task - runnable  
    public static class Task implements Runnable {  
        @Override  
        public void run() {  
            System.out.println(Thread.currentThread().getName() + " is running ...");  
            System.out.println("Task#run ...");  
        }  
    }  
}
```



Thread – Performance

```
private static class Task implements Runnable {
    private long time;
    private TimeUnit unit;

    public Task(long time, TimeUnit unit) {
        this.time = time;
        this.unit = unit;
    }

    @Override
    public void run() {
        System.out.println(Thread.currentThread().getName() + " is running ...");
        doTask(time, unit);
        System.out.println(Thread.currentThread().getName() + " tooks " + (System.currentTimeMillis
    }

    // virtual task with a certain time
    public static void doTask(long time, TimeUnit unit) {
        try {
            unit.sleep(time);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
}
```

```
public class Ex03 {
    // One Task: Take 3 seconds
    // Demo performance case with 3 calculation tasks
    // Run single thread : Took 9 seconds
    // Run parallel with 3 threads : Took 3 seconds

    private static long start = 0;

    public static void main(String[] args) {
        Task task1 = new Task(3, SECONDS);
        Task task2 = new Task(4, SECONDS);
        Task task3 = new Task(3, SECONDS);

        start = System.currentTimeMillis();

        Thread t1 = new Thread(task1, "thread1");
        t1.start();

        Thread t2 = new Thread(task2, "thread2");
        t2.start();

        Thread t3 = new Thread(task3, "thread3");
        t3.start();
    }
}
```



StringBuilder vs StringBuffer

```
public static class MutableTask implements Runnable {
    // multiple threads access this task with share data
    // share data: StringBulder, StringBuffer

    // private StringBuffer mutable;
    private StringBuilder mutable;
    private int count = 50000;

    public MutableTask() {
        // mutable = new StringBuffer();
        mutable = new StringBuilder();
    }

    @Override
    public void run() {
        System.out.println(Thread.currentThread().getName() + " is running ...");
        for (int i = 1; i <= count; i++) {
            mutable.append("a");
        }
    }

    public int length() {
        return mutable.length();
    }
}
```

```
public class Ex04 {
    public static void main(String[] args) throws InterruptedException {
        // Mutable: StringBuilder - StringBuffer
        MutableTask task = new MutableTask();

        Thread tA = new Thread(task, "Thread-A");
        tA.start();

        Thread tB = new Thread(task, "Thread-B");
        tB.start();

        // make sure threadA finish then go on with thread main
        tA.join();
        tB.join();

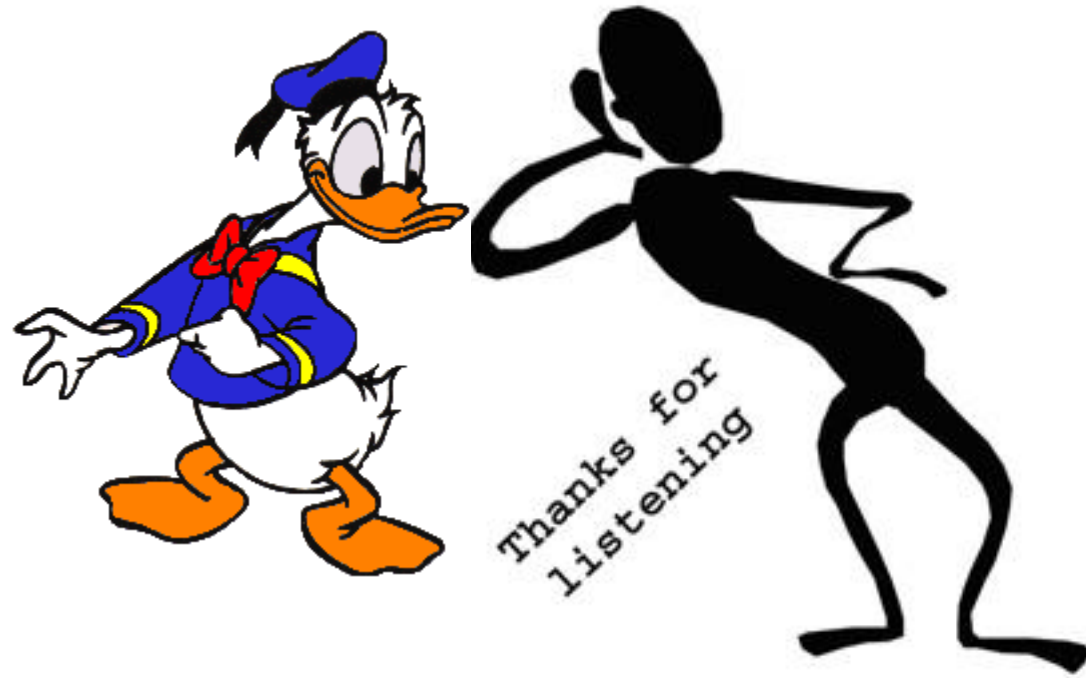
        System.out.println("length: " + task.length());
    }
}
```



String vs StringBuffer, StringBuider

	String	StringBuffer	StringBuilder
Storage Area	Constant String Pool	Heap	Heap
Modifiable	No (immutable)	Yes (mutable)	Yest (Mutable)
Thread Safe	Yes	Yes	No
Performance	Fast	Very slow	Fast

- ‡ Khi nào nên sử dụng String, StringBuffer, StringBuilder.
- ❖ Các kiểu đối tượng **StringBuffer** và **StringBuilder** được sử dụng khi cần thao tác nhiều với các chuỗi ký tự. Không giống như **Strings**, các đối tượng kiểu **StringBuffer** và **StringBuilder** có thể thay đổi giá trị nhiều lần mà không tạo ra nhiều đối tượng không sử dụng.
- ❖ Kiểu **StringBuilder** được giới thiệu từ Java 5 và điểm khác biệt lớn nhất giữa **StringBuilder** và **StringBuffer** là các phương thức của **StringBuilder** không phải là *thread safe*.
- ❖ Nếu đặc tính *thread safe* là cần thiết thì **StringBuffer** là lựa chọn tốt nhất. Trong các trường hợp khác, chúng ta nên sử dụng **StringBuilder** để cho tốc độ xử lý nhanh hơn.



END