java.lang.String class:

String immutability (String vs StringBuffer):

Case 1:

```
String s = new String("Ahmed");
s.concat(" Elhilali");
System.out.println(s); => Ahmed
```

- Once we create a `String` object, we can't perfom any change on the existing object. If we are trying to perform any change, with those changes a new object will be created. This non-changeable behaviour is nothing but ***immutability*** of `String` object.

```
StringBuffer s = new StringBuffer("Ahmed");
s.concat(" Elhilali");
System.out.println(s); => Ahmed Elhilali
```

- Once we create a `StringBuffer` object, we can perfom any change on the existing object. This changeable behaviour is nothing but ***mutability*** of `StringBuffer` object.

Case 2:

```
String s1 = new String("Ahmed");
String s2 = new String("Ahmed");
System.out.println(s1 == s2);        => false
System.out.println(s1.equals(s2));   => true
```

- In `String` class `equals()` method is overriden for content comparison, hence even though objects are different if content is the same `equals()` return `true`.

```
String s1 = new String("Ahmed");
String s2 = new String("Ahmed");
System.out.println(s1 == s2);        => false
System.out.println(s1.equals(s2));   => false
```

- In `StringBuffer` class `equals()` method is not overriden for content comparison, hence `Object` class `equals()` method got excecuted which is meant for ***reference*** comparison (*address* comparison). Due to this, if objects are different `equals()` method return false even though content is the same.

Case 3:

```
String s = new String("Ahmed");
```

- In this case, two objects will be created, one in the *heap* area and the other in the **string constant pool** (SCP), and `s` is always pointing to *object*.

Heap	SCP
s -> Ahmed	Ahmed

```
String s = "Ahmed";
```

- In this case, only one object will be created in *SCP*, and `s` is always pointing to the object.

Heap	SCP
	s -> Ahmed

Note:

- Object creation in *SCP* is optional, first it will check is there any object already present in *SCP* with the required content, if the object is already present, then the existing object will be reused. If the object is not already available then only the new object will be created.
- This rule is only applicable for *SCP* but not for the *heap*.
- *Garbage collector* (*GC*) is not allowed to access *SCP* area. Hence even though object doesn't contain *reference variable* it's not eligible for *GC* if it is present in *SCP* area.
- All *SCP* objects will be destoyed automaticaly at the time of *jvm* shutdown.

Example 1:

```
String s1 = new String("Ahmed");
String s2 = new String("Ahmed");
String s3 = "Ahmed";
String s4 = "Ahmed";
```

Heap	SCP
s1 -> Ahmed	s3, s4 -> Ahmed
s2 -> Ahmed	

Note:

- Whenever we are using a `new` operator an new object will created in the *heap* area. Hence there many be a chance of existing two objects with the same content in the *heap* area but not in *SCP*. That is duplicate objects are possible in the *heap* area but not in *SCP*.

Example 2:

```
String s1 = new String("Ahmed");
s1.concat(" Elhilali");
String s2 = s1.concat(" El");
s1 = s1.concat(" Keller");
System.out.println(s1); => Ahmed El
System.out.println(s2); => Ahmed Keller
```

Heap	SCP
s1 -> Ahmed	Ahmed
Ahmed Elhilali	Elhilali
s2 -> Ahmed El	El
s1 -> Ahmed Keller	Keller

Note:

- For every string constant one object will be placed in *SCP* area.
- Because of some *runtime* operation, if an object is required to get created, that object will be placed only in the *heap* area, but not in *SCP* area.

Example 3:

```
String s1 = new String("Spring");
s1.concat(" Summer");
String s2 = s1.concat(" Winter");
s1 = s1.concat(" Fall");
System.out.println(s1); => Spring Fall
System.out.println(s2); => Spring Winter
```

Heap	SCP
s1 -> Spring	Spring
Spring Summer	Summer
s2 -> Spring Winter	Winter
s1 -> Spring Fall	Fall

Note:

- For every string constant one object will be placed in *SCP* area.
- Because of some *runtime* operation, if an object is required to get created, that object will be placed only in the *heap* area, but not in *SCP* area.