Strings in Java – String Objects

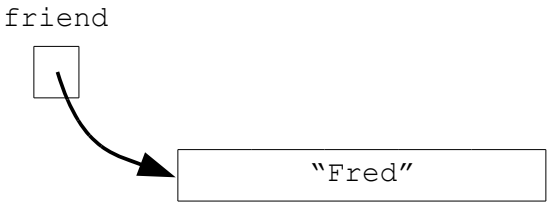
Strings in Java are objects. They are instances of the class `String` (in the package `java.lang`).

As is the case with other objects, `String` variables are actually references to a `String` object in memory. The variable with contain an address, rather than the actual string.

For example, the statement

```
String friend = new String("Fred");
```

creates a new string as shown in the following diagram.

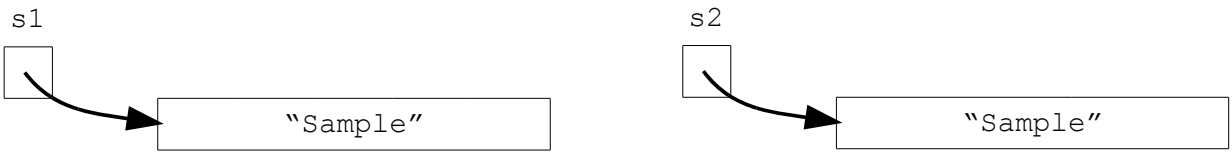


This constructor is rarely used. Java has an equivalent, simpler form to create and initialize a variable of type `String`.

```
String friend = "Fred";
```

The constructor can be useful, however, to create multiple string objects with the same value (but different instances of each object).

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



It is also possible for a String variable to be in a state where it does not refer to a string containing characters. Consider the following:

```
String s1;           // uninitialized
String s2 = null;    // null string
String s3 = "";      // empty string
```

The variable `s1` is uninitialized. It can be assigned to a String, but any other attempt to use this variable will result in an error.

The variable `s2` has been set to the `null` value. It can be printed (producing the output "null") or compared to other strings.

The variable `s3` has been set to the empty string, which has a length of zero and contains no characters, but is also valid to use with all string operations.

Changing a String Value

The String class contains no *mutator* methods. Once we create a string, its value cannot be changed. Because of this, we say strings are *immutable* objects. We can still change the values of the string variable, to reference another string, but the actual string object (in memory) is fixed.

For example, consider the fragment

```
String s = "Hello";
s = "Bonjour";
```

The first line sets the variable `s` to refer to a string object containing `"Hello"`. The second line changes the reference to another object containing `"Bonjour"`. The first object, `"Hello"`, which now has no variables referencing it, it now lost.

Now consider the fragment

```
String s2 = "Repeat";
s2 = "Repeat";
```

Once again, the first line creates s2 and sets it to refer to a string object. The second line creates a *new string object*, even though the strings are exactly the same. The result is the same – two string objects are in memory, but the variable references the second object, and the first has been lost.

Exercises

1. What would be the output of the following fragment?

```
String s;
System.out.println(s);
```

2. What would be printed by this fragment?

```
String s, t, u;
s = null;
t = "";
u = " ";
System.out.println("s is |" + s + "|");
System.out.println("t is |" + t + "|");
System.out.println("u is |" + u + "|");
```

3. What would be printed this fragment?

```
String s = "Sample";
String t = s;
s = "Simple";
System.out.println("s is: " + s);
System.out.println("t is: " + t);
```

4. What would be printed this fragment?

```
String s = "First";
String t = new String(s);
s = "Second";
System.out.println("s is: " + s);
System.out.println("t is: " + t);
```

5. What do we mean when we say strings are *immutable*?

6. Is the following fragment legal? Explain.

```
String s = "old";
s = "new";
```