Using Classes

String Class

Lecture Contents

- Declaring a `String`
- **`String`** Class Methods
  - `length()`
  - `charAt(int index)` ← not part of the AP Subset!
  - `indexOf(String s)`
  - `substring(int start)`
  - `substring(int start, int end)`
  - **`equals(String s)`**
  - **`compareTo(String s)`**

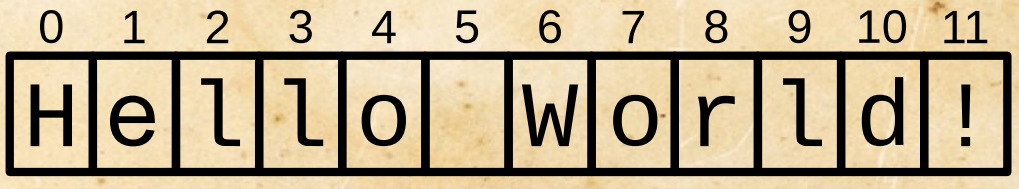
These methods were covered in part 1

`String` Class – Calling Methods

- Common `String` class methods:

```
int length()
int indexOf(String str)
String substring(int start)
String substring(int start, int end)
boolean equals(String other)
int compareTo(String other)

String toUpperCase()
String toLowerCase()
char charAt(int index)
boolean equalsIgnoreCase(String other)
```

`String` Class – `substring()`



- Since we don't cover **`char`** in the AP Subset…
- The substring method returns a String within the range given by a:
  - Start index (then end is set to the end of the string)

    `<String>.substring(int start)`
  - A start and an end index (it includes the start, excludes the end character)

    `<String>.substring(int start, int end)`

`String` Class – Calling Methods

- Common String methods:

```
int length()
int indexOf(String str)
String substring(int start)
String substring(int start, int end)
boolean equals(String other)
int compareTo(String other)

String toUpperCase()
String toLowerCase()
char charAt(int index)
boolean equalsIgnoreCase(String other)
```

Declaring a `String`

- We've been declaring strings using:

```
String s = "Hello World!";
```

Declaring a `String`

- These are (almost) equivalent ways to declare and initialize a string:

```
String s = "Hello World!";

String s;
s = "Hello World!";
```

Declaring a `String`

- These are (almost) equivalent ways to declare and initialize a string:

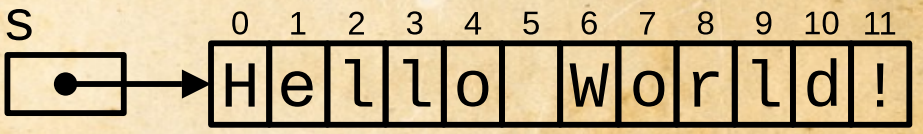
```
String s = "Hello World!";

String s;
s = "Hello World!";

String s = new String("Hello World!");

String s;
s = new String("Hello World!");
```

- The latter two are the “normal” way to declare and initialize objects.
  - The first ways are “sort cut” ways just for the `String` class objects.

Storage of Strings and the `equals` Method

- A `String` is an ***object*** in Java.
- Objects in Java are stored by ***reference***
  - This means the object data is stored in one place, and the ***variable*** contains a number that represents the ***location*** of the object.
  - The following is the code to declare and initialize a string, as well as a diagrammatic representation of the `String` object.

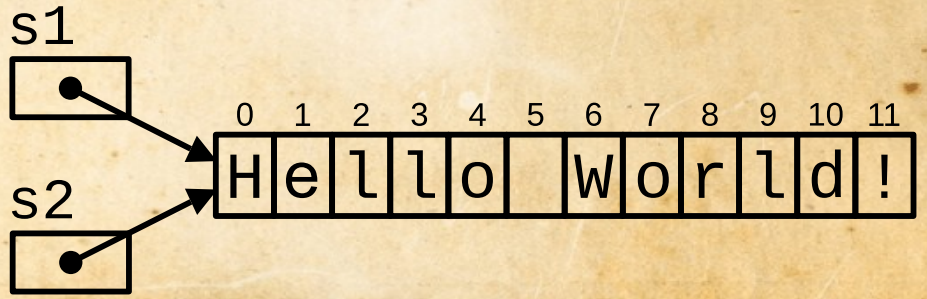
```
String s = "Hello World!";
```



Storage of Strings and the `equals` Method

- A `String` is an ***object*** in Java.
- Objects in Java are stored by ***reference***
- The Java compiler will optimize storage if the program uses the same string in different locations:

```
String s1 = "Hello World!";
String s2 = "Hello World!";
```

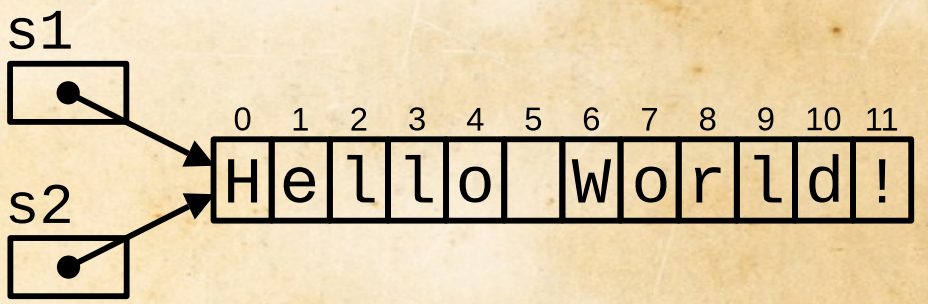


There is only one copy of the `string` *"*`Hello World!`*" stored in memory.*

Storage of Strings and the `equals` Method

- Since both S1 and S2 refer to the same location...

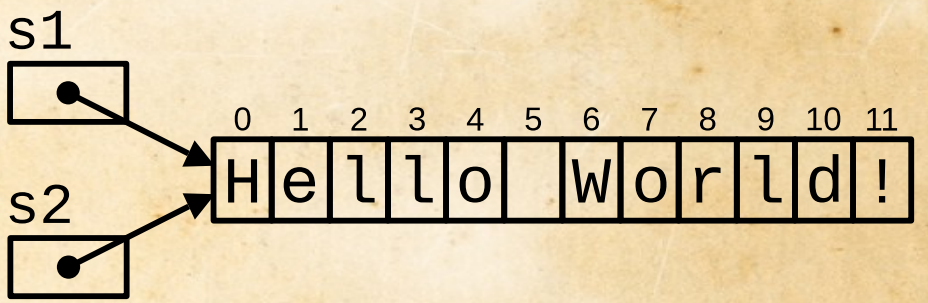
```
String s1 = "Hello World!";
String s2 = "Hello World!";
System.out.println( s1 == s2 );
```



Storage of Strings and the `equals` Method

- Since both S1 and S2 refer to the same location...

```
String s1 = "Hello World!";
String s2 = "Hello World!";
System.out.println( s1 == s2 );
```

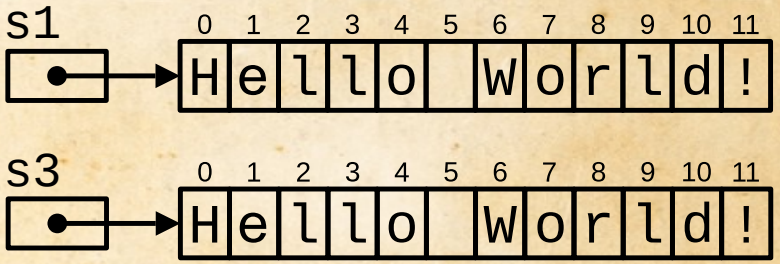



```
true
```

Storage of Strings and the `equals` Method

- When we use the `new` keyword to initialize a string, the compiler will not do the same optimization.

```
String s1 = "Hello World!";
String s3 = new String ("Hello World!");
System.out.println( s1 == s3 );
```



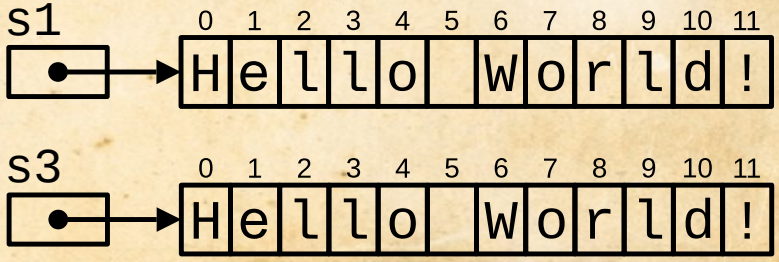
There are two copies of the `string` *“*`Hello World!`*” stored in memory.*

```
false
```

Storage of Strings and the `equals` Method

- If you wish to compare the ***value*** of the data stored in two different objects, use the **`equals`** method.
  - For `String` objects, **`equals`** compares character-by-character to decide if two strings are the same.

```
String s1 = "Hello World!";
String s3 = new String ("Hello World!");
System.out.println( s1 == s3 );
System.out.println( s1.equals(s3) );
```




```
false
true
```

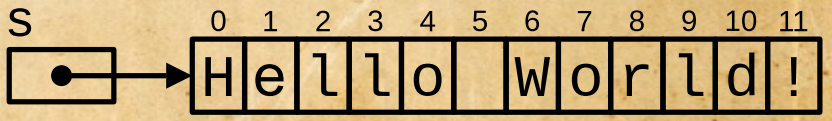
Storage of Strings and the `equals` Method

- Recall using the **`Math`** class methods…

```
int x = Math.sqrt(5);
int y = Math.abs(x);
```

- How does the usage of these **`String`** class methods differ?

```
String s = "Hello World!";
int x = s.length();
int y = s.indexOf("l");
```



Storage of Strings and the `equals` Method

- Recall using the **`Math`** class methods…

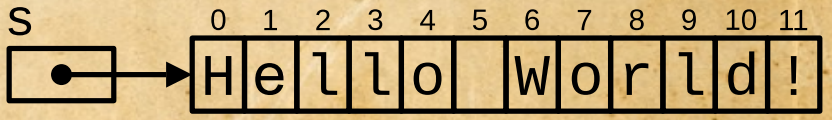
```
int x = Math.sqrt(5);
int y = Math.abs(x);
```

`Math` *is the* ***class*** *identifier*.

- How does the usage of these **`String`** class methods differ?

```
String s = "Hello World!";
int x = s.length();
int y = s.indexOf("l");
```

`s` *is the* ***object*** *identifier*.



- `String` methods operate on a `String` object, above this is object "`s`".
- We don't have a `Math` object, only a `Math` class.

`String` Class – Calling Methods

- Common String methods:

```
int length()
int indexOf(String str)
String substring(int start)
String substring(int start, int end)
boolean equals(String other)
int compareTo(String other)

String toUpperCase()
String toLowerCase()
char charAt(int index)
boolean equalsIgnoreCase(String other)
```

`String` Class – `compareTo()`

- If we wish to put two strings in alphabetical order, we can use the `compareTo` method.
- The return value for `s1.compareTo(s2)` is an integer that is:
  - Equal to zero (`0`) if the strings are exactly equal
  - A positive value if `s1` is earlier than `s2`, alphabetically
  - A negative value if `s1` is later than `s2`, alphabetically.

    (upper case comes before lower case)

`String` Class – `compareTo()`

- If we wish to put two strings in alphabetical order, we can use the `compareTo` method.
- The return value for `s1.compareTo(s2)` is an integer that is:
  - Equal to zero (`0`) if the strings are exactly equal
  - A positive value if `s1` is earlier than `s2`, alphabetically
  - A negative value if `s1` is later than `s2`, alphabetically.

In actuality, it compares the ***unicode*** values of each character of the string, one-by-one, and returns the value of the first character that differs:

```
s1.charAt(k) – s2.charAt(k)
```

Or, if they're the same until the end of a shorter string:

```
s1.length() – s2.length()
```

`String` Class – `compareTo()`

- Determine the output of each:

```
String s1 = "Hello";
String s2 = "hello";
String s3 = "Hello World!";
String s4 = new String("Hello");
System.out.println(s1.compareTo(s2));
System.out.println(s2.compareTo(s1));
System.out.println(s1.compareTo(s3));
System.out.println(s1 == s4);
System.out.println(s1.equals(s4));
System.out.println(s1.compareTo(s4));
```

The difference between unicode 64 (0x40) and unicode 96 (0x60)

`String` Class – `compareTo()`

- Determine the output of each:

```
String s1 = "Hello";
String s2 = "hello";
String s3 = "Hello World!";
String s4 = new String("Hello");
System.out.println(s1.compareTo(s2));
System.out.println(s2.compareTo(s1));
System.out.println(s1.compareTo(s3));
System.out.println(s1 == s4);
System.out.println(s1.equals(s4));
System.out.println(s1.compareTo(s4));
```

The difference between unicode 64 (0x40) and unicode 96 (0x60)

```
-32
```

`String` Class – `equals()`

- Determine the output of each:

```
String s1 = "Hello";
String s2 = "hello";
String s3 = "Hello World!";
String s4 = new String("Hello");
System.out.println(s1.compareTo(s2));
System.out.println(s2.compareTo(s1));
System.out.println(s1.compareTo(s3));
System.out.println(s1 == s4);
System.out.println(s1.equals(s4));
System.out.println(s1.compareTo(s4));
```

The difference between unicode 64 (0x40) and unicode 96 (0x60)

```
-32
32
```

`String` Class – `equals()`

- Determine the output of each:

```
String s1 = "Hello";
String s2 = "hello";
String s3 = "Hello World!";
String s4 = new String("Hello");
System.out.println(s1.compareTo(s2));
System.out.println(s2.compareTo(s1));
System.out.println(s1.compareTo(s3));
System.out.println(s1 == s4);
System.out.println(s1.equals(s4));
System.out.println(s1.compareTo(s4));
```

```
-32
32
-7
```

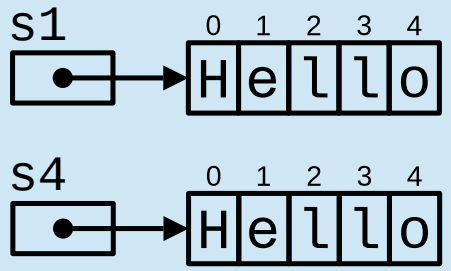
The difference in length between s1 and s2 (5 – 12)

`String` Class – `equals()`

- Determine the output of each:

```
String s1 = "Hello";
String s2 = "hello";
String s3 = "Hello World!";
String s4 = new String("Hello");
System.out.println(s1.compareTo(s2));
System.out.println(s2.compareTo(s1));
System.out.println(s1.compareTo(s3));
System.out.println(s1 == s4);
System.out.println(s1.equals(s4));
System.out.println(s1.compareTo(s4));
```

The *references* are different, but the data stored in each is the same.




```
-32
32
-7
false
true
0
```

`String` Class – Summary

- Common `String` class methods:

```
int length()
int indexOf(String str)
String substring(int start)
String substring(int start, int end)
boolean equals(String other)
int compareTo(String other)
```

Java AP Subset.

```
String toUpperCase()
String toLowerCase()
char charAt(int index)
boolean equalsIgnoreCase(String other)
```

Not included in the Java AP Subset.

Using Classes

String Class