

CISC 3115 TY3

# C08c: A Few Classes in the Java Library (II)

Hui Chen

Department of Computer & Information Science

CUNY Brooklyn College

# Outline

- Discussed
  - Concepts of two programming paradigms
    - Procedural and Object-Oriented
  - Design classes for problem solving
    - Think in terms of class
    - Discover relationship of classes
- A few classes in Java API
  - Java wrapper classes for primitive values
  - BigInteger, BigDecimal
  - String, StringBuilder, StringBuffer

# Wrapper Classes for Primitive Types

- Java has 8 primitive data types
  - char, byte, short, int, long, float, double, boolean
- Wrapper classes
  - Char, Byte, Short, Integer, Long, Float, Double, Boolean
  - They do not have no-arg constructors.
  - The instances of all wrapper classes are immutable, i.e., their internal values cannot be changed once the objects are created

# The Integer and Double Class

## java.lang.Integer

```
-value: int
+MAX VALUE: int
+MIN VALUE: int

+Integer(value: int)
+Integer(s: String)
+byteValue(): byte
+shortValue(): short
+intValue(): int
+longVlaue(): long
+floatValue(): float
+doubleValue():double
+compareTo(o: Integer): int
+toString(): String
+valueOf(s: String): Integer
+valueOf(s: String, radix: int): Integer
+parseInt(s: String): int
+parseInt(s: String, radix: int): int
```

## java.lang.Double

```
-value: double
+MAX VALUE: double
+MIN VALUE: double

+Double(value: double)
+Double(s: String)
+byteValue(): byte
+shortValue(): short
+intValue(): int
+longVlaue(): long
+floatValue(): float
+doubleValue():double
+compareTo(o: Double): int
+toString(): String
+valueOf(s: String): Double
+valueOf(s: String, radix: int): Double
+parseDouble(s: String): double
+parseDouble(s: String, radix: int): double
```

# Integer and Double

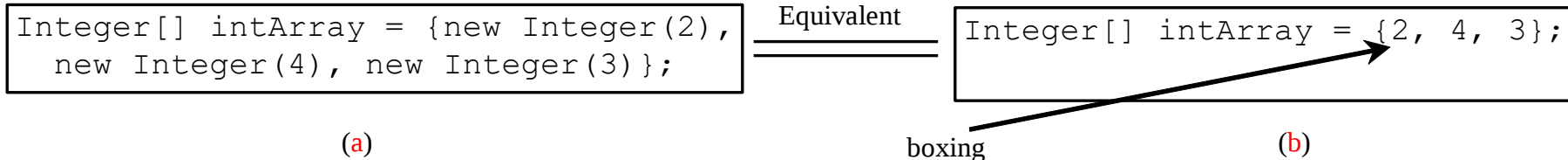
- Constructors
  - `public Integer(int value)`
  - `public Integer(String s)`
  - `public Double(double value)`
  - `public Double(String s)`
- Class constants
  - `MAX_VALUE`
  - `MIN_VALUE`
- Conversion methods
  - "convert" objects into corresponding primitive type values.

# valueOf and the parsing Methods

- The valueOf convenience method
  - `Double doubleObject = Double.valueOf("12.4");`
  - `Integer integerObject = Integer.valueOf("12");`
- The parsing methods
  - `int i = Integer.parseInt("123");`
  - `double d = Double.parseDouble("3.1415");`

# Boxing and Unboxing

- Java allows primitive type and wrapper classes to be converted automatically.
- Example: the following statement in (a) can be simplified as in (b)



`Integer[] intArray = {1, 2, 3};`  
`System.out.println(intArray[0] + intArray[1] + intArray[2]);`

Unboxing

# Questions?

- Wrapper classes of primitive data types
  - Char
  - Byte
  - Short
  - Integer
  - Long
  - Float
  - Double
  - Boolean

# BigInteger and BigDecimal

- Two classes for computing very large integers or high precision floating-point values
  - Example: finance applications
  - `java.math.BigInteger`
  - `java.math.BigDecimal`

# BigInteger: Example

```
BigInteger a = new BigInteger("9223372036854775807");  
BigInteger b = new BigInteger("2");  
BigInteger c = a.multiply(b); // 9223372036854775807 * 2  
System.out.println(c);
```

# BigDecimal: Example

```
BigDecimal a = new BigDecimal(1.0);
```

```
BigDecimal b = new BigDecimal(3);
```

```
BigDecimal c = a.divide(b, 20, BigDecimal.ROUND_UP);
```

```
System.out.println(c);
```

# Questions?

- BigInteger
- BigDecimal

# The String Class

- Constructing a String:

```
String message = "Welcome to Java";
```

```
String message = new String("Welcome to Java");
```

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

- Obtaining String length and Retrieving Individual Characters in a string
- String Concatenation (concat)
- Substrings (substring(index), substring(start, end))
- Comparisons (equals, compareTo)
- String Conversions
- Finding a Character or a Substring in a String
- Conversions between Strings and Arrays
- Converting Characters and Numeric Values to Strings

# Constructing Strings

- Since strings are used frequently, Java provides a shorthand initializer (string initializer) for creating a string:
  - `String message = "Welcome to Java";`
- The above is the preferred method to construct String objects. However, it does have a constructor
  - `String newString = new String(stringLiteral);`
  - Example
    - `String message = new String("Welcome to Java");`

# Strings Are Immutable

- A String object is immutable; its contents cannot be changed.

# Strings Are Immutable:

## Discussion

- A String object is immutable; its contents cannot be changed.
- Does the following code change the contents of the string?

```
String s = "Java";
```

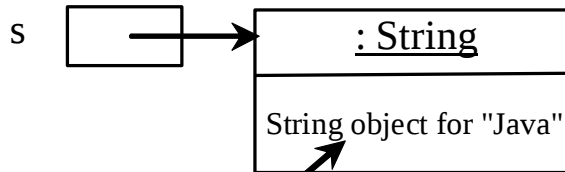
```
s = "HTML";
```

# Strings Are Immutable: Example

```
String s = "Java";
```

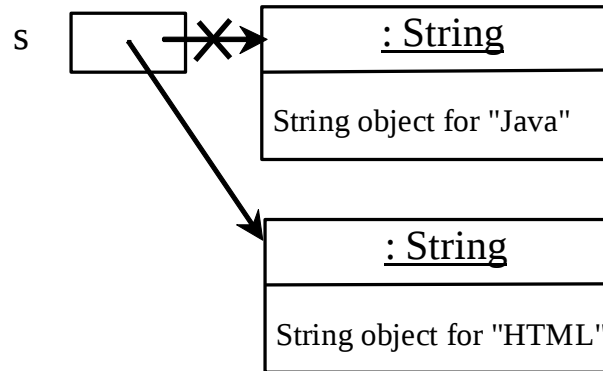
```
s = "HTML";
```

After executing `String s = "Java";`



Contents cannot be changed

After executing `s = "HTML";`



This string object is now unreferenced

# Interned Strings

- To improve efficiency and save memory, the JVM uses a unique instance for string literals with the same character sequence. Such an instance is called *interned*.

# Interned Strings: Discussion

- What should we observe?

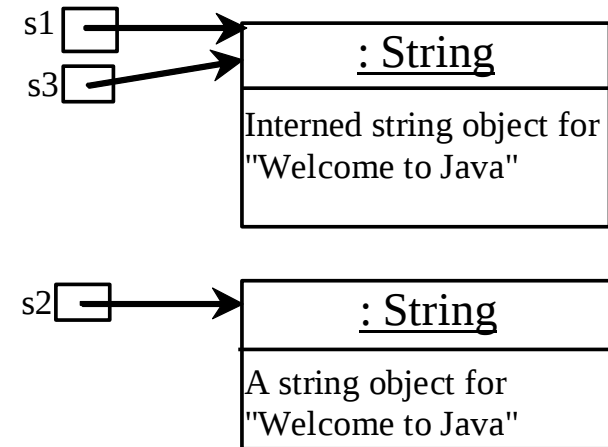
```
String s1 = "Welcome to Java";
```

```
String s2 = new String("Welcome to Java");
```

```
String s3 = "Welcome to Java";
```

```
System.out.println("s1 == s2 is " + (s1 == s2));
```

```
System.out.println("s1 == s3 is " + (s1 == s3));
```



# Interned Strings: Discussion:

## Answer

- What should we observe?

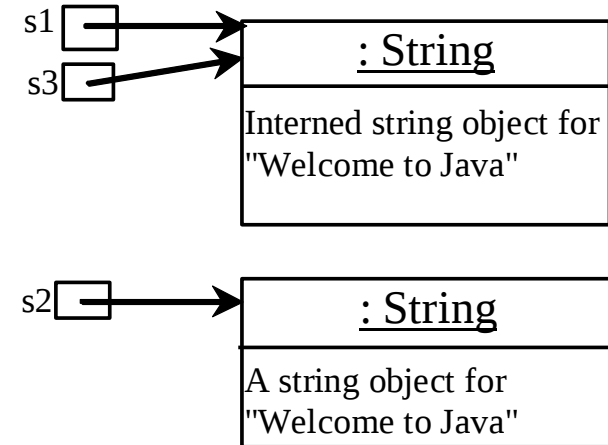
```
String s1 = "Welcome to Java";
```

```
String s2 = new String("Welcome to Java");
```

```
String s3 = "Welcome to Java";
```

```
System.out.println("s1 == s2 is " + (s1 == s2));
```

```
System.out.println("s1 == s3 is " + (s1 == s3));
```



display

s1 == s is false

s1 == s3 is true

A new object is created if you use the new operator.

If you use the string initializer, no new object is created if the interned object is already created.

# Replacing and Splitting Strings

| java.lang.String                                            |                                                                                                        |
|-------------------------------------------------------------|--------------------------------------------------------------------------------------------------------|
| +replace(oldChar: char, newChar: char): String              | Returns a new string that replaces all matching character in this string with the new character.       |
| +replaceFirst(oldString: String, newString: String): String | Returns a new string that replaces the first matching substring in this string with the new substring. |
| +replaceAll(oldString: String, newString: String): String   | Returns a new string that replace all matching substrings in this string with the new substring.       |
| +split(delimiter: String): String[]                         | Returns an array of strings consisting of the substrings split by the delimiter.                       |

# Replacing Strings: Examples

- `"Welcome".replace('e', 'A')`
  - returns a new string, `WAlcomA`.
- `"Welcome".replaceFirst("e", "AB")`
  - returns a new string, `WABlcome`.
- `"Welcome".replace("e", "AB")`
  - returns a new string, `WABlcomAB`.
- `"Welcome".replace("el", "AB")`
  - returns a new string, `WABcome`.

# Splitting Strings: Examples

```
String[] tokens =  
"Java#HTML#Perl".split("#", 0);  
for (int i = 0; i < tokens.length; i++) {  
    System.out.print(tokens[i] + " ");  
}
```

# Patterns

- We can match, replace, or split a string by specifying a pattern, commonly known as *regular expression*.
  - To be discussed in depth in "Theoretical Computer Science (CISC 3230) "
- Two simple patterns are used in this discussion

# Matching Patterns: Examples

```
"Java".matches("Java");
```

```
"Java".equals("Java");
```

```
"Java is fun".matches("Java.*");
```

```
"Java is cool".matches("Java.*")
```

# Replacing and Splitting with Patterns

- The `replaceAll`, `replaceFirst`, and `split` methods can be used with a regular expression.

# Replacing with Patterns: Example

- The following statement returns a new string that replaces \$, +, or # in "a+b\$#c" by the string NNN.

```
String s = "a+b$#c".replaceAll("[$+#]", "NNN");
```

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

- where the regular expression `[$+#]` specifies a pattern that matches \$, +, or #. So, the output is `aNNNbNNNNNNNNc`.

# Splitting with Patterns: Example

- The following statement splits the string into an array of strings delimited by some punctuation marks.

```
String[] tokens = "Java,C?C#,C++".split("[.,:;?]");
```

```
for (int i = 0; i < tokens.length; i++)
```

```
    System.out.println(tokens[i]);
```

# String: The valueOf Method

- The String class provides several static valueOf methods for converting a character, an array of characters, and numeric values to strings.
- Example
  - `String.valueOf(5.44)`.
  - The return value is a string consisting of characters '5', '.', '4', and '4'.

# Questions?

- The String class.
- Strings are immutable.
- String initializer and String constructors
- Manipulating strings
  - Matching, replacing, and splitting
- String concatenation
- String's valueOf method

# String Builder and StringBuffer

- These two classes represent string objects as well. However, they are mutable.
- We can add, insert, or append new contents into a StringBuiler or StringBuffer objects.
- StringBuffer is synchronized, which means that it can used safely in concurrent programming (but also slower than StringBuilder)
  - To be discussed in the future

# StringBuilder: Constructors

| java.lang.StringBuilder                                                        |
|--------------------------------------------------------------------------------|
| +StringBuilder()<br>+StringBuilder(capacity: int)<br>+StringBuilder(s: String) |

Constructs an empty string builder with capacity 16.

Constructs a string builder with the specified capacity.

Constructs a string builder with the specified string.

# StringBuilder: Modify String Content

| java.lang.StringBuilder                                                    |                                                                                                |
|----------------------------------------------------------------------------|------------------------------------------------------------------------------------------------|
| +append(data: char[]): StringBuilder                                       | Appends a char array into this string builder.                                                 |
| +append(data: char[], offset: int, len: int):<br>StringBuilder             | Appends a subarray in data into this string builder.                                           |
| +append(v: <i>aPrimitiveType</i> ): StringBuilder                          | Appends a primitive type value as a string to this builder.                                    |
| +append(s: String): StringBuilder                                          | Appends a string to this string builder.                                                       |
| +delete(startIndex: int, endIndex: int):<br>StringBuilder                  | Deletes characters from startIndex to endIndex.                                                |
| +deleteCharAt(index: int): StringBuilder                                   | Deletes a character at the specified index.                                                    |
| +insert(index: int, data: char[], offset: int,<br>len: int): StringBuilder | Inserts a subarray of the data in the array to the builder at the specified index.             |
| +insert(offset: int, data: char[]):<br>StringBuilder                       | Inserts data into this builder at the position offset.                                         |
| +insert(offset: int, b: <i>aPrimitiveType</i> ):<br>StringBuilder          | Inserts a value converted to a string into this builder.                                       |
| +insert(offset: int, s: String): StringBuilder                             | Inserts a string into this builder at the position offset.                                     |
| +replace(startIndex: int, endIndex: int, s:<br>String): StringBuilder      | Replaces the characters in this builder from startIndex to endIndex with the specified string. |
| +reverse(): StringBuilder                                                  | Reverses the characters in the builder.                                                        |
| +setCharAt(index: int, ch: char): void                                     | Sets a new character at the specified index in this builder.                                   |

# StringBuilder: Examples

- `StringBuilder.append("Java");`
- `StringBuilder.insert(11, "HTML and ");`
- `StringBuilder.delete(8, 11)`
- `StringBuilder.deleteCharAt(8)`
- `StringBuilder.reverse()`
- `StringBuilder.replace(11, 15, "HTML")`
- `StringBuilder.setCharAt(0, 'w')`

# StringBuilder: toString, length, capacity, setLength, charAt

| java.lang.StringBuilder                            |                                                       |
|----------------------------------------------------|-------------------------------------------------------|
| +toString(): String                                | Returns a string object from the string builder.      |
| +capacity(): int                                   | Returns the capacity of this string builder.          |
| +charAt(index: int): char                          | Returns the character at the specified index.         |
| +length(): int                                     | Returns the number of characters in this builder.     |
| +setLength(newLength: int): void                   | Sets a new length in this builder.                    |
| +substring(startIndex: int): String                | Returns a substring starting at startIndex.           |
| +substring(startIndex: int, endIndex: int): String | Returns a substring from startIndex to endIndex-1.    |
| +trimToSize(): void                                | Reduces the storage size used for the string builder. |

# Questions?

- String
  - Immutable
- StringBuilder
  - Mutable, not thread-safe, fast
- StringBuffer
  - Mutable, thread-safe, slow