

05 - Wrapper Classes and Strings

CS202: Introduction to Object Oriented Programming
Victor Mejia
CSULA

Today's Topics:

- Processing Primitive Data Type Values as Objects
- The BigInteger and BigDecimal Classes
- Strings

Processing Primitive Types as Object Types

Wrapper Classes

- Primitive data types are not objects
- language's performance would be affected if every data type would be an object
- many Java methods require the use of objects as arguments
- Java offers a convenient way to incorporate, or wrap, a primitive data type into an object

example: wrapping **int** into the Integer class

Wrapper Classes

java.lang.Integer	java.lang.Double
-value: int <u>+MAX_VALUE: int</u> <u>+MIN_VALUE: int</u>	-value: double <u>+MAX_VALUE: double</u> <u>+MIN_VALUE: double</u>
+Integer(value: int) +Integer(s: String) +byteValue(): byte +shortValue(): short +intValue(): int +longValue(): long +floatValue(): float +doubleValue(): double +compareTo(o: Integer): int +toString(): String <u>+valueOf(s: String): Integer</u> <u>+valueOf(s: String, radix: int): Integer</u> <u>+parseInt(s: String): int</u> <u>+parseInt(s: String, radix: int): int</u>	+Double(value: double) +Double(s: String) +byteValue(): byte +shortValue(): short +intValue(): int +longValue(): long +floatValue(): float +doubleValue(): double +compareTo(o: Double): int +toString(): String <u>+valueOf(s: String): Double</u> <u>+valueOf(s: String, radix: int): Double</u> <u>+parseDouble(s: String): double</u> <u>+parseDouble(s: String, radix: int): double</u>

NOTE: The wrapper classes do not have no-arg constructors.

The Integer and Double Classes

- Constructors
- Class Constants `MAX_VALUE`,
`MIN_VALUE`
- Conversion Methods

Numeric Wrapper Class Constructors

You can construct a wrapper object :

- either from a primitive data type value
- or from a string representing the numeric value.

The constructors for **Integer** and **Double** are:

```
public Integer(int value)
```

```
public Integer(String s)
```

```
public Double(double value)
```

```
public Double(String s)
```

The Integer and Double Constants

Each numerical wrapper class has the constants `MAX_VALUE` and `MIN_VALUE`.

- `MAX_VALUE` represents the maximum value of the corresponding primitive data type.
- For `Byte`, `Short`, `Integer`, and `Long`, `MIN_VALUE` represents the minimum byte, short, int, and long values.
- For `Float` and `Double`, `MIN_VALUE` represents the minimum *positive* float and double values.

The Integer and Double Constants

The following statements display the maximum integer (2,147,483,647), the minimum positive float (1.4E-45), and the maximum double floating-point number (1.79769313486231570e+308d)

```
System.out.println("The maximum integer is " + Integer.MAX_VALUE);  
System.out.println("The minimum positive float is " +  
    Float.MIN_VALUE);  
System.out.println(  
    "The maximum double-precision floating-point number is " +  
    Double.MAX_VALUE);
```

Conversion Methods

- Each numeric wrapper class implements the abstract methods `doubleValue`, `floatValue`, `intValue`, `longValue`, and `shortValue`, which are defined in the `Number` class.
- These methods “convert” objects into primitive type values.

```
new Double(12.4).intValue() returns 12;  
new Integer(12).doubleValue() returns 12.0;
```

compareTo Method

returns 1, 0, or -1, if this number is greater than, equal to, or less than the other number.

```
new Double(12.4).compareTo(new Double(12.3)) returns 1;  
new Double(12.3).compareTo(new Double(12.3)) returns 0;  
new Double(12.3).compareTo(new Double(12.51)) returns -1;
```

The static valueOf Methods

The numeric wrapper classes have a useful class method, `valueOf(String s)`. This method creates a new object initialized to the value represented by the specified string. For example:

```
Double doubleObject = Double.valueOf  
    ("12.4");
```

```
Integer integerObject = Integer.valueOf  
    ("12");
```

Parsing Strings Into Numbers

- You have used the **parseInt** method in the **Integer** class to parse a numeric string into an int value and the **parseDouble** method in the **Double** class to parse a numeric string into a double value.
- Each numeric wrapper class has two overloaded parsing methods to parse a numeric string into an appropriate numeric value.

Parsing Strings Into Numbers

```
// These two methods are in the Byte class
public static byte parseByte(String s)
public static byte parseByte(String s, int radix)

// These two methods are in the Short class
public static short parseShort(String s)
public static short parseShort(String s, int radix)

// These two methods are in the Integer class
public static int parseInt(String s)
public static int parseInt(String s, int radix)

// These two methods are in the Long class
public static long parseLong(String s)
public static long parseLong(String s, int radix)

// These two methods are in the Float class
public static float parseFloat(String s)
public static float parseFloat(String s, int radix)

// These two methods are in the Double class
public static double parseDouble(String s)
public static double parseDouble(String s, int radix)
```

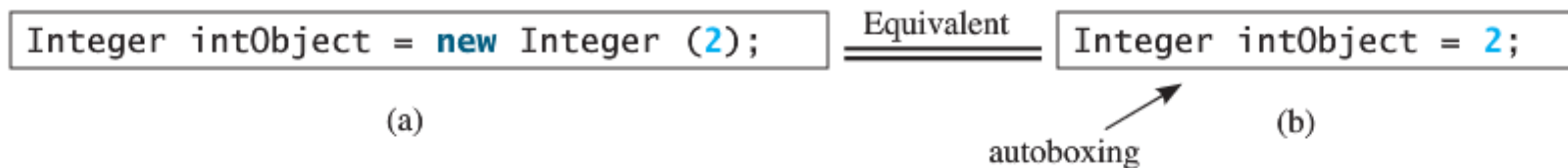
```
Integer.parseInt("11", 2) returns 3;
Integer.parseInt("12", 8) returns 10;
Integer.parseInt("13", 10) returns 13;
Integer.parseInt("1A", 16) returns 26;
```

Automatic Conversion Between Primitive Types and Wrapper Class Types

A primitive type value can be automatically converted to an object using a wrapper class, and vice versa, depending on the context.

boxing: Converting a primitive value to a wrapper object

unboxing: Converting a wrapper object to a primitive value



Automatic Conversion Between Primitive Types and Wrapper Class Types

Consider the following example:

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

In line 1, the primitive values **1**, **2**, and **3** are automatically boxed into objects **new Integer(1)**, **new Integer(2)**, and **new Integer(3)**. In line 2, the objects **intArray[0]**, **intArray[1]**, and **intArray[2]** are automatically unboxed into **int** values that are added together.

BigInteger and BigDecimal Classes

- If you need to compute with very large integers or high precision floating-point values, you can use the **BigInteger** and **BigDecimal** classes in the **java.math** package.
- Both are *immutable*.
- Both extend the Number class and implement the Comparable interface.

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

```
BigDecimal a = new BigDecimal(1.0);  
BigDecimal b = new BigDecimal(3);  
BigDecimal c = a.divide(b, 20, BigDecimal.  
ROUND_UP);  
System.out.println(c);
```

Strings

Intro

- **key point:** A String object is *immutable*: its contents cannot be changed once the string is created
- Strings are objects (reference types)
- Methods you have used before:
 - substring
 - charAt
 - length

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

```
String newString = new String(stringLiteral);
```

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

Since strings are used frequently, Java provides a shorthand initializer for creating a string:

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

Constructing Strings

You can also create a string from an array of characters.

```
char[] charArray = {'G', 'o', 'o', 'd', ' ', ' ', 'D', 'a', 'y'};  
String message = new String(charArray);
```

Strings Are Immutable

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

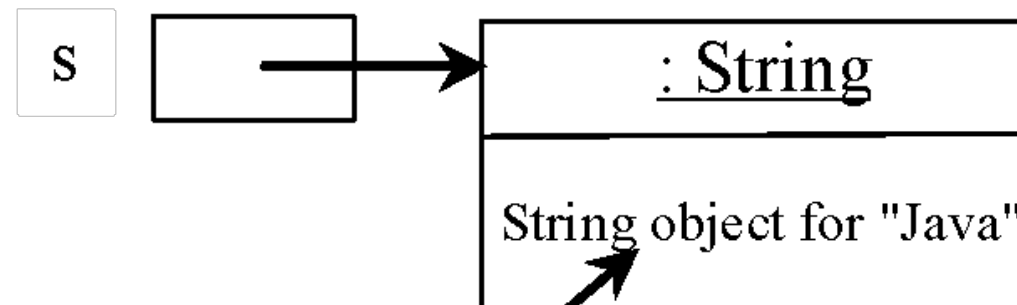
Does the following code change the contents of the string?

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

Trace Code

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

After executing `String s = "Java";`



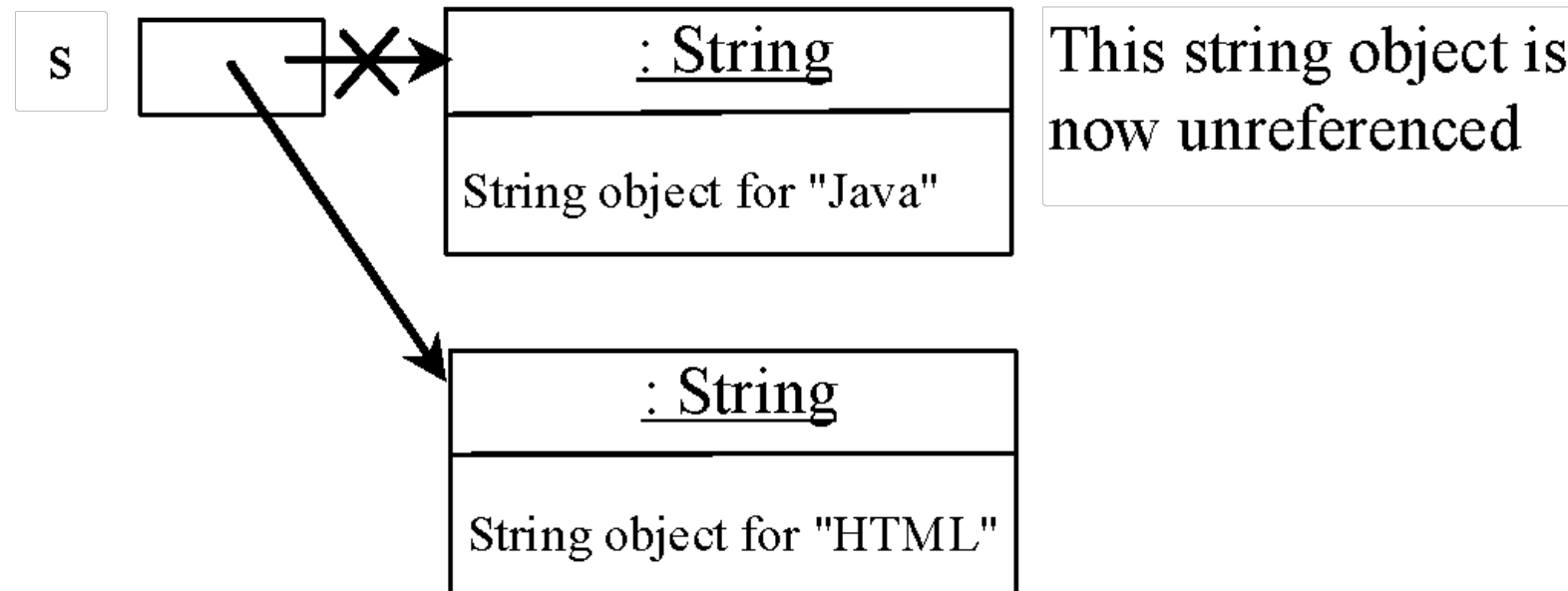
Contents cannot be changed

Trace Code

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

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

After executing `s = "HTML";`

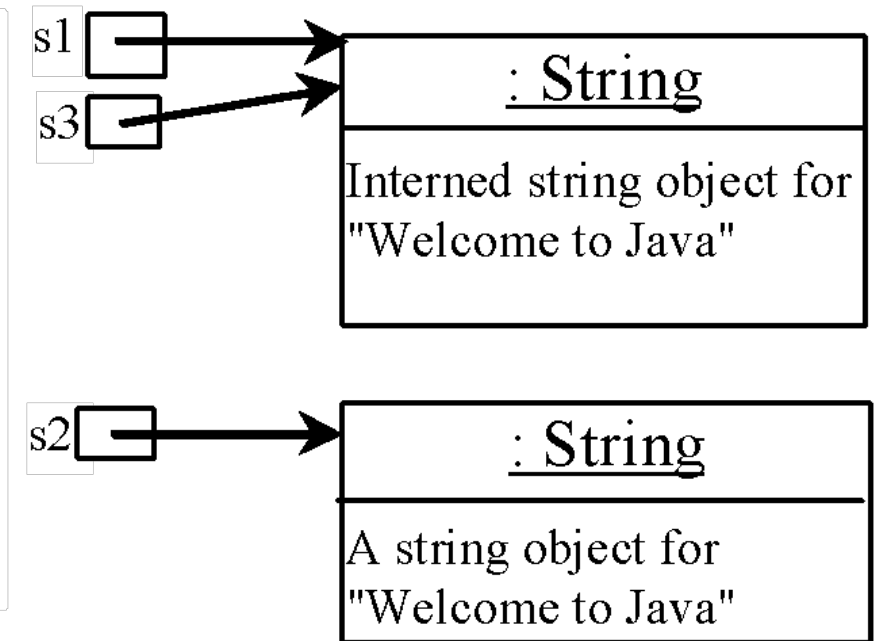


Interned Strings

Since strings are immutable and are frequently used, 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*.

Examples

```
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));
```

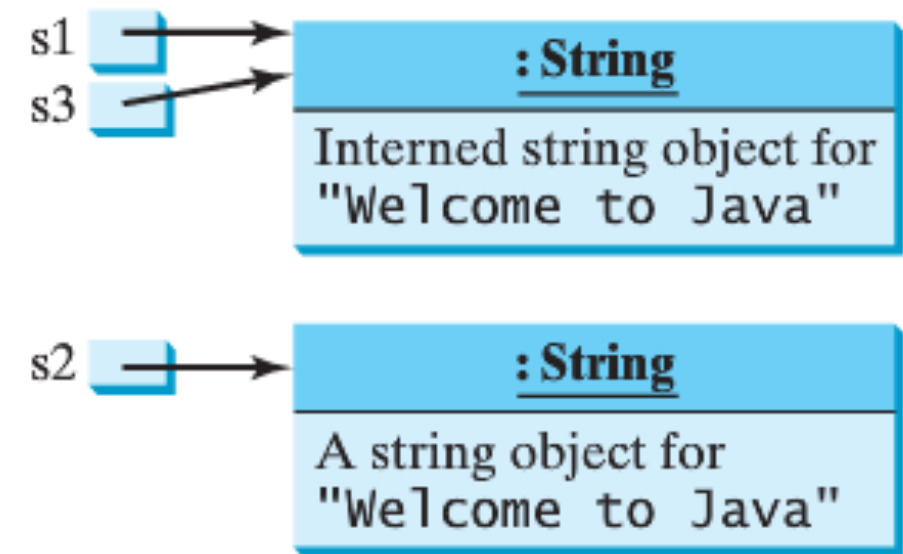


A new object is created if you use the new operator.

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

Trace Code

```
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));
```



Replacing and Splitting Strings

java.lang.String

```
+replace(oldChar: char,  
  newChar: char): String  
+replaceFirst(oldString: String,  
  newString: String): String  
+replaceAll(oldString: String,  
  newString: String): String  
+split(delimiter: String):  
  String[]
```

Returns a new string that replaces all matching characters in this string with the new character.

Returns a new string that replaces the first matching substring in this string with the new substring.

Returns a new string that replaces all matching substrings in this string with the new substring.

Returns an array of strings consisting of the substrings split by the delimiter.

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 a String

```
String[] tokens = "Java$C#$Ruby".split("$", 0);  
for (int i = 0; i < tokens.length; i++)  
    System.out.print(tokens[i] + " ");
```

Java C# Ruby

Convert Character and Numbers to Strings

The String class provides several static `valueOf` methods for converting a character, an array of characters, and numeric values to strings. These methods have the same name `valueOf` with different argument types `char`, `char[]`, `double`, `long`, `int`, and `float`. For example, to convert a double value to a string, use `String.valueOf(5.44)`. The return value is string consists of characters '5', '.', '4', and '4'.

```
char[] chars = "Java".toCharArray();
```

```
String str = new String(new char[]{'J', 'a', 'v', 'a'});
```

java.lang.String

+valueOf(c: char): String
+valueOf(data: char[]): String
+valueOf(d: double): String
+valueOf(f: float): String
+valueOf(i: int): String
+valueOf(l: long): String
+valueOf(b: boolean): String

Returns a string consisting of the character **c**.
Returns a string consisting of the characters in the array.
Returns a string representing the **double** value.
Returns a string representing the **float** value.
Returns a string representing the **int** value.
Returns a string representing the **long** value.
Returns a string representing the **boolean** value.

StringBuilder and StringBuffer

The StringBuilder/StringBuffer class is an alternative to the String class. In general, a StringBuilder/StringBuffer can be used wherever a string is used.

StringBuilder/StringBuffer is more flexible than String. You can add, insert, or append new contents into a string buffer, whereas the value of a String object is fixed once the string is created.

StringBuilder Constructors

java.lang.StringBuilder

+StringBuilder()
+StringBuilder(capacity: int)
+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.

Examples

`StringBuilder.append("Java");`

`StringBuilder.insert(11, "HTML and ");`

`StringBuilder.delete(8, 11)` changes the builder to
Welcome Java.

`StringBuilder.deleteCharAt(8)` changes the builder to
Welcome o Java.

`StringBuilder.reverse()` changes the builder to avaJ ot
emocleW.

`StringBuilder.replace(11, 15, "HTML")`

changes the builder to Welcome to HTML.

`StringBuilder.setCharAt(0, 'w')` sets the builder to
welcome to Java.

Modifying Strings in the StringBuilder class

`java.lang.StringBuilder`

```
+append(data: char[]): StringBuilder
+append(data: char[], offset: int, len: int):
  StringBuilder
+append(v: aPrimitiveType): StringBuilder

+append(s: String): StringBuilder
+delete(startIndex: int, endIndex: int):
  StringBuilder
+deleteCharAt(index: int): StringBuilder
+insert(index: int, data: char[], offset: int,
  len: int): StringBuilder
+insert(offset: int, data: char[]):
  StringBuilder
+insert(offset: int, b: aPrimitiveType):
  StringBuilder
+insert(offset: int, s: String): StringBuilder
+replace(startIndex: int, endIndex: int, s:
  String): StringBuilder
+reverse(): StringBuilder
+setCharAt(index: int, ch: char): void
```

Appends a `char` array into this string builder.

Appends a subarray in `data` into this string builder.

Appends a primitive type value as a string to this builder.

Appends a string to this string builder.

Deletes characters from `startIndex` to `endIndex-1`.

Deletes a character at the specified index.

Inserts a subarray of the data in the array into the builder at the specified index.

Inserts data into this builder at the position `offset`.

Inserts a value converted to a string into this builder.

Inserts a string into this builder at the position `offset`.

Replaces the characters in this builder from `startIndex` to `endIndex-1` with the specified string.

Reverses the characters in the builder.

Sets a new character at the specified index in this builder.

The toString, capacity, length, setLength, and charAt Methods

java.lang.StringBuilder

```
+toString(): String  
+capacity(): int  
+charAt(index: int): char  
+length(): int  
+setLength(newLength: int): void  
+substring(startIndex: int): String  
+substring(startIndex: int, endIndex: int):  
  String  
+trimToSize(): void
```

Returns a string object from the string builder.

Returns the capacity of this string builder.

Returns the character at the specified index.

Returns the number of characters in this builder.

Sets a new length in this builder.

Returns a substring starting at `startIndex`.

Returns a substring from `startIndex` to `endIndex-1`.

Reduces the storage size used for the string builder.