Java Strings

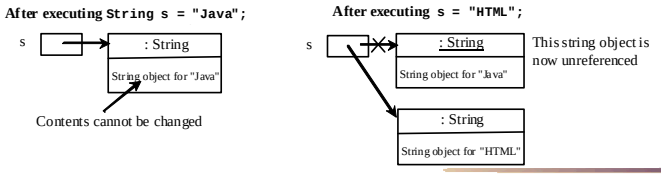
- String variables are a Reference DataType
  - Variable contains memory address of the location of string
  - String class is used to create string objects
  - String objects contain a string of characters
    - `String sFirstName, sLastName;`
- String methods are used access string object instances
  - `sFirstName.toLowerCase()`
- String operators
  - Concatenation `+`
  - Assignment `=` // Changes address not string

Variable declaration as String Object

- Strings objects are an immutable group of characters
- Variables can be declared as String objects
  - `String sFirstName;`
  - `String sFirstName = "Robert";`
  - `String sFirstName = new String("Robert");`
- Declaration as string object allows:
  - Concatenation operator `+` usage
    - `sFullName = sLastName + ", " + sFirstName;`
  - String methods usage
    - `int nLength = sEntry.length(); // returns string length`
    - `String sUpEntry = sEntry.toUpperCase(); // returns Upper Case`
    - `char c1stChar = sUpEntry.charAt(0); // returns first character`
    - `if(sUpEntry.equals("YES")) // compares and returns true or false`
      - `if(sUpEntry == "YES") // Won't work in Java because string immutable`
- Documentation available in API docs and Chapter 9
  - http://docs.oracle.com/javase/6/docs/api/java/lang/String.html

Strings Are Immutable

- A String object is immutable
  - Once created contents of memory cannot be changed
  - Improves processing efficiency
  - Saves memory for repeated strings of characters
  - Java garbage collection deletes String when unreferenced
  - Does the following code change the contents of the string?
    String sCode = "Java";
    sCode = "HTML";
  - No, it changes the reference to a new string of characters



Interned Strings

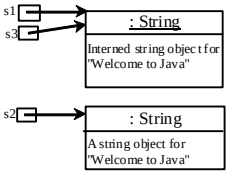
- Java VM can use a unique instance for string literals with the same character sequence created as new string object
- Such an instance is called *interned*

```
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 == s2 is false
- s1 == s3 is true

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

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

String Comparisons

java.lang.String	
+equals(s1: Object): boolean	Returns true if this string is equal to string s1.
+equalsIgnoreCase(s1: String): boolean	Returns true if this string is equal to string s1 case-insensitive.
+compareTo(s1: String): int	Returns an integer greater than 0, equal to 0, or less than 0 to indicate whether this string is greater than, equal to, or less than s1.
+compareToIgnoreCase(s1: String): int	Same as compareTo except that the comparison is case-insensitive.
+regionMatches(toffset: int, s1: String, offset: int, len: int): boolean	Returns true if the specified subregion of this string exactly matches the specified subregion in string s1.
+regionMatches(ignoreCase: boolean, toffset: int, s1: String, offset: int, len: int): boolean	Same as the preceding method except that you can specify whether the match is case-sensitive.
+startsWith(prefix: String): boolean	Returns true if this string starts with the specified prefix.
+endsWith(suffix: String): boolean	Returns true if this string ends with the specified suffix.

 5

String equals Method

❖**equals**

```
String s1 = new String("Welcome");
String s2 = "welcome";
if (s1.equals(s2)){
  // s1 and s2 have the same contents
}
if (s1 == s2) {
 // s1 and s2 have the same reference
}
```

 6

String `compareTo()` Method

❖**compareTo(Object object)**

```
String s1 = new String("Welcome“);
String s2 = "welcome";
if (s1.compareTo(s2) > 0) {
  // s1 is greater than s2
}
else if (s1.compareTo(s2) == 0) {
  // s1 and s2 have the same contents
}
else
   // s1 is less than s2
```

 7

String length, charAt, and concat Methods

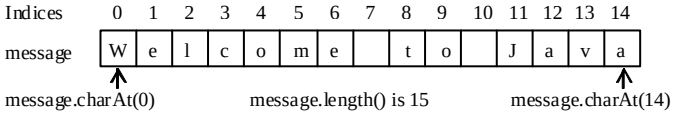
java.lang.String	
+length(): int	Returns the number of characters in this string.
+charAt(index: int): char	Returns the character at the specified index from this string.
+concat(s1: String): String	Returns a new string that concatenate this string with string s1.

❖**Finding string length using the length() method:**

```
message = "Welcome";
message.length() // returns 7
String s3 = message.concat(" to Java);
message.charAt(14)
```

❖**Concatenation operator can also be used**

```
String s3 = message + " to Java";
```



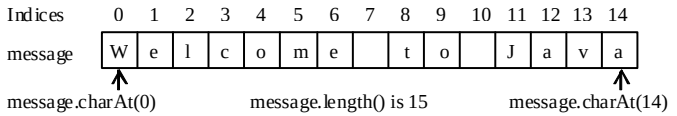
 8

Extracting Substrings

java.lang.String	
+substring(beginIndex: int): String	Returns this string's substring that begins with the character at the specified beginIndex and extends to the end of the string
+substring(beginIndex: int, endIndex: int): String	Returns this string's substring that begins at the specified beginIndex and extends to the character at index endIndex – 1. Note that the character at endIndex is not part of the substring.

You can extract a single character from a string using the charAt method. You can also extract a substring from a string using the substring method in the String class.

```
String s1 = "Welcome to Java";
String s2 = s1.substring(0, 11) + "HTML";
```



 9

Converting, Replacing, and Splitting Strings

java.lang.String	
+toLowerCase(): String	Returns a new string with all characters converted to lowercase.
+toUpperCase(): String	Returns a new string with all characters converted to uppercase.
+trim(): String	Returns a new string with blank characters trimmed on both sides.
+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.

```
"Welcome".toLowerCase()  // returns a new string, welcome.
"  Welcome  ".trim()  // returns a new string, Welcome.
"Welcome".replace('e', 'A') //  returns a new string, WAlcomA.
"Welcome".replaceFirst("e", "AB") // returns a new string, WABlcome.
```

 10

Finding Character or Substring in String

java.lang.String	
+indexOf(ch: char): int	Returns the index of the first occurrence of ch in the string. Returns -1 if not matched.
+indexOf(ch: char, fromIndex: int): int	Returns the index of the first occurrence of ch after fromIndex in the string. Returns -1 if not matched.
+indexOf(s: String): int	Returns the index of the first occurrence of string s in this string. Returns -1 if not matched.
+indexOf(s: String, fromIndex: int): int	Returns the index of the first occurrence of string s in this string after fromIndex. Returns -1 if not matched.
+lastIndexOf(ch: int): int	Returns the index of the last occurrence of ch in the string. Returns -1 if not matched.
+lastIndexOf(ch: int, fromIndex: int): int	Returns the index of the last occurrence of ch before fromIndex in this string. Returns -1 if not matched.
+lastIndexOf(s: String): int	Returns the index of the last occurrence of string s. Returns -1 if not matched.
+lastIndexOf(s: String, fromIndex: int): int	Returns the index of the last occurrence of string s before fromIndex. Returns -1 if not matched.

```
"Welcome to Java".indexOf('W') // returns 0.
"Welcome to Java".indexOf('x') // returns -1.
"Welcome to Java".indexOf('o', 5) // returns 9.
```

 11

Formatted String Output

- **Integers, floating-point numbers, and Strings appearance can be controlled with String.format() static method**
- **The format specifiers for general, character, and numeric types syntax: %[argument_index$][flags][width][.precision]conversion**
  String.format("%n RUN #%d Results for %,d dealt hands", nRun, nCount)
  String.format("%n %,12d = %s", naryKindCounter[nJ], saryKind[nJ]);
- **Especially useful in printing columns with numbers**

http://docs.oracle.com/javase/6/docs/api/java/lang/String.html#format(java.lang.String, java.lang.Object...)

Symbol	Description
%c	Character place holder [char]
%,12d	Decimal place holder 12 spaces, thousands, [int, long, short]
%8.2f	Floating point 8 wide and 2 places right . [double, float]
%s	String placeholder [String]
%n	Newline - OS independent Windows file output my need \r\n
#	A digit placeholder; zero shows as absent and not as space
0	A digit placeholder and automatic fill character
,	Grouping placeholder
%	Multiply by 100 and add a % sign

 12

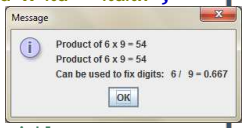
String.format() method

- The usual way to assemble strings with variables

```
sDisplay = "Product of " + nNum1 + " x " + nNum2 + " = "
     + nNum1*nNum2 + "\n";
```

- New style using String format static method similar to `printf()`

```
sDisplay = String.format("Product of %d x %d = %d%n",
     nNum1, nNum2, nNum1 * nNum2);
```




```
1.  import javax.swing.*;
2.  public class StringFormat {
3.    public static void main(String[] args) {
4.      int nNum1 = 6, nNum2 = 9;  // Declares int variables
5.      String sDisplay; // Declares String variables
6.      sDisplay = "Product of " + nNum1 + " x " + nNum2 + " = "
7.       + nNum1 * nNum2 + "\n";  // Traditional method
8.      sDisplay += String.format("Product of %d x %d = %d%n",
9.       nNum1, nNum2, nNum1 * nNum2);
10.     sDisplay += String.format("Can be used to fix digits: "
11.      + "%3d / %3d = %5.3f%n", nNum1, nNum2, (double)nNum1 / nNum2);
12.     JOptionPane.showMessageDialog(null, sDisplay);
13.   }
14. }
```


The Character Class

java.lang.Character	
+Character(value: char)	Constructs a character object with char value
+charValue(): char	Returns the char value from this object
+compareTo(anotherCharacter: Character): int	Compares this character with another
+equals(anotherCharacter: Character): boolean	Returns true if this character equals to another
+isDigit(ch: char): boolean	Returns true if the specified character is a digit
+isLetter(ch: char): boolean	Returns true if the specified character is a letter
+isLetterOrDigit(ch: char): boolean	Returns true if the character is a letter or a digit
+isLowerCase(ch: char): boolean	Returns true if the character is a lowercase letter
+isUpperCase(ch: char): boolean	Returns true if the character is an uppercase letter
+toLowerCase(ch: char): char	Returns the lowercase of the specified character
+toUpperCase(ch: char): char	Returns the uppercase of the specified character

```
Character charObject = new Character('b');
charObject.compareTo(new Character('a')); // returns 1
charObject.compareTo(new Character('b')); // returns 0
charObject.compareTo(new Character('c')); // returns -1
charObject.compareTo(new Character('d');  // returns –2
charObject.equals(new Character('b')) ;       // returns true
charObject.equals(new Character('d')) ;       // returns false
```

StringBuilder and StringBuffer

- The `StringBuilder` and `StringBuffer` classes
  - Alternative to the `String` class located in java.lang package
  - In general can be used wherever a string is used
  - Both are more flexible than String
    - String object is fixed once the string is created
    - With both StringBuilder and StringBuffer not fixed
      - Add new contents into string
      - Insert new contents into string
      - Append new contents into string
- StringBuilder
  - Most efficient for single task access with many appends
  - Not thread safe when concurrent thread access needed
- StringBuffer
  - Thread safe Allows concurrent access for multiple tasks
  - Almost same methods as StringBuilder

StringBuilder Class Methods

java.lang.StringBuilder	
+StringBuilder()	Constructs an empty string builder with capacity 16.
+StringBuilder(capacity: int)	Constructs a string builder with the specified capacity.
+StringBuilder(s: String)	Constructs a string builder with the specified string.
+append(data: char[]): StringBuilder	Appends a char array into this string builder.
+append(data: char[], offset: int, len: int): StringBuilder	Appends a subarray in data into this string builder.
+append(v: *aPrimitiveType*): 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): 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, len: int): StringBuilder	Inserts a subarray of the data in the array to the builder at the specified index.
+insert(offset: int, data: char[]): StringBuilder	Inserts data into this builder at the position offset.
+insert(offset: int, b: *aPrimitiveType*): 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: 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.

The File Class and Text I/O

- **File class**
  - The filename is a string
  - The File class is a wrapper for *filename* and *path*
- **File object**
  - Encapsulates properties of a file or a path
  - Does not contain methods for reading/writing data from/to file
- **I/O requires creating objects of appropriate I/O class**
  - Objects contain methods for reading/writing data from/to a file
  - PrintWriter class to write strings and numeric data to text file
  - Scanner class to read strings and numeric data from text file

Writing Data Using PrintWriter

java.io.PrintWriter	
+PrintWriter(filename: String)	Creates a PrintWriter for the specified file.
+print(s: String): void	Writes a string.
+print(c: char): void	Writes a character.
+print(cArray: char[]): void	Writes an array of character.
+print(i: int): void	Writes an int value.
+print(l: long): void	Writes a long value.
+print(f: float): void	Writes a float value.
+print(d: double): void	Writes a double value.
+print(b: boolean): void	Writes a boolean value.
Also contains the overloaded println methods.	A println method acts like a print method; additionally it prints a line separator. The line separator string is defined by the system. It is \r\n on Windows and \n on Unix.
Also contains the overloaded printf methods.	The printf method was introduced in §3.6, "Formatting Console Output and Strings."

Saving Data to a new Log File

```
1.  import javax.swing.*;
2.  import java.io.*;
3.  public class WriteLogFile {
4.    public static void main(String[] args) throws Exception
5.    {
6.      {
7.        JFileChooser outFile = new JFileChooser();
8.        outFile.setSelectedFile(new File("logFile_YourName.txt"));
9.        outFile.setCurrentDirectory(new File(".") /* null is default */);
10.       if(outFile.showSaveDialog(null) == JFileChooser.APPROVE_OPTION)
11.       {
12.         File logFile = outFile.getSelectedFile();
13.         PrintWriter logOutput = new PrintWriter(logFile);
14.         String sEntry = JOptionPane.showInputDialog(null, "Enter the Data");
15.         logOutput.println(sEntry + " <-- This is what you typed");
16.         logOutput.println("\n4 x 7 = " + 4*7);
17.         logOutput.print("My name is: ");
18.         logOutput.println("Bob Laurie");
19.         logOutput.printf("\nPI = %6.3f Rounded\nDone", Math.PI);
20.         JOptionPane.showMessageDialog(null, "Done and check file");
21.         logOutput.close();
22.       }
23.       else
24.         System.exit(0);
25.     }
26.   }
27. }
```

```
This is a test of the log file which can be overwritten not appended <-- This is what you typed
4 x 7 = 28
My name is: Bob Laurie
PI =  3.142 RoundedDone
```


Appending to an existing Log File

Using FileWriter

```
1.  import javax.swing.*;
2.  import java.io.*;
3.  import java.util.*;
4.  public class AppendLogFile {
5.    public static void main(String[] args) throws Exception
6.    {
7.      {
8.        Date dateToday = new Date();
9.        JFileChooser outFile = new JFileChooser();
10.       outFile.setSelectedFile(new File("AppendLogFile_YourName.txt"));
11.       outFile.setCurrentDirectory(new File(".") /* null is default */);
12.       if(outFile.showSaveDialog(null) == JFileChooser.APPROVE_OPTION)
13.       {
14.         File logFile = outFile.getSelectedFile();
15.         FileWriter logOutput = new FileWriter(logFile, true);// Just add true
16.         String sEntry = JOptionPane.showInputDialog(null, "What is your name");
17.         logOutput.write(sEntry + " type this text.\n");
18.         logOutput.write("Time Stamp: " + dateToday.toString() + "\n\n");
19.         JOptionPane.showMessageDialog(null, "Done and check file");
20.         logOutput.close();
21.       }
22.       else
23.         System.exit(0);
24.     }
25.   }
26. }
```

```
Bob Laurie type this text.
Time Stamp: Sun Apr 29 04:38:28 JST 2012

Nikola Tesla type this text.
Time Stamp: Sun Apr 29 04:41:20 JST 2012
```


Reading Data Using Scanner

java.util.Scanner	
+Scanner(source: File)	Creates a Scanner that produces values scanned from the specified file.
+Scanner(source: String)	Creates a Scanner that produces values scanned from the specified string.
+close()	Closes this scanner.
+hasNext(): boolean	Returns true if this scanner has another token in its input.
+next(): String	Returns next token as a string.
+nextByte(): byte	Returns next token as a byte.
+nextShort(): short	Returns next token as a short.
+nextInt(): int	Returns next token as an int.
+nextLong(): long	Returns next token as a long.
+nextFloat(): float	Returns next token as a float.
+nextDouble(): double	Returns next token as a double.
+useDelimiter(pattern: String): Scanner	Sets this scanner's delimiting pattern.

Scanner Class for Reading Text File

```
1.  import javax.swing.*;
2.  import java.io.*;
3.  import java.util.*;
4.  public class ReadProcessFile {
5.    public static void main(String[] args) throws Exception {
6.      String sFirstName, sLastName, sHighFirstName = "", sHighLastName = "";
7.      int nScore, nCount = 0, nHighScore = 0, nLowScore = 100;
8.      JFileChooser inFile = new JFileChooser();
9.      inFile.setSelectedFile(new File("TestResults.txt"));
10.     inFile.setCurrentDirectory(new File(".") /* null is default */);
11.     if(inFile.showOpenDialog(null) == JFileChooser.APPROVE_OPTION) {
12.       File dataFile = inFile.getSelectedFile();
13.       Scanner inputFile = new Scanner(dataFile);
14.       while(inputFile.hasNext()) {
15.         nScore = inputFile.nextInt();
16.         sFirstName = inputFile.next();
17.         sLastName = inputFile.next();
18.         if(nScore > nHighScore)  {
19.           nHighScore = nScore;
20.           sHighFirstName = sFirstName;
21.           sHighLastName = sLastName;
22.         }
23.         if(nScore < nLowScore)
24.           nLowScore = nScore;
25.         nCount++;
26.       }
27.       inputFile.close();
28.     }
29.     else
30.       System.exit(0);
31.     JOptionPane.showMessageDialog(null, "High Score = " + nHighScore + " by "
32.       + sHighFirstName + " " + sHighLastName + "\nLow Score = " + nLowScore );
33.   }
34. }
```

```
84 Jesse Ventura
72 Dustin Hoffman
100 Robert Laurie
94 Yuri Gregeron
78 David Caradine
```

Text File Write and UTF-8

- Line breaks may not show if viewing files in Windows Notepad
  - Instead use windows default newline on file writes "\r\n"
  - Another option is to use String.format("%n Line");
  - Create new String constant accessing system property for newline
    `final String NL = System.getProperty("line.separator");`
  - When creating Strings use the NL constant within string for newline
    `sbFileWrite.append(NL + "  The following hand was dealt:" );`
- Binary file formatter is best for unicode UTF-8
  - Best to assemble in a StringBuilder or StringBuffer
  - After string is fully assembled then write to file as binary data
  - Use wrapper OutputStreamWriter for FileOutputStream
  - FileOutputStream has UTF-8 encoding specification


```
1.   File logFile = null;      // Reference variable to filename
2.   JFileChooser outFile;     // Reference variable to file chooser
3.   OutputStreamWriter logOutput = null; // Reference to file writer object
4.  ---------------------------------------------------------------------
5.   outFile = new JFileChooser();
6.   outFile.setSelectedFile(new File("PokerLog_YourName.txt"));
7.   outFile.setCurrentDirectory(new File(".") /* null is default */);
8.   if(outFile.showSaveDialog(null) == JFileChooser.APPROVE_OPTION)
9.   {
10.     logFile = outFile.getSelectedFile();
11.     logOutput = new OutputStreamWriter(
12.       new FileOutputStream(logFile),"UTF-8");
13.  }
14.  else
15.     System.exit(0);
16. ---------------------------------------------------------------------
17.      JOptionPane.showMessageDialog(null, sbDisplay);
18.      if(bLogFile)
19.        logOutput.write(sbFileWrite + "\n"); // Write to file
20.    }
21.    JOptionPane.showMessageDialog(null,
22.      "Please check file log for results:\n" + logFile);
23.    if(bLogFile)
24.      logOutput.close(); // Close log file
25.  }
```