More on Strings & Intro to Input/Output

Strings

- A string is a sequence of characters
- Strings are objects of the `String` class
- String constants: `"Hello, World!"`
- String variables:
```
String message = "Hello, World!";
```
- String length:
```
int n = message.length();
```
- Empty string: `""`

Concatenation

- Use the + operator:
```
String name = "Dave";
String message = "Hello, " + name;
   // message is "Hello, Dave"
```
- If one of the arguments of the + operator is a string, the other is converted to a string
```
String a = "Agent";
int n = 7;
String bond = a + n; // bond is Agent7
```

Concatenation in Print Statements

- Useful to reduce the number of `System.out.print` instructions
```
System.out.print("The total is ");
System.out.println(total);
```
versus
```
System.out.println("The total is " + total);
```

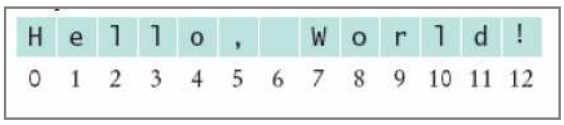
Converting between Strings and Numbers

- Convert to number:
```
String str = "12";
int n = Integer.parseInt(str);

String str2 = "52.5";
double x = Double.parseDouble(str2);
```
- Convert to string:
```
int n = 5;
String str = "" + n;
str = Integer.toString(n);
```

Substrings

-
```
String greeting = "Hello, World!";
String sub = greeting.substring(0, 5); // sub is "Hello"
```
- Supply start and "past the end" position
- First position is at 0



Reading Input

- `System.in` has minimal set of features–it can only read one byte at a time
- In Java 5.0, `Scanner` class was added to read keyboard input in a convenient manner
-
```
Scanner in = new Scanner(System.in);
System.out.print("Enter quantity: ");
int quantity = in.nextInt();
```
- `nextDouble` reads a double
- `nextLine` reads a line (until user hits Enter)
- `nextWord` reads a word (until any white space)

File `InputTester.java`

```
01: import java.util.Scanner;
02:
03: /**
04:    This class tests console input.
05: */
06: public class InputTester
07: {
08:    public static void main(String[] args)
09:    {
10:       Scanner in = new Scanner(System.in);
11:
12:       CashRegister register = new CashRegister();
13:
14:       System.out.print("Enter price: ");
15:       double price = in.nextDouble();
16:       register.recordPurchase(price);
17:
```

File `InputTester.java`

```
18:       System.out.print("Enter dollars: ");
19:       int dollars = in.nextInt();
20:       System.out.print("Enter quarters: ");
21:       int quarters = in.nextInt();
22:       System.out.print("Enter dimes: ");
23:       int dimes = in.nextInt();
24:       System.out.print("Enter nickels: ");
25:       int nickels = in.nextInt();
26:       System.out.print("Enter pennies: ");
27:       int pennies = in.nextInt();
28:       register.enterPayment(dollars, quarters, dimes,
             nickels, pennies);
29:
30:       System.out.print("Your change is ");
31:       System.out.println(register.giveChange());
32:    }
33: }
```

File `InputTester.java`

Output

```
Enter price: 7.55
Enter dollars: 10
Enter quarters: 2
Enter dimes: 1
Enter nickels: 0
Enter pennies: 0
Your change is 3.05
```

Reading Input from a Dialog Box

Figure 8:
An Input Dialog Box

Reading Input From a Dialog Box

- `String input = JOptionPane.showInputDialog(prompt)`
- Convert strings to numbers if necessary:
  `int count = Integer.parseInt(input);`
- Conversion throws an exception if user doesn't supply a number–see chapter 15
- Add `System.exit(0)` to the `main` method of any program that uses `JOptionPane`

Formatting Output

- Note: **Be sure to look at "Advanced Topic 4.4" and "Advanced Topic 4.6" in book.**
- Random Complaint about this Textbook:
  - Lists among chapter goals a topic that it hides within "advanced topic" sections.
  - Should be a subsection of this chapter.

Formatting Output: Escape Sequences

- Suppose you want the output:
  Hello, "World"!
- System.out.println("Hello, "World"!"); won't work… any ideas why?
- Instead, you need:
  - System.out.println("Hello, \"World\"!");
  - \" is an escape sequence indicating the " character
- The backslash \ within a string indicates a sequence representing a special character

Formatting Output: Escape Sequences

- The backslash \ within a string indicates a sequence representing a special character
- \\ is the escape sequence if you really want a \
  - For example:
    System.out.println("The file is located in C:\\CSIS2101\\");
  - Prints
    The file is located in C:\CSIS2101\

Other common escape sequences

- New line: \n
  System.out.print("*\n**\n***\n");
- prints
  *
  **
  ***

Other common escape sequences

- Tab: \t
  System.out.println("The following letters are tab separated:\ta\tb\tc");
- prints
  The following letters are tab separated: a b c
- Unicode characters: \u followed by its Unicode encoding
  - System.out.println("San Jos\u00E9");
  - **prints** San José.
  - See Appendix B of book for Unicode encodings

International Alphabets

Figure 5:
A German Keyboard

International Alphabets

Figure 6:
The Thai Alphabet

International Alphabets

CLASSIC SOUPS

		Sm.	Lg.
57.	House Chicken Soup (Chicken, Celery, Potato, Onion, Carrot)	1.50	2.75
58.	Chicken Rice Soup	1.85	3.25
59.	Chicken Noodle Soup	1.85	3.25
60.	Cantonese Wonton Soup	1.50	2.75
61.	Tomato Clear Egg Drop Soup	1.65	2.95
62.	Regular Wonton Soup	1.10	2.10
63.	Hot & Sour Soup	1.10	2.10
64.	Egg Drop Soup	1.10	2.10
65.	Egg Drop Wonton Mix	1.10	2.10
66.	Tofu Vegetable Soup	NA	3.50
67.	Chicken Corn Cream Soup	NA	3.50
68.	Crab Meat Corn Cream Soup	NA	3.50
69.	Seafood Soup	NA	3.50

Figure 7:
A Menu with Chinese Characters

Formatting Numbers: printf

- We've seen doubles printed with many decimal places, e.g.,
  - 434.9999999994
- What if we want to print to a specified number of decimal places or do other formatting?
- We can accomplish this using System.out.printf and *formatting specifiers*

System.out.printf

- Consider that we have:

```
double total = 3.50;
final double TAX_RATE = 8.5;
double tax = total * TAX_RATE / 100;
System.out.println("Total:\t" + total + "\nTax:\t" + tax);
```

- Prints:

```
Total:    3.5
Tax:      0.2975
```

- But we might want:

```
Total:    3.50
Tax:      0.30
```

System.out.printf

- We can accomplish this with:

```
System.out.printf("Total:\t%5.2f\nTax:\t%5.2f", total, tax);
```

- %5.2f
  - % indicates that what follows is a format specifier.
  - 5 is the desired width of the thing we're printing.
    - Width is the total number of characters to be printed
  - This is followed by a . and then the precision, 2.
  - The f ends this format specifier indicating that the number should be formatted as a *fixed floating point* number.

System.out.printf

- See tables 3 and 4 on page 138 for other formatting specifiers and flags you can use.
- Instead of f, you can use e to print in scientific notation
- Or with g, to print very large or very small numbers in scientific notation, but others as fixed floating points.