Console Input/Output

Ryan Nixon

The Console

- Think of the console as whatever tool you are using to build and run your program. It is the easiest way for the program to communicate with a user
- In NetBeans the console is the Output window. In Powershell it is the same window that you run your commands in
- The textbook simply refers to it as the Screen

System.out

- To output to the console you use a special object called System.out. This is a variable in the System class that is always defined for every program upon startup
- System.out like any other object has a number of methods available to it

println()

- `System.out.println(“Hello ” + “World”);`
- We’ve already seen println() before. This method prints the given string (or other object in string format) to the console followed by a newline character

print()

- `System.out.print(“Continue? [Y/n]: “);`
- Print works exactly like println() except that it excludes the newline character. This allows additional text to be placed on the same line before going to the next one
- This line is equivalent to the println() version of the above code, using the newline escape character: `System.out.print(“Continue? [Y/n]: \n”);`
- print() is commonly used when pulling keyboard input from the console, as it allows your program to ask the user a question and have them type the answer on the same line

printf()

- `System.out.printf("That %s costs $%6.2f.", fruit, cost);`

- Sometimes you will need to print a large number of variables to the console or have strict requirements about how they are output. printf() allows for “Formatting” your string before it is printed

- printf() uses the % sign much like strings use the \ escape character. % is a placeholder for a variable that will be inserted later. For example:
  %s - A string variable
  %f - A floating-point variable
  %d - An integer

printf()

- printf() does not specify parameters. Instead, the first parameter should be a format string containing the text you want to output. Additional parameters beyond the first designate replacements in the string *in the order they appear*.
- `System.out.printf("First param: %s. Second param: %s", first, second);`
- If the type of the variable is different than what the format string declares it to be, then Java will attempt to coerce the variable to the new type

printf()

- The format string also supports a number of options such as the *field*. The field designates how many spaces the variable will take up, regardless of what’s in the variable

- The field allows for “pretty printed” text where everything nicely lines up regardless of its length

- You define the field between the % character and the type declaration:
  %6s - A string with a field of 6 (“   hi”)
  %3d - A number with a field of 3 (“ 97”)

- Note that the field will not be enforced if the length of the variable is larger than it. Overflows are dealt with by extending the field to match:
  %3s for “Hi” will display “ Hi”
  %3s for “Hello” will display “Hello”

printf()

- The textbook mentions a few more things that printf() is capable of such as assigned decimal point length and declaring which % goes to which parameter. It’s in your best interest to look these over

- Note that many languages use string formatting differently. To name a few, the C, C++ & Java worlds generally use “%s” and “%d”, but C# and Python prefer strings such as “{!s}” or “{:f}”.

Reading From The Console

- Rather than defining variables such as the decimal geographic coordinates from the drill, it is often easier just to ask the user for input
- When dealing with user input though you should always keep in mind, “Garbage in, garbage out.” If you let the user type nonsense into the console that’s liable to be what your program produces

System.in

- System.in is a special object in the System class just like System.out
- It provides methods for reading in text from the console and placing them in variables
- Due to the variety of things that can be passed in from the console (think strings, integers, floating point numbers, dates & times, URLs, etc.) Java provides a class to help with this input

Scanner

- Scanners take a provided input (in this case System.in, an InputStream object) and assist in reading from and validating that input
- It is declared just like a String:
  ```
  Scanner keyboard;
  ```
- But initialized like a real object (remember that String generally uses “” instead):
  ```
  Scanner keyboard = new Scanner();
  ```
- You will see this *new* keyword any time that you instantiate a brand new object; we’ll cover this in a later chapter

Scanner

- When using Scanner you will also need to include this line at the top of your file, above the class { } definition: `import java.util.Scanner;`
- As Scanner is a helper utility many programs likely won't use it. To that end it is not accessible by default. The import line tells Java where the Scanner is and that your class should be able to use it.
- This is related to packages, something that will be covered in a later lecture

Scanner

- Scanners require a parameter upon initialization to tell them what input they will be scanning. Just like calling a method, you place this parameter in the parentheses:

```
Scanner keyboard = new Scanner(System.in);
```

- keyboard is now an *instance* of the Scanner class, or an object if you prefer
- You can then call all the methods that Scanners provide on System.in using this keyboard object

Scanner Methods

- To actually use the Scanner you call its methods. These methods will automatically interact with whatever the object is referencing (System.in in this case) and perform all necessary actions to validate and return what you are asking for

-
  ```
  Scanner keyboard = new Scanner(System.in);
  String line = keyboard.nextLine();
  ```

- The nextLine() method used above is probably the most common method you will use. It requests a string from the user and waits until they hit Return on the keyboard before returning all that they entered as a String.

Scanner Methods

- You can also type your input:
- `keyboard.nextInt()` returns an integer
  `keyboard.nextLong()` returns a long
  `keyboard.nextFloat()` returns a float
  etc.
- These methods require that the user type something specific. If a user attempts to write normal text (a String) but an integer is requested, Scanner will error
- We'll see how to handle these errors when we get to Exceptions later in the semester

Scanner Methods

- next() is also a useful method. It requests input from the user until a specific character (the delimiter) is typed. It then returns everything up to but not including that delimiter
- nextLine() is simply next() with the newline \n character as the delimiter
- The default delimiter is whitespace (space, tab, newline) but this can be changed by calling the useDelimiter(String) method with the first parameter whatever character you would like to separate your input with

Scanner Methods

- There are some potentially unexpected behaviors when mixing the different next() methods Scanner offers. Let’s experiment a bit to see