Introduction to Object-Oriented Programming

Basic IO

Christopher Simpkins

chris.simpkins@gatech.edu
The Java standard library provides three primary methods in the `System.out` object for sending text output to the screen.

- `System.out.print`
- `System.out.println`
- `System.out.printf` *(which just calls System.out.format)*
System.out.println takes a String parameter and sends the string to the screen. The statements

```java
System.out.print("Me");
System.out.print("ow!");
```

will produce the output

Meow!
System.out.println does the same as System.out.print but adds a newline character. The statements

```
System.out.println("Johnny");
System.out.println("Chimpo");
```

will produce the output

Johnny
Chimpo
System.out.printf

System.out.printf takes a format string and any number of additional arguments, and prints the result of inserting the additional arguments into the format string according to the format specifiers in the format string

- The format string can contain other text in addition to format specifiers
- Each format specifier begins with % and ends with a conversion character
- You can think of each format specifier as defining a field into which a value is inserted
- Like print, printf does not print a newline character at the end. End your format string with n if you want to end your output with a newline

printf is a convenience method for format
System.out.printf Examples

For full details, see http://docs.oracle.com/javase/7/docs/api/java/util/Formatter.html#syntax. Here are a few examples

- **“Decimals”** (integers) - d, Strings - s

  ```java
  System.out.printf("%d %s.\n", 7, "Samurai");
  ```
  
  prints
  
  7 Samurai.

- **Floating point numbers** - f

  ```java
  System.out.printf("I like %3.2f.\n", Math.PI);
  ```
  
  prints
  
  I like 3.14.

Play around with **ConsoleOutput.java** to get a feel for printf.
printf is useful for general formatting, but if you need to print currency amounts and you want to “internationalize” your code, use a CurrencyFormatter NumberFormat.

```java
NumberFormat us = NumberFormat.getCurrencyInstance();
System.out.println(us.format(3.14));

NumberFormat de = NumberFormat.getCurrencyInstance(Locale.GERMANY);
System.out.println(de.format(3.14));
```

prints

$3.14
3,14 €
Packages and Imports

- All Java classes are organized in packages
- We’ve been using the default package (by not specifying a package)
- To use a class from a different package, you must either fully qualify it every time you use it, or import it

**NumberFormat** is in the `java.text` package. The top of the **NumberFormat** class contains the line:

```
package java.text;
```

And **Locale** is in the `java.util` package. So for our example from the previous slide to work we must include the following import statements at the top of our source file:

```
import java.text.NumberFormat;
import java.util.Locale;
```

See **CurrencyFormatting.java**
You can read input from the console using the `Scanner` class

- First import it from the `java.util` package

```java
import java.util.Scanner;
```

- Then you can use a `Scanner` object to read, for example, three integers like this:

```java
Scanner keyboard = new Scanner(System.in);
System.out.println("Enter your 3 test scores, separated by spaces.");
exam1 = keyboard.nextInt();
exam2 = keyboard.nextInt();
exam3 = keyboard.nextInt();
examAvg = (exam1 + exam2 + exam3) / 3.0; // Why 3.0 instead of 3?
System.out.printf("Your exam average is %.1f\n", examAvg);
```
You can read from a file the same way you read from a keyboard by simply initializing with a `File` instead of `System.in`:

```java
Scanner gradeFile = new Scanner(new File("grades.txt"));
```

Scanner’s `hasNext` method tells you whether there’s more input to consume. A common idiom for reading all the lines of a text file is:

```java
Scanner fileScanner = new Scanner(new File("ScannerFun.java"));
while (fileScanner.hasNext()) {
    String line = fileScanner.nextLine();
    // do something with line
}
```

See [CourseAverage.java](#) for a more detailed example.
Look up `System.out` in the Java API documentation. What’s the type of `System`’s `out` static variable?

- `System.out` is initialized to use the program’s `stdout` file descriptor, which is the console if output hasn’t been redirected.
- We can create `PrintStreams` with other files or `OutputStreams` and write to them just like we’ve been writing to the console.

```java
PrintStream outFile = new PrintStream(new File("somefile.txt"));
outFile.println(...);
```

Stop and think about this for a moment. We can write to a text file the same way we write to a text console. What general principle in computing/programming is this an example of?
Programming Exercise

Write a program that

- reads all the lines of a file whose name is given at the command line,
- creates a new file whose file name is the original file name with “-uppercase” appended to the base name\(^1\), and
- writes all the lines of the original file to the new file but in uppercase letters.

To do this, you’ll need to look up String’s `lastIndexOf`, `substring`, and `toUpperCase` methods in the Java API.

Note: File’s constructor throws a `FileNotFoundException`. For now, deal with this by appending `throws Exception` to the signature of any method that instantiates a `File` or calls a method that does so. For example, in your solution to this exercise the main method’s signature should be:

```
public static void main(String[] args) throws Exception
```