CIS 201 Fall 2008: Lab 10
FANG Go Bye-Bye

- Non-FANG Java
- Reading a file
- Writing a file

Note to Dr. Ladd's Students: Read the text and understand it; this makes really good quiz fodder for this Friday. Just saying. Also, this lab is very easy because all the answers are in the text.

This lab, Lab 10, focuses on creating a non-FANG program. You will start by writing a program where all methods (and fields) are **`static`**. The terms *static* and *dynamic* are opposite descriptions for program attributes: static properties are values that can be completely known at *compile-time*; dynamic properties cannot be completely known until *run-time*. An easy example concerning a field in a class: the **name** and **type** of the variable are both static; in contrast, the **value** stored in that variable is dynamic (it can't be known for sure until the program is running and, even more, it can change while the program runs).

So, what does the **`static`** modifier do for a method or field? Remember that in a Java method there is an *implicit* parameter, **`this`**. The **`this`** parameter refers to the location of an object of the given type, exactly the object on which the method is being called. Implicitly, whenever you name another method in the class or a field defined in it, the compiler puts **`this`**. in front of it.

Wait a minute, that means *which* object's method or fields are meant depends on the **value** of **`this`**. But the value of a variable is not known until *run-time*; **`this`** is a dynamic value.

`static` eliminates the implicit **`this`** parameter. This means that there is only one copy of any static field. It means that the location in memory to hold the value of a static field is created at compile-time because it does not depend on **`new`**. Methods that are **`static`** do not have a **`this`** parameter which means they cannot call any method that expects such a parameter; a **`static`** method can only call other **`static`** methods and access **`static`** fields of the current class.

The nice thing about **`static`** methods is you don't need to call **`new`** to create an object of the given type. You can refer to the method with the class name and a dot. The `Math` class has many static methods such as sine and cosine:

```
double theta = 0.5;
double sin = Math.sin(theta);
double cos = Math.cos(theta);
if (1.0 == (sin * sin + cos * cos) {
  System.out.println(”Trig identities work!”);
}
```

Java starts all programs with a call to a **`static`** method; you will write a first program where all methods and fields are also **`static`** so that it is easy to use methods and data from `main`. Then you will rewrite that program two more times, each time removing **`static`** contamination until at the end you have a program, `NoFangC`, which has only one **`static`** method, `main`.

Phase 1. Create a folder for Lab10. You will be creating several programs in that folder; each new program will take as its starting point the one before it. Thus `NoFangC.java` will be copied to create `ViewTextFile.java`; after copying the file, don't forget to go in and edit the name of the class and any constructors that it has.

Question 1– In Java, what is the relationship between the name of the publicly declared class and the name of the file in which it is defined?

Phase 2. Write a class, `NoFangA`, that has a `main` method, a public **`static`** method, `run`, and two **`static`** fields. To start with, your program should look like this:

```
1  class NoFangA {
2    static private String authorA = "Grace M. Hopper";
3    static private String authorB = "Brian C. Ladd";
4
5    static public void run() {
6      // Add a countdown from 10 to 0
7      System.out.print("Program by " + authorA + " and " + authorB);
8    }
9
10   static public void main(String args[]) {
11     run();
12   }
13 }
```

Run your program. You will not see the standard FANG game come up. Instead you will see all of the output in the Emacs run pane.
You should now modify `run`, replacing the comment with code to write a countdown from 10 to 0 (inclusive; how many different values are you going to print?) on standard output. Thus the output of the above program with the loop in place would be:

```
10
9
8
7
6
5
4
3
2
1
0
Program by Grace M. Hopper and Brian C. Ladd
```

Checkpoint 1:: Show your running program (and code) to one of the lab instructors.

Phase 3. Copy `FangA.java` over to `FangB.java`; edit `FangB` so that it compiles.
Note from the discussion at the top of the lab: you should copy `NoFangA.java` to `NoFangB.java`. This uses the `cp` command at the command prompt:

```
~/CS1/Lab10% cp NoFangA.java NoFangB.java
```

Modify names so that the new program will compile.
Now remove **`static`** from all fields and methods *except* `main`. If you try to compile now you will get an error:

```
NoFangB.java:11: non-static method run() cannot be referenced from a static context
  run();
    ^
1 error
```

`run` is no longer **`static`** so it needs a value for **`this`**. How does **`this`** get a value? It refers to whatever is to the *left* of the dot when the method was called. So, if we had a reference to a `NoFangB` object, we could call `run` *on that object*. So line 11 goes to

```
11 someNoFangB.run();
```

But what type is `someNoFangB` and when was **`new`** called to give it a value? Type is `NoFangB` and it is not yet declared or initialized. Insert a new line 11 that does both: declares and initializes a local variable:

```
11 NoFangB someNoFangB = new NoFangB();
12 someNoFangB.run();
```

Now compile and run your program. It should run just as the previous phase's program ran. You have successfully transitioned from a **`static`** method where Java started your program to a non-static object method to do the real work.

Checkpoint 2:: Show your working program to a lab instructor.

Phase 4. Now you will move the initialization of the fields into a user-provided constructor.

Copy `NoFangB.java` over to `NoFangC.java`. Fix up the name of the class and get it to compile. (Notice: If you raise your hand and ask for help before the newly named program compiles, the instructors will not be able to help you.)

If you look at the code in `NoFangC`, you will notice that there is no constructor defined in it. Yet in `main`, right on line 11, you call a constructor. How is that possible?

The Java compiler creates an implicit *default* constructor for any class that defines no constructors. A default constructor is one that takes no parameters. The compiler does no special initialization in the created constructor.

We want to separate the declaration and initialization of the two author fields. The declaration will remain where it is and the initialization will move into a newly defined constructor:

```
2   private String authorA;
3   private String authorB;
4
5   public NoFangC() {
6     authorA = "Grace M. Hopper";
7     authorB = "Brian C. Ladd";
8   }
```

You have provided a constructor (any constructor) so Java will not define a default constructor for you. This version of the program looks a lot like our normal `Game` based programs: a method which is called once to set everything up (`setup` or constructor) and then another method called to make the program work (`advance` or `run`). Note that `run`, in this program, is only called once (rather than inside of a loop).

Checkpoint 3:: Show your running `NoFangC` program to one of the lab instructors.

Phase 5. Starting with `NoFangC.java`, write a program that prompts the user for the name of an input text file and then copies the lines of the file to standard output.

Copy `NoFangC.java` to `ViewTextFile.java`. Remove the fields and the body of the constructor and `run` methods. The resulting file,

```
1 class ViewTextFile {
2   public ViewTextFile() {
3   }
4
5   public void run() {
```

```
  }

  static public void main(String args[]) {
    ViewTextFile viewTextFile = new ViewTextFile();
    viewTextFile.run();
  }
}
```

can serve as a template for non-FANG programs. All you need to do to start a program is change the name of the type (and the variable) to match the file name. You might want to save a copy of this off for starting future programs.

You will be using several types, `Scanner` and `FileInputStream`, which require you to **`import`** code from the standard Java library. You can tell Java defined classes from FANG defined classes by the absence of `fang` in the package name being imported.

Define a `Scanner` field called `fin`. It is a field so that it can be initialized in one method and used in another.

In the `ViewTextFile` constructor, prompt the user for the name of a file to view until they enter one that can be opened for input successfully. Using delegation (defining useful, well-named, single-purpose methods), this should be easy to write:

```
// the constructor
public ViewTextFile() {
  fin = null; // initialize the field
  while (fin == null) {
    String fname = getFName("File to view: ");
    fin = openFileForInput(fname);
    if (fin == null) {
      System.out.println("You must specify a file that can be read.");
    }
  }
}
```

(`openFileForInput` comes from `RegularArraysWithConnectScannerCode.java` in `Week12` in Dr. Ladd's web space (`http://cs.potsdam.edu/faculty/laddbc/Teaching`). `getFName` is the version you wrote for the last lab.)

Question 2– In which method (constructor, `getFName` or `openFileForInput`) would you put a call to `ensureExtension` to make sure the file name had a particular extension on it?

`run` should read `fin` (initialized in the constructor) line-by-line (if you're reading all of a file, what should you be thinking? An *eof-controlled loop*; or a `hasNext...` loop). The body of the loop should print the current line to standard output.

Thus, if the file specified were `x.txt` and the contents of that file were:

```
This is a test of the emergency broadcast system.
This is only a test. Had this been a real emergency,
you would have been directed to a local CONLRAD
radio station for additional instructions. This concludes
the Federally mandated testing of the Emergency Broadcast
System.
```

the exact same thing (line for line) would appear in the run pain of Emacs when you ran the program. The whole output would look something like:

```
File to view: x.txt
This is a test of the emergency broadcast system.
```

```
This is only a test. Had this been a real emergency,
you would have been directed to a local CONLRAD
radio station for additional instructions. This concludes
the Federally mandated testing of the Emergency Broadcast
System.
```

Question 3– Why didn't you need to change `main` (except for the type of the object instantiated)?
Checkpoint 4:: Show your working `ViewTextFile` program to one of the lab instructors.

Phase 6. Modify the viewing program into a program which prompts the user for an input file *and* an output file and copies the text in one to the other.
Copy `ViewTextFile.java` to `CopyTextFile.java`. Make the new program compile.
Modify the program so that in the constructor in addition to the code given above, you also prompt the user for the name of a file to which the text file should be copied (you can reuse `getFName`) and attach a `PrintStream` to it with a method like `openFileForInput`. In fact, you should have your new method return a `PrintStream` attached to the given file or **`null`** if there is a problem. That sounds a lot like `openFileForInput`. There is a version of the `PrintStream` constructor that takes a `OutputFileStream`, parallel to the `Scanner` constructor that takes a `InputFileStream`. It, too, can throw an exception (and you must catch the exception so that it compiles).

> **Warning:** When you provide the program with an output file name, as soon as the `OutputFileStream` attached to it is instantiated, the contents of the file (if any) are erased. Thus if you have files you want to keep, make sure you don't specify their names as output files. Java gives no warning.

After modifying the constructor to set the `Scanner` and `PrintStream` fields (they are set in one method and used in another; this means they are state or attributes of the object as a whole), modify `run` so that instead of writing the file contents on the screen, your program writes the contents of the input file into the output file.
Question 4– **Look up** `System` in the Java docs. What are the type of the `in` field and the `out` field. Why does `openFileForOutput` return a `PrintStream`?
Checkpoint 5:: Show one of the lab instructors your working copy program.
The focus of this program is to practice reading files and learn to write files. This is not as "fun" as some of our assignments; to make up for that, the next lab assignment will focus on encrypting a file so that you can send secret messages to your friends and no one else[1] can read it.

Phase 7. Upload your Java and text files.
Make sure both authors' names are in all of the header comments! Make sure you answered all 4 questions in the header comments of the three files you are uploading: `NoFangC.java`, `ViewTextFile.java`, and `CopyTextFile.java`
Go to Moodle; you will find an assignment in this week **Lab 10**. Go there and upload all of the Java files you created/modified in lab today. Note that there will be other files in your `Lab10` directory. You want to make sure you upload just the `.java` file.
Also make sure that both partners have copies of the program. This program has some bearing on the current assignment so you probably want to have this code around. Comments will also be very helpful in this program for exactly that reason.

[1] for some sufficiently restricted definition of "no one else"