BETA

Lesson

Week 2 - Day 9 - GitHub, File Handling and Threads

Path: Back End Engineering: Java and Clojure
Unit: Week 2

Week 1 - Day 9 - Threads and File Handling

- Plan for the day
  - Review Homework
  - Submitting homework with Git and GitHub
  - Interface examples
  - File I/O
  - Threads
  - Assignment

Review Homework

Q&A

Submitting Homework with Git and GitHub

We're going to start using full GitHub repos instead of using Gist.

Prep' steps

.gitignore

Before we initialize a repo (any repo), we first need to make sure that we have a `.gitignore` file in whatever folder we want to have our repo in.

Note: we want the `.gitignore` file in the repo folder *before* we initialize the repo because it takes several steps to tell git to start ignoring files that it has already started tracking.

Copy the `.gitignore` file from the `tiy-live-bank` example into your folder for yesterday's assignment.

GitHub Repo

As we discussed before, you will have a local repo on your computer and a corresponding repo on your GitHub account. Let's start our GitHub repo first, because that will give us the URL we need for our local repo and will also give us the instructions we need to link the local repo to it.

- Go to http://www.github.com and make sure you are signed in
- Start a new repository (name it something along the lines of day7-assignment)

(Note: make sure you don't initialize your repo with anything)

You should now have a screen from GitHub that shows you your new repo and instructions on how to link it to your local repo

Local Repo

You can start a local repo (short for repository) on your computer at any time and in any folder by running:

```
git init
```

However, like everything else, we want to be organized and thoughtful about where we have repos and what the structure of the repos is.

In our case, we're still going to have a folder inside our sandbox for each assignment (day1, day2, ...) You should run the `git init` command inside that folder for each assignment. That will create a repo specific to that assignment.

Once a repo is initialized (and at any time after it's initialized), you can also ask git to give you a status of that repository:

```
git status
```

That should list out the files in your repo that have not been added to your repo yet. Note that it does not list any of the files that match the patterns in the .gitignore file.

Now we're ready to add all the files to our repo:

```
git add .
```

Now that the files have been added, we need to commit them to our local repo:

```
git commit -m "Initial Commit"
```

Run another git status to see that your local repo is clean

We are now ready to link our local repo to our GitHub repo that we created before (replace the link for my repo below with the link to your repo)

```
git remote add origin https://github.com/dbashizi/tiy-live-bank.git
```

Now we need to push our local `master` branch to the `origin` repo we just linked:

```
git push -u origin master
```

Note: once you've done that once, you'll be able to push subsequent changes from your local repo to the origin with a simple `git push`

Summary of steps:

- Decide where your project's "base folder" will be
- In that base folder, create (or copy) a `.gitignore` file
- Create your repo on GitHub
- Init your local repo

```
git init
```

- Add all your files to your local repo:

```
git add .
```

- Commit your changes:

```
git commit -m "Initial Commit"
```

- Link your local repo to your GitHub repo:

```
git remote add origin https://github.com/dbashizi/tiy-live-bank.git
```

- Push your local changes to GitHub:

```
git push -u origin master
```

Interface Examples

There are some awesome interface examples from the Java APIs. Here are just a couple:

- Comparable: https://docs.oracle.com/javase/8/docs/api/java/lang/Comparable.html
- Note: the Comparable interface is what makes things sortable in Java: https://docs.oracle.com/javase/tutorial/collections/interfaces/order.html
- Runnable: https://docs.oracle.com/javase/8/docs/api/java/lang/Runnable.html

File I/O 

Java has built-in capabilities to write to file. We'll use a simple example to write to file.

- Just like using another set of classes (like Scanner)
- Sample code to write to file:

```
public static void saveBank(Bank bank) {
    try {
        File bankFile = new File("bank.txt");
        FileWriter bankWriter = new FileWriter(bankFile);
        bankWriter.write("bank.name=" + bank.bankName + "\n");
        bankWriter.write("bank.totalBalance=" + bank.getTotalMoneyAtTheBank() + "\n");
        bankWriter.close();
    } catch (Exception exception) {
        System.out.println("Exception while writing to file ...");
    }
}
```

- Sample code to read from file:

```
public static Bank readBank() {
    try {
        File bankFile = new File("bank.txt");
        Scanner fileScanner = new Scanner(bankFile);
        String bankName = null;
        String bankBalanceStr = null;
        while (fileScanner.hasNext()) {
            String currentLine = fileScanner.nextLine();
            if (currentLine.startsWith("bank.name")) {
                bankName = currentLine.split("=")[1];
            }
            if (currentLine.startsWith("bank.totalBalance")) {
                bankBalanceStr = currentLine.split("=")[1];
            }
        }
        if (bankName != null) {
            Bank savedBank = new Bank(bankName);
            return savedBank;
        }
    } catch (Exception exception) {
        exception.printStackTrace();
    }

    return null;
}
```

Threads

In Java, a specific program runs inside of something called a "Thread". When you run instructions in your Java code, they run sequentially **and** they run one after the other, in the current (default) thread. This is what we've been referring to as "blocking calls".

Java supports "non blocking calls", aka "asynchronous" calls. It does so by allowing you, the programmer, to create a new Thread and executing your code in that Thread instead of the default Thread that was assigned to you when your main() started ...

This sounds a little tricky and complicated, so let's go through an example.

Simple Threading Example

In your tiy.networking package, create a class called ThreadRunner that will be responsible for running our threads:

```
public class ThreadRunner {
    public static void main(String[] args) {
        System.out.println("ThreadRunner running");

        int numThreadsStarted = 0;
        DecimalFormat timerFormatter = new DecimalFormat("###,###");

        long startMillis = Instant.now().toEpochMilli();

        while (true) {
            System.out.println("Number of threads started = " + numThreadsStarted);
            SampleThread localThread = new SampleThread();
            localThread.run();
            numThreadsStarted++;
            if (numThreadsStarted > 10) {
                break;
```

```
            }
        }

        long endMillis = Instant.now().toEpochMilli();

        System.out.println("Ran in " + timerFormatter.format(endMillis - startMillis) + " ms");

        System.out.println("ThreadRunner done!");
    }
}
```

Now create the SampleThread class (in the same package):

```
public class SampleThread implements Runnable {

    public void run() {
        System.out.println("Running " + Thread.currentThread().getId());

        try {
            Thread.sleep(2000);
        } catch (Exception exception) {
            exception.printStackTrace();
        }

        System.out.println("Done running " + Thread.currentThread().getId());
    }
}
```

Notice something different about the SampleThread class:

- It implements the "Runnable" interface -> that tells Java that this class will have a method called run() that will be used to start a new thread
- The code we want it to run is inside of the run() method. You can think of the run() method as the static void main() of a "threadable" class

Now run your ThreadRunner, and you'll see that it takes a while to run. That's because every call is sequential and we're making the SampleThread run() method pause for 2 seconds (with the Thread.sleep(2000) call) every time it executes.

So, even though we're using a class that implements Runnable, and we're calling its run() method, we're still executing these calls in a blocking manner. Why?

A runnable class needs a thread to run on

When you call the run() method of a runnable class directly, you're still making a blocking call. In order to make it into an asynchronous call, you need to create a new thread and tell it which runnable object it should use:

```
            Thread newThread = new Thread(localThread);
            newThread.start();
```

You will use the code above instead of the direct call to the run() method.

Notice a couple of things:

- We are not calling the run() method directly
- Instead, we are creating a new Thread object, and we're passing it our object of type SampleThread

- The Thread class is then going to take care of starting a new Thread and making it execute the code inside of our run() method when the thread starts.
- The call to newThread.start() is when the new thread starts

If you execute this new code, you will see that the overall execution of the program is much quicker. That's the power of multi-threading ...

Continue to next Assignment
Assignment

Week 1 - Day 4 - Car Garage

Car Garage

Navigate Path
Path
Back End Engineering: Java and Clojure
Prework

- Prework
- Install Party
- Learning how to type

Week 1

- Week 1 - Day 1 - Binaries and Ascii
- Week 1 - Day 2 - Java keywords, variables and methods
- Week 1 - Day 3 - User Input
- Week 1 - Day 4 - Static fields and inheritance

Week 2

- Week 2 - Day 6 - IntelliJ, Java Date/Time and Dynamic Data Structure
- Week 2 - Day 7 - Interfaces, Abstract Classes, Version Control (Git and GitHub)
- Week 2 - Day 8 - Interfaces and Abstract Classes
- Week 2 - Day 9 - GitHub, File Handling and Threads

Assignments

- Week 1 - Day 1 - Drawing with Ascii Characters
- Week 1 - Day 2 - Classes from your life
- Week 1 - Day 3 - Guess the number game
- Week 1 - Day 4 - Car Garage
- Week 2 - Day 6 - Banking System
- Week 2 - Day 7 - String Handling
- Week 2 - Day 8 - Hospital with Interfaces

- Sign Out