CS 235 - Week 14 Lab Exercise - 2021-12-03

Deadline

Due by the end of lab on 2021-12-03.

How to submit

Submit your `.java` and `.jar` files for this lab on https://canvas.humboldt.edu

Purpose

To practice creating a package, creating a jar, and creating an executable jar.

Important notes

- IF you are attending the lab via Zoom, you are expected to pair program in a breakout room (possibly trio-program if necessary based on class members' Internet and the number of class members attending via Zoom).
  - In this case, **be sure to TYPE BOTH (all) OF YOUR NAMES** in the beginning comment of EACH of your `.java` files

  But, because of the delta variant surge, if you are attending lab in person in BSS 317, you will each work on a separate computer, although discussion amongst those attending will be encouraged!

Some initial set-up for today's lab exercise

- Go to the CS50 IDE.
- Create a folder named `classes`.

Creating your own package

- You'll try creating a new package relative to this `classes` directory.
- Grab a version of `GameDie.java` -- any working version should be fine for this lab exercise.
- We're going to put the `GameDie` class into a package `edu.humboldt.cs235` (or a package name including your username, if you prefer).
- For a package named `edu.humboldt.cs235`, you will need a subdirectory `edu/humboldt/cs235` to correspond to this package; we'll place this subdirectory within the `classes` directory, using the following commands:
  - FIRST: right-click on your new `classes` folder, and select "Open Terminal Here"
  - THEN:

```
~/classes/ $ mkdir edu          # make the new subdirectories
~/classes/ $ cd edu
```

```
~/classes/edu/ $ mkdir humboldt
~/classes/edu/ $ cd humboldt
~/classes/edu/humboldt/ $ mkdir cs235
~/classes/edu/humboldt/ $ cd cs235
~/classes/edu/humboldt/cs235 $
```

  - Now click on the `classes` folder on the right until these new subdirectories show up there, also.
  - And now put your copy of `GameDie.java` into the `cs235` directory.

- THEN, add this as the VERY FIRST LINE in this new version of `GameDie.java`:

```
package edu.humboldt.cs235;
```

  - ...and save your `GameDie.java` file.

- And compile your modified `GameDie.java`:

```
~/classes/edu/humboldt/cs235 $ javac GameDie.java
```

  - you should now have `GameDie.class` in directory `edu/humboldt/cs235`

Using a class from this new package

- You will now put a Java application using `GameDie`, `DiceRoller.java` (available along with this lab exercise handout), into another subdirectory entirely, `235lab14`:
  - FIRST: right-click on your `classes` folder again, and select "Open Terminal Here"
  - THEN:

```
~/classes/ $ mkdir 235lab14
~/classes/ $ cd 235lab14
~/classes/235lab14/ $
```

  - Now click on the `classes` folder on the right until this new subdirectory shows up there, also.
  - And now put your copy of `DiceRoller.java` into the `235lab14` directory.

- To help check if you set up the package above correctly, first perform the following action **THAT SHOULD FAIL** if all is set up properly so far (because Java should not be able to know where to find `GameDie`):
  - FIRST: right-click on your `235lab14` folder again, and select "Open Terminal Here"
  - THEN:

```
~/classes/235lab14/ $ javac DiceRoller.java    # should FAIL!!
```

- Now add the needed `import` statement into `DiceRoller.java`, based on the package in which you placed `GameDie`:
  - add this as the VERY FIRST LINE in this revised version of `DiceRoller.java`:

```
import edu.humboldt.cs235.GameDie;
```

(or, if you prefer:)

```
import edu.humboldt.cs235.*;
```

- Now you should be able to compile and run `DiceRoller`: (first using the `-classpath` option to include our `classes` directory in the classpath for this particular command):

```
~/classes/235lab14/ $
javac -classpath .:/usr/share/java/cs50.jar:/home/ubuntu/classes DiceRoller.java
~/classes/235lab14/ $
java -classpath .:/usr/share/java/cs50.jar:/home/ubuntu/classes DiceRoller a b c
```

(or whatever arguments you'd like)

- You can also change the classpath for your current Terminal session, adding our `classes` directory to it, using:

```
~/classes/235lab14/ $ export CLASSPATH=$CLASSPATH:/home/ubuntu/classes:.
```

- And now you should be able to compile and run `DiceRoller` without having to use the `-classpath` option:

```
~/classes/235lab14/ $ javac DiceRoller.java
~/classes/235lab14/ $ java DiceRoller e f g h
```

Making a simple jar

- In my experiments thus far with jars, the key for me has been to make sure that the jar reflects the package structure. There may be more flexibility possible, but this is what I've gotten to work so far... 8-) So, you are going to reproduce this.
- Using package `edu.humboldt.cs235`, you'll create a jar <yr_name>`Lab14.jar` (that is, I'd create `st10Lab14.jar`, you'd create a jar based on YOUR name or user name...) containing all of the contents in directory `edu` (the "root" of my package):
  - FIRST: right-click on your `classes` folder again, and select "Open Terminal Here"
  - THEN:
- Since this is the directory where the root of the new package lives, you'll create the jar from here, using the `jar` command:

```
~/classes/ $ jar vcf st10Lab14.jar edu
                    ^^^^ put YOUR username here!
~/classes/ $ ls *.jar                  # you should see your new jar file
~/classes/ $ jar tf st10Lab14.jar      # see the contents in your new jar file
```

Using a simple jar

- Note that there is more than one way to use a jar, also! This is just one of the possible approaches.

- IF I add a jar to my classpath, THEN classes that I write can import packages within that jar file, and they'll be found; you will now try this.
- Use the following to add this new jar to your `CLASSPATH` temporarily (just for this login session):

```
~/classes/ $ export CLASSPATH=$CLASSPATH:/home/ubuntu/classes/st10Lab14.jar
                                                              ^^^^
```

(use YOUR username for this, of course...)

- Now, perform the following experiment, temporarily "hiding" the package so we can see if the jar works

```
~/classes/ $ mv edu HIDEedu      # "hide" the package by renaming it
~/classes/ $ cd 235lab14
~/classes/235lab14/ $ rm DiceRoller.class  # we want to recompile this, and see if it works
~/classes/235lab14/ $ javac DiceRoller.java  # if this works, MUST be using the jar
~/classes/235lab14/ $ java DiceRoller a b c  # or whatever arguments you'd like
```

Submit your files `DiceRoller.java`, `GameDie.java`, and <yr_name>`Lab14.jar` .

Creating an executable jar

Now you will group some classes, once of which has a `main` method, into an executable jar, which should make for a single convenient “holder” for all the files making up a Java application, that should be executable on any system running a compatible version of the Java Runtime Environment (JRE).

- Rename the root of your package so it is no longer hidden!

  ```
  ~/classes/ $ mv HIDEedu edu
  ```

- Make a copy of your `DiceRoller.java` and `DiceRoller.class` in the `classes` directory:

  ```
  ~/classes/ $ cp 235lab14/* .
  ```

- `DiceRoller` is a Java application (it has a `main` method) – can we create an executable Jar for it?

  Let’s try to create a jar file with a manifest indicating that `DiceRoller` is the class with a `main` method, where execution should start (the `e` option and the name of the desired main class make this possible); we'll also indicate what files in this directory should be placed in the jar, here the `DiceRoller` files and the package files with root `edu`:

  ```
  ~/classes/ $ jar cvfe UseDice.jar DiceRoller DiceRoller.* edu
  ```

  – You should now have a file **`UseDice.jar`** in your directory.

- IS this indeed an executable jar? Let’s try it!

  ```
  ~/classes/ $ java -jar UseDice.jar w x y z
  ```

Submit your files `DiceRoller.java`, `GameDie.java`, <yr_name>`Lab14.jar`, and `UseDice.jar`.