CS 455 Lab 2: Using objects

Fall 2021 [Bono]

Goals and Background

This lab will give you practice working with some classes from the Java API. One aspect of working with classes is being able to read the class documentation. You will also get some practice using simple console I/O.

The first few sections below are some general information about labs that apply to all labs.

Recommended advanced preparation

Before coming to lab this week we recommend you read over the lab at least through Exercise 1, and to complete that exercise. It's a pencil and paper exercise (can put answers in the README). When you come to lab you can compare your results with your lab partner.

Organizing your labs and saving your work

Recall that the first time you acess the lab starter code on Vocareum you must go through the link on d2l under "Vocareum Assignment Links". Once you are connected to Vocareum, you may complete it in your Lab 2 workspace there. More details about this process were described in Lab 1, under "Accessing Vocareum".

Unlike with Lab 1, unless you are a DEN student you will not be submitting your labs on Vocareum, but will get it checked-off face-to-face during your lab section. (Check-off for DEN students is described at the end of each lab.)

General note on saving your work: You should save all of your completed assignments, because sometimes we will build on what we have done in one assignment in a later assignment. Also, if there are any grading issues later, you will have evidence of the work you have done. As long as you don't delete your files in Vocareum, they will still be available in the workspace for the given lab throughout the semester. We recommend doing this even for different parts of a lab when you have already gotten the previous part checked off: thus the different file names for the different parts of this lab. Make sure you put the assignment number (e.g., CS 455 Lab2), and the names and roles of the lab partnership (described further in the next section) in a comment at the top of all the files for the lab (generally source code and README).

Lab pairs

Labs will generally be done working in partnerships (2 people). For those of you who requested a lab partner in the survey, the TA will announce the pairing at the beginning of lab 2. If you were not assigned a lab partner, but want one, let one of your lab TAs know. The pair of you will sit at one workstation. One person will be the designated "driver", doing the typing and mousing to use the tools, and the other will be the scribe, writing down the answers to lab questions. In subsequent labs you will switch off roles. This does *not* mean only one person is doing the work. In *pair programming* both members of the team are active participants in solving the problem, discussing as you go, but with each with a designated role. If one of you has more programming experience, we recommend the lesser experienced of you is the driver this time.

The questions to be answered by the scribe are in bold and labeled Question 1.1, 1.2, etc. You should put your answers to those into a `README` text file that you edit, but if it's more convenient, you could, alternatively, write down the answers on paper (which is a little more awkward to share on zoom). To keep your work organized, it

may be helpful to use a lab notebook for this purpose. Wherever the lab mentions the README, we mean a text file or on-paper lab write-up.

DEN students

DEN students will generally not be working in pairs since they are working remotely and away from other students in the class. DEN students will need to electronically submit their labs, so their answers to the lab questions will go in the README file, rather than on paper. Each lab includes instructions about what DEN students need to submit, and how (for example, see the end of this lab).

Reading and reference material

This section, which will appear in every lab, gives you a guideline to what you'll need to know before doing the lab, or materials to refer to during the lab. The API Documentation link below is definitely in the latter category.

- Horstmann, Ch. 2, Using objects.
- Horstmann, Section 2.6, API Documentation. Introduction to navigating in the Java API documentation pages.
- Java 1.8 API Documentation. Here's the direct link to the `LocalDate` page.
- Horstmann, P2.10 (Programming project 2.10) Shows some examples of using `LocalDate`. (Note: Programming projects are at the end of each chapter under End-of-Chapter Exercises. Exercises that start with P2 are at the end of Chapter 2.)
- Horstmann, 4.5.2 Concatenation. How to use String concatenation to print multiple items in one `println` statement.
- Horstmann, 4.3.1 Reading Input.
- Horstmann, 5.1 The `if` statement

Exercise 1 (1 checkoff point)

Look at the examples in P2.10 from the textbook (more info above) and `LocalDate` documentation in Java API (linked above). Note: You do not need to read all the documentation on that page. Use these resources to help you answer the following questions about the `LocalDate` class.

Question 1.0 In the README file (or lab write-up), put the first and last names and the roles (driver/scribe) of each of the partners, the course name and lab number.

Question 1.1 What `import` statement is necessary to use the `LocalDate` class?

Question 1.2 In your README, write a statement to create a `LocalDate` for the date `January 20, 1995`: put the date in a variable called `myDate`.

Hint: as you can see on the Java API page, this class has no constructors. The methods to create a new instance are `static` methods of `LocalDate` whose return type is `LocalDate`. (The call to `new` is hidden inside those methods.) To call a static method you use the *class* name instead of an object name before the dot; e.g., `Math.sqrt(10)`

Question 1.3 Write down a statement (or statements) to print out `myDate`: it should be formatted as follows:

```
1/20/1995
```

(We're looking for something that would work no matter what date we started with.)

Hint: In the documentation look for methods that start with "get".

Question 1.4 Write down a statement that creates a new date 20 days later than `myDate`. Call this date `later`. (We're looking for something that would work no matter what date we started with.)

Question 1.5 Suppose, instead, you wanted to *change* `myDate` to one 20 days later, Write down a statement to accomplish this. (We're looking for something that would work no matter what the old value of the date was.)

Hint: `LocalDate` is an immutable class (like `String`).

Exercise 2 (1 checkoff point)

Read over the whole exercise before starting. Using the results of your investigations above write a program `Date.java` that creates the date January 20, 1995, prints it out, and then prints the day 20 days later. Here is what the run will look like:

```
1/20/1995
2/9/1995
```

How to copy a file from another lab. If you want to start from the Hello program we started with in the first lab, and then modify it to create this program, this is a good opportunity to try copying a file in Vocareum. (You could also do this by just cutting and pasting the contents of a file, but that becomes more unwieldy with larger files later.) Here's how you copy the file itself:

1. if you are starting from inside the Lab 2 workspace, you can get out of that by clicking "Lab 2" at the upper left. That will get you back to the list of assignments, and can switch to a different lab.
2. go into the workspace for "Lab 1". Select `Hello.java` from the list of files in your `work` folder.
3. click `Copy` (upper left).
4. go back to Lab 2. (technique explained above in step 1.)
5. click `Paste` (upper left).
6. Now you can rename the file. There are two way to do this. *Either:*
    - Select the the file in the file list. Then click the `Rename` button (upper left; you type the new name in the textbox that appears right below that button); *Or:*
    - in the Terminal window use the linux `mv` command:

      ```
      mv Hello.java Date.java
      ```

Hint: Remember that the name of the main class has to match the file prefix.

Hint2: even before you write your print statements, an easy way to test if you created the correct object is to add a debug print statement and compile and run the code before adding anything else. When you put an an object variable in a `println` statement it will print out the data in that object. In this case, Java implicitly calls a method called `toString`; look at the documentation for `toString` to see what will get printed for a `LocalDate`. Here's an example of such a call:

```
System.out.println("DEBUG: " + myDate);  // Java calls myDate.toString()
```

You can comment out your debug print statements when you are done testing.

Common compile error. One common compile error is "cannot find symbol". When you get this message pay close attention to what part of your code it is flagging and what it is complaining about. Some possible causes of this error:

- you are using an undefined variable.
- you didn't import the right class or package that has this symbol.
- you changed the name of your variable, method, or class but you are still using the old name in a few places.
- you are calling a method that doesn't exist for the class you are using (e.g., `myString.sqrt(100)`).
- you are calling a method with the wrong number or type of arguments

---

Question 2.1 Describe two compile errors that you got while developing this code and their fixes. (It's ok if both errors had the same high-level message, e.g., "cannot find symbol".)

For Exercise 2 check-off, show the lab TA your program running, your source code, and your answer to the question above.

Exercise 3 (1 checkoff point)

Make a copy of your code for Exercise 2 into a file called `Birthday.java`. Hint: practice using linux `cp` command to do this. (In contrast, cutting and pasting the contents of a file is an error-prone way to copy files, and is not recommended.) *Read over the whole exercise before beginning to write code.*

Modify `Birthday.java` so it prompts the user for their birthday, and then prints out whether their birthday has happened yet this year. If their birthday is the same date the program is run, it should report that their birthday has already happened.

Your output must look exactly like the example shown. Here is what it should look like running (user input shown in italics):

```
Enter your birth month [1..12]: 1
Enter your birth day of month: 5
Enter your birth year [4-digit year]: 1991
Your birthday has already happened this year.
```

and if their birthday hadn't happened yet, the last line of output would be:

```
Your birthday has not yet happened this year.
```

Note: the output of the program depends on what day you run the program. Your program should work correctly, without changing the code, any day you run it.

For the purposes of this lab, you do not have to error-check input. E.g., if the user inputs negative numbers, a letter, or the month 5000 at the prompt, your program does not have to handle that condition.

Hint: we recommend you look up the relevant documentation and plan out the logic involved in solving this problem before you jump into coding. You can use your lab notebook to write down your ideas.

Question 3.1 Describe two compile errors that you got and their fixes while developing this code.

Question 3.2 Devise three test cases (not the one above) for which you and the TA already know the answer, to help verify that your code is working correctly. Describe these test cases and the expected results in the README. Then try out your test cases to help verify your program works properly.

For Exercise 3 check-off, show your program running, your source code, and your answers to the question above to the lab TA. When you run the program include the test cases mentioned above in your runs.

Exercise 4 (1 checkoff point)

Back up your code for Exercise 3 to a file called `Birthday3.java`. One way to do this is with the following command (the `-p` means that the copy retains the permissions and last-changed date of the original);

```
cp -p Birthday.java Birthday3.java
```

This is useful if you mess up and want to revert to back to a subset you know works. Once you copy the file continue to work with `Birthday.java` for this part of the lab.

Keep all the old functionality of the program, but now it should also print out the current age of the user. See example below:

Your output must look exactly like the example shown. Here is what it should look like running (user input shown in italics):

```
Enter your birth month [1..12]: 9
Enter your birth day of month: 30
Enter your birth year [4-digit year]: 1991
Your birthday has not yet happened this year.
You're 29 years old.
```

Note: as in the last problem, the exact output of the program depends on what day you run the program. Your program should work correctly, unchanged, any day you run it.

For Exercise 4 check-off, show your program running and your source code to the lab TA. When you run the program include the test cases from Exercise 3 in your runs.

If you want an extra challenge

This isn't part of the lab proper, but if you have some extra time, you can also do this exercise. Change `Birthday.java` so that it prints out a different message if the day the program is run is on the user's birthday. Here is the message to print:

```
Happy Birthday!
```

Hint: for this one we definitely recommend planning out your logic before jumping into coding.

Checkoff for DEN students

When you click the `Submit` button, it will be looking for and compiling the files `README`, `Date.java`, and `Birthday.java` in your home (aka, `work` directory). The lab us due by 11:59pm on Sunday Pacific Time at the end of the week for this lab.
