This lab explores programming with sets in Java. It is due in class Friday, March 13 but earlier handins are welcome and encouraged.

While you may discuss ideas and strategies for problems with other students, you should always make the first attempt on a problem yourself and **you must write up your own solutions in your own words**. *You may not collaboratively write solutions or copy a solution that one person in the group writes up. You also may not look for, copy, or use solutions from other sources, including from generative AI like ChatGPT, even if you make changes. There's no such thing as using someone else's solution for a problem "as an example" for writing your own.*

Preliminaries

This lab involves Java programming. It is recommended that you use Eclipse.

Before starting Eclipse:

- Create a workspace directory `~/cs229/workspace` to hold your Eclipse projects.

Start Eclipse.

- Create a new Eclipse project named `sets`. Make sure the "Create module-info.java file" box is *not* checked before clicking Finish in the dialog box.
- Import the files from `/classes/cs229/sets` into your project. Make sure they go into the default project under `src` rather than into a package or the top level of the project directory.

It is recommended that you configure Eclipse as described in appendix A.

Handin

- Make sure you have replaced the `<your name here>` notations in the `@author` tag near the beginning of the files you edited.
- Make sure you have autoformatted all of the files you edited.
- Hand in your work by copying the entire project directory `~/cs229/workspace/sets` to your handin directory `/classes/cs229/handin/`*`username`* (where *username* is replaced by your username).

Exercises

BitSet

The class `BitSet` represents a set of integers from 0 to 31 (inclusive), implemented as a bit set using the 32-bit binary representation of a single `int`.

A skeleton for `BitSet` has been provided. Your task is to complete the implementation of `BitSet`'s methods as indicated by the `TODO` comments in `BitSet`.

Notes:

- The `size()` method should count the number of elements in the set. (It is more efficient to store the count as an instance variable and update it as needed, but the goal here is practice with bitwise operators.)
- While it is logically correct to traverse the set's elements by iterating through the values 0..31 and using `contains()` to see if that element is in the set, see if you can avoid both the counter and the overhead of the function call. Hint: use shift to move through the bits, though be careful not to change `bits_` itself.
- The `private` modifier limits access to a particular class, not a particular object. That means you can write `other.bits_` in the body of `union`, `intersection`, and `difference` — even though `other` is a different object, it is also a `BitSet`.
- `BitSetTest` provides a demo/partial tester for `BitSet`. It is not intended to be an exhaustive tester but it should get you started. Feel free to add to it.

ClassroomDB

The second example application uses bit flags to capture classroom attributes, supporting searching for classrooms with a particular set of attributes. Most of the functionality has been provided for you; your task is to complete the implementation as indicated by the `TODO` comments in `Room` and `ClassroomApp`.

Notes:

- The constants `PROJECTOR`, `WHITEBOARD`, etc in `Room` act as flags and should be initialized to have the binary values indicated by the associated comment. There are two ways to do this: using the decimal value of the binary string (e.g. 16 for `10000`) or by shifting (e.g. `1 << 4` for `10000`).
- Look at the public methods in `ClassroomDB` to see what you can call to add and retrieve rooms as needed for the `main` program in `ClassroomApp`. Note that `Room` has a `toString` method so if `room` is a `Room` object, you can print it with

```
System.out.println(room);
```

A Eclipse Configuration

Recommended configuration for Eclipse:

- Open the preferences configuration dialog box: From the "Window" menu in Eclipse, choose "Preferences".
- On the left side of the Preferences dialog, expand "Java".
- Specify how code should be formatted:
  - Expand the "Code Style" item under "Java", then click "Formatter".
  - On the right side of the window, click "Import...". Navigate your way to `/classes/cs229/eclipse`, highlight `cs225-formatter.xml`, and click "Open".
  - Make sure that `cs225` is shown as the "Active profile".
- Specify options for code generation:
  - Expand the "Code Style" item under "Java", then click "Code Templates".
  - On the right side of the window, click "Import...". Navigate your way to `/classes/cs229/eclipse`, highlight `cs225-codetemplates.xml`, and click "Open".
  - At the bottom of the right side of the window, click the "Automatically add comments for new methods, types, modules, packages and files" box to select it.
- Tell Eclipse about the convention of naming instance variables ending with _:
  - Choose the main "Code Style" entry under "Java".
  - On the right side of the window, click on "Fields" in the box under "Conventions for variable names:" to highlight it, then click the "Edit..." button. In the box that pops up, enter _ (an underscore) in the "Suffix list" box and click OK.
- Since we'll be using Java 17, tell Eclipse to enforce Java 17 syntax rules:
  - Choose the main "Compiler" entry under "Java".
  - Set the compiler compliance level to 17.
- Tell Eclipse to store compiled classes separately from source files:
  - Choose the main "Build Path" entry under "Java".

  - Make sure that the "Folders" option is selected under "Source and output folder", and that the source folder name is `src` and the output folder name is `bin`.

- Finally, click "Apply and Close" at the bottom of the Preferences window to apply the new settings and close the window.