Download the starter project labFile.zip from Canvas and open it in Eclipse.
Class Mountain is already finished. It allows you to create objects that represent a given mountain.
Mountain.csv is a comma separated value file that includes data about a number of mountains.

class MountainApp:

MountainApp includes the main method.

- Create a variable called **`mountainList`**. It is of type **`List<Mountain>`**
  Initialize it with a generic **`LinkedList`**
- Use **`Scanner`** to read in the mountains from file **`Mountains.csv`**
  Scanner implements the interface **`AutoClosable`**. Because of that we'll use a try with resource statement to create the instance of **`Scanner`**
  Hint1:
  For information on how to create a try with resource statement check out the first paragraphs of
  https://docs.oracle.com/javase/tutorial/essential/exceptions/tryResourceClose.html
  Hint2:
  The **`main`** method is static. Because of that you won't be able to call **`this.getClass()`**. However, you can use a class literal to access members of class Class<T>
  Notice: So far we used **`getResource`** to locate a resource. The method **`getResource`** returns a URL.
  However, Class<T> also has a method called getResourceAsStream. It returns an inputStream.
- At this point your code should look like this:

```
try(Scanner reader = new Scanner(MountainApp.class.getResourceAsStream("Mountains.csv"))){
}
```

- Inside the try block add the following functionality:
  As long as there is another line in the csv file:
  a) Read the line.
  b) Convert the **`line`** into an instance of Mountain by calling the method **`getMountain`** and passing **`line`** as an argument.
  You might have noticed that the method **`getMountain`** does not exist. Not a problem. Just call it anyhow.
  Once you see a squiggly red line roll over it and have Eclipse auto-generate the method stub for you.
  The auto-generated method header should look like this:
  `private static Mountain getMountain(String nextLine)`
  Notice that the body of **`getMountain`** hasn't been implemented yet. It just returns null.
  We'll take care of that in a moment.
  For now let's finish up the content of the main method.
  c) Check whether the value returned by **`getMountain`** is different from null.
  If that is the case (we got an actual instance of Mountain) add the Mountain to **`mountainList`**
- Still inside the main method but after the try statement we are going to display all the mountains we just read in.
  We do that by listing all the elements of **`mountainList`** - one element per line.

- Run the program - no output is produced.
  Why? Because we haven't created any Mountain objects based on the csv file data yet
- Now it is time to implement the method **`getMountain`**
  It receives a line from the csv file as parameter and creates an instance of Mountain.
  If a problem occurs and the instance of mountain cannot be created, then the line should be printed to the standard error stream (**`Standard.err`**) and the method returns null (see sample output)
  Hint:
  Use the method **`split`** *from class* **`String`** *then parse the individual components to the correct types.*
  What happens if you try to parse the String "hello" to an **`Integer`**? ……
  What happens if you try to parse the String "hello" to a **`Boolean`**? ……
  Catch the exception that might arise when parsing the String to a Mountain.
  In the catch block do the following:
  Print the String, that could not be read in as a mountain, and "`.. cound not be read in as a mountain.`" to the standard error stream
- Run the program – this time it should produce a list of mountains but no error message.
  Why not? There were no faulty entries in the csv file.
- In order to test the error handling add the following line after Mt. Olympus : 8888,Mt Everest,TRUE
  The order of the mountain information is mixed up and that will cause problems when reading in the file
- Run the program again – this time the output should look like this:

Output:

```
8888,Mt Everest,TRUE .. cound not be read in as a mountain.

Mt Everest (8850) *
Mt Olympus (2751)
Mt Rainier (4027) *
K2 (8611) *
Mauna Kea (4205)
```

- After Mt. Rainier add the following line: `Matterhorn`
- Run the program– this time the you get an `ArrayIndexOutOfBoundsException`
- In order to catch the problem, that a line might miss some information do the following:
  Add `ArrayIndexOutOfBoundsException` as a second Exception type that should be caught
  If we wanted a different exception handling we would write a second catch block.
  In our case, however, we want the exact same behavior.
  For such situations Java allows us to use the specify multiple exception types separated by a vertical bar
  https://docs.oracle.com/javase/7/docs/technotes/guides/language/catch-multiple.html
  Like this: `catch(NumberFormatException | IndexOutOfBoundsException ex) { .. }`
- Run the program again – this time both erroneous csv lines will be caught.