Generics, Exceptions and Undo Command

(Three unrelated concepts)

Why use generics?

- When you write a collection (something to store data, such as a linked list, arraylist, table, etc), you want it to be usable with all kinds of data. You want it to be usable to store Strings, or usable to store integers.
- Usually this is done by letting the collection work with the Object class, which everything inherits.
- The problem is that every time you take out an element, you have to cast it from Object. Also, while you want your collection to be usable for different types, you also want to limit the collection to using one type at a time (so you don't accidentally put a String into a list of integers).
- "Generics" are a way for us to use "placeholder" types. The user chooses what they are when they first use the code, but then we plug that type everywhere in our methods.
- Example: the add() method of ArrayList<String> takes a String as a parameter. The add() method of ArrayList<Integer> takes Integer as a parameter.
  - Inside ArrayList, it is declared as ArrayList <E>, and the add method takes a parameter of type E

A Generic Stack

- A stack is a collection that only allows us to add to the end, and only remove from the end. It turns out, imposing these limitations on a collection is sometimes useful, because it determines how we interact with the contents.
- Class AStack uses generics. In the main method, we have 2 instances of AStack, one using type Integer and one String. Notice that the type of stack is determined in instantiation, so we cannot push an Integer into a stack with type is String.
- Like normal stack, AStack supports pop, push, and peek.

What are exceptions?

- Exceptions are a way for code to "bail out" when it encounters something unexpected. Usually, this results in the program crashing and printing out a trace of where the exception occurred.
- In Java, we can write code like "throw new Exception("Some message")". "throw" is a keyword, and you have to throw something that inherits from Error or Exception (both of these implement "Throwable"). These classes are ways for us to package information about what went wrong.
  - We can write our own exception classes to specify specific errors
- When we throw an exception, this tells Java to abort what it's doing and get out of the program.
- However, we can write code that "catches" thrown exceptions. That is, it responds to an exception and prevents the program from crashing immediately. Although a lot of times, all we do when catching exceptions is log the error in some way and then crash anyway.

Exception Handling

- try-catch blocks are ways for us to handle exceptions. We tell Java that if any exceptions are thrown inside a block that we label "try", Java should try to catch them with the "catch" block that follows.
- The catch block specifies what kind of exceptions it catches. If you want to do different things in response to different exceptions, you can have several catch blocks, one after the other.
- You can see that in the driver, the push statements are in a try-catch block. There are 2 catch blocks, the first one catches FullCollectionException and the second one catches all other exceptions.

Checked and unchecked exceptions

- Most exceptions are called "checked exceptions". That means that if your method throws one, it has to declare that it throws it in its header, saying something like "throws SomeException".
- Any method that calls a method that has a throws declaration must also have the same throws declaration, or else it must handle the exception by surrounding the method call in a try-catch block.
- Exceptions that inherit from RuntimeException are called "unchecked exceptions". These are not required to be declared in the header.
- The idea is that checked exceptions are ones that a programmer thinks other programmers should be ready to deal with in some way if they want to use her/his code. Unchecked exceptions are ones that the programmer may think are unrealistic to deal with, perhaps because they result from the misuse of their code, or bad programming (NullPointerException is an unchecked exception).
- Things that inherit from Error also don't have to be declared, but by convention, these are only things that should never happen (like the results of hardware failure). Do not inherit from Error.
- This code has a user-defined exception, FullCollectionException. It will be thrown when user try to push an item into a full stack. The code that throws the exception is in method push of AStack.

Undoable Command Class

- In the past, we have used command classes to execute method calls in their own threads.
- We can also use command classes to make method calls undoable.
- Like before, we wrap the method call inside the execute/run method of a command class. However, we add an undo() method.
  - The undo() method should do the opposite of what the execution did, so it will differ depending on the method.
- Then we create a class to manage our command history. We tell this "undoer" every time we want to execute a command, and it remembers it. Then, when we want to undo the command, the undoer can do this for us.
- Look at the Push and Pop undoable commands, the HistoryUndoer, and UndoerDriver. Notice that we don't really need undoable command classes for peek() and size() since those do not change the stack (they are read methods)