Project 1 Ideas: interfaces and programming groups

- The Xref app: suppose this was a serious app to be implemented by a team of programmers "a crucial tool" for programmers. (Actually, it used to be.)
- Design phase: Someone says: Hey, we need to pull the ids out of the program text, and that's a separate job from building the data structure summarizing where each id shows up
- The team agrees, so two teams are formed, one for tokenization and one for indexing provided ids.

Two teams, two classes: how to communicate between them?

- We now have two teams, one for tokenization in say class Tokenizer and one for indexing given ids, the main code, in class Xref
- Next design step: Exactly how does Xref call Tokenizer?
- Xref needs to get tokens and their line numbers
  - Example fact: token "main" was found on line 88
- We could have just one method that returns both, but to return a String and an int, we need to create a helper object with fields String id and int lineNumber.
- Someone says: I know an easier way: have two methods…

Two teams, and an interface between them

- Idea of two methods to communicate ids and line numbers, looks like this:
  - String getNextID()
  - int getLineNumber() (to be called after getNextID)
- The teams agree to this idea, though one person says "They should be tied together in one method to ensure that the ID and line number are in fact related"
- They say "Let's write an interface to codify this API"

An Interface is a contract

- By agreeing to this API, codified in the Java interface, the two teams have a contract between them:
  - The Tokenizer team implements the API
  - The Xref team implements the code that calls the API
- Note that the interface doesn't capture everything about the interactions: for example, the constructor syntax is missing, and the treatment of end-of-data.
  - But it is an excellent start on the full contract.

The story, continued

- Someone on the Tokenizer team says "I just saw a regex on StackOverflow that does most of our work! Let's go with it…"
- Another says: "I’ve heard that Java regex processing can crash an app: it builds a huge stack to do its work. Let's stick with simple parsing…"
- They let the first person build a competing Tokenizer, and then test them against each other using something like TestTokenizer, which takes advantage of the fact that a each Tokenizer ISA JavaTokenizer, so the test code can be written once for either Tokenizer.
- For project 1, we're sticking with one tokenizer. But we can appreciate this idea that we could accommodate multiple ones.
The morals of the story

- Interfaces are crucial to software design, especially for larger programs with multiple source files and multiple teams.
- Interfaces let us treat multiple implementations of a certain API as the same Java type, allowing client code to run using either one.
  - Example: LinkedList and ArrayList are both type List, so code using List can switch off List implementations at will: just change the List creation step.

Resources on Interfaces

- Slides in class 3 introducing interfaces
- Homework 1 problem 9b and its solution
  - 9b. Write a Java interface source file UnionFind.java to (as far as possible) express the API on page 219, the union-find API. Also rewrite just the first line of UF.java to assert that class UF implements this interface.
  - This is your job now for Tokenizer.java.