Computer Laboratory 6

CSCI 1913: Introduction to Algorithms,
Data Structures, and Program Development

1 Essential information

- This assignment is due Tuesday March 18th at noon and will be turned in on gradescope
- The assignment is intended to be done in pairs, although you're allowed to work on your own if you want.
- You can start the assignment before your lab period, although I recommend not getting too far – the lab TAs often have useful hints, and I don't want you wasting too much time struggling with something that the TAs might announce in-class.
- For more rules see the lab rules document on canvas.
- It will be VERY HELPFUL to find your Lab02 submission and have that ready to reference.

2 Introduction

In this lab we will get our first taste of Java. Rather than picking a simple task, we will be making a relatively large program. Fear not, however, because we will not be programming something new. Instead, we will be rebuilding our NIM game, translating the design directly from the python version to Java. The end result will be a collection of static functions that will allow us to play Nim.

In particular, this lab will:

- Give you a chance to make sure you have a setup for Java programming.
- Practice translating a program you've already written from one language to another.
- Give you practical experience with Java coding.

I feel compelled to note that simply translating code from python to java like this is not necessarily a common task. While this will lead to working code – it will not be very *java style* code – that is, it will not make use of organizational techniques common in java code. That said, this will serve as GREAT practice for where our skills are NOW – there will be time in future labs to see the principals of a more object-oriented design. For now, focus only on gaining comfort with java's syntax through simple practice, and on how two different programming languages can be used to encode the same computation.

3 Software environment setup

NOTE – there is every chance this guide is slightly out-of-date. IntelliJ seems to update their UI once every few years. Your TAs should be able to give you a better step by step tutorial in lecture.

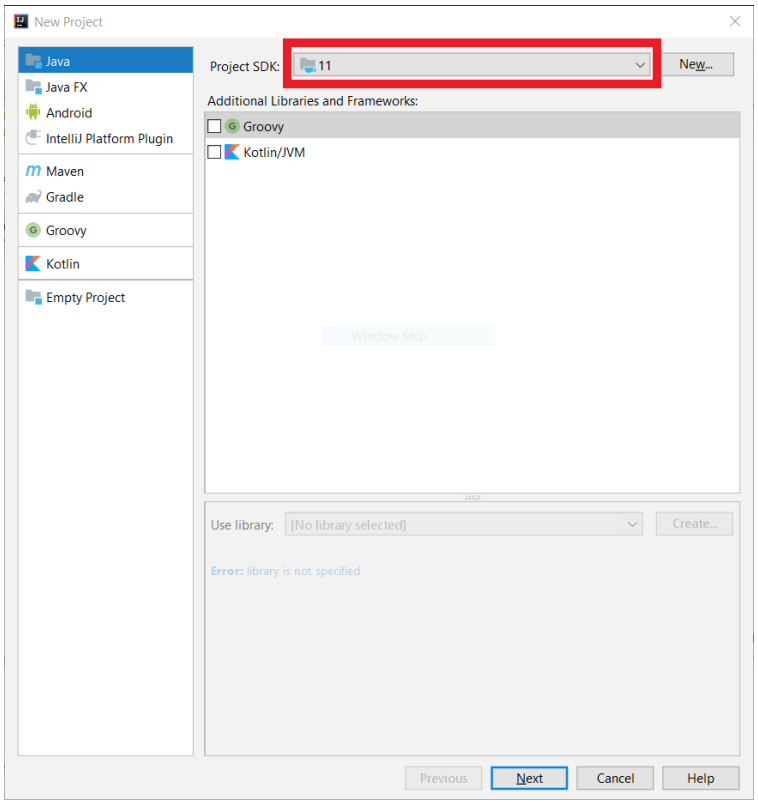
From this lab on we will be using the IntelliJ Idea IDE (integrated development environment). This section describes how to set up a new project for this lab and the Java files you are expected to download or create.

3.1 Start IntelliJ and create a new Java Project

Begin by loading IntelliJ. Depending on your past use of intelliJ you may need to enter some one-time settings such as color theme and default set of keyboard shortcuts for IntelliJ. You can set these options however you want, while I prefer default key-bindings and a light theme, many people may disagree, and it doesn't effect your coding output one bit. If you are not prompted for these settings, that too is fine. You can change these in the settings at your leisure.

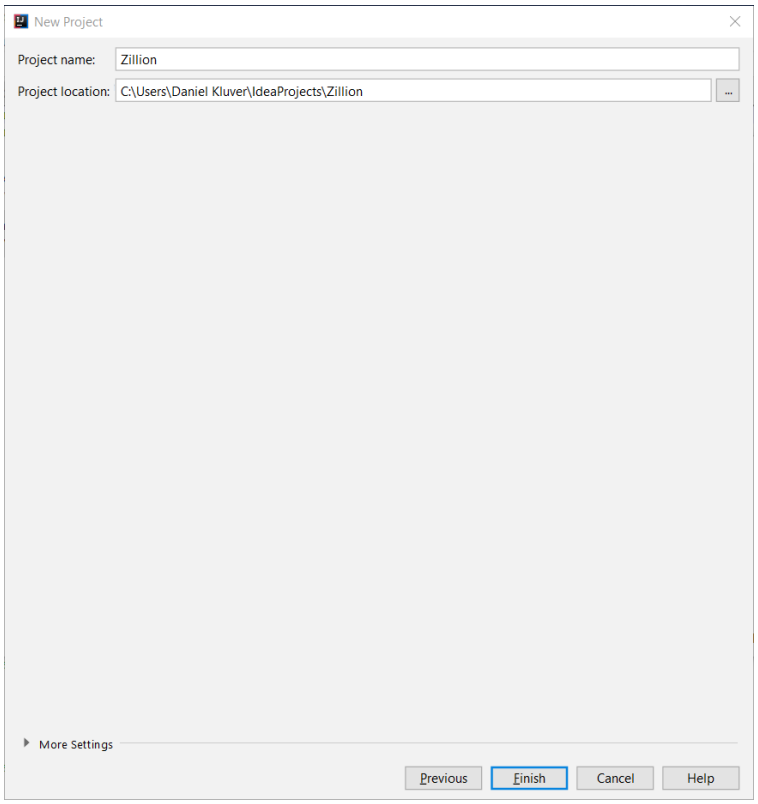
Once open, depending on settings and past use IntelliJ will either open to it's splash-screen or a previously opened project. Either way you want to **make a new project**. If you are in a previous project go to File → new → Project. Otherwise you can simply click the new project option.

The new project window should look something like this:



You want a Java project, with no additional library or framework support. If you are working on your own computer **don't forget to check the Project SDK** settings. You shouldn't need to be using an SDK version 11, like shown in the screenshot, so long as you are using 9 or above, you should be fine. If no SDK is listed, click new and setup your JDK. Typically IntelliJ will automatically open to your JDK (Java Development Kit) location.

Click next. This should bring you to the new project name and location seen:



Make sure your project has a name ("lab6" would be appropriate, but so would "NIM") **Make sure to check the project location** If necessary, update where the project is stored to keep your files well organized.

Click Finish and it should bring you to the main IntelliJ screen. If not shown, you will likely want to open the project view. The keyboard command for that is alt-1 on windows (and I believe this is true for Linux as well) This keyboard command is also shown in the background of the file viewer if no files are open.

3.2 Setup for Java files

Now that you have a project set up, click the tiny triangles in the project view to open the project itself and the source folder (src).

Download the template java files from canvas and move them to the src folder. You can do this by clicking and dragging the file from your file browser to IntelliJ's window. Make sure the file is in src, if it is not IntelliJ will not let you run this file.

Note Java, as a compiled language, has no patience for code it doesn't understand. Therefore, you may need to comment out parts of the testing code relating to functions you haven't written yet. So long as you have function calls to functions you haven't written Java wont run ANY of your code (even in unrelated files. Compiled languages tend to be all-or-nothing that way.) The template files provided should prevent this from being an issue this time but please remember this fact for future labs.

4 Files

This lab will involve the following files

- `NIM.java` - A template of this file is provided. You will be completing this file to build the code for managing the NIM game.
- `NimTester.java` – This file is provided. This file contains tests for the Nim functions. This should be your primary debugging tool for this lab, as the java autograder is typically less clear, and **much more fussy** than the python autograder.
- `PlayNIM.java` – This file is provided. When your NIM file is complete this should be able to play a 2-player game of nim!

5 Instructions

Before beginning you should:

1. Setup an IntelliJ package
2. Download the 3 provided files and place them in the src folder.
3. Ensure that NimTester can be run – it will not produce the correct outputs, but it should be runable without modifications.
4. Get a copy of your Lab02. If you do not have a copy of your original Lab02 – you can request a copy from another group, but if you do this you must add a comment to the top of the Nim.java file to make this clear.
5. Update the header comment in the provided NIM file with your name, the name of any lab partners, and if you are not working from your lab02, the name of who you got lab02 from.
6. Review Lab02.pdf to make sure you remember the basic structure of the Nim game – this will not be restated here.
7. Skim the formal requirements section and the java syntax sections. You will need information from both sections to complete this task.
8. Only after skimming the rest of this document fully should you start programming.

6 Formal Requirements

The five required functions are:

1. `public static int[] createGameState(int size, int tokenMax)`
2. `public static boolean isValidMove(int[] gameState, String row, String takes)`
3. `public static void drawGameState(int[] gameState)`
4. `public static int[] update(int[] gameState, int row, int takes)`
5. `public static boolean isOver(int[] gameState)`

6.1 createGameState(int size, int tokenMax)

This function takes two parameters, size, and tokenMax. It should return a newly created array of ints representing the number of tokens in each row. The size parameter sets the length of the newly created list, and the tokenMax parameter controls the values. In particular, the first row (position 0 in the array) should have one in it, the second row should have 2 and so-forth up to the max.

Notes:

- You can assume that both size and tokenMax are positive.
- if a size value of 0 is given, you should return a size 0 array.
- You **cannot** assume that size $<$ tokenMax. In a case like this the (size - tokenMax + 1) last positions should all be equal to token Max.

Hints:

- Unlike a list – when you make an array you give it a size – the array will then be full of 0s.
- Since the array is made full-length, you will need to assign to the array using indexing syntax `myarray[i] = value;`. This is different from the python approach where you make the list through repeated append.
- you can return an array in Java, simply use a return statement and the name of the variable pointing to your array.

6.2 isValidMove(int[] gameState, String row, String takes)

This function takes three parameters, gameState is an array representing the state of the game (similar to one that createGameState might return) row is a **string** representing which row the user has requested to take tokens from, and takes is a **stirng** representing how many tokens the user wishes to take. The function should return a boolean value – true to indicate that user has chosen a valid move, or false to indicate that the move would be invalid.

Notes:

- This function must not modify the gameState array parameter – the given array should have the same values before and after the function.
- Row and takes are strings, not integers. you CANNOT assume that they will only have integers in them or be in any particularly valid form. You can assume the string is non-empty (has at least one letter in it)
- A string that contains anything OTHER than digits (1,2,3,4,5,6,7,8,9 or 0) is invalid.
  - A function `isDigit(someString)` is provided to help check this. You should use this function. The function calling syntax remains essentially unchanged from python in this case – simply name the function and then put parameters in the parenthesis.
  - You can convert a string to an integer with the following code:
    ```
    int intTakes = Integer.parseInt(takes);
    ```
- If the row number is out-of-range then the row is invalid. The valid row numbers are between 1 and the length of the gameState array (inclusive)
  - you can get the length of an array with `gameStates.length`
- To be valid the takes number must be between 1 and 3 (inclusive)
- To be valid, the takes number must be less than or equal the number of tokens in the selected row (I.E. if a row only has 2 tokens, it is not valid to take 3 tokens)

Hints:

- This is probably one of the more complicated functions in this lab – fortunately, you already resolved these complexities in Lab02. Make sure you understand how you approached this program in Lab02, then translate that into code suitable for Lab06.
- rows will be entered by the user 1-indexed (meaning "1" represents the first row) this is different from how rows are handled in java arrays – where the gameState array is 0-indexed (0 represents the first row.) Your code will need to account for this.

6.3 update(int[] gameState, int row, int takes)

The update method takes three parameters gameState, row, and takes. While these have the same name as the parameters to the isValidMove function, there are important differences. Like before, gameState is an array representing the state of the game (similar to one that createGameState might return) row is an **int** representing which row the user has requested to take tokens from – this will be 0-indexed (not 1-indexed like the user input) and therefore will represent an array index directly. takes is an int representing how many tokens the user wishes to take. The function should return a **newly created** array that represents the board after the listed number of tokens have been taken from the listed row.

Notes:

- You can assume that row is a valid index for the gameState array, and that takes is between 1 and 3, and not greater than `gameStates[i]`. This is to say, you can assume that these parameters indicate a valid move as previously defined.
- Remember that the row is a 0-indexed integer in this problem, not a 1-indexed string like the last problem. You can use it directly like an array index.
- The gameState array input to this function should be unmodified by this function (I.E. it should have the same value before and after this function runs)
- The returned array should be a copy of the input array except for the index indicated by row – which should have fewer tokens as indicated by the takes variable.

Hints:

- Like with the earlier function you should remember that java arrays, unlike python lists, are created full-length and filled with 0. Therefore, you should plan on using repeated assignments: `newTokens[i] = something;` instead of appending when creating your return array.

6.4 drawGameState(gameState)

The drawGameState function takes one parameter, gameState is an array representing the state of the game (similar to one that createGameState might return). You can assume that the gameState array contains only non-negative numbers. The drawGameState function has no return value, and should instead use print statements to produce an output directly to the terminal.

The output of the function is a depiction of the state of the game, for instance, if given the array `[5,2,3,1]` The following should be printed:

```
====================
1 #####
2 ##
3 ###
4 #
====================
```

Formally:

- The first line should contain 20 equal-signs.
- for an array length n, the next n lines should be the row number, followed by a single space, and then a number of pound-signs `#` as indicated by the gameState array.
- The final line should contain 20 equal-signs. (there should be a final end-line after the 20 equals as well)
- No line starts-with or ends-with any spaces (with the technical exception of a line representing a row with no tokens left, which would be a number followed by a space.)
- Your output must match letter to letter – including spaces, newlines, capitalization, symbols etc.

Hints:

- A common task for this process is to repeat a string a fixed number of times. There **is not** an easy way to do this in Java. It would be easy enough, however, to make a helper function to make such strings, or to print the characters one-by-one in a loop.
- Remember that PDFs like to place "weird" but pretty version of characters. I STRONGLY recommend against using copy-paste as part of this function – you might accidentally copy a letter that IS NOT what we are expecting.
- Similarly, remember that PDFs like to use more, or fewer, spaces than actually included/intended. Another good reason to re-type this instead of copy/paste.
- You will likely need to use both `System.out.println` (print with a final new-line character) and `System.out.print` (print but do not end the line) to make this function work.

6.5 isOver(gameState)

The isOver function takes one parameter, gameState is an array representing the state of the game (similar to one that createGameState might return). This function returns a boolean value to indicate if the game is over. If every value in the input array is 0 – the game is over. If any value is not 0 – the game is not over. (You can assume the array is not empty – I.E. it has length greater than 0)

7 Java Syntax Guide

Some java has been provided in the provided Nim template file. This mostly includes:

- Function declarations
- proper javadocs
- A mostly-complete header-comment
- A single provided function which may serve as a useful example of some common syntax.

The syntax for these topics will not be explored here.

7.1 Math differences and other minor notes

Most mathematical computation does not need to be translated. The only differences:

- Logical operators are back to their C-style versions: `||`(or) `&&`(and) `!`(not)
- We no longer have exponentiation operator `**` or int-division operator `\\`
- The standard division operator now has type-dependent behavior. If you give it two integers then it will perform an integer division. Otherwise we get float-division.
- Boolean values are now lower-case `true` and `false`
- All statements should end with a semicolon. (Not needed with flow-of-control, but needed for everything else)
- The return statement is unchanged.
- printing is done with `System.out.println()`,`System.out.println(something)`,`System.out.print(something)`
  - `System.out.println()` with 0 parameters, this prints the newline character only
  - `System.out.println(something)` only 1 parameter is allowed at max. This will print `something` then print the newline character
  - `System.out.print(something)` only 1 parameter is allowed at max. This print `something` but NOT print the newline character, similar to `print(something, end='')` in python
- Strings must have double-quotes and only double-quotes.

7.2 Variables

Variables must have a type in java. For most variables in this lab the `int` type will be sufficient. Once created, variable assignments require no real modification.

Task	Python	Java
new int variables	`count = 0`	`int count = 0;`
update existing variable	`count = count + 1`	`count = count + 1;`

7.3 Basic flow of control

Task	Python	Java

if statement	`if condition:`
	`  tabbed body`
		`if (condition) {`
		`  tabbed body;`
		`}`

if/else statement	`if condition:`
	`  tabbed body`
	`else:`
	`  tabbed body`
		`if (condition) {`
		`  tabbed body;`
		`} else {`
		`  tabbed body;`
		`}`

if/else-if/else statement	`if condition:`
	`  tabbed body`
	`elif condition:`
	`  tabbed body`
	`else:`
	`  tabbed body`
		`if (condition) {`
		`  tabbed body;`
		`} else if (condition) {`
		`  tabbed body;`
		`} else {`
		`  tabbed body;`
		`}`

while loop	`while condition:`
	`  tabbed body`
		`while (condition) {`
		`  tabbed body;`
		`}`

for loop (repeat N times)	`for i in range(N):`
	`  tabbed body`
		`for (int i = 0; i < N; i++) {`
		`  tabbed body;`
		`}`

for loop (loop over a sequence)	`for i in range(len(lst)):`
	`  use lst[i]`
	`OR`
	`for elem in lst:`
	`  use elem`
		`for (int i=0; i<arr.length; i++) {`
		`  use arr[i]`
		`}`

7.4 arrays and lists

Arrays and lists are the biggest difference in this code. Python lists are created empty, and brought up to length by filling them with the append method. Java arrays are created at their final (and only) size, and cannot be resized once made. Therefore code for building and returning a list is quite different.

Task	Python	Java
create a new list/array	`lst = []`	`int[] arr = new int[10];`
update an existing position in the list/array	`lst[i] = newValue`	`arr[i] = newValue;`
reading from a list/array	`var = lst[i]`	`int var = arr[i];`
getting the size of a list/array	`size = len(lst);`	`int size = arr.length;`

"filling" a list/array	`squares = []`
	`for i in range(10):`
	`  squares.append(i*i);`
		`int[] squares = new int[10];`
		`for (int i = 0; i < 10; i++) {`
		`  square[i] = i*i;`
		`}`

for loop (loop over a list/array)	`for i in range(len(lst)):`
	`  use lst[i]`
	`OR`
	`for elem in lst:`
	`  use elem`
		`for (int i=0; i<arr.length; i++) {`
		`  use arr[i]`
		`}`

return a list/array from a function	`return lst`	`return arr;`

8 Java Autograder Notes

The java autograder is much fussier than the python autograder. This is mostly due to Java's nature as a compiled language. While we could often *partially* test python files that had errors in some functions – for your code to be testable your code must compile, have all the correct function definitions, and all the correct names. Even once you get the tests *running* you should also be aware that the autograder for java does not give as clear feedback as the python autograder when things go wrong. As such **you are expected to be testing you code on your own**. Do not use the autograder as a debugging tool.

Due to some strange things in the autograder, we have a few specific requirements for your code:

1. The code you submit must be valid and compileable java code. (We cannot test code with syntax errors. If we can't test it, we can't give you autograder credit for it)

2. The code you submit must have no package statement in it. (We have to do some weird things with package structures to get the code to compile correctly.) **YOU WILL GET 0 points if your code has a package statement!** Your IDE may have put one in without you noticing. If you see code like `package com.whatever;` at the top of your file you need to delete it.
3. The code you submit must match the provided function signature (name, parameter types, and modifiers) EXACTLY. Any mis-match, no matter how minor, may cause the test code to not compile, which would prevent testing.

As a general rule, if you started with the template file on canvas and didn't modify anything other than the function bodies you should be fine this time. Likewise, if your code can be run with the provided testing code (with no modifications to this testing code) you should be fine. If you do run into testing issues, do not hesitate to reach out to course staff to help with debugging.

9 Submission

For this lab you only need to turn in `NIM.java`. You can submit other files if you wish, but we do not promise to look at them during grading. Your submission must be entered into gradescope before the beginning of your next lab, although it can be up to 24 hours late for a 10% deduction.

Grading on this assignment will be 50% autograded and 50% manually graded. Like past labs, 10% will be based on code style, and 40% on manual review. Of particular note for code style – while java does not require proper indentation for functional behavior – this is always understood to be a core part of code style. Failure to properly indent your code will be seen as a code style issues. Beyond that – see past labs for general guidelines on code style – your code should be easily readable, your variables should be clear, etc. For this lab, proper documentation (javadoc format) have been provided. So long as you don't mess with the formal documentation you should be fine.