# Lab 2: Never Odd Or Even

*Pre-Lab Notes*

Lab submissions will **not be accepted late** – they must be submitted by **midnight the day after they're assigned** (to account for time zone issues). Some latitude might be allowed (I won't disqualify a submission that's mere minutes late!) but that's purely discretionary.

Labs are treated differently from assignments – you're encouraged to work with fellow students, ask questions of the TA during your lab section, and generally collaborate to complete your work. You should still submit your own work, **including citing with whom you collaborated in your submission**, but using answers developed by others will not be treated as plagiarism.

Nevertheless, you are still encouraged to make a good faith effort to complete your labwork, in order to exercise your understanding of the topics it covers. Lab submissions are marked purely 1 or 0 by whether the TA believes you at least made an honest attempt, even if you didn't succeed, so the value is in what the attempt teaches you rather than simply having the right answer.

## 1. Setting Up Java

If you missed last week's lab and haven't started the assignment yet, you may still need to set up your Java programming environment. If that's you, it's highly recommended you use the **IntelliJ IDE**, which provides a fairly standardized Java coding environment that cuts down on the risk that your code won't run for the TAs.

Install IntelliJ Community Edition (don't worry, it's free) :
https://www.jetbrains.com/idea/download/

Also be sure you're using Java 14 under File -> Project Structure, you may need to download a new SDK in order to have it available! 15 *should* be compatible as well, though testing if that's true is one of the purposes of a good technical shakedown.

## 2. Detecting Palindromes

A palindrome is a word or phrase that reads the same way backward and forward. The name “Otto”, for example, or “radar”, or the title of this particular lab.

Determining if a string is a palindrome or not is a fairly simple task. You already know that you can turn a string into an array of characters, and that an array makes comparing different letters at different places within the string very simple (so long as you remember to remove spaces and punctuation and lower-case all the letters).

So, obviously, I want you to find them using a list instead.

I will simplify things a little bit – we'll only consider single words, so you don't have to worry about spaces or punctuation. You must use a singly-linked list, each node of which will contain a single character element. As for what words to test, that's up to you, though you **must include a test** in your project that shows your function testing both a palindrome and a non-palindrome.

## *2.1 The Node*

To get you started, and to constrain any solutions that short-circuit the problem, I've included a code snippet for the node you should use in your list below:

```
class Node {
	private char letter;
	private Node next;
	Node (char input)
	{
		letter = input;
		next = null;
	}
	public void setNext(Node newNext)
	{
		next = newNext;
	}
	public Node getNext()
	{
		return next;
	}
	public void setLetter(char newLetter)
	{
		letter = newLetter;
	}
	public char getLetter()
	{
		return letter;
	}
}
```

Be sure to write your own LinkedList class, with whatever additional variables (a tail? A size?) and functions you think will help you solve the problem. You'll likely need a function you can call that returns true or false depending on whether the stored list is a palindrome, and your testing class (a main function should work) will need to populate the list by adding nodes containing letters, either hard-coded or by processing user input.

## *2.2 Opportunities and Optional Challenges*

There is more than one potential solution to this lab that makes use of techniques we've learned in class. While you should feel free to use any data structure or algorithm to make it work, I will say that simply moving everything from a list into an array so you can implement the array-based solution I

described above isn't very sporting (but I won't hold it against you!).

If you're done with time to spare, why not try and analyze your solution's worst-case run time? You can even try comparing your result with other students in the lab, or see if you can find a more time-efficient solution!

## 3. Deliverables

All lab submissions will be done through CourSys at https://courses.cs.sfu.ca/. For this lab, that should include:

1. A .zip archive of your code, organized into a single Java project. To ensure compatibility, it's recommended to use IntelliJ while developing your code, then using the export function under File -> Export -> Project To .zip File. **Don't forget to include a test file**, a Main.java with a main method that prints the results of a few runs of your function to the terminal would be best.

**Be sure you tidy up your code before submitting!** Check the Java code style guide posted on the course website, and do your best to provide helpful comments.