Week 12 - Monday

COMP 2000

Last time

- What did we talk about last time?
- Generics

Questions?

Project 4

Exam 2 Post Mortem

Java Collections Framework

Java Collections Framework

- Storing data is a fundamental part of programming
- The Java Collections Framework (JCF) provides a rich set of libraries for storing data in different ways
- It is the Java counterpart of the Standard Template Library (STL) provided for C++
- The JCF provides many interfaces that are implemented by particular classes (or you can write classes to implement them too)
- COMP 2100 focuses on implementing many of these classes, but you should rarely implement them yourself in the real world
  - Why reinvent the wheel?
  - Especially when the wheel has been very well tested

Container interfaces

- **`Collection`** Parent interface of most containers
- **`Iterable`** A collection that can be iterated over
- **`List`** A collection that contains items in an order
- **`Queue`** A collection that supports FIFO operations
- **`Set`** A collection of unordered objects
- **`Map`** A collection of (key, value) pairs

Container classes

- **`LinkedList`** List implementation using a linked list
- **`ArrayList`** List implementation using a dynamic array
- **`Stack`** FILO data structure
- **`Vector`** Like an **`ArrayList`**, but thread-safe
- **`HashSet`** Set implementation using a hash table
- **`TreeSet`** Set implementation using binary search trees
- **`HashMap`** Map implementation using a hash table
- **`TreeMap`** Map implementation using binary search trees

Tools

- Collections
  - sort()
  - max()
  - min()
  - replaceAll()
  - reverse()
- Arrays
  - binarySearch()
  - sort()

`List<E>` interface

- Often, you will need to keep ordered lists of things
- This functionality is built into Python
- In Java, you need to use a library:
- Interface:
  - **`List<E>`**
- Common implementing classes:
  - **`ArrayList<E>`**
  - **`LinkedList<E>`**

List<E> methods

- The **List<E>** interface is one of the biggest you'll ever see
- Here are a few important methods in it

Returns	Method	Description
boolean	**add(E element)**	Adds **element** to the end of the list
void	**add(int index, E element)**	Adds **element** before **index**
boolean	**addAll(Collection<? extends E> collection)**	Adds everything from **collection** to this list
void	**clear()**	Removes everything from this list
boolean	**contains(Object object)**	Returns **true** if this list contains **object**
E	**get(int index)**	Return the element at **index**
int	**indexOf(Object object)**	Returns the first index where something that equals **object** can be found
boolean	**isEmpty()**	Returns **true** if the list is empty
boolean	**remove(int index)**	Remove the element at **index**
E	**set(int index, E element)**	Set the item at location **index** to **element**
int	**size()**	Returns the size of the list

`ArrayList` vs. `LinkedList`

- As you will learn (or have learned) in COMP 2100, **`ArrayList`** uses an array inside to store datay
  - When you need more space, it makes a new array and copies all the old stuff into the new array
- **`LinkedList`** uses a (wait for it) linked list to store the data
- In principle, **`LinkedList`** is faster for lots of unpredictable adds and removals
  - Especially adds and removals at the beginning of the list
- In practice, **`ArrayList`** is almost always faster
  - Modern machines are really good at ripping through arrays

Using the `List` interface

- **`ArrayList`** and **`LinkedList`** do have a few methods that the other one doesn't have
- However, you almost always want to treat them like a **`List`**
- It's a very common practice to store the class in a **`List`** variable
- Then, if you decide that you really wanted an **`ArrayList`** instead of a **`LinkedList`**, you only have to change one thing

```
List<Wombat> wombats = new LinkedList<Wombat>();  // Change to ArrayList?
Wombat walter = new Wombat("Walter");
wombats.add(walter);
wombats.add(new Wombat("Wilma"));
wombats.add(new Wombat("Winona"));
System.out.println("Size: " + wombat.size());
if(wombats.contains(walter))
        System.out.println("We've got Walter!");
```

List practice 1 (Fizz Buzz)

- Create an **`ArrayList`** of **`String`** values to hold
- Prompt the user for a positive integer
- From 1 up to the number they enter, add the **`String`** equivalent of that number to the list
- Exceptions:
  - If the number is divisible by 3, add Fizz to the list instead
  - If the number is divisible by 5, add Buzz to the list instead
  - If the number is divisible by both, add Fizz Buzz to the list instead
- Output the list
- Example for 16:
  - **`1, 2, Fizz, 4, Buzz, Fizz, 7, 8, Fizz, Buzz, 11, Fizz, 13, 14, Fizz Buzz, 16`**

List practice 2 (a real job interview question)

- There are ***n*** prisoners standing in a circle, about be executed
- The executions are carried out starting with the ***k***th person, and removing every successive ***k***th person going clockwise until no one is left
- Prompt the user for ***n*** and ***k***
- Determine where a prisoner should stand in order to be the last survivor
- For example, if ***n*** = 5 and ***k*** = 2, the order of executions would be [1, 3, 0, 4, 2] (assuming 0-based numbering)
- **Hint:** Use a list and repeatedly remove indexes

Quiz

Upcoming

Next time…

- Sets
- Maps

Reminders

- **Start Project 4**
  - **Get your teams figured out immediately!**
- Keep reading Chapter 18