ArrayLists and For:Each Loops

For this lecture we are going to cover:

- lists
- a specialized loop for list like structures - For:each

Your future in CS

I used to include this on my slides, but since these slides have changed - going to just leave it up here for every notebook. I get a lot of questions about more programming courses, the concentrations, and minors in computer science. Here is a brief reminder.

CS 165 – Next Course In Sequence, also consider CS 220 (math and stats especially)

- CO Jobs Report 2021 – 77% of *all* new jobs in Colorado require programming
- 60% of all STEM jobs requires *advanced* (200-300 level)
- 31% of all Bachelor of Arts degree titled jobs also required coding skills
- 2016 Report found on average jobs that require coding skills paid $22,000 more

- Concentrations in CS:

    - Computer science has a number of concentrations. General concentration is the most flexible, and even allows students to double major or minor pretty easily. The others are specialized paths in CS. Software Engineering, Computing Systems, Human Centered Computing (Psychology+CS), Networking and Security, Artificial Intelligence, and Computer Science Education.
    - Minors: We have three minors. Minor in computer science (choose your own adventure minor), Minor in Machine Learning (popular with stats/math), and Minor in Bioinformatics (Biology+Computer Science)

Reading Check-in

Given the following code, what is the size of the list?

In [5]:
```
ArrayList<String> list = new ArrayList<>();

list.add("link");
list.add("zelda");
list.add("ganon");
list.add("epona");
list.add("sages");

list.remove(2);

System.out.println(list.size());
```

```
// question 2, what is the element at position 2?

System.out.println(list.get(2));
```

4
epona

ArrayLists

So far in programming, we have had to keep a variable for every value we want to keep. But

- What happens if we had 100 values, like grades for one student
- Or even number of students in this class?
- What about students across multiple classes?

Keeping every variable would be impossible! Thus enter an ArrayList.

ArrayList is a Object (called a DataStructure):

- Allows me to keep items
  - Stored in a set order!
- Let's lets me add items at any point
- Let's me remove items
- Keeps track of the size, and other internal features

> Advanced Topic
> The "Array" part of the ArrayList we will come back to in the future. Java actually has multiple types of lists. However, we are starting with ArrayLists as it makes file handling easier.

So how do how can we visualize array lists? Let's take the code above.

Stack

variable	value
list	ArrayList@7ba7

ArrayList@7ba7 (before the remove call)

index	object
0	"link"
1	"zelda"
2	"ganon"
3	"epona"
4	"sages"

However, when we removed ganon, all the values shifted up, to create | index | object | | - | - | | 0 | "link" | | 1 | "zelda" | | 2 | "epona" | | 3 | "sages" |

> Memory Note:
> For simplicity, I put the String directly in the table. In truth, since every String is an Object, it is actually a memory address to that Object location. This matters if we are changing values in Objects we create.

Another Way to Visualize

```
["link", "zelda", "ganon", "epona", "sages"]
```

- This way is actually the standard way in which they print.
- This is actually how I tend to write it down
- I then insert (mentally or visually) 0-N under the items

In class activity:

- Create an ArrayList of Strings
- Add the name of each person to the ArrayList
- Print out the ArrayList
  - For reference, you an just type `System.out.println(listname);` to print out the list

ArrayLists require the following line at the top of the file `import java.util.ArrayList;` It has already been provided.

In [6]:
```
ArrayList<String> names = new ArrayList<>();
names.add("link");
names.add("zelda");
names.add("ganon");

System.out.println(names);
```

```
[link, zelda, ganon]
```

ArrayList Methods

There are a number of useful ArrayList methods.

- `.get(int)` - returns the item at the set index
- `.remove(int)` - removes the item at the set index (giving the item if needed)
- `.set(int, value)` - replaces an item at an already exist index
- `.add(value)` - adds an item to the end of the list
- `.add(int, value)` - inserts an item at a set location
- `.size()` - returns the total number of elements in the list
  - yes, this one is confused with .length() all the time.

In [7]:
```
System.out.println(list.get(0));

System.out.println(list.get(3));
```

```
link
sages
```

But what happens if I try to access something not in the list?

In [9]:
```
System.out.println(list.get(100));
```

```
--------------------------------------------------------------------------
java.lang.IndexOutOfBoundsException: Index 100 out of bounds for length 4
        at java.base/jdk.internal.util.Preconditions.outOfBounds(Preconditions.java:64)
        at java.base/jdk.internal.util.Preconditions.outOfBoundsCheckIndex(Preconditions.java:70)
        at java.base/jdk.internal.util.Preconditions.checkIndex(Preconditions.java:266)
        at java.base/java.util.Objects.checkIndex(Objects.java:359)
        at java.base/java.util.ArrayList.get(ArrayList.java:427)
        at .(#38:1)
```

This error is arguably one of the most common errors you will see in programming. Why?

- Sometimes we aren't adding to lists when we think
- Sometimes we forgot we removed an item
- Sometime our loops to access them are simply off-by-one (OB1)

Speaking about loops, now that we have .get(), we can easily get every item.

In [12]:
```
for(int i = 0; i < list.size(); i++) {
    System.out.printf("%d: %s\n", i, list.get(i));
}
```

```
0: link
1: zelda
2: epona
3: sages
```

In Class Activity

Part 1

Take the list you already made, and print it with the indexes, similar to the example above.

Part 2

Modify that loop to print every *other* name.

Part 3

Print out the length of each item in the list.

In [16]:
```
// to be filled out in class
```

```
for(int i = 0; i < names.size(); i++) {
    System.out.printf("%d: %s\n", i, names.get(i));
}
System.out.println();
for(int i = 0; i < names.size(); i += 2) {
    System.out.printf("%d: %s\n", i, names.get(i));
}
System.out.println();
for(int i = 0; i < names.size(); i++) {
    String name = names.get(i);
    System.out.println("length is " + name.length());
}
```

```
0: link
1: zelda
2: ganon

0: link
2: ganon

length is 4
length is 5
length is 5
```

For:each loop

This last idea of grabbing an item, storing it, and using methods on that item is **very** common.

```
for(int i = 0; i < names.size(); i++) {
    String name = names.get(i);
    System.out.println("length is " + name.length());
}
```

As such, there is a specialized loop structure we use. The **for each loop** or **enhanced for loop**.

```
for(String name : names) {
    System.out.println("length is " + name.length());
}
```

Yes, the above are equivalent!

In [17]:
```
for(String name : names) {
    System.out.println("length is " + name.length());
}
```

```
length is 4
length is 5
length is 5
```

In Class Activity

Write a loop (using a for:each), that prints out the following for every item in your ArrayList you have been using.

```
Name is N characters long, and the first character is X and the middle
character is Y and the last character is Z.
```

Example:

```
Link is 4 characters long, and the first character is L, and the middle
character is n, and the last character is k.
```

Some useful String methods to help out:

- .length()
- .charAt(int)

In [19]:
```
for(String name : names) {
    int len = name.length();
    char first = name.charAt(0);
    char middle = name.charAt(len/2);
    char last = name.charAt(len-1);

    System.out.printf("%s is %d characters long, and the first character is %c, and th
                      name, len, first, middle, last);
}
```

```
link is 4 characters long, and the first character is l, and the middle character is
n, and the last character is k.
zelda is 5 characters long, and the first character is z, and the middle character is
l, and the last character is a.
ganon is 5 characters long, and the first character is g, and the middle character is
n, and the last character is n.
```