Module 1 - Lecture 7

Collections Part 1

TPS Reports!

https://te-reign.azureedge.net/

username: <email>
password: TechElevatorStudent

Review

- Objects
- Classes
- Reference types vs Primitive types
- Strings

Collections!

Wait, I thought we did this...

We learned about Arrays already, so why are we learning about other collections?

Downside of Arrays

Resizing requires a new array.

What if I want to add a third item to the array below?

```
int[] myIntArray = new int[2];
myIntArray[0] = 128;
myIntArray[1] = -5;
```

ArrayList<T>

ArrayList<T>

- Zero-indexed
- An ordered set of elements accessible by index
- Allows duplicates

… but there's more!

- Can grow and shrink as you add and remove elements.
- You can also add to and remove from the middle.

ArrayList<T>

Create

```
List<String> arrayList = new ArrayList<String>();
```

Add elements at the end

```
arrayList.add("First");
arrayList.add("Third");
arrayList.add("Fourth");
```

Add at an index

```
arrayList.add(1, "Second");
```

Remove at an index

```
arrayList.remove(3); // Removes "Fourth"
```

Get an element at an index

```
String firstElement = arrayList.get(0); // "First"
String thirdElement = arrayList.get(2); // "Third"
```

Get the *current size*

```
int currSize = arrayList.size();
```

Let's Code!

It’s not all sunshine & rainbows

- An ArrayList uses an Array and it’s making a best guess when resizing for you.

- **ArrayLists can only be made up of reference types (objects)**.
  - You must use a Primitive Wrapper class instead of working with primitive types directly.

Primitive Wrapper classes

- Each primitive data type has a corresponding wrapper class.

Primitive	Primitive Wrapper
boolean	Boolean
char	Character
byte	Byte
short	Short
int	Integer
float	Float
double	Double

Boxing

When converting from a primitive type to a Primitive Wrapper class, this process is called **autoboxing**.

When converting from a Primitive Wrapper class to a primitive type, this process is called **unboxing**.

Let's Code!

Queue<T>

- A simplified collection. We add to the end and access from the beginning.

- First in, first out (FIFO)

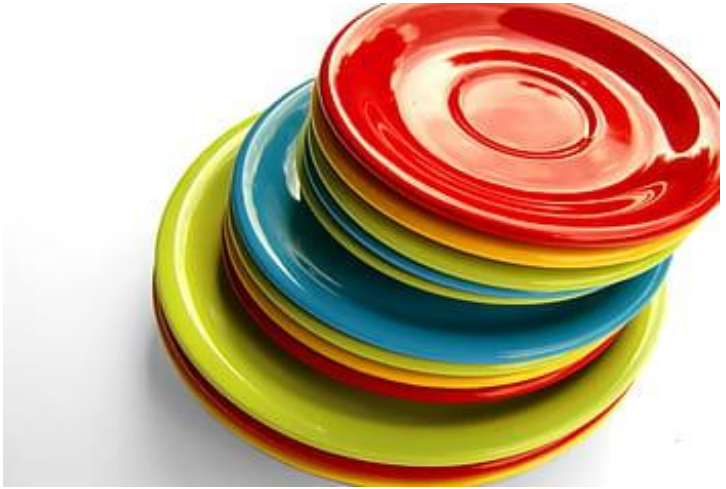
Queue<T>

```
Queue<String> myQueue = new LinkedList<String>();

myQueue.offer("First"); // Add "First"
myQueue.offer("Second"); // Add "Second"

// Get "First" and remove it from the queue
String nextItem = myQueue.poll();
```

Stack<T>

- A simplified collection. We add to the top and access from the top.

- Last in, first out (LIFO)



Stack<T>

```
Stack<String> myStack = new Stack<String>();

myStack.push("First"); // Add "First"
myStack.push("Second"); // Add "Second"

// Get "Second" and remove it from the queue
String nextItem = myStack.pop();
```

Let's Code!

QUESTIONS?