

Why did the Java Developer quit his job?

Because he didn't get arrays.

# Module 1-7

Collections: Lists

# Objectives

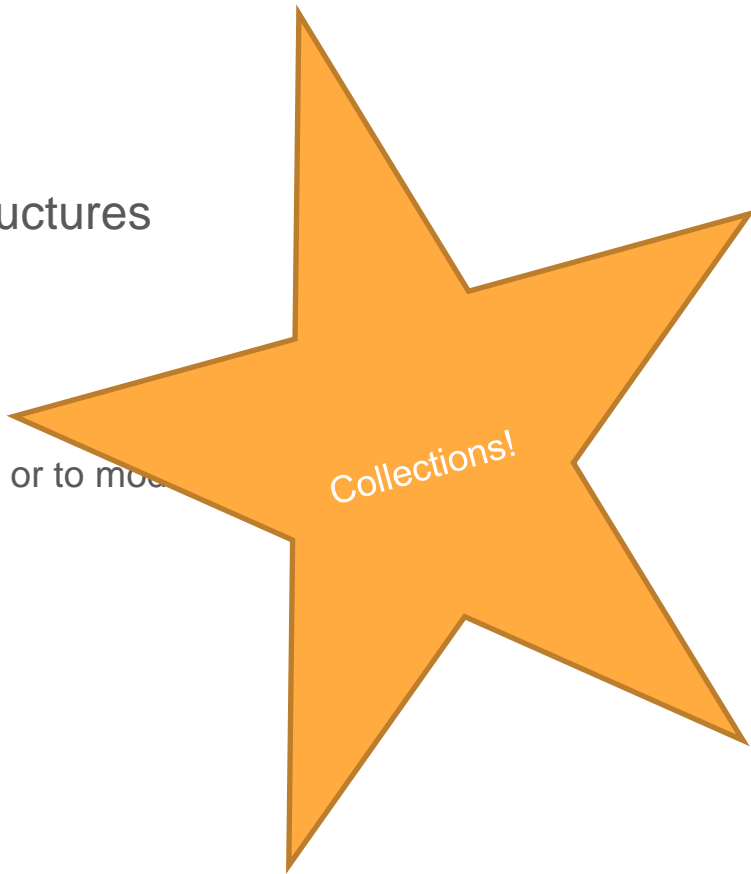
- Students should be able to describe the purpose and use of Collections
- Students should be able to describe the differences between a List and an array and the different use-cases for them
- Students should be able to demonstrate an understanding of code libraries and the use of namespaces in C# or packages in Java to help organize them
- Students should be able to use the for-each loop to iterate through a collection
- Students should be able to describe what a Stack and Queue are and how they work
- Students should be able to identify from syntax what variables are arrays and what are Lists

# Objectives

1. What are Collections and why we use them
2. Differences between array and List
3. Packages in Java for organization
4. For-each loop to iterate through a collection
5. Stack and Queue

# Array Recap

- Arrays are simple data structures
  - Hold collection of like data
- Not flexible
  - Difficult to add new element or to modify



# Collections

1. Classes that live in a package
2. Come from standard library of classes
  1. java.util package
3. Already written for you!
4. Can use import statement or fully qualified statement

```
java.util.Scanner input = new java.util.Scanner(System.in);
```

# Package

- Organizes classes within libraries
- Creates scope to prevent two classes with same name from overlapping
- `java.lang` package automatically imported
  - String class, System class, wrapper classes (Boolean, Integer, Double)

# List class

A List is:

- Zero-indexed like array
  - Ordered set of elements (accessible by index)
  - Allows duplicates
  - Dynamic in size
- 
- Java List is an interface, so we use ArrayList
    - Called Programming to an Interface
  - Must be imported from `java.util` package



# List syntax

List <datatype> objectName = new ArrayList<>();

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

```
names.add("Rick");
```

```
names.add("Beth");
```

```
names.add("Jerry");
```

```
names.add(0, "Sam");
```

```
for (int i = 0; i < names.size(); i++) {
```

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

```
}
```

- The add method is overloaded – add name is the same, but takes in different parameter listings

# List methods

```
List <String> moreNames = new ArrayList<>(Arrays.asList("Tom",  
"Tim", "Joe", "Jim"));

System.out.println(moreNames.size()); // prints 4
moreNames.add(0, "Jane");
System.out.println(moreNames); // prints out array elements
moreNames.remove(3);           // removes element in pos 3

System.out.println(moreNames.contains("Tom")); // prints true

moreNames.removeAll(moreNames);
// removes all elements from ArrayList

System.out.println(moreNames.isEmpty()); // prints true
```

- The add method is overloaded – add name is the same, but takes in different parameter listings

Let's code!

# Primitive Wrapper objects

Lists and other collections can only hold objects!

```
List <Integer> ages = new ArrayList<>();

ages.add(29);
ages.add(21);
ages.add(35);
ages.add(32);

for (int i = 0; i < ages.size(); i++) {
    System.out.println(ages.get(i));
}
```

- Wrapper class wraps primitive types so they can be references types
- Autoboxing is process of converting primitive type to reference type (moving from stack to heap)
- Unboxing is moving from heap to stack, converting back to primitive type

# Foreach loop

```
List <Integer> ages = new ArrayList<>();
```

```
ages.add(29);
```

```
ages.add(21);
```

```
ages.add(35);
```

```
ages.add(32);
```

```
for (Integer age: ages) {  
    System.out.println(age);  
}
```

- Convenience method to iterate through a collection
- Cannot modify contents during iteration
- Useful for when you don't need the index, just want to go through to each element

Let's code!

# Queues

- List, but used in a certain way to get certain result
- VERY COMMON data structure
- FIFO
  - First In, First Out
  - Elements are inserted at end of list, and deleted from beginning
- Line of customers waiting to be helped
- Print queue (documents printed in order received)

# Queue syntax

Queue <datatype> objectName = new LinkedList<>();

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

```
todoList.offer("Rick");
```

```
todoList.offer("Beth");
```

```
todoList.offer("Jerry");
```

```
todoList.offer("Sam");
```

```
while (todoList.size() > 0) {
```

```
    String nextTodo = todoList.poll();
```

```
    System.out.println("NEXT ON MY LIST: " +  
        nextTodo);
```

```
}
```

- To add elements to Queue, we can use add or offer (offer is preferred because add will throw an exception)
- To remove elements, we can use remove or poll (poll is preferred because remove will throw an exception)



# Stacks

- List, but with different behavior
- VERY COMMON data structure
- LIFO
  - Last In, First Out
  - Elements are inserted at front of list, and deleted from beginning
- Plates at a buffet
- Undo feature of an edit

# Stack syntax

```
Stack <datatype> objectName = new Stack<>();
```

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

```
numberStack.push("123");
```

```
numberStack.push("456");
```

```
numberStack.push("789");
```

```
while (numberStack.size() > 0) {  
    String number = numberStack.pop();  
    System.out.println(number);  
}
```

- To add elements to Stack, we push the elements on the stack
- To remove elements, we pop them off the stack