

MODULE 1 - DAY 7

Collections Part 1: Lists
Plus Boxing and For-Each Loops

Module 1 Day 7 - Collections Part 1 Reference

Java Primitive Type	Wrapper Class	Constructor Argument
byte	Byte	byte or String
short	Short	short or String
int	Integer	int or String
long	Long	long or String
float	Float	float, double, or String
double	Double	double or String
char	Character	char
boolean	Boolean	boolean or String

Java For-Each Loop


```
for (String name : names) {  
    //do something  
}
```

TODAY'S OBJECTIVES

- **Collections** – why and how do we use them?
- **List vs array**
- Understand the concept of **code libraries** and **namespaces**
- Use the **for-each** loop to iterate through a **collection**
- **Stacks*** and **Queues***
- **Java Wrapper Objects**

***Not included in lecture but good to know**

ARRAYS

- Arrays let us work with a collection of like data types.
- Arrays aren't very flexible.
- Us as we learned them: 

COLLECTIONS

- A collection represents a group of objects, known as its elements.
 - ◆ Some collections allow duplicate elements and others do not.
 - ◆ Some are ordered and others unordered.
- There are multiple types of collections
- Collection classes are available to us in Java's standard library of classes
- We import them into our projects using import statements
- In Java, Collections are in the `java.util` package

LISTS

→ A List:

- ◆ Is a collection of Objects of the same type
- ◆ Is **Zero-indexed**, similar to arrays
- ◆ Is an **ordered set of elements** accessible by index
- ◆ **Allows duplicates**
- ◆ **Can grow and shrink** as elements are added and removed
 - Methods: **add()** and **remove()**

→ In Java, we use an **ArrayList** (object) to implement a List (interface)

- ◆ **This will make sense... later :)**
Just know ArrayList is the most common implementation of a List.

PRIMITIVE WRAPPER OBJECTS

→ Lists and other collections can hold objects.

◆ Wait... what if I want a list of ints? Or floats? Or doubles?

→ Java has a wrapper class for each primitive data type.

Java Primitive Type	Wrapper Class	Constructor Argument
byte	Byte	byte or String
short	Short	short or String
int	Integer	int or String
long	Long	long or String
float	Float	float, double, or String
double	Double	double or String
char	Character	char
boolean	Boolean	boolean or String

AUTOBOXING AND UNBOXING

- **Autoboxing** is the automatic conversion that the Java compiler makes between the primitive types and their corresponding object wrapper classes.
- **Unboxing** is converting an object of a wrapper type to its corresponding primitive value.
 - ◆ The Java compiler applies unboxing when an object of a wrapper class is:
 - Passed as a parameter to a method that expects a value of the corresponding primitive type.
 - Assigned to a variable of the corresponding primitive type.

AUTOBOXING EXAMPLE

<https://docs.oracle.com/javase/tutorial/java/data/autoboxing.html>

```
List<Integer> li = new ArrayList<>();  
for (int i = 1; i < 50; i += 2) {  
    li.add(i);  
}
```

Without autoboxing, the above code would result in a compiler error since we must add *Objects* to Lists, not primitive types.

Autoboxing allows the java compiler to interpret the preceding code as the code below at compile time, “wrapping” the int in an Integer wrapper.

```
List<Integer> li = new ArrayList<>();  
for (int i = 1; i < 50; i += 2)  
    li.add(Integer.valueOf(i));  
}
```

UNBOXING EXAMPLE

```
public class Unboxing {
```

```
    public static void main(String[] args) {
```

```
        Integer i = new Integer(-8);
```

```
        // 1. Unboxing through method invocation
```

```
        int absVal = absoluteValue(i);
```

```
        System.out.println("absolute value of " + i + " = " + absVal);
```

```
        List<Double> ld = new ArrayList<>();
```

```
        ld.add(3.1416);    // autoboxed through method invocation
```

```
        // 2. Unboxing through assignment
```

```
        double pi = ld.get(0);
```

```
        System.out.println("pi = " + pi);
```

```
    }
```

```
    public static int absoluteValue(int i) {
```

```
        return (i < 0) ? -i : i;
```

```
    }
```

```
}
```

<https://docs.oracle.com/javase/tutorial/java/data/autoboxing.html>

FOR-EACH LOOP

- For-each is a loop specifically created to iterate through collections.
- Cannot modify contents of the collection used by the for-each loop during iteration.
- Useful to work with the elements when we don't need to know index. When working with collections, the content is usually what is most useful, not the actual index position.

QUEUES

- Queues are a collection that provide additional functionality when adding and removing items.
- To add items, we **offer()**
 - ◆ Offer returns a boolean to indicate success of adding an Object
- To remove items, we **poll()**
 - ◆ Poll returns the next list item, or null if the queue is empty
- Queues operate as a **FIFO** (First In, First Out) structure
- Queue is an interface, so we instantiate a **LinkedList**

STACKS

- Queues are a collection that provide additional functionality when adding and removing items.
- To add items, we **push()**
 - ◆ Pushes an item onto the top of a stack
- To remove items, we **pop()**
 - ◆ Removes the object at the top of the stack
- Stacks operate as a **LIFO** (Last In, First Out) structure

WHAT QUESTIONS DO
YOU HAVE?

INDIVIDUAL EXERCISES

Due Thursday, 1/23, 9am

READING FOR WEDNESDAY

Book > Module 1 > Collections Part 2