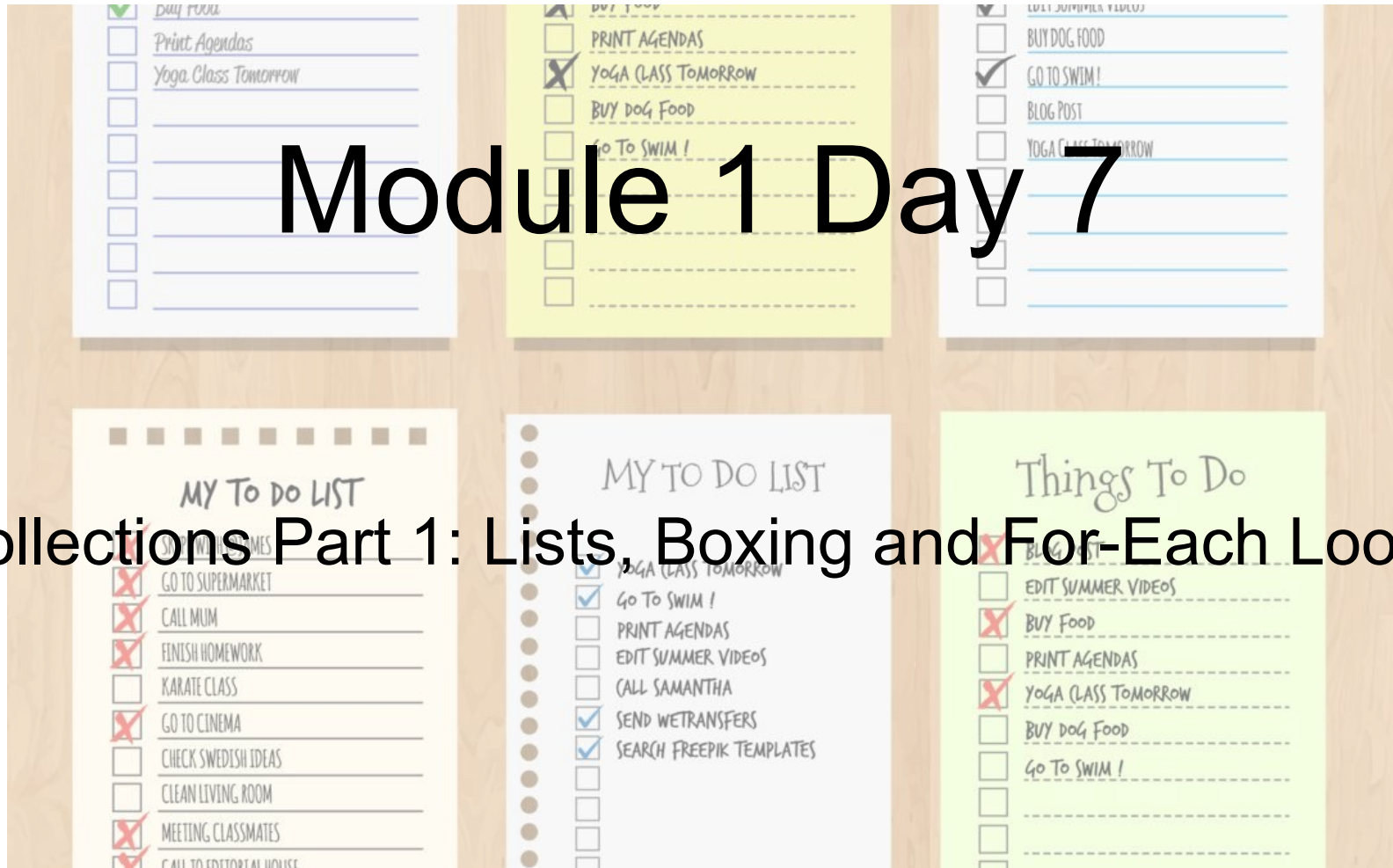


Module 1 Day 7

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



Module 1 Day 7

Can\Do you?

1. ... describe the purpose of Packages and Namespaces
2. ... describe the purpose and use of Collections?
3. ... describe the differences between a List and an Array?
4. ... speak to the different use-cases for Lists and Arrays?
5. ... understand & explain the need for and use of wrapper objects in Lists (Boxing) ?
6. ... use the common API operations (Methods) of a List and how to use them?
7. ... use the Foreach Loop to iterate through a collection ?

BONUS TOPIC:**

1. ... Describe the Stack & Queue Collection Objects and their basic use?

** These objects are not part of the formal lecture, but they are good to know!

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 Pattern](#)

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

Packages & Namespaces

- Packages are used to organize our code. Putting code into collections helps identify its association to other classes and helps to limit the size of our final application by only importing what is needed
- In other words, these collections contain classes with similar functions or functional focus
- Namespaces define the organizational hierarchy of a package. Consider the package we will work with today `java.util`
 - `java` is the top level namespace of the Java language
 - `util` is the namespace for utilities
 - Together, they identify a package
- Within `java.util` is the `List` class, it is imported using `java.util.List` or `java.util.*`

Arrays

- Arrays let us work with a collection of like data types.
- Arrays aren't very flexible.
- Us as we learned them:
 - Happy: When we learned that there was a structure to group similar types and work with them as a set
 - Meh: when we learned how to work with them in tedious loops
 - Sad: When started dealing with I/O and realized the implications of them being statically sized

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: Array, List, Stack, Queue, Map and more
- 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 part of 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 instantiate a List (interface)
 - Remember, generally, with objects we Declare = new Instance(Initialize)

**This concept of an interface will make sense later.
Just know that ArrayList is the most common implementation of a
List.**

Primitive Wrapper Objects

- Lists and other collections ~~can~~ hold objects.
 - You may ask... what if I want a list of ints? Or floats? Or doubles?
- Remember, Java has a wrapper class for each primitive data type.

- These are the Types we use
To place primitives into Lists

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 Part II: Beyond the definition

- **Autoboxing** is the automatic conversion 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

<https://docs.oracle.com/javase/7/tutorial/java/data/autoboxing.html>

```
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;
    }
}
```

For-each Loops

- The For-each loop is specifically created to iterate through collections of *Objects*.
- You cannot modify the collection being used by the for-each loop during or within iteration, **but** you may modify Object properties and invoke methods.
- For-each Loops are useful when we want to interact with the elements in a collection and do not need to know index.
- When working with collections, the content is usually what is most useful or important, not the actual index position.



BONUS SLIDES!

... and a trip to the Apple Store

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?



Day 7 Individual Exercises

Due Thursday, 5/20/2021, 9am

Reading for Wednesday

Book > Module 1 > Collections Part 2