Week 4 – Topics

- String vs StringBuilder
- Classes, Objects, Interfaces
- Lists, Sets, & Maps
- Collections
- Menu App Example

String vs StringBuilder

String

In Java, a String is immutable, which means that it can not be modified. Strings are very useful in programming, but they should only be used if you are not modifying a String once you create it!

To further unpack the immutability of the String Object, check this out:
You can assign a new value to a String variable, but both values will remain in memory. In the following example, Java does not use the same space in memory, it leaves the value "Bob Brown" and creates a new space for "Sue Brown".

The syntax works, but it is not good coding practice to use the String datatype when you want to dynamically create or modify a String value within your code.

```
String name = "Bob Brown";
System.out.println(name);
name = "Sue Brown";
System.out.println(name);
```

https://github.com/ckiefriter1/Java-Code-Examples/blob/main/src/strings/StringTest.java

StringBuilder

Java provides another Object that allows a programmer to build a String dynamically called StringBuilder, and it is mutable, which means that a StringBuilder can be modified.

Internally, these objects are treated like variable-length arrays that contain a sequence of characters. At any point, the length and content of the sequence can be changed through method invocations.

```java
package strings;

public class StringBuilderExample {
    public static void main(String[] args) {

        String firstName = "Bob";
        String lastName = "Brown";
        String middleInitial = "B";
        String space = " ";

        System.out.println(firstName);
        System.out.println(lastName);

        StringBuilder sb = new StringBuilder();

        // use append() to dynamically create a String using StringBuilder
        sb.append(firstName);
        sb.append(space);
        sb.append(lastName);
        System.out.println(sb.toString());

        // Add a middle Initial
        sb.insert(firstName.length() + 1, middleInitial + space);
        System.out.println(sb.toString());
    }
}
```

https://github.com/ckiefriter1/Java-Code-Examples/tree/main/src/strings
https://github.com/ckiefriter1/Java-Code-Examples/blob/main/src/strings/StringBuilderExample.java
https://github.com/ckiefriter1/Java-Code-Examples/blob/main/src/strings/StringTest.java

Lists

Collections

Think of a Collection as an Object, but that Object can contain (or represent) a group of Objects. Java provides a Collection Framework, which is an architecture that allows a unified access to a variety of different types of collections.

Collection is technically an Interface vs a Class → Collection Interfaces: Set, List, Map, and others

Collection Implementations: There are a number of classes provided in Java that are used to implement the Collection Interfaces. See the Java Collections Framework Overview link below for additional information on Collection Implementations.

Interface --> Implementations
Set --> HashSet, TreeSet, LinkedHashSet
List --> ArrayList, LinkedList
Map --> HashMap, TreeMap, LinkedHashMap

The Collections Class contains static methods that either return collections or perform some operation on collections. A NullPointerException is thrown by all methods of this class if the collections or Class Objects provided are null.

https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/util/doc-files/coll-overview.html

https://docs.oracle.com/javase/8/docs/api/java/util/Collection.html

Lists

Lists are used to "collect" elements. By creating a List, you can store any number of elements, dynamically modifying the List, and the location of each item is managed by the List. Each element is accessible through a variable that you declare as a List.

A List in Java is an Interface. To use a List in Java, you will import the java.util library, and a List will need to be declared and initialized. A List is often implemented by the ArrayList class, but can also be implemented by one of these classes: LinkedList, Vector and Stack. If you choose ArrayList, you will also have to import ArrayList from java.util.
Example declaring a List of String:

```
List<String> myInstruments = new ArrayList<String>();
myInstruments.add("Tuba");
myInstruments.add("Trombone");
myInstruments.add("Trumpet");
myInstruments.add("Triangle");
```

Notice that unlike an Array, you can dynamically add additional elements to your List, inserting as many elements as you wish to this same List, without throwing an exception:

```
myInstruments.add("Flute");
myInstruments.add("Clarinet");
myInstruments.add("Oboe");
```

https://github.com/ckiefriter1/Java-Code-Examples/blob/main/src/collections/ListExample.java

Lists

A List is an ordered Collection (sometimes called a sequence). Lists may contain duplicate elements. In addition to the operations inherited from Collection, the List interface includes operations for the following:

- Positional access — manipulates elements based on their numerical position in the list. This includes methods such as get, set, add, addAll, and remove.
- Search — searches for a specified object in the list and returns its numerical position. Search methods include indexOf and lastIndexOf.
- Iteration — extends Iterator semantics to take advantage of the list's sequential nature. The listIterator methods provide this behavior.
- Range-view — The sublist method performs arbitrary range operations on the list.

A List:
- Is Ordered
- Preserves Insertion Order
- Allows Positional Access and Insertion of Items
- Allows Duplicates
- Is a Collection of Objects

https://github.com/ckiefriter1/Java-Code-Examples/blob/main/src/collections/ListExample.java

Sets

A Set is a Collection that cannot contain duplicate elements. It models the mathematical set abstraction. The Set interface contains only methods inherited from Collection and adds the restriction that duplicate elements are prohibited.

The Java platform contains three general-purpose Set implementations: HashSet, TreeSet, and LinkedHashSet. HashSet, which stores its elements in a hash table, is the best-performing implementation; however it makes no guarantees concerning the order of iteration.

Set:
No Duplicates
At most one Null element
Not guaranteed to be in any particular order

https://github.com/ckiefriter1/Java-Code-Examples/blob/main/src/collections/SetExample.java

Maps

A Map is an object that maps keys to values. A map cannot contain duplicate keys: Each key can map to at most one value. It models the mathematical function abstraction. The Map interface includes methods for basic operations (such as put, get, remove, containsKey, containsValue, size, and empty), bulk operations (such as putAll and clear), and collection views (such as keySet, entrySet, and values).

The Java platform contains three general-purpose Map implementations: HashMap, TreeMap, and LinkedHashMap. Their behavior and performance are precisely analogous to HashSet, TreeSet, and LinkedHashSet, as described in The Set Interface section.

https://github.com/ckiefriter1/Java-Code-Examples/blob/main/src/collections/MapExample.java

Java Wrapper Classes

Wrapper classes provide a way to use primitive data types (int, boolean, etc..) as objects.

The table below shows the primitive type and the equivalent wrapper class:

Primitive Data Type	Wrapper Class
byte	Byte
short	Short
int	Integer
long	Long
float	Float
double	Double
boolean	Boolean
char	Character

Sometimes you must use wrapper classes, for example when working with Collection objects, such as ArrayList, where primitive types cannot be used (the list can only store objects):

Menu-Driven Applications

Menu-Driven Applications

Menu-driven Applications are very useful in the coding world. They allow a program to receive data directly from a user. The user is provided a menu, and then is instructed to select an option from that menu. The chosen option will be used by the program to execute a code path (or branch) specific to that option.
In the Back End, the menus we create are text based. Adding a Client, or Front End program, which reads in user data, and then communicates to a Back End Server could provide a graphical user interface for the Menu.
Menu-driven applications are used in a variety of industries, including but not limited to computing, application development, banking (ATMs), websites, tablets, self-guided machines, word-processors, gaming, and more

Menu-Driven Application:
Advantages:
•User-friendly
- Provide guidance to the user
- No need for a user to remember commands
•Allow a user to control how and in what order a program executes

Disadvantages:
•Difficulty finding content, especially with nested sub-menus

Appendix