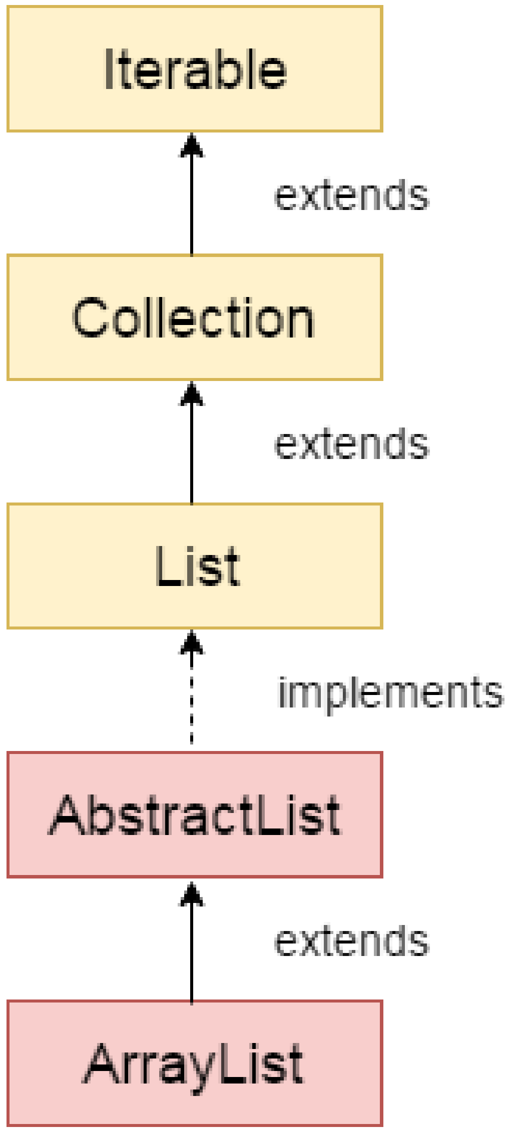
Lecture – 23

ArrayList Class

ArrayList Class

- Java **ArrayList** class uses a *dynamic array* for storing the elements
- It is available in `java.util` package
- Unlike an array it has no size limit
- We can add or remove elements anytime
- ArrayList can have duplicate elements in it



ArrayList Class

- Syntax (Generic)

```
ArrayList<BaseType> aList = new ArrayList<BaseType>();
                              OR
ArrayList<BaseType> aList = new ArrayList<>();
```

- **ArrayList** is known as a generic class with a generic type **E**.
  - Java generic collection allows you to have only one type of object in a collection
  - Java collection framework was non-generic before JDK 1.5
    - `ArrayList list=new ArrayList();`
    - It can contain any type of the elements in it

Important Points

- Java ArrayList class can contain duplicate elements.
- Java ArrayList class maintains insertion order.
- Java ArrayList allows random access because the array works on an index basis.
- In ArrayList, manipulation is a little bit slower than the LinkedList in Java because a lot of shifting needs to occur if any element is removed from the array list.
- We can not create an array list of the primitive types

ArrayList Class

- ArrayList can be created in 3 ways.
  - **ArrayList()** —> It creates an empty ArrayList with initial capacity of 10.
    - `ArrayList<Integer> list1 = new ArrayList<Integer>();`
  - **ArrayList(int initialCapacity)** —> It creates an empty ArrayList with supplied initial capacity.
    - `ArrayList<String> list2 = new ArrayList<String>(20);`
  - **ArrayList(Collection c)** —> It creates an ArrayList containing the elements of the supplied collection.
    - `ArrayList<Integer> list3 = new ArrayList<Integer>(list1);`

ArrayList Methods

java.util.ArrayList<E>	
`+ArrayList()`	Creates an empty list.
`+add(o: E): void`	Appends a new element o at the end of this list.
`+add(index: int, o: E): void`	Adds a new element o at the specified index in this list.
`+clear(): void`	Removes all the elements from this list.
`+contains(o: Object): boolean`	Returns true if this list contains the element o.
`+get(index: int): E`	Returns the element from this list at the specified index.
`+indexOf(o: Object): int`	Returns the index of the first matching element in this list.
`+isEmpty(): boolean`	Returns true if this list contains no elements.
`+lastIndexOf(o: Object): int`	Returns the index of the last matching element in this list.
`+remove(o: Object): boolean`	Removes the first element o from this list. Returns true if an element is removed.
`+size(): int`	Returns the number of elements in this list.
`+remove(index: int): boolean`	Removes the element at the specified index. Returns true if an element is removed.
`+set(index: int, o: E): E`	Sets the element at the specified index.

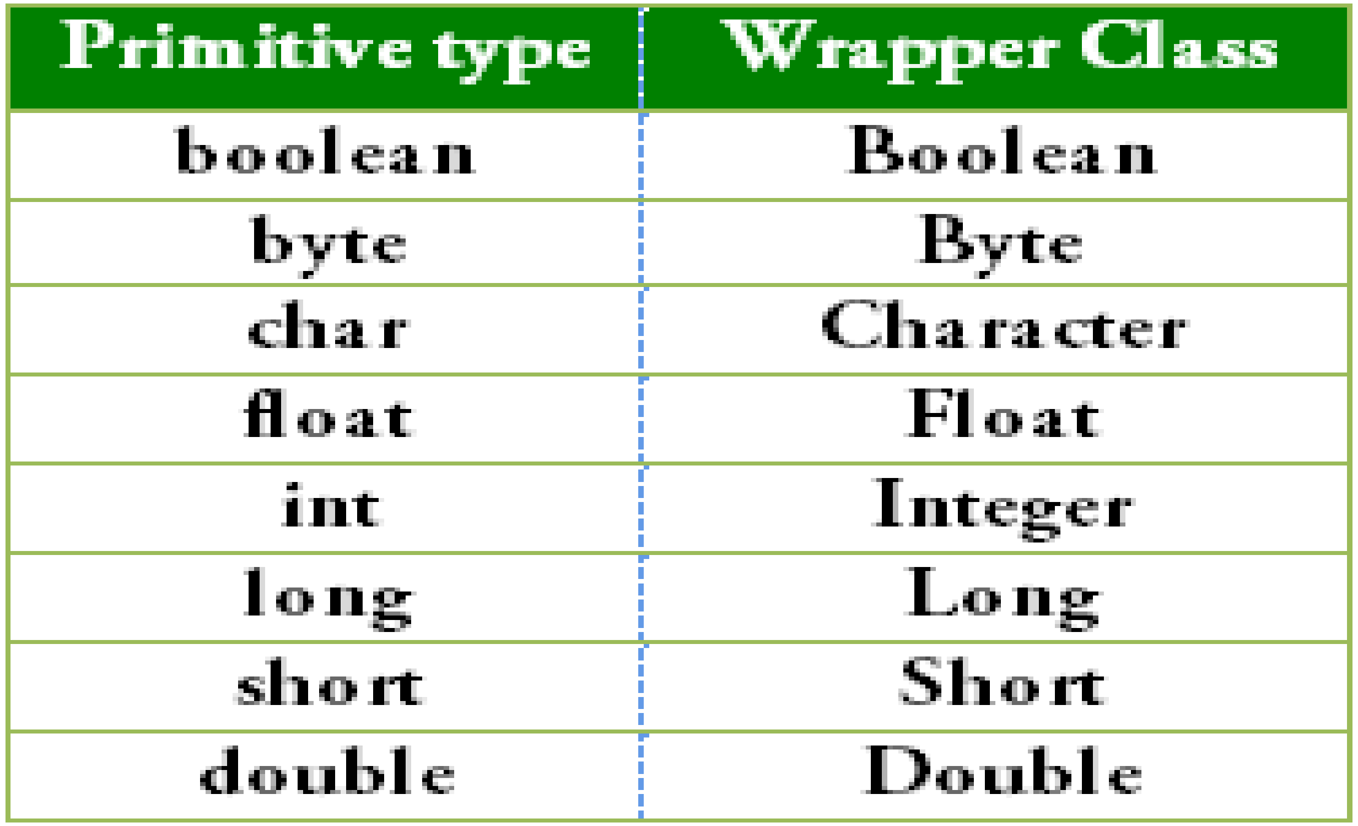
Differences and Similarities between Arrays and **ArrayList**

Operation	*Array*	*ArrayList*
Creating an array/ArrayList	String[] a = **new** String[**10**]	ArrayList<String> list = **new** ArrayList<>();
Accessing an element	a[index]	list.get(index);
Updating an element	a[index] = **"London"**;	list.set(index, **"London"**);
Returning size	a.length	list.size();
Adding a new element		list.add(**"London"**);
Inserting a new element		list.add(index, **"London"**);
Removing an element		list.remove(index);
Removing an element		list.remove(Object);
Removing all elements		list.clear();

Wrapper Classes

- *Wrapper classes* provide a class type corresponding to each of the primitive types
  - This makes it possible to have class types that behave somewhat like primitive types
  - The wrapper classes for the primitive types **`byte`**, **`short`**, **`long`**, **`float`**, **`double`**, and **`char`** are (in order) **`Byte`**, **`Short`**, **`Long`**, **`Float`**, **`Double`**, and **`Character`**
- Wrapper classes also contain a number of useful predefined constants and static methods

Wrapper Classes



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

Wrapper Classes

- In Java, **boxing** and **unboxing** are terms used to describe the conversion between primitive data types and their corresponding wrapper classes.
- **Boxing** is the process of converting a primitive type into its corresponding wrapper class.
  - For example, converting an int to an Integer or a double to a Double.
  - Boxing is automatically handled by the Java compiler

```
int num = 10;
Integer wrappedNum = num; // Boxing: int to Integer automatically
//OR
Integer myInt = Integer.valueOf(num);//converting int into Integer explicitly
```

Wrapper Classes

- Unboxing, is the process of converting a wrapper class object back to its corresponding primitive type.
  - It allows you to extract the value from the wrapper object.
  - Unboxing is also automatically handled by the Java compiler.

```
Integer wrappedNum = 20;
int num = wrappedNum; // Unboxing: Integer to int OR
Int num = wrappedNum.intValue(); //converting Integer to int explicitly
```

Wrapper Classes

- ArrayList <Integer> list = new ArrayList<Integer> (19);
- List.add(new Integer(6));
- List.add(6);
- Integer x_obj= list.get(0);
- int x = x_obj.intValue();
- int x = list.get(0);

Automatic Boxing and Unboxing

- ArrayList<Integer> array = new ArrayList<Integer> (2);
-         array.add(5);
-         array.add(6);
-         int x = array.get(1);