Unit 3 AND 4 Study Guide – ArrayList + Constructors

3.2 ArrayList

1. What is ArrayList, and how is it different from Arrays -> Array vs. ArrayList

ArrayList is a class-> Part of the collection framework
-> different collections help to work with many data

The class is located in java.util package

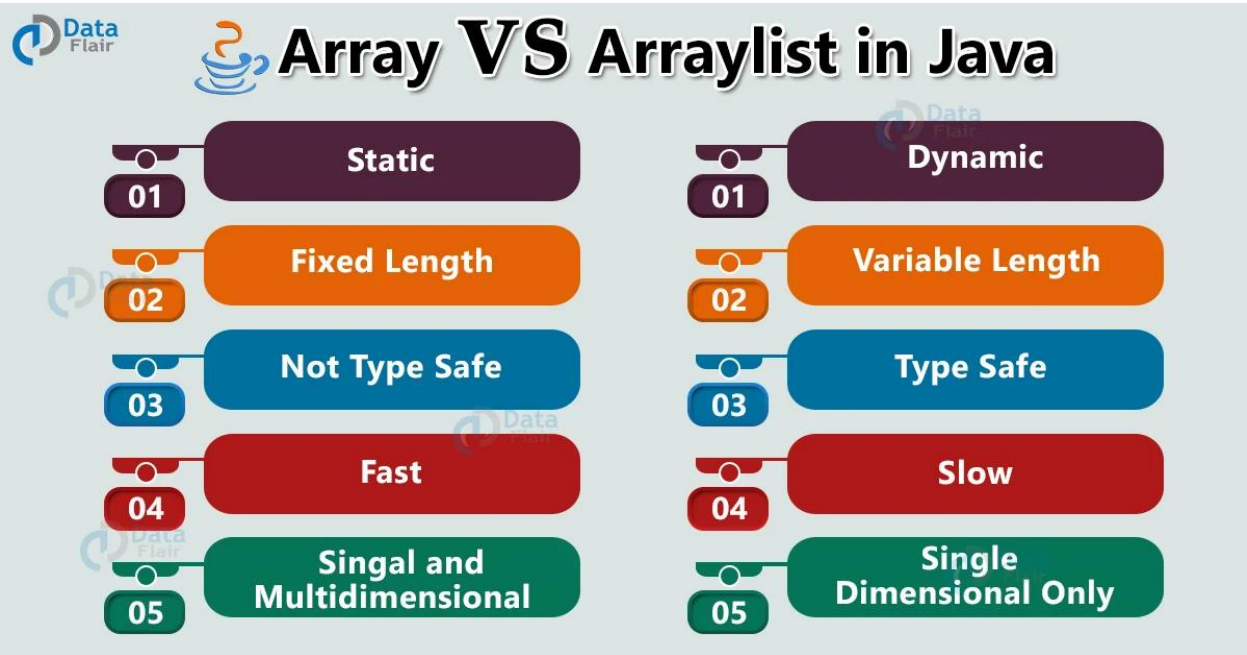
ArrayList uses arrays internally.

ArrayList allows duplicate elements.
ArrayList is ordered. You can access the elements by indexes. The order of the elements is known(insertion order)

-> ArrayList is a dynamic array
-> in ArrayList, the size is NOT fixed
the size will grow and shrink when needed

SYNTAX:

```
ArrayList<DATATYPE> name = new ArrayList<>();
```



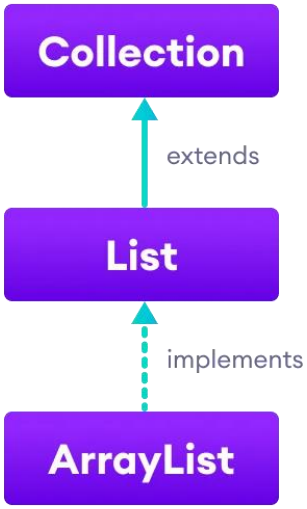
Array

- fixed size
- use both used with primitive & non-primitive types

- *object type*
- *no methods*
- *in general, takes less memory*
- *We need to use Arrays.toString() to print the array*
- *java.util.Array is a class*
- *No need to box and unbox the elements*

ArrayList

- dynamic size
- only use object types (nonprimitive types)
- has methods to use the data
- in general, it takes more memory
- can be printed using only print statement, no need for anything else
- *Needs to box and unbox the elements*



SUMMARY

- *The array is a part of core Java programming, and ArrayList is part of the collection framework.*
- *The significant difference is that Array is a fixed length data structure, so we can't change length, but ArrayList is re-sizeable.*
- *The other major one is that Array can contain both primitive and object elements, but ArrayList can only contain objects.*
- *It cannot contain primitive types.*

CREATING AN ARRAYLIST

```
ArrayList<type> name = new ArrayList<>();

      creates an empty ArrayList

ArrayList<type> name = new ArrayList<>( collection type );
```

collection type: there is a collection framework that has many components
ArrayList is a collection type

```
ArrayList<type> name = new ArrayList<>( arraylist object );
```

Using Arrays.asList()

parameters: an array (var arg)
returns: collection type

Use both parts together to create an ArrayList with values

```
ArrayList<type> name = new ArrayList<>( Arrays.asList() );
Ex: .asList("hello", "world", "Friday")
    .asList(3, 4, 5, 1, 5, 5)
```

Use **Arrays.asList()** method

1. Creating ArrayList of integers with values

```
List<Integer> listOfInteger = new ArrayList<>(Arrays.asList(1, 2, 3, 4, 5));
```

2. ArrayList of Strings with values

```
List<String> listOfString = new ArrayList<>(Arrays.asList("Ron", "Hermoine", "Harray");
```

3. ArrayList of floats with values

```
List<Float> listOfFloat = new ArrayList<>(Arrays.asList(3.12f, 3.23f, 6.32f));
```

2. for each with ArrayList

ForEach loop in ArrayLists

→ to apply the for each loop to ArrayList

```
for(Integer each : numbers){
    //code here
}
```

// can also be int each: numbers <- must match the data type in the ArrayList

→ the iteration order of the loop is fixed and depends on the elements of the ArrayList, it cannot reverse or alter the indexes in any way

→shortcut
numbers.for -> this generates the for each loop for the ArrayList

```
list.forEach(x -> System.out.println(x));
```

-> above is an example of the Lambda forEach() method

3. The main ArrayList methods (add, remove, get, set) ArrayList Methods:

size(): returns the number of elements

return int

last index: size() - 1

add(element): adds the element to the end of the ArrayList
return boolean -> returns true; we don't usually use this return value.

add(int, element): adds the element at the given index number. All the other elements move in position.

get(int): return the element at the given index number
return element

clear(): removes all the elements from the ArrayList

remove(int): removes the element at the given index. All elements shift over
-> returns the element that is removed
-> if you try to remove an index that doesn't exist out of bounds

remove(element): remove the element defined from the ArrayList.
returns boolean
true: if the element was removed
false: if the element was not removed - if the element did not exist

isEmpty(): checks if there is any element in the ArrayList
returns boolean
true: ArrayList is empty; no elements
false: if there are any elements

contains(element): checks if the ArrayList has the given element
returns boolean
true: ArrayList has the element
false: ArrayList does not have the element

equals(ArrayList) checks if the given ArrayList is equal to the ArrayList using the method
return boolean
true: if the two ArrayList are the same, which means they have the same elements, same order
false: if the ArrayList is not the same
set(int, element): *replace the element at the given index with the given new element*

BULK OP. Methods

The arguments of each method are collection types.

Collections.addAll(collection type): add all the elements from the given argument to the ArrayList

Collections.removeAll(collection type): removes every occurrence of every element from the given argument

Collections.containsAll(collection type): checks if every element from the given argument is in the ArrayList

Collections.retainAll(collection type): keeps all the defined elements but deletes all the others

4.3 Constructors

1. What is the purpose of the constructor in a class?

Definition

- a special method that is called when an object is created
  it is called in relation to new operator(how objects are created)
  The constructor is used mainly to initialize our instance variables.

2. Constructor's relation with new

- new keyword is used to create an instance of the class
- it instantiates a class by allocating memory for a new object and returning a reference to that memory
- *The object of a class can be created by using the **new** keyword.*

3. The rules of creating constructors

- You can use public, protected & private access specifiers with constructors.
- *The Java compiler provides a default constructor if we do not have any constructor.*

CREATING CONSTRUCTORS

o a particular method that matches the name of the class and has NO RETURN type nor a specifier

TYPES OF CONSTRUCTORS -> NO-ARG

o constructor that has no parameter
o is known as a default constructor
o if we DO NOT define a constructor as a class, then a compiler creates a default constructor

```
class Test{
 /* Added by the Java Compiler at the Run Time
 public Test(){
 }
 */
 public static void main(String args[]) {
   Test testObj = new Test();
  }
}
```

PARAMETERIZED

- constructors that have parameters
- if we want to initialize the fields(instance variables) of the class with our values, then pass the parameters to the constructors

SYNTAX:

```
public class Test {
   int appId;
   String appName;
//parameterized constructor with two parameters
Test(int id, String name){
   this.appId = id;
   this.appName = name;
}
void info(){
   System.out.println("Id: "+appId+" Name: "+appName);
}

public static void main(String args[]){
   Test obj1 = new Test(11001,"Facebook");
   Test obj2 = new Test(23003,"Instagram");
   obj1.info();
   obj2.info();
 }
}
```

The use of super()

Constructors use super to invoke the superclass's constructor. If a constructor uses super, it must use it in the first line; otherwise, the compiler will complain.

Using this keyword with a Constructor

From within a constructor, you can also use the this keyword to call another constructor in the same class. Doing so is called an explicit constructor invocation. Here's another Rectangle class, with a different implementation from the one in the Objects section.

```
public class Rectangle {
    private int x, y;
    private int width, height;

    public Rectangle() {
        this(0, 0, 1, 1);
    }
    public Rectangle(int width, int height) {
        this(0, 0, width, height);
    }
    public Rectangle(int x, int y, int width, int height) {
        this.x = x;
        this.y = y;
        this.width = width;
        this.height = height;
    }
    ...
}
```

This class contains a set of constructors. Each constructor initializes some or all of the rectangle's member variables. The constructors provide a default value for any member variable whose initial value is not provided by an argument. For example, the no-argument constructor creates a 1x1 Rectangle at coordinates 0,0. The two-argument constructor calls the four-argument constructor, passing in the width and height but always using the 0,0 coordinates. As before, the compiler determines which constructor to call, based on the number and the type of arguments.

4. What is the default constructor, and what happens if you create a constructor manually?

-> if we don't make a constructor, the compile will generate a default constructor, which is a no-argument constructor

-> The compiler doesn't ever enforce the existence of a default constructor. You can have any constructor as you wish

-> if you define another constructor (with arguments), the default constructor will not be generated. If you still want one, you need to define it yourself.

-> you can create objects without any default constructor defined in your class; there comes the concept of auto-generated default constructor