# Object-Oriented Development (CIS1056-N) Worksheet 09: Java Collections

## Before You Start

Remember: You are not expected to complete the entire brief within the allotted two hours, but to make a start and continue outside of the class.

## Introduction

### Arrays

An array is a container object that holds a fixed number of values of a single type. The **length** of an array is **established** when **the array is created**. After creation, **its length is fixed**. Each item in an array is called an *element*, and each *element* is accessed by its numerical index.



Illustration of an array as 10 boxes numbered 0 through 9; an index of 0 indicates the first element in the array. As shown in the preceding illustration, numbering begins with 0. The 9th element, for example, would therefore be accessed at index 8.

A handy introduction to arrays can be found at the [Oracle Java Documentation](https://docs.oracle.com/javase/tutorial/java/nutsandbolts/arrays.html) site.

### ArrayList

An ArrayList is a resizable-array implementation of the *List* interface found in the java.util package.

ArrayLists allow elements to be added and removed at runtime, with the size of the ArrayList growing to accommodate new elements (unlike an array that requires a new instance with increased size to be created).

Click the following link to see the Oracle documentation for an [ArrayList](https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/util/ArrayList.html), including a full list of methods.

## 1. Array Basics

Create a new NetBeans project called **ArrayDemo**.

1. Add a new class called ArrayDemo with a main method. Inside the main method declare an integer array with a size of 10 elements:

public class ArrayDemo {

    public static void main(String[] args) {

        int[] numbers = new int[10];

    }

}

1. Now set the value of all elements to be 1. You could do this individually (but that would be bad practice! Use a for loop like so:

        for (int i=0; i < numbers.length; i++)

            numbers[i] = 1;

1. Note the use of numbers.length instead of integer literal 10. Why is numbers.length preferable?
2. Set the first element to the value 5. Remember, arrays are indexed from 0.

numbers[0] = 5;

1. Set the final element to value 42. We could use a fixed integer value (9) for the index, or an expression such as numbers.length-1.

        numbers[numbers.length-1] = 42;

1. Choose at least 4 other elements in the array and assign different integer values.
2. Print each element to console using a for loop.

        for (int num: numbers){

            System.out.println(num);

        }

1. Iterate through the array and provide a sum of all values. Use pen and paper to ensure this sum is correct.

## 2. Array Times Tables

Add a new class ArrayTimesTables to the **ArrayDemo** project and create a main method.

1. Declare an integer array timesTable but do not initialise it.

int[] timesTable;

1. Prompt the user to enter 2 integer values: length and multiplier. Use a Scanner object to do this.
2. Create an instance of timesTable with the array size set to length.   
     
    timesTable = new int[length];
3. Iterate over timesTable, setting the value of each element to:

currentIndex \* multiplier

1. Print each element to the console using a for loop.

## 3. ArrayList Basics

Add a new class ArrayListBasics to the **ArrayDemo** project and create a main method.

1. Create an ArrayList object called colours that will store Strings:

import java.util.ArrayList;

public class ArrayListBasics {

    public static void main(String[] args) {

        ArrayList<String> colours = new ArrayList<>();

    }

}

1. To add an element to an ArrayList we use the add method. "Red" has been added below, you should also add "Green" and "Blue".

        colours.add("Red");

        System.out.println(colours);

1. To retrieve an element, use the get method and specify an index. Index numbers work in the same way as with arrays. To retrieve the first element use.

        colours.get(0);

1. To modify an existing element, use the set method. We must pass the index of the element to update, along with the element itself. To set the third element to "Teal" use:

        colours.set(2, "Teal");

1. To retrieve the number of elements in the ArrayList use the size method. To print the current size use:

        System.out.println(colours.size());

1. Individual elements can be removed from a list using the remove method by passing the index of the element to remove. To remove the second element use:

        colours.remove(1);

1. To remove all elements, we use the clear method:

        colours.clear();

## 4. Car Sales

Create a new NetBeans project called **CarSales**. Add the following basic Car class.

public class Car {

    private String make;

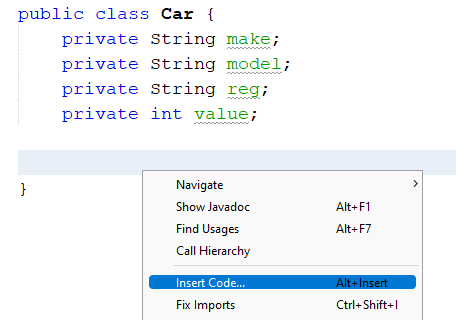
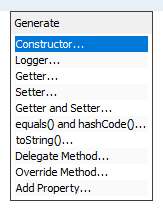
    private String model;

    private String reg;

    private int value;

}

Using NetBeans Insert Code… feature right click to add a **Constructor** that sets each property, **Getter and Setter** for each property, and a **toString()** method.



If it does not already exist, add a new class CarSales to the **CarSales** project. Remove the main method if there is one.

Add the following code:

package carsales;

import java.util.ArrayList;

public class CarSales {

    private ArrayList<Car> cars;

    public CarSales() {

        cars = new ArrayList<>();

    }

    public void addCar(String make, String model, String reg, int value) {

        Car car = new Car(make, model, reg, value);

        cars.add(car);

    }

    public void removeCar(String reg) {

        // Remove a car by iterating through the car list to

        // find a matching registration, then removing.

        // Note that an ArrayList is NOT the best collection to be

        // using if we intend to do this sort of operation often.

    }

    public void clearCars() {

        // Tax man is onto us, empty the inventory!

    }

    public int totalValue() {

        // Iterate through all cars summing the value

        return 0;

    }

    @Override

    public String toString() {

        return "CarSales{" + "cars=" + cars + '}';

    }

}

Now add the following class to drive our solution

package carsales;

public class SalesHarness {

    public static void main(String[] args) {

        CarSales sales = new CarSales();

        sales.addCar("Ford", "Focus", "NU70 YBX", 20000);

        sales.addCar("Nissan", "Micra", "YP53 HZZ", 1100);

        System.out.println(sales.toString());

    }

}

Improve the solution by implementing the missing methods in CarSales. Improve the presentation of the string output provided by toString() method(s).

Inside SalesHarness, create a menu system that allows operators to record sales. You should provide at least the following options:

1. List all existing sales.
2. Add sale. User must then specify make, model, reg, and value.
3. Remove sale. User must then specify reg to remove.
4. Print total value.
5. Remove all.

Test your final solution for robustness.

## 5. Faulty Iteration

Examine the following code:

**public class Main {**

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

**Random random = new Random();**

**ArrayList<Integer> listOfIntegers = new ArrayList<>();**

**for(int n = 0; n < 100; n++) {**

**listOfIntegers.add(random.nextInt(200));**

**}**

**for(Integer i : listOfIntegers) {**

**if(i % 15 == 0) {**

**listOfIntegers.remove(i);**

**continue;**

**}**

**System.out.println(i);**

**}**

**}**

**}**

Unfortunately, it is unreliable because it crashes.

1. Identify the cause of the problem and explain it to your tutor.
2. Write an alternative solution to remove an element from a list whilst iterating over it.

**Hint**: Review the lecture slides.

## 6. More Java Collections

ArrayList is just one of many collections in Java. As you progress through your course you will be exposed to more complex collections. For those who have programmed before, explore the [Java Collections Framework](https://docs.oracle.com/javase/tutorial/collections/intro/index.html).

## 7. Stacks (LIFO data structure)

At the end of the Collections lecture, you were introduced to Stacks, a **L**ast-**I**n-**F**irst-**O**ut (LIFO) collection type.

Create a new Java Netbeans project and implement the stack example to reverse the input – you will find the examples on **slide 28** (stack behaviour animation) and **slide 29** (the code).

You will need a test harness class that contains the main method with the code on **slide 29**.

Stack documentation: <https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/util/Stack.html>

## 8. Stacks – Balanced Brackets

The final example in the lecture about collection types was about how a stack could be used to check whether an input string was balanced, meaning corresponding bracket symbols (i.e. **[]**, **(),** **{}**, etc.) were correctly matched. Nested brackets are permitted, but each close backet must match the most recent open bracket – for example, if the last open bracket was a **[** then the close bracket must be a **]**.

This is an exercise about processing data in a Last-In-Last-Out (LIFO) manner using a Stack – other solutions are possible.

1. Using pseudocode, develop a paper solution to this problem. Remember that stack operations include:
   1. **push()** – place a new item at the top of the stack.
   2. **pop()** – remove and return the item at the top of the stack.
   3. **peek()** – examine, but do not remove the item at the top of the stack.
   4. **empty()** – check if the stack is empty or not.
   5. **size()** – get the number of elements currently in the stack.
2. Once you’re satisfied with your pseudocode solution, create a new Java Netbeans solution for it.
3. Test your solution with a variety of inputs, including empty input, mismatching and matching brackets, simple/complex/long/short combinations of brackets and non-bracket input.

## 9. Stacks – Balanced Brackets (again)

You are required to keep track of any data (i.e. not brackets) along with the opening bracket type. For example, if the input is:

**<HELLO, WORLD![1234(ABC)]>**

Then the stack would look like this prior to popping data from it:

TOP

|  |  |
| --- | --- |
| **(** | **ABC** |
| **[** | **1234** |
| **<** | **HELLO, WORLD!** |

When popping the data from the stack, print out the non-bracket data only, for example:

**ABC**

**1234**

**HELLO, WORLD!**

Finally, you’d display whether the input was balanced or not.

Another example, given the input:

**<HELLO, WORLD![1234(ABC)[5678<DEF>]]>**

The output would be:

**ABC**

**DEF**

**5678**

**1234**

**HELLO, WORLD!**

**<HELLO, WORLD![1234(ABC)[5678<DEF>]]> is balanced.**

## 10. Start the ICA

If you have not already done so, begin the ICA. A good place to start would be to write methods that iterate through the provided text files, printing each line to the console.

You have several keyboard inputs you must provide. Put together methods that take and then validate these inputs.

## Document History

Revision 0 (21-Nov-22): This is the initial version of the 2022/23 exercise.