SLIIT UNI
THE KNOWLEDGE UNIVERSITY

Sri Lanka Institute of Information Technology

Faculty of Computing

SE1020 - Object-Oriented Programming

Year 01 and Semester 02

Lecture 08

ArrayList

Recall. . .

- Understand different types of errors and exceptions.
- Differentiate between compile-time and run-time errors.
- Describe the Java exception hierarchy.
- Use try-catch-finally for exception handling.
- Create and use custom exceptions.

Learning Objectives

- Explain the limitations of fixed-size arrays in Java and the need for a dynamic data structure.
- Describe the ArrayList class within the Java Collections Framework, including its purpose and key characteristics.
- Demonstrate how to perform core operations on an ArrayList, creating one, adding, accessing, modifying, and removing elements, as well as checking size and emptiness.
- Implement iteration over an ArrayList using both the classic for loop and the enhanced for (foreach) loop.
- Understand and apply Java Generics to enforce compile-time type safety in collections, recognizing that generics work only with reference types and employing wrapper classes as needed.

Exercise 01 - Dynamic Registration List

Scenario: You are organizing a tech workshop where attendees register on arrival.

Requirements:

- Add each new attendee immediately.
- Remove cancellations quickly.
- Maintain the order of arrival.
- Retrieve any attendee by their registration position.

Implementing the scenario with a String array

```java
public class GuestList {
    public static void main(String[] args) {
        String[] guests = new String[5];
        guests[0] = "Amali Gunasekara";
        guests[1] = "Saman Rathnayake";
        guests[2] = "Kalani Gamage";
        guests[3] = "Kamal Rathnayake";
        guests[4] = "Roshel Fernando";

        guests[5] = "Nihal Perera";

        for (int i = 0; i < guests.length; i++) {
            System.out.println("Guest " + (i + 1) + guests[i]);
        }
    }
}
```

Output :

```
C:\Users\thilini.j\.jdks\openjdk-23.0.2\bin\java.exe "-javaagent:D:\IntelliJ IDEA 2024.3.2.2\lib\idea_rt.jar=64241:D:\I
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException Create breakpoint : Index 5 out of bounds for length 5
    at GuestList.main(GuestList.java:10)

Process finished with exit code 1
```

Challenges with a fixed-size array

- Arrays are of fixed length. You can not change the size of the arrays once they are created.
- You can not accommodate an extra element in an array after they are created.
- Memory is allocated to an array during it's creation only, much before the actual elements are added to it.
- You must know the exact number of attendees in advance.
- Resizing requires creating a new array and copying elements.

How would you implement these requirements using Java?

ArrayList Class

- ArrayList is an in-built class under Java Collections Framework.
- ArrayList support dynamic arrays (can increase and decrease size dynamically).
- Elements can be inserted at or deleted from a particular position.
- ArrayList class has many methods to manipulate the stored objects.
- If **Generics** are not used, ArrayList can hold any type of objects.

Implementing the scenario with a ArrayList

```java
import java.util.ArrayList;

public class GuestList {
    public static void main(String[] args) {
        ArrayList guests = new ArrayList();
        guests.add("Amali Gunasekara");
        guests.add("Saman Rathnayake");
        guests.add("Kalani Gamage");
        guests.add(1, "Kamal Rathnayake");
        guests.add("Roshel Fernando");
        guests.add("Nihal Perera");

        for (int i = 0; i < guests.size(); i++) {
            System.out.println("Guest " + (i + 1) + " - " + guests.get(i));
        }

        //Display the entire arraylist
        System.out.println("\nPrinting the entire array");
        System.out.println(guests);
    }
}
```

Implementing the scenario with a ArrayList

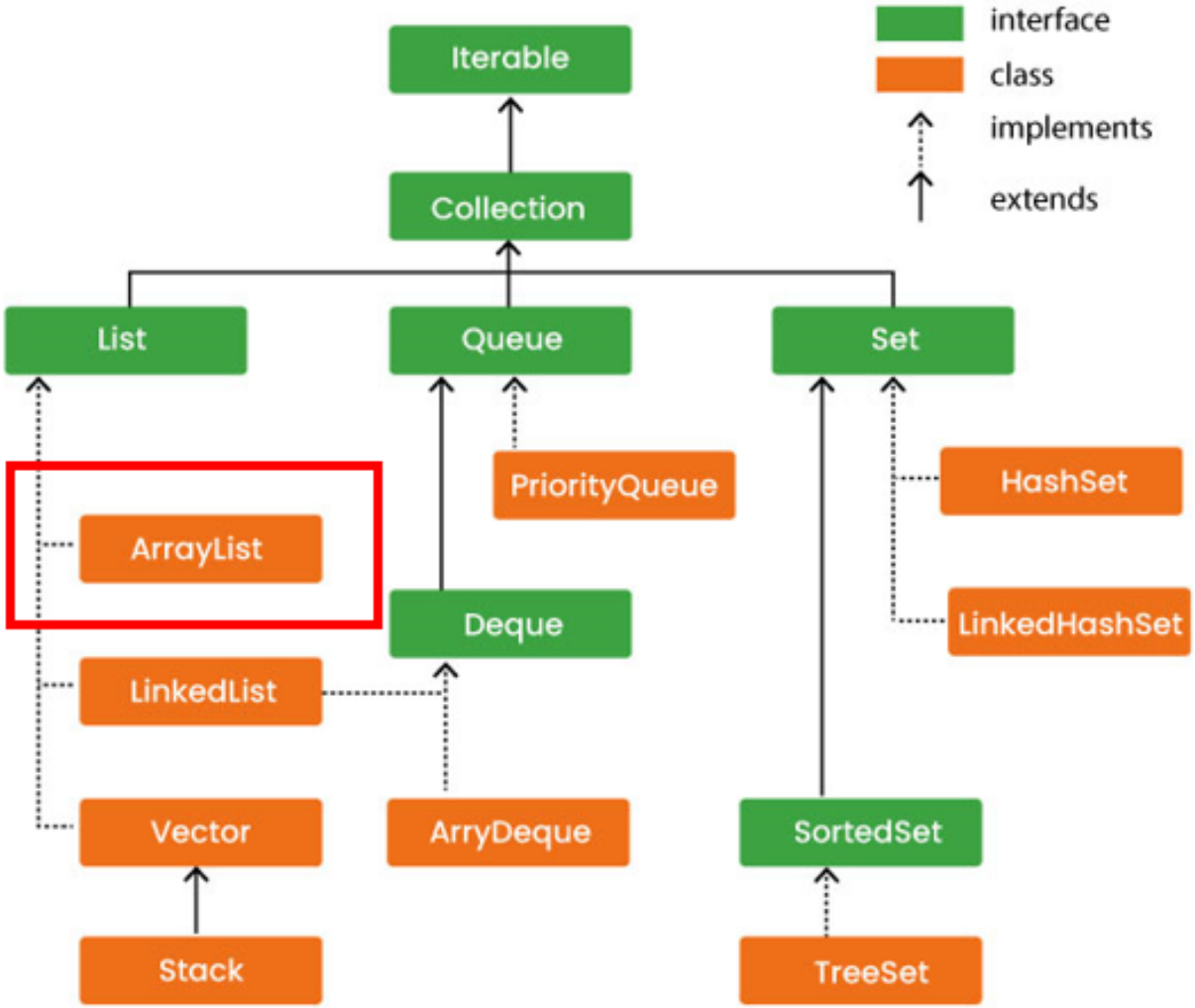
Output :

```
C:\Users\thilini.j\.jdks\openjdk-23.0.2\bin\java.exe "-javaagent:D:\IntelliJ IDEA 2024.3.2.2\lib\idea
Guest 1 - Amali Gunasekara
Guest 2 - Kamal Rathnayake
Guest 3 - Saman Rathnayake
Guest 4 - Kalani Gamage
Guest 5 - Roshel Fernando
Guest 6 - Nihal Perera

Printing the entire array
[Amali Gunasekara, Kamal Rathnayake, Saman Rathnayake, Kalani Gamage, Roshel Fernando, Nihal Perera]

Process finished with exit code 0
```

Introduction to Collection Framework

- A framework is a set of classes and interfaces that provide a ready-made architecture.
- A collection is an object that represents a group of objects (such as the classic ArrayList class).
- All classes and interfaces for the collection framework are available in **java.utli package**.
- The following diagram shows the hierarchy of the collection framework in Java:

Hierarchy of Collection Framework



How to use an ArrayList?

- **Step 01 : Import ArrayList class**

```java
//Step 01 : Import ArrayList class
import java.util.ArrayList;

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

    }
```

- **Step 02 : Create/Declare an object from ArrayList class**

```java
public class Test {
    public static void main(String[] args) {

        //Step 02 : Create/Declare an object from ArrayList class
        ArrayList testArray = new ArrayList();
```

How to use an ArrayList?

- **Step 03 : Add elements (objects) to the created arraylist object**

```
ArrayList testArray = new ArrayList();

//Step 03 : Add elements (objects) to the created arraylist object

//Adding the elements according to the insertion order
testArray.add("Amali Gunasekara");
testArray.add(85.5);
testArray.add(false);

//Adding the elements to a particular index
testArray.add(2, 'G');
testArray.add(4, 27);
testArray.add(5, "Kamal");
```

- **Step 04 : Modifying Elements**

```
//Step 04 : Modifying Elements
testArray.set(0, 200.56);
```

How to use an ArrayList?

- **Step 05 : Accessing Elements**

```
//Step 05 : Accessing Elements
System.out.println(testArray.get(2));
```

- **Step 06 : Removing Elements**

```
//Step 06 :Removing Elements
testArray.remove(85.5); // by value
testArray.remove(2); // by index
```

- **Step 07 : Retrieve Size & Emptiness**

```
//Step 07 :Retrive Size & Emptiness
int sizeOfTestArray = testArray.size();
boolean emptiness = testArray.isEmpty();
```

How to use an ArrayList?

- **Step 08 : Iteration - Classic For Loop**

```java
//Step 08 : Iteration - Classic For Loop
for (int i = 0; i < testArray.size(); i++) {
    System.out.println(testArray.get(i));
}
```

- **Step 09 : Iteration - Enhanced For Loop**

```java
//Step 09 : Iteration - Enhanced For Loop
for (Object obj : testArray) {
    System.out.println(obj);
}
```

Generics

- Generics means parameterized types. The idea is to allow a type (like Integer, String, etc., or user-defined types) to be a parameter to methods, classes, and interfaces.

Why Generics?

- Before Generics, the Java developers used Object to store any type of data. The Object is the superclass of all other classes, and an Object reference can refer to any object. These features lack type safety. For example, if you want to restrict the arraylist to hold only the Integer values, we need Generics to add that type of safety feature.

Limitations: Generics Work Only with Reference Types

- When we declare an instance of a generic type, the type argument passed to the type parameter must be a reference type. We cannot use primitive data types like int, char.

```
ArrayList<int> guests = new ArrayList<int>();
```

The above line results in a compile-time error that can be resolved using type wrappers to encapsulate a primitive type. To fix it, use the corresponding wrapper class name inside the angle brackets

```
ArrayList<String> guestsList = new ArrayList<String>();
ArrayList<Integer> NumberList = new ArrayList<Integer>();
```

Wrapper class / Covering class

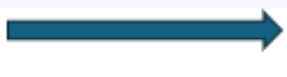
- Wrapper class in java provides the mechanism to convert primitive datatype into object and object into primitive type.
- For each primitive datatype, there exists a covering class in the java.lang package.

byte	→	Byte
short	→	Short
int	→	Integer
long	→	Long
float	→	Float
double	→	Double
char	→	Character
Boolean	→	Boolean

Exercise 02 - Dynamic Registration List

In the earlier exercise, restrict the ArrayList to hold only the String values.

```java
import java.util.ArrayList;

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

        ArrayList<String> guests = new ArrayList<String>();
        guests.add("Amali Gunasekara");
        guests.add("Saman Rathnayake");
        guests.add("Kalani Gamage");

        guests.add(100);

        for (int i = 0; i < guests.size(); i++) {
            System.out.println("Guest " + (i + 1) + " - " + guests.get(i));
        }

        //Display the entire arraylist
        System.out.println("\nPrinting the entire array");
        System.out.println(guests);
    }
}
```



Now, if you try to add a value other than type String, it will display as a compile time error

Exercise 03

Create a class called **Student** with related attributes, a default constructor, and a displayDetails() method to print the values of the attributes.

Implement another class called **StudentApp** with a main method. Create an ArrayList which accommodates only the Student type objects to be saved.

Create a few Student objects and add those to the created ArrayList.

Student Class

```java
public class Student {  9 usages
    private String name;  2 usages
    private String StdId;  2 usages
    private int batch;  2 usages

    public Student(){  3 usages
        this.name = "test name";
        this.StdId = "test student id";
        this.batch = 0;
    }

    public void dispalyDetails(){  1 usage
        System.out.println(this.name);
        System.out.println(this.StdId);
        System.out.println(this.batch);
    }
}
```

StudentApp Class

```java
import java.util.ArrayList;

public class StudentApp {
    public static void main(String[] args) {
        ArrayList<Student> studentList = new ArrayList<Student>();

        Student s1 = new Student();
        Student s2 = new Student();
        Student s3 = new Student();

        studentList.add(s1);
        studentList.add(s2);
        studentList.add(s3);

        for(Student student : studentList) {
            student.dispalyDetails();
            System.out.println();
        }
    }
}
```

Thank You!