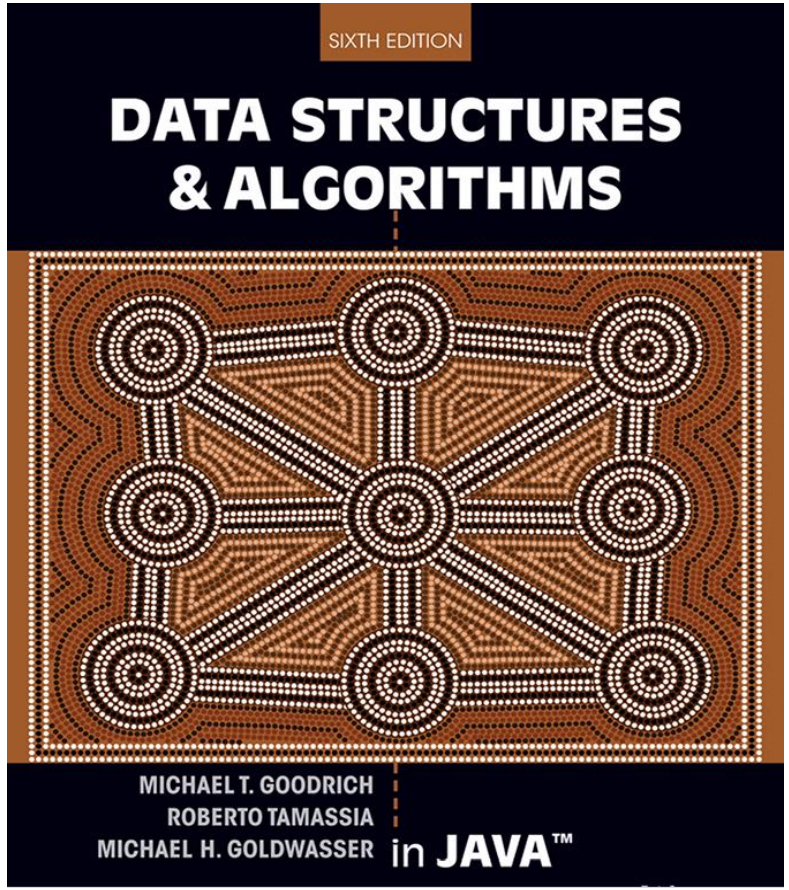
DATA STRUCTURES & ALGORITHMS



Data Structures & Algorithms

Lecture 4

List

BTEC Alliance with FPT Education

Pearson

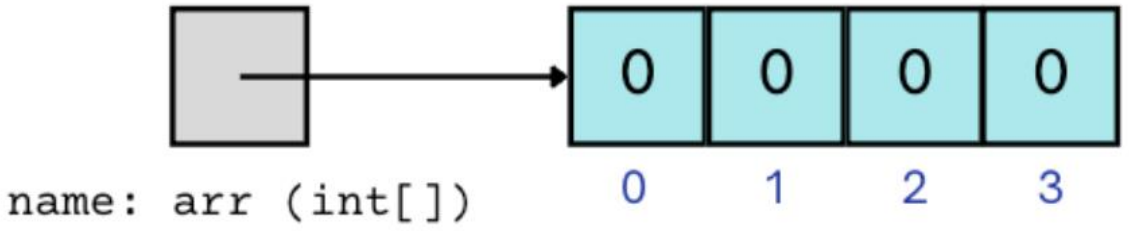
Topics

- Introduction to ArrayList
- Typed ArrayList
- ArrayList Operations
- Collections Utility Class
- Iterator

Arrays

- The type of an array is **ElementType[]**
  - ElementType can be any type!
- Can store multiple elements of the same type
- Need to specify length of array and type of elements it will store at creation

```
int[] arr = new int[4];
```



Disadvantages of Arrays

The array has a fixed number of elements. Therefore, it has the following disadvantages:

- It is not possible to add or remove elements
- Wastes memory:
  - If declaring an array with a large size to hold a few elements.
  - Declaring an array with a small size is not sufficient to accommodate.

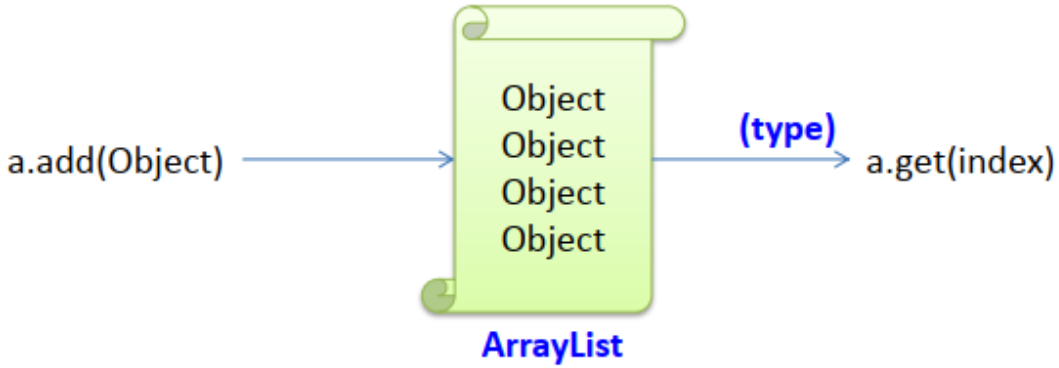
ArrayList

- Java **ArrayList** class uses a *dynamic array* for storing the elements. It is like an array, but there is *no size limit*. We can add or remove elements anytime. So, it is much more flexible than the traditional array. It is found in the *java.util* package
- ArrayList also allows performing set operations such as **union**, **intersection**, **difference…**
- It inherits the **AbstractList** class and implements *List interface*.

ArrayList

```
ArrayList a = new ArrayList();
a.add("Jisoo");
a.add(true);
a.add(1);
a.add(2.5)
Integer x = (Integer)a.get(2);
```

- When adding primitive integers, they are automatically converted to **wrapper** object types.
- When accessing elements, it is necessary to **cast back to the original type** of the element for processing.



Typed ArrayList

ArrayList (Untyped)

An ArrayList can contain elements of any data type.

+ The elements in an ArrayList are treated as a collection of objects (of type **Object**).

+ When accessing elements, it's necessary to **cast back to the original type** of the element for processing.

ArrayList<Type> (Typed)

An ArrayList only contains elements of the specified type.

+ When accessing elements, there is **no need to cast back to the original type** of the element for processing

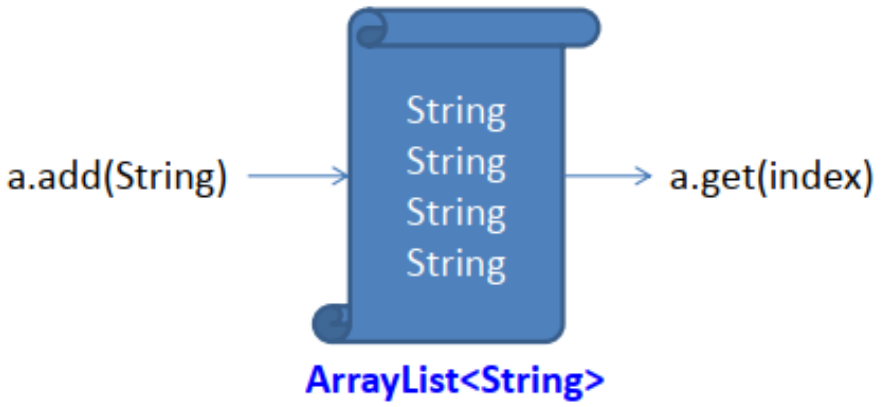
+ Tightly controlled, avoiding the risk of incorrect data manipulation.

+ Faster processing performance.

ArrayList<Type> type declaration

```
ArrayList<String> a = newArrayList<String>();
a.add("Maria");
a.add("John");
a.add("Tommy");
a.add("Lisa")
String s = a.get(2);
```

When accessing elements, there is **no need to cast back to the original type** of the element for processing.



Note: **<Type>** is a non-primitive data type (wrapper type must be used).

Important Points about the Java ArrayList

- Java ArrayList class can contain duplicate elements.
- Java ArrayList class maintains insertion order.
- Java ArrayList class is non **synchronized**.
- 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.

Important Points about the Java ArrayList

- We can not create an array list of the primitive types, such as int, float, char, etc. It is required to use the required wrapper class in such cases. For example:

```
ArrayList<int> al = ArrayList<int>(); // does not word
ArrayList<Integer> al = new ArrayList<Integer>(); // works fine
```

- Java ArrayList gets initialized by the size. The size is dynamic in the array list, which varies according to the elements getting added or removed from the list.

Methods of ArrayList

Method	Description
void add(int index, E element)	It is used to insert the specified element at the specified position in a list.
boolean add(E e)	It is used to append the specified element at the end of a list.
E get(int index)	It is used to fetch the element from the particular position of the list.
E set(int index, E element)	It is used to replace the specified element in the list, present at the specified position
void clear()	It is used to remove all of the elements from this list.
boolean contains(Object o)	It returns true if the list contains the specified element.
E remove(int index)	It is used to remove the element present at the specified position in the list.

Methods of ArrayList

Method	Description
boolean remove(Object o)	It is used to remove the first occurrence of the specified element.
boolean removeAll(Collection<?> c)	It is used to remove all the elements from the list.
boolean isEmpty()	It returns true if the list is empty, otherwise false.
int size()	It is used to return the number of elements present in the list.
void trimToSize()	It is used to trim the capacity of this ArrayList instance to be the list's current size.
Object[] toArray()	It is used to return an array containing all of the elements in this list in the correct order.

ArrayList Manipulation

```
ArrayList<String> a = new ArrayList<String>();
a.add("Maria");          ←   [Maria]
a.add("John");           ←   [Maria, John]
a.add("Rosie");          ←   [Maria, John, Rosie]
a.add("Lisa");           ←   [Maria, John, Rosie, Lisa]
a.add(1, "Tommy");       ←   [Maria, Tommy, John, Rosie, Lisa]
a.set(0, "Lyly");        ←   [Lyly, Tommy, John, Rosie, Lisa]
a.remove(3)              ←    [Lyly, Tommy, John, Lisa]
```

Java ArrayList Example

```
// Creating arraylist
ArrayList<String> list = new ArrayList<String>();
     list.add("Mango");     // Adding object in arraylist
     list.add("Apple");
     list.add("Banana");
     list.add("Grapes");
     // Printing the arraylist object
    System.out.println(list);  // [Mango, Apple, Banana, Grapes]
```

Get and Set ArrayList Example

```
// accessing the element
// it will return the 2nd element, because index starts from 0
System.out.println("Returning element: "+list.get(1));  // Apple
//changing the element
  al.set(1,"Dates");
//Traversing list
  for(String fruit:al)
          System.out.println(fruit);
   }
// Mango, Dates, Banana, Grapes
```

For-Each Loop

- Traversal by **index** using a for loop or using a **for-each** loop. For ArrayLists, the for-each loop is commonly preferred.

```
ArrayList<Integer> a = newArrayList<Integer>();
a.add(5);
a.add(9);
a.add(4);
a.add(8)
```

```
for(int i=0;i<a.size();i++){
    Integer x = a.get(i);
    <<xử lý x>>
}
```

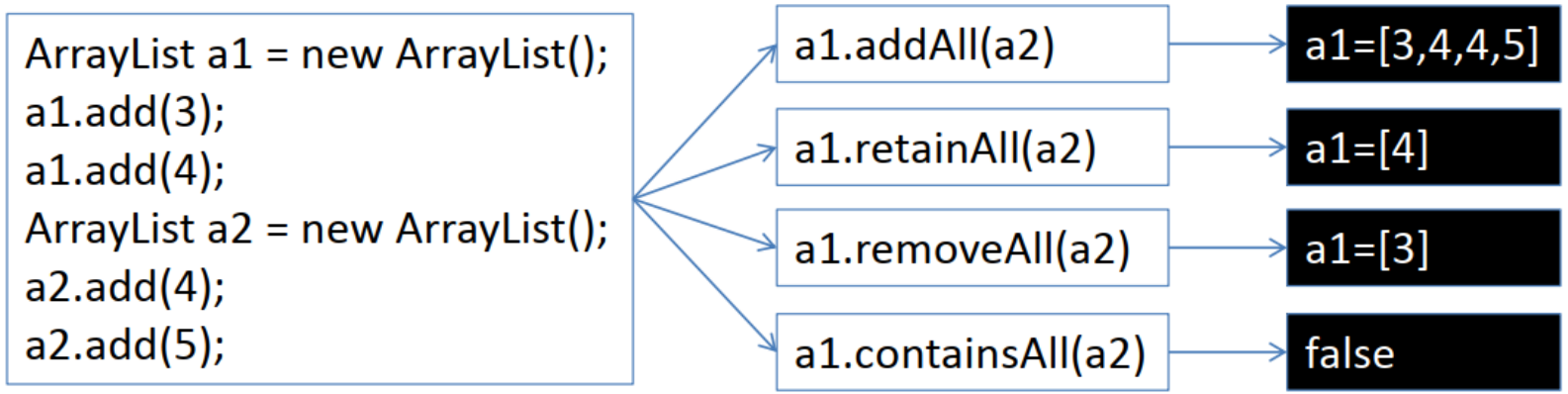
Java ArrayList Example

This is a sample program to demonstrate isEmpty functionality:

```
ArrayList<String> al = new ArrayList<String>();
System.out.println("Is ArrayList Empty: "+al.isEmpty());   //true
 al.add("Ravi");
 al.add("Vijay");
 al.add("Ajay");
 System.out.println("After Insertion");
 System.out.println("Is ArrayList Empty: "+al.isEmpty());  //false
```

Operations Arraylist

Method	Description
addAll(Collection)	Union of Two Sets
removeAll(Collection)	Difference of Two Sets
removeAll(Collection)	Intersection of Two Sets
removeAll(Collection)	Check Existence
toArray(T[])	Convert to Array



BTEC Alliance with FPT Education
Pearson

Advanced ArrayList Operations

- The **Collections** utility class provides convenient utility functions to support ArrayList handling

Method	Description
int binarySearch (List list, Object key)	Binary Search Algorithm
int binarySearch (List list, Object key)	Assign a Value to All Elements
void shuffle (List list)	Random Permutation
void sort (List list)	Ascending Sorting
void reverse (List list)	Convert to Array
void rotate (List list, int distance)	Rotate
void swap(List list, int i, int j)	Swap

Example

```
ArrayList<Integer> a = new ArrayList<Integer>();
a.add(3);
a.add(9);
a.add(8);
a.add(2);                          // [3, 9, 8, 2]
Collections.swap(a, 0, 2);         // [8, 9, 3, 2]

Collections.shuffle(a);            // [X, X, X, X]

Collections.sort(a);               // [2, 3, 8, 9]

Collections.reverse(a);            // [9, 8, 3, 2]
```

Advanced Sorting

There are 2 ways to use Collections.sort() to sort an ArrayList<Object>:

- Collections.**sort(ArrayList)** for elements that are comparable (Integer, Double, String...)
- Collections.**sort(ArrayList, Comparator)** adds comparison criteria for elements. This approach is often used for user-defined classes (NhanVien, SinhVien...)

Example Collections.sort(ArrayList, Comparator)

- The comparison criterion is indicated to perform the sorting. In this example, the comparison criterion for two SV objects is based on comparing their scores

```
ArrayList<SV> list = new ArrayList<SV>();
Comparator<Student> comp = new Comparator<Student>() {
    @Override
    public int compare(Student o1, Student o2) {
        return o1.avgScore.compareTo(o2.avgScore);
    }
};
Collections.sort(list, comp);
```

There are three cases:

- = 0: o1 = o2
- 0: o1 > o2
- < 0: o1 < o2

Iterator (1/4)

The **ArrayList.Iterator ()** returns an iterator over the elements in this list

```
public Iterator<E> Iterator()
```

Specifically, the java.util.ListIterator interface includes the following methods:

- **add(e):** Adds the element e at the current position of the iterator.

Iterator (2/4)

- **hasNext():** Returns true if this list iterator has more elements when traversing the list in the forward direction
- **hasPrevious():** Returns true if this list iterator has more elements when traversing the list in the reverse direction
- **next():** Returns the next element in the list and advances the cursor position

Iterator (3/4)

- **nextIndex():** Returns the index of the element that would be returned by a subsequent call to **next()**
- **previous():** Returns the previous element in the list and moves the cursor position backwards
- **previousIndex():** Returns the index of the element that would be returned by a subsequent call to **previous()**

Iterator (4/4)

- **remove():** Removes from the list the last element that was returned by **next()** or **previous()** (optional operation)
- **set(E e):** Replaces the last element returned by **next()** or **previous()** with the specified element (optional operation)

Example

```
ArrayList<String> a = new ArrayList<String>();
a.add(“d”);
a.add(“dd”);
a.add(“ddd”);
a.add(“dddd”);
a.add(“ddddd”);

Iterator<String> iterator = arrlist.iterator();
while (iterator.hasNext())  {
        String i = iterator.next();
        System.out.println(i);
}
```

Output:

d
dd
ddd
dddd
ddddd

Group Exercise: Managing an Integer List

Objective: Write a program to manage an integer list using ArrayList. The program should allow adding new integers to the list, displaying the list, and calculating the sum of the integers

Group exercise: Managing an Integer List

Requirements:

- Create a class NumberManagement to manage an integer list using ArrayList
- Provide the following functionalities:
  - Add a new integer to the list
  - Display the entire list of integers
  - Calculate and display the sum of the integers in the list

Summary

- ArrayList
- Typed ArrayList
- ArrayList Operations
- Collections Utility Class

BTEC Alliance with FPT Education

Pearson