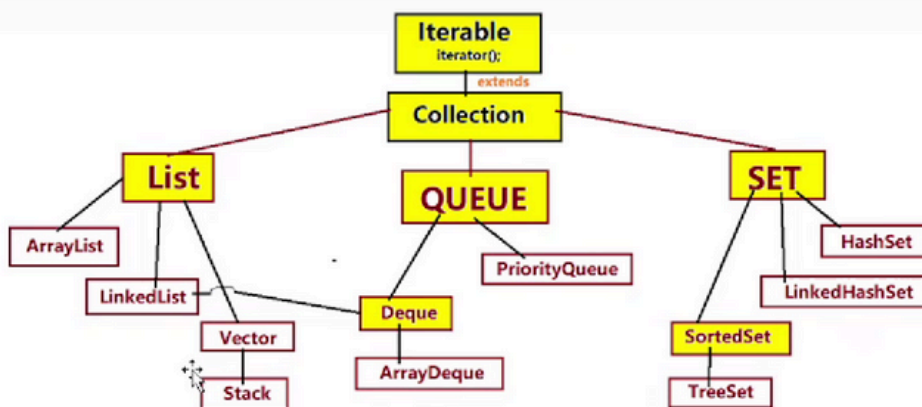


Lec 36 Collections P1:

The collection framework in Java helps store and manage groups of data more flexibly than arrays. Unlike arrays, collections can handle varying data sizes and types, making them essential for real-world applications where data requirements change. Collections only store objects, not primitive types, and offer many built-in features and data structures to suit different needs.

COLLECTION HIERARCHY



Arrays and Their Limitations

Arrays can store multiple values, but their size is fixed at creation and they only allow one data type (homogeneous data). They are type-safe, meaning only the specified data type can be stored. These

restrictions make arrays less practical for applications with changing or mixed data.

Real-World Data Storage Needs

In real applications like social media, data is retrieved from databases and can vary in both size and type (names, numbers, booleans, etc.). Arrays struggle with these requirements since they can't resize and need homogeneous data.

Introduction to Collection Framework

The collection framework in Java provides a set of ready-made data structures and tools to store and manipulate groups of objects. It solves the limitations of arrays by being dynamic in size and supporting heterogeneous data through objects.

Comparing Arrays and Collections

Arrays have a fixed size, can store primitives or objects, and are always type-safe. Collections are dynamic in size, store only objects, and can be made type-safe using generics. Collections also provide built-in iterators for easier data traversal.

Flexibility and Features of Collections

Collections are resizable and flexible, allowing storage of as much data as needed without worrying about size at compile time. They offer rich features for data manipulation, such as insertion, deletion, and iteration, but may be slower than arrays due to these extra capabilities.

Collection Hierarchy Overview

The collection framework is organized as a hierarchy of interfaces and classes. At the top is the `Iterable` interface, followed by `Collection`, which branches into main interfaces like `List`, `Queue`, and `Set`. Each interface has different implementations (e.g., `ArrayList`, `LinkedList`, `HashSet`) suited for specific needs.

Implementation Classes in Collections

`List`, `Queue`, and `Set` interfaces have various implementation classes like `ArrayList`, `LinkedList`, `Vector`, `Stack`, `PriorityQueue`, `HashSet`, and `TreeSet`. Each class has unique features and use-cases, such as maintaining order, allowing duplicates, or sorting data.

Storing Only Objects in Collections

Collections can only store objects, not primitive data types like int or float. To store primitive values, they must be converted to their object equivalents using wrapper classes (e.g., Integer for int, Float for float).

Wrapper Classes and Boxing

Wrapper classes in Java (Integer, Float, Character, etc.) "wrap" primitive values into objects so they can be stored in collections. Manual boxing is the old way of converting primitives to objects, while Java 5 introduced auto-boxing and auto-unboxing, which automate this process.

Auto-boxing and Auto-unboxing

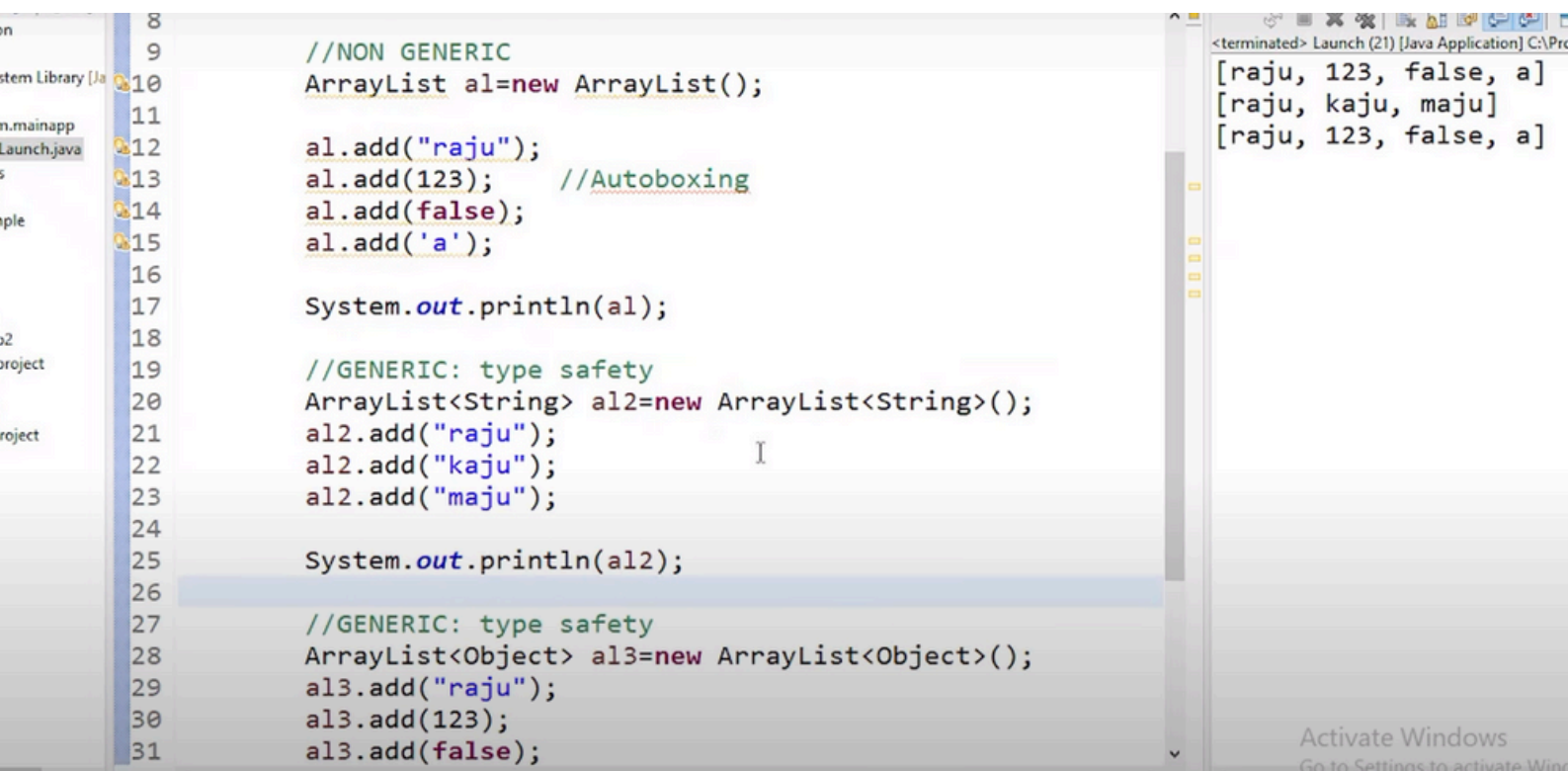
Auto-boxing automatically converts primitives to their object forms when needed, and auto-unboxing does the reverse. This feature removes the need for manual conversion, making code cleaner and easier to write.

Choosing the Right Collection

The key to using collections effectively is knowing which implementation fits the data and required operations.

Each collection class is designed for specific scenarios, such as needing order, fast access, or frequent updates.

The concept of the Collection Framework in Java helps store and manage groups of data easily and flexibly. The ArrayList, a key part of this framework, allows dynamic storage, easy access, and many operations on data, making it more powerful than arrays. Using generics in collections provides type safety, and ArrayLists are best for fast data access but not for frequent modifications.



```
8
9 //NON GENERIC
10 ArrayList al=new ArrayList();
11
12 al.add("raju");
13 al.add(123); //Autoboxing
14 al.add(false);
15 al.add('a');
16
17 System.out.println(al);
18
19 //GENERIC: type safety
20 ArrayList<String> al2=new ArrayList<String>();
21 al2.add("raju");
22 al2.add("kaju");
23 al2.add("maju");
24
25 System.out.println(al2);
26
27 //GENERIC: type safety
28 ArrayList<Object> al3=new ArrayList<Object>();
29 al3.add("raju");
30 al3.add(123);
31 al3.add(false);
```

Output:

```
<terminated> Launch (21) [Java Application] C:\Pro
[raju, 123, false, a]
[raju, kaju, maju]
[raju, 123, false, a]
```

Activate Windows
Go to Settings to activate Windows

Introduction to Collections and Arrays

Collections provide a flexible way to store and manipulate data, unlike arrays, which have a fixed size and limited features. Collections can handle multiple sets of data and offer more operations than basic arrays.

Array vs. Collection

Arrays have fixed size and type safety but lack advanced features. Collections, on the other hand, offer more capabilities, such as dynamic resizing and various data operations.

Storing Primitives and Wrapper Classes

Collections cannot store primitive data types directly; primitives must be converted to objects using wrapper classes. Java supports auto-boxing, which automatically converts primitives to their object equivalents.

Introduction to ArrayList

ArrayList is a class in the Java Collection Framework that allows dynamic storage of objects. It is a child of the List interface and is easy to create and use.

Inserting Data into ArrayList and Auto-Boxing

ArrayLists can store different types of objects, and when primitives are added, auto-boxing converts them to objects automatically. This allows easy insertion of various data types.

Introduction to Generics in Collections

Generics allow you to specify the type of data an ArrayList will store, providing type safety and reducing runtime errors. Non-generic collections can store any object type but may lead to type mismatch issues.

Using Generic and Non-Generic ArrayLists

A generic ArrayList restricts the type of objects it can store, while a non-generic ArrayList can store multiple types. You can also use the Object type to allow any object in a generic list.

Wildcard Generics

Wildcard generics use the question mark (?) to represent an unknown type, making the collection read-only and preventing modification except for adding null.

This is useful in scenarios where the type is not known in advance.

ArrayList Features: Order, Index, and Random Access

ArrayList preserves the order of inserted elements and supports index-based access, allowing random access and modification at any position. Not all collections support these features.

ArrayList Synchronization and Duplicates

ArrayList is not synchronized, meaning it is not thread-safe for concurrent operations. It allows duplicate values and null entries, making it suitable for certain use cases but not all.

ArrayList Capacity and Internal Structure

ArrayList starts with a default capacity (usually 10) and grows dynamically as more elements are added. The capacity increases using a formula based on the current size, and you can set the initial capacity if needed.

Internal Working and Efficiency of ArrayList

Internally, ArrayList uses a dynamic array. Modifying (adding or removing) elements can be inefficient due to shifting elements, especially with large lists. Thus, ArrayList is best used for fast data access rather than frequent modifications.

Practical Use Case for ArrayList

ArrayLists are ideal for holding and accessing large amounts of data, such as records fetched from a database, especially when the primary need is to access rather than modify the data.

Common ArrayList Methods

Key ArrayList methods include add (insert), remove (delete by index or object), get (retrieve by index), contains (check for presence), indexOf (find position), clear (empty the list), and conversion to array. Iteration can be done using loops or iterators.

SubList and Advanced Features

The subList method allows creating a new list from a range of elements in the original ArrayList. ArrayLists also support sorting and other advanced features not available in arrays.

Summary and Conclusion

ArrayList offers rich features like dynamic sizing, easy access, multiple data operations, and flexibility, which are not present in arrays. It is best used for scenarios requiring fast access and minimal modifications.