Structuring Data with Java

By Dr. Subrat Kumar Nayak

Associate Professor

Department of CSE

ITER, SOADU

Structuring Data with Java

- Except very few application, all needs to keep track of the structured data.
- Hence, a well-defined data structure is present in different sections in order to access data systematically.
- There are data structures
  - **in the memory of a running program**
  - in the data in a file on disk
  - in the information stored in a database

Using Arrays for Data Structuring

- To keep track of a fixed amount of information and retrieve it (usually) sequentially, we need an array.
- Arrays can be used to hold any linear collection of data. The items in an array must all be of the same type.
- An array can be formed by using either any **primitive type** or **any object type**.

Using Arrays for Data Structuring

Examples of creating and initializing a one dimensional array.

```
(1) int[] monthLen1; // declare a reference
    monthLen1 = new int[12]; // construct it
(2) int[] monthLen2 = new int[12]; // short form
(3) // even shorter is this initializer form:
    int[] monthLen3 = { 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31,};
```

Examples of creating a two dimensional array.

```
// Two-Dimensional Arrays
// Want a 10-by-24 array
int[][] me = new int[10][];
for (int i=0; i<10; i++)
me[i] = new int[24];
```

Using Arrays for Data Structuring

How to find and print the length we can do the following.

- `System.out.println(me.length);`
- `System.out.println(me[0].length);`

Implement the following programs.

- Q1. Write a program to create an array of integer and display it.
- Q2. Write a program to create an array of Strings object and display it.

Hint: Create an array of Strings and display individual string from that array.

Resizing an Array

- We can not add more elements unless until the array is allocated with a reasonable size.
- Or we can take the help of ArrayList collection class that dynamically changes its size.

The collection Frameworks

- The **Collection in Java** is a framework that provides an architecture to store and manipulate the group of objects.
- Java Collections can achieve all the operations that you perform on a data such as searching, sorting, insertion, manipulation, and deletion.
- **Java Collection** means a single unit of objects, i.e., a group
- Java Collection framework provides many interfaces (Set, List, Queue, Deque) and classes (ArrayList, Vector, LinkedList, PriorityQueue, HashSet, LinkedHashSet, TreeSet).

What is a framework in Java

- It provides readymade architecture.
- It represents a set of classes and interfaces.

What is Collection framework

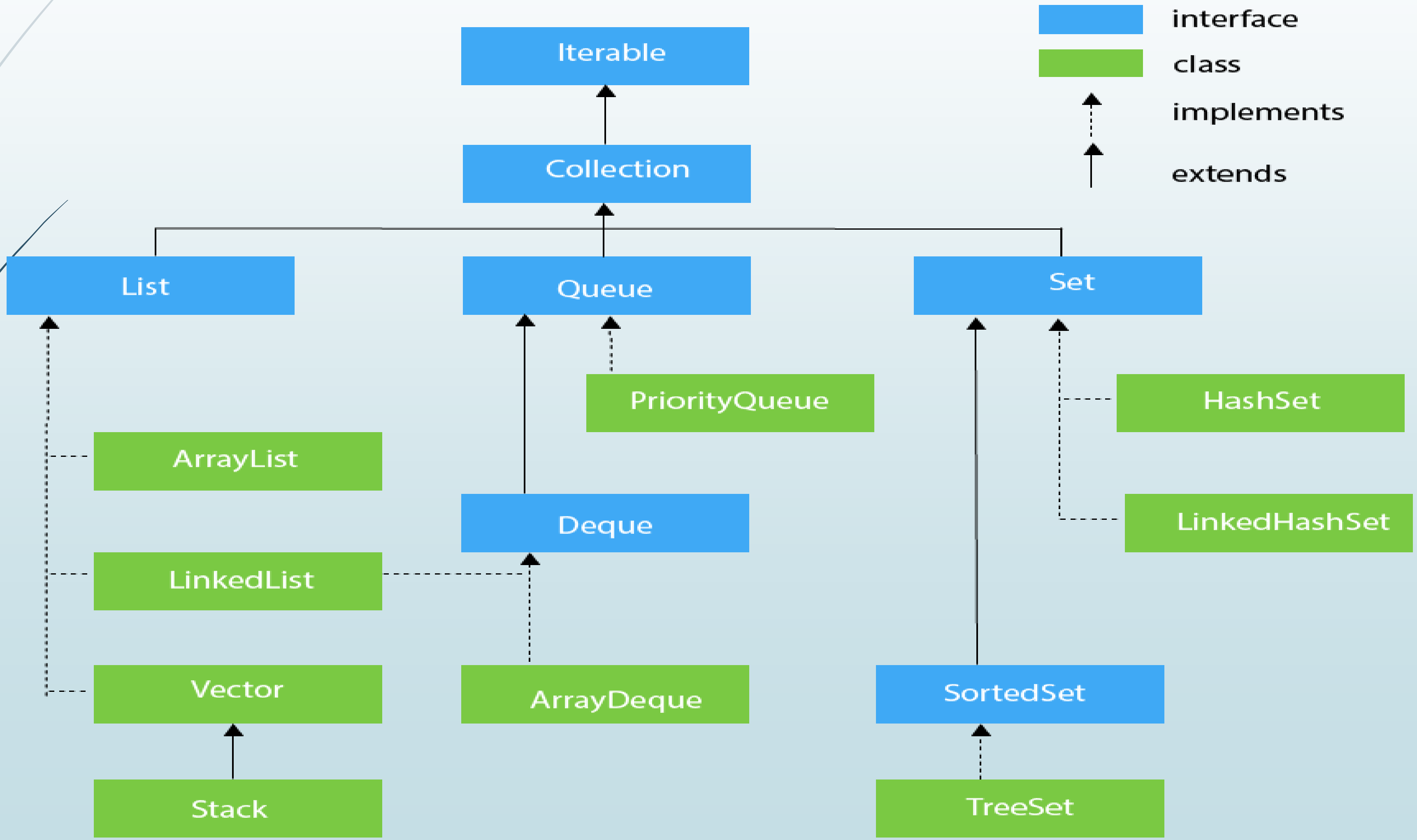
- The Collection framework represents a unified architecture for storing and manipulating a group of objects. It has:

1) Interfaces and its implementations, i.e., classes

1) Algorithm

Hierarchy of Collection Framework

- The **java.util** package contains all the classes and interfaces for the Collection framework.



Like an Array, but More Dynamic

- Java ArrayList class uses a **dynamic array** for storing the elements. It implements List interface. The important points about Java ArrayList class are:

- ✓ Java ArrayList class can contain **duplicate** elements.
- ✓ Java ArrayList class maintains insertion order.
- ✓ Java ArrayList class is non synchronized. (Multiple thread can access ArrayList at a time)
- ✓ Java ArrayList allows random access because array works at the index basis.
- ✓ In Java ArrayList class, manipulation is slow because a lot of shifting needs to occur if any element is removed from the array list.

- The List interface extends the Collection and Iterable interfaces in hierarchical order.
- **ArrayList can not be used for primitive types, like int, char, etc. We need a wrapper class for such cases .**

Consructor:

- *ArrayList()*

This constructor is used to build an empty array list

Like an Array, but More Dynamic

Example:

```
import java.util.ArrayList;
public class TestArrayList
{
   public static void main(String[] arg){
      ArrayList al=new ArrayList();
      //or
      List al=new ArrayList();
      //Because ArrayList implements List interface
   }
}
```

Like an Array, but More Dynamic

Methods:

- `add(Object o)` Add the given element at the end
- `add(int i, Object o)` Insert the given element at the specified position
- `clear( )` Remove all element references from the Collection
- `contains(Object o)` True if the List contains the given Object
- `get(int i)` Return the object reference at the specified position
- `indexOf(Object o)` Return the index where the given object is found, or -1
- `remove(Object o)` or `remove(int i)`

Remove an object by reference or by position

- toArray( ) Return an array containing the objects in the Collection

Like an Array, but More Dynamic

- Q. Write a program to add 3 string object to an ArrayList and display it.

```
import java.util.ArrayList;
public class TestArrayList
{
    public static void main(String[] arg)
    {
        ArrayList al=new ArrayList();
        al.add("Subrat");
        al.add("Kumar");
        al.add("Nayak");
        for(int i=0;i<as.size();i++)
        {
            System.out.println(as.get(i));
        }
    }
}
```

Like an Array, but More Dynamic

Q. Write a program to create an ArrayList and insert 5 integer in it and find its sum.

The issue can be resolved by:

- Casting
- By using generics

Like an Array, but More Dynamic

(Avoid Casting by Using Generics)

Java Non-generic Vs. Generic Collection

- Java collection framework was non-generic before JDK 1.5. Since 1.5, it is generic.
- Java new generic collection **allows you to have only one type of object** in a collection. Now it is type safe so typecasting is not required at runtime.
- The old non-generic example of creating java collection.

```
ArrayList al=new ArrayList();
//creating old non-generic arraylist
```

- The new generic example of creating java collection.

```
ArrayList<object_type> al=new ArrayList<Object_type>();
//creating new generic arraylist
```

Advantages of Generic:

- **Type-safety:** We can hold only a ***single type*** of objects in generics. It does not allow to store other objects. Without Generics, we can store any type of objects.
- **Type casting is not required:** There is no need to typecast the object.
- **Compile-Time Checking:** It is ***checked at compile time*** so problem will not occur at runtime.

End of Session