Object Oriented Programming (with JAVA)

by
Dr. Partha Roy,
Professor
Bhilai Institute of Technology, Durg

UNIT- V

Collections Frameworks: HashSet, TreeSet, ArrayList, LinkedList, Vector, HashMap, TreeMap, Hashtable classes. Generics in Java: Creating instances of generic classes, generic types, Declaring (and invoking) methods that take generic types. Creating and running executable JAR (Java ARchives).

Collections Framework

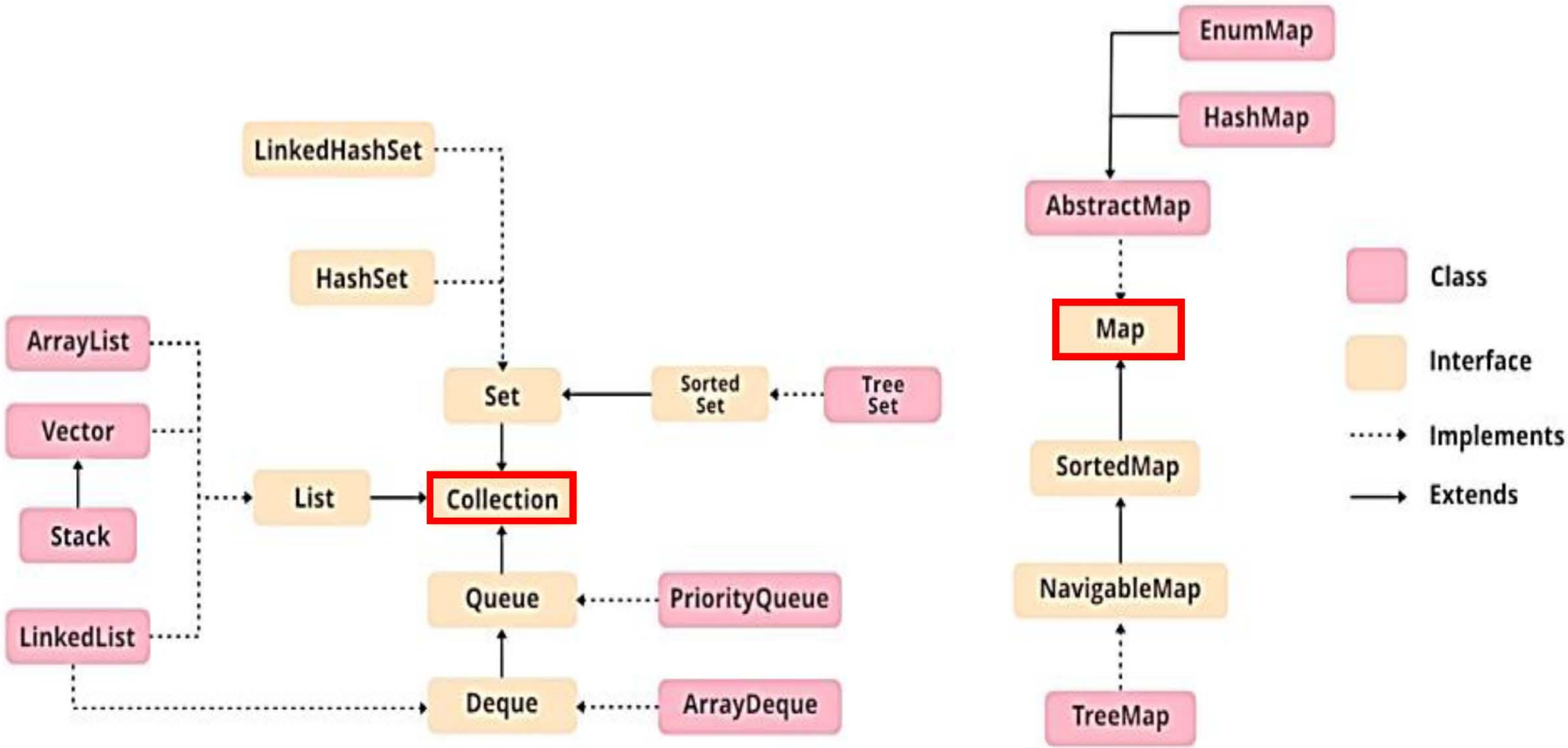
Collection (java.util)

- Any group of individual objects which are represented as a single unit is known as the collection of the objects. In Java, a separate framework named the "Collection Framework" has been defined in JDK 1.2 which holds all the collection classes and interface in it.
- The Collection interface (**java.util.Collection**) and Map interface (**java.util.Map**) are the two main "root" interfaces of Java collection classes.

Framework

- A framework is a set of classes and interfaces which provide a ready-made architecture.
- In order to implement a new feature or a class, there is no need to define a framework.
- **Advantages:**
  - Consistent API
  - Reduces programming effort
  - Increases program speed and quality

Collection



Some basic Collection member methods

Method	Description
add(Object)	This method is used to add an object to the collection.
addAll(Collection c)	This method adds all the elements in the given collection to this collection.
clear()	This method removes all of the elements from this collection.
contains(Object o)	This method returns true if the collection contains the specified element.
containsAll(Collection c)	This method returns true if the collection contains all of the elements in the given collection.
equals(Object o)	This method compares the specified object with this collection for equality.

Some basic Collection member methods

Method	Description
isEmpty()	This method returns true if this collection contains no elements.
iterator()	This method returns an iterator over the elements in this collection.
max()	This method is used to return the maximum value present in the collection.
remove(Object o)	This method is used to remove the given object from the collection. If there are duplicate values, then this method removes the first occurrence of the object.
removeAll(Collection c)	This method is used to remove all the objects mentioned in the given collection from the collection.
size()	This method is used to return the number of elements in the collection.
toArray()	This method is used to return an array containing all of the elements in this collection.

The "List" Interface

- This is a child interface of the collection interface.
- This interface is dedicated to the data of the list type in which we can store all the **ordered collection of objects.**
- List also **allows duplicate** data to be present in it.
- List interface is implemented by various classes like *ArrayList*, *Vector*, *Stack*, etc. Since all the subclasses implement the list, we can instantiate a list object with any of these classes.
- **Example Syntax:**
  - List <T> al = new ArrayList<> ();
  - List <T> ll = new LinkedList<> ();
  - List <T> v = new Vector<> ();
- Where T is the type of the object

ArrayList

- ArrayList provides us with dynamic arrays in Java.
- Though, it may be slower than standard arrays but can be helpful in programs where lots of manipulation in the array is needed.
- The size of an ArrayList is increased automatically if the collection grows or shrinks if the objects are removed from the collection.
- Java ArrayList allows us to randomly access the list.
- ArrayList can not be used for primitive types, like int, char, etc. We will need a wrapper class for such cases.

LinkedList

- LinkedList class is an implementation of the LinkedList data structure which is a linear data structure where the elements are not stored in contiguous locations and every element is a separate object with a data part and address part.
- The elements are linked using pointers and addresses. Each element is known as a node.

Vector

- A vector provides us with dynamic arrays in Java.
- Though, it may be slower than standard arrays but can be helpful in programs where lots of manipulation in the array is needed.
- This is identical to ArrayList in terms of implementation.
- However, the primary difference between a vector and an ArrayList is that a Vector is synchronized and an ArrayList is non-synchronized.

The "Set" Interface

- A set is an **unordered collection of objects** in which **duplicate values cannot be stored**.
- Set is used when we wish to avoid the duplication of the objects and wish to store only the unique objects.
- Set interface is implemented by various classes like HashSet, TreeSet, LinkedHashSet, etc. Since all the subclasses implement the set, we can instantiate a set object with any of these classes.
- **Example Syntax:**
  - Set<T> hs = new HashSet<> ();
  - Set<T> lhs = new LinkedHashSet<> ();
  - Set<T> ts = new TreeSet<> ();
- Where T is the type of the object

HashSet

- The HashSet class is an inherent implementation of the hash table data structure.
- The objects that we insert into the HashSet do not guarantee to be inserted in the same order.
- The objects are inserted based on their hashcode.
- This class also allows the insertion of NULL elements.

TreeSet

- The TreeSet class uses a Tree for storage.
- The ordering of the elements is maintained by a set using their natural ordering whether or not an explicit comparator is provided during the creation of the creation of the set.

The "Map" Interface

- A map is a data structure that supports the **key-value pair** mapping for the data.
- Map doesn't support duplicate keys because the same key cannot have multiple mappings.
- A map is useful if there is data and we wish to perform operations on the basis of the key.
- This map interface is implemented by various classes like HashMap, TreeMap, etc. Since all the subclasses implement the map, we can instantiate a map object with any of these classes.
- **Example Syntax:**
  - Map<T> hm = new HashMap<> ();
  - Map<T> tm = new TreeMap<> ();
- Where T is the type of the object

HashMap

- HashMap provides the basic implementation of the Map interface of Java.
- It stores the data in **(Key, Value) pairs.**
- To access a value in a HashMap, we must know its key.
- HashMap uses a technique called Hashing.
- Hashing is a technique of converting a large String to a small String that represents the same String so that the indexing and search operations are faster.
- HashSet also uses HashMap internally.
- It is not Synchronized.

TreeMap

- The TreeMap in Java is used to implement Map interface.
- The map is sorted according to the natural ordering of its keys, or by a Comparator provided at map creation time, depending on which constructor is used.
- This proves to be an efficient way of sorting and storing the key-value pairs.

HashTable

- The Hashtable class implements a hash table, which maps keys to values.
- Any non-null object can be used as a key or as a value.
- To successfully store and retrieve objects from a hashtable, the objects used as keys must implement the hashCode() method and the equals method.
- It is similar to HashMap, but is synchronized.

Generics in Java

Dr. Partha Roy, Professor, B.I.T, Durg

Generics

- It is datatype polymorphism.
- Similar to templates in C++.
- Primitive datatypes cannot be used to represent Generic types, only class types are allowed.
- Better than using Object as using Object can make the code vulnerable to type mismatch errors.

Generics

Consider the following code:

```
public class Test {
    private Object obj;
    public void set(Object obj) { this.obj = obj; }
    public Object get() { return obj; }
}
```

Since its methods accept or return an Object, we are free to pass in whatever we want, provided that it is not one of the primitive types.

There is no way to verify, at compile time, how the class is used.

One part of the code may place an Integer in the Test and expect to get Integers out of it, while another part of the code may mistakenly pass in a String, resulting in a runtime error.

Generics

So it is better to use Generics:

```
public class Test<T> {
    // T stands for "Type"
    private T t;
    public void set(T t) { this.t = t; }
    public T get() { return t; }
}
```

Now when we want to create an instance of Test we need to specify the datatype:

```
Test<Integer> ob1=new Test<Integer>();
Test<Double> ob2=new Test<Double>();
```

So, for ob1 T will represent Integer and for ob2 T will represent Double, so there is no runtime error as in the case with previous example where Object is used.

Generics

- Syntax for Generic class:

```
class class-name<T1,T2,..>{
        //class definition
}
```

Generic Class Example

- Syntax for Generic member method:

```
class class-name{
<T1,T2..> return-type method-name(i/p args){
        //definition of the method
}
}
```

Generic Method Example

JAR files

JAR (Java Archive)

- JAR stands for Java ARchive,
- Its format is based on the zip format.
- The JAR files format is mainly used to aggregate a collection of files into a single one.

Creating a JAR file

jar cf jar-file.jar input-file(s)

This command will generate a compressed JAR file and place it in the current directory.

- The **c** option indicates that we want to create a JAR file.
- The **f** option indicates that we want the output to go to a file rather than to stdout.
- jar-file is the name that we want the resulting JAR file to have. We can use any filename for a JAR file. By convention, JAR filenames are given a .jar extension, though this is not required.
- The input-file(s) argument is a space-separated list of one or more files that we want to include in our JAR file. The input-file(s) argument can contain the wildcard * symbol. If any of the "input-files" are directories, the contents of those directories are added to the JAR archive recursively.

Extracting a JAR file

jar xf jar-file.jar

This command will extract the contents of the JAR file and place it in the current directory.

- To open a JAR file and extract the contents to the file system, we must use two JAR utility switches, namely, "**x**" to extract the contents, and "**f**" to specify the name of the JAR file being opened. The utility then extracts the contents to the file system.

Creating & Running JAR file (Method-1)

java -jar jar-file.jar

This command will run the class file that contains the main method in the jar file.

Before we run this we need to edit the MANIFEST.MF file in the jar file.

We need to add

Main-Class: classname

ex: Main-Class: test

// where test is the name of the .class file that has the main method

The classname should be the name of the class containing the main method which we want to run when the jar file is executed using above command.

Creating & Running JAR file (Method-2)

java -cp jar-file.jar class-file

This command will run the class-file that contains the main method in the jar file.

- We should ensure that the class-file mentioned should contain the main method.
- The option –cp is used to specify that the classpath is within the jar file mentioned after –cp.

End of UNIT-V