Java Collections

Java Collections

- The **Collection in Java** is a framework that provides an architecture to store and manipulate the group of objects.
- 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.
- 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.
- Java Collection framework provides many

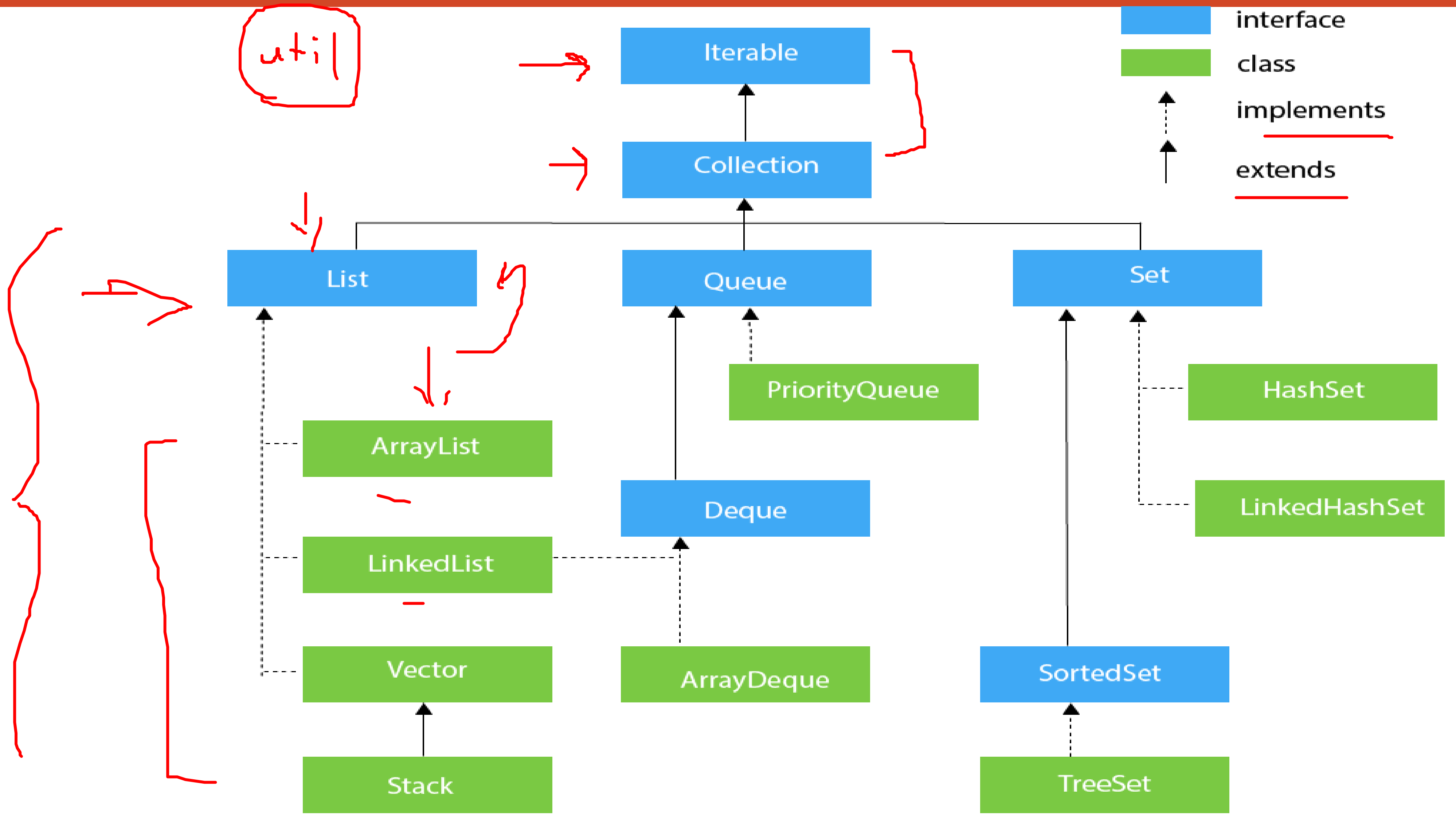
1. interfaces (Set, List, Queue, Deque) and
2. classes (ArrayList, Vector, LinkedList, PriorityQueue, HashSet, LinkedHashSet, TreeSet).

Java Collections

- What is Collection in Java
- A Collection represents a single unit of objects, i.e., a group.
- 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
2. Algorithm

Java Collections



Java Collections

Methods of Collection interface

No.	Method	Description
1	public boolean add(E e)	It is used to insert an element in this collection.
2	public boolean addAll(Collection<? extends E> c)	It is used to insert the specified collection elements in the invoking collection.
3	public boolean remove(Object element)	It is used to delete an element from the collection.
4	public boolean removeAll(Collection<?> c)	It is used to delete all the elements of the specified collection from the invoking collection.
5	default boolean removeIf(Predicate<? super E> filter)	It is used to delete all the elements of the collection that satisfy the specified predicate.
6	public boolean retainAll(Collection<? > c)	It is used to delete all the elements of invoking collection except the specified collection.
7	public int size()	It returns the total number of elements in the collection.

Java Collections

Methods of Collection interface

c $\boxed{a\ b\ c\ d}$ $\boxed{b}$

8	public void clear()	It removes the total number of elements from the collection.
9	public boolean contains(Object element)	It is used to search an element.
10	public boolean containsAll(Collection<?> c) b	It is used to search the specified collection in the collection.
11	public Iterator iterator()	It returns an iterator.
12	public Object[] toArray()	It converts collection into array.
13	public <T> T[] toArray(T[] a)	It converts collection into array. Here, the runtime type of the returned array is that of the specified array.
14	public boolean isEmpty()	It checks if collection is empty.

Java Collections

- List Interface
- List interface is the child interface of Collection interface. It inhibits a list type data structure in which we can store the ordered collection of objects. It can have duplicate values.
- List interface is implemented by the classes ArrayList, LinkedList, Vector, and Stack.
- To instantiate the List interface, we must use :

1. List <data-type> list1= new ArrayList();
2. List <data-type> list2 = new LinkedList();
3. List <data-type> list3 = new Vector();
4. List <data-type> list4 = new Stack();

Java Collections

- ArrayList
- The ArrayList class implements the List interface.
- It uses a dynamic array to store the duplicate element of different data types.
- The ArrayList class maintains the insertion order.
- The elements stored in the ArrayList class can be randomly accessed. Consider the following example.
- For example: ArrayListDemo.java
  - ArrayListDemoObj.java

Java Collections

- Stack
- The stack is the subclass of Vector.
- It implements the last-in-first-out data structure, i.e., Stack.
- The stack contains all of the methods of Vector class and also provides its methods like
- boolean push(),
- boolean peek(),
- boolean push(object o), which defines its properties.
- For example: StackCollectionDemo.java

Generics in java

- The Java Generics programming is introduced in J2SE 5 to deal with type-safe objects.
- It makes the code stable by detecting the bugs at compile time.
- Before generics, we can store any type of objects in the collection, i.e., non-generic.
- Now generics force the java programmer to store a specific type of objects.
- Advantage of Java Generics

1. Type-safety
2. Type casting is not required
3. Compile-Time Checking

Generics in java

1. **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.

```
List list = new ArrayList();
list.add(10);
list.add("10");
```

- With Generics, it is required to specify the type of object we need to store.

```
List<Integer> list = new ArrayList<Integer>();
list.add(10);
list.add("10");// compile-time error
```

Generics in java

2. **Type casting is not required:**

- There is no need to typecast the object.
- Before Generics, we need to type cast.

```
List list = new ArrayList();   ✓ ✓
list.add("hello");   1 b   e l
String s = (String) list.get(0);//typecasting
```

- After Generics, we don't need to typecast the object.

```
List<String> list = new ArrayList<String>();
list.add("hello");
String s = list.get(0);   ✓
```

Generics in java

3. Compile-Time Checking

- It is checked at compile time so problem will not occur at runtime.
- The good programming strategy says it is far better to handle the problem at compile time than runtime.

```java
List<String> list = new ArrayList<String>();
list.add("hello");
list.add(32);//Compile Time Error
```

Generics in java

- Generic class
- A class that can refer to any type is known as a generic class.
- Here, we are using the T type parameter to create the generic class of specific type.

```
class MyGen<T>{
T obj;
void add(T obj){this.obj=obj;}
}
}
```

- The T type indicates that it can refer to any type (like String, Integer, and Employee). The type you specify for the class will be used to store and retrieve the data.

Generics in java

```
class MyGen<T>{
T obj;
void add(T obj){this.obj=obj;}
T get(){return obj;}
}
```

```
class TestGenerics{
public static void main(String args[]){
MyGen<Integer> m=new MyGen<Integer>();

m.add(2);
//m.add("vivek");//Compile time error
System.out.println(m.get());
}}
```

Generics in java

- Type Parameters

- The type parameters naming conventions are important to learn generics thoroughly.

- The common type parameters are as follows:

- T - Type

- E - Element

- K - Key

- N - Number

- V - Value

Generics in java

- Generic Method
- Like the generic class, we can create a generic method that can accept any type of arguments.
- Here, the scope of arguments is limited to the method where it is declared. It allows static as well as non-static methods.
- TestGenericsDemo1.java