Marwadi University

Unit – 8

Language and Utility Framework

Prepared By

Prof. Ravikumar Natarajan

Assistant Professor, CE Dept.

KNOWLEDGE IS THE CURRENCY
FOR THE 21st CENTURY

Contents

- String class
- Character class
- StringBuffer class
- StringBuilder class
- Primitive type Wrapper classes
- **Collections overview**
- **Collection interfaces**
- **Collection classes**
- **Maps**
- **Comparators**
- **Lists**
- **Vector class**
- **Stack class**
- **Scanner**
- **Formatter**

String Class

Marwadi University

- In Java, string is basically an object that represents sequence of char values. An array of characters works same as Java string.
- For example:

```
char[] ch={'j','a','v','a'};
String s=new String(ch);
```

is same as:

```
String s="java";
```

String Class

Marwadi University

- String is a collection of characters. **In Java, string is an object that represents a sequence of characters.**
- The Java String is immutable which means it cannot be changed. Whenever we change any string, a new instance is created.
- Java String class provides a lot of methods to perform operations on strings such as compare(), concat(), equals(), split(), length(), replace(), compareTo(), intern(), substring() etc.
- **The java.lang.String class implements Serializable, Comparable and CharSequence interfaces.**
- **The CharSequence interface is used to represent the sequence of characters. String, StringBuffer and StringBuilder classes implement it. It means, we can create strings in Java by using these three classes.**

How to create a string object?

Marwadi University

- There are two ways to create String object:

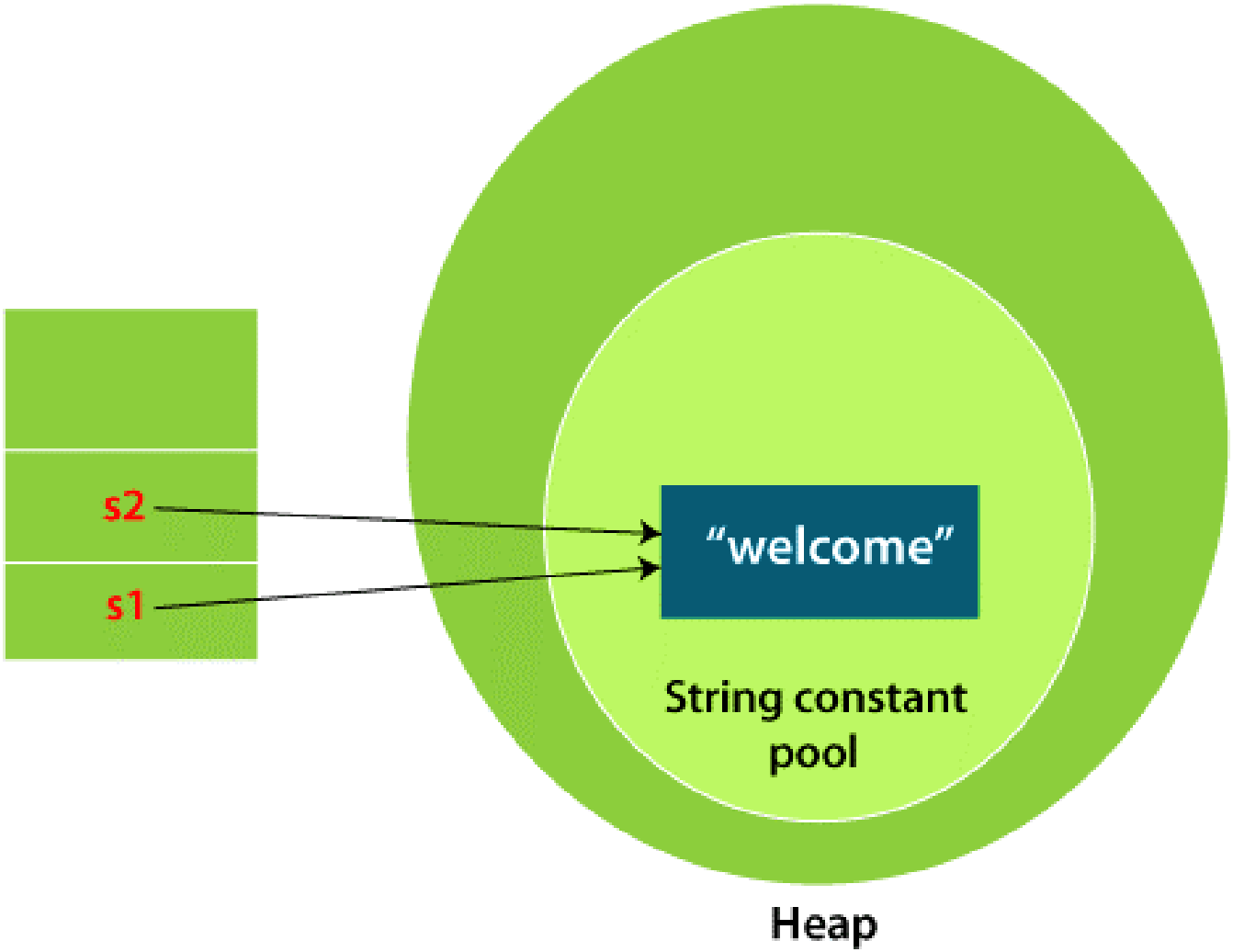
  1. **By string literal**
  2. **By new keyword**

- Method 1: Java String literal is created by using double quotes. For Example: String s="welcome";

- Each time you create a string literal, the JVM checks the "string constant pool" first. If the string already exists in the pool, a reference to the pooled instance is returned. If the string doesn't exist in the pool, a new string instance is created and placed in the pool. For example:

- String s1="Welcome";
- String s2="Welcome";//It doesn't create a new instance

How to create a string object?



In the above example, only one object will be created.

How to create a string object?

Marwadi University

Why Java uses the concept of String literal?

To make Java more **memory efficient** (because no new objects are created if it exists already in the string constant pool).

Method 2: By new keyword

String s=new String("Welcome"); //creates two objects and one reference variable

In such case, JVM will create a new string object in normal (non-pool) heap memory, and the literal "Welcome" will be placed in the string constant pool. The variable s will refer to the object in a heap (non-pool).

How to create a string object?


```
public class Main{
public static void main(String args[]){
String s1="java";
char ch[]={'s','t','r','i','n','g','s'};
String s2=new String(ch);
String s3=new String("example");
 System.out.println(s1);
System.out.println(s2);
System.out.println(s3);
}}
```

Immutable string (Cannot change object)

- String class represents immutable string that means once one has created a string object it cannot be changed.
- If want to modify string object need to use StringBuffer class.

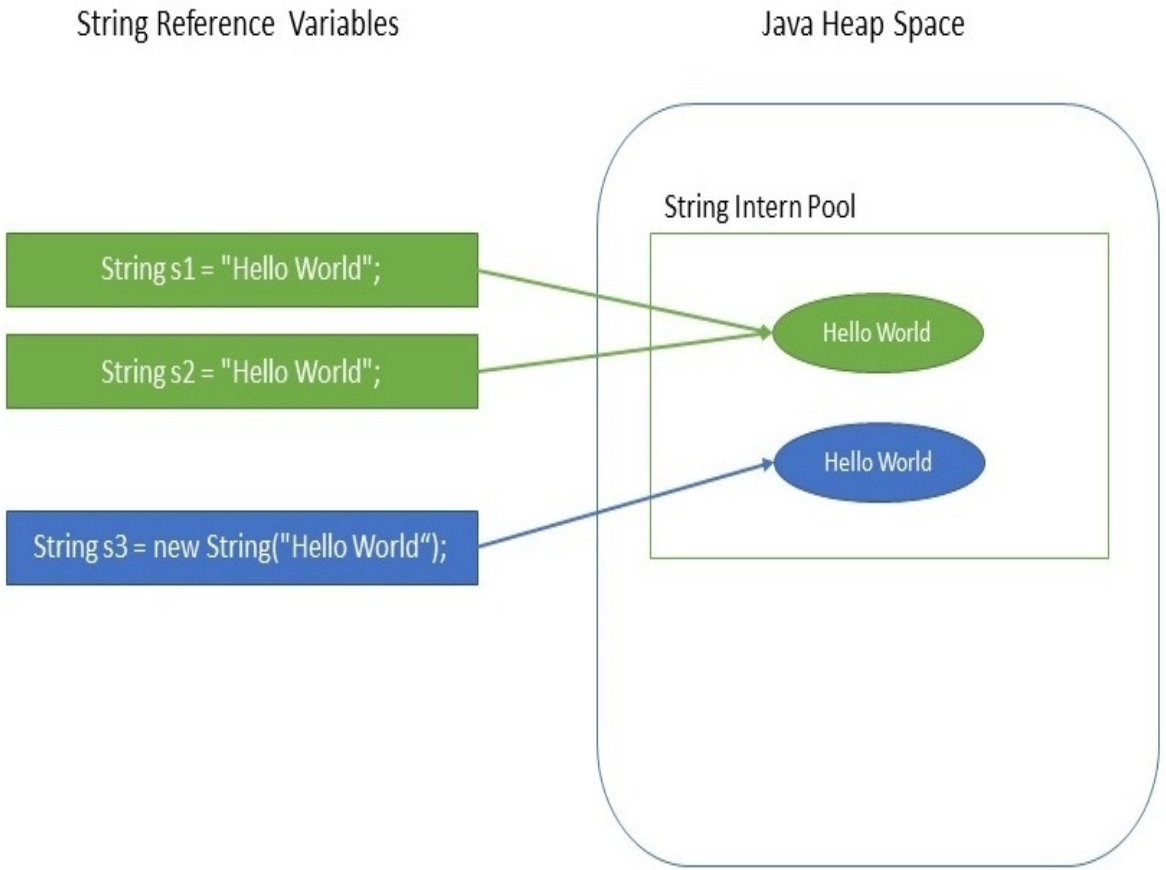
```
class a
{
public static void main (String[] args) {
{
String s = “Hi”;
s.concat(“Hello”); //appends string at end
System.out.println(s); //prints only Hi because strings are immutable.
}}
```

Advantages of Immutable String

- The key benefits of keeping this class as immutable are caching, security, synchronization, and performance.

What is Java String Pool?

Java String Pool is the special memory region where Strings are stored by the JVM. Since Strings are immutable in Java, the JVM optimizes the amount of memory allocated for them by storing only one copy of each literal String in the pool. This process is called interning.



Advantages of Immutable String

String Pool example

```
class Main{
public static void main (String[] args) {
   {
      String s1 = "Hello World";
      String s2 = "Hello World";
      String s3 = "Hello";
      String s4 = new String("Hello World");

      System.out.println(s1.equals(s2));
      System.out.println(s1.hashCode());
      System.out.println(s2.hashCode());
      System.out.println(s3.hashCode());
      System.out.println(s4.hashCode());

      System.out.println(s1==s2); //true
      System.out.println(s2==s4);  //false
} } }
```

//s4 hashCode ll be same as s1 and s2. Because new object is created in non heap but as the literal is same it ll be placed in constant pool.

To justify check the reference of s1 = = s2 and s2 == s4. Here s2==s4 will be false. It shows s4 is from different address.

Advantages of Immutable String

String Pool example

```
class Main{
public static void main (String[] args) {
   {
      String s1 = "Hello World";
      String s2 = "Hello World";
      String s3 = "Hello";
      String s4 = new String("Hello World").intern();

      System.out.println(s1.equals(s2));
      System.out.println(s1.hashCode());
      System.out.println(s2.hashCode());
      System.out.println(s3.hashCode());
      System.out.println(s4.hashCode());

      System.out.println(s1==s2); //true
      System.out.println(s2==s4);  //Now this will be true, because intern means it will
create a copy of the object, not the new object.
} }  }
```

String Class

```
class Main
{
public static void main(String args[])
{
String s ="Hello";
System.out.println(s.length());

char ch;
ch =s.charAt(2);
System.out.println(ch);


//substring
System.out.println("Substring:"
+s.substring(2,5));

//replace
String str;
str=s.replace('H','Y');
System.out.println(str);


//Upper
String str1 = s.toUpperCase();
System.out.println(str1);

//Lower
String str2 = s.toLowerCase();
System.out.println(str2);

} }
```

String Class

- String Constructor
- String Concatenation
- length() method
- charAt() method
- getChars() method
- Equals() mthod
- equalsIgnoreCase() method
- startsWith() method
- endsWith() method
- compareTo() method
- indexOf() method with char
- indexOf() method with substring
- substring() method
- concat() method
- replace() method
- trim() method
- Changing the case

Overriding toString() method

- The default toString() method from object class prints "Class name @ hashcode".
- Default toString() method can be overridden

```
class Main
{
int rollno;
String sname;

Main(int rollno, String sname)
{
this.rollno=rollno;
this.sname=sname;
}
public String toString() //override
{
return rollno + " " +sname;
}
public static void main(String
args[])
{
Main x1 = new Main(101,"Ram");
System.out.println(x1);
//xxx@3e25a5
}}
```

Java StringBuffer Class

Marwadi University

- Java StringBuffer class is used to create mutable (modifiable) String objects.
- The StringBuffer class in Java is the same as String class except it is mutable i.e. it can be changed.
- StringBuffer and StringBuilder classes are used for creating mutable strings.

```
class Main{
public static void main(String args[]){
StringBuffer sb=new StringBuffer("Hello ");
sb.append("Java");//now original string is changed
System.out.println(sb);//prints Hello Java
}
}
```

Java StringBuffer Class


```
class Main{
public static void main(String args[]){
StringBuffer sb=new StringBuffer("Hello");
sb.reverse();
System.out.println(sb);//prints olleH
}
}
```

Java StringBuilder Class


```
class Main{
public static void main(String args[]){
StringBuilder sb=new StringBuilder("Hello ");
sb.insert(1,"Java");//now original string is changed
System.out.println(sb);//prints HJavaello   }   }
```

No.	StringBuffer	StringBuilder
1)	StringBuffer is synchronized i.e. thread safe. It means two threads can't call the methods of StringBuffer simultaneously.	StringBuilder is non-synchronized i.e. not thread safe. It means two threads can call the methods of StringBuilder simultaneously.
2)	StringBuffer is less efficient than StringBuilder.	StringBuilder is more efficient than StringBuffer.
3)	StringBuffer was introduced in Java 1.0	StringBuilder was introduced in Java 1.5

Java StringBuilder Class

- StringBuilder()
- StringBuilder(capacity)
- StringBuilder(str)
- append(data)
- append(data,offset,len)
- append(val)
- append(str)
- delete(startIndex,endIndex)
- deleteCharAt(index)
- insert(index,data,offset,len)
- insert(offset, data)
- insert(offset,val)
- insert(offset,str)
- replace(startIndex,endIndex,str)
- reverse()
- setCharAt(index,ch)
- toString()
- capacity()
- charAt(index)
- length()
- setLength(newLength)
- substring(startIndex)
- substring(startIndex,endIndex)
- trimToSize()

Primitive Data Type and Wrapper Class Types

Marwadi University

- A primitive type value is not an object, but it can be wrapped in an object using a wrapper class in the Java API.

- Java offers a convenient way to incorporate, or wrap, a primitive data type into an object.
- e.g. wrapping int into the Integer class, wrapping double into the Double class, and wrapping char into the Character class etc.
- Java provides Boolean, Character, Double, Float, Byte, Short, Integer, and Long wrapper classes in the java.lang package for primitive data types.

Wrapper Classes

Marwadi University

- The wrapper class in Java provides the mechanism to convert primitive into object and object into primitive.

- Since J2SE 5.0, autoboxing and unboxing feature convert primitives into objects and objects into primitives automatically. The automatic conversion of primitive into an object is known as autoboxing and vice-versa unboxing.

Use of Wrapper classes in Java

Marwadi University

- Change the value in Method: Java supports only call by value. So, if we pass a primitive value, it will not change the original value. But, if we convert the primitive value in an object, it will change the original value.
- Serialization: We need to convert the objects into streams to perform the serialization. If we have a primitive value, we can convert it in objects through the wrapper classes.
- Synchronization: Java synchronization works with objects in Multithreading.
- java.util package: The java.util package provides the utility classes to deal with objects.
- Collection Framework: Java collection framework works with objects only. All classes of the collection framework (ArrayList, LinkedList, Vector, HashSet, LinkedHashSet, TreeSet, PriorityQueue, ArrayDeque, etc.) deal with objects only.

Use of Wrapper classes in Java

Marwadi University

The eight classes of the java.lang package are known as wrapper classes in Java. The list of eight wrapper classes are given below:

Primitive Type	**Wrapper class**
boolean	Boolean
char	Character
byte	Byte
short	Short
int	Integer
long	Long
float	Float
double	Double

Use of Wrapper classes in Java

Marwadi University

```
//Java program to convert primitive into objects
//Autoboxing example of int to Integer

public class WrapperExample1{
public static void main(String args[]){

//Converting int into Integer
int a=20;
Integer i=Integer.valueOf(a);    //converting int into Integer explicitly
Integer j=a;   //autoboxing, now compiler will write Integer.valueOf(a) internally

System.out.println(a+" "+i+" "+j);
}}
```

Use of Wrapper classes in Java

Marwadi University

```
//Java program to convert object into primitives
//Unboxing example of Integer to int

public class WrapperExample2{
public static void main(String args[]){

//Converting Integer to int
Integer a=new Integer(3);
int i=a.intValue();//converting Integer to int explicitly

int j=a;   //unboxing, now compiler will write a.intValue() internally

System.out.println(a+" "+i+" "+j);
}}
```

Big integer and Big decimal class

Marwadi University

- The BigInteger and BigDecimal classes can be used to represent integers or decimal numbers of any size and precision.

- If you need to compute with very large integers or high-precision floating-point values, you can use the BigInteger and BigDecimal classes in the java.math package.
- Both are immutable.

Big Integer

Marwadi University

```
import java.math.*;
public class Main{
public static void main(String[] args) {
int aa = 92233720368547758O7;
int bb = 2 ;
int cc = aa*bb;
System.out.println(cc); //Will not work

BigInteger a = new BigInteger("92233720368547758O7");
BigInteger b = new BigInteger("2");
BigInteger c = a.multiply(b); // 92233720368547758O7 * 2
System.out.println(c); //This will work
}}
```

Big Decimal


```
import java.math.*;
public class Main{
public static void main(String[] args)    {

BigDecimal bd1 = new BigDecimal("124567890.0987654321");
BigDecimal bd2 = new BigDecimal("987654321.123456789");

bd1 = bd1.add(bd2);
System.out.println("BigDecimal1 = " + bd1);
bd1 = bd1.multiply(bd2);
System.out.println("BigDecimal1 = " + bd1);
}}
```

Character Class

Marwadi University

- Java provides a wrapper class Character in java.lang package. An object of type Character contains a single field, whose type is char. The Character class offers a number of useful class (i.e., static) methods for manipulating characters. You can create a Character object with the Character constructor.

- Creating a Character object:
- Character ch = new Character('a');

- **Methods in Character Class**
  - **boolean isLetter(char ch)**
  - **boolean isDigit(char ch)**
  - **boolean isWhitespace(char ch), etc**

Character Class


```
public class Test {
    public static void main(String[] args)
    {
        System.out.println(Character.isLetter('A'));
        System.out.println(Character.isLetter('0'));

        System.out.println(Character.isDigit('A'));
        System.out.println(Character.isDigit('0'));

        System.out.println(Character.isWhitespace('A'));
        System.out.println(Character.isWhitespace(' '));

    }
}
```

Next

Collections overview,
Collection interfaces,
Collection classes,

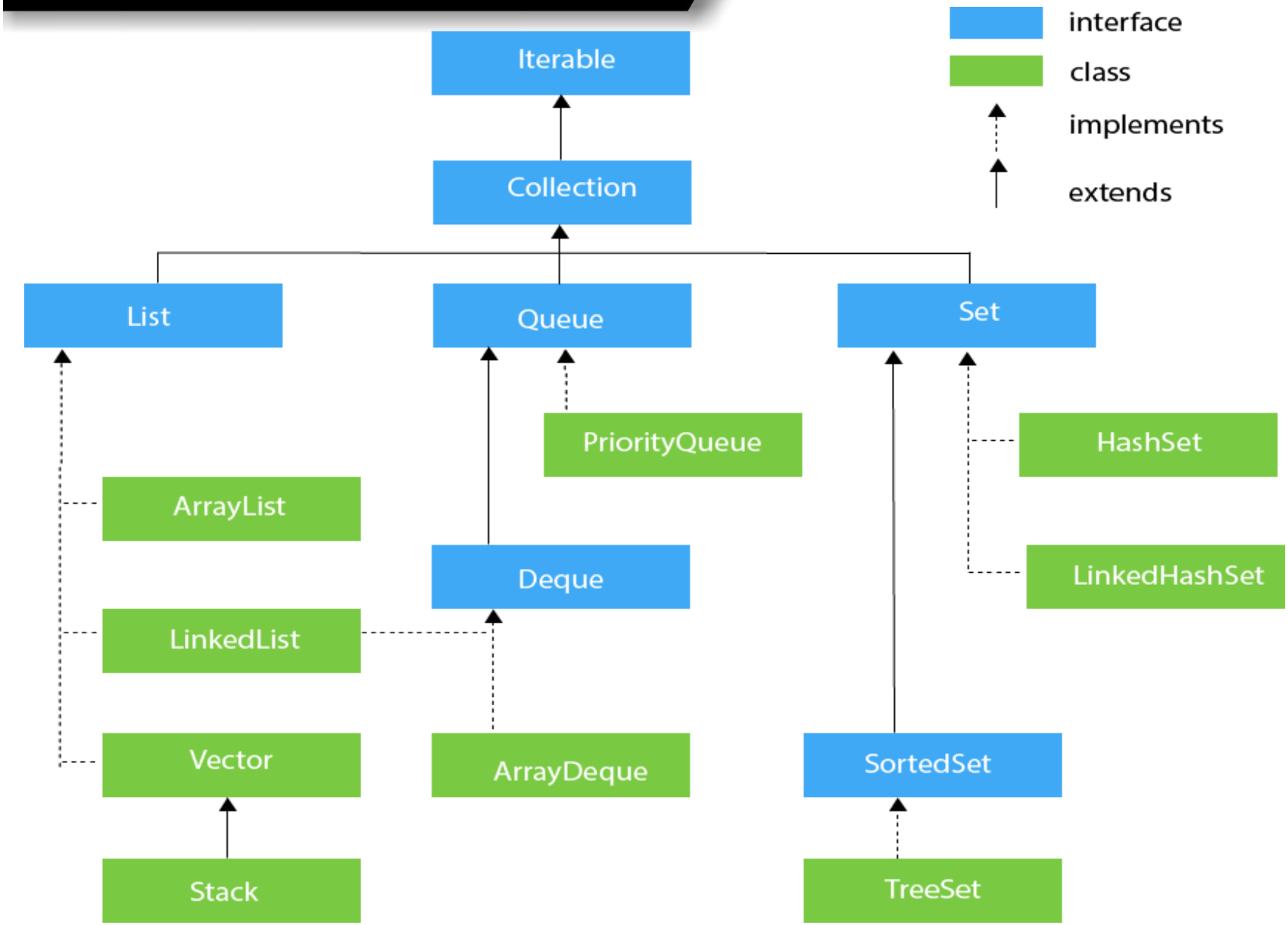
- Maps
- Comparators

- Lists
- Vector class
- Stack class
- Scanner
- Formatter

Collections

- A Collection represents a single unit of objects, i.e., a group.
- 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. Java Collection framework provides many interfaces **(Set, List, Queue, Deque)** and classes **(ArrayList, Vector, LinkedList, PriorityQueue, HashSet, LinkedHashSet, TreeSet).**

Collection Interface and Classes



Iterators

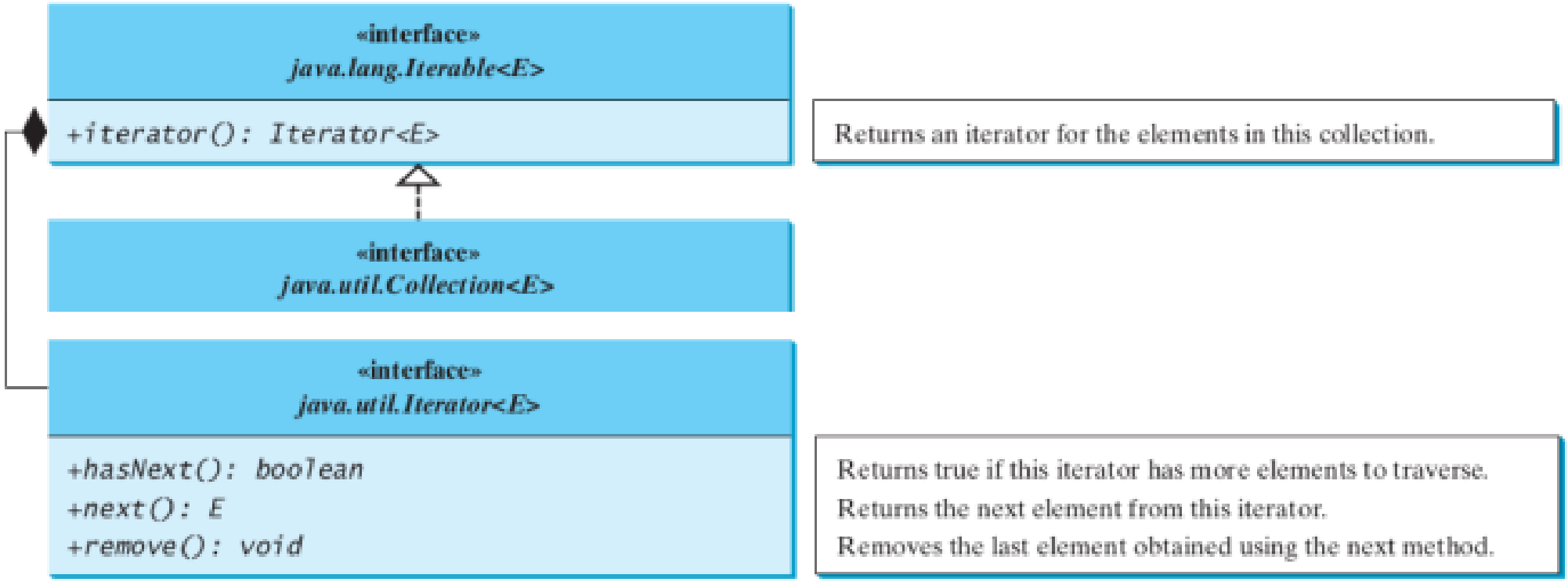
Marwadi University

- Iterator is a classic design pattern for walking through a data structure without having to expose the details of how data is stored in the data structure.

- The Collection interface extends the Iterable interface.

- The Iterable interface defines the iterator method, which returns an iterator.
- The Collection interface contains the methods for manipulating the elements in a collection, and you can obtain an iterator object for traversing elements in the collection.

Iterators


«interface»
java.lang.Iterable<E>
+iterator(): Iterator<E>
Returns an iterator for the elements in this collection.
«interface»
java.util.Collection<E>
«interface»
java.util.Iterator<E>
+hasNext(): boolean
+next(): E
+remove(): void
Returns true if this iterator has more elements to traverse.
Returns the next element from this iterator.
Removes the last element obtained using the next method.

Iterators


```
import java.util.*;

 public class Main {
 public static void main(String[] args) {
 Collection<String> collection = new ArrayList<>();
 collection.add("New York");
 collection.add("Atlanta");
 collection.add("Dallas");
 collection.add("Madison");

Iterator<String> iterator = collection.iterator();
while (iterator.hasNext()) {
System.out.print(iterator.next().toUpperCase() + " ");
}
System.out.println();  }  }
```

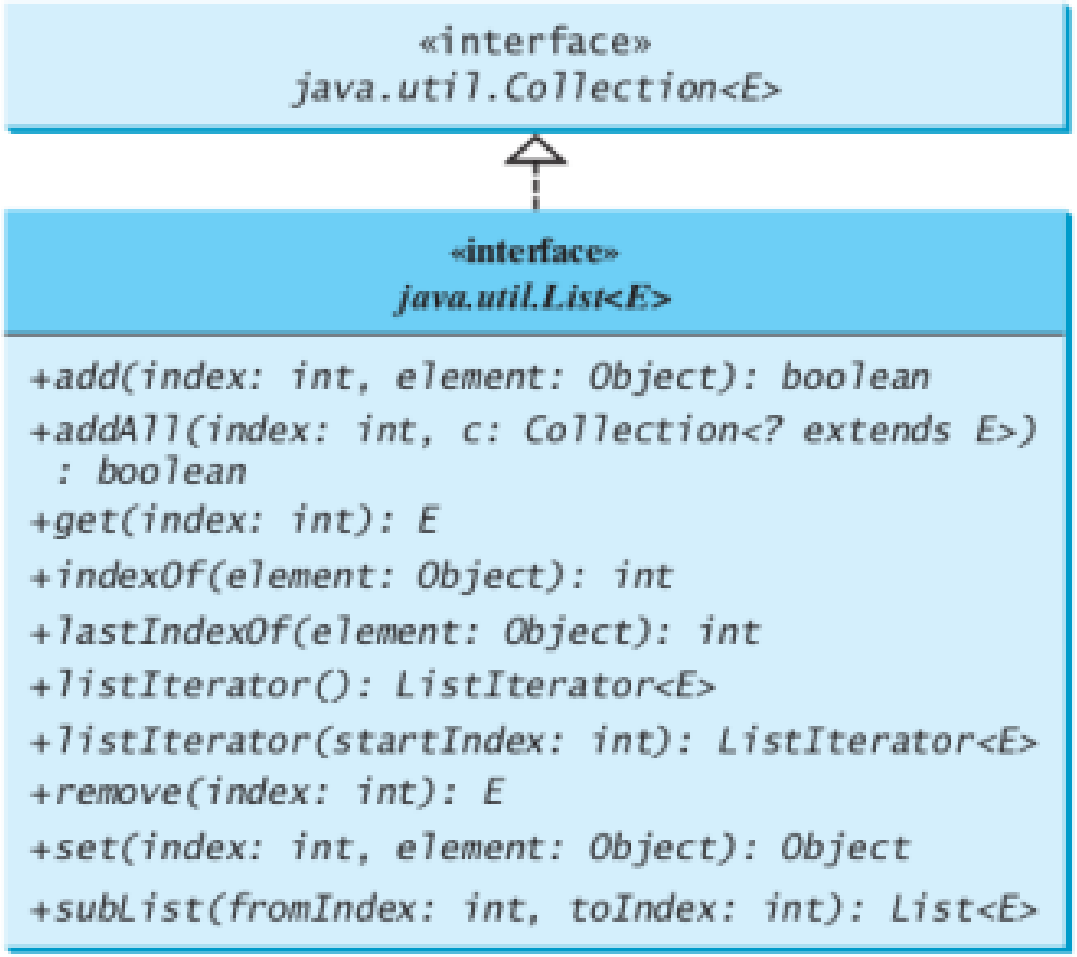
Output:
NEW YORK ATLANTA DALLAS MADISON

Lists

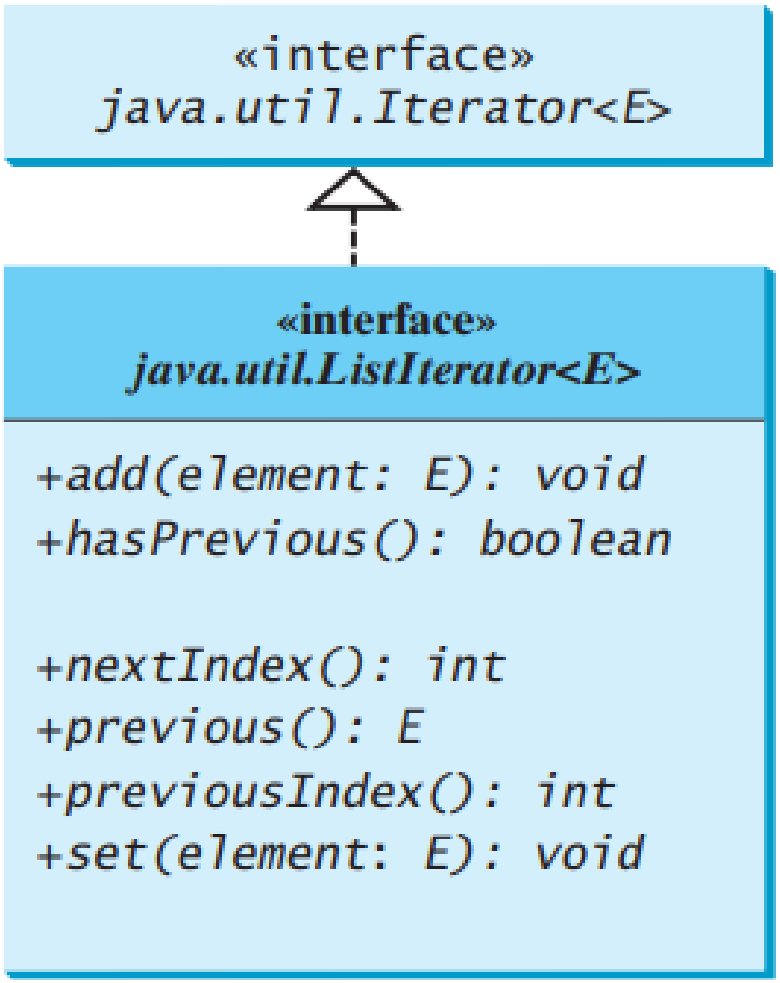
Marwadi University

- The List interface extends the Collection interface and defines a collection for storing elements in a sequential order.
- **To create a list, use one of its two concrete classes: ArrayList or LinkedList.**
- The List interface adds position-oriented operations, as well as a new list iterator that enables the user to traverse the list bidirectionally.

Lists


«interface»
java.util.Collection<E>
«interface»
java.util.List<E>
+add(index: int, element: Object): boolean
+addAll(index: int, c: Collection<? extends E>) : boolean
+get(index: int): E
+indexOf(element: Object): int
+lastIndexOf(element: Object): int
+listIterator(): ListIterator<E>
+listIterator(startIndex: int): ListIterator<E>
+remove(index: int): E
+set(index: int, element: Object): Object
+subList(fromIndex: int, toIndex: int): List<E>
Adds a new element at the specified index.
Adds all the elements in c to this list at the specified index.
Returns the element in this list at the specified index.
Returns the index of the first matching element.
Returns the index of the last matching element.
Returns the list iterator for the elements in this list.
Returns the iterator for the elements from startIndex.
Removes the element at the specified index.
Sets the element at the specified index.
Returns a sublist from fromIndex to toIndex-1.

List Iterator



- The listIterator() or listIterator(startIndex) method returns an instance of ListIterator.
- The ListIterator interface extends the Iterator interface to add bidirectional traversal of the list.

`+add(element: E): void`	Adds the specified object to the list.
`+hasPrevious(): boolean`	Returns true if this list iterator has more elements when traversing backward.
`+nextIndex(): int`	Returns the index of the next element.
`+previous(): E`	Returns the previous element in this list iterator.
`+previousIndex(): int`	Returns the index of the previous element.
`+set(element: E): void`	Replaces the last element returned by the previous or next method with the specified element.

Iterator vs List Iterator

- Iterator can traverse only in forward direction whereas ListIterator traverses both in forward and backward directions.

- Using Iterator we cannot modify or replace elements present in Collection. But, ListIterator can modify or replace elements with the help of set(E e).

- **Iterator methods are next(), remove(), hasNext()** whereas ListIterator methods are next(), previous(), hasNext(), hasPrevious(), add(E e).

ArrayList

Marwadi University

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.

- 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.**
- 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<Integer> al = **new** ArrayList<Integer>();

ArrayList

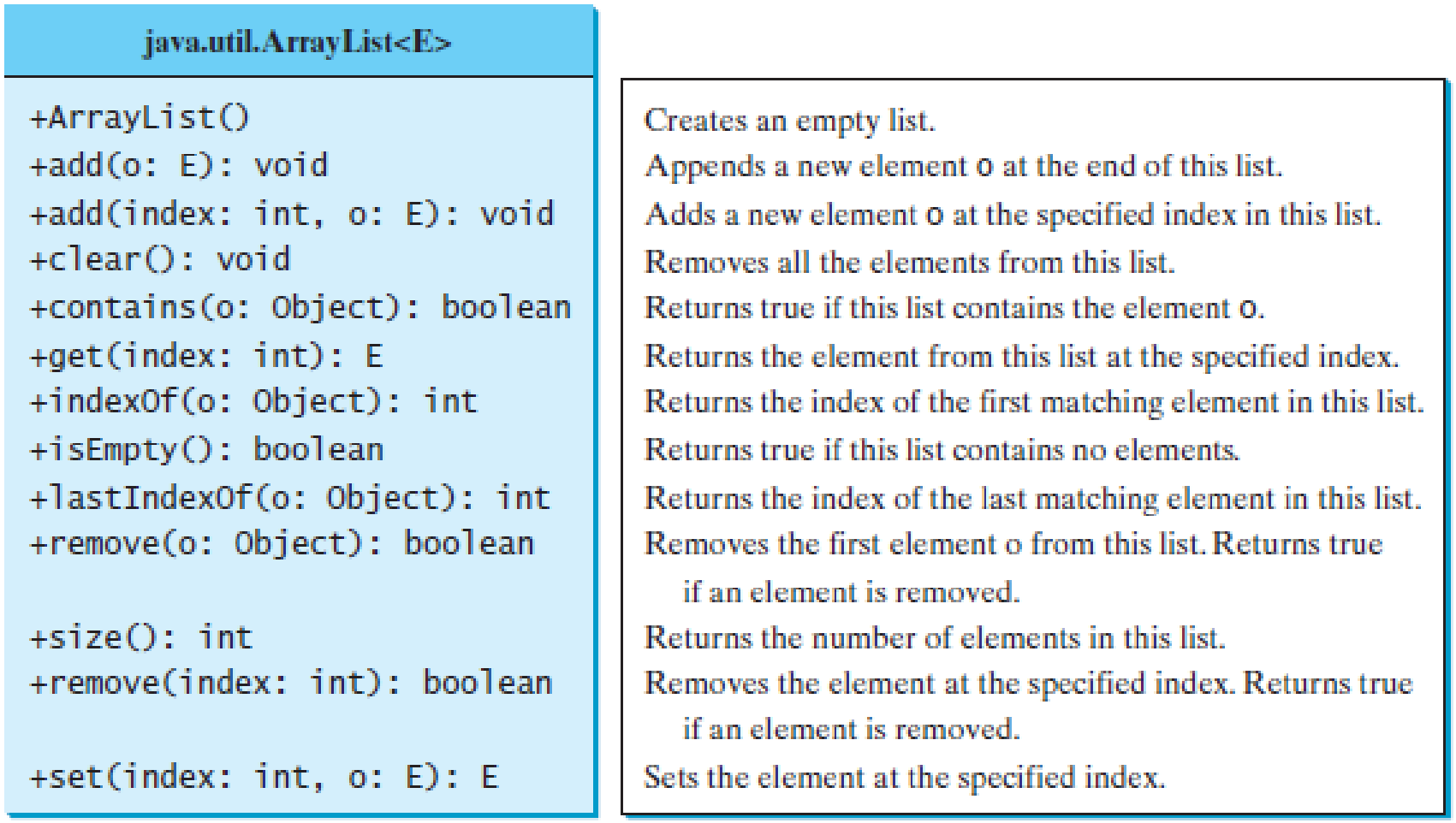


FIGURE 11.3 An `ArrayList` stores an unlimited number of objects.

ArrayList


```
import java.util.*;
public class Main{
public static void main(String args[]){
ArrayList<String> al=new ArrayList<String>(); //ArrayList al = new ArrayList();
al.add("Ravi");
al.add("Vijay");
al.add("Aarav");
al.add("Ajay");
for(String s2:al)
System.out.println("Elements:"+s2);

al.remove("Ravi");
for(String s:al)
System.out.println("After remove:"+s);

al.clear();
System.out.println(al.size());
} }
```

LinkedList

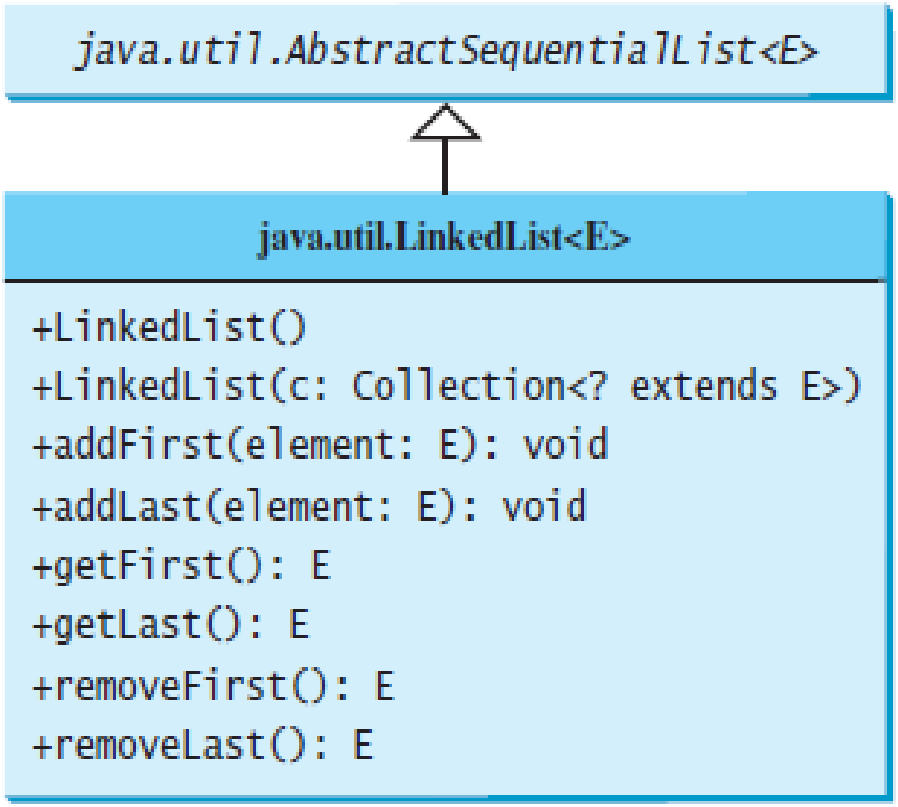
Marwadi University

Java LinkedList class uses a doubly linked list to store the elements.
It provides a linked-list data structure.
It inherits the AbstractList class and implements List and Deque interfaces.

The important points about Java LinkedList are:

- Java LinkedList class can contain duplicate elements.
- Java LinkedList class maintains insertion order.
- Java LinkedList class is non synchronized.
- **In Java LinkedList class, manipulation is fast because no shifting needs to occur.**
- Java LinkedList class can be used as a list, stack or queue.

LinkedList



Creates a default empty linked list.	
Creates a linked list from an existing collection.	
Adds the element to the head of this list.	
Adds the element to the tail of this list.	
Returns the first element from this list.	
Returns the last element from this list.	
Returns and removes the first element from this list.	
Returns and removes the last element from this list.	

LinkedList


```
import java.util.*;
public class Main{
public static void main(String args[]){
LinkedList<String> al=new LinkedList<String>();
al.add("Ravi");
al.add("Vijay");
System.out.println(al);

al.addFirst("Zumba");
System.out.println("After addFirst:"+al);

al.removeFirst();
System.out.println("After removeFirst:"+al);
} }
```

The ArrayList and LinkedList Classes

Marwadi University

- The ArrayList class and the LinkedList class are two concrete implementations of the List interface.
- ArrayList stores elements in an array. The array is dynamically created.
- If the capacity of the array is exceeded, a larger new array is created and all the elements from the current array are copied to the new array.
- LinkedList stores elements in a linked list. A list can grow or shrink dynamically.
- Use of these depends on your specific needs.
- **List or ArrayList when to choose?**
- If you need to support random access through an index without inserting or removing elements at the beginning of the list, **ArrayList** offers the most efficient collection.

- **If your application requires the insertion or deletion of elements at the beginning of the list, you should choose LinkedList.**

The ArrayList and LinkedList Classes

Marwadi University

Extra Example for reference

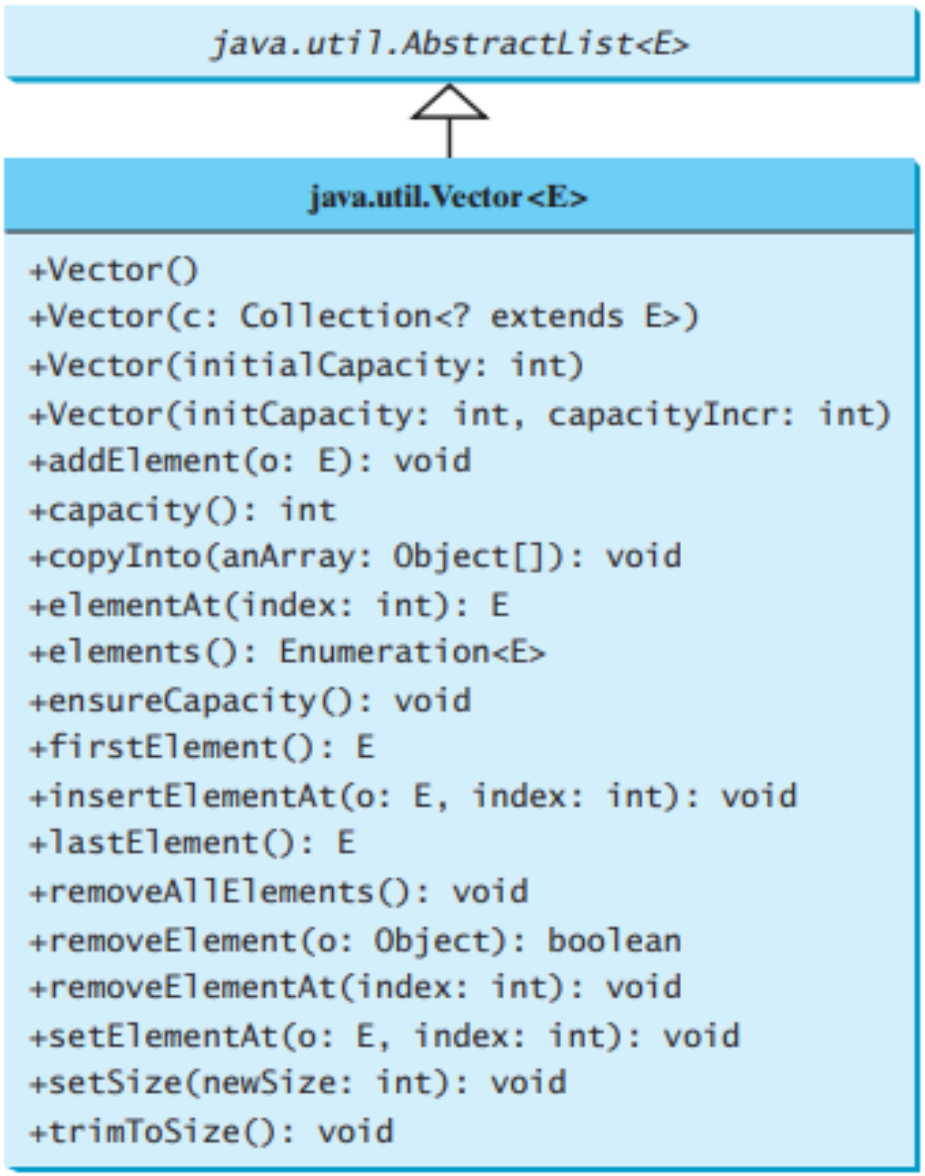
```
import java.util.*;

 public class Main {
 public static void main(String[] args) {
 List<Integer> arrayList = new ArrayList<>();
 arrayList.add(1); // 1 is autoboxed to new Integer(1)
 arrayList.add(2);
 arrayList.add(3);
 arrayList.add(1);
arrayList.add(4);
arrayList.add(0, 10);
arrayList.add(3, 30);

System.out.println("A list of integers in the array list:");
System.out.println(arrayList);

 LinkedList<Object> linkedList = new
LinkedList<>(arrayList);
 linkedList.add(1, "red");
 linkedList.removeLast();
 linkedList.addFirst("green");
System.out.println("Display the linked list forward:");
ListIterator<Object> listIterator = linkedList.listIterator();
while (listIterator.hasNext()) {
System.out.print(listIterator.next() + " ");
}
System.out.println();

System.out.println("Display the linked list backward:");
listIterator = linkedList.listIterator(linkedList.size());
while (listIterator.hasPrevious()) {
System.out.print(listIterator.previous() + " ");
}
}
}
```

Vector and Stack Classes

Marwadi University

- The Collections class contains static methods to perform common operations in a collection and a list.
- The Collections class contains the sort, binarySearch, reverse, shuffle, copy, and fill methods for lists, and **max, min, disjoint, and frequency methods for collections.**
- **Vector is a subclass of AbstractList, and Stack is a subclass of Vector in the Java API.**
- Vector is the same as ArrayList, except that it contains synchronized methods for accessing and modifying the vector. Synchronized methods can prevent data corruption when a vector is accessed and modified by two or more threads concurrently.

Vector Classes



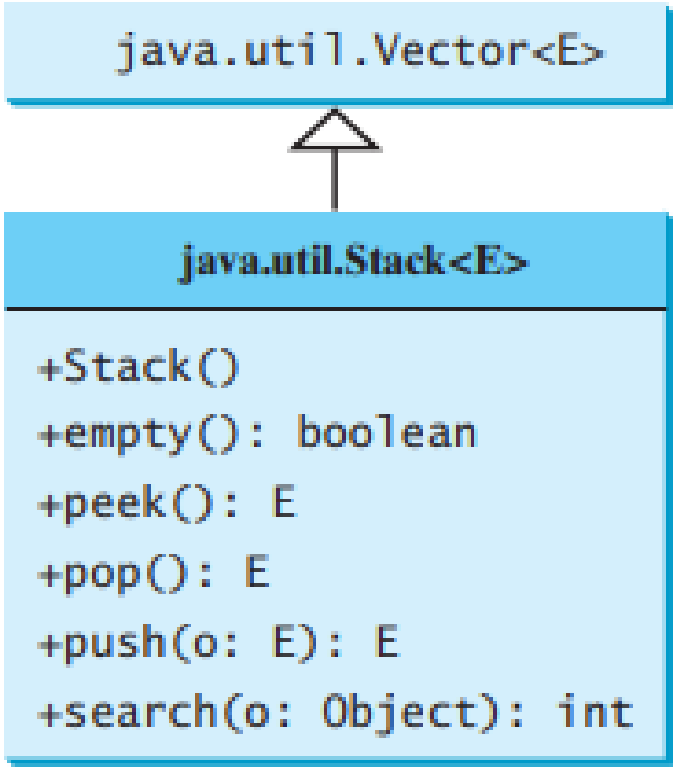
Creates a default empty vector with initial capacity 10.	
Creates a vector from an existing collection.	
Creates a vector with the specified initial capacity.	
Creates a vector with the specified initial capacity and increment.	
Appends the element to the end of this vector.	
Returns the current capacity of this vector.	
Copies the elements in this vector to the array.	
Returns the object at the specified index.	
Returns an enumeration of this vector.	
Increases the capacity of this vector.	
Returns the first element in this vector.	
Inserts o into this vector at the specified index.	
Returns the last element in this vector.	
Removes all the elements in this vector.	
Removes the first matching element in this vector.	
Removes the element at the specified index.	
Sets a new element at the specified index.	
Sets a new size in this vector.	
Trims the capacity of this vector to its size.	

Vector Classes


```
// Java Program to Add Elements in Vector Class
import java.util.*;
class Main {
        public static void main(String[] arg)
        {
            Vector v1 = new Vector();
           //Vector<Integer> v2 = new Vector<Integer>();

                v1.add(1);
                v1.add(2);
                v1.add("Hi");
                v1.add("Hello");
                v1.add(3);

                System.out.println("Vector v1 is " + v1);
        }
}
```

Stack Classes

- In the Java Collections Framework, Stack is implemented as an extension of Vector.
- Stack follows LIFO



+Stack()	Creates an empty stack.
+empty(): boolean	Returns true if this stack is empty.
+peek(): E	Returns the top element in this stack.
+pop(): E	Returns and removes the top element in this stack.
+push(o: E): E	Adds a new element to the top of this stack.
+search(o: Object): int	Returns the position of the specified element in this stack.

Stack Classes

Marwadi University

```
// Java program to add & remove the elements in the stack
import java.util.*;

class Main {
        public static void main(String[] args)
        {
                Stack stack1 = new Stack();
                stack1.push(4);
                stack1.push("All");
                stack1.push("Hello");

                System.out.println(stack1);

                stack1.pop();
                System.out.println(stack1);
        } }
```

Scanner Classes

Marwadi University

Already discussed

Formatter Class

Marwadi University

- The Formatter is a built-in class in java used for layout justification and alignment, common formats for numeric, string, and date/time data, and locale-specific output in java programming.

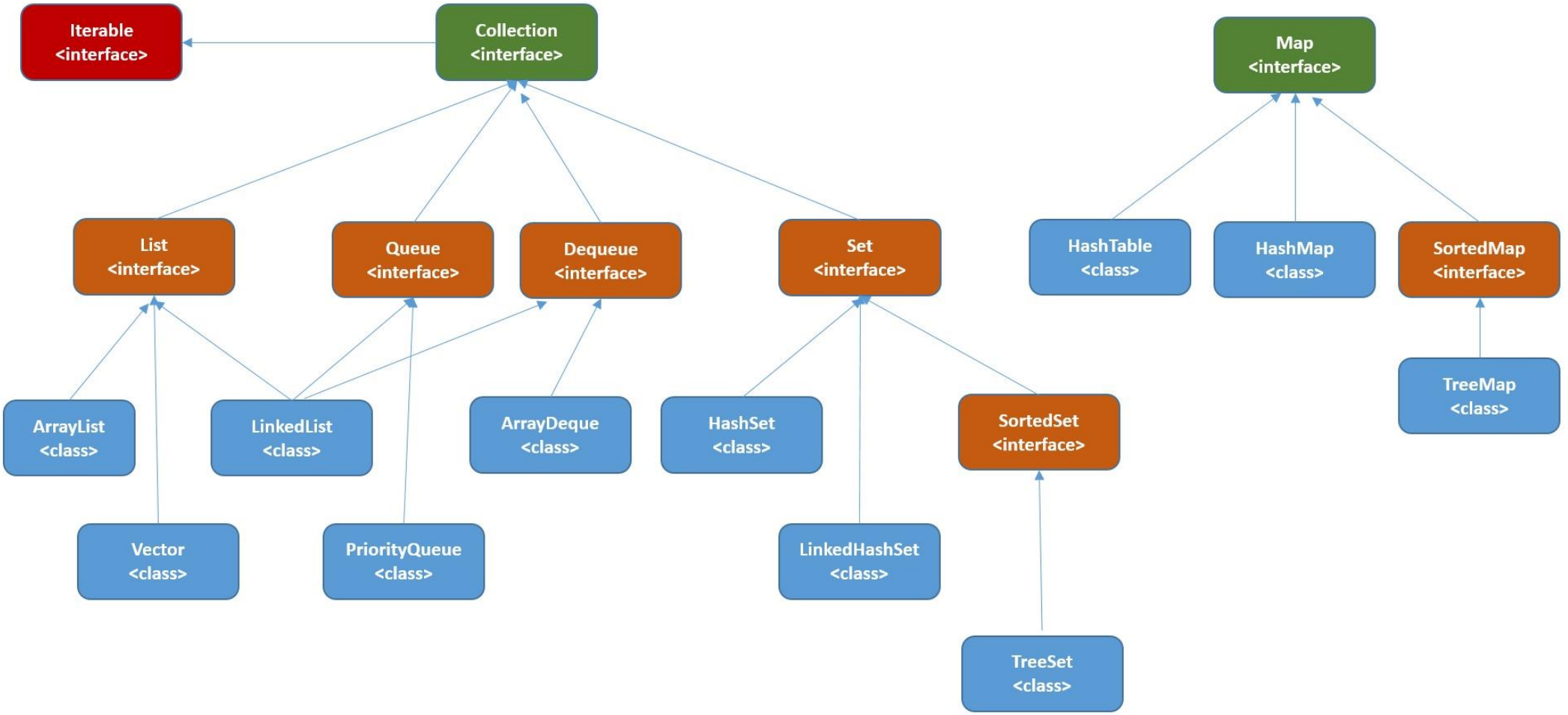
- The Formatter class is defined as final class inside the java.util package.

```
import java.util.*;
 public class Main {
 public static void main(String[] args) {
 Formatter f = new Formatter();
 f.format("%s age is %d", "MU", 22);
 System.out.println(f);
}}
```

The ArrayList and LinkedList Classes

Marwadi University

Collection Framework Hierarchy



Maps

Marwadi University

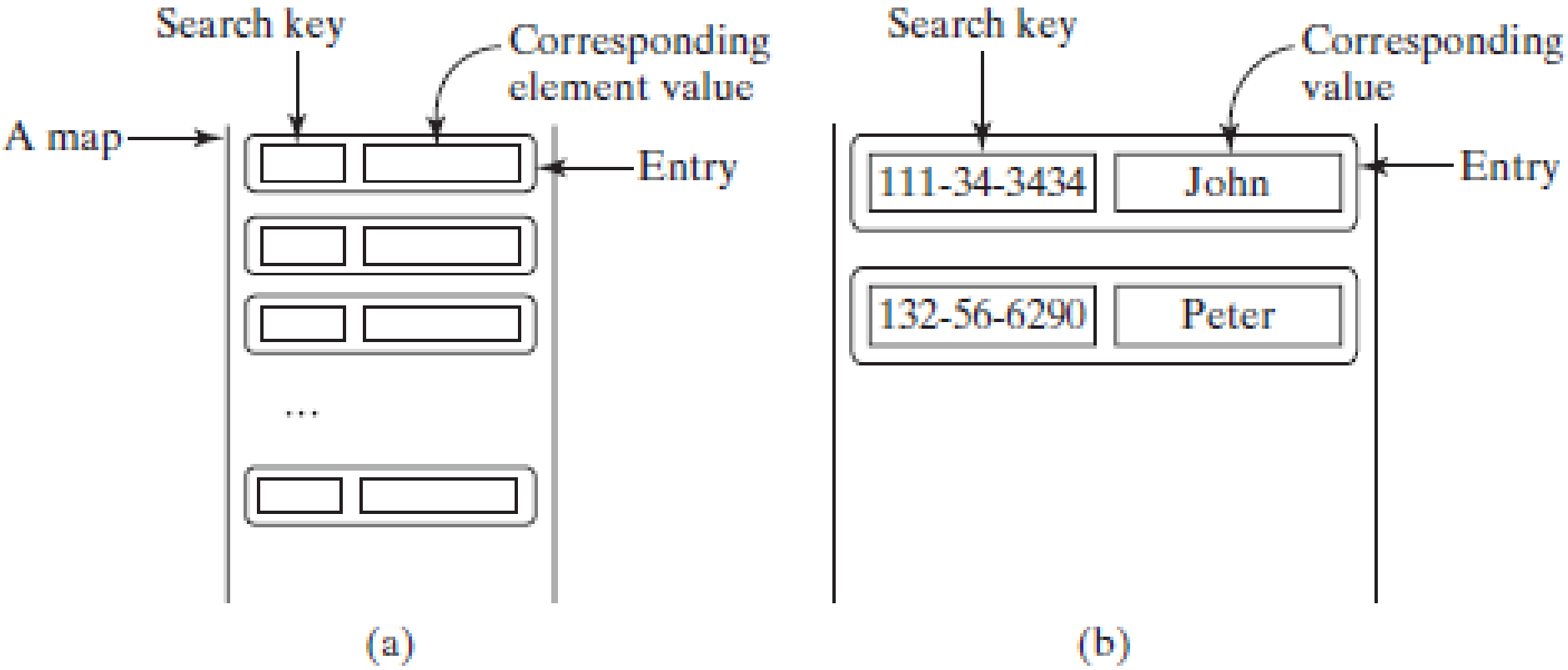
- You can create a map using one of its three concrete classes: HashMap, LinkedHashMap, or TreeMap.

- A map is a container object that stores a collection of key/value pairs. It enables fast retrieval, deletion, and updating of the pair through the key.
- A map stores the values along with the keys. The keys are like indexes.
- **In List, the indexes are integers. In Map, the keys can be any objects.**

- **A map cannot contain duplicate keys.**

- Each key maps to one value. A key and its corresponding value form an entry stored in a map.

Maps



The entries consisting of key/value pairs are stored in a map.

Maps

«interface»
java.util.Map<K,V>

```
+clear(): void
+containsKey(key: Object): boolean

+containsValue(value: Object): boolean

+entrySet(): Set<Map.Entry<K,V>>
+get(key: Object): V
+isEmpty(): boolean
+keySet(): Set<K>
+put(key: K, value: V): V
+putAll(m: Map<? extends K,? extends
 V>): void
+remove(key: Object): V
+size(): int
+values(): Collection<V>
```

Removes all entries from this map.

Returns true if this map contains an entry for the specified key.

Returns true if this map maps one or more keys to the specified value.

Returns a set consisting of the entries in this map.

Returns the value for the specified key in this map.

Returns true if this map contains no entries.

Returns a set consisting of the keys in this map.

Puts an entry into this map.

Adds all the entries from `m` to this map.

Removes the entries for the specified key.

Returns the number of entries in this map.

Returns a collection consisting of the values in this map.

Maps

Marwadi University

```
import java.util.*;
public class Main {
public static void main(String[] args) {

Map<String, String> map = new LinkedHashMap<>();

map.put("123", "John Smith");
map.put("111", "George Smith");
map.put("123", "Steve Yao");
map.put("222", "Steve Yao");
System.out.println(map);

}
}
```

Output:

{123=Steve Yao, 111=George Smith, 222=Steve Yao}

Comparator Interface

A comparator interface is used to order the objects of user-defined classes. A comparator object is capable of comparing two objects of the same class.

```
 import java.util.*;
  public class Main {
  public static void main(String[] args) {
  String[] cities = {"Savannah", "Boston", "Atlanta", "Tampa"};
  Arrays.sort(cities);
  for (String city: cities)
  System.out.print(city + " ");
  System.out.println();
}}
```

Marwadi University

Queries??

END OF UNIT – 8