Objective:

At the end of this lecture, students will understand :

- What is a Framework and Collection Framework in Java?
- Types of Interface in Java and Subtypes of Collections in Java
- Iterable Interface in Java and Collection Interface in Java
- Different Collection Interface Methods
- Iterator Interface Methods in Java
- List Interface: ArrayList, LinkedList, Vector, Stack
- Queue Interface: PriorityQueue
- Deque Interface: ArrayDeque
- Set Interface: HashSet, LinkedHashSet
- SortedSet Interface: TreeSet

What is a Framework in Java?

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

What is Collection Framework in Java?

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

- Interfaces and its implementations, i.e., classes
- Algorithm

What are Collections in Java?

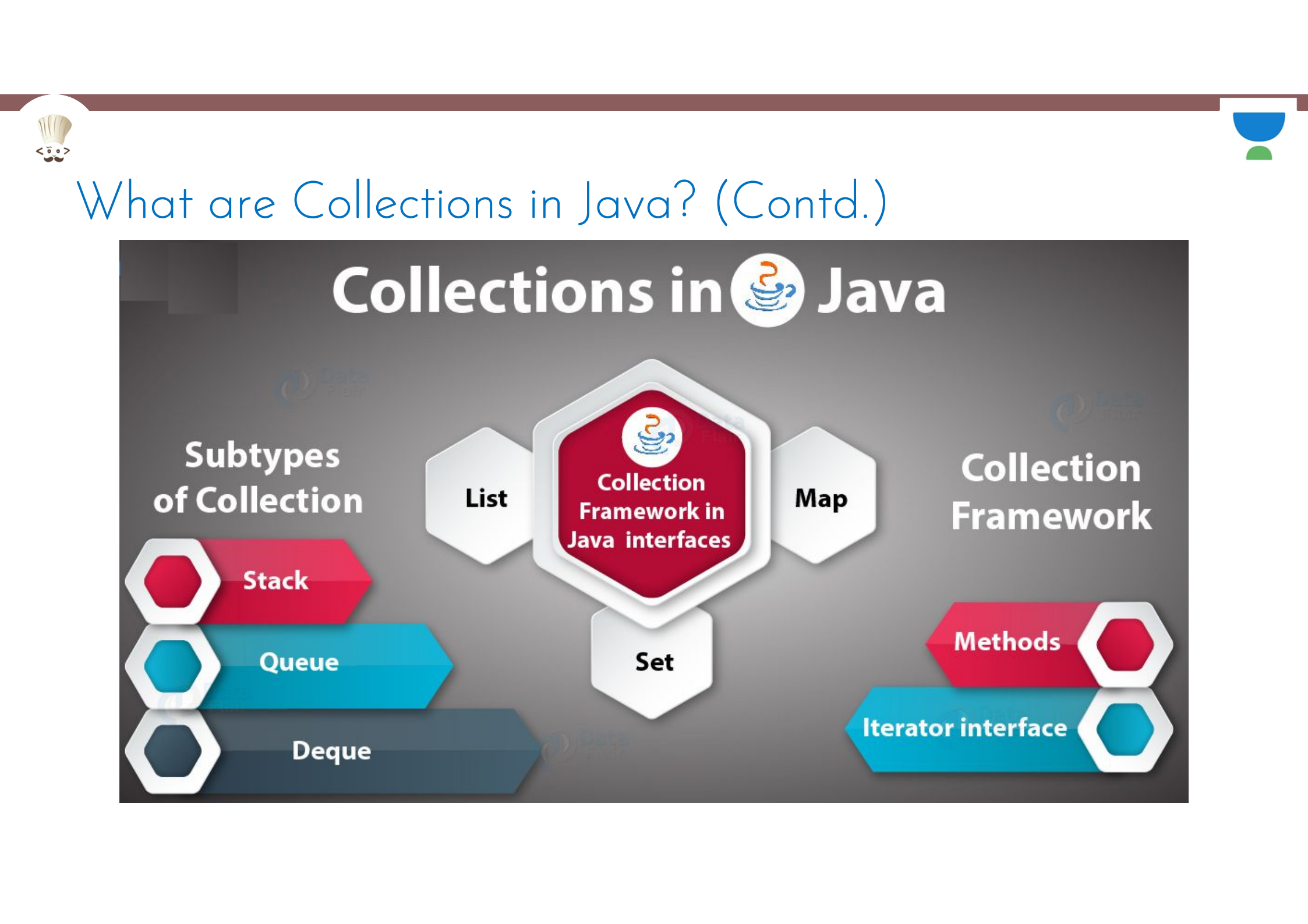
- Java collections are set of Java classes that assist the objects to group them and manage them.
- The least complex technique to use the arrays but, as we know for some situations it is too complex.
- For instance, array can’t change the size and also the elements can’t be effectively embedded.
- Arrays don’t permit indexing components by key (ex: string).
- To overcome such constraints specific classes are required.

What are Collections in Java? (Contd.)

The three main interfaces in Java Collections are –

- Set
- List
- Map

- The subtypes of collections in Java are –
  - Stack
  - Queue
  - Deque



What are Collections in Java? (Contd.)

Types of Interface in Java:

a. Set Interface in Java

Set accumulation that doesn’t permit copies and furthermore doesn’t permit getting to components by index. Rather, it gives techniques that check if component or components exist.

Types of Interface in Java: (Contd.)

a. Set Interface in Java (Contd.)

EnumSet – Enumset particular class to work with enum types.

HashSet – HashSet keeps an unordered list of components (arrange is eccentric).

LinkedHashSet – LinkedHashSet keeps requested rundown of components.

TreeSet – Treeset makes sure that there are no duplicates.

SortedSet – Sortedset provides ordering on its elements.

Types of Interface in Java: (Contd.)

b. List Interface in Java

Java list is accumulation that permits copies and carries on, like arrays (file components by the whole number) yet is more adaptable. To start with component has list = 0, last one has record = length - 1.

ArrayList – Keeps an unordered list of components utilizing exhibit.

LinkedList – Keeps requested list of components utilizing doubly-connected rundown.

Vector – For the most part, the same as ArrayList, however, it is string safe.

Types of Interface in Java: (Contd.)

c. Map Interface in Java

Java Map Interface is accumulation that permits copies and is like rundown with the exception of that record components by (key can be any protest) Map can be expected as an affiliated exhibit.

HashMap – Keeps unordered rundown of list utilizing exhibit

LinkedHashMap – Keeps requested list of components utilizing doubly-connected rundown.

Types of Interface in Java: (Contd.)

c. Map Interface in Java (Contd.)

TreeMap – Keeps requested list of components utilizing RBT (Red Black Tree). Components are requested by regular request or by a custom comparator.

Hashtable – Keeps an unordered list of components as HashMap, however, it is synchronized. This class is outdated.

EnumMap – Keeps ordered collection and are maintained in natural order.

Properties – It is subclass of HashTable. Provides methods to read and store data in properties file.

Subtypes of Collections in Java:

Java Stack – Elements include (push) and remove (i.e. pop) from best of accumulation. This rule is called LIFO (Last In, First Out)

Java Line or Java Queue – Elements include (push) and removed (pop) all together they include. This guideline is called FIFO (First In, First Out)

Deque – Elements include and expel from the two sides. Name deque (articulated “Deck”) is an easy route for “double finished line”.

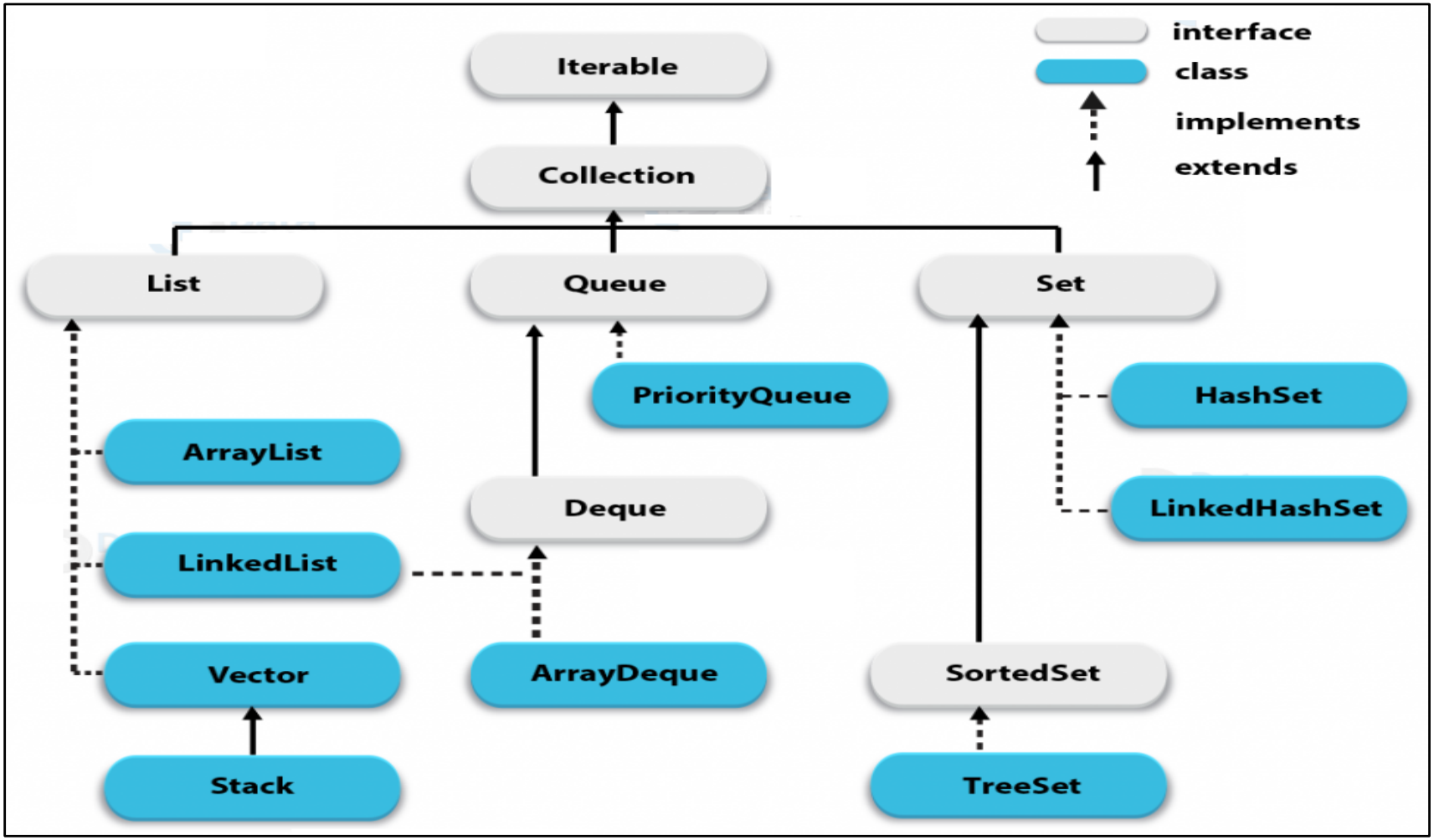
Collection Framework in Java:

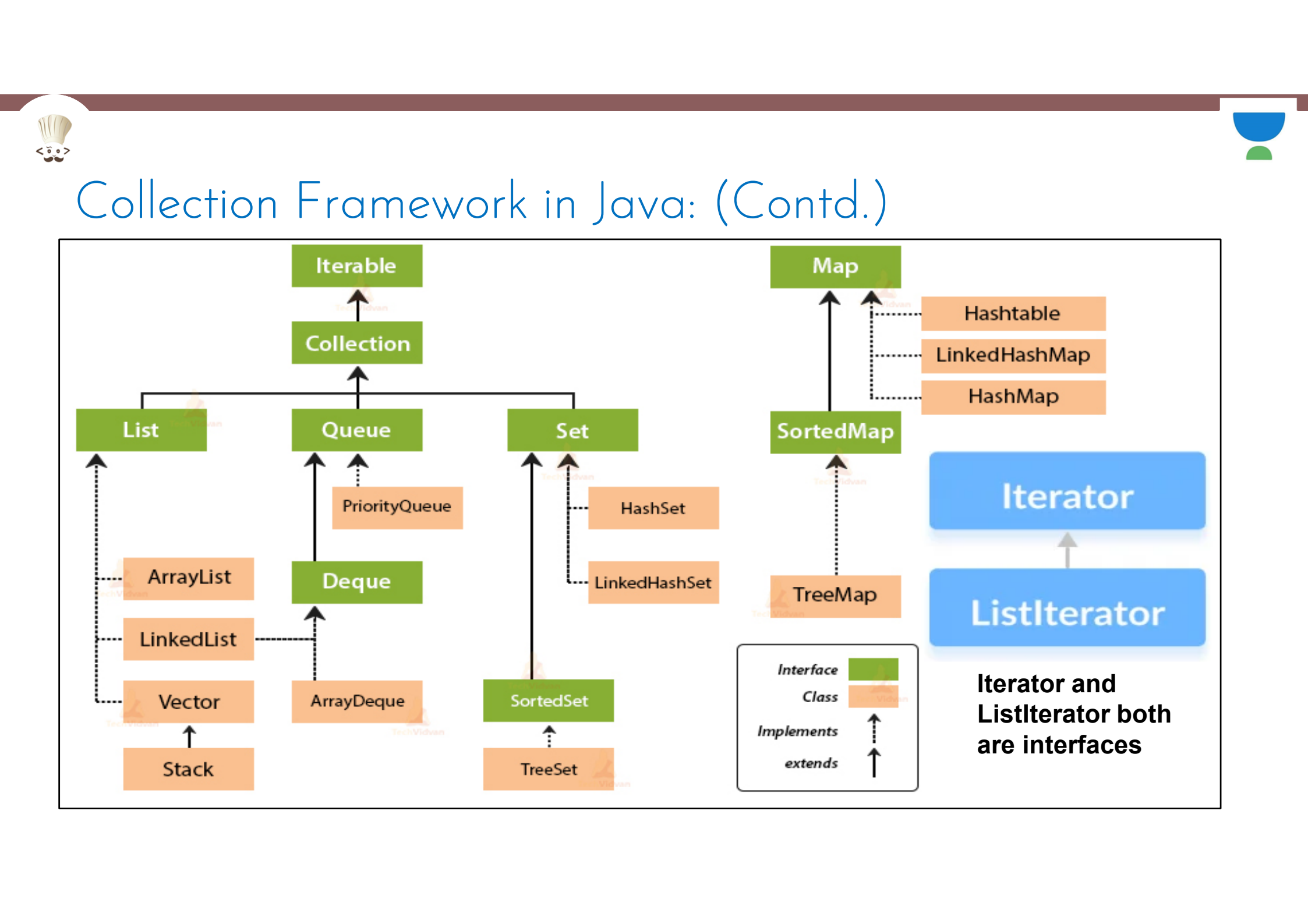
Collections in Java is a readymade architecture with an option to represent classes and objects.

On the other hand, Collection framework in Java represents a better and unified method to store objects and classes.

It has an algorithm, interfaces and implementations.

Collection Framework in Java: (Contd.)





Collection Framework in Java: (Contd.)

Iterable Interface in Java:

The Iterable interface is the root interface for all the collection classes. The Collection interface extends the Iterable interface and therefore all the subclasses of Collection interface also implement the Iterable interface.

It contains only one abstract method. i.e.,

Iterator<T> iterator()

It returns the iterator over the elements of type T.

Collection Interface in Java:

- The Collection interface is the interface which is implemented by all the classes in the collection framework. It declares the methods that every collection will have. In other words, we can say that the Collection interface builds the foundation on which the collection framework depends.
- Some of the methods of Collection interface are Boolean add (Object obj), Boolean addAll (Collection c), void clear(), etc. which are implemented by all the subclasses of Collection interface.

Different Collection Interface Methods:

public boolean add(Object element): To insert an element

public boolean addAll(Collection c): Insert a specified collection

public boolean remove(Object element): To delete

public boolean removeAll(Collection c): To delete all elements of invoking collection

public boolean retainAll(Collection c): Delete all elements of the specified collection

Different Collection Interface Methods: (Contd.)

public int size(): return total number of elements

public void clear(): delete total number of elements

public boolean contains(Object element): Search and element

public boolean containsAll(Collection c): Search a specific collection

public Iterator iterator(): To return an iterator

Different Collection Interface Methods: (Contd.)

public Object[ ] toArray(): To convert a collection to an array

public boolean isEmpty(): The isEmpty() method of Java Collection Interface returns the boolean value 'true' if this collection contains no elements.

public boolean equals(Object element): It returns true if the argument is not null and is a Boolean object that represents the same Boolean value as this object, else it returns false.

public int hashCode(): The hashCode() method returns the hash code value for this collection.

Iterator Interface Methods in Java:

public boolean hasNext() – True if it has more value.

public Object next() – Returns an element and shifts the cursor to the next element.

public void remove() – Removes the last element that iterator returned.

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.

List Interface: (Contd.)

To instantiate the List interface, we must use :

List <data-type> list1= new ArrayList();

List <data-type> list2 = new LinkedList();

List <data-type> list3 = new Vector();

List <data-type> list4 = new Stack();

There are various methods in List interface that can be used to insert, delete, and access the elements from the list.

List Interface: 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 and is non-synchronized. The elements stored in the ArrayList class can be randomly accessed.

List Interface: ArrayList (Sample Code)

```
package com.practice;
import java.util.*;
public class ArrayListDemo {
    public static void main(String args[]){
        // Creating ArrayList
        ArrayList<String> list = new ArrayList<String>();
        list.add("Monday"); //Adding object in ArrayList
        list.add("Tuesday");
        list.add("Wednesday");
        list.add("Friday");
        // Traversing list through Iterator
        Iterator itr = list.iterator();
        while(itr.hasNext()){
            System.out.println(itr.next());
        }
    }
}
```

Output:

Monday
Tuesday
Wednesday
Friday

List Interface: LinkedList

LinkedList implements the Collection interface. It uses a doubly linked list internally to store the elements. It can store the duplicate elements. It maintains the insertion order and is not synchronized. In LinkedList, the manipulation is fast because no shifting is required.

List Interface: LinkedList (Sample Code)

```
package com.practice;
import java.util.*;
public class LinkedListDemo {
    public static void main(String[] args) {
        LinkedList<String> cars = new LinkedList<String>();
        cars.add("Volvo");
        cars.add("BMW");
        cars.add("Ford");
        cars.add("Mazda");
        System.out.println(cars);
    }
}
```

Output:

Output:
[Volvo, BMW, Ford, Mazda]

List Interface: Vector

Vector uses a dynamic array to store the data elements. It is similar to ArrayList. However, It is synchronized and contains many methods that are not the part of Collection framework.

List Interface: Vector (Sample Code)

```
package com.practice;
import java.util.*;
public class VectorDemo {
    public static void main(String args[]){
        Vector<String> v = new Vector<String>();
        v.add("One");
        v.add("Two");
        v.add("Three");
        v.add("Four");
        Iterator<String> itr = v.iterator();
        while(itr.hasNext()){
            System.out.println(itr.next());
        }
    }
}
```

Output:

One
Two
Three
Four

List Interface: 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 pop(), boolean peek(), boolean push(object o), which defines its properties.

List Interface: Stack (Sample Code)

```
package com.practice;
import java.util.*;
public class StackDemo {
    public static void main(String args[]){
        Stack<String> stack = new Stack<String>();
        stack.push("One");
        stack.push("Two");
        stack.push("Three");
        stack.push("Four");
        stack.push("Five");
        stack.pop();
        Iterator<String> itr=stack.iterator();
        while(itr.hasNext()){
            System.out.println(itr.next());
        }
    }
}
```

Output:

One
Two
Three
Four

Queue Interface:

Queue interface maintains the first-in-first-out order. It can be defined as an ordered list that is used to hold the elements which are about to be processed. There are various classes like PriorityQueue, Deque, and ArrayDeque which implements the Queue interface.

Queue interface can be instantiated as:

```
Queue<String> q1 = new PriorityQueue();
Queue<String> q2 = new ArrayDeque();
```

There are various classes that implement the Queue interface e.g. PriorityQueue, ArrayQueue. And one interface Deque.

Queue Interface: PriorityQueue

The PriorityQueue class implements the Queue interface. It holds the elements or objects which are to be processed by their priorities. PriorityQueue doesn't allow null values to be stored in the queue.

Queue Interface: PriorityQueue (Sample Code)

```
package com.practice;
import java.util.*;
public class PriorityQueueDemo {
    public static void main(String args[]){
        PriorityQueue<String> queue=new PriorityQueue<String>();
        queue.add("Amit Sharma");
        queue.add("Vijay Raj");
        queue.add("JaiShankar");
        queue.add("Raj");
        System.out.println("head:"+queue.element());
        System.out.println("head:"+queue.peek());
        System.out.println("iterating the queue elements:");
        Iterator itr=queue.iterator();
        while(itr.hasNext()){
            System.out.println(itr.next());
        }
        queue.remove();
        queue.poll();
        System.out.println("after removing two elements:");
        Iterator<String> itr2=queue.iterator();
        while(itr2.hasNext()){
            System.out.println(itr2.next());
        }
    }
}
```

Output:

```
head:Amit Sharma
head:Amit Sharma
iterating the queue elements:
Amit Sharma
Raj
JaiShankar
Vijay Raj
after removing two elements:
Raj
Vijay Raj
```

Deque Interface:

Deque interface extends the Queue interface. In Deque, we can remove and add the elements from both the side. Deque stands for a double-ended queue which enables us to perform the operations at both the ends.

Deque can be instantiated as:

Deque d = new ArrayDeque();

Deque Interface: ArrayDeque

ArrayDeque class implements the Deque interface. It facilitates us to use the Deque. Unlike queue, we can add or delete the elements from both the ends.

ArrayDeque is faster than ArrayList and Stack and has no capacity restrictions.

Deque Interface: ArrayQueue (Sample Code)

```
package com.practice;
import java.util.*;
public class ArrayQueueDemo {
    public static void main(String[] args) {
        //Creating Deque and adding elements
        Deque<String> deque = new ArrayDeque<String>();
        deque.add("Gautam");
        deque.add("Karan");
        deque.add("Ajay");
        //Traversing elements
        for (String str : deque) {
            System.out.println(str);
        }
    }
}
```

Output:

Gautam
Karan
Ajay

Set Interface:

Set Interface in Java is present in java.util package. It extends the Collection interface. It represents the unordered set of elements which doesn't allow us to store the duplicate items. We can store at most one null value in Set. Set is implemented by HashSet, LinkedHashSet, and TreeSet.

Set can be instantiated as:

```
Set<data-type> s1 = new HashSet<data-type>();
Set<data-type> s2 = new LinkedHashSet<data-type>();
Set<data-type> s3 = new TreeSet<data-type>();
```

Set Interface: HashSet

HashSet class implements Set Interface. It represents the collection that uses a hash table for storage. Hashing is used to store the elements in the HashSet. It contains unique items.

Set Interface: HashSet (Sample Code)

```
package com.practice;
import java.util.*;
public class HashSetDemo {
    public static void main(String args[]){
        //Creating HashSet and adding elements
        HashSet<String> set=  new HashSet<String>();
        set.add("Ravi");
        set.add("Vijay");
        set.add("Ravi");
        set.add("Ajay");
        //Traversing elements
        Iterator<String> itr = set.iterator();
        while(itr.hasNext()){
            System.out.println(itr.next());
        }
    }
}
```

Output:

Vijay
Ravi
Ajay

Set Interface: LinkedHashSet

LinkedHashSet class represents the LinkedList implementation of Set Interface. It extends the HashSet class and implements Set interface. Like HashSet, It also contains unique elements. It maintains the insertion order and permits null elements.

Set Interface: LinkedHashSet (Sample Code)

```
package com.practice;
import java.util.*;
public class LinkedHashSetDemo {
    public static void main(String args[]){
        LinkedHashSet<String> set = new LinkedHashSet<String>();
        set.add("Ravi");
        set.add("Vijay");
        set.add("Ravi");
        set.add("Ajay");
        Iterator<String> itr = set.iterator();
        while(itr.hasNext()){
            System.out.println(itr.next());
        }
    }
}
```

Output:

Ravi
Vijay
Ajay

SortedSet Interface:

SortedSet is the alternate of Set interface that provides a total ordering on its elements. The elements of the SortedSet are arranged in the increasing (ascending) order. The SortedSet provides the additional methods that inhibit the natural ordering of the elements.

The SortedSet can be instantiated as:

SortedSet<data-type> set = new TreeSet();

SortedSet Interface: TreeSet

Java TreeSet class implements the Set interface that uses a tree for storage. Like HashSet, TreeSet also contains unique elements. However, the access and retrieval time of TreeSet is quite fast. The elements in TreeSet stored in ascending order.

SortedSet Interface: TreeSet (Sample Code)

```
package com.practice;
import java.util.*;
public class TreeSetDemo {
    public static void main(String args[]){
        //Creating and adding elements
        TreeSet<String> set = new TreeSet<String>();
        set.add("Ravi");
        set.add("Vijay");
        set.add("Ravi");
        set.add("Ajay");
        //traversing elements
        Iterator<String> itr = set.iterator();
        while(itr.hasNext()){
            System.out.println(itr.next());
        }
    }
}
```

Output:

Ajay
Ravi
Vijay

Quiz!

1. The collection is a _____

A) framework and interface

B) framework and class

C) only interface

D) only class

1. The collection is a _____

A) framework and interface

B) framework and class

C) only interface

D) only class

Explanation:
Collection is both framework and interface

2. List, Set and Queue _____ Collection.

A) extends

B) implements

C) both of the above

D) none of the above

2. List, Set and Queue _____ Collection.

A) extends

B) implements

C) both of the above

D) none of the above

Explanation:
List, Set and Queue are all interfaces and they extends Collection interface. (interface extends interface)

3. Which collection class allows you to grow or shrink its size and provides indexed access to its elements, but whose methods are not synchronized?

A) java.util.HashSet
B) java.util.LinkedHashSet
C) java.util.List
D) java.util.ArrayList

3. Which collection class allows you to grow or shrink its size and provides indexed access to its elements, but whose methods are not synchronized?

A) java.util.HashSet
B) java.util.LinkedHashSet
C) java.util.List
D) java.util.ArrayList ✓

Explanation:
All of the collection classes allow you to grow or shrink the size of your collection. ArrayList provides an index to its elements. The newer collection classes tend not to have synchronized methods. Vector is an older implementation of ArrayList functionality and has synchronized methods; it is slower than ArrayList.

4. You need to store elements in a collection that guarantees that no duplicates are stored and all elements can be accessed in natural order. Which interface provides that capability?

A. java.util.Map
B. java.util.Set
C. java.util.List
D. java.util.Collection

4. You need to store elements in a collection that guarantees that no duplicates are stored and all elements can be accessed in natural order. Which interface provides that capability?

A. java.util.Map

B. java.util.Set

C. java.util.List

D. java.util.Collection

Explanation:

Option B is correct. A set is a collection that contains no duplicate elements. The iterator returns the elements in no particular order (unless this set is an instance of some class that provides a guarantee). A map cannot contain duplicate keys but it may contain duplicate values. List and Collection allow duplicate elements.

Option C is wrong. A list is an ordered collection (also known as a sequence). The user of this interface has precise control over where in the list each element is inserted. The user can access elements by their integer index (position in the list), and search for elements in the list. Unlike sets, lists typically allow duplicate elements.

Option D is wrong. A collection is also known as a sequence. The user of this interface has precise control over where in the list each element is inserted. The user can access elements by their integer index (position in the list), and search for elements in the list. Unlike sets, lists typically allow duplicate elements.

Option A is wrong. A map is an object that maps keys to values. A map cannot contain duplicate keys; each key can map to at most one value. The Map interface provides three collection views, which allow a map's contents to be viewed as a set of keys, collection of values, or set of key-value mappings. The order of a map is defined as the order in which the iterators on the map's collection views return their elements. Some map implementations, like the TreeMap class, make specific guarantees as to their order (ascending key order); others, like the HashMap class, do not (does not guarantee that the order will remain constant over time).

5. Which collection class allows you to access its elements by associating a key with an element's value, and provides synchronization?

A) java.util.SortedMap
B) java.util.TreeMap
C) java.util.TreeSet
D) java.util.Hashtable

5. Which collection class allows you to access its elements by associating a key with an element's value, and provides synchronization?

A) java.util.SortedMap
B) java.util.TreeMap
C) java.util.TreeSet
D) java.util.Hashtable

Explanation:
Hashtable is the only class listed that provides synchronized methods. If you need synchronization great; otherwise, use HashMap, it's faster.

6. Which is valid declaration of a float?

A) float f = 1F;
B) float f = 1.0;
C) float f = "1";
D) float f = 1.0d;

6. Which is valid declaration of a float?

A) float f = 1F;
B) float f = 1.0;
C) float f = "1";
D) float f = 1.0d;

Explanation:
Option A is valid declaration of float.

Option B is incorrect because any literal number with a decimal point u declare the computer will implicitly cast to double unless you include "F or f"

Option C is incorrect because it is a String.

Option D is incorrect because "d" tells the computer it is a double so therefore you are trying to put a double value into a float variable i.e there might be a loss of precision.

7.What line of code should replace the missing statement to make this program compile?

```
public class foo {
   public static void main(String[]args)throws Exception {
      java.io.PrintWriter out = new java.io.PrintWriter();
      new java.io.OutputStreamWriter(System.out,true);
      out.println("Hello");
   }
}
```

A) No statement required.
B) import java.io.*;
C) include java.io.*;
D) import java.io.PrintWriter;

7.What line of code should replace the missing statement to make this program compile?

```
public class foo {
    public static void main(String[]args)throws Exception {
        java.io.PrintWriter out = new java.io.PrintWriter();
        new java.io.OutputStreamWriter(System.out,true);
        out.println("Hello");
    }
}
```

✓A) No statement required.
B) import java.io.*;
C) include java.io.*;
D) import java.io.PrintWriter;

Explanation:
The usual method for using/importing the java packages/classes is by using an import statement at the top of your code. However it is possible to explicitly import the specific class that you want to use as you use it which is shown in the code above. The disadvantage of this however is that every time you create a new object you will have to use the class path in the case "java.io" then the class name in the long run leading to a lot more typing.

8.Suppose that you would like to create an instance of a new Map that has an iteration order that is the same as the iteration order of an existing instance of a Map. Which concrete implementation of the Map interface should be used for the new instance?

A) TreeMap
B) HashMap
C) LinkedHashMap
D) The answer depends on the implementation of the existing instance.

8.Suppose that you would like to create an instance of a new Map that has an iteration order that is the same as the iteration order of an existing instance of a Map. Which concrete implementation of the Map interface should be used for the new instance?

A) TreeMap
B) HashMap
C) LinkedHashMap ✓
D) The answer depends on the implementation of the existing instance.

Explanation:
The iteration order of a Collection is the order in which an iterator moves through the elements of the Collection. The iteration order of a LinkedHashMap is determined by the order in which elements are inserted.

When a new LinkedHashMap is created by passing a reference to an existing Collection to the constructor of a LinkedHashMap the Collection.addAll method will ultimately be invoked.

The addAll method uses an iterator to the existing Collection to iterate through the elements of the existing Collection and add each to the instance of the new LinkedHashMap.

Since the iteration order of the LinkedHashMap is determined by the order of insertion, the iteration order of the new LinkedHashMap must be the same as the interation order of the old Collection.

9. Suppose that you would like to create an instance of a new Map that has an iteration order that is the same as the iteration order of an existing instance of a Map. Which concrete implementation of the Map interface should be used for the new instance?

A) TreeMap
B) HashMap
C) LinkedHashMap
D) The answer depends on the implementation of the existing instance.

9. Suppose that you would like to create an instance of a new Map that has an iteration order that is the same as the iteration order of an existing instance of a Map. Which concrete implementation of the Map interface should be used for the new instance?

A) TreeMap
B) HashMap
C) LinkedHashMap ✓
D) The answer depends on the implementation of the existing instance.

Explanation:
The iteration order of a Collection is the order in which an iterator moves through the elements of the Collection. The iteration order of a LinkedHashMap is determined by the order in which elements are inserted.

When a new LinkedHashMap is created by passing a reference to an existing Collection to the constructor of a LinkedHashMap the Collection.addAll method will ultimately be invoked.

The addAll method uses an iterator to the existing Collection to iterate through the elements of the existing Collection and add each to the instance of the new LinkedHashMap.

Since the iteration order of the LinkedHashMap is determined by the order of insertion, the iteration order of the new LinkedHashMap must be the same as the interation order of the old Collection.

10. What is the output of this program?

```
import java.util.*;
class Array {
      public static void main(String args[]) {
         int array[] = new int [5];
         for (int i = 5; i > 0; i--)
            array[5-i] = i;
         Arrays.fill(array, 1, 4, 8);
         for (int i = 0; i < 5 ; i++)
            System.out.print(array[i]);
      }
   }
```

A) 12885

B) 12845

C) 58881

D) 54881

10. What is the output of this program?

```
import java.util.*;
class Array {
      public static void main(String args[]) {
         int array[] = new int [5];
         for (int i = 5; i > 0; i--)
            array[5-i] = i;
         Arrays.fill(array, 1, 4, 8);
         for (int i = 0; i < 5 ; i++)
            System.out.print(array[i]);
      }
   }
```

A) 12885
B) 12845
C) 58881 ✓
D) 54881

Explanation: array was containing 5,4,3,2,1 but when method Arrays.fill(array, 1, 4, 8) is called it fills the index location starting with 1 to 4 by value 8 hence array becomes 5,8,8,8,1.

11. What is the output of this program?

```
import java.util.*;
class vector {
    public static void main(String args[]) {
        Vector obj = new Vector(4,2);
        obj.addElement(new Integer(3));
        obj.addElement(new Integer(2));
        obj.addElement(new Integer(5));
        obj.removeAll(obj);
        System.out.println(obj.isEmpty());
    }
}
```

A) 0
B) 1
C) true
D) false

11. What is the output of this program?

```
import java.util.*;
class vector {
    public static void main(String args[]) {
        Vector obj = new Vector(4,2);
        obj.addElement(new Integer(3));
        obj.addElement(new Integer(2));
        obj.addElement(new Integer(5));
        obj.removeAll(obj);
        System.out.println(obj.isEmpty());
    }
}
```

A) 0
B) 1
C) true
D) false

Explanation: firstly elements 3, 2, 5 are entered in the vector obj, but when obj.removeAll(obj); is executed all the elements are deleted and vector is empty, hence obj.isEmpty() returns true.

12. What will be the output of the following Java code?

```
import java.util.*;
   class Array {
      public static void main(String args[]) {
         int array[] = new int [5];
         for (int i = 5; i > 0; i--)
            array[5-i] = i;
         Arrays.fill(array, 1, 4, 8);
         for (int i = 0; i < 5 ; i++)
            System.out.print(array[i]);
      }
   }
```

A) 12885

B) 12845

C) 58881

D) 54881

12. What will be the output of the following Java code?

```
import java.util.*;
   class Array {
      public static void main(String args[]) {
         int array[] = new int [5];
         for (int i = 5; i > 0; i--)
            array[5-i] = i;
         Arrays.fill(array, 1, 4, 8);
         for (int i = 0; i < 5 ; i++)
            System.out.print(array[i]);
      }
   }
```

A) 12885
B) 12845
C) 58881 ✓
D) 54881

Explanation: array was containing 5,4,3,2,1 but when method Arrays.fill(array, 1, 4, 8) is called it fills the index location starting with 1 to 4 by value 8 hence array becomes 5,8,8,8,1.

13. Deque and Queue inherit from _____

A Collection

B AbstractList

C AbstractCollection

D List

13. Deque and Queue inherit from _____

A Collection

B AbstractList

C AbstractCollection

D List

Explanation: Its illegal to call max() only with comparator, we need to give the collection to be serched into.

14. What is the output of the following code?

```
import java.util.*;
public class Main {
  public static void main(String args[]) {
     LinkedList<Integer> lang = new LinkedList<Integer>();
     lang.add(8);
     lang.add(2);
     lang.add(1);
     lang.add(6);
     Iterator it = lang.iterator();
     Collections.reverse(lang);
     Collections.sort(lang);
     while(it.hasNext())
        System.out.print(it.next() + " ");
  }
}
```

A) 6 1 2 8
B) 1 2 6 8
C) 8 6 2 1
D) 8 2 1 6

14. What is the output of the following code?

```
import java.util.*;
public class Main {
  public static void main(String args[]) {
     LinkedList<Integer> lang = new LinkedList<Integer>();
     lang.add(8);
     lang.add(2);
     lang.add(1);
     lang.add(6);
     Iterator it = lang.iterator();
     Collections.reverse(lang);
     Collections.sort(lang);
     while(it.hasNext())
        System.out.print(it.next() + " ");
  }
}
```

A) 6 1 2 8

B) 1 2 6 8 ✓

C) 8 6 2 1

D) 8 2 1 6

15. Deque and Queue inherit from _____

A) Collection

B) AbstractList

C) AbstractCollection

D) List

15. Deque and Queue inherit from _____

A) Collection

B) AbstractList

C) AbstractCollection

D) List

Explanation: Deque and Queue interfaces inherit from the Collection interface.