JAVA @11

Collection Framework

Objective

After completing this session you will be able to understand,

- Define collection

- Describe usage of collections

- Describe the benefits of collections

- Understand the core collection interfaces

- Understand the implemented classes.

Collections

A "Collection" is a container that groups multiple elements into single unit.

Collections in java is a framework that provides an architecture to store and manipulate the group of objects.

Collections Framework

- A collections framework is a unified architecture for representing and manipulating varieties of collections.

- Some collections **allow duplicate** *elements* and others **do not**.

- Can be used only to hold object type data (non – primitive type).

- Some are **ordered** and others are **unordered**.

- The collection framework is available in the **java.util** package.

Collection framework contains,

- A set of **Interfaces**.

- Concrete **class** implementations for the interfaces.

- The classes in turn has standard **API's** for processing collections.

Benefits of Collection framework

Reduce programming Effort

Example: This provides methods for sorting which need not be developed, from scratch thus reducing the development effort.

Easy to maintain

The API's are easy to use and maintain.

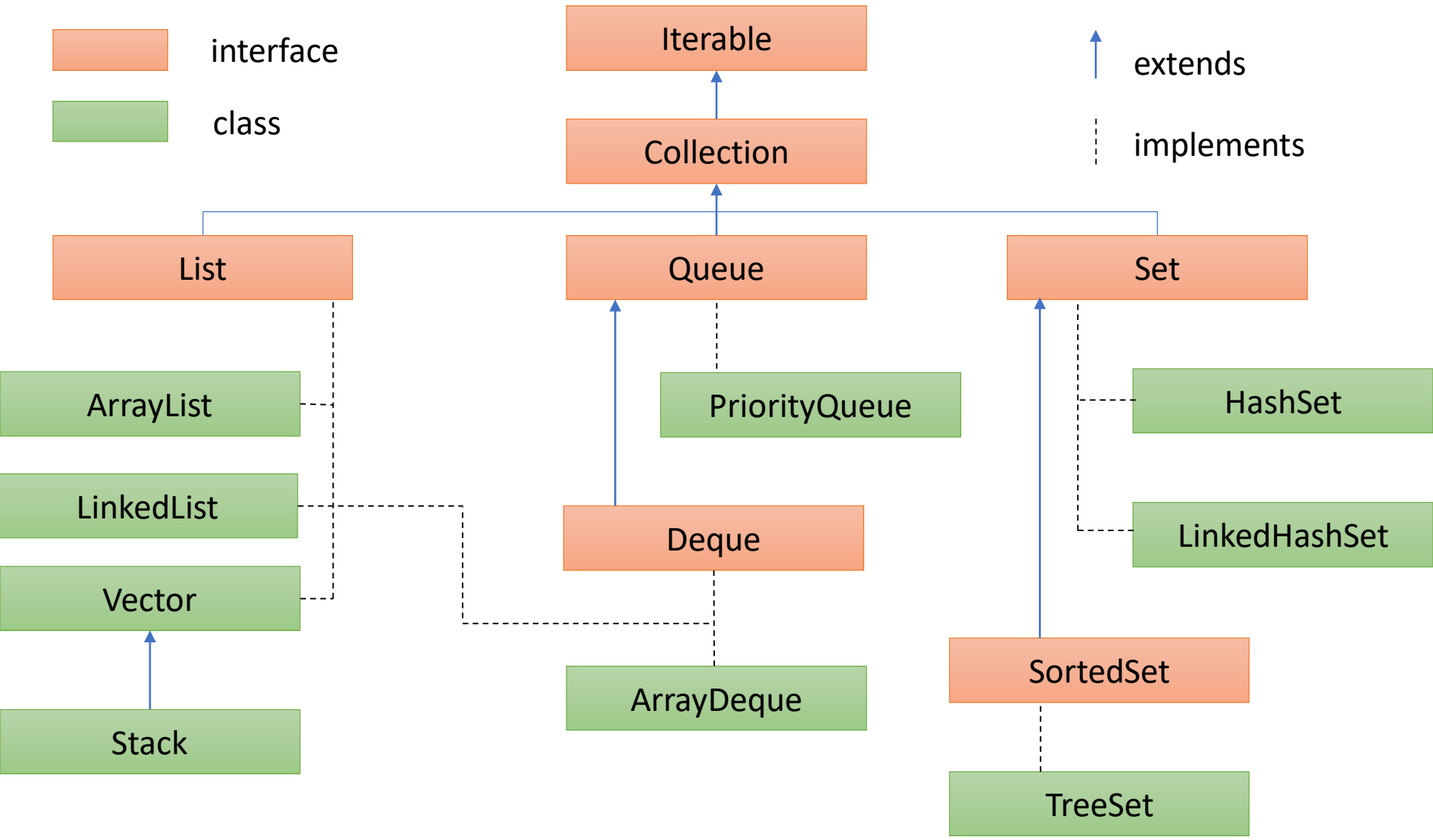**Collection Framework Benefits**

High performance APIs

Example: The API's provided are proven tested ones with high performance.
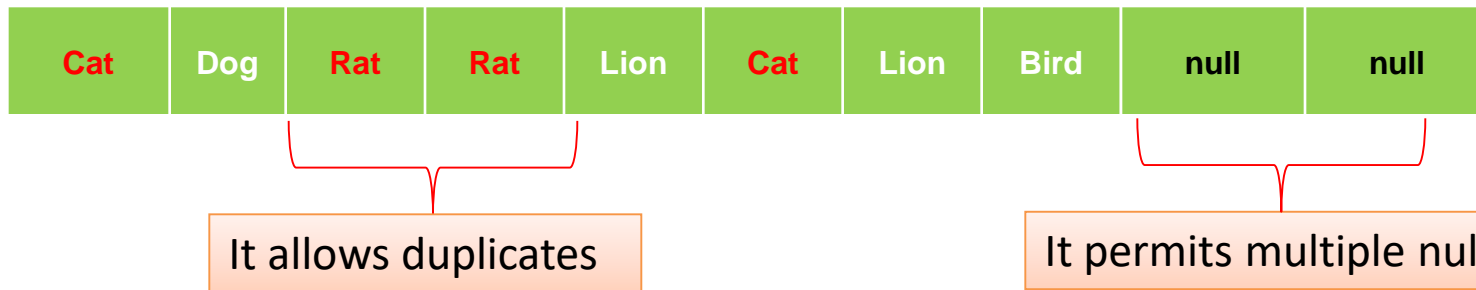
Easy to use

Resizable and grows

Example: The collection sizes dynamically changes that is it gets shrunk or increased based on the elements stored.
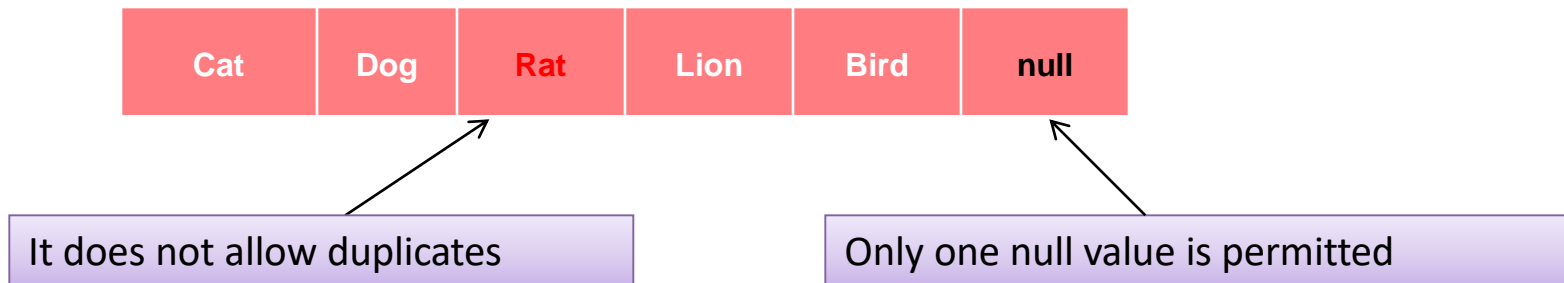
Collection Framework Components

interface

class

Iterable

↑ extends

Collection

⁞ implements

List

Queue

Set

ArrayList

PriorityQueue

HashSet

LinkedList

Vector

Deque

LinkedHashSet

Stack

ArrayDeque

SortedSet

TreeSet

Difference between List & Set

List :

Cat	Dog	Rat	Rat	Lion	Cat	Lion	Bird	null	null

It allows duplicates

It permits multiple null values.

Set:

Cat	Dog	Rat	Lion	Bird	null

It does not allow duplicates

Only one null value is permitted

Ordered vs Sorted Collection

Consider you have a collection fruits names.

1. Grapes

2. Orange

3. Banana

4. Apple

5. Pineapple

A SortedMap is a Map that maintains its entries in ascending order, sorted based on the key.

Problem 1 : You need to insert the above list of fruits in the same order given above . What type of collection can be used ?

Solution : Any ordered collection can be used. Example: *List*.

Problem 2: Suppose you want to automatically sort the fruit names and store it in a collection ? What type of collection can be used ?

Solution : A sorted collection like *SortedSet* can be used which automatically sorts and rearranges the value each time a new fruit is added to the list.

Collection Interface

The root interface in the collection hierarchy.

- JDK does not provide any direct implementations of this interface, it provides implementations of more specific sub-interfaces like *Set* and *List* etc.

- Contains methods which needs to implemented directly or indirectly by the sub classes.

Collection Methods

Methods	Description
public boolean **add**(Object element)	is used to insert an element in this collection.
public boolean **addAll**(Collection c)	is used to insert the specified collection elements in the invoking collection.
public boolean **remove**(Object element)	is used to delete an element from this collection.
public boolean **removeAll**(Collection c)	is used to delete all the elements of specified collection from the invoking collection.
public boolean **retainAll**(Collection c)	is used to delete all the elements of invoking collection except the specified collection.
public int **size**()	return the total number of elements in the collection.
public void **clear**()	removes the total no of element from the collection.

Collection Methods

Methods	Description
public boolean **contains**(Object element)	is used to search an element.
public boolean **containsAll**(Collection c)	is used to search the specified collection in this collection.
public Iterator **iterator**()	returns an iterator.
public Object[] **toArray**()	converts collection into array.
public boolean **isEmpty**()	checks if collection is empty.
public boolean **equals**(Object element)	matches two collection.
public int **hashCode**()	returns the hashcode number for collection.

List Interface

Used for storing the elements in a ordered way.

- Supports duplicate entries.

- Supports index based additions and retrieval of items.

public interface List<E> **extends** Collection<E>

List implementations:

1. Vector

2. ArrayList

3. LinkedList

List Methods

Methods	Description
void **add**(int index,Object element)	It is used to insert element into the invoking list at the index passed in the index.
boolean **addAll**(int index,Collection c)	It is used to insert all elements of c into the invoking list at the index passed in the index.
object **get**(int index)	It is used to return the object stored at the specified index within the invoking collection.
object **set**(int index,Object element)	It is used to assign element to the location specified by index within the invoking list.
object **remove**(int index)	It is used to remove the element at position index from the invoking list and return the deleted element.

ArrayList

- Implements *List* interface

- *ArrayList* can grow and shrink in size dynamically based on the elements stored hence can be considered as a variable array.

- Values are stored internally as an array hence random insert and retrieval of elements are allowed.

public class ArrayList<E> **extends** AbstractList<E> **implements** List<E>, RandomAccess, Cloneable, Serializable

- Can be traversed using a *foreach* loop, *iterators*, or *indexes*.

- Initially gets created with an initial capacity , when this get's filled the list automatically grows.

- Can hold only object type data – Cannot hold primitive type values.

- Supports duplicate entries.

How to create an ArrayList ?

Constructors of Java ArrayList

Constructor	Description
ArrayList()	It is used to build an empty array list.
ArrayList(Collection c)	It is used to build an array list that is initialized with the elements of the collection c.
ArrayList(int capacity)	It is used to build an array list that has the specified initial capacity.

Non-generic:

ArrayList list = new **ArrayList**();

Generic:

ArrayList<String> list = new **ArrayList**<String>();

How to add elements in a List?

Values can be added to an **ArrayList** using any of the add methods.

Syntax :

listName.add(element);

Example :

Create an arraylist

List myList=new **ArrayList**()

⓪ ① ② ③ ④ ⑤ ⑥

| Banana | Lemon | Orange | | | | |

myList.add("Apple");

myList.add("Orange");

myList.add(1,"Lemon");

myList.set(0,"Banana");

Adds the element lemon at position 1 moving orange to position 2.

Adds elements one by one to the list.

When set is used the item in the particular index will be replaced by the new item.

Example – using ArrayList

Objective: In this lend a hand we are going to learn about creating and adding elements into the array list.

Problem Statement 1: Create a method that accepts the names of five fruits and loads them to an array list and returns the list.

Problem Statement 2: Create an method which can return an array list holding values 1-10.

Problem Statement 3: Create a method needs to return an array list holding value 1-15. The method should make use of the already created array list containing values up to 10 and then add values from 11-15 using a for loop.

Set Interface

- The **Set** interface is a collection that cannot contain duplicate elements.

- It permits a single element to be null.

- **Set** interface contains methods inherited from **collection** interface.

- **SortedSet** interface is a set that maintains its elements in ascending order.

Set Implementations:

1. HashSet

2. LinkedHashSet

3. TreeSet

HashSet

Java HashSet class is used to create a collection that uses a hash table for storage.

It inherits the AbstractSet class and implements Set interface.

The important points about Java HashSet class are:

- HashSet stores the elements by using a mechanism called **hashing.**

- *HashSet* contains unique elements only.

- Cannot predict the order in which the items get stored in the Set

- Not thread safe.

- Permits null value(only one).

syntax:

public class HashSet<E> **extends** AbstractSet<E> **implements** Set<E>, Cloneable, Serializable

HashSet - Methods

Method	Description
void **clear**()	It is used to remove all of the elements from this set.
boolean **contains**(Object o)	It is used to return true if this set contains the specified element.
boolean **add**(Object o)	It is used to adds the specified element to this set if it is not already present.
boolean **isEmpty**()	It is used to return true if this set contains no elements.
boolean **remove**(Object o)	It is used to remove the specified element from this set if it is present.
Object **clone**()	It is used to return a shallow copy of this HashSet instance: the elements themselves are not cloned.
Iterator **iterator**()	It is used to return an iterator over the elements in this set.
int **size**()	It is used to return the number of elements in this set.

How to add elements to HashSet?

Elements can be added one by one using a *add()* method or an entire collection can be added using the *addAll()* method.

Example:

mySet.add("Grapes");

Adds two elements into the set.

mySet.add("Banana");

mySet.add("Grapes")

Returns false since India already exists in the set.

mySet.addAll(mySet1);

Adds all the items of Set mySet1 to mySet

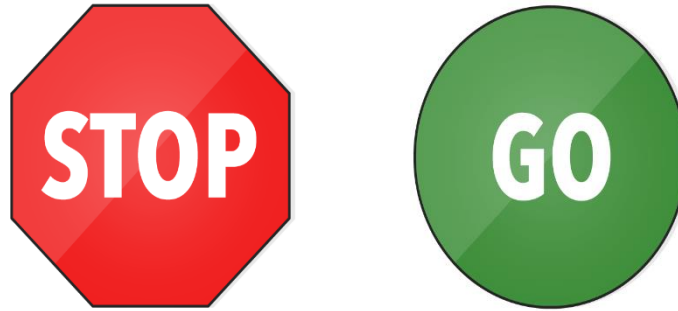⚠ Note : If an entire set is added to another set , the duplicate elements will not get added.

Objective: Let us learn how to create a *HashSet* and add elements into it.

Problem Statement 1: Create a method that accepts the names of five books details and loads them to an HashSet and returns the Set.

Problem Statement 2 : Create an method which can return a set holding values 1-10.

Problem Statement 3 : Create a method needs to return a set holding value 1-15. The method should make use of the already created set containing values up to 10 and then add values from 11-15 using a for loop.

Time To Reflect

Trainees to reflect the following topics before proceeding.
- What method can be used to merge to two array lists ?
- What happens when a duplicate entry is tried to be added to a set ?
- What is the difference between add and set method in array list?
- How to add an element to a particular position in an array list?

Generics

Let us take a real time example
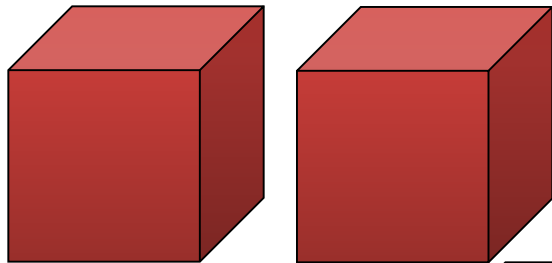
The buyer does not know what these red shape boxes contain. Hence the buyer needs to open these boxes every time to find what is inside it

Roses

Jewels

Chocolate
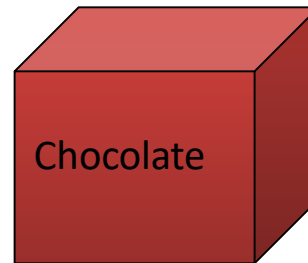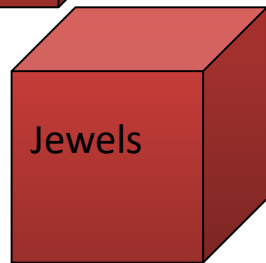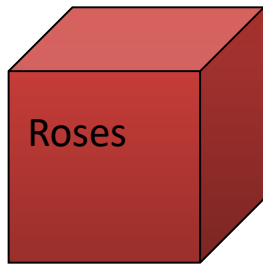
Here, the buyer is aware of what is inside the boxes ,since the box is labeled. This makes his job easier since he knows what he can put in and what he can take out from the box

What is Generics?

- Generics is a feature allows the programmer to specify the data type to be stored in a collection or a class when developing the program.

- Compile throws error if the data stored is different from the data type specified in the generic.

Advantage of using Generics using Collection

1. **Avoid unnecessary type casting** : Using generics avoid the need of typecasting while retrieving the elements from the collection.

   Consider an ArrayList named countries ,elements can be retrieved from the List as shown below

   Without Generics

   With Generics

```java
public void printList() {

    List countries = new ArrayList();
    countries.add("India");
    countries.add("Australia");
    String[] countryArray = new String[countries.size()];
    for (int i = 0; i < countries.size(); i++) {
        countryArray[i] = (String) countries.get(i);
    }
}
```

```java
public void printList() {

    List<String> countries = new ArrayList<String>();
    countries.add("India");
    countries.add("Australia");
    String[] countryArray = new String[countries.size()];
    for (int i = 0; i < countries.size(); i++) {
        countryArray[i] = countries.get(i);
    }
}
```

Type Casted to String

No need of type casting

Advantage of using Generics using Collection

2. **Minimize CastException during run time:** For example assume a method returns a List of integers. The client is unaware about the type of data held by the collection and he may type cast it to some incompatible type and end up in *CastException* during runtime. In case of any invalid casting compiler will throw an error..

3. **Easy Maintenance & Readability :** Looking at the code the developer can easily know the data type stored in the collection and thus helps him to process the collection easily.

How to Iterate through collection elements?

Let us now see how to iterate through the collection and read elements one by one.

The following are the ways to iterate through a collection

For loop	• **Reads elements by specifying the index using the get() method** • **Can be used only with List.**
For-each loop	• **Iterate over a list and gets the elements one by one until the complete list is covered** • **Can be applied with both List and Set.**
Iterator	• *Iterator* **provides methods to iterate through a collection.** • **Can be applied with both List and Set.**
ListIterator	• **An iterator which support both forward and back ward iteration.** • **Can be applied only with List.**

Iteration using "for" and "for each" loop

```java
public class ForIteration{
    public static void main(String args[]){
        LinkedList<String> al=new LinkedList<String>();
        al.add("Ravi");
        al.add("Vijay");
        al.add("Hari");
        al.add("Ajay");
        // for - iteration block
        for(int i=0;i<al.size();i++){
            System.out.println(al.get(i));
        }
        // for each - iteration block
        for(String name:al) {
            System.out.println(name);
        }
    }
}
```

Using for block to iterate the list elements

Using for each block to iterate the list elements

Iterator

What is an Iterator interface?

- This interface contains methods which is used for iterating & accessing all the elements of a collection in sequence.

- The List and Set collections can be iterated using iterators.

Iterator interface methods:

Method	Description
boolean **hasNext**()	true if there are more elements for the iterator.
Object **next**()	Returns the next object or elements.
void **remove**()	Removes the element that was returned by next from the collection. This method can be invoked only once per call to next .

How to create an Iterator?

An Iterator is created for a particular collection object by calling the **iterator**() method on the collection object.

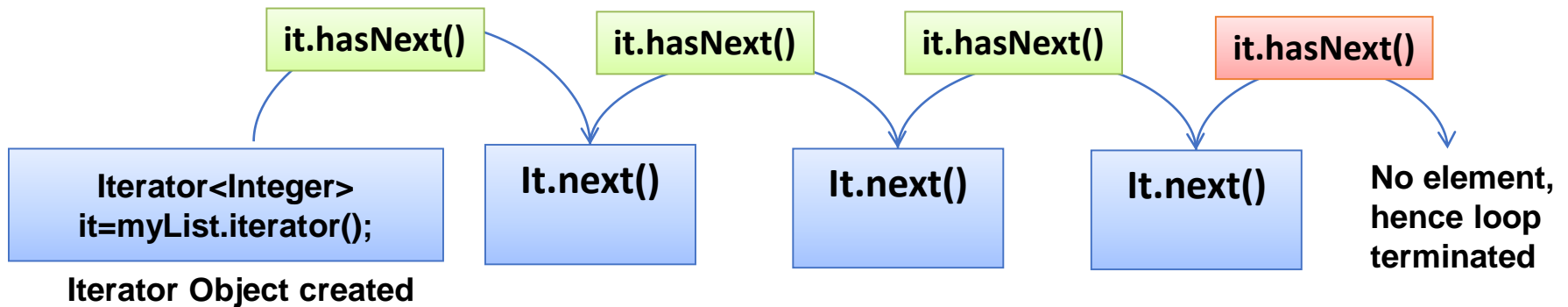Syntax :

Iterator<type> iteratorName=collection.**iterator**();

Where type is the generic type of the Iterator

Example : Creates an iterator to iterate the elements of the myList containing Integer objects.

Iterator<Integer> myListIterator = myList.**iterator**();

Iterator - Illustration

Working of Iterator

it.hasNext()   **it.hasNext()**   **it.hasNext()**   **it.hasNext()**

Iterator<Integer> it=myList.iterator();

Iterator Object created

It.next()   **It.next()**   **It.next()**

No element, hence loop terminated

The next element is retrieved everytime '***it.next()***' is executed. The ***hasnext()*** method is used for verifying the existence of element in the collection before firing the ***next().***

Example – an Iterator

```java
public class ForIteration{
    public static void main(String args[]){
        LinkedList<String> al=new LinkedList<String>();
        al.add("shri");
        al.add("Vijay");
        al.add("Hari");
        al.add("Ajay");
        Iterator<String> it = al.iterator();
        while(it.hasNext()) {
            System.out.println(it.next());
        }
    }
}
```

Create an iterator to work on the Linked List

Invoke the hasNext() method to check whether the iterator contains elements to be read.

Invoke the next() method to read each country from the Set.

Note : The Iterator is also declared with generics to avoid type casting.

ListIterator

- An list iterator can be used by programmer to traverse the list in either direction, modify the list during iteration, and obtain the iterator's current position in the list.

- A ListIterator has no current element; its cursor position always lies between the element that would be returned by a call to **previous**() and the element that would be returned by a call to **next**().

Time To Reflect
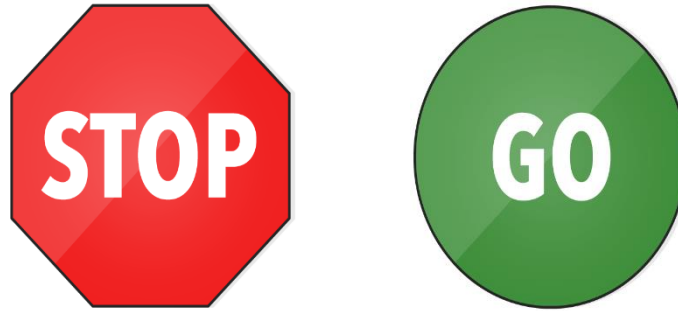
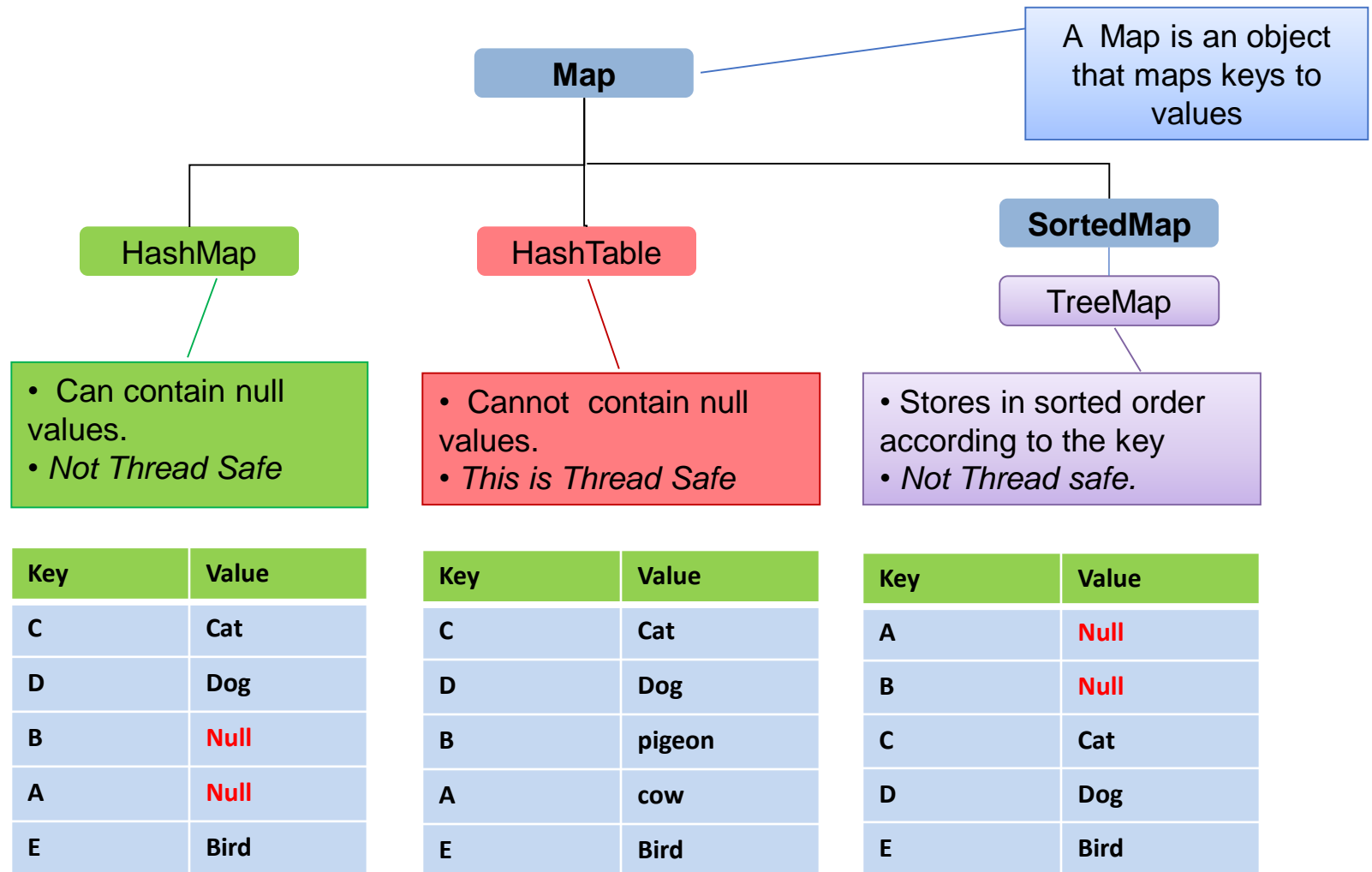**Trainees to reflect the following topics before proceeding.**
- How to make a collection accept only Integer objects?
- What are all the methods available to iterate through a Set?
- Can a ListIterator be used with a Set?
- Which iterator can be used to add elements to a list in between the iteration process?
- What may be the reason that for loop is not used to iterate through set?

Maps

- Map stores Elements as key – value pairs.
- Each key maps to one value stored in the map.
- The values can be accessed by passing the key.
- The key should be unique used to identify the value.
- The key and value can only be objects.

Key	Value
1	Grapes
2	Pineapple
3	Banana
4	Apple
5	Orange

Map Interfaces

Map

A Map is an object that maps keys to values

HashMap

HashTable

SortedMap

TreeMap

- Can contain null values.
- *Not Thread Safe*

- Cannot contain null values.
- *This is Thread Safe*

- Stores in sorted order according to the key
- *Not Thread safe.*

Key	Value
C	Cat
D	Dog
B	**Null**
A	**Null**
E	Bird

Key	Value
C	Cat
D	Dog
B	pigeon
A	cow
E	Bird

Key	Value
A	**Null**
B	**Null**
C	Cat
D	Dog
E	Bird

Difference between Collection Interfaces and Map

Collection

Cat	Dog	Bat	Lion	Bird

How Lion look up happens?

Map

Key	Value
C	Cat
D	Dog
B	Bat
L	Lion
Bi	Bird

In the case of collection we can see that the search traverses across the entire collection to get the required object. But in the case of map it is directly accessed using the key.

Methods in Map

Method	Description
void **clear**()	Removes all mappings from this map (optional operation).
boolean **containsKey**(Object key)	Returns true if this map contains a mapping for the specified key.
boolean **containsValue**(Object ob)	Returns true if this map maps one or more keys to the specified value.
Set<Map.Entry<K, V>> **entrySet**()	Returns a set view of the mappings contained in this map.
Object **get**(Objectkey)	Returns the value to which this map maps the specified key.
boolean **isEmpty**()	Returns true if this map contains no key-value mappings.
Set **keySet**()	Returns a set with all the keys contained in this map.
Object **put**(Object key, Object value)	Associates the specified value with the specified key in this map.
void **putAll**(Map t)	Copies all of the mappings from the specified map to this map

Map.Entry

- Entry is the sub interface of Map.
- It provides methods to get key and value.

Methods in Map.Entry

Methods	Description
Object **getKey**()	It is used to obtain key.
Object **getValue**()	It is used to obtain value.

HashMap

- A HashMap contains values based on the key.

- It contains only unique elements.

- It may have one null key and multiple null values.

- It maintains no order.

HashMap class declaration

> **public class** HashMap<K,V> **extends** AbstractMap<K,V> **implements** Map<K,V>, Cloneable, Serializable

HashMap class Parameters

- **K**: It is the type of keys maintained by this map.

- **V**: It is the type of mapped values.

How to create a HashMap Object ?

Syntax:

Map<Type1,Type2> mapName = new HashMap<Type1,Type2>();

Creates a map with generics defined key of the type Type1 and value of the type Type2.
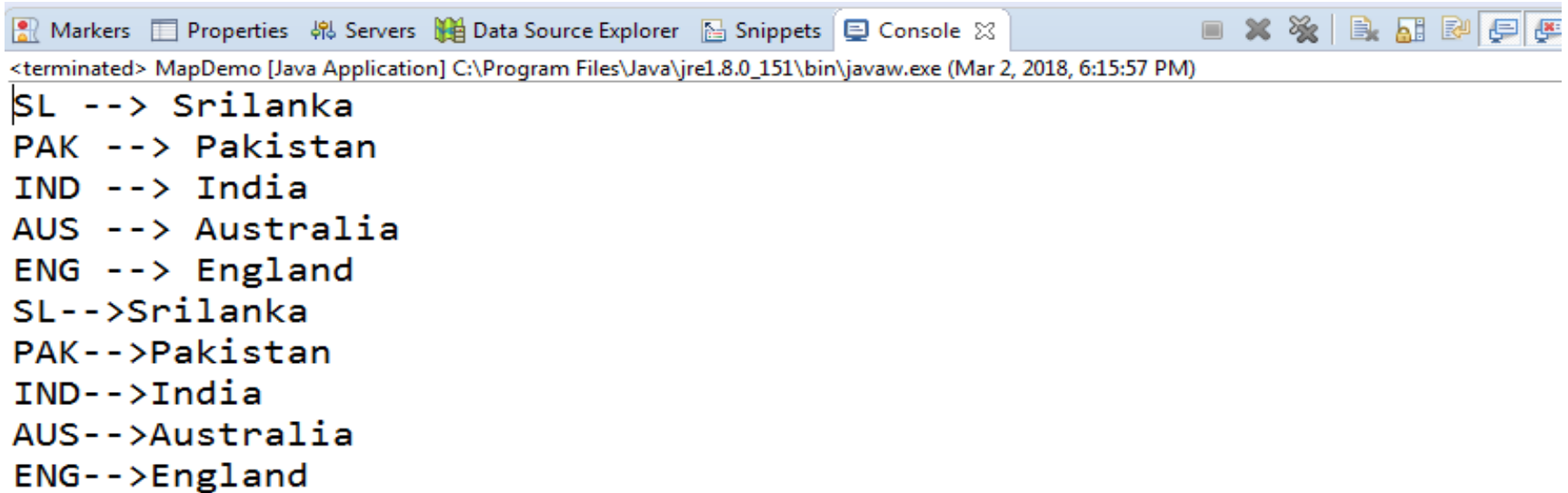
Example: Creates map object supporting Integer key and String value.

Map<Integer,String> myMap = new HashMap<Integer,String>();

Example – HashMap

```java
6  public class MapDemo {
7      public static void main(String[] args) {
8          HashMap<String, String> cMap = new HashMap<>();
9          cMap.put("IND", "India");
10         cMap.put("SL", "Srilanka");
11         cMap.put("PAK", "Pakistan");
12         cMap.put("AUS", "Australia");
13         cMap.put("ENG", "England");
14 //      Set keySet();
15         Set<String> keys = cMap.keySet();
16 //      Object get(Object key)
17         for(String key:keys) {
18             System.out.println(key+" --> "+cMap.get(key));
19         }
20 //      Set<Map.Entry<K,V>> entrySet()
21         for(Map.Entry<String, String> entry:cMap.entrySet()) {
22             System.out.println(entry.getKey() +"-->"+entry.getValue());
23         }
24     }
```

Output - HashMap

```
Markers   Properties   Servers   Data Source Explorer   Snippets   Console ☒           ■  ✖  ✖  🗎  🗎  🗎  🗎  🗎
<terminated> MapDemo [Java Application] C:\Program Files\Java\jre1.8.0_151\bin\javaw.exe (Mar 2, 2018, 6:15:57 PM)
SL --> Srilanka
PAK --> Pakistan
IND --> India
AUS --> Australia
ENG --> England
SL-->Srilanka
PAK-->Pakistan
IND-->India
AUS-->Australia
ENG-->England
```

What can be done to automatically sort the map according to the country code?
The solution is *TreeMap*, since Tree implements the **SortedMap** interface, it
automatically stores the elements in a sorted order based on the key.

Example - TreeMap

In this example we will see how **TreeMap** can be used to store elements sorted by the key. We will tweak the previous program to use **TreeMap** rather than **HashMap** implementation.

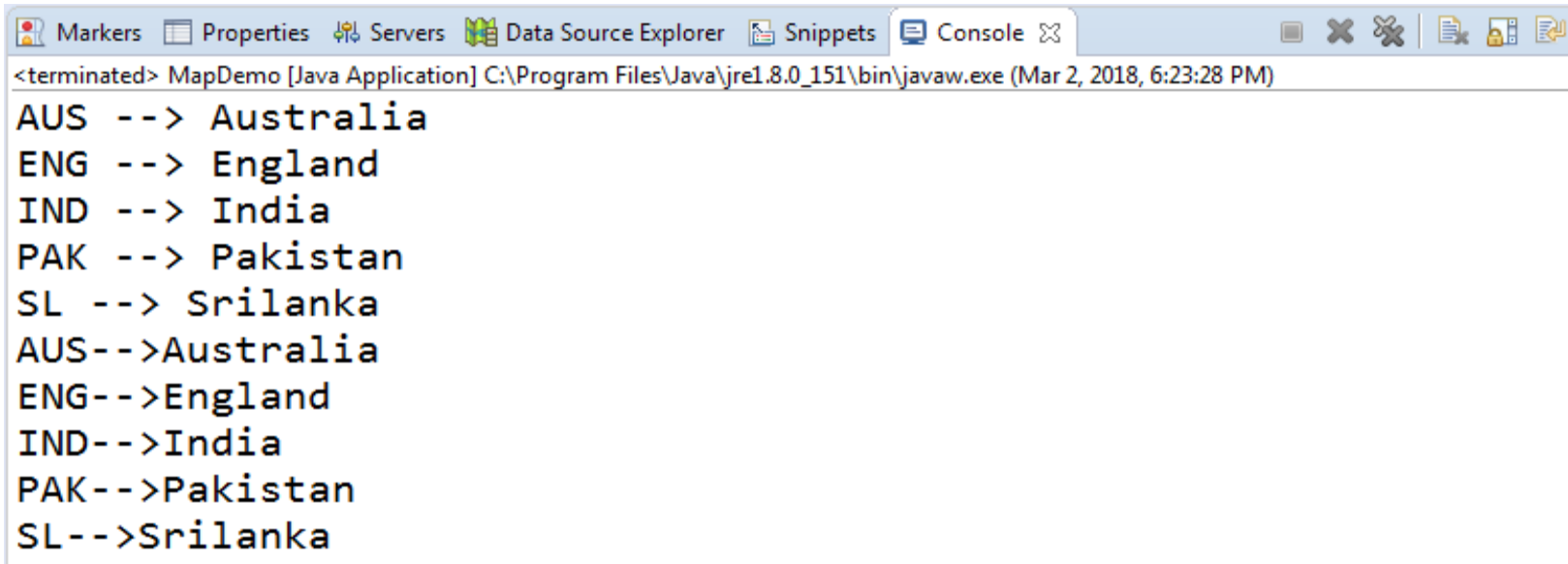Solution: Change the **HashMap** to **TreeMap** in the MapDemo class.

Map<String,String> studentMap=new HashMap<String, String>();

Map<String,String> studentMap=new TreeMap<String, String>();

Note : See how easily we changed the implementation from *HashMap* to *TreeMap* by making just one change. This is the advantage of declaring collections with the interface *Map*.

Output - TreeMap

```
Markers   Properties   Servers   Data Source Explorer   Snippets   Console ⌷

<terminated> MapDemo [Java Application] C:\Program Files\Java\jre1.8.0_151\bin\javaw.exe (Mar 2, 2018, 6:23:28 PM)
AUS --> Australia
ENG --> England
IND --> India
PAK --> Pakistan
SL --> Srilanka
AUS-->Australia
ENG-->England
IND-->India
PAK-->Pakistan
SL-->Srilanka
```

Notice the output the this **MapDemo** class are displayed based on the country code order.

Time To Reflect

Trainees to reflect the following topics before proceeding.

- What types of map can be used to if want to allow null values?
- Suppose you are asked to create a telephone directory with person's name in a sorted order which map implementation do you prefer to use ?

How to store user defined Objects in Collections

- User defined classes can be stored in any of the collections like list, set, map etc.

In the next lend a hand we will see how user defined objects can be added and retrieved from an *ArrayList*. Same procedure applies for all other collections.

NOTE: To add a user defined object to a sorted collection developers should either

- Make the User Defined object to implement Comparable interface (or)
- Create a custom comparator class and pass it as constructor parameter when creating the collection object.

Example – Add user defined object in collection

Consider a scenario in which you want to send the details of products to a method. The product details includes product id , product name , quantity, price and supplier name. Suppose there are 50 products what can be done to pass the details of all the 50 products together ?

Let us see how to solve it?

Step 1: Create a Product class with the prodId, prodName, quantity, price, supplierName as it's fields.

Step 2: Create an object of Product class for each product and load the details.

Step 3: Create an ArrayList and add each of the Product objects to the list.

Step 4: Pass the list as argument to the method.

Example – Add user defined object in collection

Let us create a ***ProductManager*** class with method ***printProductDetails()*** accepts the product details as a list of product objects.

Components to be developed:

1. **Product** Class : For holding the product details.

2. **ProductManager** class : Contains method for printing the product details.

3. **MainClass** class : Creates 5 product objects , adds them to a list and pass it to the ***printProductDetails()*** method of the ***ProductManager*** class.

Example – Create a Product class

The Product class should have the following fields to hold the product details

1. prodId

2. prodName

3. quantity

4. price

5. supplierName

All the fields should be created as private and accessed using public methods.

Product.java

```java
3  public class Product {
4      private int prodId;
5      private String prodName;
6      private int quantity;
7      private long price;
8      private String supplierName;
9      public Product(int prodId, String prodName, int quantity, long price, String supplierName) {
10         this.prodId = prodId;
11         this.prodName = prodName;
12         this.quantity = quantity;
13         this.price = price;
14         this.supplierName = supplierName;
15     }
16     public int getProdId() {
17         return prodId;
18     }
19     public String getProdName() {
20         return prodName;
21     }
22     public int getQuantity() {
23         return quantity;
24     }
25     public long getPrice() {
26         return price;
27     }
28     public String getSupplierName() {
29         return supplierName;
30     }
31 }
```

All the methods starting with the word "**get**" are used to read the associated field value.

Example : *getName()* returns the name.

The values to the member variables can be initialized using the overloaded constructor.

ProductManager.java

```java
6  public class ProductManager {
7      public static void printProductDetails(List<Product> list) {
8          Iterator<Product> it = list.iterator();
9          while(it.hasNext()) {
10             Product p = it.next();
11             System.out.println("Product Id: "+p.getProdId());
12             System.out.println("Product Name: "+p.getProdName());
13             System.out.println("Price: "+p.getPrice());
14             System.out.println("Ordered Quantity: "+p.getQuantity());
15             System.out.println("Supplier Name: "+p.getSupplierName());
16         }
17     }
18 }
19
```

MainClass.java

```java
 6 public class MainClass {
 7⊖     public static void main(String[] args) {
 8         Product p1 = new Product(11,"Acer",2,12500L,"Flipkart");
 9         Product p2 = new Product(12,"Samsung",10,25600L,"Amazon");
10         Product p3 = new Product(13,"Asus",3,14800L,"Flipkart");
11         Product p4 = new Product(14,"Sony",1,45000L,"Amazon");
12         Product p5 = new Product(15,"Oppo",5,12300L,"Snapdeal");
13         List<Product> pList = new ArrayList<>();
14         pList.add(p1);
15         pList.add(p2);
16         pList.add(p3);
17         pList.add(p4);
18         pList.add(p5);
19         ProductManager.printProductDetails(pList);
20     }
21 }
22
```

Creates five **Product** objects

Creates ArrayList objects and adds the **product** objects to the list.

Passes the list as input to **printProductDetails** method

Time To Reflect
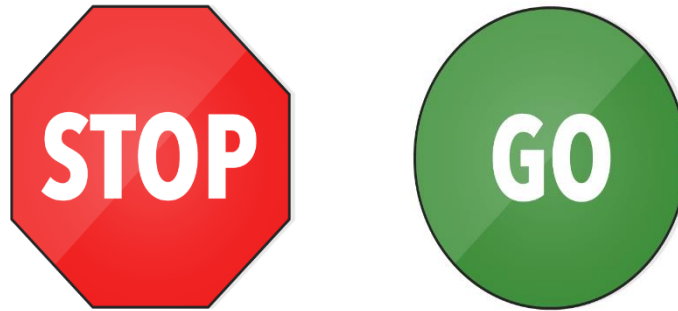
Trainees to reflect the following topics before proceeding.

- What types of map can be used to if want to allow null values?
- Suppose you are asked to create a telephone directory with person's name in a sorted order which map implementation do you prefer to use ?

Thank you

You have successfully completed
Collection
Framework