

## Core Java

# Collections in Java

LEVEL – PRACTITIONER



# About the Author

<b>Created By:</b>	Ashok Kumar Kanuparthi(g-Ashok3)/Renjith(t-renjith)/ Shanmu (105110)
<b>Credential Information:</b>	Trainer/Trainer/ Sr Architect
<b>Version and Date:</b>	1.0, January 1 <sup>st</sup> 2012

## Cognizant Certified Official Curriculum

# Icons Used



**Questions**



**Tools**



**Hands on  
Exercise**



**Coding  
Standards**



**Test Your  
Understanding**



**Case Study**



**Demonstration**



**Best Practices  
& Industry  
Standards**



**Workshop**

A graphic in the top-left corner showing several interlocking puzzle pieces. One piece is red, while the others are white and light blue.

# Objectives

After completing this session, you will be able to :

- Define Collections
- Describe usage of collections
- Describe the benefits of collections
- Understand the core Collection Interfaces
- Understand the implementations

# What are Collections?

## What is a collection?

A “collection” is a container that groups multiple elements into a single unit.



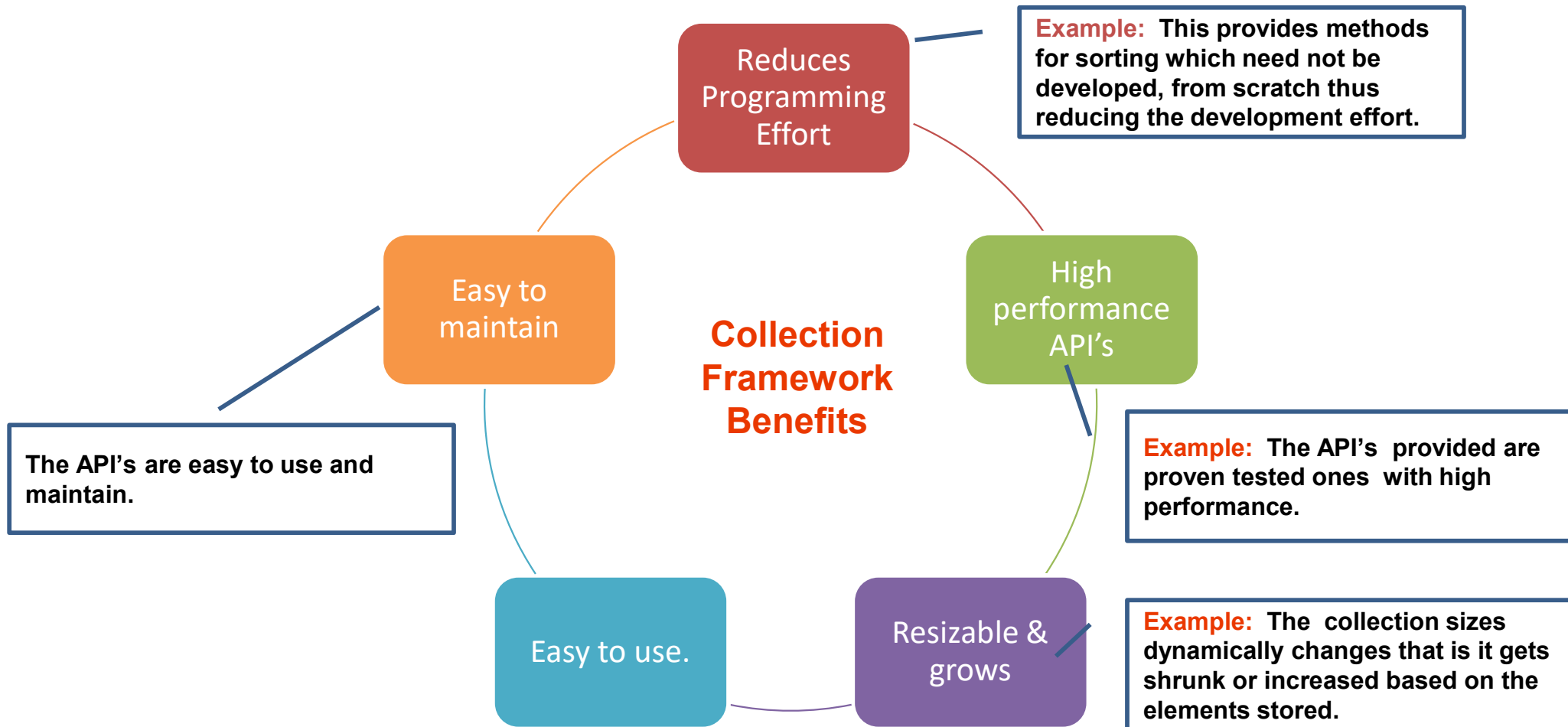
In java world, a collection represents a group of objects, known as ***elements***.

# Collections Framework

## What is a 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





# Collection Interfaces

## What are collection Interfaces?

These are set of predefined java interfaces defined in ***java.util*** package which can be used for performing any logic related to collection of Objects.

**Example:** Set, List, Map

## What are collection Implementations?

These are predefined classed which implements one of the collection interfaces. The programmer uses these for processing collection of Objects.

**Example:** ArrayList, HashMap.



# Sorted Vs Ordered Collection

**Consider you have a collection country names.**

1. India
2. Australia
3. England
4. New Zealand
5. South Africa

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 countries in the same order given above .

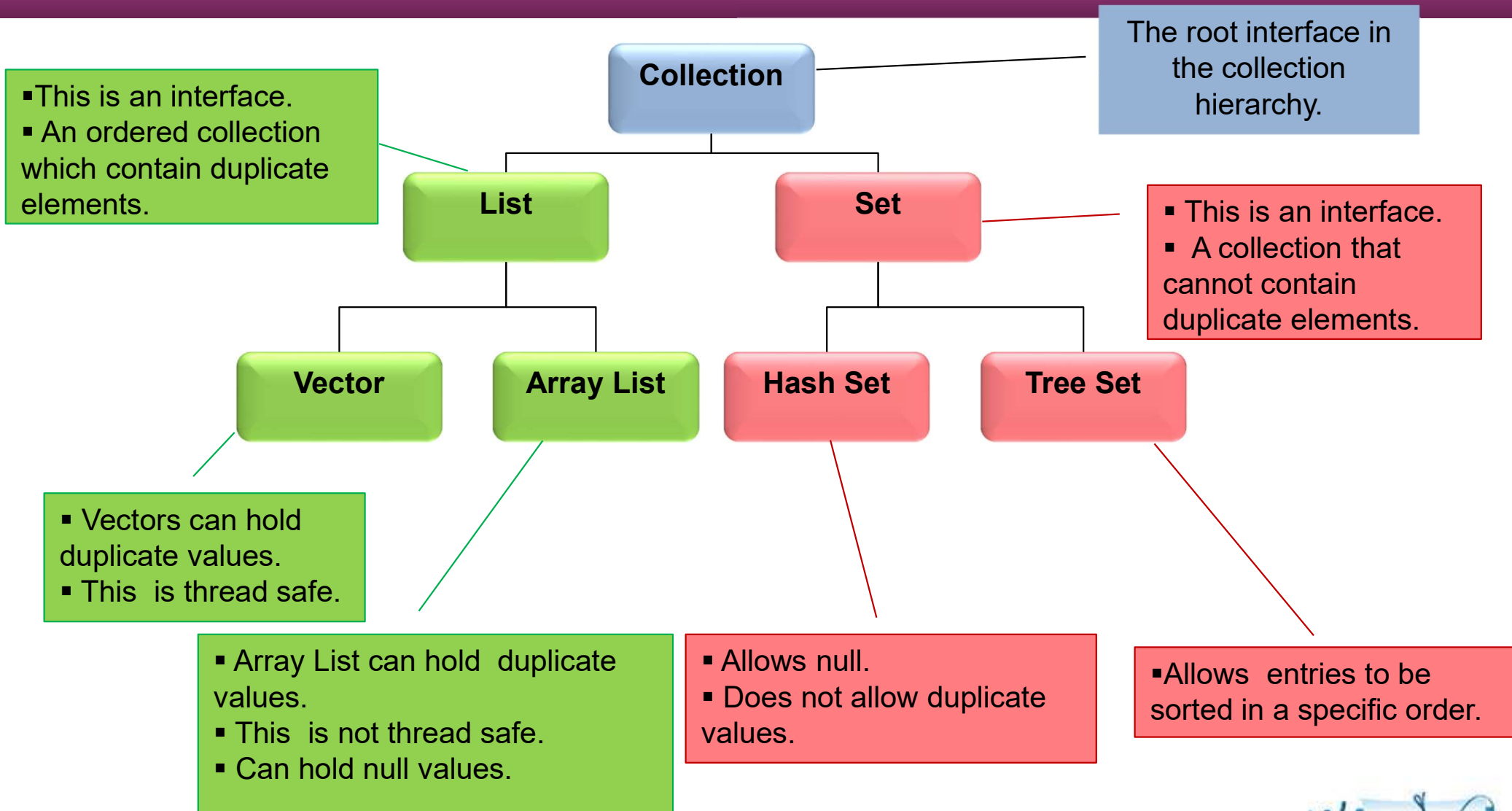
What type of collection can be used ?

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

**Problem 2:** Suppose you want to automatically sort the country 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 country is added to the list.

# Collection Framework Components



# Difference between Set and List

## List :

Cat	Dog	Bat	Bat	Lion	Cat	Lion	Bird	null	null
-----	-----	-----	-----	------	-----	------	------	------	------

It allows duplicates

It permits multiple null values.

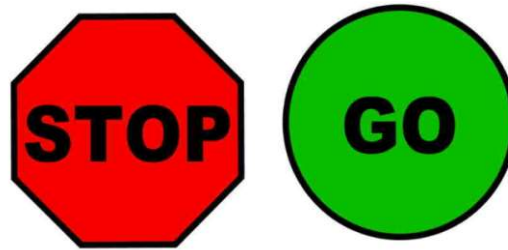
## Set:

Cat	Dog	Bat	Lion	Bird	null
-----	-----	-----	------	------	------

It does not allow duplicates

Only one null value is permitted

# Time To Reflect



Associates to reflect the following before proceeding.

- What is the collection type that can be used to hold duplicate values?
- What type of collection can be used for thread safe storage of elements with duplication?
- What type of collection can be used to store unique elements in a sorted order?

# 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.

For more details on the methods specified in Collection interface refer,

<http://docs.oracle.com/javase/6/docs/api/java/util/Collection.html>

# List Interface

- Used for storing the elements in a ordered way.
- Supports duplicate entries.
- Supports index based additions and retrieval of items.

## List implementations:

1. Vector
2. ArrayList
3. LinkedList

We will be covering only **ArrayList** implementation in details. For details on the other **List** Implementations refer

**LinkedList**- <http://docs.oracle.com/javase/6/docs/api/java/util/LinkedList.html>.

**Vector** - <http://docs.oracle.com/javase/6/docs/api/java/util/Vector.html>.



# Array List

- 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.
- 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.



# Important ArrayList methods

Method	Description
void add(int index, Object element)	Inserts the specified element at the specified position in this list.
boolean add(Object o)	Appends the specified element to the end of this list.
boolean addAll(Collection)	Appends all of the elements in the specified Collection to the end of this list, in the order that they are returned by the specified Collection's Iterator.
boolean addAll(int index, Collection c)	Inserts all of the elements in the specified Collection into this list, starting at the specified position.
void clear()	Removes all of the elements from this list.
boolean contains(Object elem)	Returns true if this list contains the specified element.
Object get(int index)	Returns the element at the specified position in this list.

# Important ArrayList methods

Method	Description
<code>int indexOf(Object elem)</code>	Searches for the first occurrence of the given argument, testing for equality using the equals method.
<code>int lastIndexOf(Object elem)</code>	Returns the index of the last occurrence of the specified object in this list.
<code>Object remove(int index)</code>	Removes the element at the specified position in this list.
<code>void removeRange(int fromIndex, int toIndex)</code>	Removes from this List all of the elements whose index is between fromIndex, inclusive and toIndex, exclusive.
<code>Object set(int index, Object element)</code>	Replaces the element at the specified position in this list with the specified element.
<code>int size()</code>	Returns the number of elements in this list.
<code>Object[] toArray()</code>	Returns an array containing all of the elements in this list in the correct order.

<http://docs.oracle.com/javase/1.4.2/docs/api/java/util/ArrayList.html>

# How to Create an Array List ?

ArrayList can be created by using any of the three possible constructors given below

1. **ArrayList()** : Constructs an empty list.
2. **ArrayList(Collection c)** : Constructs a list from an existing collection, containing the elements of the specified collection.
3. **ArrayList(int initialCapacity)** : Constructs an empty list with the specified initial capacity.

**Syntax :**

```
List listName=new ArrayList();
```

**Example :**

```
List mylist=new ArrayList();
```

Why **ArrayList** object is declared as the interface type **List**?

# Need for declaring using interface type?

Consider the code below what happens if the **ArrayList** API becomes deprecated ? How many changes should be done in the below code ?

```
public void managerList() {  
    ArrayList numberList = getList();  
    printList(numberList);  
}  
  
public void printList(ArrayList numbers) {  
    System.out.println(numbers);  
}  
  
public ArrayList getList() {  
    ArrayList numberList = new ArrayList();  
    for (int i = 0; i < 10; i++) {  
        numberList.add(i);  
    }  
    return numberList;  
}
```

All the red boxes are changes to be done.

**Lets see how to reduce it.**

# Need for declaring using interface type? (Cont)

The solution is declaring the reference variable as the interface type rather than of the implementation type. Example all the list objects can be declared using **List** type.

```
public void managerList() {  
    List numberList = getList();  
    printList(numberList);  
}  
  
public void printList(List numbers) {  
    System.out.println(numbers);  
}  
  
public List getList() {  
    List numberList = new ArrayList();  
    for (int i = 0; i < 10; i++) {  
        numberList.add(i);  
    }  
    return numberList;  
}
```

Now change needs to be done in one place, where the object is created.

# How to add elements in a ArrayList?

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()
```

```
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.





# Lend a Hand - 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 countries 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.



# Lend a Hand Solution – Problem Statement 1

Create a class named ArrayListEx and develop the method as mentioned below and from the main method trigger this method and check the output.

```
public List getCountryList(String c1, String c2, String c3, String c4,  
    String c5) {  
    List countryList = new ArrayList();  
    countryList.add(c1);  
    countryList.add(c2);  
    countryList.add(c3);  
    countryList.add(c4);  
    countryList.add(c5);  
    return countryList;  
}
```

Creates an array list object and adds the country names to it.



Always remember to declare the list using **List** interface.

## Lend a Hand Solution – Problem Statement 2

Add the following method in the ***ArrayListEx*** class and from the main method trigger this method and check the output.

```
public List get1To10() {  
    List numList = new ArrayList();  
    for (int i = 1; i <= 10; i++) {  
        numList.add(i);  
    }  
    return numList;  
}
```



**Note:** When a primitive is added to a collection it gets automatically converted to its equivalent wrapper object and loaded in the list.

## Lend a Hand Solution – Problem Statement 3

Add the following method in the ArrayListEx class and from the main method trigger this method and check the output.  
The list returned by the previous problem statement should be passed as an input to this method.

```
public List get1To15(List numList) {  
    List numList2 = new ArrayList();  
    numList2.addAll(numList);  
    for (int i = 11; i <= 15; i++) {  
        numList2.add(i);  
    }  
    return numList2;  
}
```

**addAll ()** method adds the entire list to the new list

# Lend a Hand Solution – Run the program

Create a Main Class and execute the methods.

```
public class ArrayExMain {  
    public static void main(String args[]){  
        ArrayListEx ex1=new ArrayListEx();  
        System.out.println(ex1.getCountryList("India", "Australia", "England", "South Africa","New Zealand"));  
        List numList=ex1.get1To10();  
        System.out.println(numList);  
        System.out.println(ex1.get1To15(numList));  
    }  
}
```

## Output



```
<terminated> ArrayExMain [Java Application] C:\Program Files\Java\jdk1.6.0_20\bin\javaw.exe (Feb 3, 2012 6:13:26 PM)  
[India, Australia, England, South Africa, New Zealand]  
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]  
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]
```



# 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.
- **Sorted Set** interface is a set that maintains its elements in ascending order.

## Set Implementations:

1. HashSet

2. TreeSet

We will be covering only **HashSet** in detail in this session. For details on **TreeSet** visit <http://docs.oracle.com/javase/1.4.2/docs/api/java/util/TreeSet.html>



# HashSet

- Implements the Set interface hence doesn't support duplicate entries.
- Does not support random access or retrieval.
- Cannot predict the order in which the items get stored in the Set.
- Not Thread safe.
- Permits Null value(Only one).

# How to create an HashSet ?

**HashSet** can be created by using the following syntax

```
Set setName=new HashSet();
```

**Example :**

```
Set countries=new HashSet();
```



# Important Methods in HashSet

Method	Description
<code>boolean add(Object element)</code>	Adds the specified element to this set if it is not already present.
<code>void clear()</code>	Removes all of the elements from this set.
<code>boolean contains(Object o)</code>	Returns true if this set contains the specified element.
<code>boolean isEmpty()</code>	Returns true if this set contains no elements.
<code>Iterator&lt;E&gt; iterator()</code>	Returns an iterator over the elements in this set.
<code>boolean remove(Object o)</code>	Removes the specified element from this set if it is present.
<code>int size()</code>	Returns the number of elements in this set.

<http://docs.oracle.com/javase/1.4.2/docs/api/java/util/HashSet.html>

# 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("India");
```

```
mySet.add("Australia");
```

```
mySet.add("India")
```

```
mySet.addAll(mySet1);
```

Adds two elements into the set.

Returns false since India already exists in the set.

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.



# Lend a Hand - HashSet

**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 countries 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.

# Lend a Hand Solution – Problem Statement 1

Create a class SetEx, add the below method to the class. Invoke the method from a main method and test the method.

```
public Set getCountries(String c1, String c2, String c3, String c4,  
    String c5) {  
    Set countries = new HashSet();  
    countries.add(c1);  
    countries.add(c2);  
    countries.add(c3);  
    countries.add(c4);  
    countries.add(c5);  
    return countries;  
}
```



Always remember to declare the collection using the appropriate interface. In the example it is **Set**.

## Lend a Hand Solution – Problem Statement 2

Add the below method to the SetEx class. Invoke the method from a main method and test the method.

```
public Set get1To10() {  
    Set numSet = new HashSet();  
    for (int i = 0; i <= 10; i++) {  
        numSet.add(i);  
    }  
    return numSet;  
}
```



**Note:** When a primitive is added to a collection it gets automatically converted to its equivalent wrapper object and loaded in the list.

## Lend a Hand Solution – Problem Statement 3

Add the below method to the SetEx class. Invoke the method from a main method and test the method.

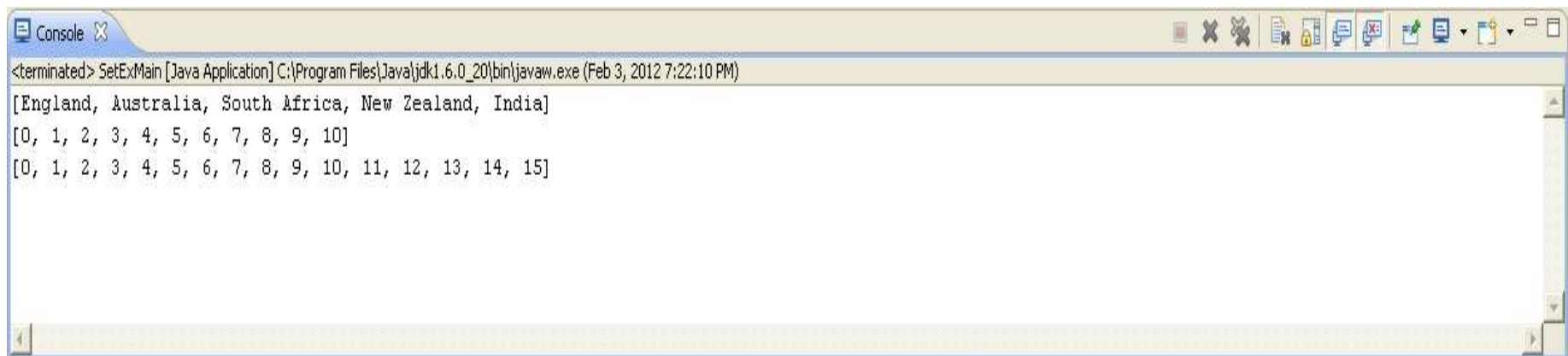
```
public Set get11To15(Set numSet1) {  
    Set numSet2 = new HashSet();  
    numSet2.addAll(numSet1);  
    for (int i = 11; i <= 15; i++) {  
        numSet2.add(i);  
    }  
    return numSet2;  
}
```

# Lend a Hand Solution – Main Class

Create a main class as given below and execute it.

```
public class SetExMain {  
    public static void main(String args[]) {  
        SetEx ex1 = new SetEx();  
  
        System.out.println(ex1.getCountries("India", "Australia",  
            "South Africa", "England", "New Zealand"));  
        Set num1To10 = ex1.get1To10();  
        System.out.println(num1To10);  
        System.out.println(ex1.get11To15(num1To10));  
    }  
}
```

## Output

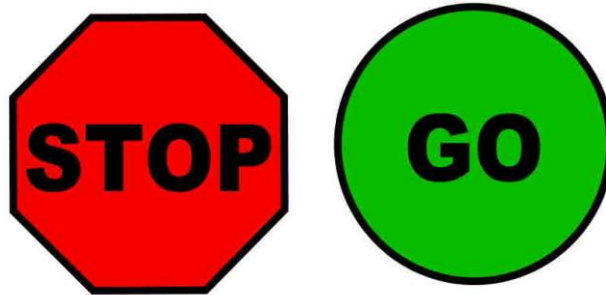


The screenshot shows a Java console window titled "Console X". The command prompt shows the execution of the SetExMain class using javaw.exe. The output consists of three lines: the first line lists the countries "England, Australia, South Africa, New Zealand, India"; the second line shows the set of numbers from 0 to 10; and the third line shows the set of numbers from 0 to 15.

```
<terminated> SetExMain [Java Application] C:\Program Files\Java\jdk1.6.0_20\bin\javaw.exe (Feb 3, 2012 7:22:10 PM)  
[England, Australia, South Africa, New Zealand, India]  
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]  
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]
```



# Time To Reflect



Associates to reflect the following before proceeding.

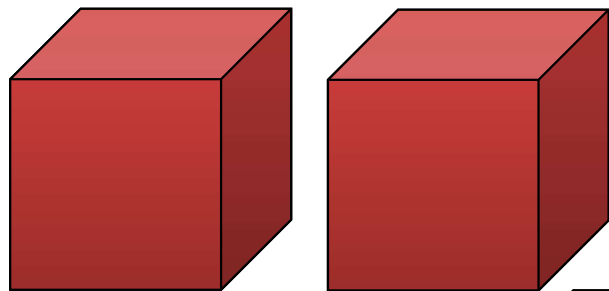
- What method can be used to merge to two array lists ?
- What happens when a duplicate entry is tried to be added to a set ?
- What is the difference between add and set method in arraylist?
- How to add an element to a particular position in an array list?



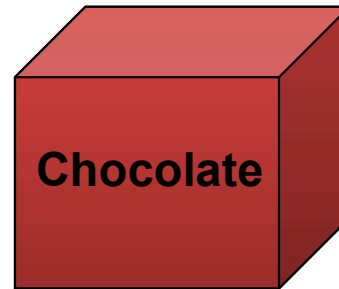
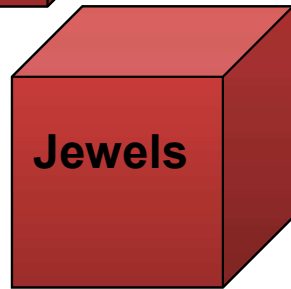
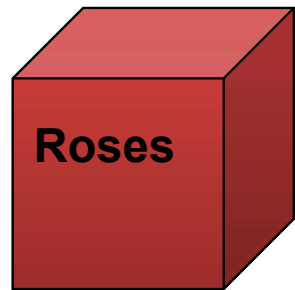
Let us now learn  
about “**Generics**”

# Real World Example

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



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

# How to identify data type in a collection?

Similar to the boxes how can one know what is the data type of the element stored in a collection?

**Generics is the solution.**

# 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.

# Let's Analyze Generics Programmatically.

We will take the same box example but instead of jewels , chocolates and roses we will store java objects inside it. Consider a Box class,

```
public class Box
{
    private Integer width;
    public void addWidth(Integer width) {
        this.width = width;
    }
    public Integer getWidth() {
        return width;
    }
}
```

Assume that I have to pass width as a Float (or) Double to the Box class. How can I achieve this?

**Generics is the solution.**

# Generics solves the problem.

We will modify the Box class with generics as shown below.

```
public class Box<Type> {  
    // T stands for "Type"  
    private Type width;  
    public void addWidth(Type width) {  
        this.width = width;  
    }  
    public Type getWidth() {  
        return width;  
    }  
}
```

As shown in the example **Type** refers to the data type of the “**width**” field which is declared as generic argument of the class. Instead of declaring “**width**” as Integer we have declared as a generic data type “**Type**”. When declaring Box object, we can pass any data type for **Type** which affects all the **Type** declarations inside the class.

**Example:** **Box<Float>** will declare a box object whose width field data type is “Float”

An object can be created as

```
Box<Float> box=new Box<Float>();
```

By doing this we can ensure the same Box behaving differently for different data type.



# Generics usage in Collections

- Collections declared with generic (data type) ensures that the collections hold only elements of particular data type.
- Java collections have been defined with generics for it accept objects of any data type.

**Example:** Java *ArrayList* class is defined with a generic as shown below depicting that the collection can accept generics with any data type.

```
public class ArrayList<E> extends AbstractList<E>
    implements List<E>, RandomAccess, Cloneable, java.io.Serializable
{
    private static final long serialVersionUID = 8683452581122892189L;
```

# Using Generics With Collection

We can declare collection to hold a specific data type elements using generics .

## Syntax:

```
InterfaceName <Type> varName=new Implementation<Type>();
```

**Example :** Declares an array list for holding only integer values.

```
List<Integer> myList=new ArrayList<Integer>();
```

# Advantages of using Generics with 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

```
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);  
    }  
}
```

Type Casted to String

## With Generics

```
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);  
    }  
}
```

No need of type casting



# Advantages of using Generics with 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.



# Lend a Hand- Generics

**Objective:** Let us learn how to declare/process collections declared with generics.

**Problem Statement 1 :** Create a method which can accept a collection of country names and add it to ArrayList with generic defined as String and return the List.

**Problem Statement 2 :** Create a method which can create a HashSet containing values 1-10. The Set should be declared with the generic type Integer. The method should return the Set.

# Lend a Hand – Solution Problem Statement 1

Create a class named GenericEx and add the method given below.

## Generic Declaration

```
public List<String> getCountryList(String c1, String c2, String c3, String c4,  
    String c5) {  
    List<String> countryList = new ArrayList<String>();  
    countryList.add(c1);  
    countryList.add(c2);  
    countryList.add(c3);  
    countryList.add(c4);  
    countryList.add(c5);  
    return countryList;  
}
```



Try to add a Integer number to the List and see how the program behaves.

## Lend a Hand Solution – Problem Statement 2

Add the below method to GenericEx class.

```
public Set<Integer> get1To10() {  
    Set<Integer> numSet = new HashSet<Integer>();  
    for (int i = 1; i <= 10; i++) {  
        numSet.add(i);  
    }  
    return numSet;  
}
```

Generic Declaration

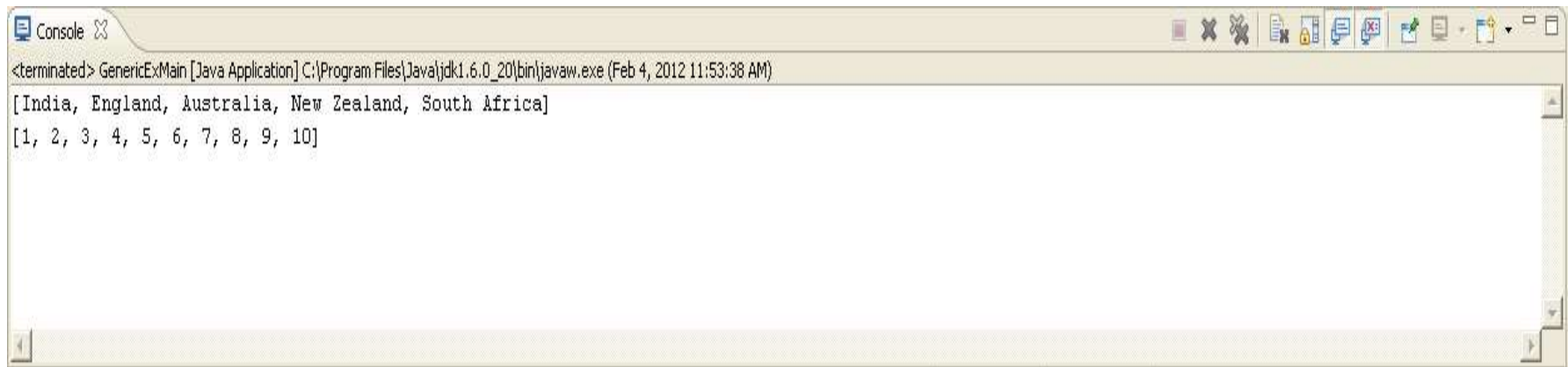


# Lend a Hand – Main Class

Create a class named GenericExMain with a main method

```
public class GenericExMain {  
    public static void main(String args[]) {  
        GenericsEx ex1 = new GenericsEx();  
        System.out.println(ex1.getCountryList("India", "England", "Australia",  
            "New Zealand", "South Africa"));  
        System.out.println(ex1.get1To10());  
    }  
}
```

## Output



The screenshot shows a Java console window titled "Console". The command prompt shows the execution of the GenericExMain class. The output consists of two lines: the first line lists the countries [India, England, Australia, New Zealand, South Africa] and the second line lists the numbers [1, 2, 3, 4, 5, 6, 7, 8, 9, 10].

```
<terminated> GenericExMain [Java Application] C:\Program Files\Java\jdk1.6.0_20\bin\javaw.exe (Feb 4, 2012 11:53:38 AM)  
[India, England, Australia, New Zealand, South Africa]  
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

# 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.

# Lend a Hand – Iterating Collection

We will go through each of the iteration mechanism using a lend a hand example.

We will reuse the classes we developed as part of the previous lend a hand,

1. ArrayListEx
2. SetEx



# Lend a Hand :For Loop

**Objective:** We will learn how to the iterate through the List objects using a for loop. Create a class with a main method which reads the lists from the ArrayListEx and iterates through the list.

## Problem Statement 1 :

The main method should read the country list and read it one by one , store in a String object and print the country names in separate lines.

## Problem Statement 2:

The main method should retrieve the list of numbers from 1-10 using the get1To10 method in ArrayListEx class and find the sum of numbers in the list and print the sum.

# Lend a Hand– Problem Statement 1

Create a class named ForLoopExMain with a main method. Add the following lines of code inside main method which can read the Country list from ArrayListEx and print it.

```
ArrayListEx ex1 = new ArrayListEx();  
List countryList = ex1.getCountryList("India", "Australia", "England",  
    "South Africa", "New Zealand");  
int count = countryList.size();  
for (int i = 0; i < count; i++) {  
    String country = (String) countryList.get(i);  
    System.out.println(country);  
}
```

Reads the size of the list

The for loop iterates from 0 to size-1 and reads elements one by one.



**Note :** Since Generics is not used, we can see that elements are explicitly casted to String type.

## Lend a Hand Solution – Problem Statement 2

Add the following lines of code inside the main method to read the number list from ArrayListEx and calculate the sum of all the numbers in the list.

```
List numList = ex1.get1To10();  
int numCount = numList.size();  
int sum = 0;  
for (int i = 0; i < numCount; i++) {  
    sum = sum + (Integer) numList.get(i);  
}  
System.out.println("Sum : " + sum);
```

# For-Each loop

1. For each loop was added as a part of java 5
2. Can be used to read elements from a collection or array
3. The advantage of “**for each**” is that we don’t need to specify the index. It automatically iterates over the specified object and reads values one by one.

## Syntax :

```
for(datatype variableName : collectionName){  
    loop body  
}
```

Reads each element of the collection **collectionName** with the type **datatype** and assigns it to the variable **variableName** one by one.

## Example

```
for(Object a : myList){  
    System.out.println(a);  
}
```

Reads elements as objects one by one from the list “**MyList**” and assign to the variable **a**.



# Lend a Hand – for Each loop

**Objective:** We will learn how to use for each loop for iterating through Sets.

We will use the Set created in **SetEx** class which we developed in our previous lend a hand. Create a class named **ForEachExMain** with a main method which iterates through the Set.

**Problem Statement 1 :** The main method should read the set of countries returned by **getCountries()** method in **SetEx** class and print the names of countries in separate lines.

**Problem Statement 2 :** The main method also should calculate the sum of numbers in the set returned by the **get1To10()** method in the **SetEx** class.

**Prerequisite:** Declare the Sets with respective generic types. The **getCountries()** method should return a Set with String generic type and **get1To10()** method should return an Set with generic type Integer.

# Lend a Hand :For Each loop

Adding generics to the collections of **getCountries()** and **get1To10()** methods.

```
public Set<String> getCountries(String c1, String c2, String c3, String c4,
    String c5) {
    Set<String> countries = new HashSet<String>();
    countries.add(c1);
    countries.add(c2);
    countries.add(c3);
    countries.add(c4);
    countries.add(c5);
    return countries;
}

public Set<Integer> get1To10() {
    Set<Integer> numSet = new HashSet<Integer>();
    for (int i = 0; i <= 10; i++) {
        numSet.add(i);
    }
    return numSet;
}
```

# Lend a Hand Solution – Problem Statement 1

Create a class named ***ForEachExMain*** with a main method

Add the following lines of code to read the countries set from the SetEx class and iterate using for each loop and print it.

```
SetEx ex1 = new SetEx();  
Set<String> countrySet = ex1.getCountries("India", "Australia",  
    "England", "New Zealand", "South Africa");  
for (String country : countrySet) {  
    System.out.println(country);  
}
```

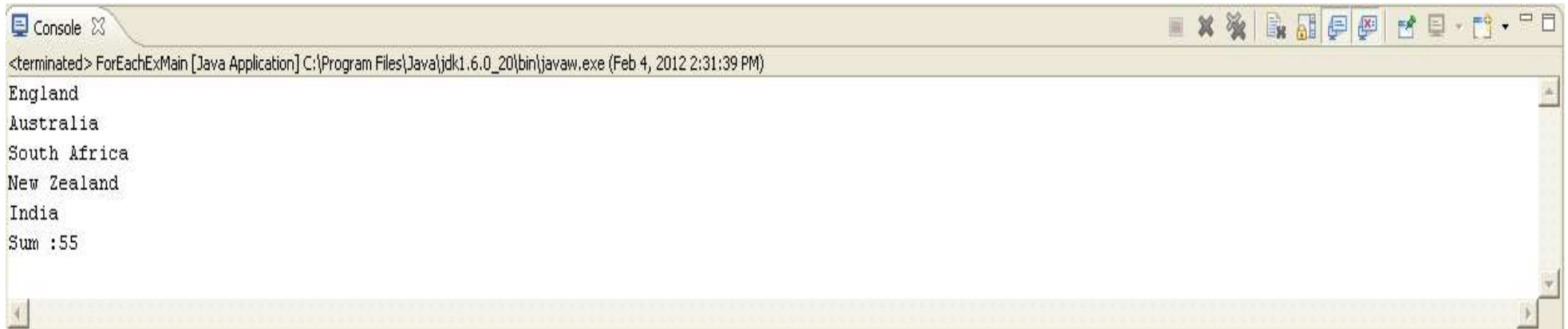
## Lend a Hand – Problem Statement 2

Add the following lines of code to the main method to find the sum of numbers in the number set read from the SetEx class

```
Set<Integer> num1To10 = ex1.get1To10();  
int sum = 0;  
for (Integer number : num1To10) {  
    sum = sum + number;  
}  
System.out.println("Sum :" + sum);
```

# Lend a Hand – Execute the main class

Run the class ***ForEachExMain***



The screenshot shows a Java console window titled "Console". The command prompt shows the execution of the `ForEachExMain` class using `javaw.exe`. The output lists five countries and their sum.

```
<terminated> ForEachExMain [Java Application] C:\Program Files\Java\jdk1.6.0_20\bin\javaw.exe (Feb 4, 2012 2:31:39 PM)  
England  
Australia  
South Africa  
New Zealand  
India  
Sum :55
```



# 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

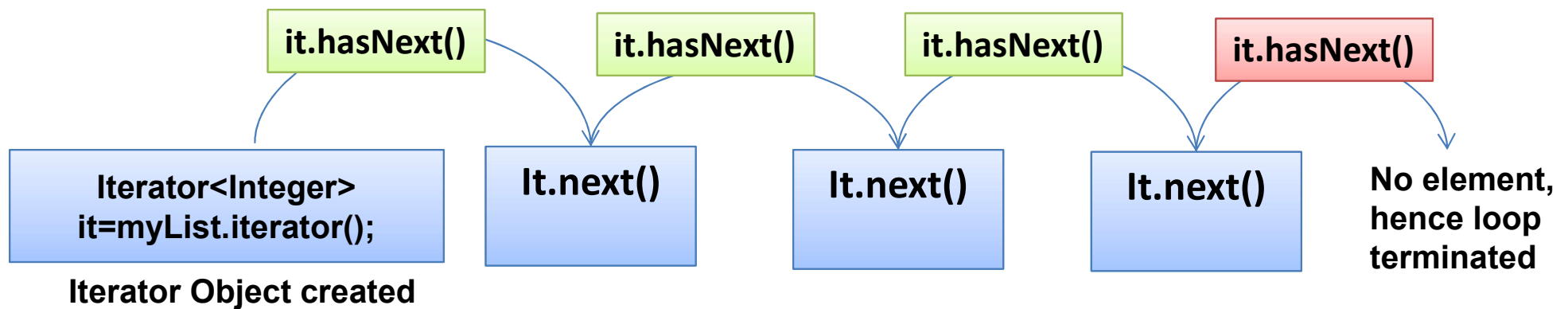
**Example :** Creates an iterator to iterate the elements of the **myList** containing **Integer** objects.

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



# How an iterator works?

## Working of Iterator



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()***.

# Steps for using Iterator

**Step 1 :** Create Iterator object

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

**Step 2 :** Check whether next element exist using the ***hasNext()*** method

```
while(iterator.hasNext()) {  
    // Read Elements  
}
```

**Step 3 :** Read the element using the ***next()*** method.

```
while(iterator.hasNext()) {  
    int number=iterator.next();  
}
```



# Lend a Hand Iterator

**Objective:** We will learn how to use an Iterator to iterate through a Set. We will use the SetEx class developed in the previous lend a hand for this demo. The Sets in the SetEx class is expected to be declared using generics..

**Problem Statement :** Iterate through the countries (Set) returned by the ***getCountries()*** method of the SetEx class and print the country names one by one.

# Lend a Hand – Iterator Solution.

Create a class named IteratorExMain as shown below.

```
public class IteratorExMain {  
    public static void main(String args[]) {  
        SetEx ex1 = new SetEx();  
        Set<String> countrySet = ex1.getCountries("India", "Australia",  
            "England", "New Zealand", "South Africa");  
        Iterator<String> iterator = countrySet.iterator();  
        while (iterator.hasNext()) {  
            String string = iterator.next();  
            System.out.println(string);  
        }  
    }  
}
```

Create an iterator to work on the set

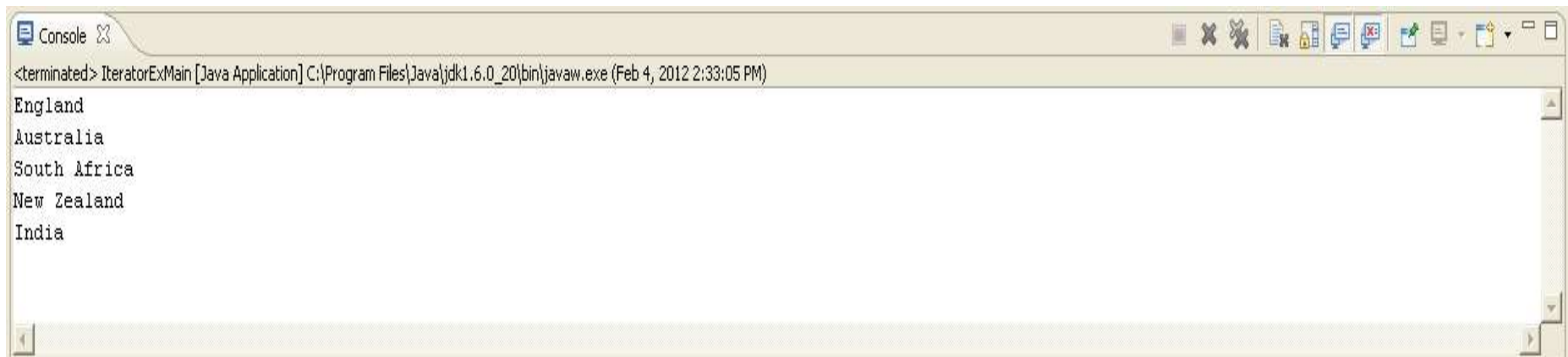
Invoke the hasNext() method to check whether the iterator contains elements to be read.

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

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

# Lend a Hand – Execute the main method

**Run the class named IteratorExMain**



The screenshot shows a Java IDE console window titled "Console". The output text is as follows:

```
<terminated> IteratorExMain [Java Application] C:\Program Files\Java\jdk1.6.0_20\bin\javaw.exe (Feb 4, 2012 2:33:05 PM)  
England  
Australia  
South Africa  
New Zealand  
India
```

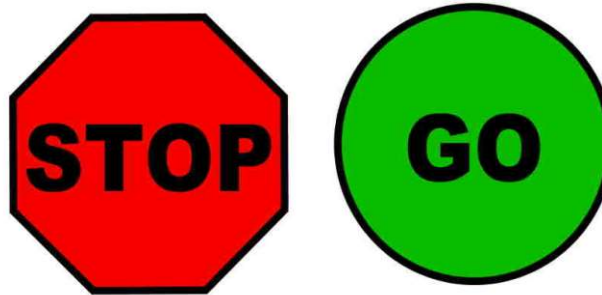
# ListIterator

- 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().



**Note :** In projects Iterator interface are used commonly. So we are not going to learn the implementation of ListIterator. Associates can refer to the oracle documentation for more details on ***ListIterator***.

# Time To Reflect



Associates to reflect the following 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?



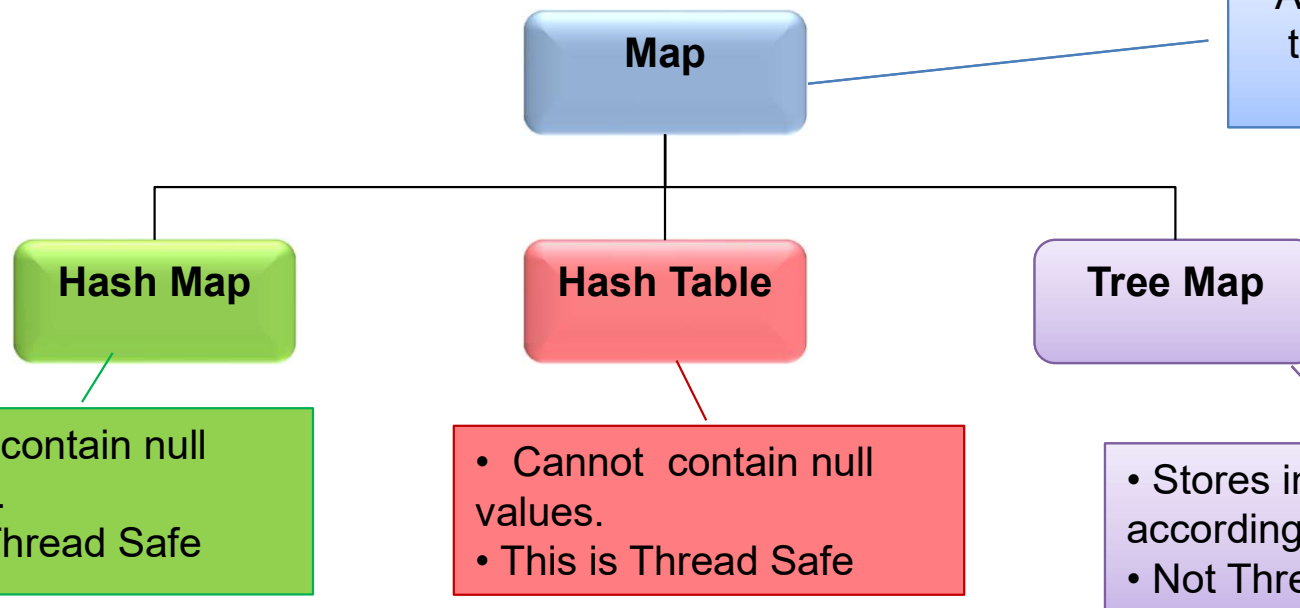
# Maps

- 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	India
2	Australia
3	England
4	South Africa
5	New Zealand

# Map Interfaces

A Map is an object that maps keys to values



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.

# Map Interface

Map interface is the root interface in the map hierarchy

All map implementations should implement this interface directly or indirectly.

## Important Methods:

Methods	Description
<code>void clear()</code>	Removes all mappings from this map (optional operation).
<code>boolean containsKey(Object key)</code>	Returns true if this map contains a mapping for the specified key.
<code>boolean containsValue(Object value)</code>	Returns true if this map maps one or more keys to the specified value.
<code>Set entrySet()</code>	Returns a set view of the mappings contained in this map.
<code>Object get(Object key)</code>	Returns the value to which this map maps the specified key.
<code>boolean isEmpty()</code>	Returns true if this map contains no key-value mappings.

# Map Interface – Important Methods(Cont)

Method	Description
Set keySet()	Returns a set with all the keys contained in this map.
Object put(Object key, Objectvalue)	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
Object remove(Object key)	Removes the mapping for this key from this map if it is present
int size()	Returns the number of key-value mappings in this map.
Collection values()	Returns a collection with all the values contained in this map.

A graphic in the top-left corner showing several interlocking puzzle pieces. One piece in the foreground is red, while the others are white and blue.

# HashMap

- Implementation of the **Map** interface.
- Permits null values for key(**only one key**) and value.
- Not Thread safe.
- Does not maintain the order of elements in the Map.

# 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>();
```



Always remember to declare the Map using the appropriate interface. In the example it is **Map**.



# How to add and retrieve elements in a HashMap?

**Adding Elements to a Map:** Elements can be added to a map as key - value pairs using the `put()` method

**Syntax :** `map.put(key,value);`

**Example:** `map.put(1,"India");` // This adds an country *India* with a key value 1.

**Reading Elements From a Map:** Elements can be retrieved from a map by passing the key using the `get()` method.

**Syntax :** `variable =map.get(key);`

**Example:** `String country=map.get(1);` // This retrieves the country specific to key 1 which is "*India*".

# keySet() method

- **keySet()** returns the set of keys for the map
- Using the keys obtained one can iterate the map.

```
Set<Integer> keys=map.keySet();
```

Get the keys of the Map.

```
Iterator<Integer> iterator=keys.iterator();
```

Get the iterator of the Set.

```
while(iterator.hasNext())
```

```
{
```

```
    System.out.println(iterator.next());
```

Iterating the set and printing the key values.

```
}
```



# Lend a Hand - HashMap

**Objective:** Let us learn to store values in a map, iterate through the map and print the values.

**Problem Statement 1:** Create a method that returns a Hash map containing the details of students. Roll no is the key and student name is the value.

**Problem Statement 2 :** Create a main method that reads the student map and prints the roll number and name of all the students by iterating through the keys.

# Lend a Hand Solution – Problem Statement 1

Create a class named MapEx. It should loads a map with student registration number as key and student name as value and returns the student map.

```
public Map<String,String> getStudentDetails(){  
    Map<String,String> studentMap=new HashMap<String, String>();  
    studentMap.put("2","Arun");  
    studentMap.put("3","Prithvi");  
    studentMap.put("1","Tom");  
    studentMap.put("4","Irfan");  
    return studentMap;  
}
```



Always remember to declare the Map using the appropriate interface. In the example it is **Map**.

# Lend a Hand Solution – Problem Statement 2

Let us now iterate through the map and print the name and the registration number.

```
public class MapExMain {  
    public static void main(String args[]) {  
        MapEx ex1 = new MapEx();  
        Map<String, String> studentMap = ex1.getStudentDetails();  
        System.out.println(studentMap);  
        Set<String> keys = studentMap.keySet();  
        Iterator<String> iterator = keys.iterator();  
        System.out.println("Students Details");  
        while (iterator.hasNext()) {  
            String rollNo = iterator.next();  
            String name = studentMap.get(rollNo);  
            System.out.println("Roll No : " + rollNo + " Name : " + name);  
        }  
    }  
}
```

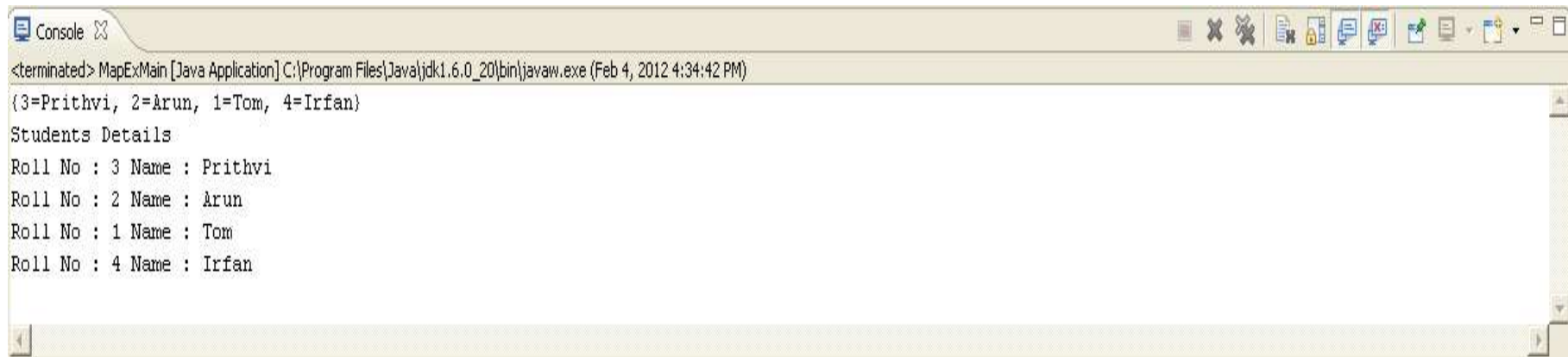
Prints the map

Gets the key Set for the map

Gets the iterator for the keyset

Iterates over the key set and reads the value for each key.

# Lend a Hand - Output



```
<terminated> MapExMain [Java Application] C:\Program Files\Java\jdk1.6.0_20\bin\javaw.exe (Feb 4, 2012 4:34:42 PM)
{3=Prithvi, 2=Arun, 1=Tom, 4=Irfan}
Students Details
Roll No : 3 Name : Prithvi
Roll No : 2 Name : Arun
Roll No : 1 Name : Tom
Roll No : 4 Name : Irfan
```

**What can be done to automatically sort the map according to the registration number ?**

The solution is ***TreeMap***, since Tree implements the SortedMap interface, it automatically stores the elements in a sorted order based on the key.

# Lend a Hand – TreeMap Demo

In this lend a hand 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 getStudentDetails() method in MapEx class.

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

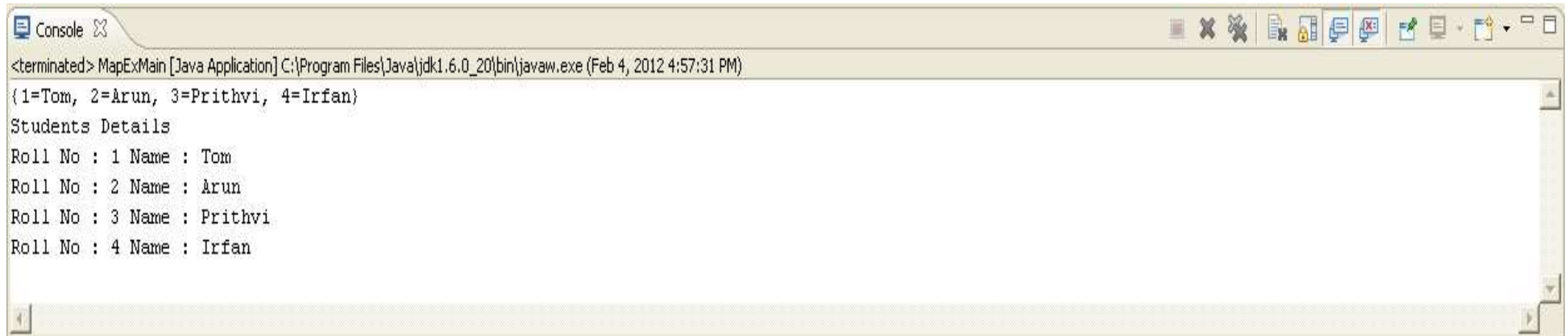


```
Map<String,String> studentMap=new TreeMap<String, String>();
```

**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**.



# Lend a Hand – TreeMap Output

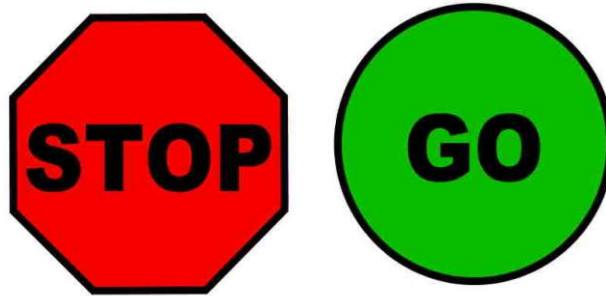


The screenshot shows a Java console window titled "Console". The output text is as follows:

```
<terminated> MapExMain [Java Application] C:\Program Files\Java\jdk1.6.0_20\bin\javaw.exe (Feb 4, 2012 4:57:31 PM)
{1=Tom, 2=Arun, 3=Prithvi, 4=Irfan}
Students Details
Roll No : 1 Name : Tom
Roll No : 2 Name : Arun
Roll No : 3 Name : Prithvi
Roll No : 4 Name : Irfan
```

Notice the output the student records are displayed based on the registration number order.

# Time To Reflect



Associates to reflect the following 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.

Refer the following links for more details :

<http://docs.oracle.com/javase/1.4.2/docs/api/java/lang/Comparable.html>

<http://docs.oracle.com/javase/1.3/docs/api/java/util/Comparator.html>



## Lend a Hand – Adding user defined object in collection

Consider a scenario in which you want to send the details of students to a method. The student details includes roll number , name , address, phone number and email id. Suppose there are 50 students what can be done to pass the details of all the 50 students together ?

**Let us see how to solve it?**

**Step 1 :** Create a Student class with the roll number, name , address as it's fields.

**Step 2 :** Create an object of Student class for each student and load the details.

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

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



## Lend a Hand – Adding user defined object in collection

Let us create a ***StudentManager*** class with method ***printStudentDetails()*** accepts the student details as a list of student objects.

### Components to be developed:

1. **Student Class** : For holding the student details.
2. **StudentManager class** : Contains method for printing the student details.
3. **MainClass class** : Creates 5 student objects , adds them to a list and pass it to the ***printStudentDetails()*** method of the ***StudentManager*** class.

# Lend a Hand : Create Student class

**The Student class should have the following fields to hold the student details**

1. roll number
2. name
3. address
4. phone number
5. email id

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



# Lend a Hand : Student class Code

```
public class Student {  
    private String rollNo;  
    private String name;  
    private String address;  
    private String phone;  
    private String email;  
  
    public Student(String rollNo, String name, String address, String phone,  
        String email) {  
        this.rollNo = rollNo;  
        this.name = name;  
        this.address = address;  
        this.phone = phone;  
        this.email = email;  
    }  
    public String getRollNo() {  
        return rollNo;  
    }  
  
    public String getName() {  
        return name;  
    }  
  
    public String getAddress() {  
        return address;  
    }  
  
    public String getPhone() {  
        return phone;  
    }  
  
    public String getEmail() {  
        return email;  
    }  
}
```

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.



# Lend a Hand : Main Class Code

```
public class MainClass {  
    public static void main(String args[]) {  
        Student s1 = new Student("1", "Arun", "Flat No: 126/A ", "1223654789",  
            "arun@gmail.com");  
        Student s2 = new Student("2", "Irfan", "Flat No: 134/A ", "1298774914",  
            "irfan@gmail.com");  
        Student s3 = new Student("3", "Tom", "Flat No: 180/c ", "9874587487",  
            "tomy@gmail.com");  
        Student s4 = new Student("4", "Jancy", "Flat No: 116/A ", "8087874698",  
            "jancy@gmail.com");  
        Student s5 = new Student("5", "Vikram", "Flat No: 141/A ",  
            "7879521455", "vikz@gmail.com");  
        List<Student> studentList = new ArrayList<Student>();  
        studentList.add(s1);  
        studentList.add(s2);  
        studentList.add(s3);  
        studentList.add(s4);  
        studentList.add(s5);  
        new StudentManager().printStudentDetails(studentList);  
    }  
}
```

Creates five Student objects

Creates ArrayList objects and adds the student objects to the list.

Passes the list as input to printStudentDetails method

# Lend a Hand : Create StudentManager class

## Code

The printStudentDetails() method accepts list of students as argument.

```
public class StudentManager {  
    public void printStudentDetails(List<Student> studentList) {  
        Iterator<Student> iterator = studentList.iterator();  
        while (iterator.hasNext()) {  
            Student student = iterator.next();  
            System.out.println("Roll No : " + student.getRollNo());  
            System.out.println("Name : " + student.getName());  
            System.out.println("Address : " + student.getAddress());  
            System.out.println("Phone : " + student.getPhone());  
            System.out.println("Email : " + student.getEmail());  
        }  
    }  
}
```

## Core Java

**You have successfully completed -  
Collections in Java**

