

# JAVA@11

Java Scanner API

# Objective

After completing this session you will be able to understand,

- Working with new Scanner API

# Why Scanner API?

What good is a video game if you are not allowed to provide any input?



You all know to display text on the console from a java program, but how can you take an input from a user?

**Use Scanner  
Class**

# java.util.Scanner

- Prior to Java 1.5, getting input from the console involved some complex steps.
- Java 1.5 introduced the Scanner class which simplifies console input.
- It can also be used to read from files and Strings.
- It also can be used as a pattern matching utility.
- Scanner is available in the java.util package.

# How Scanner is works?

## How Scanner works?

- The Scanner class parses input from the source into tokens by using delimiters to identify the token boundaries.

**Example:** Shri, Hari, Saro, Iyan

**Token 1:** Shri **Token 2:** Hari **Token 3:** Saro ....

- The default set of delimiters consists of the whitespace characters, space, tab, newline and carriage return.
- Scanner will read a line of input from its source.

## Syntax:

```
Scanner <objectName>= new Scanner(System.in);
```

System.in is an **InputStream** used to receive data inputs from console using the keyboard.

# Scanner APIs

Method	Description
<i>String</i> <b>next()</b>	Finds and returns the next token from the input as a string
<i>String</i> <b>nextLine()</b>	Finds and returns the next token from the input as a string
<i>boolean</i> <b>nextBoolean()</b>	Scans the next token of the input as a boolean
<i>byte</i> <b>nextByte()</b>	Scans the next token of the input as a byte
<i>short</i> <b>nextShort()</b>	Scans the next token of the input as a short
<i>int</i> <b>nextInt()</b>	Scans the next token of the input as a integer
<i>long</i> <b>nextLong()</b>	Scans the next token of the input as a long
<i>float</i> <b>nextFloat()</b>	Scans the next token of the input as a float
<i>double</i> <b>nextDouble()</b>	Scans the next token of the input as a double



**Important: InputMismatchException** can be thrown if you try to get the next token using a next method that does not match the type of the token.

# How to work with Scanner Class?

Let us learn on how to use the Scanner class to read user input

**Step 1:** Create an instance of Scanner class.

```
Scanner scan = new Scanner(System.in);
```

**Step 2:** User name retrieved as input from the console, Invoke the `nextLine()` method in the following way

```
String userName = scan.nextLine();
```

**Step 3:** Age retrieved as input from the console, Invoke `nextInt()` method in the following way.

```
int age = scan.nextInt();
```

# Example – Scanner class

**Problem Statement:** Develop a java program which requests the user to enter the user name and his age and print a message welcoming the user

Develop a class ***ScanDemo*** with a main method that performs the below operations

1. The program should display the message in console **“Enter your name”** once the user has entered the name.
2. The program should display the message **“Enter your age”**
3. Once the user has entered the age.
4. The program should display a welcome message as mentioned below,  
**“Welcome to Mr./Ms.<user name>, Your age is <user age>”**



# Example – Scanner class

Solution:

```
public class ScanDemo {  
    public static void main(String[] args) {  
        Scanner scan = new Scanner(System.in);  
        System.out.println("Enter your name");  
        String name = scan.nextLine();  
        System.out.println("Enter your age");  
        int age = scan.nextInt();  
        System.out.println("Welcome to Mr./Ms."+name+", Your age is "+age);  
        scan.close();  
    }  
}
```

Thank you

*You have successfully completed*  
**Scanner API**

# JAVA @11

String API

# Objective

After completing this session you will be able to understand,

- Introduction to String Class
- StringBuffer and StringBuilder classes
- StringTokenizer class
- Define equals() and hashCode() methods.

# String class

**Strings**, which are widely used in Java programming, they *are a sequence of characters*.

- Java provides **String** class to create and process strings.
- Strings are **objects**.

## How to create a String?

**Option 1:** String *greeting* = "Hello world!";

// Create a string literal and assign it to a String reference.

(OR)

**Option 2:** String *greeting* = new String("Hello world!");

// Using the String constructor.

# About String class

String class is available from ***java.lang*** package.

## **What does String class contains?**

String class contains the APIs used for creating and processing strings.

### **Example:**

- Comparing Strings
- Searching Strings
- Extracting the sub string
- Also constructors for creating String in different ways.

# Example – String constructor

Lets develop a program to explore the various String constructors to build a String

```
public class StringConstructDemo {  
  
    public static void main(String[] args) {  
        String s1 = new String();  
        { char chars[] = { 'h', 'e', 'l', 'l', 'o' };  
          String s2 = new String(chars);  
        { byte bytes[] = { 'w', 'o', 'r', 'l', 'd' };  
          String s3 = new String(bytes);  
          String s4 = new String(chars, 1, 3);  
          String s5 = new String(s2);  
          String s6 = s2;  
        }  
        System.out.println(s1);  
        System.out.println(s2);  
        System.out.println(s3);  
        System.out.println(s4);  
        System.out.println(s5);  
        System.out.println(s6);  
    }  
}
```

Creating string  
objects from a  
character array.

From byte array.

From other char  
objects.

From other string objects.

# String class APIs

Some commonly used APIs inside the String class

Return type	Method	Description
<i>char</i>	<b>charAt(int index)</b>	Returns the character at the specified index.
<i>int</i>	<b>compareTo(String obj)</b>	Compares two strings lexicographically.
<i>boolean</i>	<b>equalsIgnoreCase(String str)</b>	Compares this String to another String, ignoring case considerations.
<i>Boolean</i>	<b>equals(Object another)</b>	checks the equality of string with object
<i>int</i>	<b>indexOf(int ch)</b>	Returns the index within this string of the first occurrence of the specified character.
<i>int</i>	<b>length()</b>	Returns the length of this string.



# String class APIs

Return type	Method	Description
<i>String</i>	<b>concat(String str)</b>	Concatenates the specified string to the end of this string.
<i>String</i>	<b>replace(char oldChar, char newChar)</b>	Returns a new string resulting from replacing all occurrences of oldChar in this string with newChar.
<i>String</i>	<b>substring(int beginIndex, int endIndex)</b>	Returns a new string that is a substring of this string.
<i>String</i>	<b>trim()</b>	Returns a copy of the string, with leading and trailing whitespace omitted.
<i>String[]</i>	<b>split(String regex)</b>	returns splitted string matching regex

# Key points

## Key points on Strings:

- Strings are **immutable** i.e. once it is created a String object cannot be changed.
- If you assign a String reference to a new String, the old String will be lost.

## Example:

```
String str1="Hello";
```

```
str1="World";
```

Now, str1 contains "World"

- All string operations (**concatenate, trim, replace, substring**) construct and return new strings.
- String class is final.

# Example – String APIs

Lets develop a program to explore the few API's of String class, namely,  
print a specific character of a string, compare, print length of string, replace a string etc

```
public class StringImplDemo {  
    public static void main (String[] args) {  
        String str = "Eyeopen";  
        String str1 = "Eyeopen";  
        String str2 = "eyeopen";  
        String str3 = "          Eyeopen Technologies          ";  
        System.out.println("4th Position is "+str.charAt(4));  
        System.out.println("Size of string is "+str.length());  
        System.out.println("To compare str & str1:: "+str.equals(str1));  
        System.out.println("To compare str & str2:: "+str.equals(str2));  
        System.out.println("To compare str & str2:: "+str.equalsIgnoreCase(str2));  
        System.out.println("To upper case:: "+str2.toUpperCase());  
        System.out.println("To lower case:: "+str.toLowerCase());  
        System.out.println(str3.trim());  
        System.out.println(str3.substring(8));  
        System.out.println(str3.substring(2, 5));  
        String str4="India,Pakistan,Australia,South Africa,Sri lanka,England";  
        String[] country = str4.split(",");  
        for(String name:country) {  
            System.out.println(name);  
        }  
    }  
}
```

# StringBuffer

**StringBuffer** objects unlike Strings are **mutable** i.e. string value can be changed. So string buffer is like a String, that can be **modified**.

- Using the API's in the StringBuffer the content and the length of the string can be changed without creating a new object.
- The API's of StringBuffer are **synchronized**.
- This class is preferred when modification of character strings are needed since it is efficient in memory utilization.

## Examples:

- Appending String
- Inserting characters in string.
- Deleting characters from a string

# StringBuffer class

*StringBuffer* is a final class.

StringBuffer objects can be created empty,

```
StringBuffer strBuf = new StringBuffer();
```

StringBuffer can be created from a String.

```
StringBuffer strBuf = new StringBuffer("Bob");
```

StringBuffer can be created with a capacity.

```
StringBuffer strBuf = new StringBuffer(100);
```

# StringBuffer class

String buffers are used by developers to concatenate String rather than using concatenation operator "+" on string objects.

StringBuffer is efficient than "+" concatenation.

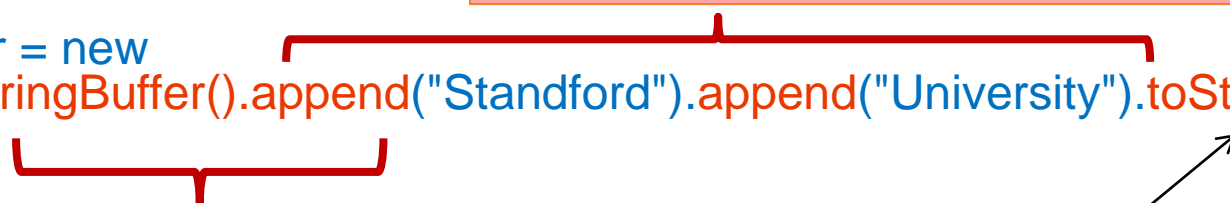
## Example:

```
String str="Stanford";
```

```
str = str+ "University";
```

Can be developed as

```
str = new  
StringBuffer().append("Stanford").append("University").toString();
```



Step 2: Appends the string representation of each operand to the string buffer in turn.

Step 1: Compiler creates a new string buffer initially empty.

Step 3: Converts the contents of the string buffer to a string as 'str' is a string object.

# StringBuffer APIs

Return type	Method	Description
<i>void</i>	<b>setCharAt(int index, char ch)</b>	The character at the specified index of this string buffer is set to the character ch.
<i>StringBuffer</i>	<b>insert(int offset, String str)</b>	Inserts the string argument into this string buffer.
<i>StringBuffer</i>	<b>delete(int start, int end)</b>	Removes the characters in a substring of this StringBuffer.
<i>StringBuffer</i>	<b>replace(int start, int end, String str)</b>	Replaces the characters in a substring of this StringBuffer with characters in the specified String.

# StringBuffer APIs

Return type	Method	Description
<i>StringBuffer</i>	<b>reverse()</b>	The character sequence contained in this string buffer is replaced by the reverse of the sequence.
<i>StringBuffer</i>	<b>append(String str)</b>	Appends the string to this string buffer.
<i>void</i>	<b>setLength(int newLength)</b>	Sets the length of this String buffer.
<i>String</i>	<b>substring(int start, int end)</b>	Returns a new String that contains a subsequence of characters currently contained in this StringBuffer.



# Example – StringBuffer APIs

Lets develop a program to explore the few API's of String Buffer class.

We will solve the following problem,

1. Append two Strings “Hello” & “World” . Output: “Hello World”
2. Insert a string “\_Java” in the String after “Hello”. Output: “Hello\_Java World”
3. Replace \_ with space. Output: “Hello Java World”.
4. Print the character at the 6'th position . Output: J
5. Delete the character in the third position. Output: “Helo World”
6. Print the capacity of the buffer.
7. Reverse the string and print the string. Output: “dlroW avaJ oleH”

# Solution – StringBuffer APIs

```
public class StringBufferDemo {  
    public static void main(String[] args) {  
        StringBuffer strbuf = new StringBuffer("Hello");  
        System.out.print(strbuf.length());  
        strbuf.append("World");  
        System.out.println(strbuf);  
        strbuf.insert(5, "_Java ");  
        System.out.println(strbuf);  
        strbuf.setCharAt(5, ' ');  
        System.out.println(strbuf);  
        System.out.print("Character at 6th position : ");  
        System.out.println(strbuf.charAt(6));  
        strbuf.deleteCharAt(3);  
        System.out.println(strbuf);  
        System.out.print("Capacity of StringBuffer object : ");  
        System.out.println(strbuf.capacity());  
        strbuf.reverse();  
        System.out.print("Reversed string : ");  
        System.out.println(strbuf);  
    }  
}
```

This returns 5.

This returns "HelloWorld".

This returns "Hello\_Java World".

This returns "Hello Java World".

This returns 'J'.

This returns "Helo Java World".

This returns '21'.

This returns "dlroW avaJ oleH".

# StringBuilder

***StringBuilder*** class, which is a drop-in replacement for StringBuffer.

- StringBuilder is not synchronized which means it is not thread-safe.
- Use StringBuilder class where thread safety is not an issue.
- It offers faster performance than StringBuffer.
- All the methods available on StringBuffer are also available on StringBuilder, so it really is a drop-in replacement.

# Example – StringBuilder APIs

Lets develop a program to explore the few API's of String Builder class.

We will solve the following problem,

1. Append two Strings “Hello” & “World” . Output: “HelloWorld”
2. Insert a string “\_Java” in the String after “Hello”. Output: “Hello\_Java World”
3. Replace \_ with space. Output: “Hello Java World”.

# Example – StringBuilder APIs

```
public class StringBufferDemo {  
    public static void main(String[] args) {  
        StringBuilder strbuf = new StringBuilder("Hello");  
        System.out.println(strbuf.length());  
        strbuf.append("World");  
        System.out.println(strbuf);  
        strbuf.insert(5, "_Java ");  
        System.out.println(strbuf);  
        strbuf.setCharAt(5, ' ');  
        System.out.println(strbuf);  
    }  
}
```

The diagram consists of four red rectangular boxes with black text, each connected to a line of code by a black arrow. The arrows originate from the right side of the code lines and point to the left side of the boxes. The boxes contain the following text: "This returns 5.", "This returns 'HelloWorld'.", "This returns 'Hello\_Java World'.", and "This returns 'Hello Java World'."

- This returns 5.
- This returns "HelloWorld".
- This returns "Hello\_Java World".
- This returns "Hello Java World".

# StringTokenizer class

The ***StringTokenizer*** class is used to break a string into tokens based on some delimiters.

**Example:** India, USA, UK, Russia – This string can be split based on the delimiter “,”

- It is available in **java.util** package.
- ***StringTokenizer*** implements the Enumeration interface.
- The given string can be enumerated, you can enumerate the individual tokens contained in it using ***StringTokenizer***.

***Tokens from the String:***

**Token 1-** India **Token 2-** USA **Token 3-** UK **Token 4-** Russia

# StringTokenizer APIs

The default delimiters are whitespace characters. space, tab, newline, and carriage return.

Return type	Method	Description
<i>boolean</i>	<b>hasMoreTokens()</b>	Tests if there are more tokens available from this tokenizer's string.
<i>String</i>	<b>nextToken()</b>	Returns the next token in this string tokenizer's string.

# Example - StringTokenizer

Lets develop a program to explore the ***StringTokenizer***

```
public class TokenizerDemo {  
    public static void main(String[] args) {  
        String str = "Eyeopen Technologies, 18th Main Road, Anna Nagar, Chennai.";  
        StringTokenizer sToken = new StringTokenizer(str, ",");  
        while(sToken.hasMoreTokens()) {  
            System.out.println(sToken.nextToken());  
        }  
    }  
}
```

Run the program and check the output.

Now execute the same program without using delimiter and see the output.

```
StringTokenizer sToken = new StringTokenizer(str);
```



# equals() hashCode() method

**java.lang.Object** has methods called *hashCode()* and *equals()*.

These methods can be overridden and implemented with the object specific logic,

## **What is a Hash code?**

Hash code is an unique id number allocated to an object by JVM. This number is maintain through the lifecycle of the Object.

## **hashCode()**

- This method by default returns the hash code value of the object on which this method is invoked.
- This method returns the hash code value as an integer.
- You can develop your own logic of generating hash code.

# equals() hashCode() method

## What is a Equals method used for?

This particular method is used for comparing two objects for equality.

## **equals()**

***equals()*** refers to equivalence relation **i.e.** you say that two objects are equivalent they satisfy the “equals()” condition.

Override the ***equals()*** with a logic which needs to be used for comparing for equivalence.

## **Example:**

Assume we have a object Employee with the following instance variables, `EmployeeId`, `EmployeeName`.

The developer can override the equals method and compare the employee Id for checking equivalence.

# Example – equals and hashCode

Lets develop a program to explore how equals and hash code works.

Create a *Employee* object with instance variable age and name. Override the hash code and equals method as mentioned below,

- ***Equals*** – The method overridden to compare the age of the employees if same they should return a true else return false.
- ***hashCode*** – Should return the age as hash code.

# Solution – equals and hashCode

```
public class Emp {  
    private int age ;  
  
    public Emp( int age )  
    {  
        super();  
        this.age = age;  
    }  
  
    public int hashCode()  
    {  
        return age;  
    }  
  
    public boolean equals( Object obj )  
    {  
        boolean flag = false;  
        Emp emp = ( Emp )obj;  
        if( emp.age == age )  
            flag = true;  
        return flag;  
    }  
}
```

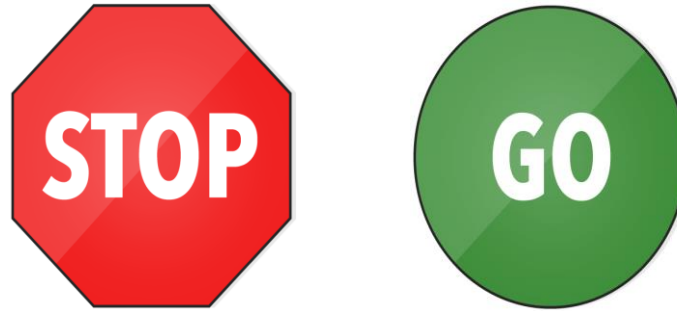
```
public class TestEmp {  
    public static void main(String[] args) {  
        Emp emp1 = new Emp(23);  
        Emp emp2 = new Emp(23);  
        System.out.println("emp1.equals(emp2)--->>>" + emp1.equals(emp2));  
    }  
}
```

Overriding hashCode ().

This returns true.

Overriding equals().

# Time To Reflect



**Trainees to reflect the following topics before proceeding.**

- What makes implementing Runnable interface better than extending Thread class for Thread Creation?
- How can a Thread wait with out finishing for another Thread to get completed?

Thank you

*You have successfully completed* **String**  
**API**

# 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.

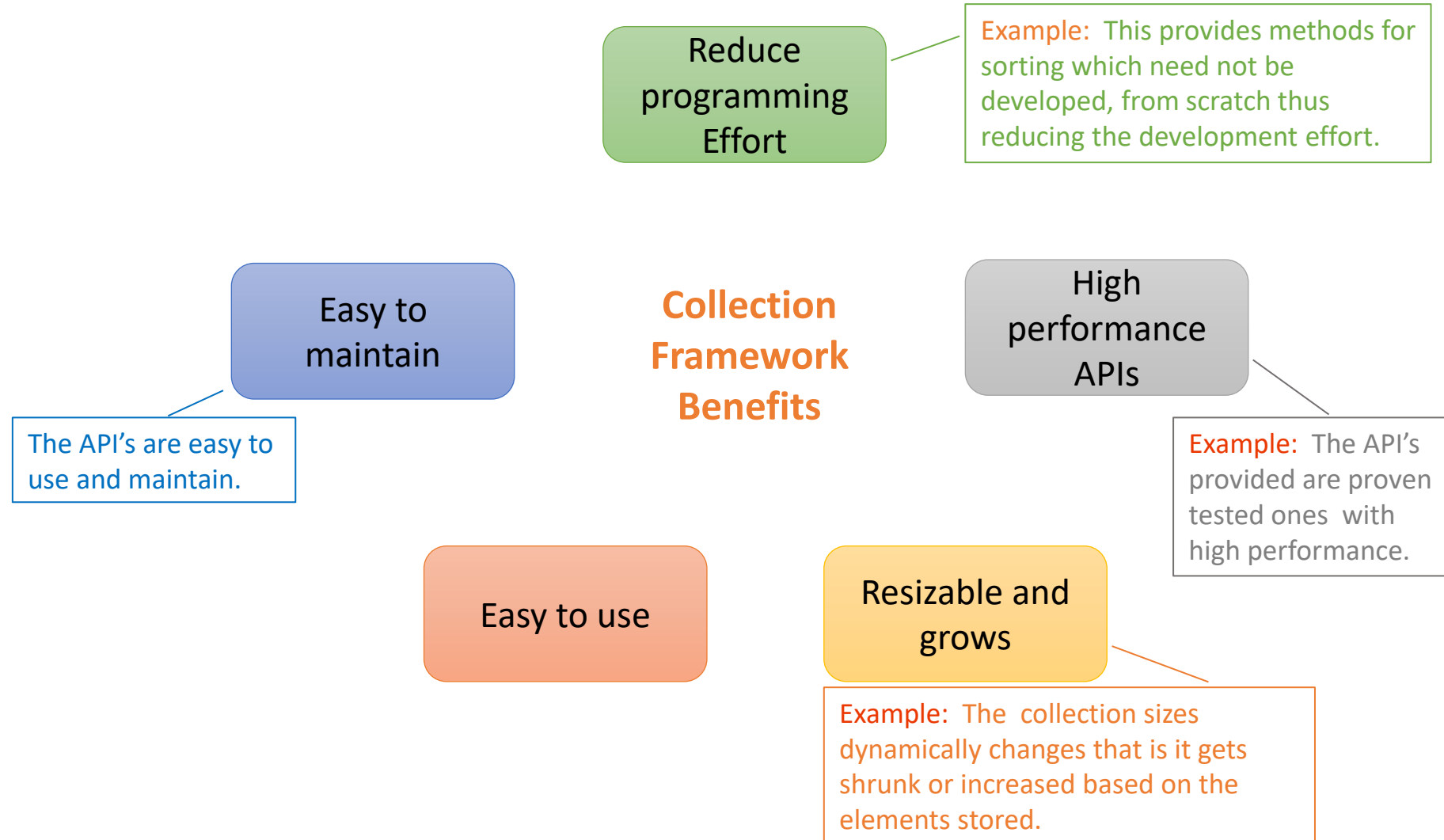
# Collections Framework

- 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.

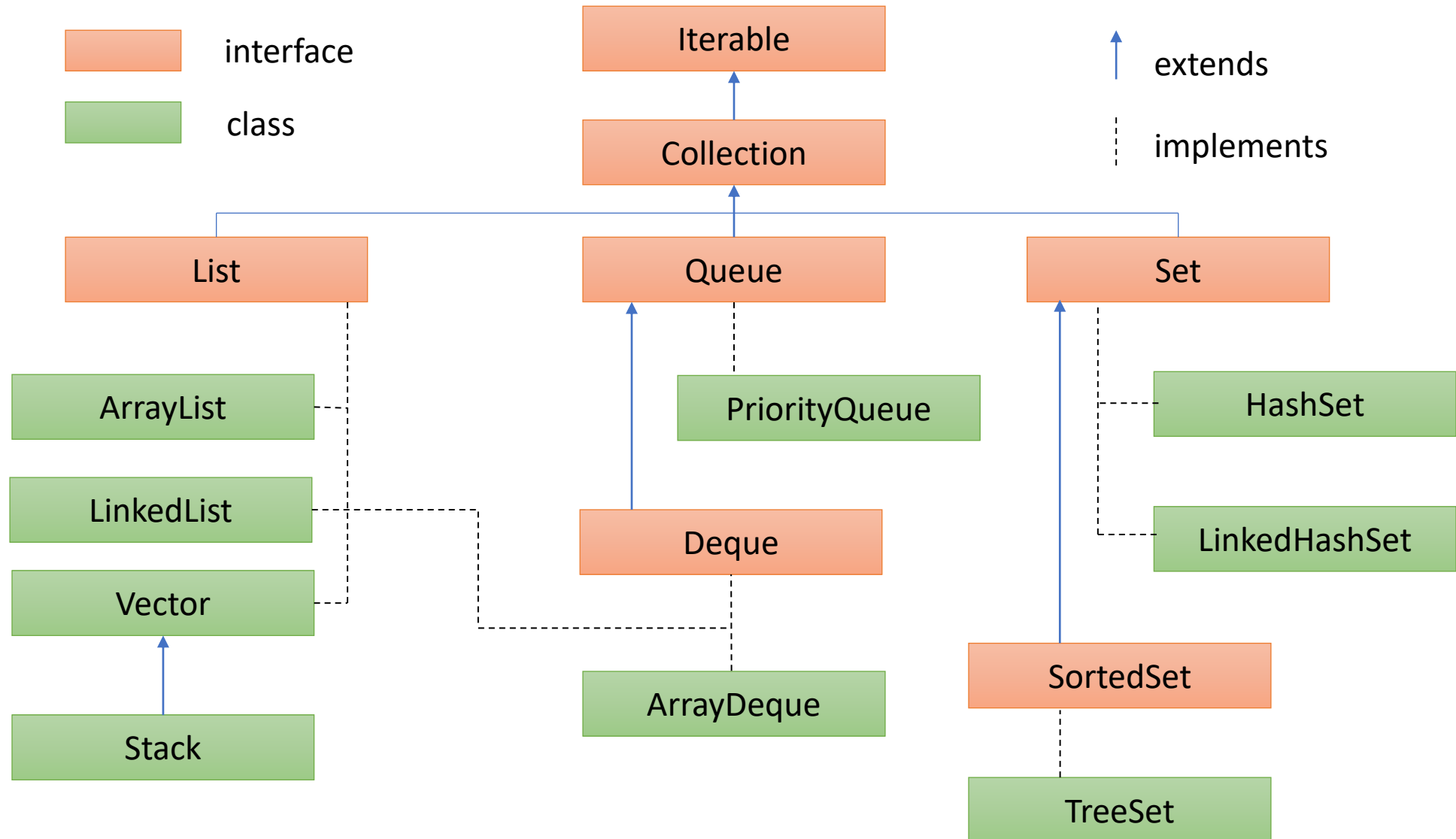
Collection framework contains,

- 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



# Collection Framework Components

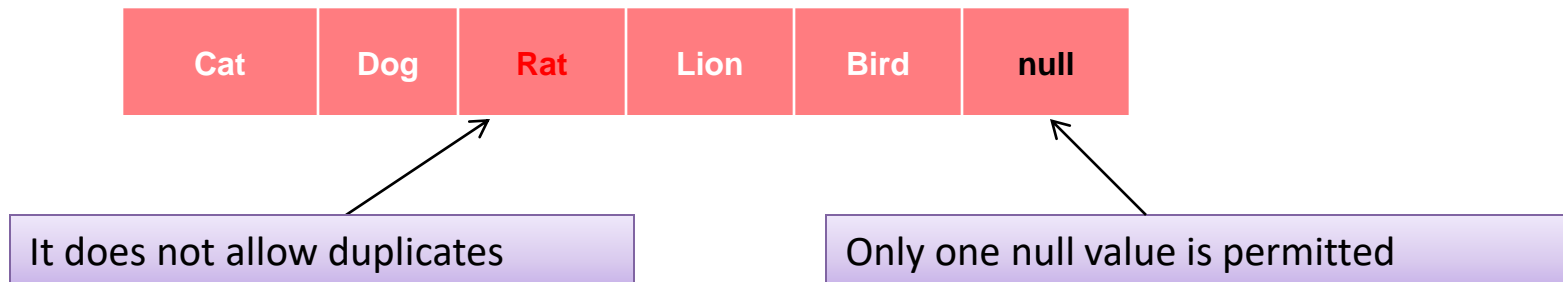


# Difference between List & Set

List :



Set:



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



# Collection Methods

Methods	Description
public boolean <b>contains</b> (Object element)	is used to search an element.
public boolean <b>containsAll</b> (Collection c)	is used to search the specified collection in this collection.
public Iterator <b>iterator</b> ()	returns an iterator.
public Object[] <b>toArray</b> ()	converts collection into array.
public boolean <b>isEmpty</b> ()	checks if collection is empty.
public boolean <b>equals</b> (Object element)	matches two collection.
public int <b>hashCode</b> ()	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 <b>add</b> (int index,Object element)	It is used to insert element into the invoking list at the index passed in the index.
boolean <b>addAll</b> (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 <b>get</b> (int index)	It is used to return the object stored at the specified index within the invoking collection.
object <b>set</b> (int index,Object element)	It is used to assign element to the location specified by index within the invoking list.
object <b>remove</b> (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
<b>ArrayList()</b>	It is used to build an empty array list.
<b>ArrayList(Collection c)</b>	It is used to build an array list that is initialized with the elements of the collection c.
<b>ArrayList(int capacity)</b>	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

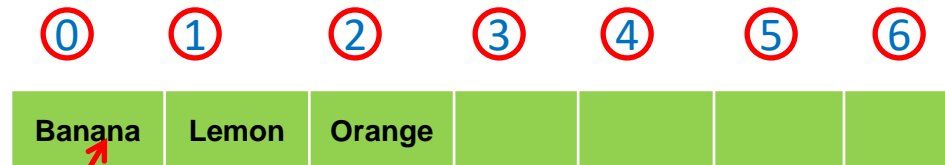
```
List myList=new ArrayList()
```

```
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 lesson 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 a 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 <b>clear()</b>	It is used to remove all of the elements from this set.
boolean <b>contains</b> (Object o)	It is used to return true if this set contains the specified element.
boolean <b>add</b> (Object o)	It is used to adds the specified element to this set if it is not already present.
boolean <b>isEmpty()</b>	It is used to return true if this set contains no elements.
boolean <b>remove</b> (Object o)	It is used to remove the specified element from this set if it is present.
Object <b>clone()</b>	It is used to return a shallow copy of this HashSet instance: the elements themselves are not cloned.
Iterator <b>iterator()</b>	It is used to return an iterator over the elements in this set.
int <b>size()</b>	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");
```

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

```
mySet.add("Grapes")
```

```
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.

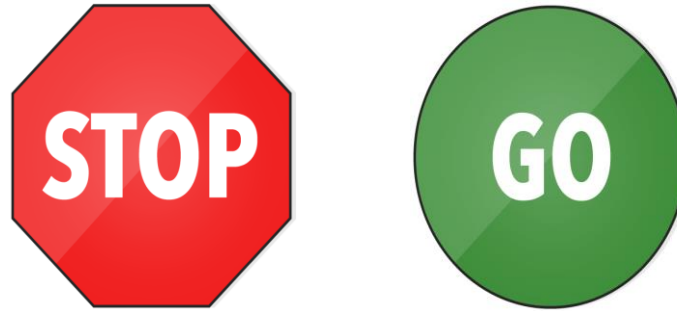
**Objective:** Let us learn how to create a *HashSet* and add elements into it.

**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

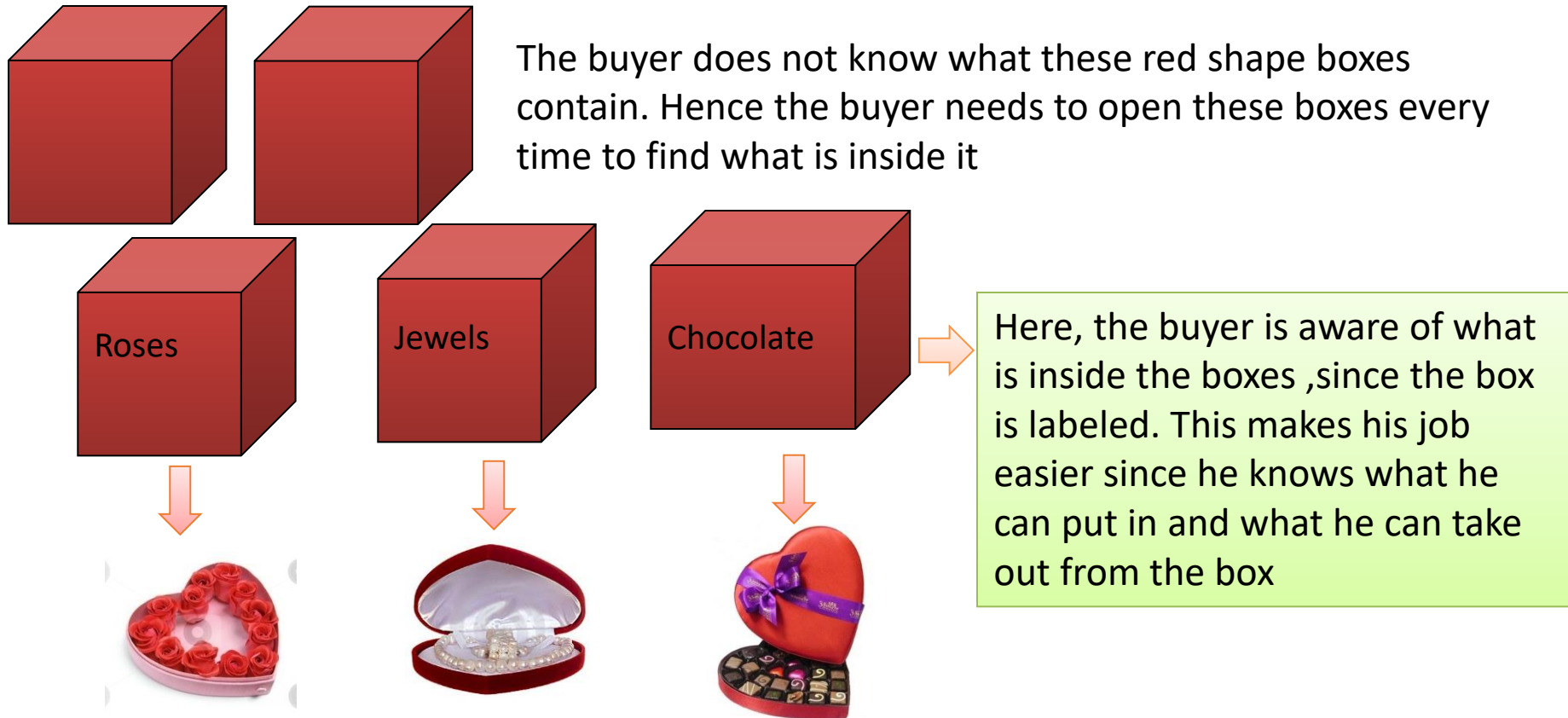


**Trainees to reflect the following topics before proceeding.**

- What method can be used to merge 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



# 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

```
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



# 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

```
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 <b>hasNext()</b>	true if there are more elements for the iterator.
Object <b>next()</b>	Returns the next object or elements.
void <b>remove()</b>	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 :

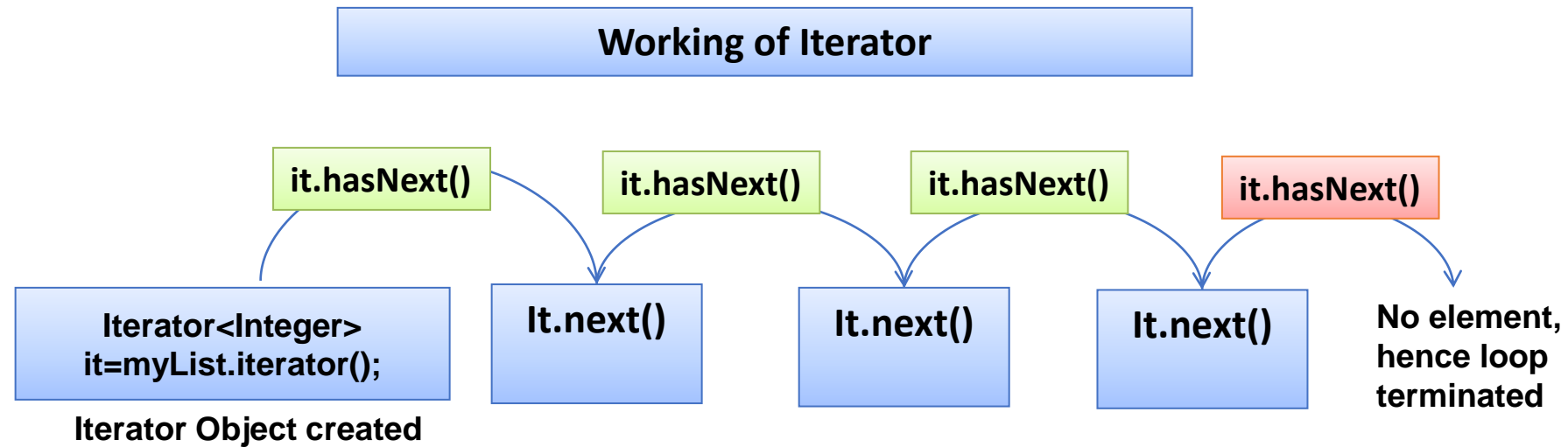
```
Iterator<type> iteratorName=collection.iterator();
```

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



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

```
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.

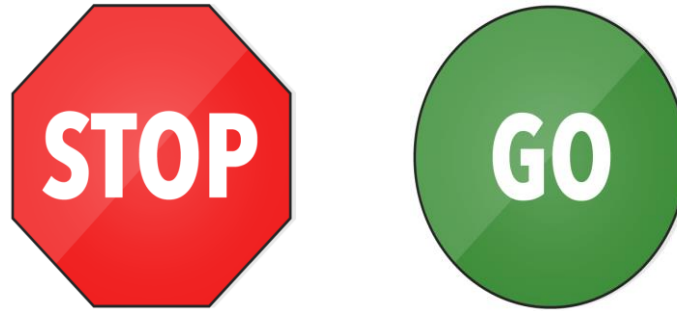
**Note :**The Iterator is also declared with generics to avoid type casting.

# 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.**

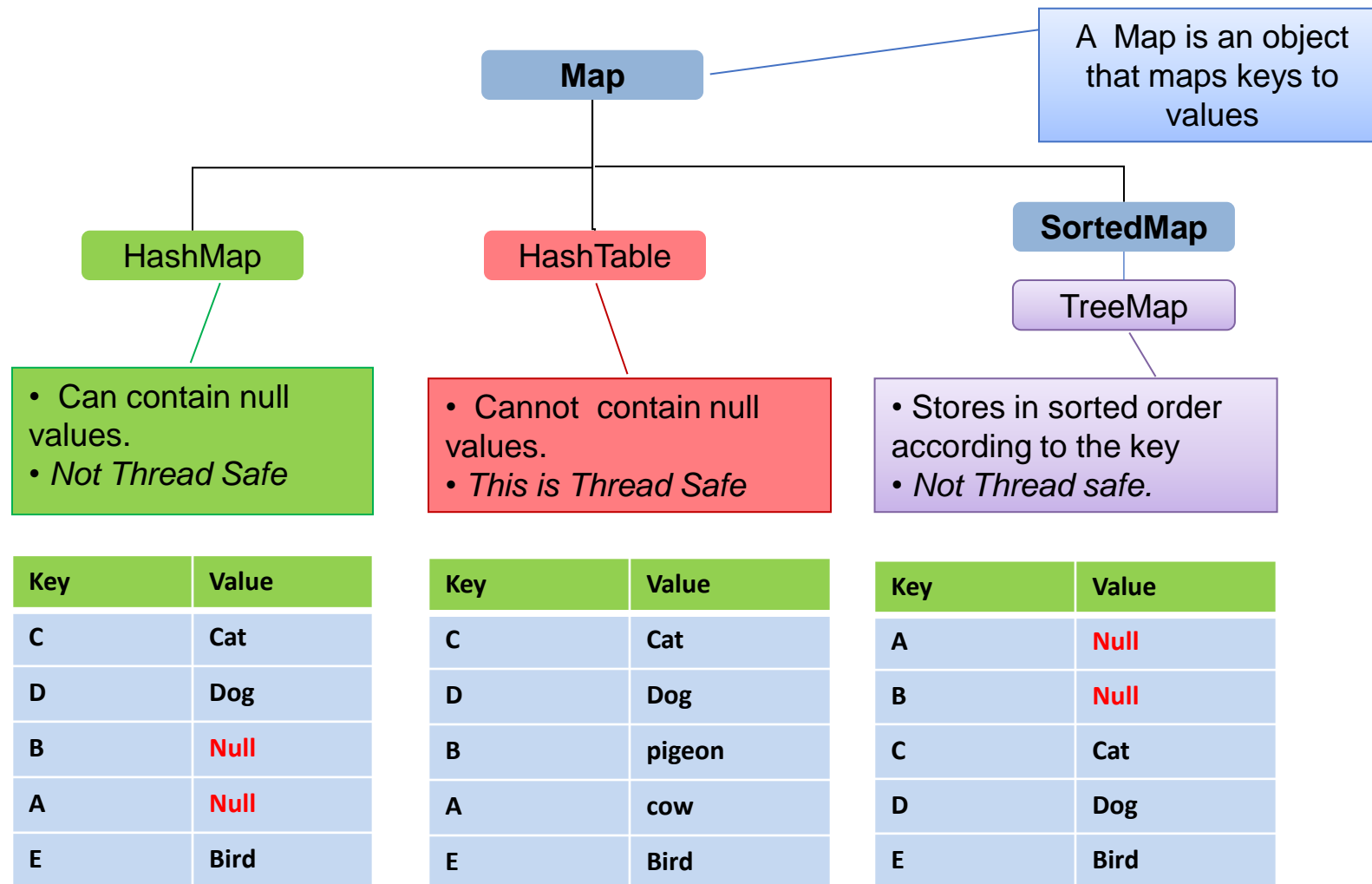
- How to make a collection accept only Integer objects?
- What are all the methods available to iterate through a Set?
- Can a ListIterator be used with a Set?
- Which iterator can be used to add elements to a list in between the iteration process?
- What may be the reason that for loop is not used to iterate through set?

# Maps

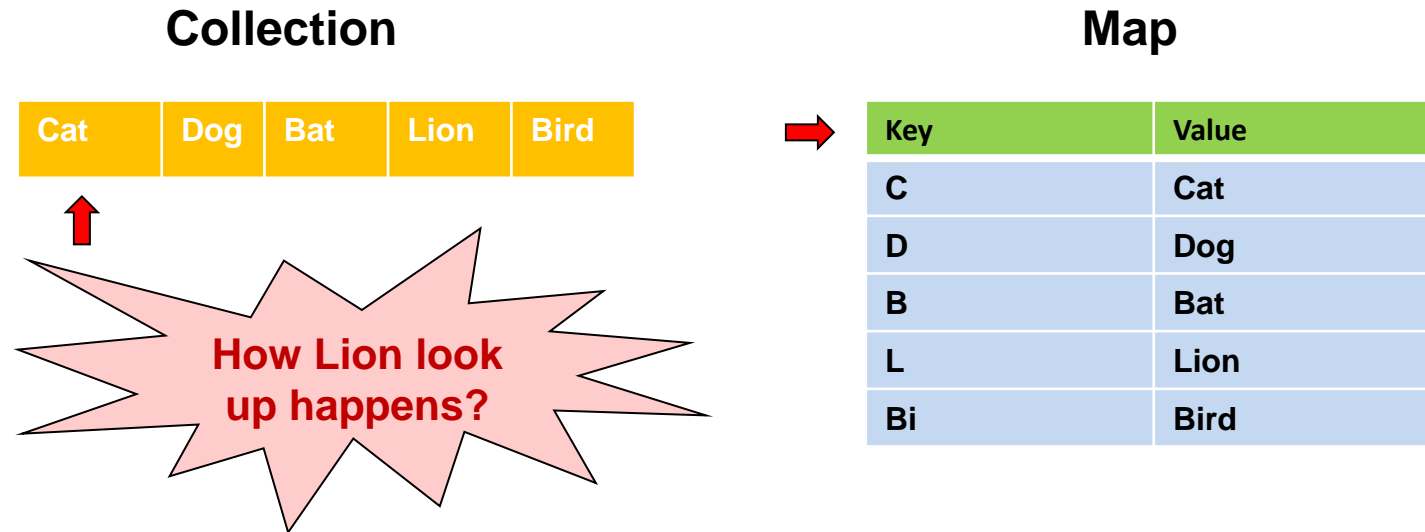
- Map stores Elements as key – value pairs.
- Each key maps to one value stored in the map.
- The values can be accessed by passing the key.
- The key should be unique used to identify the value.
- The key and value can only be objects.

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

# Map Interfaces



# Difference between Collection Interfaces and Map



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

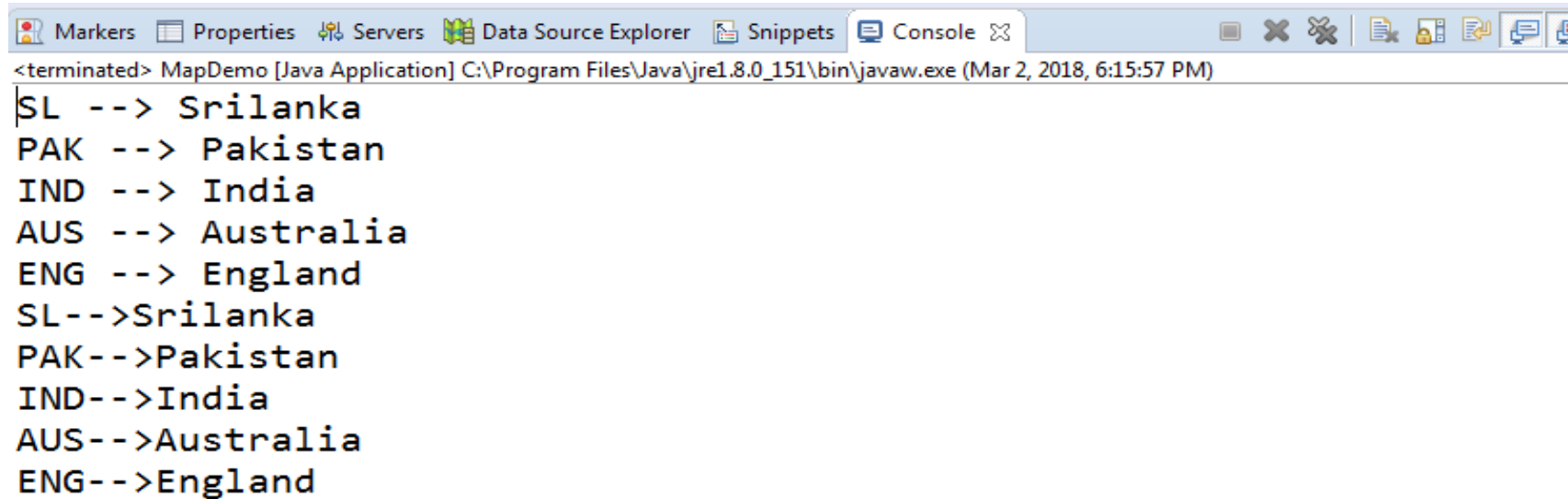
```
Map<Integer,String> myMap = new HashMap<Integer,String>();
```



# Example – HashMap

```
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



```
<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.

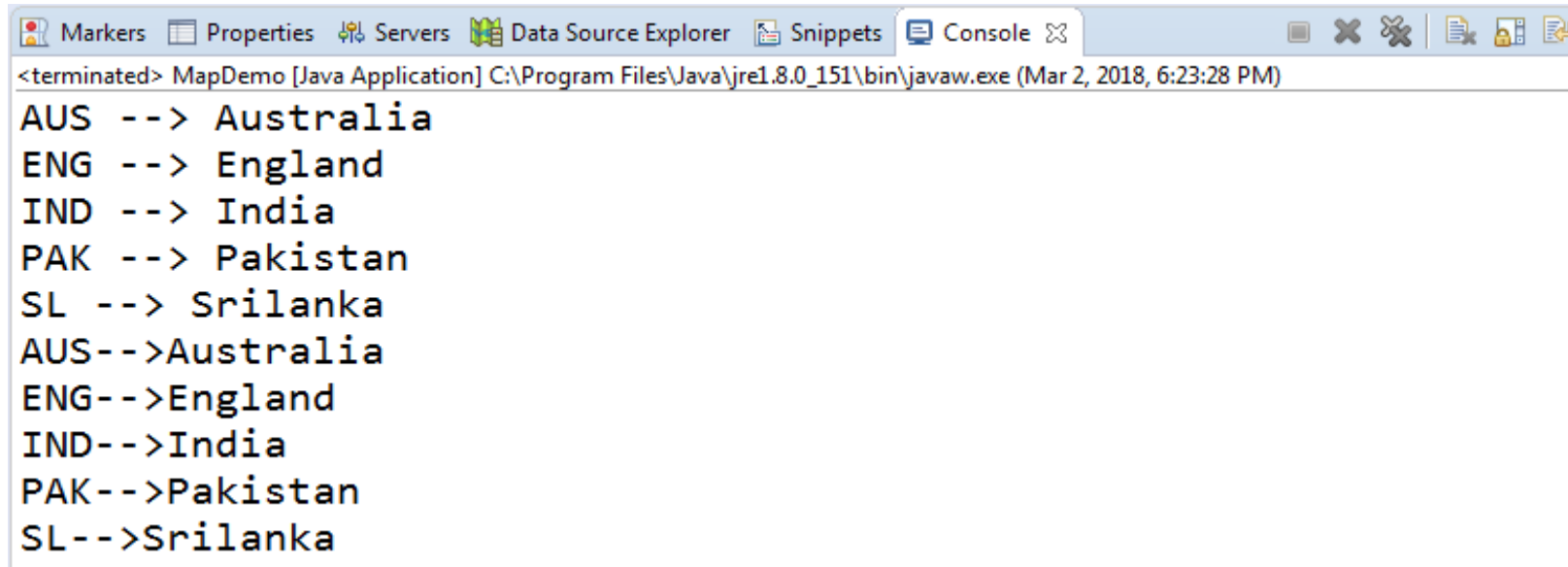
```
Map<String,String> studentMap=new HashMap<String, String>();
```



```
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**.

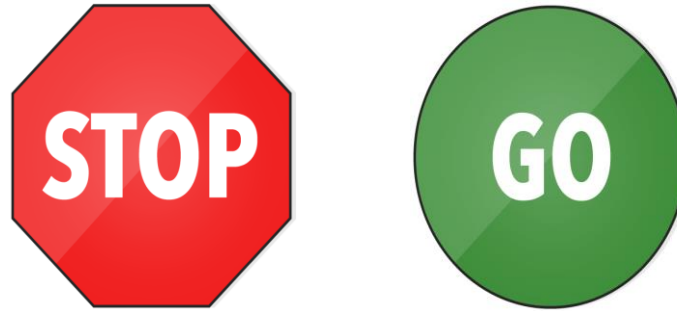
# Output - TreeMap



```
<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

```
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

The **printProductDetails()** method accepts list of students as argument.



```
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

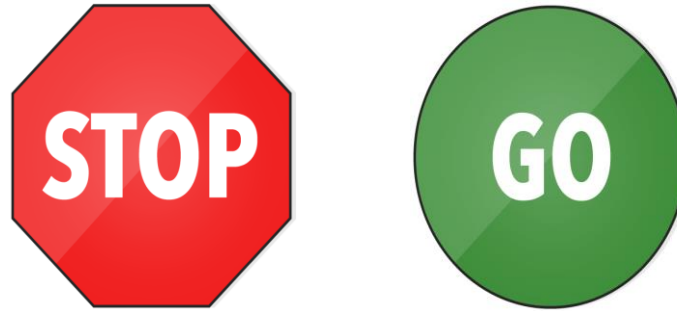
```
6 public class MainClass {  
7     public static void main(String[] args) {  
8         Product p1 = new Product(11,"Acer",2,12500L,"Flipkart");  
9         Product p2 = new Product(12,"Samsung",10,25600L,"Amazon");  
10        Product p3 = new Product(13,"Asus",3,14800L,"Flipkart");  
11        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**

# JAVA @11

Java Thread – Part I

# Objective

After completing this session you will be able to understand,

- What is Multithreading?
- Life cycle of a Thread
- Creating Thread
- Thread Scheduler
- Thread Interruption techniques
- Thread Priority



# Process vs Thread

Before we learnt about threads let's first understand the difference between a thread and a process.

***Process*** are executables which run in separate memory space.

***Threads*** are small process which run in shared memory space within a process.

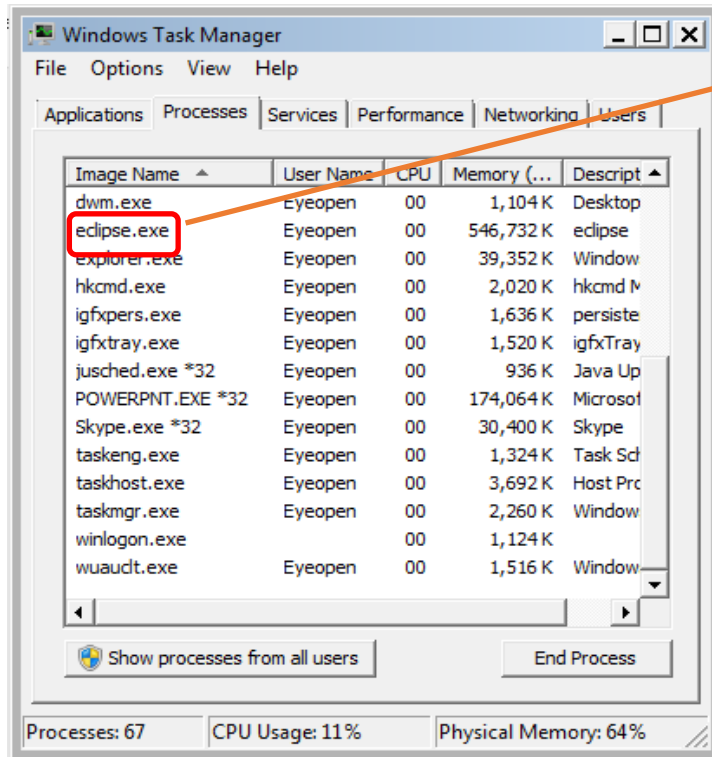
***So in a nutshell Process is a container for Threads, all thread runs inside the process.***



**Still confused let's look at an example to understand it better.**

# Process

Lets consider Eclipse IDE application to understand it better, what happens when you start an eclipse application.



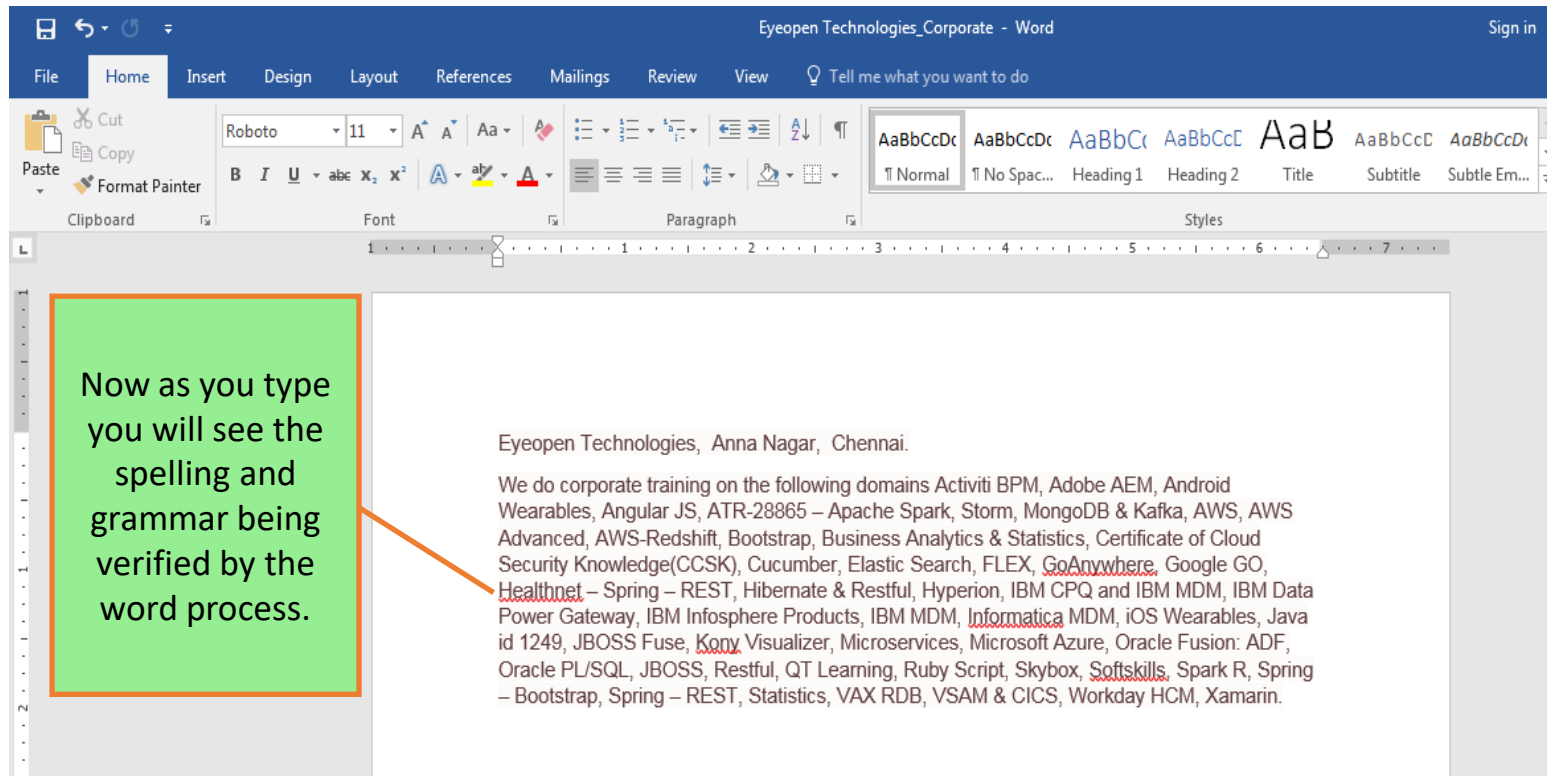
A process for eclipse ide application will be started.

Now lets see what happens when the user starts using word application.

# Thread

The spell check has been implemented as a *thread* within the worde.exe process which runs continuously and verifies what you type.

Word.exe is the process and spell check is a thread running inside the process.



# Real time scenario

**Ram** was developing an application where he has a requirement where user can register his profile in the application, assume registration has three steps

**Validate user details** – Takes 3 seconds for execution for each user.

**Validate user Citizenship** - Takes 4 seconds for execution for each user.

Now customer wants to complete the registration process is less than 5 seconds.



Guess how **Mr. Ram** would have achieved it?

He used **Multi Threading**.

Lets see what it is and how it can be implemented in this session.

# Multitasking

## What is Multitasking?

- Refers to a computer's ability to perform multiple jobs concurrently
- More than one program are running concurrently,

Example: In windows you can run Word, power point , media player at the same time. Yu will working on word and in parallel media player might play some music.

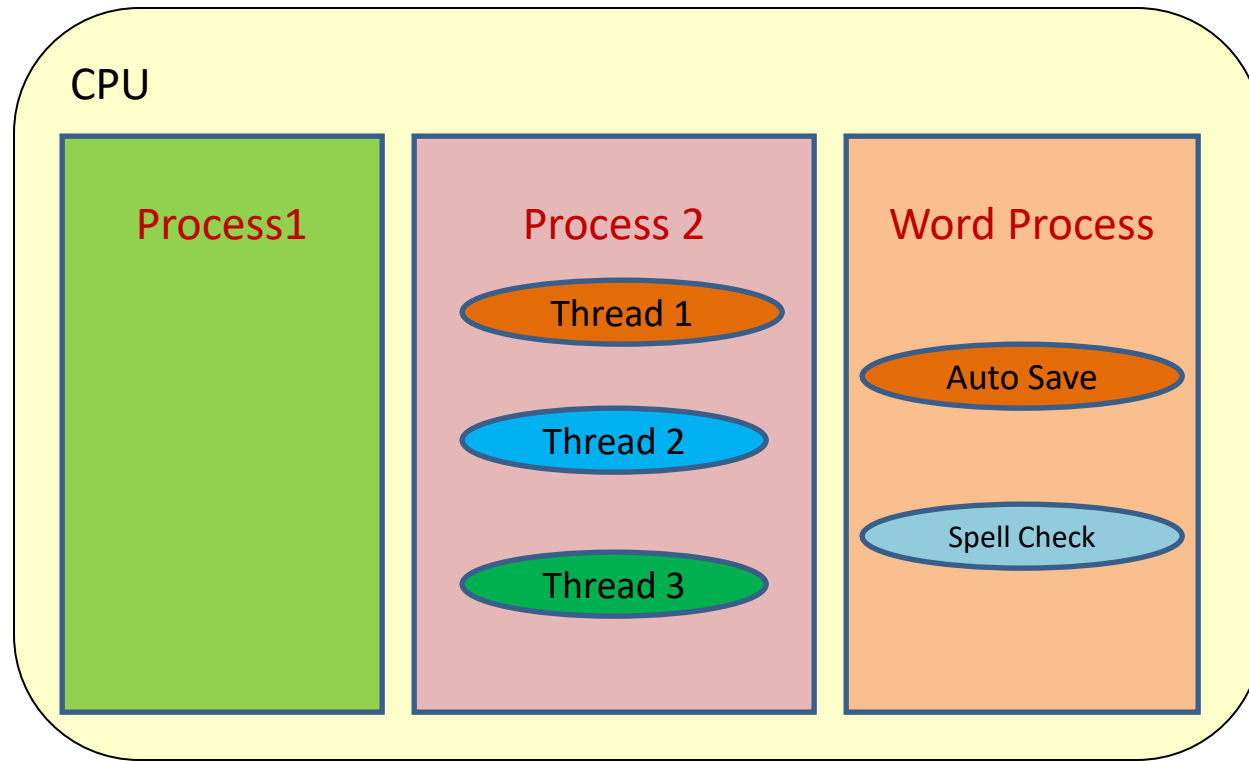
# Multithreading

## What is Multithreading?

- A thread is a single sequence of execution within a program/process.
- This refers to multiple threads of control within a single program.
- Each program can run multiple threads of control within it.

Example: Microsoft word process having multiple threads like spell check, auto save etc.

# System Illustration



The CPU is running three processes. Process 2 in turn has three threads running inside it.

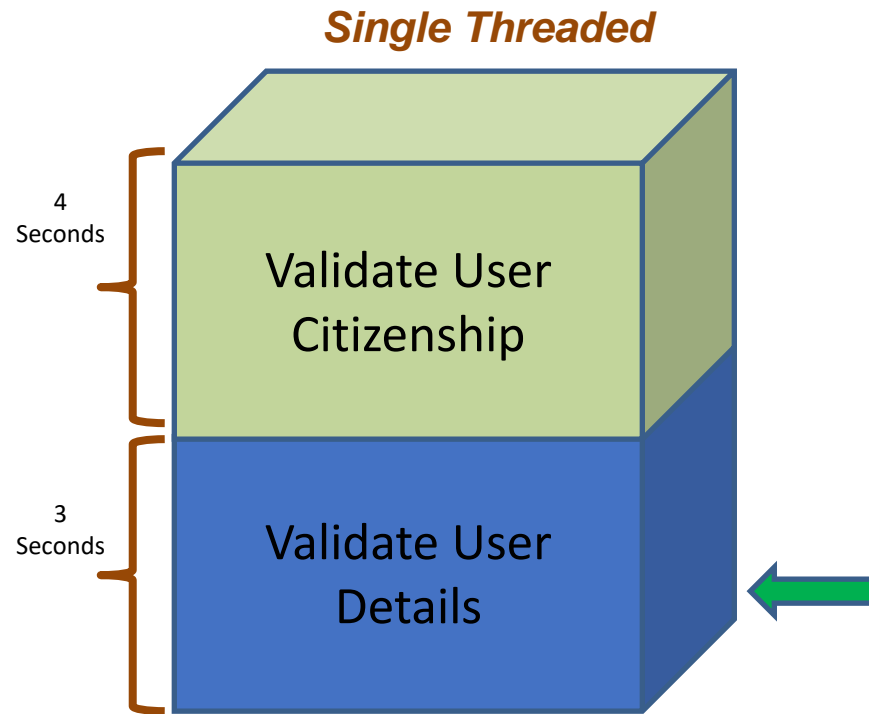
# Benefits of Multithreading

- To reduce response (execution) time of a process.
- Support Parallel Operation of Functions.
- Increase System Efficiency.
- Requires less overheads compared to Multitasking.



# How Ram Solved the Problem?

Ram implemented two threads for processing the **Validate user details** and **Validate user Citizenship**.



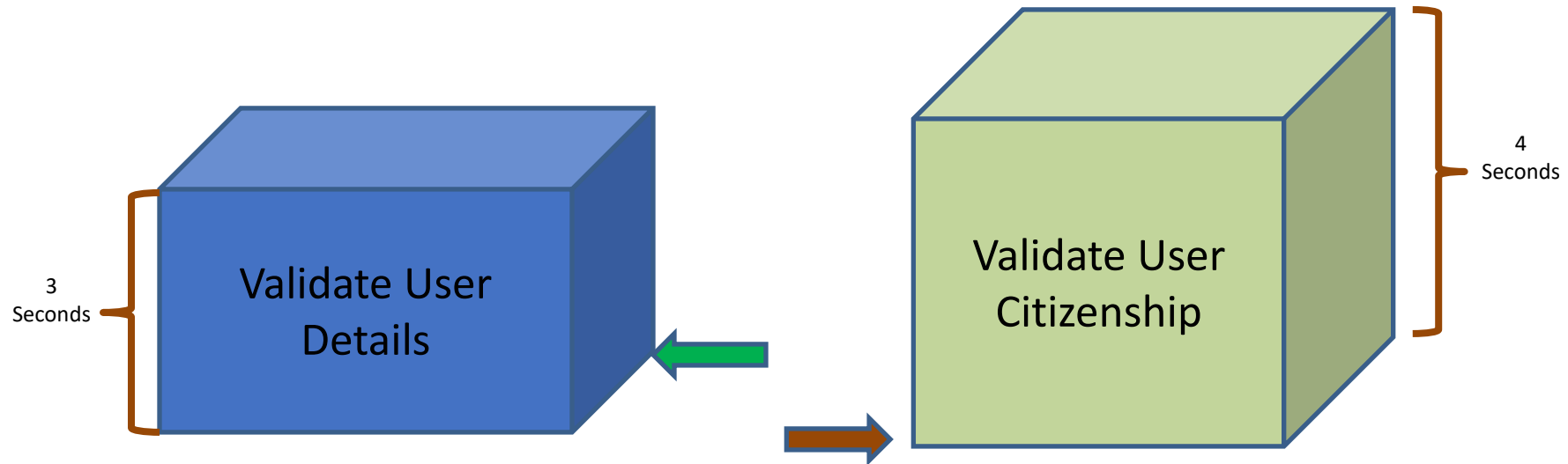
The thread takes 7 seconds for completing the registration process.

Let look how Ram implemented it.

# How Ram Solved the Problem?

Ram implemented two threads one thread for each method **Validate user details** and **Validate user Citizenship**.

*Multi Threaded*



The process will be completed in 4 seconds.  
Since both the threads works in parallel.

# What is an Application Thread?

*When we execute an application,*

- 1. The JVM creates a thread (T1) object which invokes the main() method and starts the application.*
- 2. The thread executes the statements of the program one by one (or) starts other threads.*
- 3. After executing all the statements, the method returns and the thread dies.*

The thread **T1** which is responsible for starting the application by invoking the main method is called “***Application Thread***”.

# Ways of Implementing Threads

**Method 1** : Extend ***Thread*** Class

**Method 2** : Implement ***Runnable*** Interface

# Using Thread in Java



**How To Use  
Threads in  
Java**

Using Thread  
class

java.lang.Thread class is used to construct and access individuals thread in a multi threaded application.

# Using Thread in Java

```
public class Thread extends Object implements Runnable {  
    public Thread();  
    public Thread(String name); // Thread name  
    public Thread(Runnable r); // Thread  $\Rightarrow$  r.run()  
    public Thread(Runnable r, String name);  
    public void run();  
    public void start(); // begin thread execution  
}
```

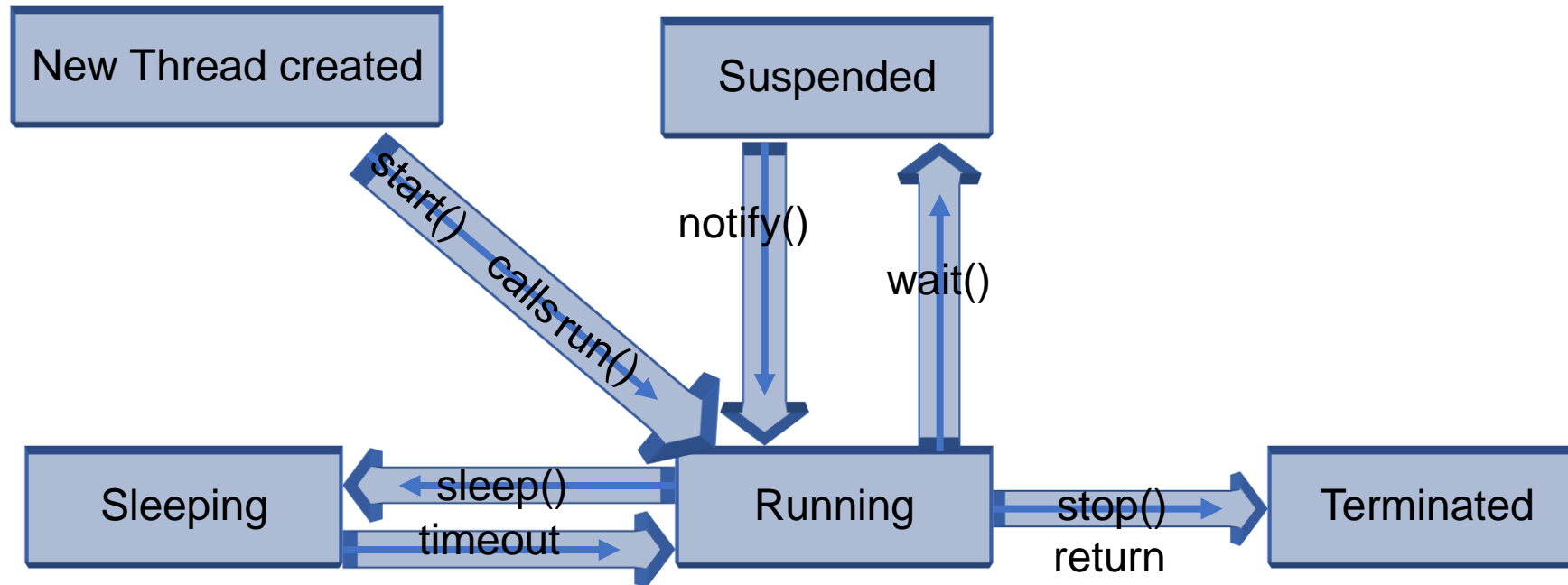
# Thread class methods

Method	Description
<b>void run()</b>	<ul style="list-style-type: none"><li>• The thread logic should be implemented in this method.</li><li>• This method should be overridden in all the Thread class.</li></ul>
<b>void start()</b>	Creates a new thread and invokes run method of the thread.
<b>getName()</b>	Returns the thread's name.
<b>int getPriority()</b>	Returns the thread's priority
<b>boolean isAlive()</b>	Tests if the thread is alive.

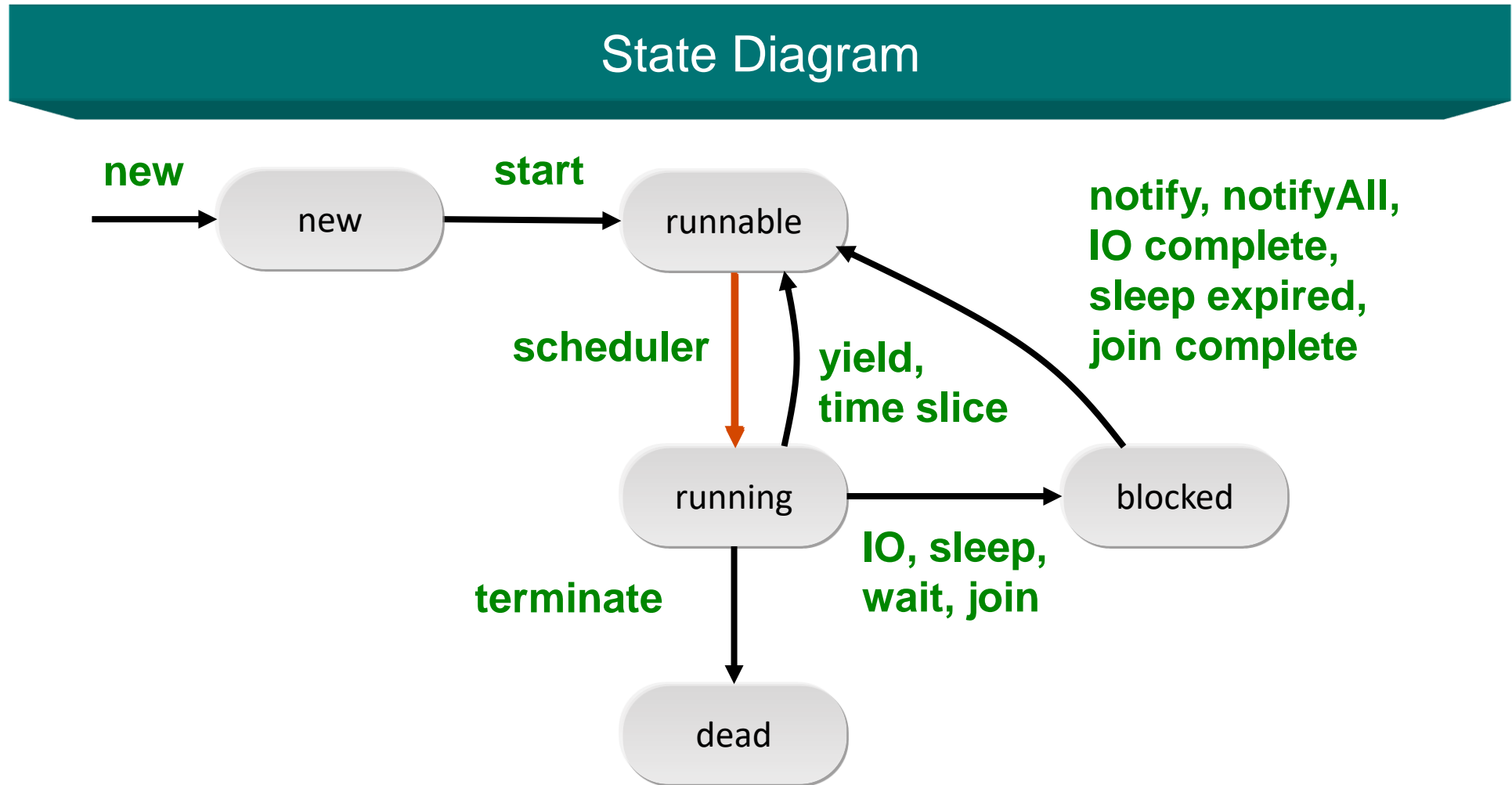
Method	Description
<b>Thread currentThread()</b>	Return a reference to the currently executing thread object
<b>setName(String name)</b>	Sets a name for the thread to be equal to the argument name.
<b>setPriority (int newPriority)</b>	Changes the priority of the thread
<b>static void sleep(long millis)</b>	Causes the currently executing thread to sleep (temporarily cease execution) for the specified number of milliseconds.
<b>void yield()</b>	Causes the currently executing thread object to temporarily pause and allow other threads of same priority to execute



# Lifecycle of Thread



# Java thread states



# Example – How to develop a thread?

In this demo we will learn how to

1. Create a Thread by extending the Thread class
2. Override the run method
3. Create a thread object and start the thread using the start method
4. How the following methods operates
  - a. getName()
  - b. setName()
  - c. setPriority()
  - d. getPriority()
  - e. sleep()

## **Components to be developed,**

1. ThreadEX – The Thread class should loop and print values 0...4.
2. ThreadExMain – The main class to execute the Threads

# Solution - Thread

```
public class ThreadEx extends Thread {  
    int i = 0;
```

Class ThreadEx extends Thread class and becomes a Thread class

```
    public ThreadEx(String name) {  
        this.setName(name);
```

Sets the name of the Thread using setName method()

```
    }
```

Overrides the run() method

Prints the name of the current Thread using the getName() method

```
    public void run() {  
        for (i = 0; i < 5; i++) {  
            System.out.println("Printing from " + Thread.currentThread().getName() +  
                               " the value of i :: " + i);  
            try {  
                Thread.sleep(300);  
            } catch (InterruptedException ex) {  
                ex.printStackTrace();  
            }  
        }  
    }  
}
```

Makes the Thread sleep for 300 milliseconds. Invoking the sleep may cause an InterruptedException to be thrown which should be handlers.

# Solution - Thread

```
public class ThreadExMain {  
    public static void main(String args[]) throws InterruptedException {  
        Thread t = Thread.currentThread();  
        System.out.println("The Current Thread is " + t.getName());  
        System.out.println("The Current Thread Priority is " + t.getPriority());  
        t.setName("myThread");  
        t.setPriority(6);  
        System.out.println("The Current Thread is " + t.getName());  
        System.out.println("The Current Thread Priority is " + t.getPriority());  
        ThreadEx ex1 = new ThreadEx("first");  
        ThreadEx ex2 = new ThreadEx("second");  
        System.out.println("The Thread Priority of ex1 is "  
            + ex1.getPriority());  
        System.out.println("The Thread Priority of ex2 is "  
            + ex2.getPriority());  
        ex1.start();  
        ex2.start();  
        for (int i = 0; i < 5; i++) {  
            System.out.println("Printing from : "  
                + Thread.currentThread().getName() + " :: " + i);  
            Thread.sleep(100);  
        }  
    }  
}
```

Gets the priority and name of the main thread

Sets the priority and name of the main thread

Creates two new Thread objects with name "first" and "second"

Starts the Thread by invoking the start () which will invoke the run() method of the Thread class

# Solution - Thread

- Execute the main class – ***ThreadExMain***
- The output will be something as shown below – Output may vary for each execution since Thread execution is based on the underlying operating system.

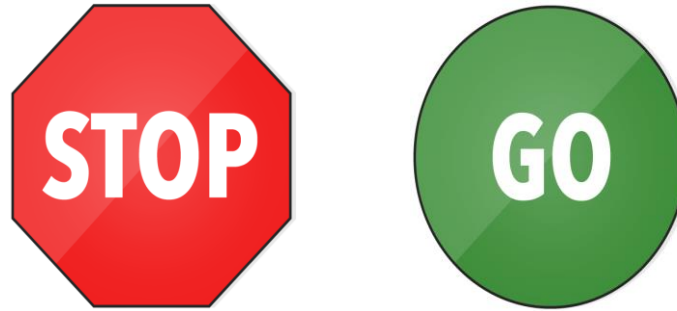
```
The Current Thread is main
The Current Thread Priority is 5
The Current Thread is myThread
The Current Thread Priority is 6
The Thread Priority of ex1 is 6
The Thread Priority of ex2 is 6
Printing from : myThread :: 0
Printing from first the value of i :: 0
Printing from second the value of i :: 0
Printing from : myThread :: 1
Printing from : myThread :: 2
Printing from first the value of i :: 1
Printing from second the value of i :: 1
Printing from : myThread :: 3
Printing from : myThread :: 4
Printing from second the value of i :: 2
Printing from first the value of i :: 2
Printing from second the value of i :: 3
Printing from first the value of i :: 3
Printing from second the value of i :: 4
Printing from first the value of i :: 4
```



Different output  
displayed during  
different run

```
The Current Thread is main
The Current Thread Priority is 5
The Current Thread is myThread
The Current Thread Priority is 6
The Thread Priority of ex1 is 6
The Thread Priority of ex2 is 6
Printing from : myThread :: 0
Printing from first the value of i :: 0
Printing from second the value of i :: 0
Printing from : myThread :: 1
Printing from : myThread :: 2
Printing from second the value of i :: 1
Printing from first the value of i :: 1
Printing from : myThread :: 3
Printing from : myThread :: 4
Printing from second the value of i :: 2
Printing from first the value of i :: 2
Printing from first the value of i :: 3
Printing from second the value of i :: 3
Printing from first the value of i :: 4
Printing from second the value of i :: 4
```

# Time To Reflect



**Trainees to reflect the following topics before proceeding.**

- What is Multithreading?
- How can we implement threading in Java API?
- Explain about Thread Lifecycle methods in Java API?

Thank you

*You have successfully completed*  
**Thread-1**



# JAVA @11

Java Thread – Part II

# Objective

After completing this session you will be able to understand,

- How to create thread by implementing runnable interface?
- Comparison between the two method of thread creation
- Learn about thread join method

# Runnable Interface

- Threads can be created by implementing the *Runnable* interface and overriding the run method.
- *Runnable* interface contains only one method – the *run()* method which should be overridden to start a new Thread.

# Implementing Runnable Interface

**Step 1:** Create a class ***ThreadRunnableEx*** by implementing the *Runnable* interface

**Step 2:** Override the ***run()*** method , the logic each thread should execute.

**Step 3:** Create an ***main*** method which can start the thread.

**Step 4:** Inside the main method create an object of the ***ThreadRunnableEx*** class.

**Step 5:** Inside the main method create a *Thread* object using the ***ThreadRunnableEx***. This is done by passing the ***ThreadRunnableEx*** object to the constructor of the Thread class.

**Step 6:** Invoke the start method on the Thread object which in turn calls the *run()* method of the ***ThreadRunnableEx*** class and starts a thread.

# Example – Runnable Interface

**In this demo we will learn how to**

1. Create a Thread by using Runnable interface.
2. Implement the run method.
3. Create a thread object using the Runnable interface,

**Components to be developed,**

1. **ThreadRunnableEx** – Thread implemented using Runnable interface. Each Thread should loop and print values 0...9
2. **RunnableExMain** – The main class to start the Threads

# Solution – Runnable Interface

Create a class named ***ThreadRunnableEx*** implementing the *Runnable* interface

```
public class ThreadRunnableEx implements Runnable {  
  
    public void run() {  
        for (int i = 0; i < 10; i++) {  
            System.out.println("Printing from "  
                               + Thread.currentThread().getName()  
                               + " , the value of i  :" + i);  
            try {  
                Thread.sleep(100);  
            } catch (InterruptedException ex) {  
                ex.printStackTrace();  
            }  
        }  
    }  
}
```

Class ThreadRunnableEx implements *Runnable* interface override the run method to print values between 0..9.

# Solution – Runnable Interface

Create class *RunnableExMain* with a main method which will start the threads.

```
public class RunnableExMain {  
    public static void main(String args[]) throws InterruptedException {  
        ThreadRunnableEx r1 = new ThreadRunnableEx();  
        ThreadRunnableEx r2 = new ThreadRunnableEx();  
        Thread t1 = new Thread(r1, "first");  
        Thread t2 = new Thread(r2, "second");  
        t1.start();  
        t2.start();  
        for (int i = 0; i < 10; i++) {  
            System.out.println(Thread.currentThread().getName() + " :: " + i);  
            Thread.sleep(100);  
        }  
    }  
}
```

This creates two thread to be run

Create the Thread objects using the Runnable objects

Start the two Threads

# Execute the Runnable

- Run the main class – ***ThreadExMain***
- The output will be something as shown below.

Output may vary for each execution since Thread execution depends upon the operating system.

```
main :: 0
Printing from second , the value of i :0
Printing from first , the value of i :0
Printing from first , the value of i :1
main :: 1
Printing from second , the value of i :1
Printing from second , the value of i :2
main :: 2
Printing from first , the value of i :2
Printing from second , the value of i :3
Printing from first , the value of i :3
main :: 3
Printing from first , the value of i :4
Printing from second , the value of i :4
main :: 4
```

Different output  
displayed during  
different run

```
main :: 0
Printing from second , the value of i :0
Printing from first , the value of i :0
main :: 1
Printing from second , the value of i :1
Printing from first , the value of i :1
main :: 2
Printing from second , the value of i :2
Printing from first , the value of i :2
main :: 3
Printing from second , the value of i :3
Printing from first , the value of i :3
main :: 4
Printing from second , the value of i :4
Printing from first , the value of i :4
```



# Runnable vs Thread



Which Method is better? Why?

Implementing Runnable interface is recommended.

## Why Runnable is better?

- If we inherit the *Thread* class for creating threads we cannot extend any other class. *Since Java does not support multiple inheritance*
- By implementing the *Runnable* interface we can make the thread class extend other classes.

# Ram faces another problem

In the previous examples we can see that the main Thread (***Application Thread***) exits before the other child Threads which was originated by it completes execution. This may not be desirable in some situations.

**Example:** Assume tax needs to be calculated for 10 employees using **calculateTax()** method, Tim the programmer decides to speedup the process by spawning 10 thread one thread per employee. Also Tim needs to ensure that the program should exit only after all the ten threads complete the tax calculation.

How can this be solved?  
Tim used Thread **Join** Method.

# What is join method?

join method in Thread class,

- ***join()*** is used to ensure that the application main thread will complete only after all the thread spawned by it inside the process completes execution.
- In other words, one thread can wait for another to complete using the join() method.
- Invoking join() guarantees that the method will not return until the threads started from it returns and join.

# Example – join()

In this demo we will see how a method works with and without join .

- Step 1: We will create a Runnable class **JoinEx** and a main class **JoinExMain**
- Step 2: First we will create the main method without join() and look at the output
- Step 3: Then will inject join() to the main method and see how join changes the output.

# Solution – join()

```
public class JoinEx implements Runnable {  
    public void run() {  
        for (int i = 0; i < 5; i++) {  
            System.out.println("Printing from " +  
                               Thread.currentThread().getName() +  
                               " , the value of i : " + i);  
            try {  
                Thread.sleep(100);  
            } catch (InterruptedException ex) {  
                ex.printStackTrace();  
            }  
        }  
    }  
}
```

Main method without join, create two threads and start it.

```
public class JoinExMain {  
    public static void main(String args[]) throws InterruptedException {  
        JoinEx r1 = new JoinEx();  
        JoinEx r2 = new JoinEx();  
        Thread t1 = new Thread(r1, "first");  
        Thread t2 = new Thread(r2, "second");  
        t1.start();  
        t2.start();  
        System.out.println("Main Thread Ends Here ");  
    }  
}
```

# Output – without join()

The main method ends before the completion of the other threads



Main Thread Ends Here

```
Printing from second , the value of i :0
Printing from first , the value of i :0
Printing from second , the value of i :1
Printing from first , the value of i :1
Printing from second , the value of i :2
Printing from first , the value of i :2
Printing from first , the value of i :3
Printing from second , the value of i :3
Printing from first , the value of i :4
Printing from second , the value of i :4
```

# Add the join()

```
public class JoinExMain {  
    public static void main(String args[]) throws InterruptedException {  
        JoinEx r1 = new JoinEx();  
        JoinEx r2 = new JoinEx();  
        Thread t1 = new Thread(r1, "first");  
        Thread t2 = new Thread(r2, "second");  
        t1.start();  
        t2.start();  
        t1.join();  
        t2.join();  
        System.out.println("Main Thread Ends Here ");  
    }  
}
```

The threads join method is triggered.



# Output – with join()

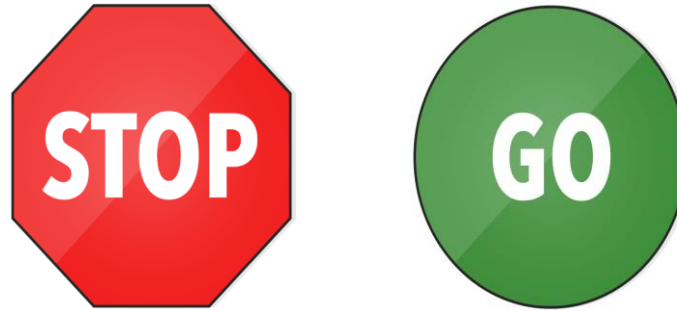
```
Printing from second , the value of i :0  
Printing from first , the value of i :0  
Printing from second , the value of i :1  
Printing from first , the value of i :1  
Printing from second , the value of i :2  
Printing from first , the value of i :2  
Printing from second , the value of i :3  
Printing from first , the value of i :3  
Printing from second , the value of i :4  
Printing from first , the value of i :4
```

Main Thread Ends Here



The main method ends after the completion of the other threads

# Time To Reflect



Trainees to reflect the following topics before proceeding.

- What is Runnable interface?
- Which method is better either Thread or Runnable? Why?
- What is Thread join() method?

Thank you

*You have successfully completed*  
**Thread-II**

# JAVA @11

Java Thread – Part III

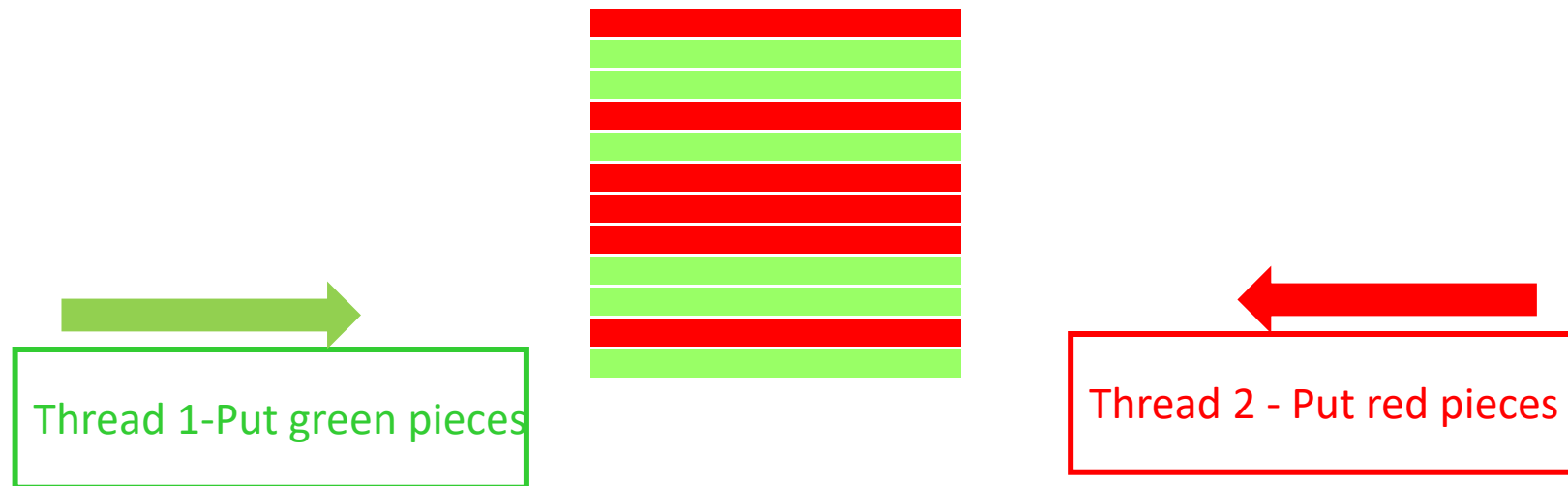
# Objective

After completing this session you will be able to understand,

- What is synchronization?
- Various thread method wait, notify and notifyAll

# What is race condition?

**Scenario:** Consider a scenario in which you want to keep green blocks sequentially at the bottom and red blocks be placed on top of the green blocks. You have developed a program in such a way that two threads one for placing the green blocks and other for red blocks. Lets see how the program places the blocks.



You couldn't get your expected output. The red blocks and greens blocks where placed randomly. This is what we call a **Race Condition**

# When does race condition occur?

- A race condition occurs when two or more threads are able to access shared data and they try to change it at the same time.
- Both of them try to modify the data at the same time that is both are racing to get access/change the data
- Since the thread scheduling is system dependent we cannot predict a reliable behavior of the application.

**So what can be done to make only one Thread access a shared resource at a time ?**

**The solution is **synchronization**.**

# What is synchronization?

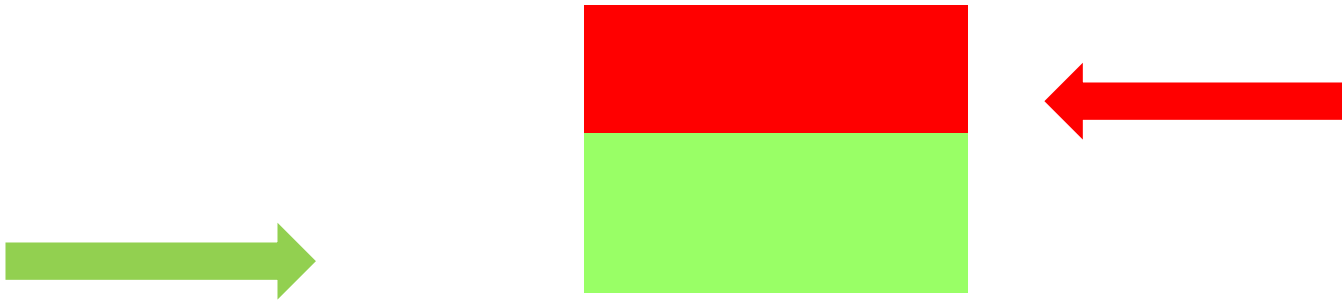
***Synchronizing*** threads means access to shared data (or) method execution logic in a multithreaded application is controlled in such a way that only one thread can access the method/shared data at a time. The other threads will wait till the thread releases the control. The shared data is considered to be ***Thread Safe***.

Java uses the concept of ***monitors*** to implement synchronization.



# Problem solved using Synchronization?

Lets see how synchronization can solve our problem in placing the colored blocks



Here the red thread waits till the green thread completes the job.  
This is achieved using **Synchronization**.

# Thread Monitor

- A **monitor** is like a room of a building that can be occupied by only one thread at a time.
- The room contains a data or logic.
- From the time a thread enters this room to the time it leaves, it has exclusive access to any data/logic in the room.
  - This is done by gaining control on the monitor which is nothing but the room.
- Entering the special room inside the building is called "**acquiring the monitor**".
- Occupying the room is called "**owning the monitor**" or exclusive access to the data.
- Leaving the room is called "**releasing the monitor**".

# Synchronization methods

***Synchronization*** is achieved by using one of the two methods,

**Method 1:** Synchronized blocks

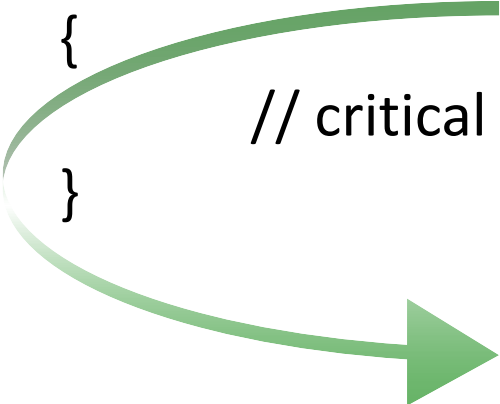
**Method 2:** Synchronized methods

# Method 1: synchronized method

- ***Synchronized methods*** can be created by using the keyword ***synchronized*** for defining the method.
- The code which should be thread safe is written inside this method.
- During the execution of a synchronized method, the thread holds the monitor of that method's object (or) if the method is static it holds the monitor of that method's class.
- If another thread is executing the synchronized method, your thread is blocked until that thread releases the monitor.

# Synchronizing a Method

```
public synchronized void updateRecord()  
{  
    // critical code goes here ...  
}
```



Only one thread will be inside the body of the updateRecord() method. The second call will be blocked until the first call returns or wait() is called inside the synchronized method.

# Synchronizing an Object

```
public void updateRecord()
{
    synchronized (this)
    {
        // critical code goes here ...
    }
}
```

# Example – Synchronize method

***Demonstration:*** In this demo we will see how to implement synchronization and how it works?. We will create a small application which prints a text message on the console.

This demo will also help you to understand the behavior of synchronized and unsynchronized methods.

# Example – Synchronize method

We will develop a program to spawn three threads, each thread should accept two string arguments and print the String message.

The following two messages should be printed by each thread,

Thread 1 – “Hello..” and “There”

Thread 2 – “How” and “are you”

Thread 3 – “Thank you,” and “very much”

## Classes

1. **PrinterThread** : Thread class used for printing the String.
2. **StringPrinter** : This class is used by the Thread class to print the Strings, contains a method which prints the strings.
3. **SyncExMain** : Main method to initiate the threads.

## Output Expected

```
Hello..There
```

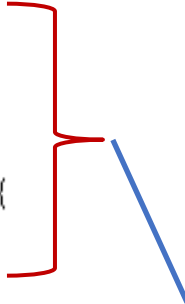
```
How are you
```

```
Thank you,very much
```



# Solution – Synchronize method

```
public class StringPrinter {  
    static void printStrings(String stringA, String stringB) {  
        System.out.print(stringA);  
        try {  
            Thread.sleep(500);  
        } catch (InterruptedException ie) {  
            ie.printStackTrace();  
        }  
        System.out.println(stringB);  
    }  
}
```



Methods to print the strings.

NOTE:

- The strings are printed one after the other with a pause. This has been implemented using the sleep method.
- The first message should be printed without a new line.
- Second message should be printed with a new line.


# Solution – Synchronize method

```
public class PrinterThread implements Runnable {  
    Thread t;  
    String stringA;  
    String stringB;  
  
    public PrinterThread(String stringA, String stringB) {  
        this.stringA = stringA;  
        this.stringB = stringB;  
        Thread t = new Thread(this);  
        t.start();  
    }  
  
    public void run() {  
        StringPrinter.printStrings(stringA, stringB);  
    }  
}
```

} Creates a thread object on “this” Runnable object and starts the thread.

# Solution – Synchronize method

```
public class SyncExMain {  
    public static void main(String args[]) {  
        new PrinterThread("Hello..", "There");  
        new PrinterThread("How", " are you");  
        new PrinterThread("Thank you,", "very much");  
    }  
}
```



Create three threads and pass the appropriate String messages.

# Output – without sync method

Execute the class, the output could be displayed as below.

```
Thank you,HowHello..very much  
are you  
There
```

This is the output we get, not the expected one. Note the order in which the messages are printed could be different depending upon the OS scheduler.

Inference:

Each thread overruns the other thread resulting in the message being jumbled.

# add - synchronization

Now lets control the threads by using *synchronized* key word.

Make the *printStrings()* method of the *StringPrinter* class *synchronized*.

```
public class StringPrinter {  
    static synchronized void printStrings(String stringA, String stringB) {  
        System.out.print(stringA);  
        try {  
            Thread.sleep(500);  
        } catch (InterruptedException ie) {  
            ie.printStackTrace();  
        }  
        System.out.println(stringB);  
    }  
}
```

Add the *synchronized* keyword in method declaration.

# Output – with sync method

Execute the class, the output could be displayed as below.

```
Hello..There  
How are you  
Thank you,very much
```

**The output is as expected.**

Inference:

The *synchronized* keyword has ensured that the thread executes in order. Only after a thread completes printing both the messages the other thread starts the printing.

# Method 2: sync statements

## Where is Synchronized statement used?

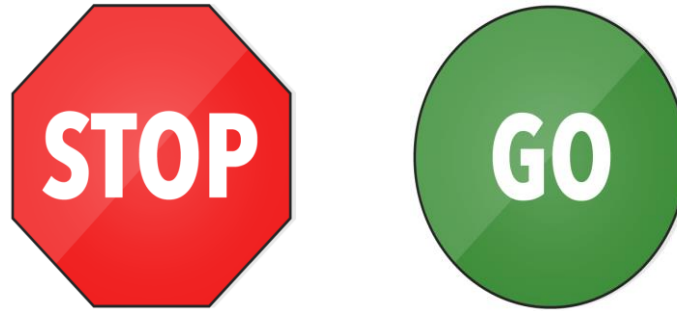
If only a small portion of code in a method needs to be Thread safe we can mark that portion only as synchronized rather than marking the entire method as ***synchronized***.

## Syntax:

```
public class SyncMethods{  
    public void methodA() {  
        synchronized(this){  
            //This block contains a set of  
            //statements which needs to be synchronized  
        }  
    }  
}
```

The syntax is **synchronized(this)** where **this** denotes the current object meaning the thread should acquire a lock on the current object to execute the statements.

# Time To Reflect



**Trainees to reflect the following topics before proceeding.**

- What is the need of synchronization in multi threaded programming?
- What is a race condition?
- How can Synchronization be implemented?



# Inter Thread Communication


In many instance we end up in a situation where we need one thread to communicate with other threads in a process. This is referred to as “***Inter Thread Communication***”

Inter Thread communication is achieved using one or more of the below methods,

1.wait()

2.notify()

3.notifyAll()



Occasionally used in  
application  
development.

# wait()

- ***wait()*** method causes a thread to release the lock it is holding on an object allowing another thread to run.
- ***wait()*** can only be invoked within a synchronized code.
- It should always be wrapped in a try block and handle ***IOException***.
- ***wait()*** can only be invoked by the thread that owns the lock on the object using `synchronized(this)`.

# How wait method works?

- When ***wait()*** is invoked, the thread becomes dormant until one of the below four things occur,
  - Another thread invokes the ***notify()*** method for this object and the scheduler arbitrarily chooses to run the thread.
  - Another thread invokes the ***notifyAll()*** method for this object.
  - Another thread interrupts this thread.
  - The specified ***wait()*** time elapses.
- When one of the above occurs, the thread becomes re-available to the Thread scheduler and competes for a lock on the object.
- Once it regains the lock on the object, everything resumes as if no suspension had occurred.

# notify()

- ***notify()*** method wakes up a single thread that is waiting on the monitor of this object
  - If many threads are waiting on this object, then one of them is chosen to be awakened.
  - The choice is arbitrary and occurs at the discretion of the JVM implementation.
- ***notify()*** method can only be used within synchronized code.
- The awakened thread will not be able to proceed until the current thread relinquishes the lock on this object.

# notifyAll()

- Similar to notify method ***notifyAll()*** is also used to wake up threads that wait on an object
- The difference is ***notify()*** awakens only one thread whereas ***notifyAll()*** awakens all the threads

# Deadlocks

***Deadlock*** describes a situation where two or more threads are blocked forever, waiting for each other.

Example to understand dead locks:

Tim & Ron are two good friends they belong to a country where they have a weird culture, whenever they meet each other one has to bow and wish the other “*Good Day*” he has to remain bowed till the other person bows back and greets back “*Thank you*”

Lets see what happened?

# Example – Inter thread communication

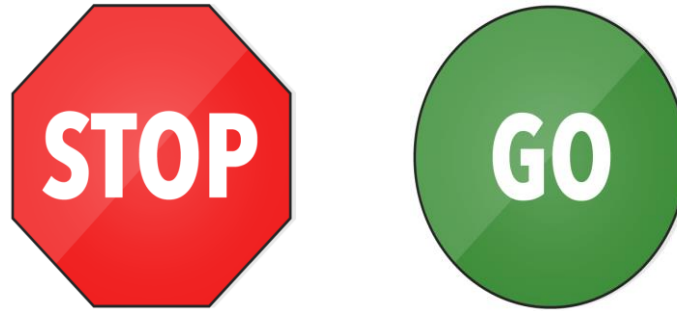
```
class Customer{
    int balance = 10000;
    synchronized void withdraw(int amount) {
        System.out.println("I am going to withdraw amount...");
        if(this.balance<amount) {
            System.out.println("Less balance; wait for some time.");
            try {
                wait();
            }catch(Exception e) {}
        }
        this.balance-=amount;
        System.out.println("Amount Received. Thank you.!!");
    }
    synchronized void deposit(int amount) {
        System.out.println("Going to Deposit...");
        this.balance+=amount;
        System.out.println("Deposit Completed..");
        notify();
    }
}
```

# Example – Inter thread communication

```
public class InterThreadDemo {  
    public static void main(String[] args) {  
        final Customer cust = new Customer();  
        new Thread() {  
            public void run() {  
                cust.withDraw(9000);  
            }  
        }.start();  
        new Thread() {  
            public void run() {  
                cust.deposit(10000);  
            }  
        }.start();  
    }  
}
```



# Time To Reflect



**Trainees to reflect the following topics before proceeding.**

- How can we make a thread wait on an object it holds the lock?
- What is the difference between notify and notify all?
- What is a dead lock?

Thank you

*You have successfully completed*  
**Thread-III**