MODULE-5
COLLECTIONS

Course Topics

DURSIKSHYA
Transforming Education

- → Module 1
  - » Introduction to Java
- → Module 2
  - » Data Handling and Functions
- → Module 3
  - » Object Oriented Programming in Java
- → Module 4
  - » Packages and Multi-threading
- → Module 5
  - » **Collections**
- → Module 6
  - » XML
- → Module 7
  - » JDBC
- → Module 8
  - » Servlets
- → Module 9
  - » JSP
- → Module 10
  - » Hibernate
- → Module 11
  - » Spring
- → Module 12
  - » Spring, Ajax and Design Patterns
- → Module 13
  - » SOA
- → Module 14
  - » Web Services and Project

Objectives

DURSIKSHYA
Transforming Education

At the end of this module, you will be able to

→ Identify and use important Inbuilt Java Packages like java.lang, java.io, java.util etc.

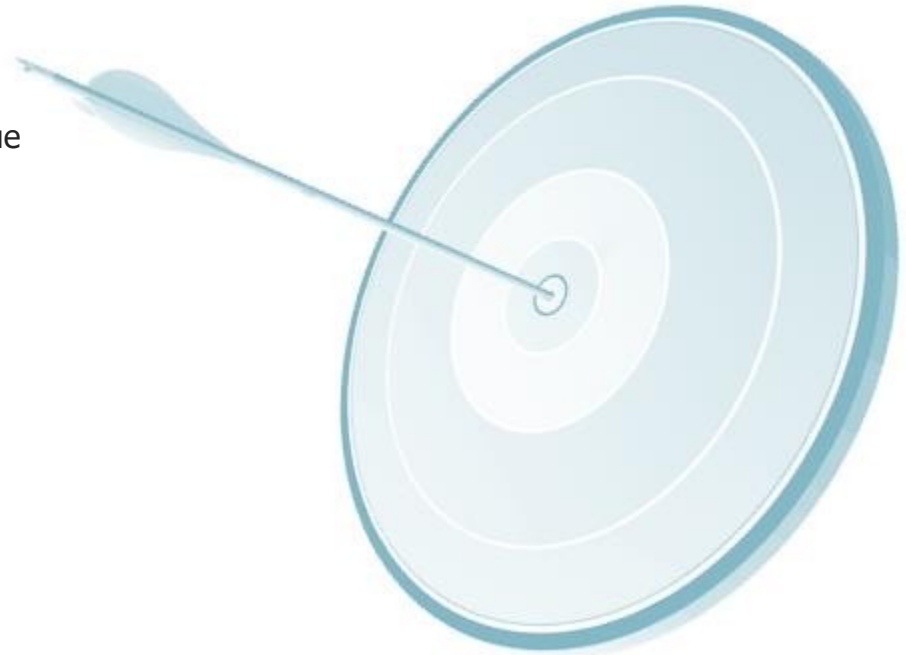
→ Use Wrapper classes

→ Understand collections framework

→ Implement logic using ArrayList and Vector and Queue

→ Use set, HashSet and TreeSet

→ Implement logic using Map HashMap and Hashtable



Object Level Locking

DURSIKSHYA
Transforming Education

Object level locking is mechanism when you want to synchronize a non-static method or non-static code block such that only one thread will be able to execute the code block on given instance of the class.

This should always be done to make instance level data thread safe.

```
public class Dursikshya
{
    public synchronized void
demoMethod(){}
}
```

```
public class Dursikshya
{
    public void demoMethod(){
        synchronized (this)
        {
            //other thread safe code
        }
    }
}
```

```
public class Dursikshya
{
    private final Object lock = new
Object();
    public void demoMethod(){
        synchronized (lock)
        {
            //other thread safe code
        }
    }
}
```

Class Level Locking

DURSIKSHYA
Transforming Education

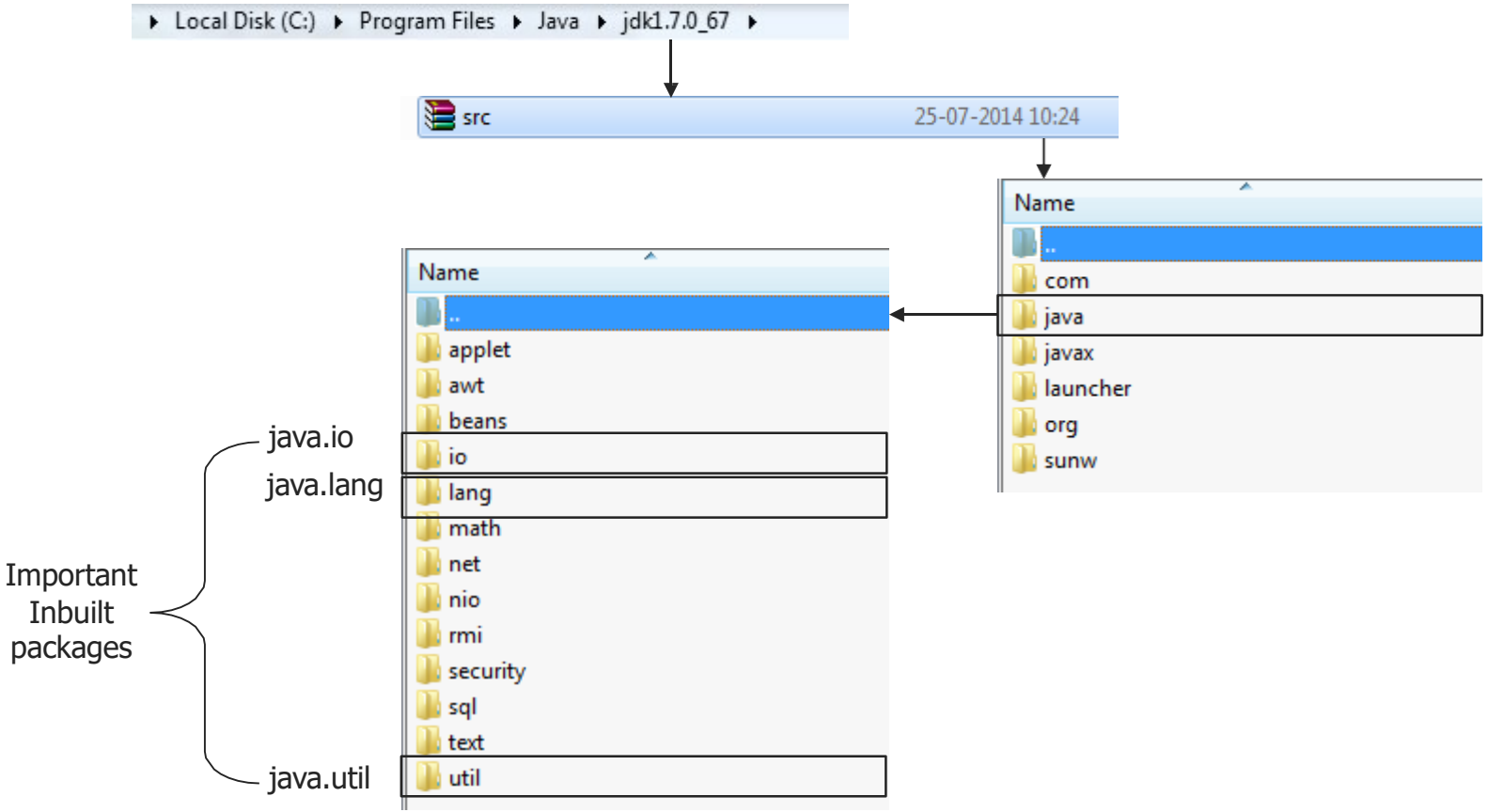
Class level locking prevents multiple threads to enter in synchronized block in any of all available instances on runtime. This should always be done to make static data thread safe.

```
public class Dursikshya
{
    public synchronized static
void demoMethod(){}
}
```

```
public class Dursikshya
{
    public void demoMethod(){
        synchronized (Dursikshya.class)
        {
            //other thread safe code
        }
    }
}
```

```
public class Dursikshya
{
    private final static Object lock = new
Object();
    public void demoMethod(){
        synchronized (lock)
        {
            //other thread safe code
        }
    }
}
```

Inbuilt Java Packages



java.lang

→ This package provides the classes which are fundamental to Java language.

→ Some of the classes in this package are:

- » Byte, Character, Integer, Long, Float – Wrapper classes.
- » Math, String, StringBuffer, StringBuilder
- » System
- » Thread
- » Exceptions

java.lang

DURSIKSHYA
Transforming Education

Math - The class Math contains methods for performing basic numeric operations such as the elementary exponential, logarithm, square root, and trigonometric functions.

String - The String class represents character strings. All string literals in Java programs, such as "abc", are implemented as instances of this class.

StringBuffer - A thread-safe, mutable sequence of characters. A string buffer is like a String, but can be modified. At any point of time, it contains some particular sequence of characters, but the length and content of the sequence can be changed through certain method calls.

StringBuilder - A mutable sequence of characters. This class provides an API compatible with StringBuffer, but with no guarantee of synchronization.

System - The System class contains several useful class fields and methods for standard input, standard output, and error output streams etc. It cannot be instantiated.

Thread - A thread is a thread of execution in a program.

Exceptions - The class Exception and its subclasses are a form of Throwable that indicates conditions that a reasonable application might want to catch.

Note: These classes has been covered in the previous modules

Wrapper Class

Why Wrapper Classes?

DURSIKSHYA
Transforming Education

→ Since Java is an Object Oriented language, sometimes it is required to send object instead of a primitive data type, in this case wrapper classes object is used.

→ For example, in Collections like ArrayList (You will learn about collections little later in this module) collection classes can only store objects. Here primitive data type can't be used.

→ If integer or character data is to be stored in collections then it's corresponding objects are required. Hence wrapper classes are created.

Use Cases - Wrapper Classes

DURSIKSHYA
Transforming Education

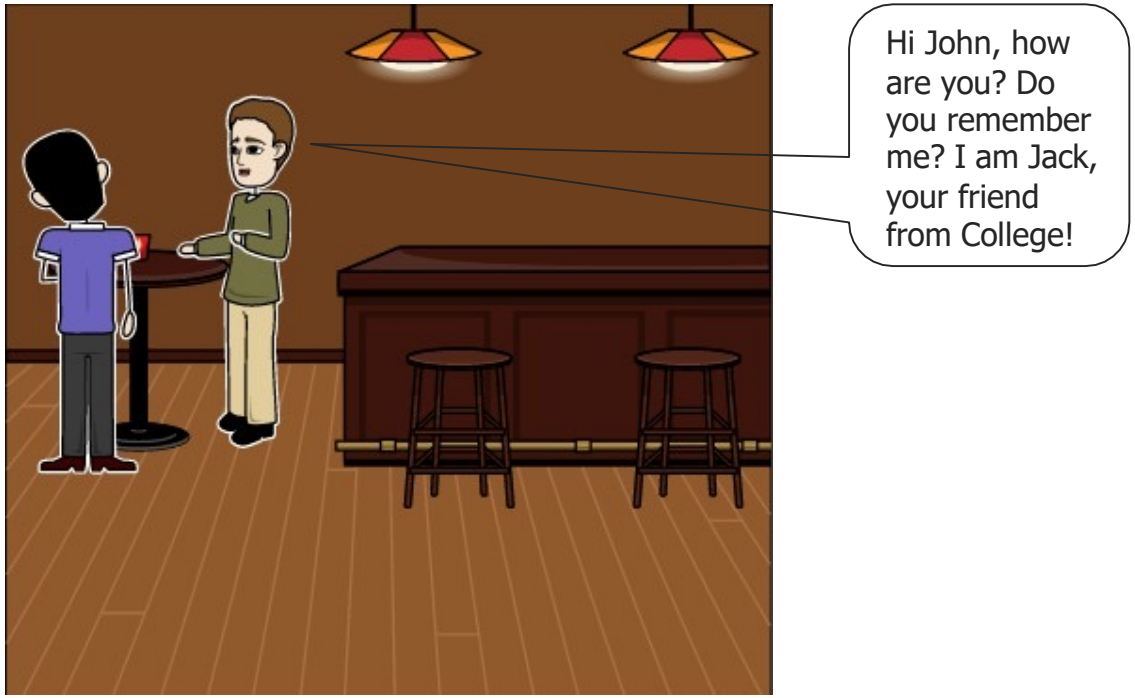
Use Cases, * Dataset/Problem, * Solution of the Dataset/Problem

John takes a laptop from India to USA. There he finds that battery charging pins are different. He will have to find an adapter which works for USA laptops plug points. [He will use the adapter and will charge the laptop] Here the adapter is like a wrapper class.

http://www.primap.com/en/html/maps_-_world.html

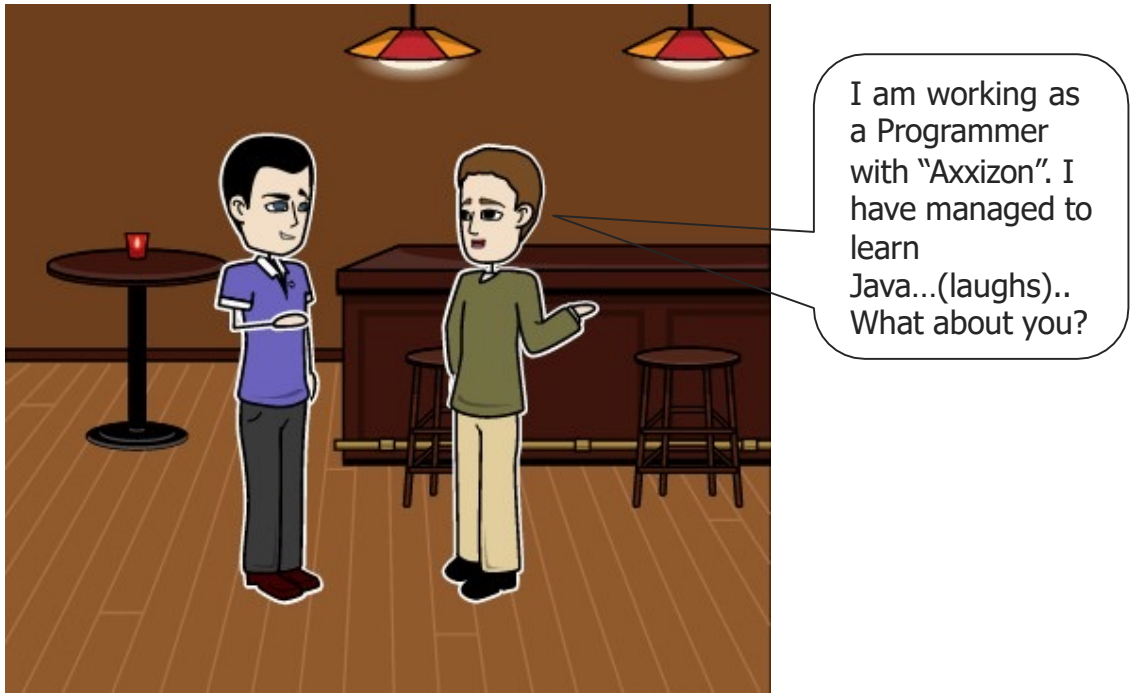
John Meets Jack

John meets Jack in a Coffee shop. Jack was John's classmate in college.

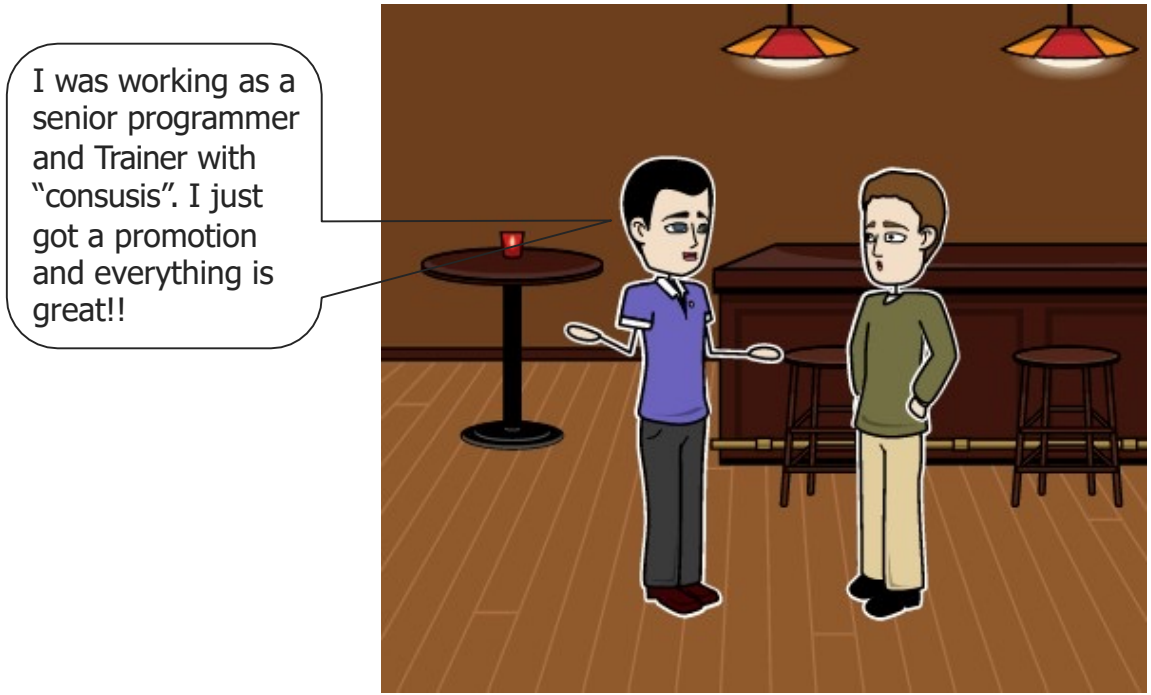


John Meets Jack

Jack is a Programmer too..

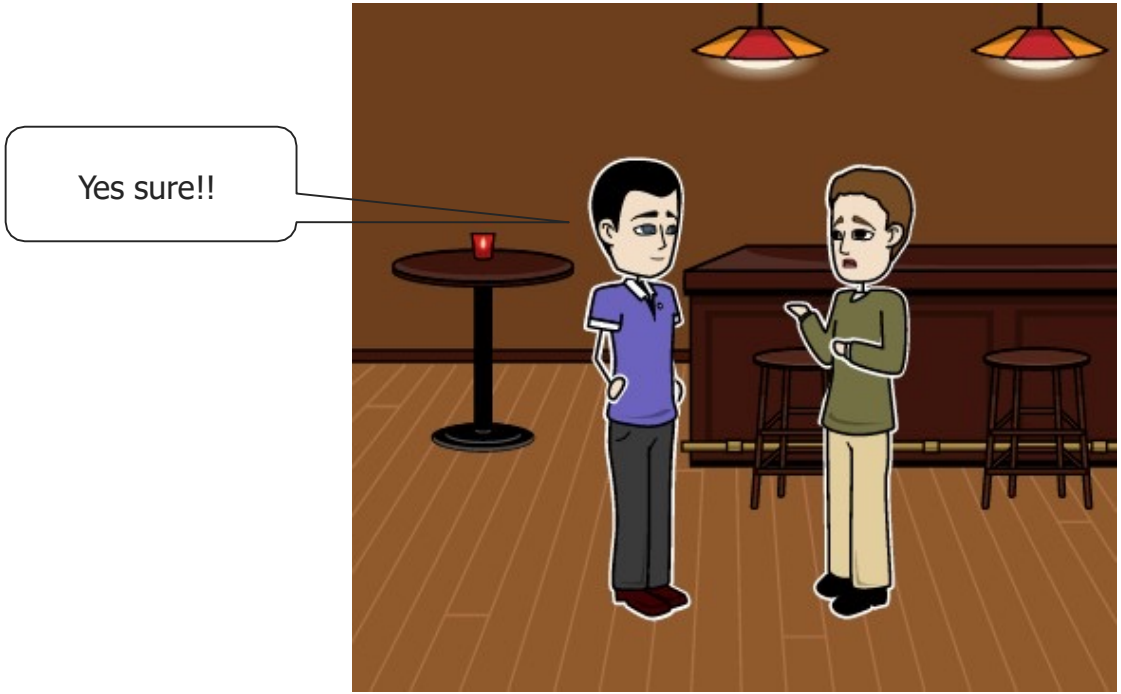


John Talks about his Role..



Jack needs Help!

Jack needs Help!



John explains “Wrapper Class”

Take an example of this chocolate. The manufacturer wraps the chocolate with foil or paper to prevent it from dust. The consumer takes the chocolate, removes and throws the wrapper and then eats it.

John explains “Wrapper Class”

Similarly…..

A Wrapper class wraps a data type and gives it an object appearance. This object can be used wherever, the data type is required as an object. Wrapper classes has methods to unwrap the object and return the data type.

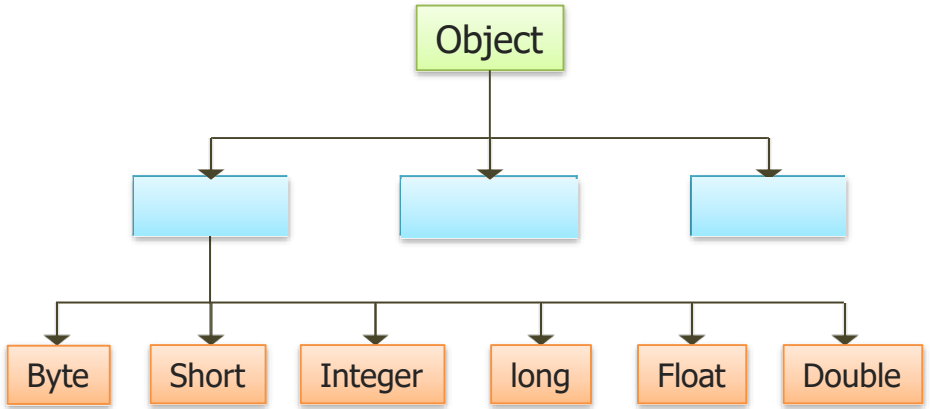
Lets Learn More…….

More on Wrapper classes

DURSIKSHYA
Transforming Education

→ Each of Java's eight primitive data types has a class dedicated to it. These are known as wrapper classes, because they "wrap" the primitive data type into an object of that class.

Primitive data type	Wrapper class
byte	Byte
short	Short
Int	Integer
long	Long
float	Float
double	Double
char	Character
boolean	Boolean



Wrapper classes Hierarchy

Where do we use Wrapper class?

DURSIKSHYA
Transforming Education

In collections, wrapper classes are used for primitive data types.

All the places where primitive data type objects are required, wrapper class is used.

How it works?

An integer wrapper class is declared as below:

```
Integer I = new Integer (10);
```

Features of Wrapper Classes

DURSIKSHYA
Transforming Education

→ We can use various class methods like converting integers to string.

→ This is one of the way to store primitive data type into object.

→ valueOf() is available in all the wrapper classes to get the value of given data into the Wrapper class data type.

Program on Wrapper Classes

DURSIKSHYA
Transforming Education

Wrapper class Integer can be used for the primitive data type int.

```
public class wrapper {

    public static void main(String args[]) {

        int i = 10;
        // Creating Integer Object
        Integer intObject = new Integer(i);
        System.out.println(intObject);
        // Using class methods to convert int to string
        System.out.println( intObject.toString() );
    }
}
```

Java.io

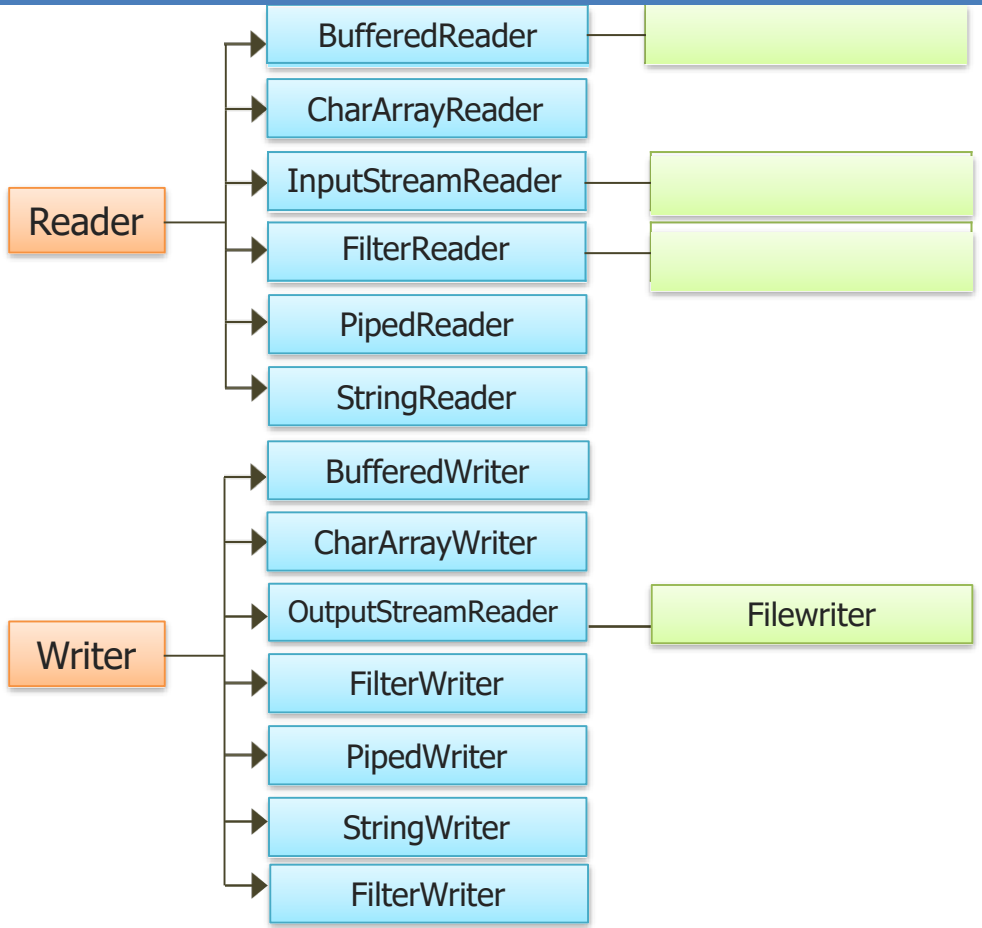
→ This package provides classes to perform IO operations using streams in Java.

→ A stream in Java is a sequence of bytes of undetermined length. It can be compared to water streams.

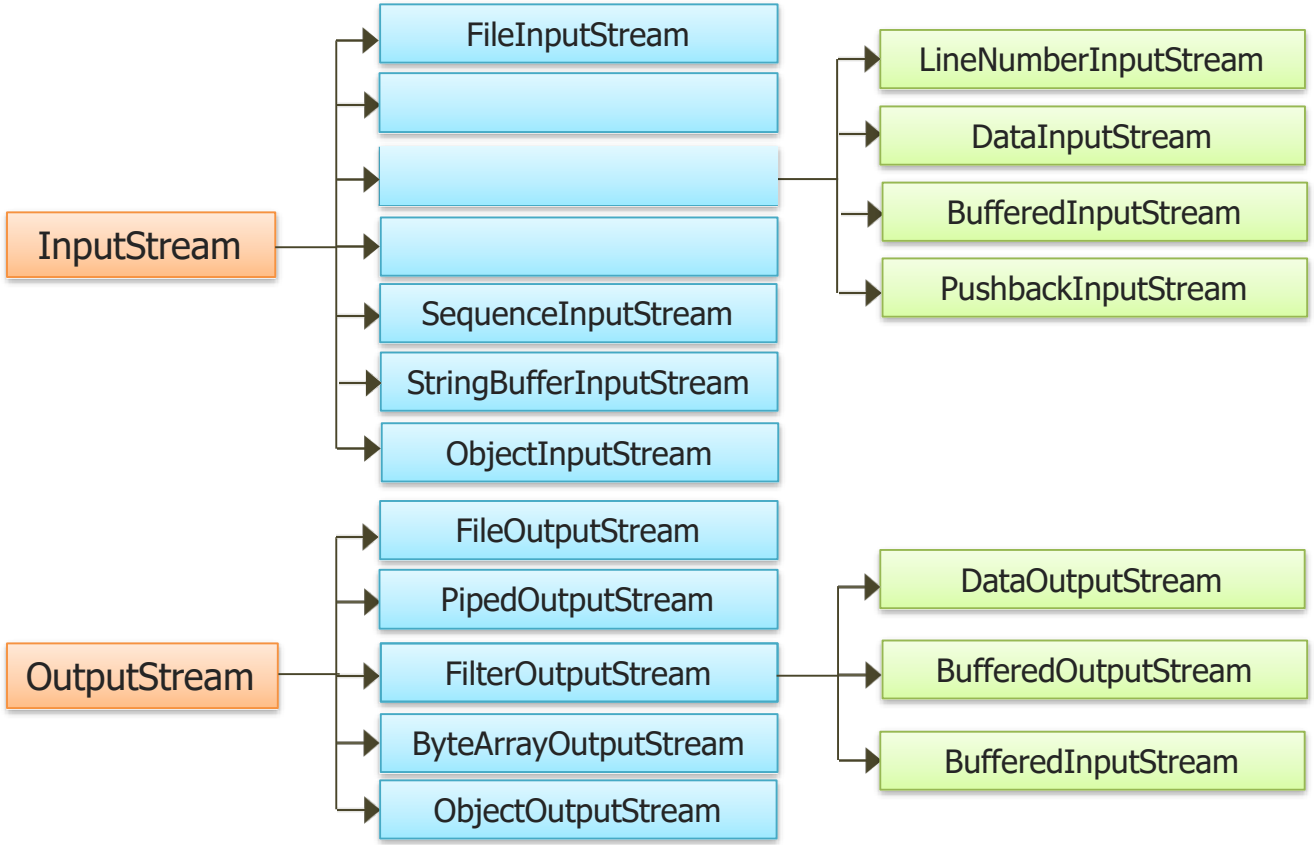
→ InputStream class can be used to read the data from a data file.

→ OutputStream class can be used to write into a data file.

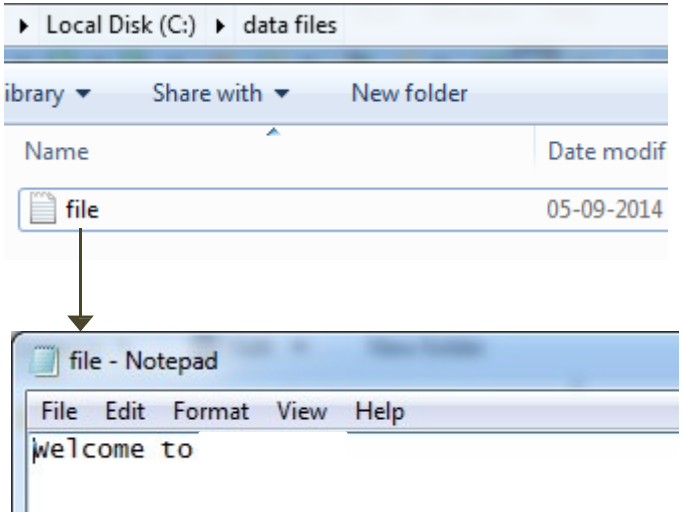
java.io - Classes



java.io - Classes

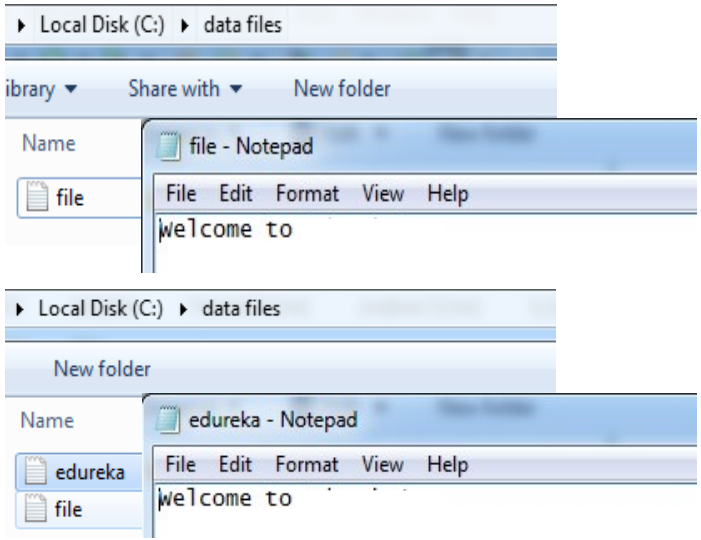


Program to Read a File




```java
import java.io.*;

public class FileReadDemo {
    public static void main(String[] args) throws IOException {
        // file.txt present in c drive
        File file = new File("c:\\data files\\file.txt");
        FileInputStream fis = null;
        try {
            fis = new FileInputStream(file);
            int input;
            // Reading the file
            while ((input = fis.read()) != -1) {
                // convert to char and display it
                System.out.print((char) input);
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
        fis.close();
    }
}
```

Program to Copy one File to Another




```java
import java.io.*;

public class FileWriteDemo {
    public static void main(String[] args) throws IOException {
        File file = new File("c:\\data files\\file.txt");
        FileInputStream fis = null;
        FileOutputStream fout = null;
        try {
            fis = new FileInputStream(file);
            fout = new FileOutputStream("c:\\data files\    ABC    txt");
            int content;
            while ((content = fis.read()) != -1) {
                fout.write(content);
            }
            System.out.println("File copied successfully...");
        } catch (IOException e) {
            e.printStackTrace();
        }

        fis.close();
        fout.close();
    }
}
```

java.util

Collections are very important in Java programming.

→ This package has all the utility classes required in Java program.

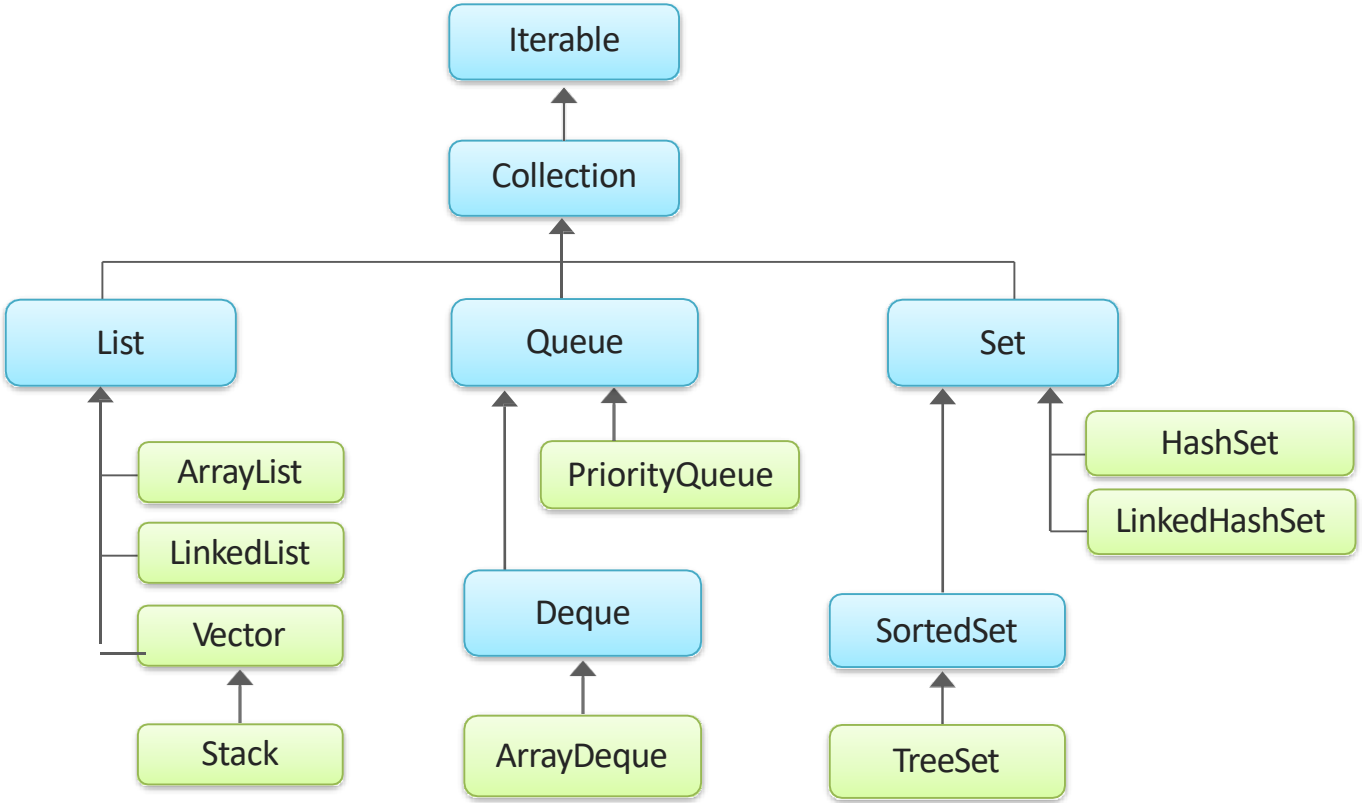
→ All the collection classes which will be discussed in the next slides are part of java.util.

→ Apart from collections, it has following important classes:

- » Date
- » Scanner

Collections

Collections: provide set of classes to store and manipulate objects or group of objects.



Collection Classes

Important Collection classes are:

→ ArrayList

→ Vector

→ HashSet

→ TreeSet

→ HashMap

→ Hashtable

→ Properties

Why ArrayList when Array is there?

DURSIKSHYA
Transforming Education

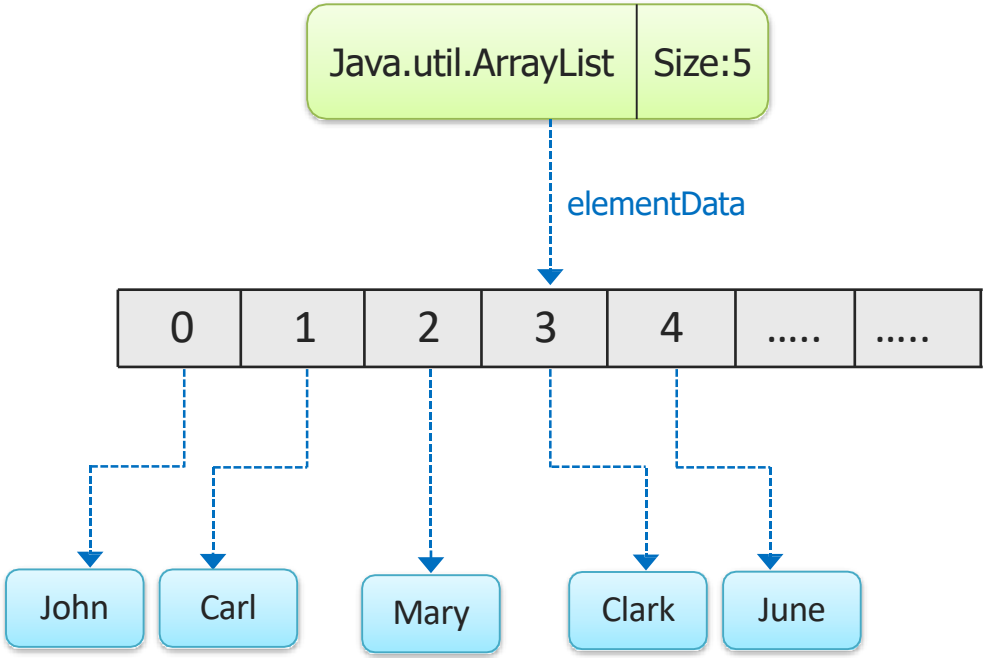
→ Array is fixed.

→ ArrayList is re-sizeable.

ArrayList

- → ArrayList is the implementation of List Interface.
- → Array size is fixed but ArrayList size can grow dynamically.
- → ArrayList is used to store objects and perform operations on it.
- → ArrayList is not Synchronized. Vector is similar to ArrayList which is synchronized.

If your application does not require insertion or deletion of elements, the most efficient data structure is the array



ArrayList (contd.)

→ ArrayList is declared as:

```
ArrayList <Object to store> ArrayList object = new ArrayList <Object to store> ();
```

→ The given program stores String objects. Prints all the objects of the ArrayList in new type of array.

→ Removes the string object by name and position and displays the complete collection object.

Methods in ArrayList

→ boolean add(Object e)
→ void add(int index, Object element)
→ boolean addAll(Collection c)

→ Object get(int index)
→ Object set (int index, Object element)

→ Object remove(int index)

→ Iterator iterator()
→ ListIterator listiterator()

→ int indexOf()
→ int lastIndexOf()

→ int index (Object element)
→ int size()
→ void clear()

Program on ArrayList

DURSIKSHYA
Transforming Education

```java
import java.util.ArrayList;

public class arrayList {
    public static void main(String[] args) {
        // Declaring an arraylist
        ArrayList<String> al = new ArrayList<String>();
        // Adding elemnts to arraylist
        al.add("John");
        al.add("Dean");
        al.add("Sam");
        System.out.println(al);
        // Traversing arraylist using for loop
        for (String str : al) {
            System.out.println("Welcome : " + str);
        }
        // Printing size of arraylist
        System.out.println("Size of ArrayList: "+al.size());
        //Removing elements from arraylist
        al.remove(2);
        al.remove("Dean");
        System.out.println("Arraylist after removing elements:");
        System.out.println(al);
        // Printing size of arraylist
        System.out.println("Size of ArrayList: "+al.size());
    }
}
```

How to trace the elements of ArrayList?

→ For-each loop

→ Iterator

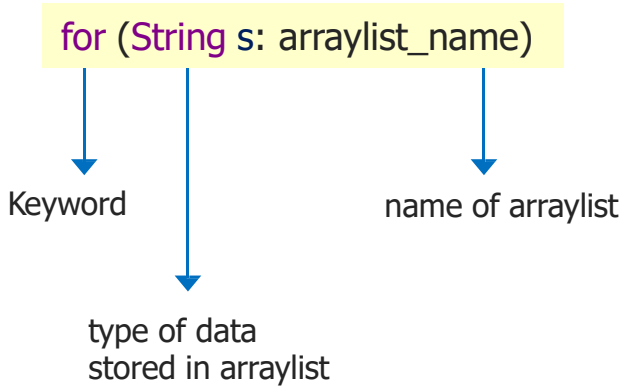
→ ListIterator

→ Enumeration

For-each Loop

→ It's action is similar to for-loop. It traces through all the elements of array or ArrayList.

→ No need to mention size of ArrayList.



Iterator

→ Iterator is an interface that is used to traverse through the elements of collection.

→ It traverses only in forward direction with the help of methods.

Iterator Methods

→ Boolean hasNext()

→ element next()

→ void remove()

Displaying Items Using Iterator

DURSIKSHYA
Transforming Education

```
Iterator iterator = arraylist.iterator();

while (iterator. hasNext()) {
        Object object = iterator.next();
        System.out.print(object + "   ");
}
```

ListIterator

→ ListIterator is an interface that traverses through the elements of the collection.

→ It traverses in both forward and reverse direction.

ListIterator Methods

→ boolean hasNext()

→ element next()

→ void remove()

→ boolean hasPrevious()

→ element previous()

Displaying Items Using ListIterator

DURSIKSHYA
Transforming Education

```
// To modify objects we use ListIterator
ListIterator  listiterator  =
arraylist.listIterator();

  while (listiterator.hasNext())  {
        Object object = listiterator.next();
         listiterator.set("(" + object  +  ")");
}
```

Enumeration

→ Enumeration is an interface whose action is similar to iterator.

→ But the difference is that it has no method for deleting an element from ArrayList.

Enumeration Methods

→ boolean hasMoreElement()

→ element nextElement()

Displaying Items Using Enumeration

DURSIKSHYA
Transforming Education

```
Enumeration  enumeration =
Collections.enumeration(arraylist);

while (enumeration.hasMoreElements()) {
        Object object = enumeration.nextElement();
         System.out.print(object  +  "   ");
}
```

Vector

Vector is same as ArrayList. The only difference is that, Vector is synchronized but ArrayList is not. That means Vector is ThreadSafe.

```
import java.util.Vector;

public class arrayList {
    public static void main(String[] args) {
        // Declaring a Vector
        /* Constructor --> Vector(int size, int incr)
           The increment specifies the number of elements to
           allocate each time that a vector is resized upward  */
        Vector<Integer> v = new Vector<Integer>(3, 2);
        // Adding elemnts to vector
        v.add(10);
        v.add(20);
        System.out.println(v.size());
    }
}
```

Queue

→ Queue is an interface.

→ PriorityQueue is a class which implements Queue interface and sorts the data stored. Deque stands for double ended queue.

→ The queue can be operated at both the ends. Same holds for Insertion and deletion too.

→ Queue operates on the principle of "First in First out". The element which gets inserted in the beginning is the one which comes out first. The element which gets inserted in the last will come out last.

Program on PriorityQueue

This program inserts the data from 10 to 1 in the queue and displays it in a sorted order.

```
import java.util.PriorityQueue;

public class priorityQueue {
    public static void main(String[] args) {
        PriorityQueue<Integer> pq = new PriorityQueue<Integer>();

        for (int i = 10; i > 0; i--) {
            pq.add(new Integer(i));
        }

        for (int i = 1; i <= 10; i++) {
            Integer c = pq.poll();
            System.out.println(c);
        }
    }
}
```

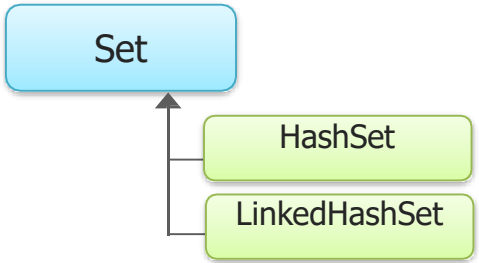
Set

→ A set is a collection interface. HashSet and TreeSet are implemented from Set.

→ Set cannot have duplicates.

→ Class HashSet removes the duplicates and gives better performance over TreeSet. It doesn't guarantee to store the data in the same order it is inserted.

→ Class TreeSet sorts the added data apart from removing the duplicates.



HashSet

```java
import java.util.HashSet;
import java.util.Iterator;

public class priorityQueue {
    public static void main(String[] args) {
        HashSet<Integer> hs = new HashSet<Integer>();
        hs.add(1);
        hs.add(1);
        hs.add(2);
        hs.add(3);
        hs.add(4);

        Iterator<Integer> it = hs.iterator();
        while (it.hasNext()) {
            int i = (Integer) it.next();
            System.out.println(i);
        }
    }
}
```

This program adds numbers into the HashSet and prints it. You can observe that 1 is inserted twice. As Hashing removes duplicates, it prints 1 only once and removes the second 1.

TreeSet

DURSIKSHYA
Transforming Education

```
import java.util.TreeSet;

public class priorityQueue {
    public static void main(String[] args) {
        TreeSet<String> ts = new TreeSet<String>();
        ts.add("Pamela");
        ts.add("Pamela");
        ts.add("Anglina");
        ts.add("Britney");
        ts.add("Shakira");
        ts.add("Fergie");

        System.out.println(ts);
    }
}
```

This program adds names, sorts them and displays the names using TreeSet. Observe that Pamela is added twice in the collection. When printed Pamela will be displayed only once as the second entry is removed as duplicate entry by TreeSet.

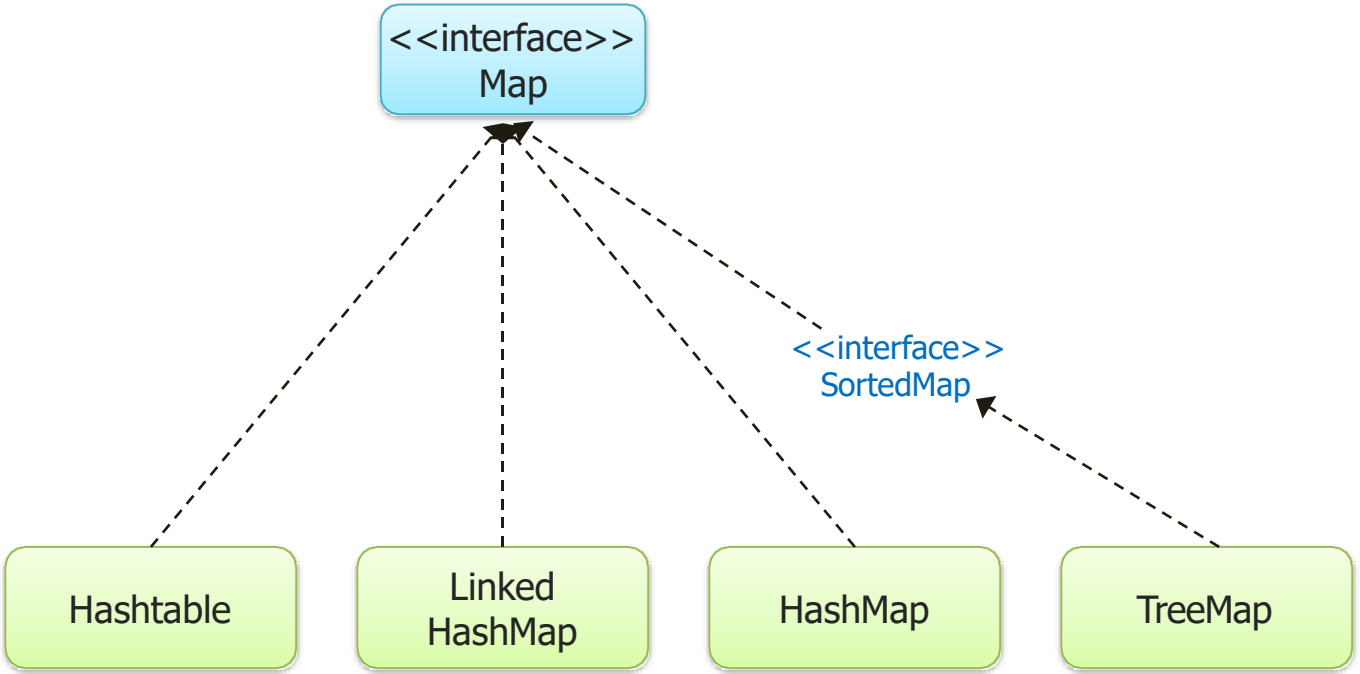
Map

→ Map is an interface which has key value pair.

→ An object is identied by a key. If key is passed, its corresponding object can be retrieved. It is like in a employee record, employee data is identied by employee id. Employee id is the key and employee data is the object.

→ Some of the important classes of Map are HashMap and Hashtable.

Classes Implementing map Interface



HashMap

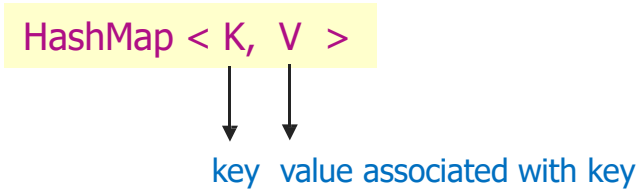
→ Implemented map interface which is used to perform operations such as inserting, deleting and locating elements in a map.

→ HashMap stores data as key value pair.

→ It is unsorted and unordered.

→ It is not synchronized.

→ Map allows one null key and multiple null values



key act as indexes and can be any objects

Methods in HashMap

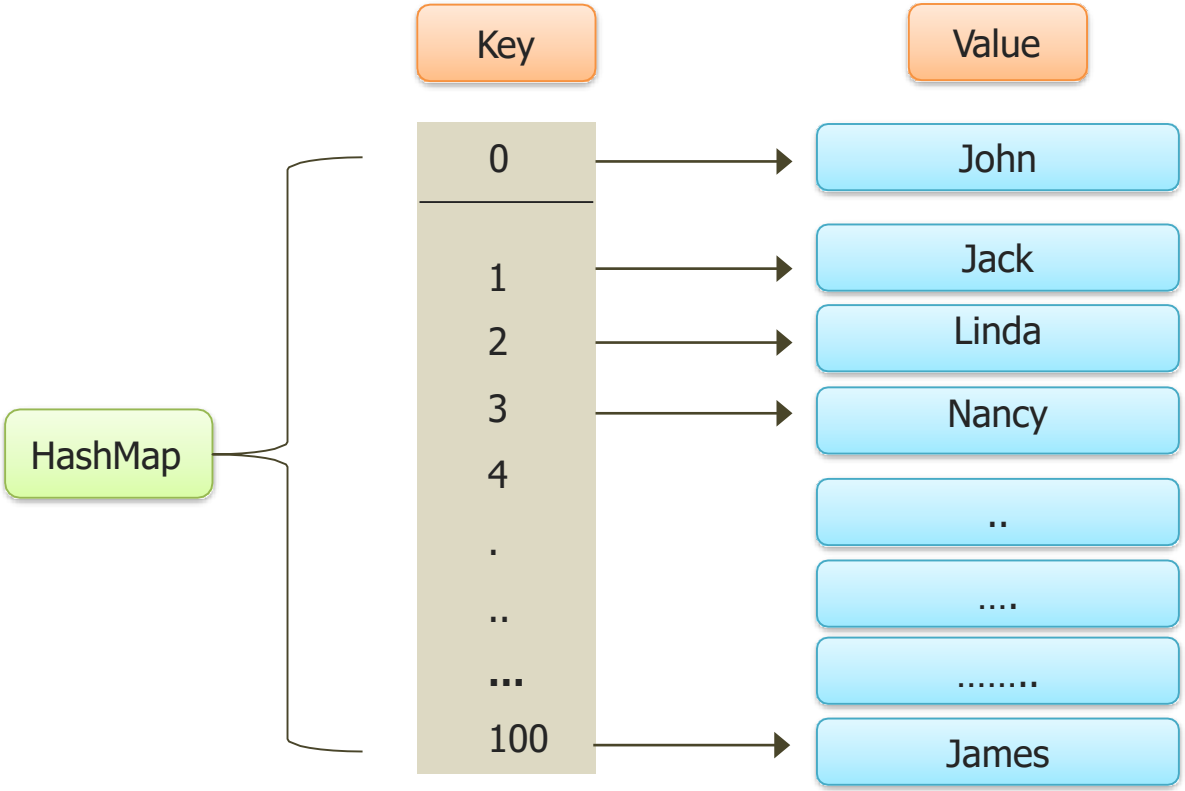
- Object put(Object key, Object value)

- Enumeration keys()
- Enumeration elements()
- Object get(Object keys)

- boolean containsKey(Object key)
- boolean containsValue(Object key)

- Object remove(Object key)
- int size()
- String toString()

HashMap – Abstract View



HashMap – Insertion

DURSIKSHYA
Transforming Education

```
// Create a hash map
HashMap hashmap = new HashMap();

//Putting elements
hashmap.put("John", 9634.58);
hashmap.put("Jack", 1283.48);
hashmap.put("Linda", 1458.10);
hashmap.put("Nancy", 199.11);
```

HashMap – Display

DURSIKSHYA
Transforming Education

```
// Get an iterator
Iterator iterator = hashmap.entrySet().iterator();


//Display elements
while (iterator.hasNext()) {
      Map.Entry entry = (Map.Entry) iterator.next();
      System.out.print(entry.getKey() + ":");
      System.out.println(entry.getValue());
}
```

Program on HashMap

DURSIKSHYA
Transforming Education

```
HashMap<String, Double> hm = new HashMap<String, Double>();
// Put elements to the map
hm.put("Fergie", new Double(3434.00));
hm.put("Britney", new Double(123.22));
hm.put("Anglina", new Double(1378.00));
hm.put("Shakira", new Double(99.22));
hm.put("Rihana", new Double(-19.08));
// Adding null key and value
hm.put(null, null);
// Get a set of the entries.
Set<Map.Entry<String, Double>> set = hm.entrySet();
// Display the set.
for (Map.Entry<String, Double> me : set) {
    System.out.print(me.getKey() + ": ");
    System.out.println(me.getValue());
}
System.out.println();
// Deposit 1000 into Fergie's account.
double balance = hm.get("Fergie");
hm.put("Fergie", balance + 1000);
System.out.println("Fergie's new balance: " + hm.get("Fergie"));
```

Hashtable

- → Hashtable is implemented from map interface which is used to perform operations such as inserting, deleting and locating elements similar to HashMap.
- → Hashtable is Synchronized. That is, it is thread safe.
- → It does not allow null keys or values, if inserted compiler throws NullPointerException.
- → Hashtable stores data as key value pair.
- → It is unsorted and unordered. This is due to hashing.

Methods in Hashtable

DURSIKSHYA
Transforming Education

→ Object put(Object key, Object Value)

→ Enumeration keys()
→ Enumeration elements()
→ Object get(Object keys)

→ boolean containsKey(Object key)
→ boolean containsValue(Object key)

→ Object remove(Object key)
→ int size()
→ String toString()

Hashtable - Insertion

DURSIKSHYA
Transforming Education

```
// Create a hash map
Hashtable hashtable = new Hashtable();

// Putting elements
hashtable.put("John", 9634.58);
hashtable.put("Jack", 1283.48);
hashtable.put("Linda", 1478.10);
hashtable.put("Nancy", 199.11);
```

Hashtable - Display

DURSIKSHYA
Transforming Education

```
// Using Enumeration
Enumeration enumeration = hashtable.keys();

// Display elements
while (enumeration.hasMoreElements()) {
     String key = enumeration.nextElement().toString();

     String value = hashtable.get(key).toString();

     System.out.println(key + ":"+value);
}
```

Program on HashMap

DURSIKSHYA
Transforming Education

```
Hashtable<String, Double> balance = new Hashtable<String, Double>();
Enumeration<String> names;
String str;
double bal;
balance.put("Fergie", 3434.00);
balance.put("Nicolas", 123.22);
balance.put("Leonardo", 1378.00);
balance.put("Cristina", 99.22);
balance.put("Tom", -19.08);
//balance.put(null, 100.20);

// Show all balances in hashtable.
names = balance.keys();
while (names.hasMoreElements()) {
    str = names.nextElement();
    System.out.println(str + ": " + balance.get(str));
}

System.out.println();
// Deposit 7,000 into Fergie's account.
bal = balance.get("Fergie");
balance.put("Fergie", bal + 7000);
System.out.println("Fergie's new balance: " + balance.get("Fergie"));
```

Why Use Inner Classes?

Compelling reasons for using nested classes include the following:

→ It is a way of logically grouping classes that are only used in one place: If a class is useful to only one other class, then it is logical to embed it in that class and keep the two together. Nesting such "helper classes" makes their package more streamlined.

→ It increases encapsulation: Consider two top-level classes, A and B, where B needs access to members of A that would otherwise be declared private. By hiding class B within class A, A's members can be declared private and B can access them. In addition, B itself can be hidden from the outside world.

→ It can lead to more readable and maintainable code: Nesting small classes within top-level classes places the code closer to where it is used.

Inner Classes

Sometimes it is required to have an object inside another object in Object Oriented programming. This is possible through inner classes in Java.

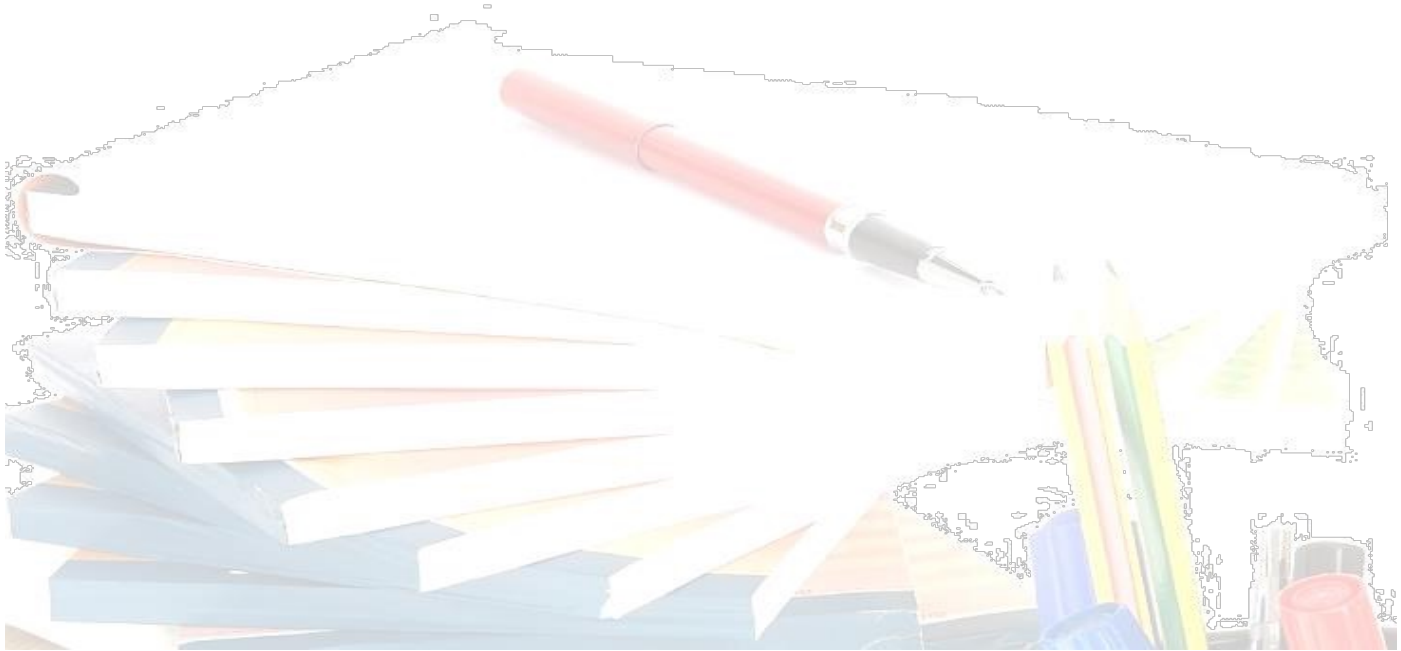
```
class outer {
    class inner {
        void test() {
            System.out.println("In the inner class..");
        }
    }
}

public class wrapper {
    public static void main(String args[]) {
        outer out = new outer();
        outer.inner ob = out.new inner();
        ob.test();
    }
}
```

QUESTIONS

Assignment - Wrapper Classes

DURSIKSHYA
Transforming Education

→ Try all the Wrapper classes as shown for Integer.



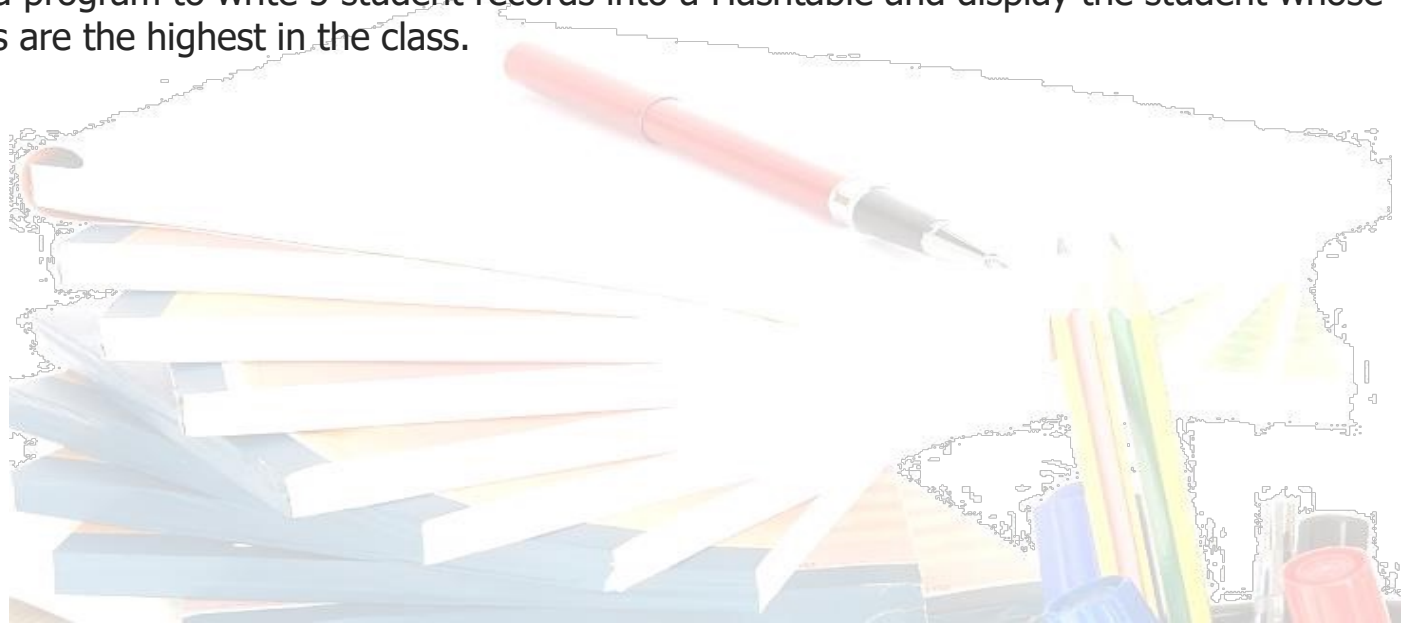
Assignment in Collections

DURSIKSHYA
Transforming Education

→ Write a program to insert 1 to 10 numbers in ArrayList and display them.

→ Write a program to write 5 employee records using HashMap and display them.

→ Write a program to write 5 student records into a Hashtable and display the student whose marks are the highest in the class.



Agenda of the Next Class

DURSIKSHYA
Transforming Education

In the next module, you will be able to:

→ Understand why we need XML?

→ Understand the features of XML

→ Create and use XML files

→ Understand and use DTD

→ Understand XSD and XPath

→ Use XML parsers like SAX and DOM parser

→ Create and use XSL files

Pre-work

- Preparation for the next module: Go through XML and its format.

DURSIKSHYA
Transforming Education