Unit 3 Notes Java:

Files Handling In Java:

In simple language File Handling means Reading / Writing data into the file.

In java all the classes related file handling resides inside **java.io** package.

1) **Creating a file with File class:**

example:

```
File obj = new File("MS21159.txt");
System.out.println("File Created");
```

//File will be created in our project directory, In case if you get an error surround the block with try and catch.

2) **Reading Data from a file:**

We will use *FileReader* class for reading contents from a file.

```
FileReader fr = new FileReader("C:\\Users\\tusha\\Desktop\\test.txt");
        BufferedReader br = new BufferedReader(fr);
        String line;
        while((line = br.readLine()) != null){
           System.out.println(line);
        }   fr.close();
```

One thing to remember, FileReader class will read single character at a time from the file, so We're using BufferedReader which will read single line from the file.

BufferedReader class has readLine() method, if the file has line then it'll return that line, if it doesn't have line anymore.. it'll return null.
We'll iterate through loop as long as file has line. Loop will stop execution once it returns null.

There is one another method JVS tried, which includes using InputStreamReader, that's actually very complex so we will stick with FileReader class.

3) Writing Data Inside File:

We will use FileWriter class of Java.io package, in order to write the contents inside a file.

```
FileWriter x = new FileWriter("C:\\Users\\tusha\\Desktop\\test.txt");
        x.write("It is very beautiful day");
        x.close();
       System.out.println("Written Successfully");
```

FileWriter class has write() method which will write data inside the specified file.

There is one another method JVS tried, which includes using OutputStreamReader, that's actually very complex so we will stick with FileWriter class.

4) Deleting a file:

Example:

File obj = new File("MS21159.txt");

obj.delete();

The file name should be accurate, that means file with specified name should be present at project directory, otherwise it'll throw FileNotFoundException.

Collection Framework:

Framework is nothing but just a collection of Classes and Interfaces.

Each and every collection resides inside Java.util.* package.

We'll look at 5/6 Classes in depth.

1) **ArrayList**:

Use: It's Just a typical Array, but it provides much more functionality than simple Array. It has so many methods which reduces a lot of work for developer. To be honest, Collection framework got to be one of the finest concept of Java.

Syntax / Example:

```
ArrayList<Integer> obj = new ArrayList<Integer>();
```

You have to use Wrapper classes while using the ArrayList.

Some methods:

add(20) - >Will add element at the end.

Remove(3) : will remove the element of specified index.

Contains(10): will check if arraylist has this element or not.

addAll(arrList): this will add another ArrayList into an existing/current ArrayList.

ToArray(): will convert ArrayList into typical Array.

2) **Stack:**

Stack class in Collection framework works exactly like Stack Data Structure do. It's basic principle is "Last IN First Out".

Syntax / Example:

```
Stack<String> s = new Stack<String>();
s.push("ABC");
s.push("XYZ");
s.pop();
```

You guys know the functionality of Push and Pop.

3) Set:

Just like Python, Set is an unordered list of elements, one more thing is Sets can not have a duplicate items.

Set is actually an Abstract class which has 3 more classes inside it.

1) HashSet() 2) LinkedHashSet() 3) TreeSet();

4) HashMap:

Remember, in python we had "Dictionaries"? But we haven't seen them in Java yet. Don't worry. Collection framework has HashMap which provides the functionality of dictionary.

Syntax / Example:

```
HashMap<Integer, String> dict = new HashMap<Integer, String>();
dict.put(1, "XYZ");
dict.put(2, "ABC");
```

There are several methods here also,

1) get(1): pass key into get, and it will give you the value.

2) size(): common method, will give us size of HashMap

Most of the Collection classes actually share common methods.

5) LinkedList:

LinkedList does exactly same work as LinkedList Data Structure we studied in previous semester.

Syntax:

```
LinkedList<Integer> ll = new LinkedList<Integer>();
ll.add(10);
ll.remove(0);
```

There are several more classes too. But for now these are enough.

Multithreading:

What is the basic function of Thread? To handle multiple events at a same time. Just like that Here in Java also using thread we can "**Perform multiple action at a same time**".

In order to use Thread, you have to extend *Thread* class in your current class. *Thread* exists in a java.lang package. And in that class you have to override *run()* method. Run() method is of Thread class. We'll override it and will provide the facility we want.

Example:

```
public class MultiThreadThing extends Thread
{
        public void run(){
        for(int i=0; i<5; i++){
            System.out.println(i + "for thread " + tno);
        }
        }
}
```

Now we will go into our main class. And will create the objects *MultiThreadThing* Class.

```
 */
public static void main(String[] args) {

    MultiThreadThing m = new MultiThreadThing();
    MultiThreadThing m2 = new MultiThreadThing();
    m.start();
    m2.start();

}
```

Remember to use start method, if you use start method it'll execute that class for BOTH OBJECTS CONCURRENTLY.

If you use run(), it'll execute for **m** first, and the for **m2**.

Another approach is using interface "Runnable".

Would suggest you to watch "Code with John's" explanation of Threading.

Introduction to Network Programming:

Java networking is used for connection two or more computers together so they can share resources.
Java has socket programming which gives facility to share the resources between the totally different computing devices.

Common Network Protocols: 1) UDP 2) TCP
Common Terms in Java Networking:

- IP address
- Port Number
- Protocol
- Mac Address
- Socket
- Connection Oriented and Connection Less protocol

Remember every class for networking resides inside;
java.net package.

The main concept here is socket, and due to less time they have removed the socket from the course. So this is all you need to know about Networking In Java.

Lambda Expression:

Lambda expressions are similar to methods, but they do not need a name and they can be implemented right in the body of method.

Syntax:

Parameter → Expression

Expressions are limited, They Have to

1) return a value

2) can not contain variables

3) can not contain if, for blocks.

If there is some more work to do you can always put curly braces. Like this and you can also add multiple parameters

Syntax:

(parameter1, parameter2) → {Hard Expression}

Example:

```
interface demo{
        public void draw();
}
```

```
main mehod(){
        demo d2 = () -> {System.out.println(“I am called”); }
}
```

Since there is only one method in Interface, the line System.out.println(“I am called”); will go into draw() method.

Your knowledge about interfaces should be clear, then you’re ready to understand this.

Okay, you still might have problem understanding this, see without Lambda it will look like this;

```
interface demo{
        public void draw();
}
main mehod(){
        demo d2 = new demo(){System.out.println(“I am called”);}
        d2.draw();
}
```

I hope you got the better understanding now.

Serialization:

It is used to write state of an object into a ***Byte Stream.***

In order to work with serialization, we have to implement **java.io.Serializable** interface.

* String class and other wrapper classes implement this serializable interface already.

TestClass.java:

```
class TestClass implements Serializable{

}
```

Now this TestClass class, uses serializable interface, we can convert its *Object* into stream. Let's have a look at Main Method.

```
Main method(){
TestClass t = new TestClass();
//created stream for writing object into byte stream format.
FileOutputStream fout = new FileOutputStream("demo.txt");
ObjectOutputStream out = new ObjectOutputStream(fout);
```

```
out.writeObject(t); //writeObject method writes the current object in
stream

out.flush(); //flushes the OutPutStream, I.e saving.
out.close(); //closing stream.
}
```

These are enough points to understand basics of Serialization.