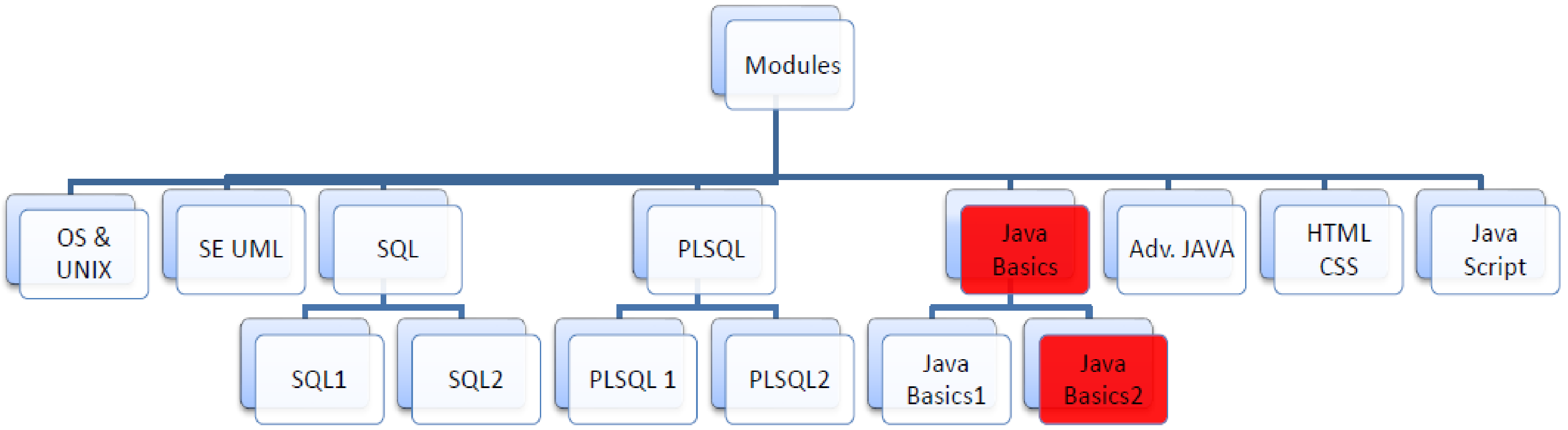
Accelerating Business Outcomes

Java Basics
Part 2

Module Overview

Purpose:

- The following module hierarchy presents the technical modules required to build the basic IT skills and acquaints you with relevant technology basics.
- The current module –**Java Basics2** (highlight in red) underwrites the usage of Multithreading, Collections and Input/output in Java Programming and will enable you to enhance one’s coding skills in multithreaded applications, collections and file handling.



*Recommended duration to complete Java Basics2 module: 12 hours

Module Objectives

By the end of this module, you will be able to:

Collections Framework:

- Define Java Collections API and their usage in real time applications
- Implement Collection Implementation classes i.e. List, Set, Queue and Map
- Sorting in Collections and use the Utility classes (i.e. Collections and Arrays)

Multithreading:

- Define the use of Multithreading and a Thread lifecycle in a Java application
- Write and Execute Multithreaded Java programs
- Use Thread Synchronization and Inter-thread communication in a Thread-safe application

IO Streams:

- Define Standard Input and Output Stream classes and use them in a Java application
- Define File Input and Output Stream classes and use them for data storage and retrieval
- Implement Object Serialization and Deserialization

Collections Framework

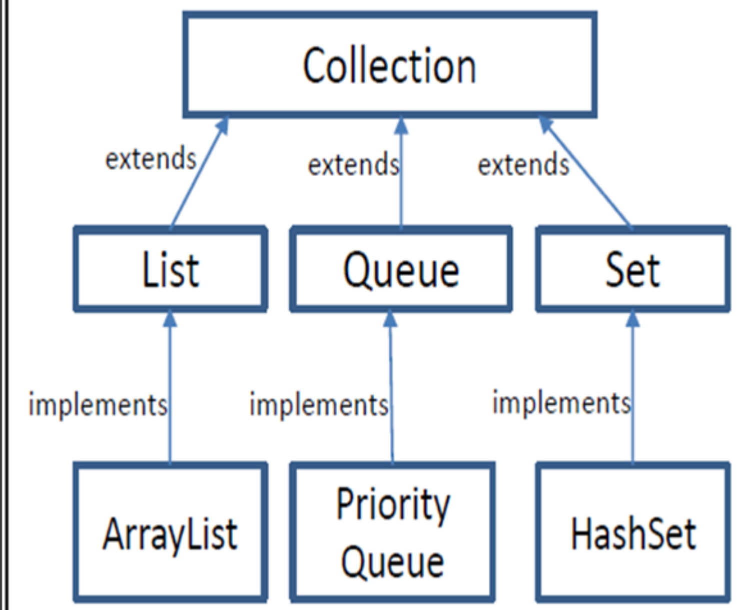
Define Java Collections API and their usage in real time applications

What is Collection?

- A *collection* is a group of data manipulated as a single object.

Usage of Collections in Java

- Collections in java is a framework that provides an architecture to store and manipulate the group of objects.
- All the operations that you perform on a data such as searching, sorting, insertion, manipulation, deletion etc. can be performed by Java Collections.
- List, Queue, Set interfaces extend Collection interface.
- ArrayList, PriorityQueue and HashSet class implements the List, Queue and Set interface respectively.



Collection Hierarchy in Java

Reference

- http://www.javatpoint.com/collections-in-java

Collections Framework

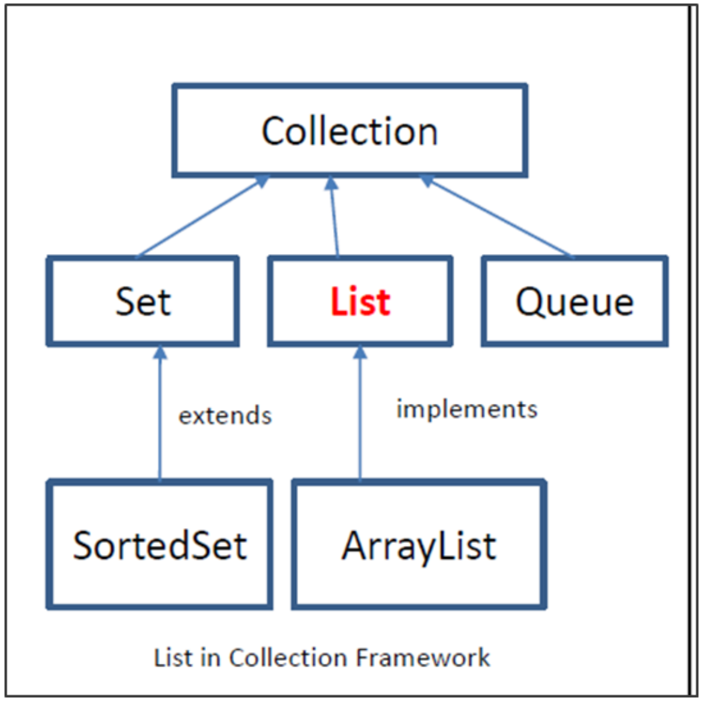
Implement ArrayList in Java Collections

What is ArrayList?

- **ArrayList** is a variable-length array of object references.

Usage of List:

- The **ArrayList** class extends **AbstractList** and implements the **List** interface. **ArrayList** supports dynamic arrays that can grow as needed.
- In Java, standard arrays are of a fixed length. After arrays are created, they cannot grow or shrink, which means that you must know in advance how many elements an array will hold.
- To handle this situation, the collections framework defines **ArrayList**.



List in Collection Framework

Reference

- http://www.javatpoint.com/collections-in-java

Collections Framework

Implement HashSet in Java Collections

What is HashSet?

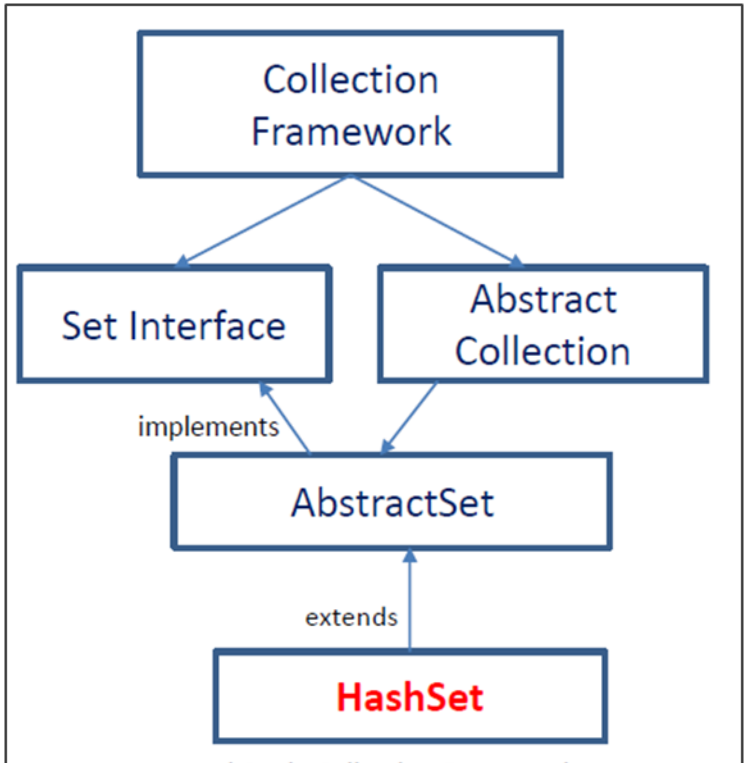
- **HashSet** is a collection variable-length array of object references.

Usage of HashSet

- **HashSet** extends **AbstractSet** and implements the **Set** interface. It creates a collection that uses a hash table for storage.
- A *hash table* stores information by using a mechanism called *hashing*.
- In *hashing,* the informational content of a key is used to determine a unique value, called its *hash code.*
- The advantage of hashing is that it allows the execution time of basic operations, such as **add()**, **contains()**, **remove()**, and **size()**, to remain constant even for large sets.

Reference

- http://www.javatpoint.com/collections-in-java



HashSet in Collection Framework

Collections Framework

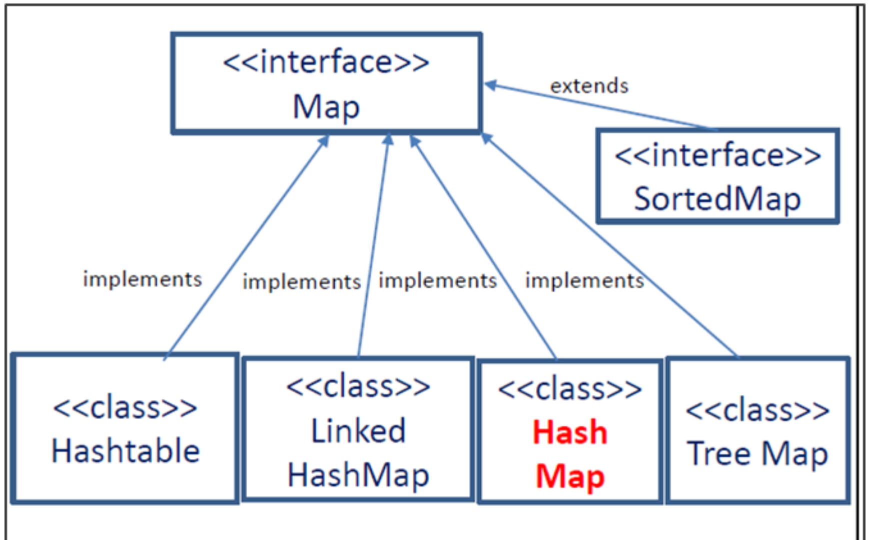
Implement HashMap in Java Collections

What is HashMap?

- The **HashMap** class uses a hash table to implement the **Map** interface.

Usage of HashMap

- **HashMap** implements **Map** and extends **AbstractMap**. It does not add any methods of its own.
- A *hash map* does *NOT* guarantee the order of its elements.
- The order in which elements are added to a hash map is not necessarily the order in which they are read by an iterator.



HashMap in Collection Framework

Reference

- http://www.javatpoint.com/collections-in-java

Collections Framework

Implement Comparable and Comparator in Java Collections

What is Comparable interface?

- Comparable interface is used to order the Collection objects.

Usage of Comparable interface

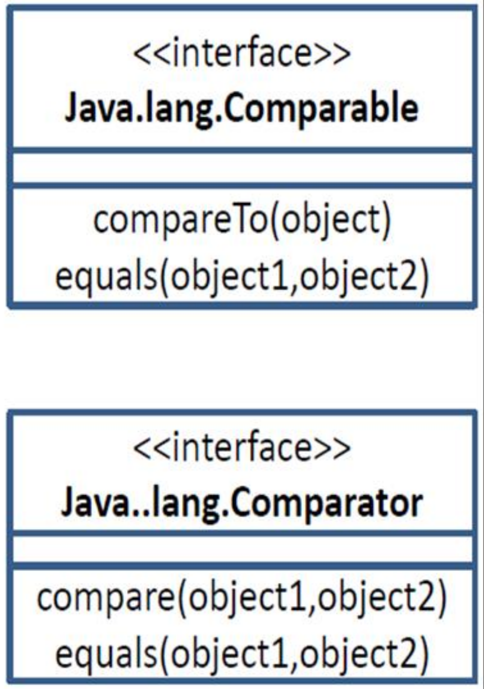
- This interface is found in *java.lang* package and contains only one method named *compareTo(Object).* It provides only single sorting sequence i.e. you can sort the elements on based on single data member only.

What is Comparator interface?

- Comparator interface is used to order the Collection objects.

Usage of Comparator interface

- This interface is found in *java.util* package and contains 2 methods *compare(Object obj1,Object obj2)* and *equals(Object element).*
- It provides multiple sorting sequence i.e. you can sort the elements based on any data member.



Comparable and Comparator in Collection framework

Reference

- http://www.javatpoint.com/Comparable-interface-in-collection-framework
- http://www.javatpoint.com/Comparator-interface-in-collection-framework

Multithreading

Implement Multithreading in Java

What is Multithreading?

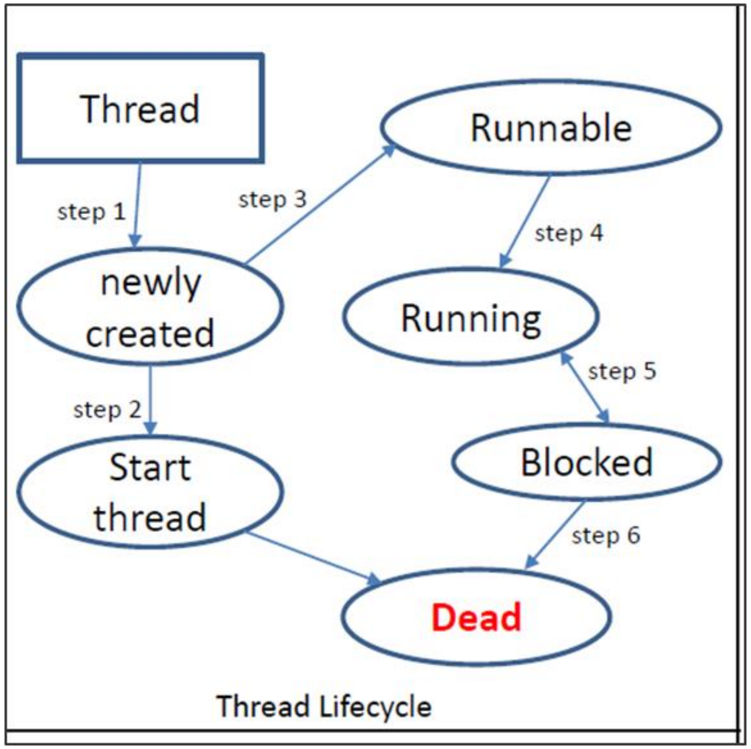
- Multithreading in java is a process of executing multiple threads simultaneously.
- Thread is basically a *lightweight sub-process*, a smallest unit of processing.

Usage of Multithreading

- Java Multithreading is mostly used in games, animation etc.

Advantage of Java Multithreading

- User can perform multiple operations at same time.
- Threads are independent so it doesn't affect other threads if exception occur in a single thread.
- Thread has a Lifecycle from *new* state to *dead* state.



Thread Lifecycle

Reference

- http://www.javatpoint.com/multithreading-in-java
- http://www.javatpoint.com/life-cycle-of-a-thread

Multithreading

Creating Threads in Java

How to create thread

There are *two* ways to create a thread:

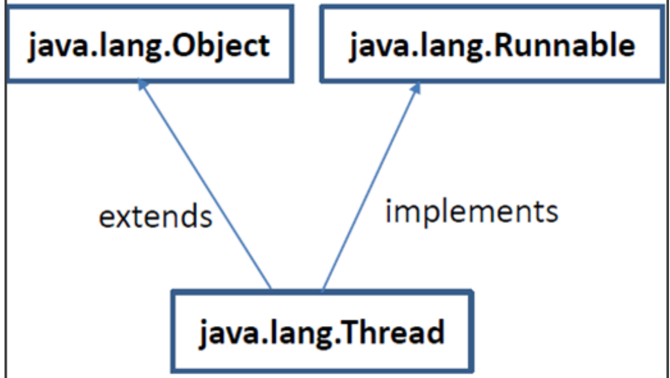
- By extending **Thread** class
- By implementing **Runnable** interface

Usage of Thread class:

- Thread class provides constructors and methods to create and perform operations on a thread.
- Thread class extends Object class and implements Runnable interface.

Usage of Runnable interface:

- The Runnable interface should be implemented by any class whose instances are intended to be executed by a thread. Runnable interface have only one method named run().
- public void run(): is used to perform action for a thread.



Thread class and Runnable interface

Reference

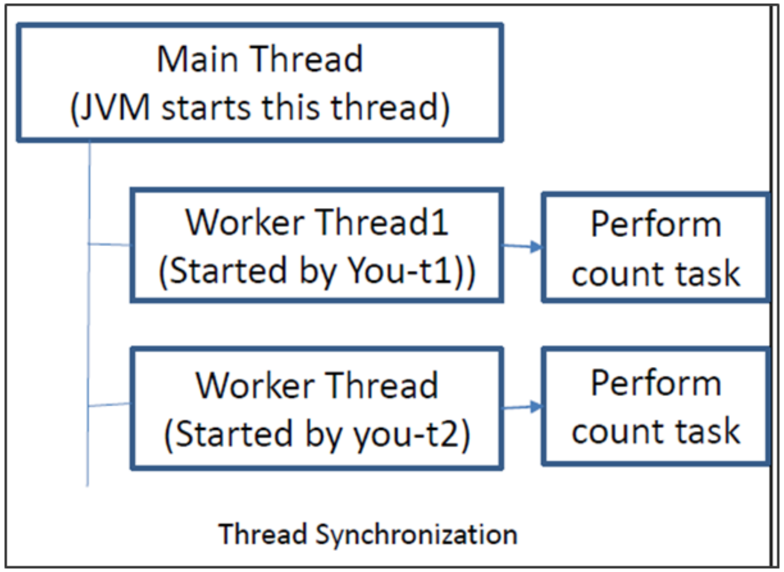
- http://www.javatpoint.com/creating-thread

Multithreading

Thread Synchronization in java

What is Synchronization?

- Synchronization in java is the capability of control the access of multiple threads to any shared resource.
- Java Synchronization is a better option where we want to allow only one thread to access the shared resource.



Thread Synchronization

Why use Synchronization?

The synchronization is mainly used to prevent:

- Thread interference
- Consistency problem

Reference

- http://www.javatpoint.com/synchronization-in-java

Input/output Stream in Java

IO Stream in java

What is a Stream?

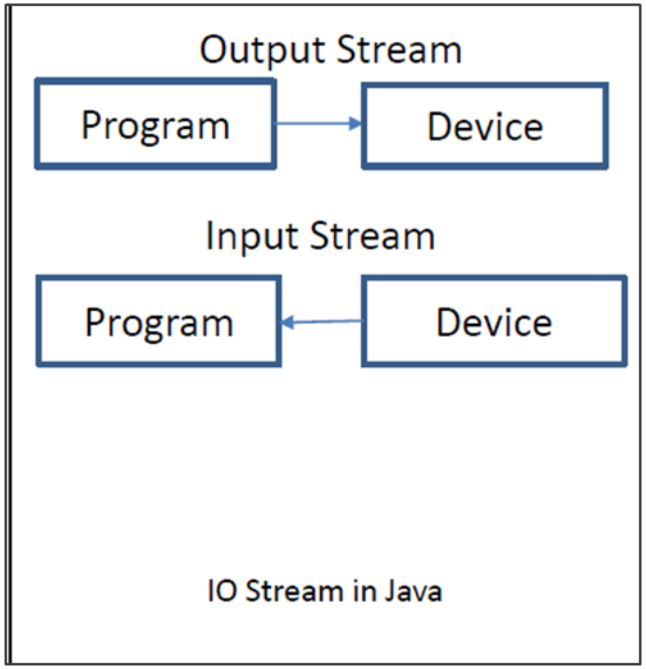
- A stream is a sequence of data. In Java a stream is composed of bytes. It is called a stream because it's like a stream of water that continues to flow.

Three streams are created for us automatically:

1) ***System.out***: standard output stream

2) ***System.in***: standard input stream

3) ***System.err***: standard error



IO Stream in Java

Usage of Stream:

Java encapsulates Stream under ***java.io*** package. Java defines *two* types of streams:

1. **Byte Stream**: It provides a convenient means for handling input and output of byte.

2. **Character Stream**: It provides a convenient means for handling input and output of characters.

Reference

- http://www.javatpoint.com/java-io

Input/output Stream in Java

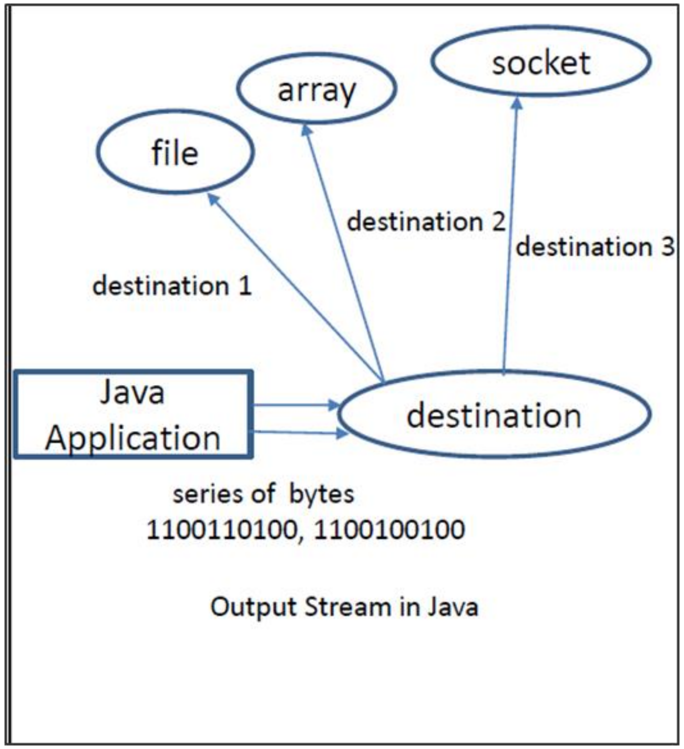
OutputStream in java

What is OutputStream?

- The **Java.io.OutputStream** class is the superclass of all classes representing an output stream of bytes.
- Applications that need to define a subclass of OutputStream must always provide at least a method that writes one byte of output.

Usage of OutputStream

```
OutputStream output = new
FileOutputStream("c:\\data\\output-text.txt");

while(moreData) {

intdata = getMoreData();

output.write(data);

}

output.close();
```



Reference

- http://tutorials.jenkov.com/java-io/outputstream.html

Input/output Stream in Java

InputStream in java

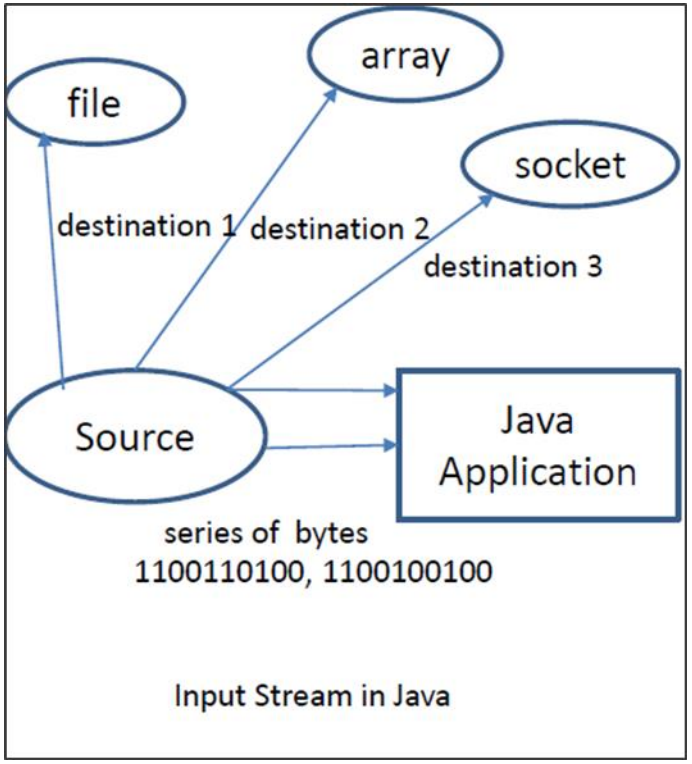
What is InputStream?

The **Java.io.InputStream** class is the superclass of all classes representing an input stream of bytes.

Applications that need to define a subclass of InputStream must always provide a method that returns the next byte of input.

Usage of InputStream

```
InputStream input = new FileInputStream("c:\\data\\input-text.txt");
intdata = input.read();
while(data != -1) {
//do something with data...
doSomethingWithData(data);
data = input.read();
}
input.close();
```



Input Stream in Java

Reference

- http://tutorials.jenkov.com/java-io/fileinputstream.html

Input/output Stream in Java

Serialization and Deserialization in java

What is Serialization?

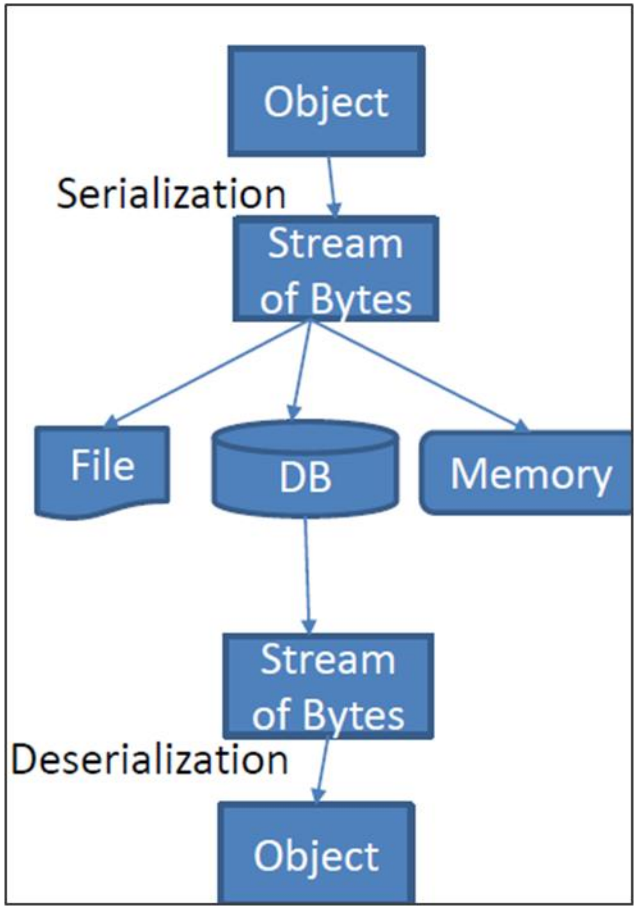
- Serialization is a process of converting an object into a sequence of bytes which can be persisted to a disk or database or can be sent through streams.

What is Deserialization?

- The reverse process of creating object from sequence of bytes is called deserialization.

Java Serializable Interface

- **java.io.Serializable** is a marker interface and has no fields or methods to implement.
- Serialization process is implemented by **ObjectInputStream** and **ObjectOutputStream**, so all we need is a wrapper over them to either save it to file or send it over the network.



Reference

- http://www.studytonight.com/java/serialization-and-deserialization.php

Additional References

To explore more on the subject, refer the below links and books:

Links:

Collection Frame Work

- http://docs.oracle.com/javase/tutorial/collections/index.html
- http://tutorials.jenkov.com/java-collections/index.html

Multi threading

- http://docs.oracle.com/javase/tutorial/essential/concurrency/
- http://tutorials.jenkov.com/java-concurrency/index.html

IO streams

- http://docs.oracle.com/javase/tutorial/essential/io/
- http://tutorials.jenkov.com/java-io/file.html

Books:

- Head First Java
- Java Complete Reference

Self Check

Instructions to write Self Evaluation Sheet:

Open the excel sheet, refer Java Basics – Part 2, write down the solutions for all questions, save a local copy in your machine

Lab Assignment

- Refer ***Assignment Document*** for this module to proceed with **Lab Assignment.**
- Do **submit the Solutions** for the given assignment and refer the ***Participant guide*** for submission procedure.

Module Summary

Now that you have completed this module, you will be able to:

- Define Collections API and its usage in Java
- Write a Java program with CRUD operations
- Explain –Multithreading in Java
- Write a multithread java program with concurrent actions
- Write and Read a stream of data using files
- Write and Read a Character set using files
- Serialize and De-serialize the object using files

Thank you