Lambda Expressions & Java I/O

NucleusTeq

Things to discuss....

- Java Lambda Expressions
- Functional Interface
- Lambda Parameters
- Lambda as an Object
- Lambda Variable Capture
- Method Reference as Lambdas

Java Lambda Expressions

- Lambda expressions are fundamental to functional programming in Java.
- It is an anonymous function that doesn't have a name and doesn't belong to any class.
- It provides a clear and concise way to represent a method interface via an expression.
- It provides the implementation of a functional interface & simplifies the software development.

Java Lambda Expressions

Syntax :

parameter -> expression body

- The arrow operator (->) is used to divide lambda into its parameters and body.
- Few optional things:
  - Type Declarations
  - Parenthesis Around Parameters
  - Curly Braces
  - return keyword
- Lambda expressions works with functional interfaces only.

Functional Interface

- Functional Interface is an interface that contains exactly one abstract method
- It can have any number of default or static methods along with object class methods
- Java provides predefined functional interfaces to deal with functional programming
- Some of the examples of functional interfaces :
  - Runnable
  - ActionListener
  - Comparable

Lambda Parameters

- Lambda parameters can take parameters just like methods.
- It can take zero, one or multiple parameters.
- Some examples :
    - () -> System.out.println(“Zero parameter passed”);
    - (param) -> System.out.println(“One parameter “+ param);
    - (param1, param2) -> System.out.println(“Multiple parameters “+ param1 + “,“ + param2)

Lambda as an Object

A java lambda expression is essentially an object that can be assigned to a variable and passed around.

Lambda Variable Capture

- We can use local variable, instance variable and static variables in lambda expressions.

Method Reference as Lambdas

- Static Method Reference
- Parameter Method Reference
- Instance Method Reference
- Constructor Method Reference

Thank you

Any questions ?

JAVA IO

NucleusTeq

JAVA IO

Objectives:

1. Java IO Overview
2. Streams
3. ByteStream I/O Demo
4. CharacterStream I/O Demo
5. Overwriting a File
6. Closing a File
7. Serialization & Deserialization

Java I/O Streams Overview

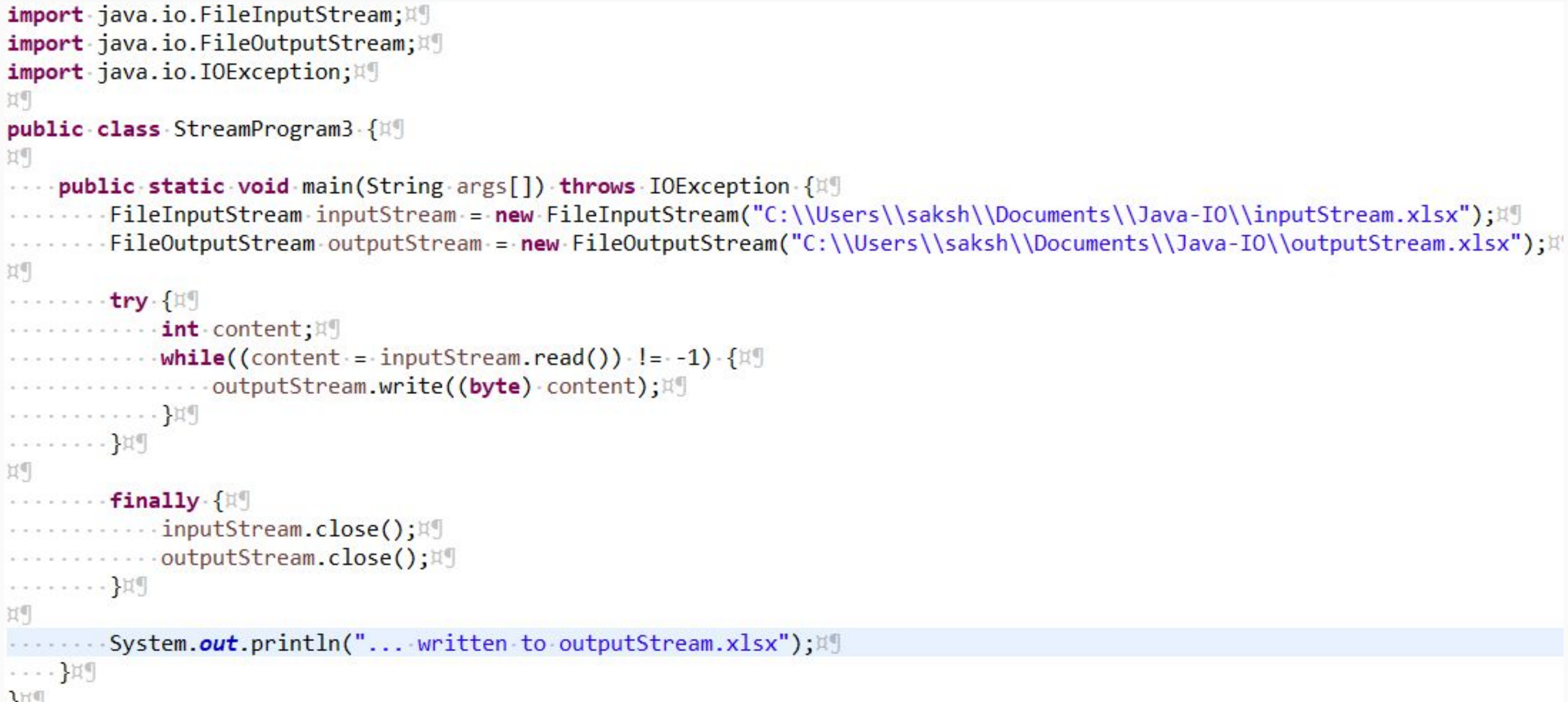
- I/O = Input/Output
- Java I/O (Input/Output) Streams facilitate data input and output in Java programs.
- Key purpose: Handling data from various sources such as files, network connections, or other streams.
- Advantages:
  - Reading & Writing Files
  - Filtering Data
  - Compress & Decompress data

Streams

- **Definition**: An ordered sequence of bytes of indeterminate length. This includes both input & output streams.
- **Types of Streams**
    - Byte streams (InputStream, OutputStream)
    - Character streams (Reader, Writer).

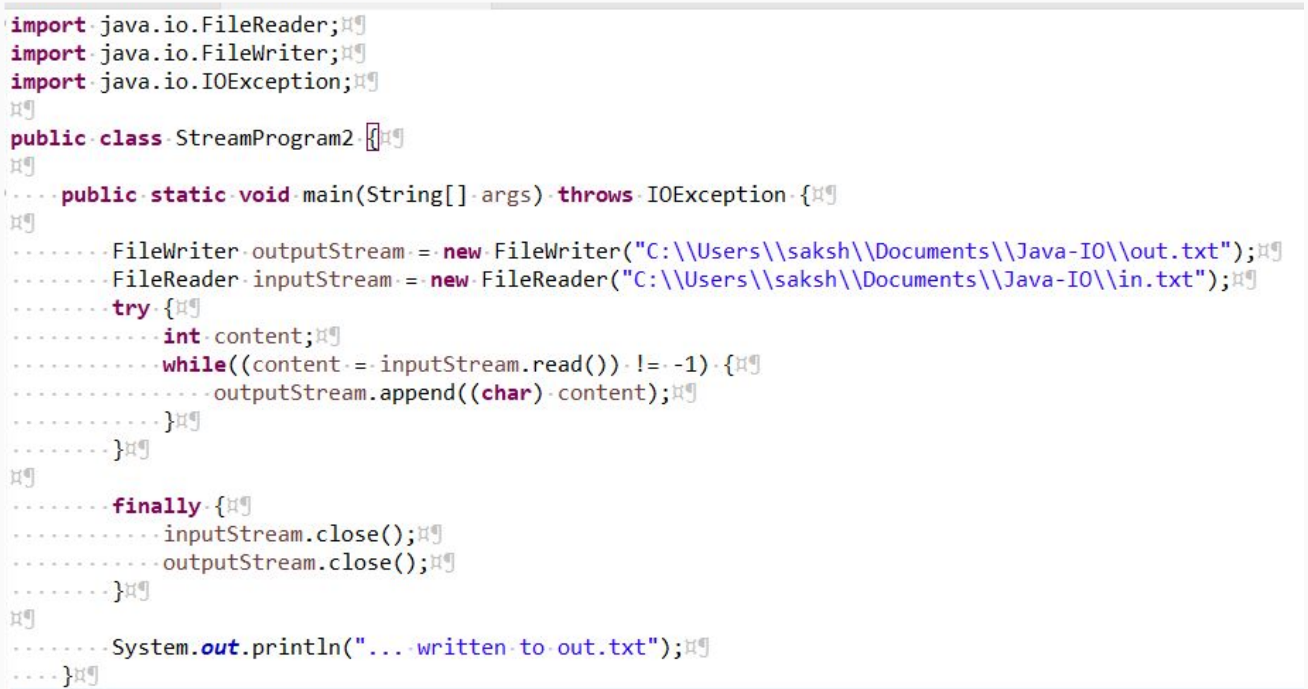
- **Input stream**: reads data from source
    - `System.in` is an input stream
- **Output stream**: writes data into the destination
    - `System.out` is an output stream
- A stream connects a program to an I/O object
    - `System.out` connects a program to the screen
    - `System.in` connects a program to the keyboard

I/O Byte Streams Demo



```
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;

public class StreamProgram3 {

    public static void main(String args[]) throws IOException {
        FileInputStream inputStream = new FileInputStream("C:\\Users\\saksh\\Documents\\Java-IO\\inputStream.xlsx");
        FileOutputStream outputStream = new FileOutputStream("C:\\Users\\saksh\\Documents\\Java-IO\\outputStream.xlsx");

        try {
            int content;
            while((content = inputStream.read()) != -1) {
                outputStream.write((byte) content);
            }
        }

        finally {
            inputStream.close();
            outputStream.close();
        }

        System.out.println("... written to outputStream.xlsx");
    }
}
```

I/O Character Streams Demo



```java
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;

public class StreamProgram2 {

    public static void main(String[] args) throws IOException {

        FileWriter outputStream = new FileWriter("C:\\Users\\saksh\\Documents\\Java-IO\\out.txt");
        FileReader inputStream = new FileReader("C:\\Users\\saksh\\Documents\\Java-IO\\in.txt");
        try {
            int content;
            while((content = inputStream.read()) != -1) {
                outputStream.append((char) content);
            }
        }

        finally {
            inputStream.close();
            outputStream.close();
        }

        System.out.println("... written to out.txt");
    }
```

Overwriting a File

- Opening an output file creates an empty file
- Opening an output file creates a new file if it does not already exist
- Opening an output file that already exists eliminates the old file and creates a new, empty one data in the original file is lost
- To see how to check for existence of a file, see the section of the text that discusses the `File` class (later slides).

Closing a File

- An output file should be closed when you are done writing to it (and an input file should be closed when you are done reading from it).
- Use the `close` method of the class `PrintWriter` (`BufferedReader` also has a `close` method).
- For example, to close the file opened in the previous example:

```
outputStream.close();
```

- If a program ends normally it will close any files that are open.
- It's important to close a file to save memory.

Text File Input

•To open a text file for input: connect a text file to a stream for reading

–Goal: a `BufferedReader` object,

•which uses `FileReader` to open a text file

–`FileReader` " connects" `BufferedReader` to the text file

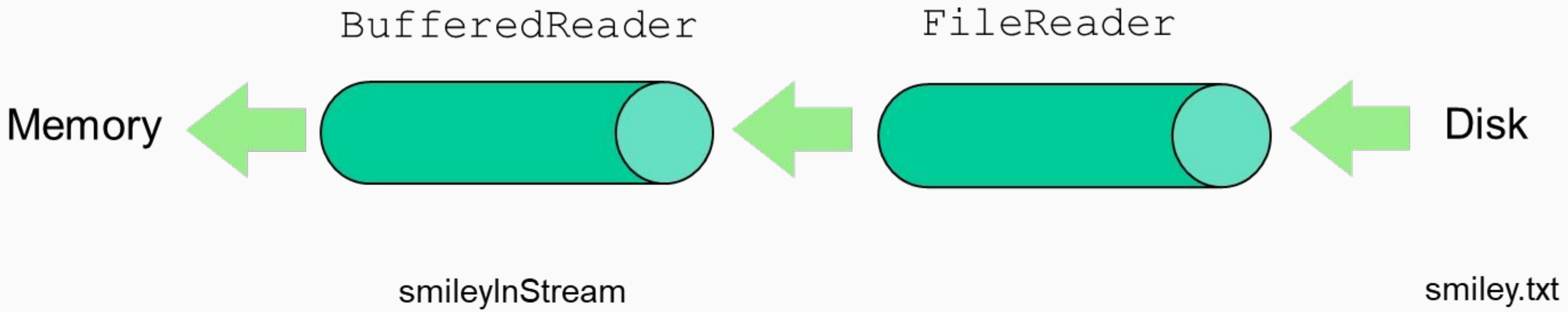
•For example:

```
        BufferedReader smileyInStream =
          new BufferedReader(new FileReader("smiley.txt"));
```

•Similarly, the long way:

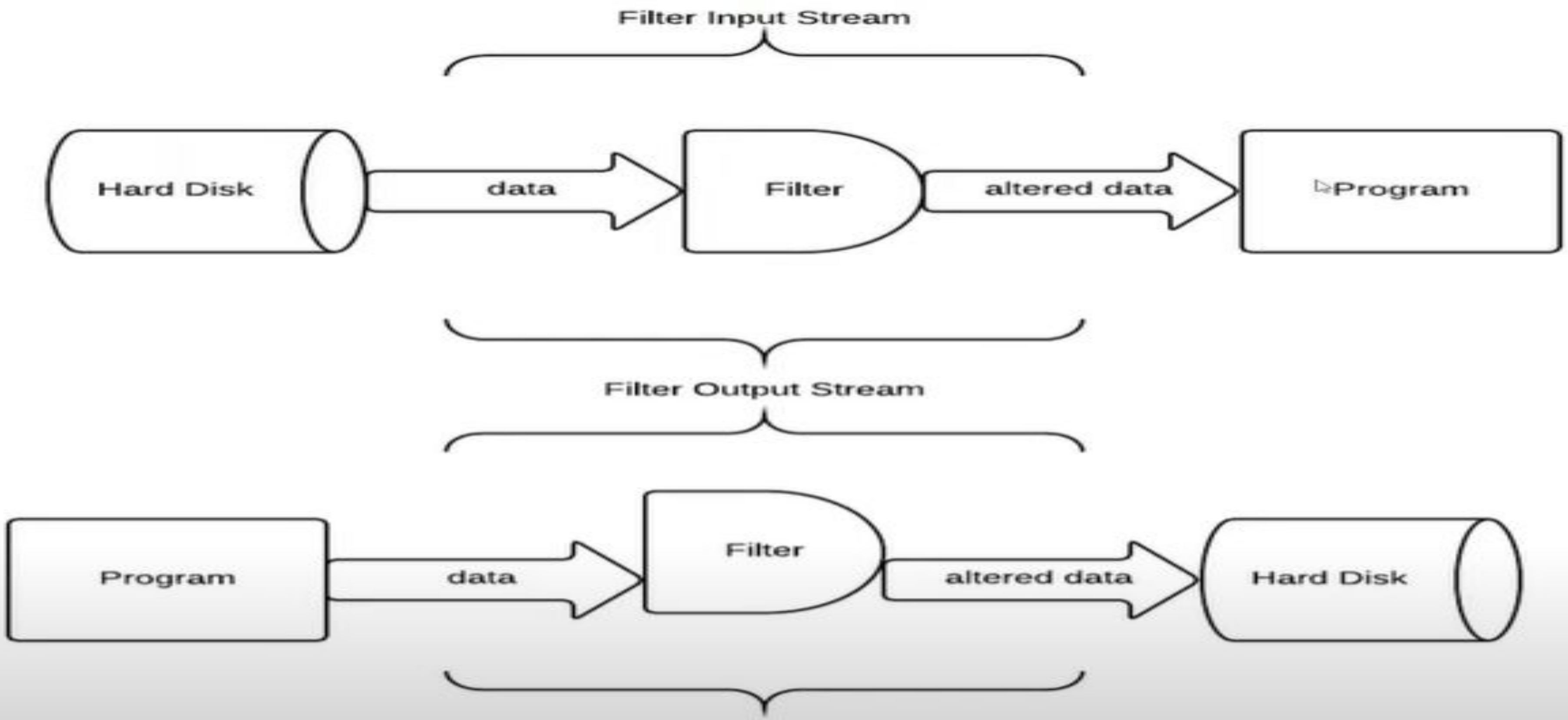
```
        FileReader s = new FileReader("smiley.txt");
        BufferedReader smileyInStream = new
        BufferedReader(s);
```

Input File Streams

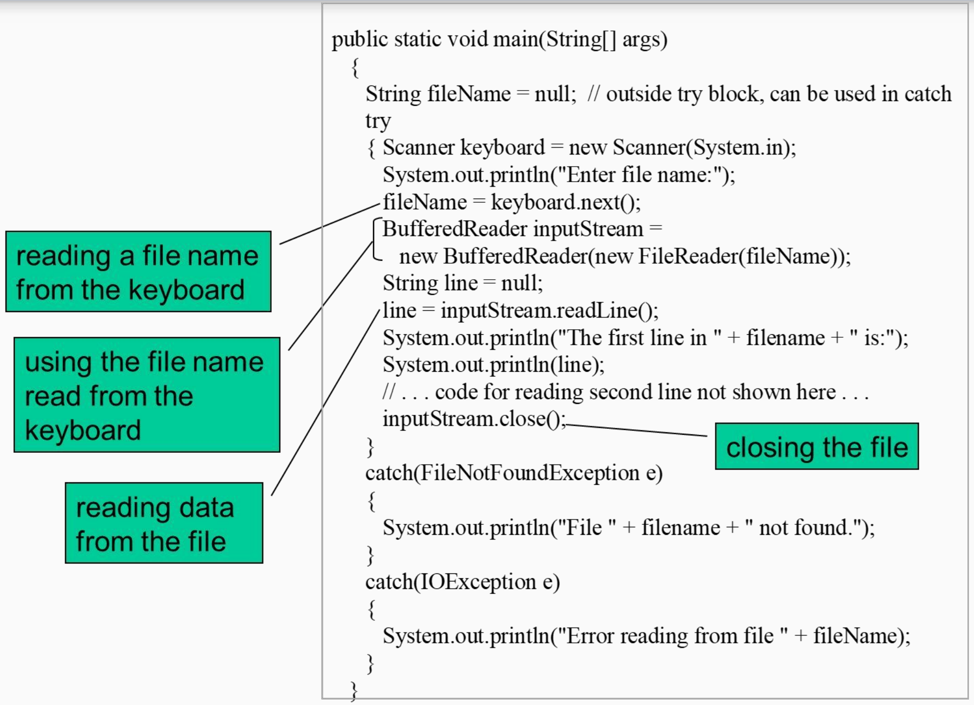


BufferedReader smileyInStream = new BufferedReader(new FileReader(“smiley.txt”));

Filter Streams



Example:
Reading a File Name from the Keyboard



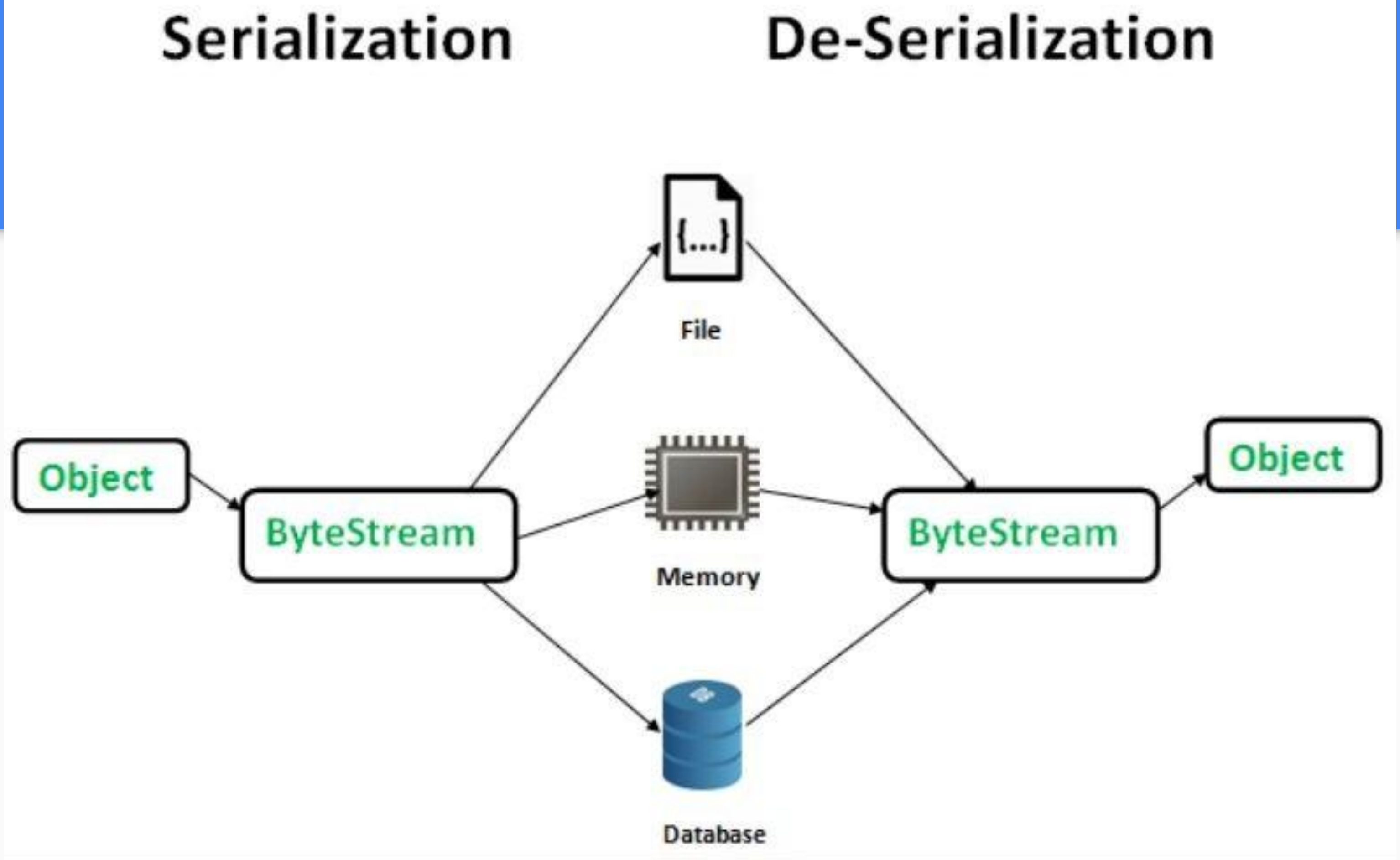
```
public static void main(String[] args)
  {
    String fileName = null;  // outside try block, can be used in catch
    try
    { Scanner keyboard = new Scanner(System.in);
      System.out.println("Enter file name:");
      fileName = keyboard.next();
      BufferedReader inputStream =
        new BufferedReader(new FileReader(fileName));
      String line = null;
      line = inputStream.readLine();
      System.out.println("The first line in " + filename + " is:");
      System.out.println(line);
      // . . . code for reading second line not shown here . . .
      inputStream.close();
    }
    catch(FileNotFoundException e)
    {
      System.out.println("File " + filename + " not found.");
    }
    catch(IOException e)
    {
      System.out.println("Error reading from file " + fileName);
    }
  }
```

reading a file name from the keyboard
using the file name read from the keyboard
reading data from the file
closing the file

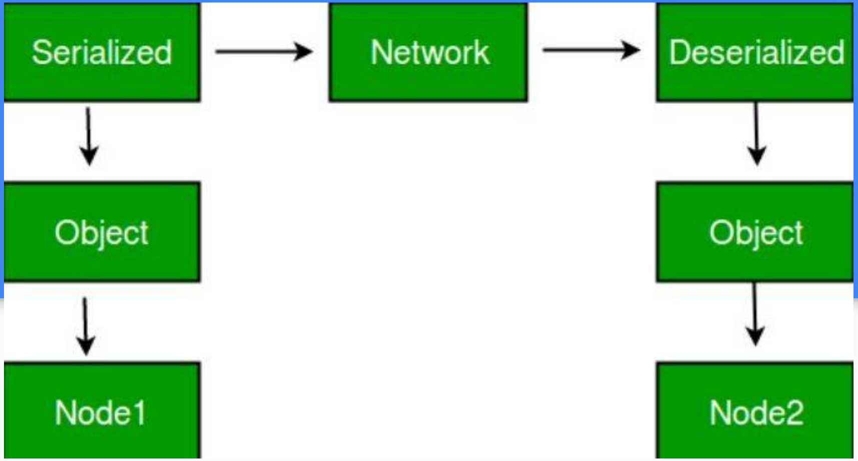
Serialization and DeSerialization

Serialization is a mechanism of converting the state of an object into a byte stream.

Deserialization is the reverse process where the byte stream is used to recreate the actual Java object in memory. This mechanism is used to persist the object.


Serialization
De-Serialization
Object
ByteStream
File
{...}
Memory
Database
ByteStream
Object

Advantages



1. To save/persist state of an object.
2. To travel an object across a network.

Serializable is a marker interface (has no data member and method). It is used to "mark" Java classes so that the objects of these classes may get a certain capability.

- The Cloneable and Remote are also marker interfaces.
- The Serializable interface must be implemented by the class whose object needs to be persisted.
- The String class and all the wrapper classes implement the java.io.Serializable interface by default.

Points to remember

1. If a parent class has implemented Serializable interface then child class doesn’t need to implement it but vice-versa is not true.
2. Only non-static data members are saved via Serialization process.
3. Static data members and transient data members are not saved via Serialization process.So, if you don’t want to save value of a non-static data member then make it transient.
4. Constructor of object is never called when an object is deserialized.
5. Associated objects must be implementing Serializable interface.

```java
import java.io.Serializable;

public class Student implements

Serializable{

 int id;

 String name;

 public Student(int id, String name) {

  this.id = id;

  this.name = name;

 }

}
```

Student class implements Serializable interface. Now its objects can be converted into stream.

```java
import java.io.*;

class Persist{

 public static void main(String args[]){

  try{

  //Creating the object

  Student s1 =new Student(211,"ravi");
  //Creating stream and writing the object

  FileOutputStream fout=new FileOutputStream("f.txt");

  ObjectOutputStream out=new ObjectOutputStream(fout);

  out.writeObject(s1);


  out.flush();

  //closing the stream

  out.close();

  System.out.println("success");

  }catch(Exception e){System.out.println(e);}

 }

}
```

Tasks For Everyone:

- Explore the different type of subclasses in InputStream & OutputStream.
- Explore the FilterInputStream & FilterOutputStream.
- Create a program to "invert" the contents of a text file: create a file with the same name ending in ".txt" and containing the same lines as the original file but in reverse order (the first line will be the last one, the second will be the penultimate, and so on, until the last line of the original file, which should appear in the first position of the resulting file.)
- Write a Java program to get a list of all file/directory names from the given.
- Write a program to copy contents of one txt file to other txt file.

Thank you

Any questions ?

Lambda Expressions Questions

- Write a Java program to implement a lambda expression to replace vowels with ‘#’ in a given string.
- Create a functional interface with name ‘Shape’ and a method ‘area’. Write a program to implement these interface for shapes rectangle, square, circle, cube, sphere, cylinder and find out its area. All the implementation should be within a single class.