MODULE 3

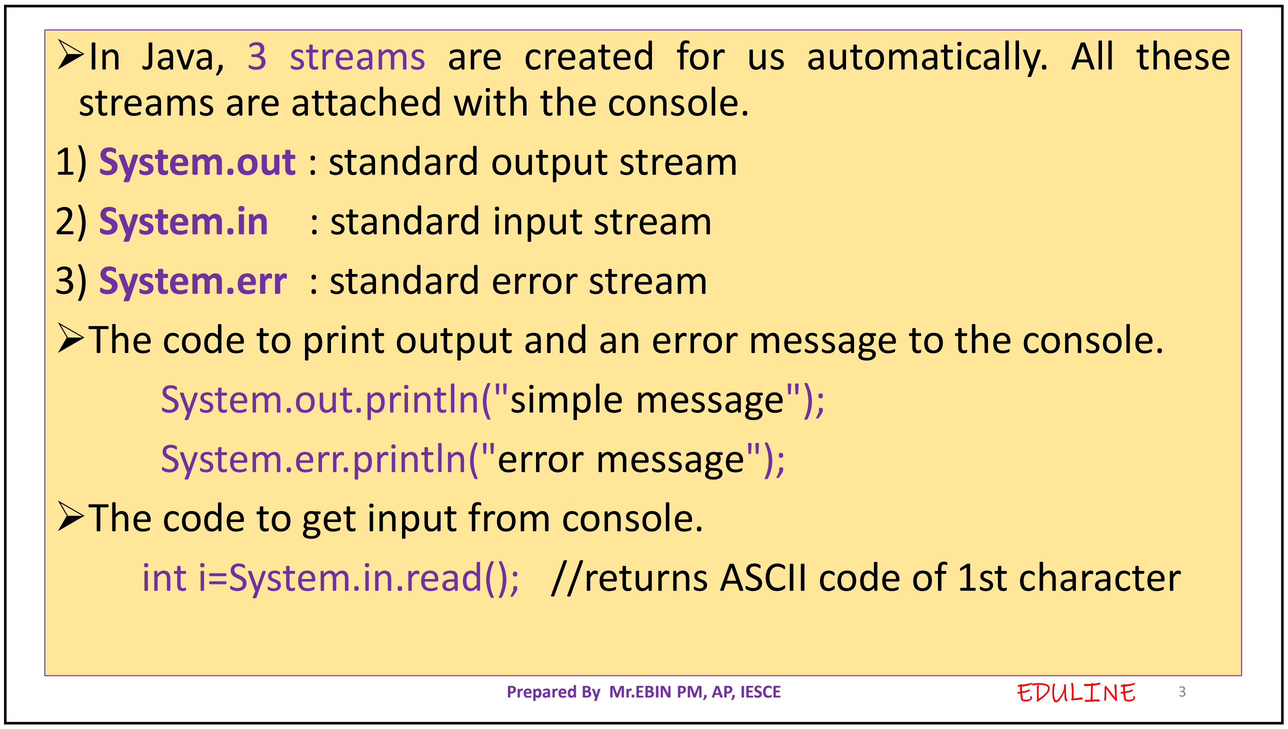
CHAPTER 2 JAVA INPUT OUTPUT (I/O) & FILES

STREAM

- Java I/O (Input and Output) is used to process the input and produce the output.
- Java uses the concept of a stream to make I/O operation fast. The java.io package contains all the classes required for input and output operations.
- We can perform file handling in Java by Java I/O API.

STREAM

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



➢In Java, 3 streams are created for us automatically. All these streams are attached with the console.

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

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

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

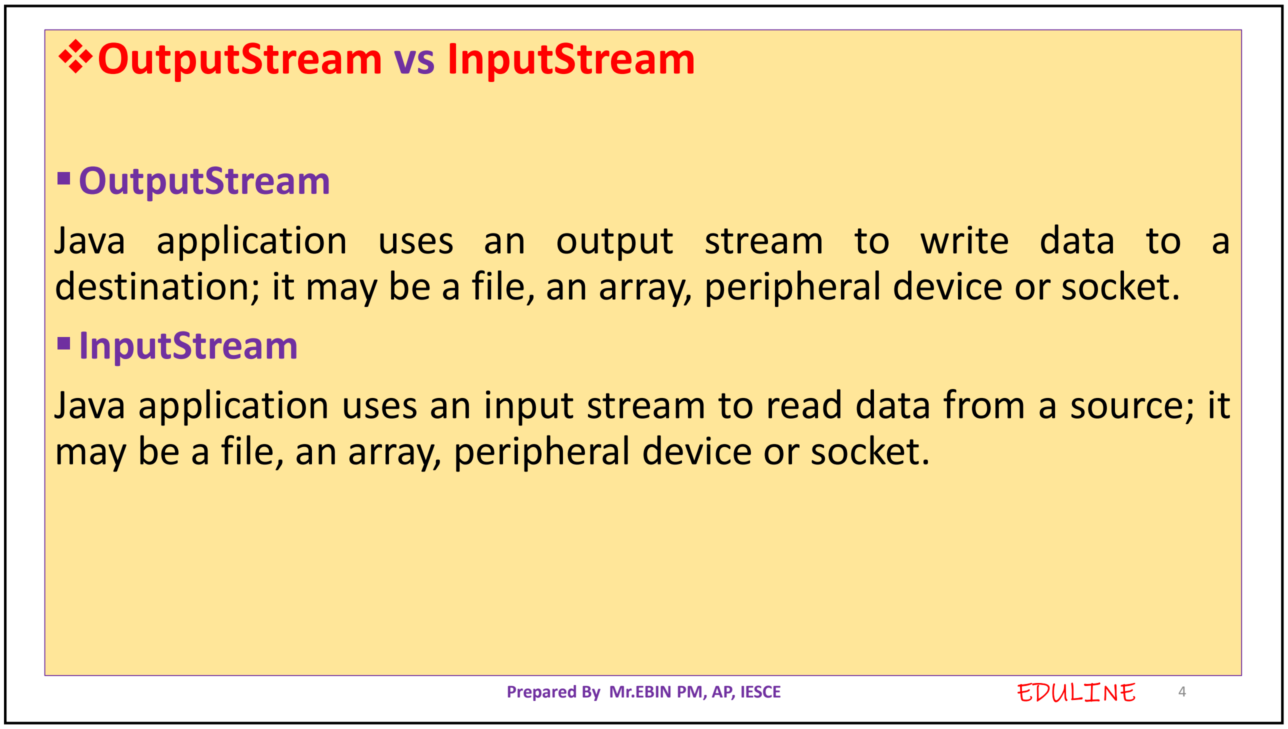
➢The code to print output and an error message to the console.

```
System.out.println("simple message");
System.err.println("error message");
```

➢The code to get input from console.

```
int i=System.in.read();   //returns ASCII code of 1st character
```

Prepared By Mr.EBIN PM, AP, IESCE EDULINE 3



❖OutputStream vs InputStream

▪ **OutputStream**

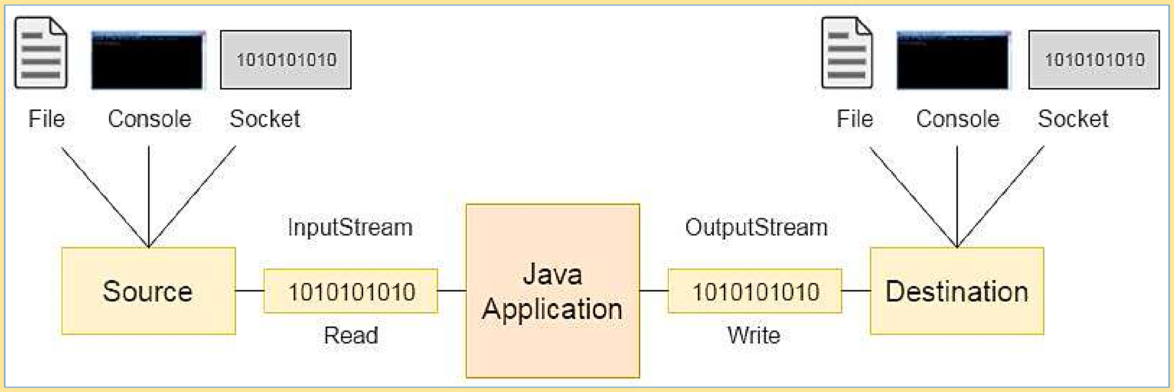
Java application uses an output stream to write data to a destination; it may be a file, an array, peripheral device or socket.

▪ **InputStream**

Java application uses an input stream to read data from a source; it may be a file, an array, peripheral device or socket.

Prepared By Mr.EBIN PM, AP, IESCE EDULINE 4

❖The working of Java OutputStream and InputStream



Prepared By Mr.EBIN PM, AP, IESCE EDULINE 5

❖OutputStream class

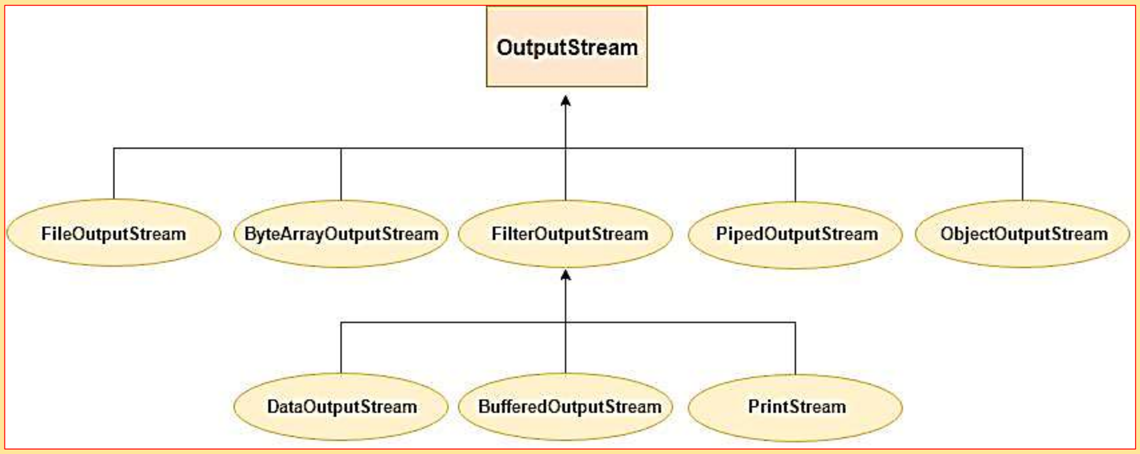
- OutputStream class is an abstract class.
- It is the superclass of all classes representing an output stream of bytes. An output stream accepts output bytes and sends them to some sink.

Useful methods of OutputStream

Method	Description
1) public void write(int)throws IOException	is used to write a byte to the current output stream.
2) public void write(byte[])throws IOException	is used to write an array of byte to the current output stream.
3) public void flush()throws IOException	flushes the current output stream.
4) public void close()throws IOException	is used to close the current output stream.

Prepared By Mr.EBIN PM, AP, IESCE EDULINE 6

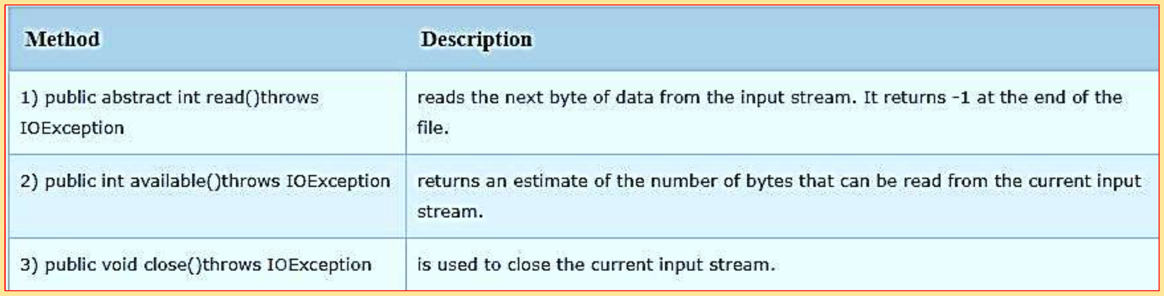
❖OutputStream Hierarchy



❖InputStream class

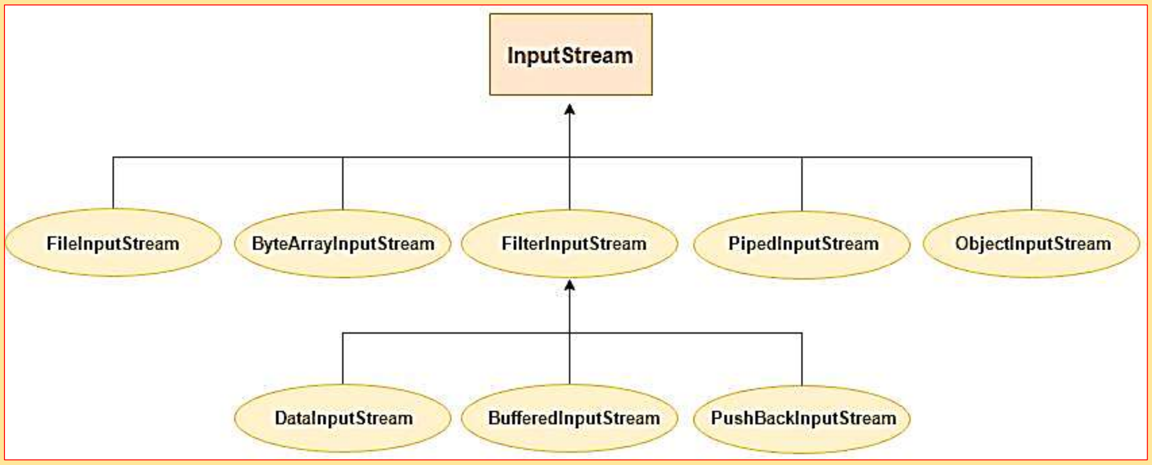
- InputStream class is an abstract class.
- It is the superclass of all classes representing an input stream of bytes.

Useful methods of InputStream



Method	Description
1) public abstract int read()throws IOException	reads the next byte of data from the input stream. It returns -1 at the end of the file.
2) public int available()throws IOException	returns an estimate of the number of bytes that can be read from the current input stream.
3) public void close()throws IOException	is used to close the current input stream.

❖InputStream Hierarchy



Prepared By Mr.EBIN PM, AP, IESCE EDULINE 9

READING CONSOLE INPUT

➢In Java, there are three different ways for reading input from the user in the command line environment(console).

1.Using Buffered Reader Class

- This is the Java classical method to take input, Introduced in JDK1.0.
- This method is used by wrapping the System.in (standard input stream) in an InputStreamReader which is wrapped in a BufferedReader, we can read input from the user in the command line.
- **Advantage** - The input is buffered for efficient reading
- **Drawback** - The wrapping code is hard to remember.

Prepared By Mr.EBIN PM, AP, IESCE EDULINE 10

```
// Java program to demonstrate BufferedReader
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
public class Test
{
        public static void main(String[] args) throws IOException
        {
                //Enter data using BufferReader
                BufferedReader reader = new BufferedReader(new
                                        InputStreamReader(System.in));
                // Reading data using readLine
                String name = reader.readLine();
                // Printing the read line
                System.out.println(name);
        }
}
```

2. Using Scanner Class

- This is probably the most preferred method to take input.
- The main purpose of the Scanner class is to parse primitive types and strings using regular expressions, however it is also can be used to read input from the user in the command line.

Advantages:

- Convenient methods for parsing primitives (nextInt(), nextFloat(), …) from the tokenized input.
- Regular expressions can be used to find tokens.

Drawback:

- The reading methods are not synchronized

```
// Java program to demonstrate working of Scanner in Java
import java.util.Scanner;
class GetInputFromUser
{
        public static void main(String args[])
        {
                // Using Scanner for Getting Input from User
                Scanner in = new Scanner(System.in);
                    String s = in.nextLine();
                System.out.println("You entered string "+s);
                int a = in.nextInt();
                System.out.println("You entered integer "+a);
                float b = in.nextFloat();
                System.out.println("You entered float "+b);
        }
}
```

Input:

```
HelloStudents
12
3.4
```

Output:

```
You entered string
HelloStudents
You entered integer 12
You entered float 3.4
```

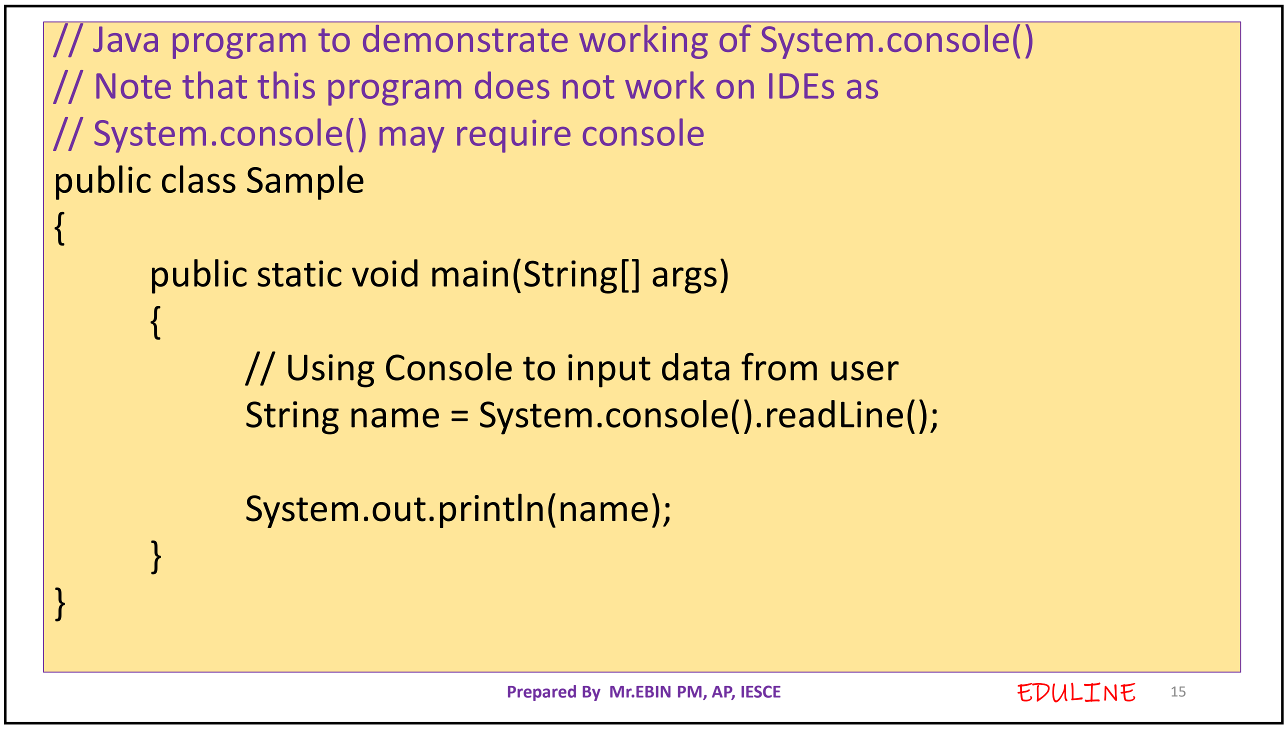
3. Using Console Class

- It has been becoming a preferred way for reading user's input from the command line.
- In addition, it can be used for reading password-like input without echoing the characters entered by the user; the format string syntax can also be used (like System.out.printf()).

Advantages:

- Reading password without echoing the entered characters.
- Reading methods are synchronized.
- Format string syntax can be used.

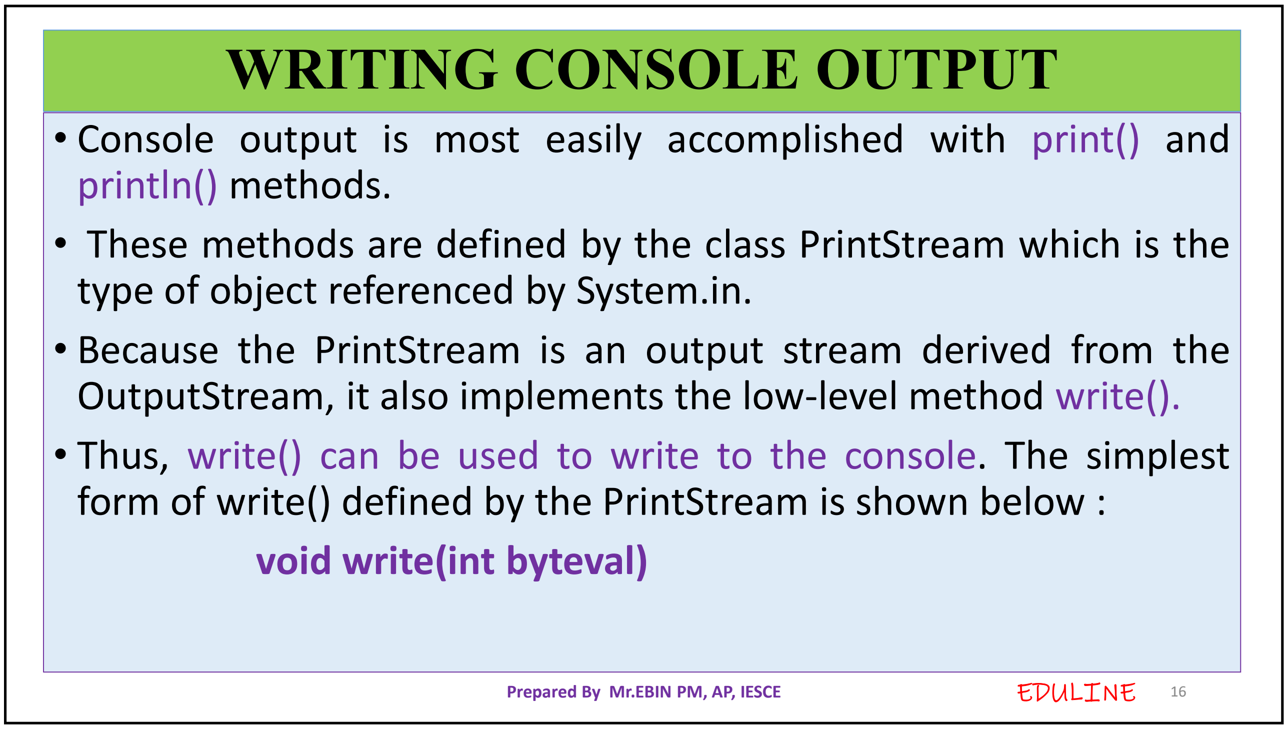
Drawback: Does not work in non-interactive environment (such as in an IDE).



```
// Java program to demonstrate working of System.console()
// Note that this program does not work on IDEs as
// System.console() may require console
public class Sample
{
        public static void main(String[] args)
        {
                // Using Console to input data from user
                String name = System.console().readLine();

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

Prepared By Mr.EBIN PM, AP, IESCE EDULINE 15



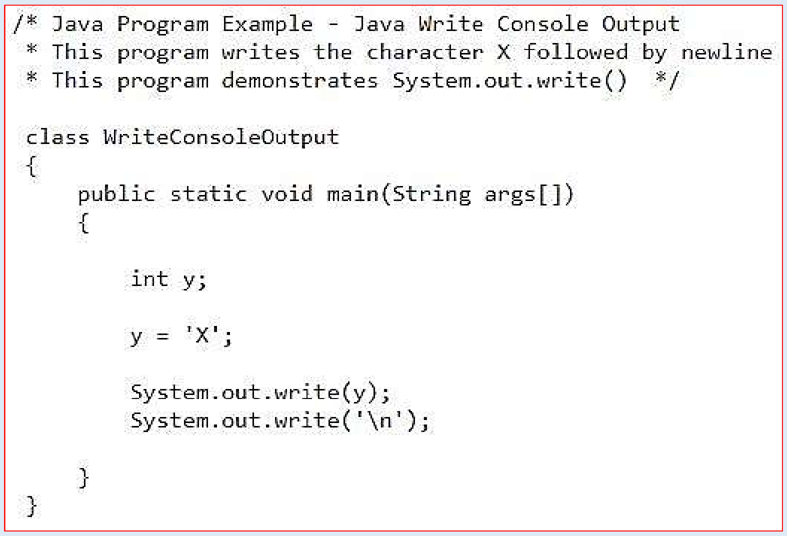
WRITING CONSOLE OUTPUT

- Console output is most easily accomplished with print() and println() methods.
- These methods are defined by the class PrintStream which is the type of object referenced by System.in.
- Because the PrintStream is an output stream derived from the OutputStream, it also implements the low-level method write().
- Thus, write() can be used to write to the console. The simplest form of write() defined by the PrintStream is shown below :

void write(int byteval)

Prepared By Mr.EBIN PM, AP, IESCE EDULINE 16

- Following is a short example that uses write() to output the character 'X' followed by a newline to the screen:



```
/* Java Program Example - Java Write Console Output
 * This program writes the character X followed by newline
 * This program demonstrates System.out.write()  */

 class WriteConsoleOutput
 {
     public static void main(String args[])
     {

         int y;

         y = 'X';

         System.out.write(y);
         System.out.write('\n');

     }
 }
```

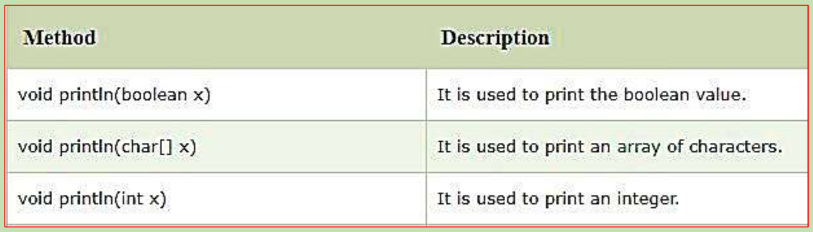
PrintWriter CLASS

- Java PrintWriter class is the implementation of Writer class.
- It is used to print the formatted representation of objects to the text-output stream.

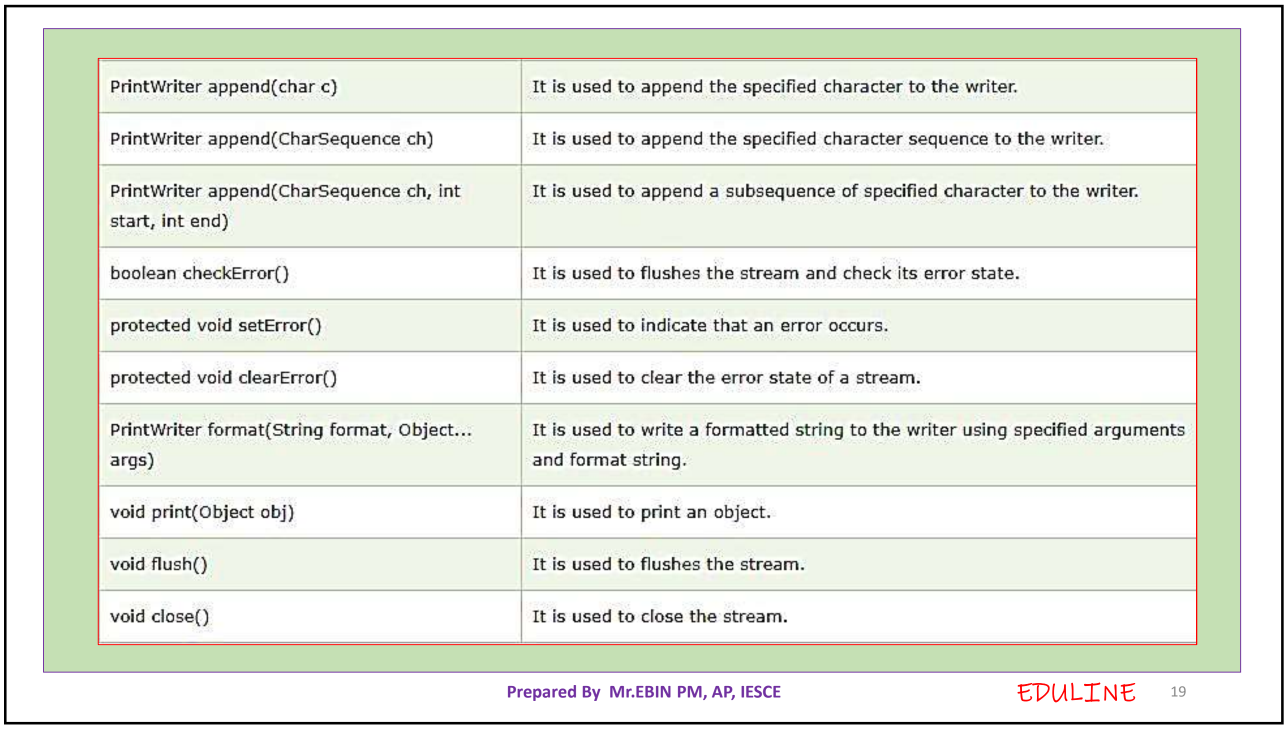
Class declaration

public class PrintWriter extends Writer

Methods of PrintWriter class

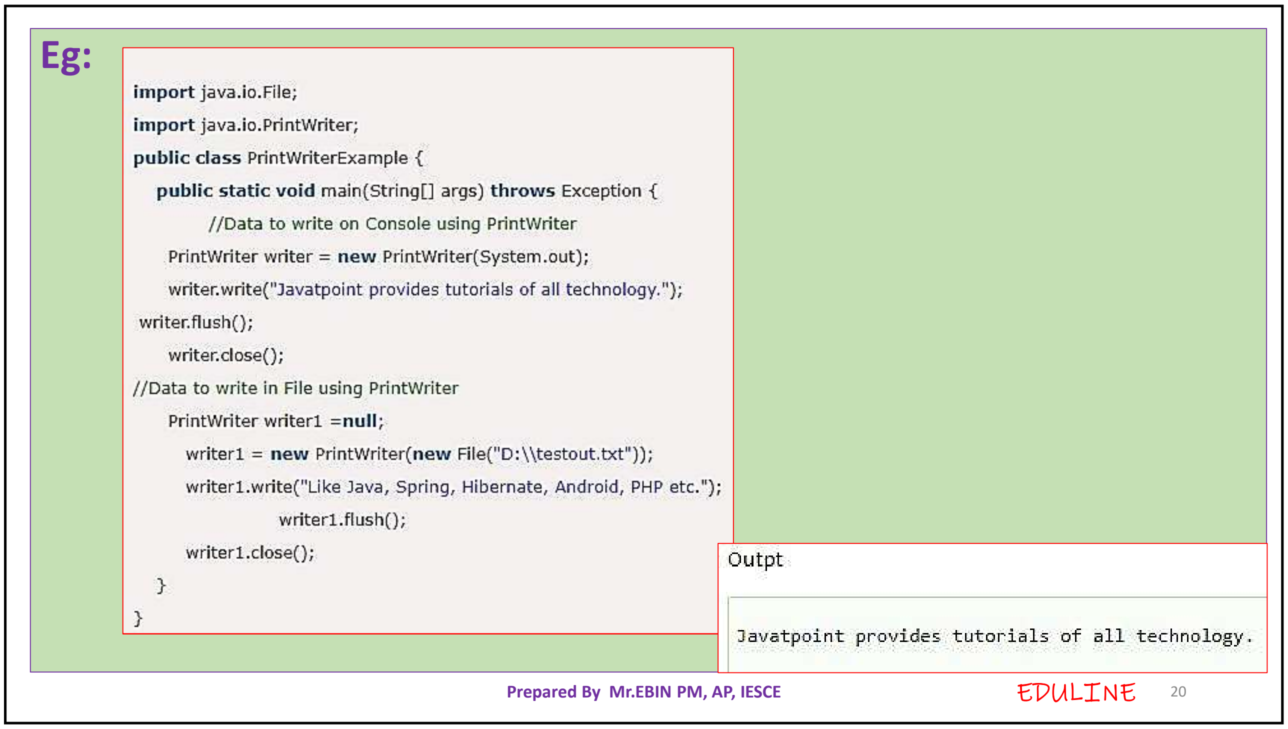


Method	Description
void println(boolean x)	It is used to print the boolean value.
void println(char[] x)	It is used to print an array of characters.
void println(int x)	It is used to print an integer.



PrintWriter append(char c)	It is used to append the specified character to the writer.
PrintWriter append(CharSequence ch)	It is used to append the specified character sequence to the writer.
PrintWriter append(CharSequence ch, int start, int end)	It is used to append a subsequence of specified character to the writer.
boolean checkError()	It is used to flushes the stream and check its error state.
protected void setError()	It is used to indicate that an error occurs.
protected void clearError()	It is used to clear the error state of a stream.
PrintWriter format(String format, Object... args)	It is used to write a formatted string to the writer using specified arguments and format string.
void print(Object obj)	It is used to print an object.
void flush()	It is used to flushes the stream.
void close()	It is used to close the stream.

Prepared By Mr.EBIN PM, AP, IESCE EDULINE 19



Eg:

```
import java.io.File;
import java.io.PrintWriter;
public class PrintWriterExample {
   public static void main(String[] args) throws Exception {
          //Data to write on Console using PrintWriter
     PrintWriter writer = new PrintWriter(System.out);
     writer.write("Javatpoint provides tutorials of all technology.");
 writer.flush();
     writer.close();
//Data to write in File using PrintWriter
     PrintWriter writer1 =null;
       writer1 = new PrintWriter(new File("D:\\testout.txt"));
       writer1.write("Like Java, Spring, Hibernate, Android, PHP etc.");
                   writer1.flush();
       writer1.close();
   }
}
```

Outpt

```
Javatpoint provides tutorials of all technology.
```

Prepared By Mr.EBIN PM, AP, IESCE EDULINE 20

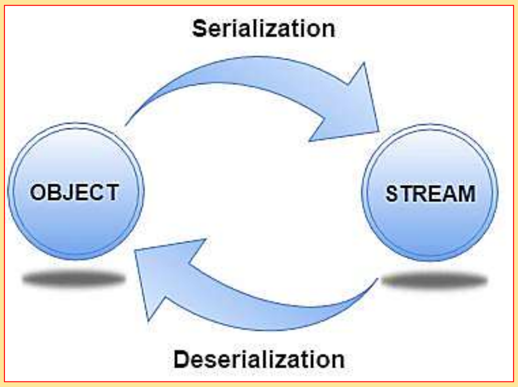
SERIALIZATION

- Serialization in Java is the process of converting the Java code Object into a Byte Stream, to transfer the Object Code from one Java Virtual machine to another and recreate it using the process of Deserialization.
- Most impressive is that the entire process is JVM independent, meaning an object can be serialized on one platform and deserialized on an entirely different platform.
- For serializing the object, we call the writeObject() method of ObjectOutputStream, and for deserialization we call the readObject() method of ObjectInputStream class.

- We must have to implement the *Serializable* interface for serializing the object.

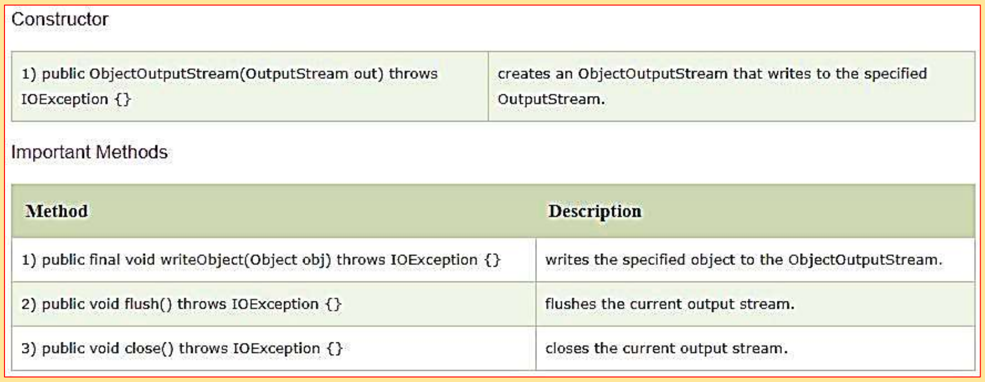
Advantages of Java Serialization

- It is mainly used to travel object's state on the network (which is known as marshaling).



❖ObjectOutputStream class

- The ObjectOutputStream class is used to write primitive data types, and Java objects to an OutputStream.
- Only objects that support the java.io.Serializable interface can be written to streams.


Constructor

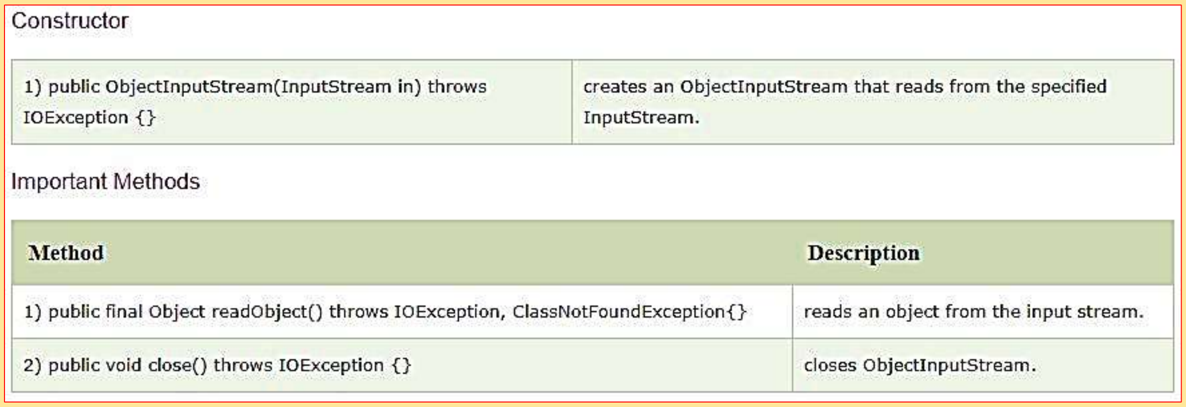
1) public ObjectOutputStream(OutputStream out) throws IOException {}	creates an ObjectOutputStream that writes to the specified OutputStream.

Important Methods

Method	Description
1) public final void writeObject(Object obj) throws IOException {}	writes the specified object to the ObjectOutputStream.
2) public void flush() throws IOException {}	flushes the current output stream.
3) public void close() throws IOException {}	closes the current output stream.

❖ObjectInputStream class

- An ObjectInputStream deserializes objects and primitive data written using an ObjectOutputStream.


Constructor

1) public ObjectInputStream(InputStream in) throws IOException {}	creates an ObjectInputStream that reads from the specified InputStream.

Important Methods

Method	Description
1) public final Object readObject() throws IOException, ClassNotFoundException{}	reads an object from the input stream.
2) public void close() throws IOException {}	closes ObjectInputStream.

Example of Java Serialization

- In this example, we are going to serialize the object of Student class. The writeObject() method of ObjectOutputStream class provides the functionality to serialize the object. We are saving the state of the object in the file named f.txt.

```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);}
 }
}
```

Output

```
success
```

WORKING WITH FILES

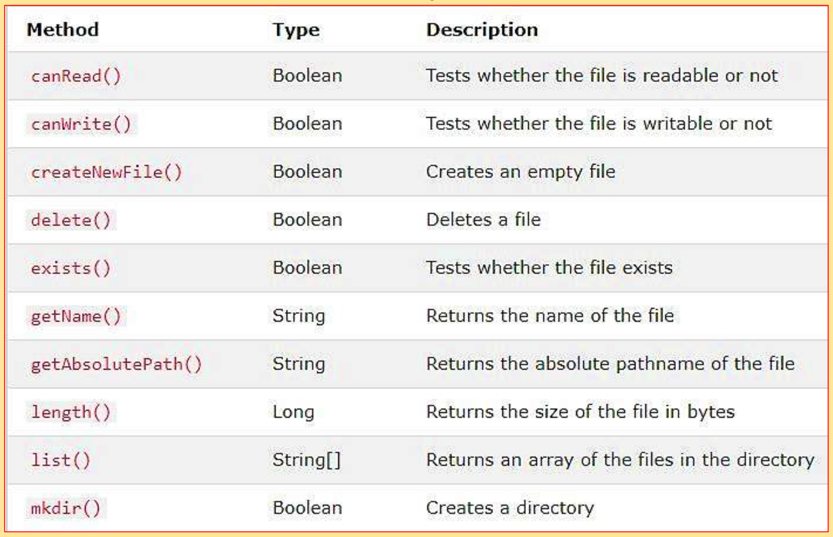
- File handling is an important part of any application.
- Java has several methods for creating, reading, updating, and deleting files.
- The File class from the java.io package, allows us to work with files.
- To use the File class, create an object of the class, and specify the filename or directory name:

Example

```java
import java.io.File;  // Import the File class

File myObj = new File("filename.txt"); // Specify the filename
```

- The File class has many useful methods for creating and getting information about files. For example:



Method	Type	Description
canRead()	Boolean	Tests whether the file is readable or not
canWrite()	Boolean	Tests whether the file is writable or not
createNewFile()	Boolean	Creates an empty file
delete()	Boolean	Deletes a file
exists()	Boolean	Tests whether the file exists
getName()	String	Returns the name of the file
getAbsolutePath()	String	Returns the absolute pathname of the file
length()	Long	Returns the size of the file in bytes
list()	String[]	Returns an array of the files in the directory
mkdir()	Boolean	Creates a directory

❖Create a File

- To create a file in Java, you can use the createNewFile() method.
- This method returns a boolean value: true if the file was successfully created, and false if the file already exists.
- Note that the method is enclosed in a try...catch block.
- This is necessary because it throws an IOException if an error occurs (if the file cannot be created for some reason):

Example

```java
import java.io.File;  // Import the File class
import java.io.IOException;  // Import the IOException class to handle errors

public class CreateFile {
  public static void main(String[] args) {
    try {
      File myObj = new File("filename.txt");
      if (myObj.createNewFile()) {
        System.out.println("File created: " + myObj.getName());
      } else {
        System.out.println("File already exists.");
      }
    } catch (IOException e) {
      System.out.println("An error occurred.");
      e.printStackTrace();
    }
  }
}
```

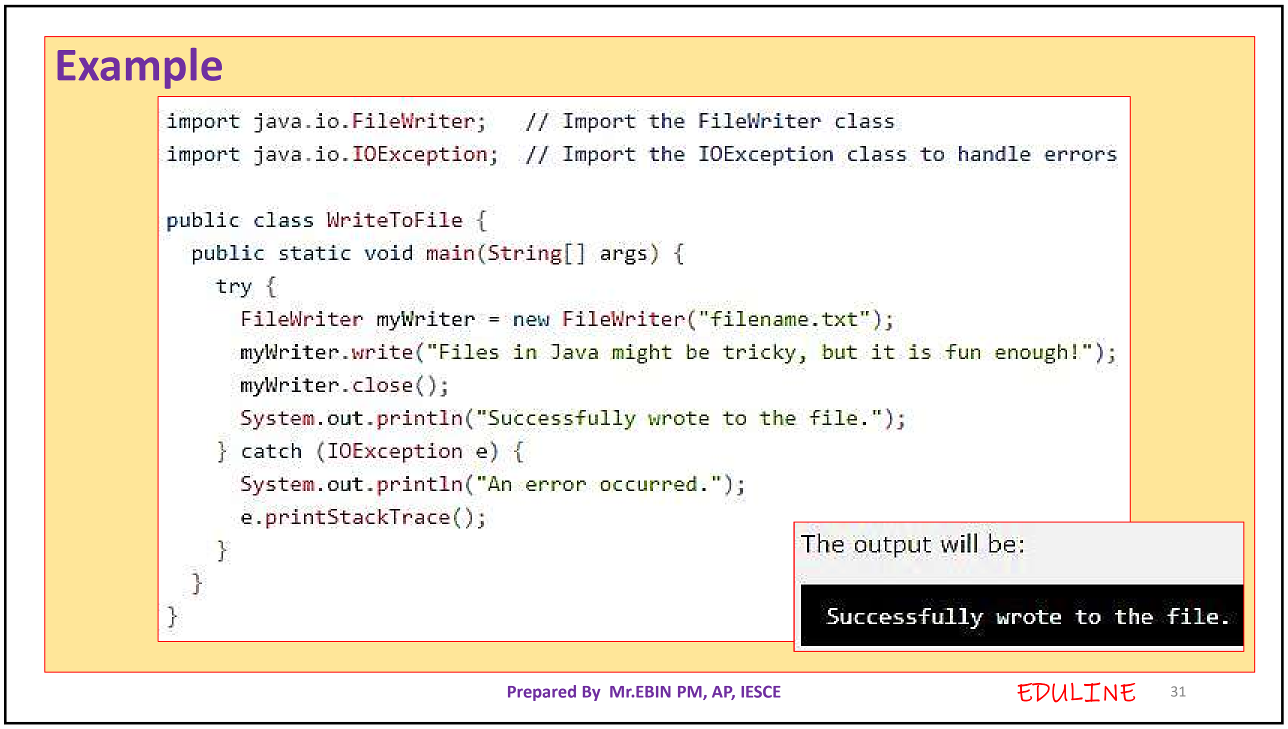
The output will be:

```
File created: filename.txt
```

❖Write To a File

➢In the following example, we use the FileWriter class together with its write() method to write some text to the file we created in the example above.

➢Note that when we are done writing to the file, we should close it with the close() method:



Example

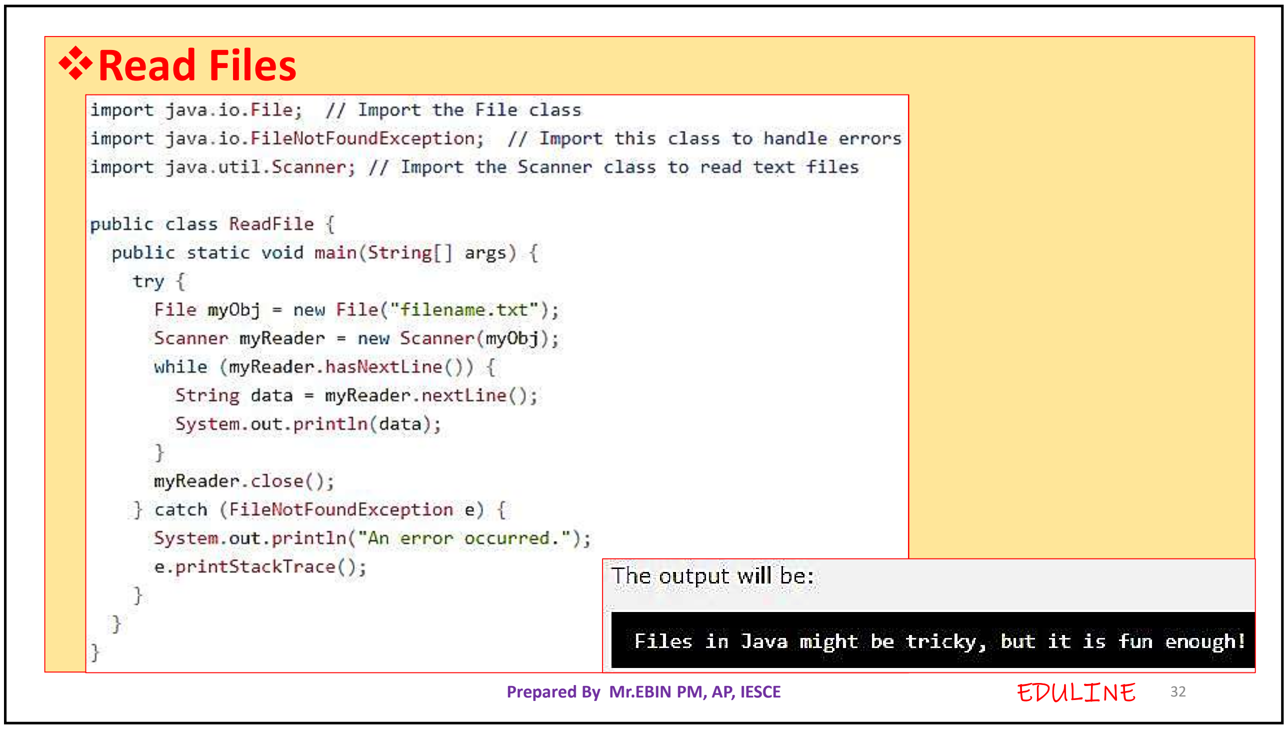
```
import java.io.FileWriter;   // Import the FileWriter class
import java.io.IOException;  // Import the IOException class to handle errors

public class WriteToFile {
  public static void main(String[] args) {
    try {
      FileWriter myWriter = new FileWriter("filename.txt");
      myWriter.write("Files in Java might be tricky, but it is fun enough!");
      myWriter.close();
      System.out.println("Successfully wrote to the file.");
    } catch (IOException e) {
      System.out.println("An error occurred.");
      e.printStackTrace();
    }
  }
}
```

The output will be:

```
Successfully wrote to the file.
```

Prepared By Mr.EBIN PM, AP, IESCE EDULINE 31



❖Read Files

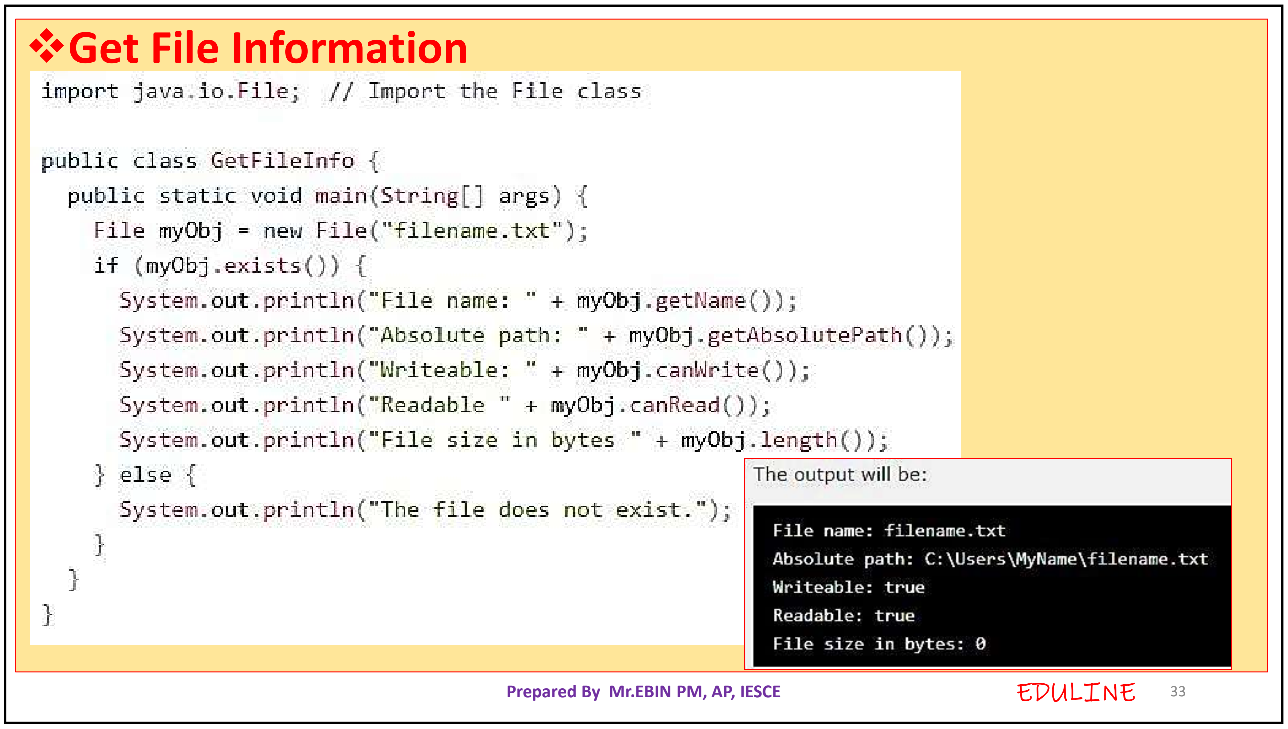
```
import java.io.File;  // Import the File class
import java.io.FileNotFoundException;  // Import this class to handle errors
import java.util.Scanner; // Import the Scanner class to read text files

public class ReadFile {
  public static void main(String[] args) {
    try {
      File myObj = new File("filename.txt");
      Scanner myReader = new Scanner(myObj);
      while (myReader.hasNextLine()) {
        String data = myReader.nextLine();
        System.out.println(data);
      }
      myReader.close();
    } catch (FileNotFoundException e) {
      System.out.println("An error occurred.");
      e.printStackTrace();
    }
  }
}
```

The output will be:

```
Files in Java might be tricky, but it is fun enough!
```

Prepared By Mr.EBIN PM, AP, IESCE EDULINE 32



❖Get File Information

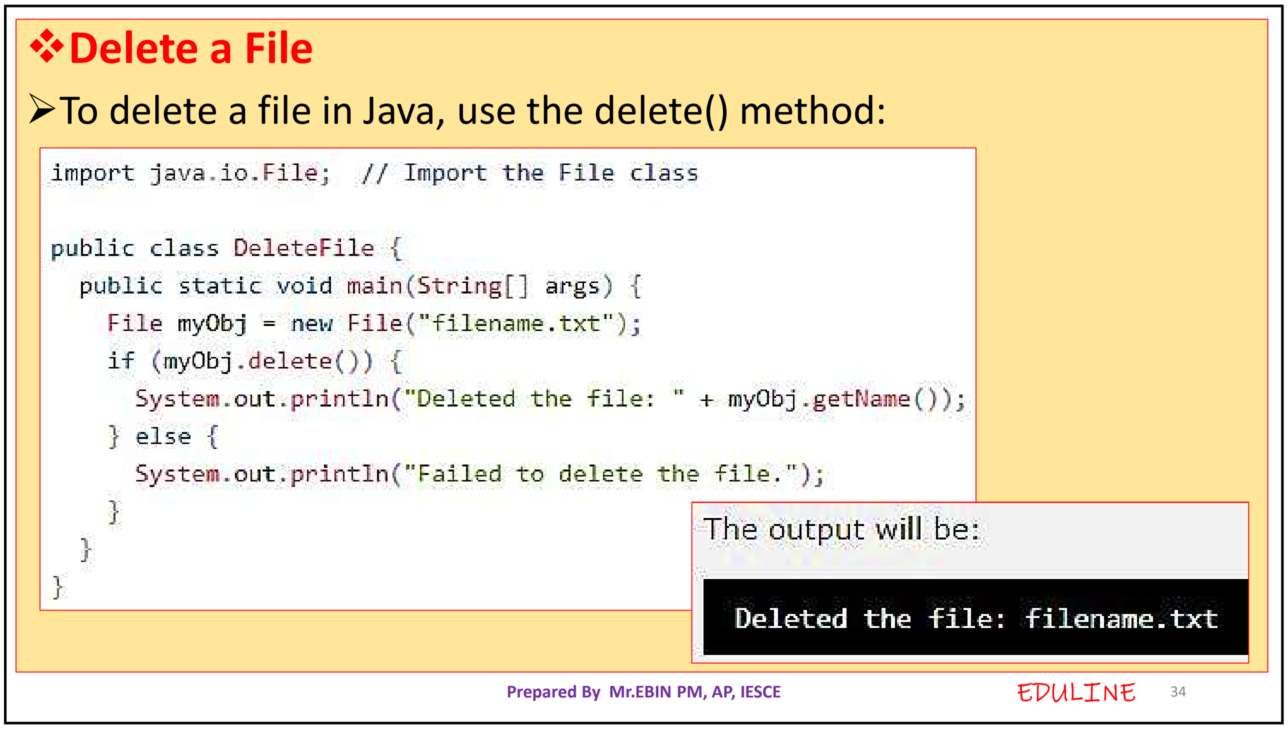
```java
import java.io.File;  // Import the File class

public class GetFileInfo {
  public static void main(String[] args) {
    File myObj = new File("filename.txt");
    if (myObj.exists()) {
      System.out.println("File name: " + myObj.getName());
      System.out.println("Absolute path: " + myObj.getAbsolutePath());
      System.out.println("Writeable: " + myObj.canWrite());
      System.out.println("Readable " + myObj.canRead());
      System.out.println("File size in bytes " + myObj.length());
    } else {
      System.out.println("The file does not exist.");
    }
  }
}
```

The output will be:

```
File name: filename.txt
Absolute path: C:\Users\MyName\filename.txt
Writeable: true
Readable: true
File size in bytes: 0
```

Prepared By Mr.EBIN PM, AP, IESCE EDULINE 33



❖Delete a File

➢To delete a file in Java, use the delete() method:

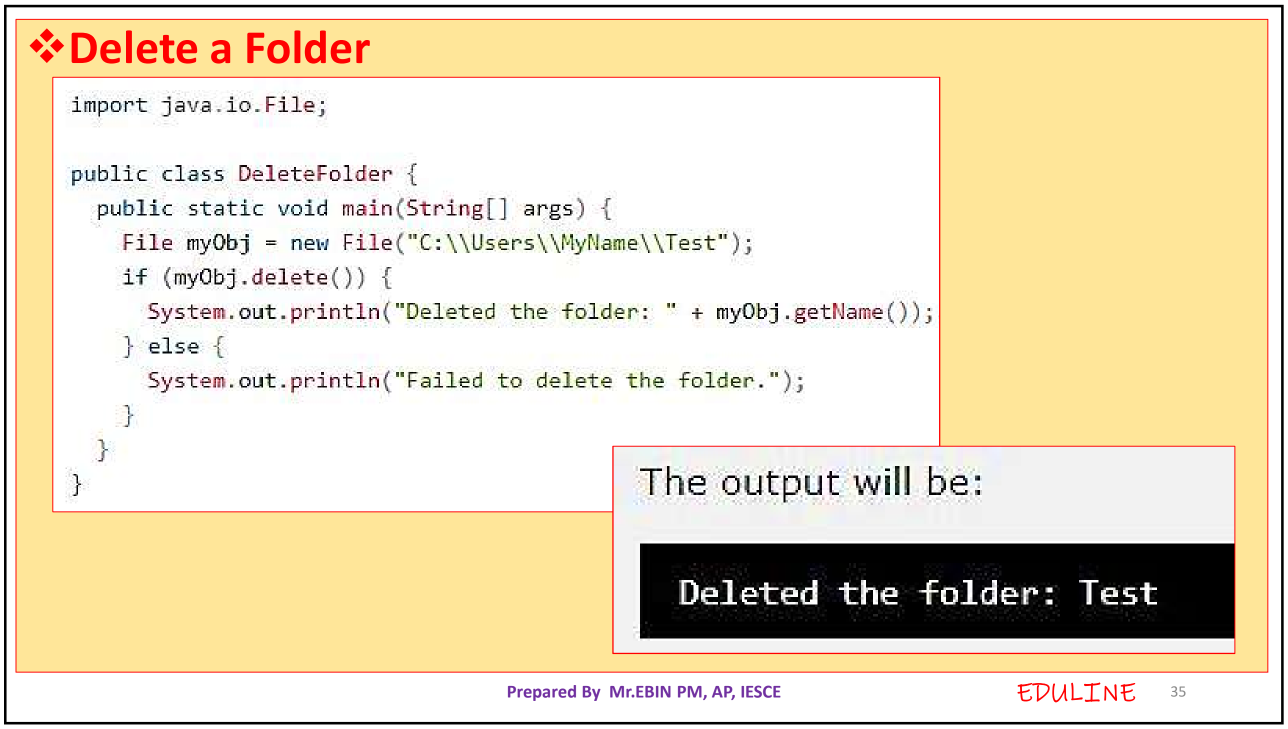
```java
import java.io.File;  // Import the File class

public class DeleteFile {
  public static void main(String[] args) {
    File myObj = new File("filename.txt");
    if (myObj.delete()) {
      System.out.println("Deleted the file: " + myObj.getName());
    } else {
      System.out.println("Failed to delete the file.");
    }
  }
}
```

The output will be:

```
Deleted the file: filename.txt
```

Prepared By Mr.EBIN PM, AP, IESCE EDULINE 34



❖Delete a Folder

```
import java.io.File;

public class DeleteFolder {
  public static void main(String[] args) {
    File myObj = new File("C:\\Users\\MyName\\Test");
    if (myObj.delete()) {
      System.out.println("Deleted the folder: " + myObj.getName());
    } else {
      System.out.println("Failed to delete the folder.");
    }
  }
}
```

The output will be:

```
Deleted the folder: Test
```

Prepared By Mr.EBIN PM, AP, IESCE EDULINE 35