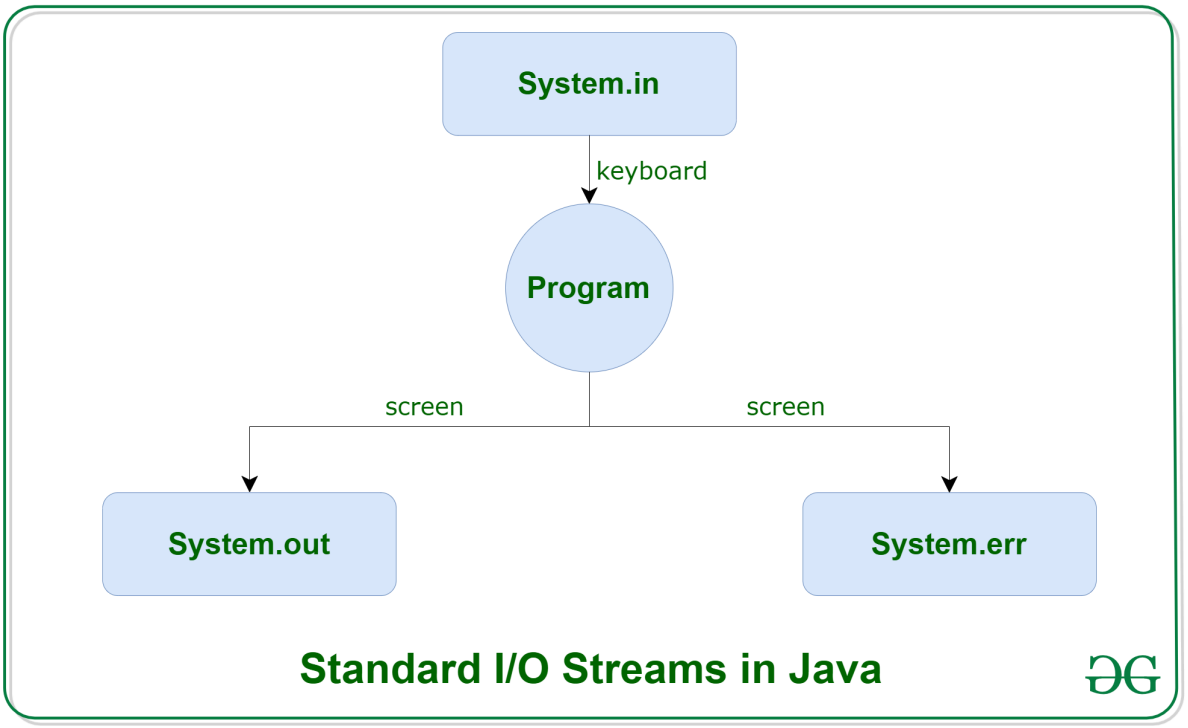
IO Operations
Assignment

Q1. What is Input and Output Stream in java?
Ans.

Java brings various Streams with its I/O package that helps the user to perform all the input-output operations. These streams support all the types of objects, data-types, characters, files etc to fully execute the I/O operations.

Before exploring various input and output streams lets look at 3 standard or default streams that Java has to provide which are also most common in use:



1. System.in: This is the standard input stream that is used to read characters from the keyboard or any other standard input device.
2. System.out: This is the standard output stream that is used to produce the result of a program on an output device like the computer screen.

   Here is a list of the various print functions that we use to output statements:

   - print(): This method in Java is used to display a text on the console. This text is passed as the parameter to this method in the form of String. This method prints the text on the console and the cursor remains at the end of the text at the console. The next printing takes place from just here.

     Syntax:

     ```
     System.out.print(parameter);
     ```

Q2. What are the methods of OutputStream?
Ans. Constructor and Description

- **OutputStream() : Single Constructor**

Methods:

void close() : Closes this output stream and releases any system resources associated with this stream.

```
Syntax :public void close()
            throws IOException
Throws:
```

- **IOException**

void flush() : Flushes this output stream and forces any buffered output bytes to be written out.

```
Syntax :public void flush()
           throws IOException
Throws:
```

- `IOException`

void write(byte[] b) : Writes b.length bytes from the specified byte array to this output stream.

```
Syntax :public void write(byte[] b)
           throws IOException
Parameters:
b - the data.
Throws:
```

- `IOException`

void write(byte[] b, int off, int len) : Writes len bytes from the specified byte array starting at offset off to this output stream.

```
Syntax :public void write(byte[] b,
         int off,
         int len)
           throws IOException
Parameters:
b - the data.
off - the start offset in the data.
len - the number of bytes to write.
Throws:
```

- `IOException`

abstract void write(int b) : Writes the specified byte to this output stream.

```
Syntax :
public abstract void write(int b)throws IOException
Parameters:
b - the byte.
Throws:
```

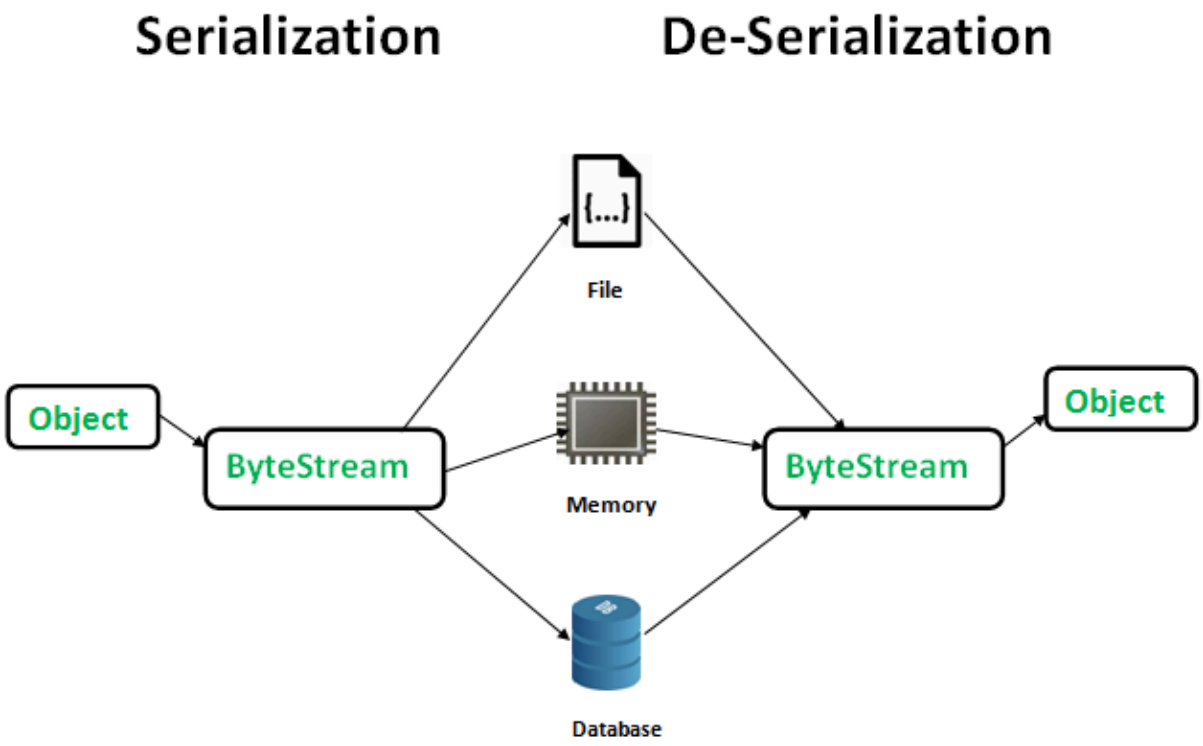
- `IOException`

Example-:

```
import java.io.*;
class OutputStreamDemo
{
    public static void main(String args[])throws Exception
    {
        OutputStream os = new FileOutputStream("file.txt");
        byte b[] = {65, 66, 67, 68, 69, 70};

                os.write(b);

                os.flush();

                for (int i = 71; i <75 ; i++)
        {
            os.write(i);
        }

        os.flush();
        os.close();
    }
```

Q3. What is serialisation in Java?

Ans. Serialisation 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.



The byte stream created is platform independent. So, the object serialized on one platform can be deserialized on a different platform. To make a Java object serializable we implement the **java.io.Serializable** interface. The ObjectOutputStream class contains **writeObject()** method for serialising an Object.

```
public final void writeObject(Object obj)
                        throws IOException
```

The ObjectInputStream class contains a readObject**()** method for deserializing an object.

Q4. What is the Serializable interface in java?

Ans. java.io.Serializable interface

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.

```
1. import java.io.Serializable;
2. public class Student implements Serializable{
3.  int id;
4.  String name;
5.  public Student(int id, String name) {
6.   this.id = id;
7.   this.name = name;
8.  }
9. }
```

Q5. What is deserialization in java?

Ans. 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. The byte stream created is platform independent. So, the object serialised on one platform can be deserialized on a different platform

```
1.  import java.io.*;
2. class Depersist{
3.  public static void main(String args[]){
4.   try{
5.     ObjectInputStream in=new ObjectInputStream(new FileInputStream("f.txt"));
6.   Student s=(Student)in.readObject();
7.    System.out.println(s.id+" "+s.name);
8.     in.close();
9.   }catch(Exception e){System.out.println(e);}
10. }
11.}
```

Q6. How is serialisation achieved in java?

Ans. To make a Java object serializable we implement the java.io.Serializable interface. The ObjectOutputStream class contains writeObject() method for serialising an Object.

```
public final void writeObject(Object obj)
                        throws IOException
```

Q7. How is deserialisation achieved in java?

Ans. The ObjectInputStream class contains a readObject() method for deserializing an object.

```
public final Object readObject()
                  throws IOException,
               ClassNotFoundException
```

Q8. How can you avoid certain member variables of class from getting Serialized?

Ans. Transient variables are not saved during the Serialization process. As the name implies, they do not constitute part of the object's state. We can use this variable to prevent certain fields from being serialised. For instance, a field that is not serializable should be marked transient or static.

Q9. What classes are available in the java IO File Classes API?

Ans. Java File class is Java's representation of a file or directory pathname. Because file and directory names have different formats on different platforms, a simple string is not adequate to name them. Java File class contains several methods for working with the pathname, deleting and renaming files, creating new directories, listing the contents of a directory, and determining several common attributes of files and directories.

- It is an abstract representation of files and directory pathnames.
- A pathname, whether abstract or in string form can be either absolute or relative. The parent of an abstract pathname may be obtained by invoking the getParent() method of this class.
- First of all, we should create the File class object by passing the filename or directory name to it. A file system may implement restrictions to certain operations

on the actual file-system object, such as reading, writing, and executing. These restrictions are collectively known as access permissions.

- Instances of the File class are immutable; that is, once created, the abstract pathname represented by a File object will never change.

Field	Type	Description
pathSeperator	String	the character or string used to separate individual paths in a list of file system paths.
pathSeperatorChar	Char	the character used to separate individual paths in a list of file system paths.

separator	String	default name separator character represented as a string.
separatorChar	char	default name separator character.

Q10. What is Differences between Externalizable and Serialisation interface.
Ans.

Serializable	Externalizable
A serializable interface is used to	An externalizable interface used to implement Externalization

implement serialisation.	
Serializable is a marker interface i.e. it does not contain any method.	The externalizable interface is not a marker interface and thus it defines two methods *writeExternal()* and *readExternal()*.
Serializable interface passes the responsibility of serialisation to JVM and the programmer has no control over serialisation, and it is a default algorithm.	The externalizable interface provides all serialisation responsibilities to a programmer and hence JVM has no control over serialisation.
Serialisation using a serializable interface has bad performance.	Serialisation using an externalizable interface has better performance.

Default serialisation does not require any no-arg constructor.	A public no-arg constructor is required while using an Externalizable interface.
It is hard to analyse and modify class structure because any change in structure may break serialisation.	It is relatively easy to analyse and modify class structure because of complete control over serialisation logic.
Using a serializable interface we save the total object to a file, and it is not possible to save part of the object.	Based on our requirements we can save either the total object or part of the object.
Transient keywords play an important role here.	Transient keywords won’t play any role.