***HOMEWORKS (PRE-MID)***

**Lecture #2:**

**Q1: What is JIT compiler? What is the purpose of JIT compiler?**

JIT stands for Just-In-Time. A JIT compiler improves the performance of Java programs by compiling bytecode into native machine code just before the code is executed.

**Q2: How JIT compiler tackles the problem of slow processing of Java Programs? (Bytecode also remains using JIT compiler).**

**Answer:**

When we use JVM, it compiles all methods and functions in a Java program. Even if your computer has high processing speed, still JVM takes a lot of time for compiling the code or interpreting bytecode into machine code.

But, if we use JIT compiler (which is by default present in java), it only executes the method which is being called, and does not compile all methods. It only compiles the methods which are called at run time, interprets its byte code file into machine time at run time as well.

In short, it only brings those methods to compilation which are required at a specific time.

If we add packages to java, the size of code does not increase because JIT compiler makes sure to compile only required method at run time, and after compilation it removes it from memory.

**Lecture #3:**

**Q1: What is the purpose of:**

1. ***finalize();*** 🡪 this method is of object class, that is used to perform cleanup activity before destroying any object. It is called by garbage collector before destroying the objects from memory.
2. ***System.gc();*** 🡪 The gc() method of System class runs the garbage collector. Through this method, JVM makes best effort to reuse the memory of discarded objects for quick reuses.

**Lecture #4:**

**Q1: What is the meaning of @override?**

* @Override is an annotation in Java that is used to indicate that a method in a child class is meant to override a method in its parent class.
* When a method in a child class has the @Override annotation, it indicates that the method is supposed to override a method with the same name in the parent class.
* It is only a check to see the validity of an overriding function.

**Q2: What type of errors exist in following code and why? How to resolve these errors?**

**Employee e = new Employee();**

**Teacher t;**

**t = (Teacher) e;**

**Answer:**

* **t = (Teacher) e;** 🡪 not possible/allowed.

Child reference variable can never contain that reference variable (which contain object of parent class). We use keyword of ***“instanceof”*** to do it.

Employee e = new Employee();

Teacher t;

if(t instanceof Employee) {

t = (Teacher) e;

} else System.out.println(“Error”);

* Two types of errors occur:

1. **Run Time Errors**
2. **ClassCast Exception**

**Lecture #5:**

**Q1: Can we declare following functionalities as abstract?**

1. **Private member functions**

* We cannot declare private member functions as abstract because they cannot be accessed or overridden by child classes.
* Abstract methods are always overridden by child classes, but since private members are not accessible outside the class, it doesn't make sense to declare them as abstract.

1. **Static member functions**

* Static member functions cannot be declared as abstract because they are not associated with a particular object of a class.
* Abstract methods are overridden by instance methods in child classes, but static methods cannot be overridden in this way.

1. **Constructors**

* Constructors cannot be declared as abstract because constructors are responsible for initializing the value of an object when it is created.
* If we declare a constructor as abstract, it means that we want to force child classes to provide their own implementation of the constructor.
* This does not make sense because constructors are always invoked when an object is created using the new keyword.
* If the constructor is abstract, then we cannot create an object of the child class, as we cannot initialize its value.

**Q2: Can we declare a class as an abstract, even it do not have any abstract function? If yes, then what would be the behavior of this class?**

* Yes, we can declare a class as abstract even if it does not contain any abstract methods.
* An abstract class is a special type of class that cannot be directly created.
* It is usually used as a base class for other classes to inherit from and customize as needed.
* If an abstract class does not contain any abstract methods, it can still provide some default implementation (include instance variables, methods, and constructors) for its child classes to inherit.

**Q3: Can we declare a constructor in an abstract class? If yes? When it will be executed?**

* Yes, we can declare constructors in an abstract class.
* When a class is declared as abstract, it means that it cannot be directly created.
* An abstract class can have constructors that are used to initialize its instance variables.
* The constructor of an abstract class is called when an instance of a concrete child class is created using the new keyword.
* When a child class is created, its constructor first calls the constructor of its immediate abstract parent class to perform initialization, and then executes its own code.

**Q4: Can we declare a data member (i.e., variable) as an abstract?**

* No, we cannot declare a data member (i.e., variable) as abstract in Java. The concept of abstract is only applicable to methods and classes.

**Lecture #6:**

**Q1: Can we have inner/nested try-catch blocks? If yes, then in which order catch block is executed?**

Yes, we can have nested try-catch blocks. They will execute in top-to-bottom manner (approach).

**Q2: How many catch blocks can we write?**

In Java, you can write as many catch blocks as you need in a try-catch block.

**Q3: If inner catch block does not match, then for unchecked exceptions, Will java provide its default handler or Control move towards outer catch block?**

* In case of inner catch blocks, if innermost catch block does not match for exceptions, then the control move towards outer catch block, and check whether the outer catch block is capable to deal with exceptions.
* If outer catch block also does not match (exception is unchecked), then JAVA will provide its default handler for it.

**Example:**

try{

try{

//NumberFormatException --> unchecked

} catch(ArithmeticException ex){

}

} catch (Exception ex){

//So here it will come.

}

**NOTE:**

If inner and outer both matches, then only inner will execute, and below it (including outer catch block) all catch blocks will be skipped/bypass.

**Q4: What is finally block? How finally block is used for exception handling?**

* A finally block is a code block that is used in conjunction with a try-catch block.
* It always execute whether exception occur or not.
* Finally block is used to free up (closing) resources /connections which are allocated by ourselves.
* It also executes whether we have return statement in our code.
* A finally block is used for cleanup operations or releasing resources.

**Syntax:**

try {

// some code that might throw an exception

} catch (Exception e) {

// handle the exception

} finally {

// code that should always be executed, regardless of whether or not an exception is thrown

}

**Q5: If we have finally block in both inner and outer try-catch blocks and exception matches with outer catch block, then which finally block will execute?**

* Both the inner and outer finally blocks will be executed in the order in which they appear.

**Example:**

public class FinallyExample {

public static void main(String[] args) {

try {

try {

int[] arr = {1, 2, 3};

System.out.println(arr[4]);

} catch (ArrayIndexOutOfBoundsException e) {

System.out.println("Inner catch block executed.");

} finally {

System.out.println("Inner finally block executed.");

}

} catch (Exception e) {

System.out.println("Outer catch block executed.");

} finally {

System.out.println("Outer finally block executed.");

}

}

}

**Q6: In which case, we can skip execution of finally block? (We can never skip execution of finally block except in 1 case).**

The only way to skip finally block is by calling ***System.exit(0)*** method, at the end of catch block, which is just before finally block.

***System.exit(0)*** will terminate the program.

**Q7: Does JAVA give default handler for our user defined/ custom exception classes?**

As our customized exception classes directly inherits from “Exception class”. So, they basically fall under checked exception category. So, JAVA does not provide default handler for our user defined/ custom exception classes.

**Lecture #7:**

**Q1: How will you avoid execution of finally block?**

We can avoid finally block by using ***System.exit()*** to terminate a JAVA program.

**Lecture #8:**

**Q1: How to open a file in append mode?**

We can append text into an existing file in java by opening a file using FileWriter class in append mode.

We can do this by using a special constructor provided by FileWriter class, which accepts a file and a Boolean, which if passed as true then open the file in append mode.

**For example:**

fw = new FileWriter("names.txt", true);

**Q2: Explore the API 🡪 java.io.file**

* The `java.io.File` class provides an abstract representation of file and directory pathnames.
* It can be used to perform operations such as file creation, deletion, renaming, and modification, as well as to obtain information about files and directories.
* Here are some of the commonly used methods of the `File` class:
* **File(String pathname)**: creates a new `File` instance by converting the given pathname string into an abstract pathname.
* **`boolean exists()`**: tests whether the file or directory denoted by this abstract pathname exists.
* **`boolean isFile()`**: tests whether the file denoted by this abstract pathname is a normal file.
* **`boolean isDirectory()`:** tests whether the file denoted by this abstract pathname is a directory.
* **`boolean createNewFile()`**: atomically creates a new, empty file named by this abstract pathname if and only if a file with this name does not yet exist.
* **`boolean delete()**`: deletes the file or directory denoted by this abstract pathname.
* **`String[] list()`:** returns an array of strings naming the files and directories in the directory denoted by this abstract pathname.
* **`boolean mkdir()`:** creates the directory named by this abstract pathname.
* `**long lastModified()`:** returns the time when the file denoted by this abstract pathname was last modified.
* **`String getName()`:** returns the name of the file or directory denoted by this abstract pathname.
* **`long length()`:** returns the length of the file denoted by this abstract pathname.

**Q3: What is the purpose/use of InputStreamReader?**

InputStreamReader is a bridge from byte streams to character streams. It reads bytes and decodes/converts them into characters using a specified charset.

**Example:**

BufferedReader bf = new BufferedReader(new InputStreamReader(System.in));

**Q4: Purpose of flush() in bufferedStream?**

The flush() method in the BufferedOutputStream class is used to force any buffered output to be written to the underlying stream.

**Example:**

BufferedOutputStream bos = new BufferedOutputStream(new FileOutputStream("output.txt"))

bos.write("Hello, world!".getBytes());

bos.flush(); // Force any buffered output to be written to the file

**Q5: What is the optimal size of buffer in Buffered Streams?**

* For general purpose I/O operations, a buffer size of around 8192 bytes (8 KB) is often considered a good default value.
* For network I/O operations, a larger buffer size of around 65536 bytes (64 KB) or even 131072 bytes (128 KB).
* For disk I/O operations, a larger buffer size of around 32768 bytes (32 KB) to 262144 bytes (256 KB).

**Q6: How can we save and retrieve data in hashmap?**

In Java, a HashMap is a collection that stores key-value pairs. To save data in a HashMap, you can use the put() method to associate a key with a value. To retrieve data from a HashMap, you can use the get() method to retrieve the value associated with a given key.

**Steps:**

1. **First import package:**

import java.util.HashMap;

1. **Create a new HashMap and pass key-value pairs**

public class HashMapExample {

HashMap<String, Integer> myMap = new HashMap<>();

1. **Add some key-value pairs to the HashMap**

myMap.put("STRING", INT);

1. **Retrieve the value associated with a key**

int value = myMap.get("STRING");

System.out.println("Value associated with key 'STRING': " + value);

}

1. **To retrieve all key-value pairs in the HashMap using a for loop and the keySet() method**

for (String key : myMap.keySet()) {

int val = myMap.get(key);

System.out.println("Key: " + key + ", Value: " + val);

}

**NOTE:**

There are also some functions of ***getKey(),*** ***getValue()*** and ***set()***, used in HashMap.

**Q7: How to check the type of objects while getting them through loop? How to downcast them?**

It can be done by using ***‘instanceOf’*** keyword.

**Pseudo code:**

if (obj instanceof Teacher) {

Teacher t = (Teacher) obj;

}

else if (obj instanceof Student) {

Student s = (Student) obj;

}

**Lecture #9:**

**Q1: Can we save the state of object of the following variables?**

1. **Private data member**

Their state can be saved, because even if they are private (not publically available), every object have their own value for private data member. Whenever we will save the state of an object, the value of private data member will also be saved.

**For example:**

**private String passcode; 🡪** it is a private data member. So, the value which I initialize to it will be saved in file or stream.

1. **Static data member**

We cannot save their state, because static data members are not directly related to object. They are class-wise members (non-instance variables). All the objects of a class share the same copy of static data member, so they are not linked with objects. As in deserialization, we save the state of objects, so only data members related to objects will be saved and not static members.

**Q2: If available number of objects in a file are not known and we need to retrieve the objects from the file using loop? How many times the loop need to execute? (Deserialization)**

Whenever we are doing deserialization, and as we know the objects are saved in the file, and we are reading objects, so we will not execute the loop (for reading objects) to null, because in case of objects, the file might not end with null. And in this case, endOfFileException occurs.

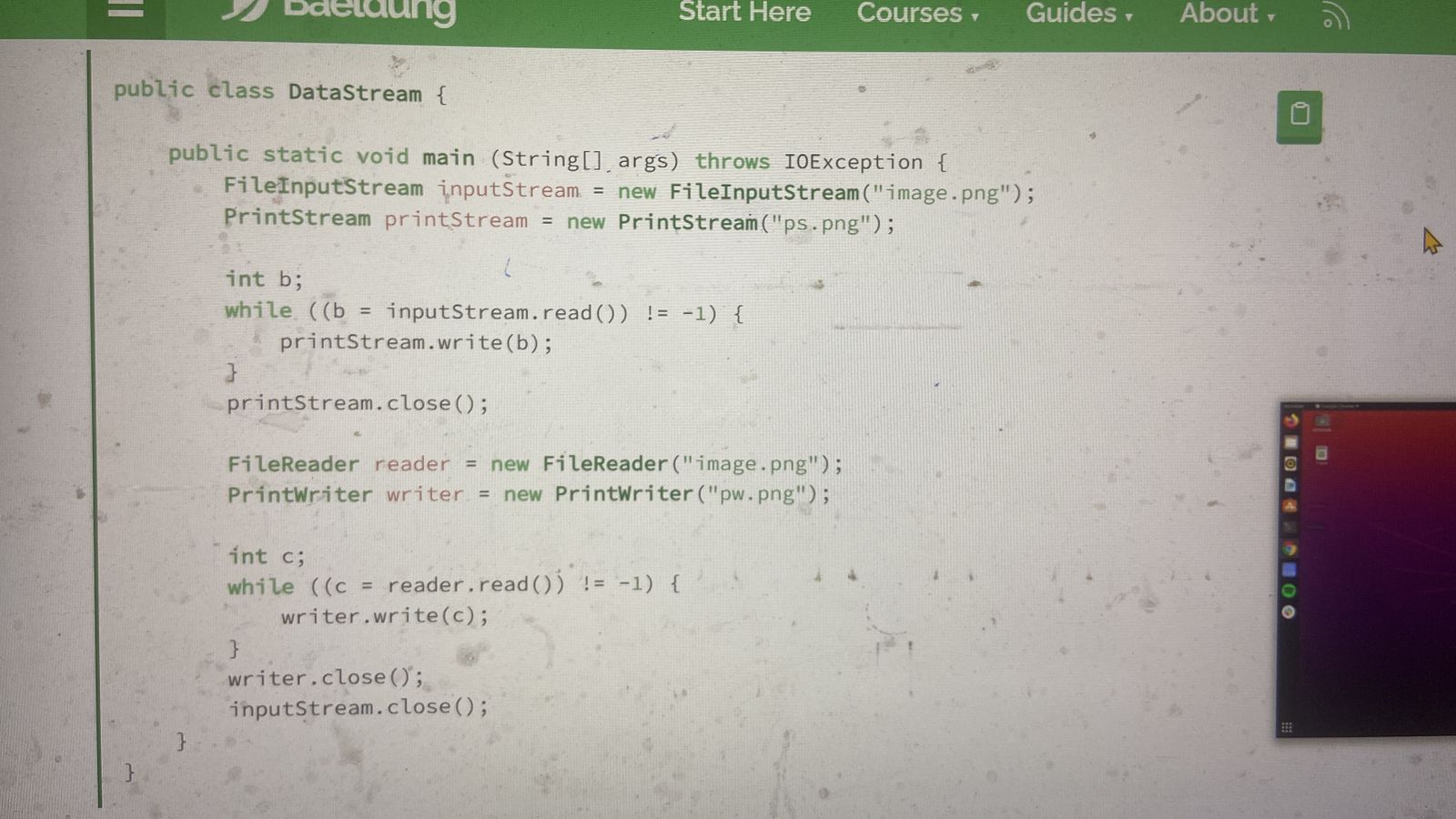
We apply a check on a low level stream, to see if there is still data in a low level stream or not, using available function. If a low level stream doesn't exist (fis.available =0), means no more data is coming and All the objects are read completely. Now terminate.

But if this is not the case, then data is present in fis. (fis.available≠ 0), so keep reading object by object.

**Q3: Actual difference between printWriter and printStream?**

|  |  |
| --- | --- |
| **printStream** | **printWriter** |
| * PrintStream is a byte-based stream. * It belongs to the category of ***OutputStream*** class. * It can write data only byte by byte. * It is a specialized stream class which can convert raw data (byte by byte) into a special format, and write it to some end-point like file, console, etc. * We have functions of ***‘println’, ‘write’*** and ***‘print’*** in this stream, which accept string (and convert it into 1 byte at back-end itself) and further send it on end-point. * It is also used to write the primitive types of data onto output Stream by formatting them. * We use it transferring raw data (images, videos, audios, pngs, etc.) * Example: | * PrintWriter is a character-based stream. * It belongs to the category of ***Writer*** class. * It can write data only character by character. * It is a specialized stream class which can convert raw data (character by character) into a special format, and write it to some end-point like file, console, etc. * We have functions of ***‘println’, ‘write’*** and ***‘print’*** in this stream, which accept string (of 2 bytes) and further send it on end-point. * We use it transferring text data * Examples: |

* **Combine Example** of **PrintStream** and **PrintWriter**:



**Q4: How will you use Externalizable Interface for manual serialization?**

[Externalizable interface in Java - GeeksforGeeks](https://www.geeksforgeeks.org/externalizable-interface-java/)

Externalization serves the purpose of custom Serialization, where we can decide what to store in stream. Externalizable interface present in java.io, is used for Externalization which extends Serializable interface. It consist of two methods which we have to override to write/read object into/from stream which are-

// to read object from stream

void readExternal(ObjectInput in)

// to write object into stream

void writeExternal(ObjectOutput out)

**Key differences between Serializable and Externalizable :**

Implementation : Unlike Serializable interface which will serialize the variables in object with just by implementing interface, here we have to explicitly mention what fields or variables you want to serialize.

Methods : Serializable is marker interface without any methods. Externalizable interface contains two methods: writeExternal() and readExternal().

Process: Default Serialization process will take place for classes implementing Serializable interface. Programmer defined Serialization process for classes implementing Externalizable interface.

Backward Compatibility and Control: If you have to support multiple versions, you can have full control with Externalizable interface. You can support different versions of your object. If you implement Externalizable, it’s your responsibility to serialize super class.

public No-arg constructor: Serializable uses reflection to construct object and does not require no arg constructor. But Externalizable requires public no-arg constructor.

Below is the example for Externalization-

// Java program to demonstrate working of Externalization

// interface

import java.io.\*;

class Car implements Externalizable {

static int age;

String name;

int year;

public Car()

{

System.out.println("Default Constructor called");

}

Car(String n, int y)

{

this.name = n;

this.year = y;

age = 10;

}

@Override

public void writeExternal(ObjectOutput out)

throws IOException

{

out.writeObject(name);

out.writeInt(age);

out.writeInt(year);

}

@Override

public void readExternal(ObjectInput in)

throws IOException, ClassNotFoundException

{

name = (String)in.readObject();

year = in.readInt();

age = in.readInt();

}

@Override public String toString()

{

return ("Name: " + name + "\n"

+ "Year: " + year + "\n"

+ "Age: " + age);

}

}

public class ExternExample {

public static void main(String[] args)

{

Car car = new Car("Shubham", 1995);

Car newcar = null;

// Serialize the car

try {

FileOutputStream fo

= new FileOutputStream("gfg.txt");

ObjectOutputStream so

= new ObjectOutputStream(fo);

so.writeObject(car);

so.flush();

}

catch (Exception e) {

System.out.println(e);

}

// Deserialization the car

try {

FileInputStream fi

= new FileInputStream("gfg.txt");

ObjectInputStream si

= new ObjectInputStream(fi);

newcar = (Car)si.readObject();

}

catch (Exception e) {

System.out.println(e);

}

System.out.println("The original car is:\n" + car);

System.out.println("The new car is:\n" + newcar);

}

}

**Output:**

Default Constructor called

The original car is:

Name: Shubham

Year: 1995

Age: 10

The new car is:

Name: Shubham

Year: 1995

Age: 10

In the example, the class Car has two methods- writeExternal and readExternal. So, when we write “Car” object to OutputStream, writeExternal method is called to persist the data. The same applies to the readExternal method.

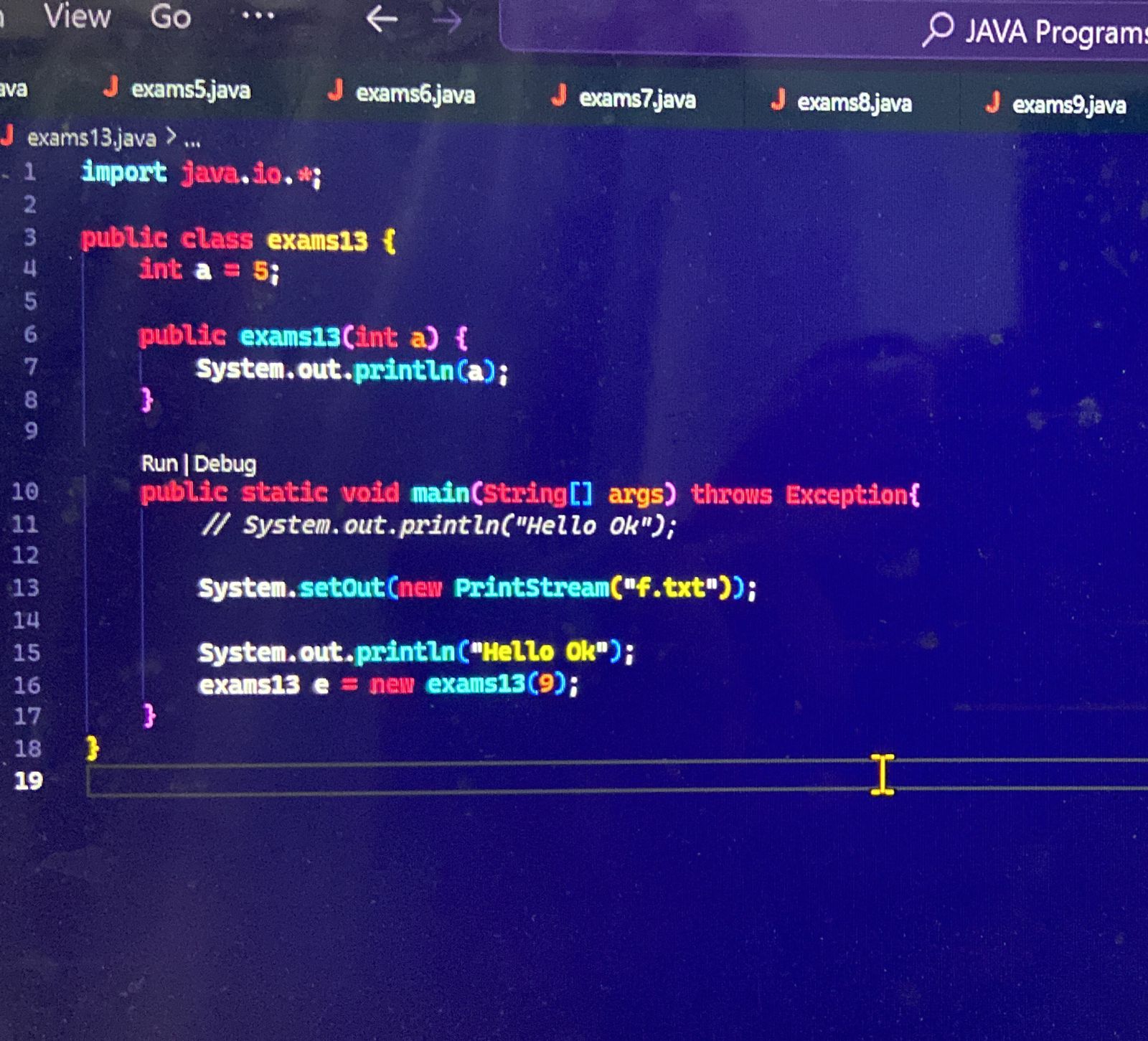
When an Externalizable object is reconstructed, an instance is created first using the public no-argument constructor, then the readExternal method is called. So, it is mandatory to provide a no-argument constructor.

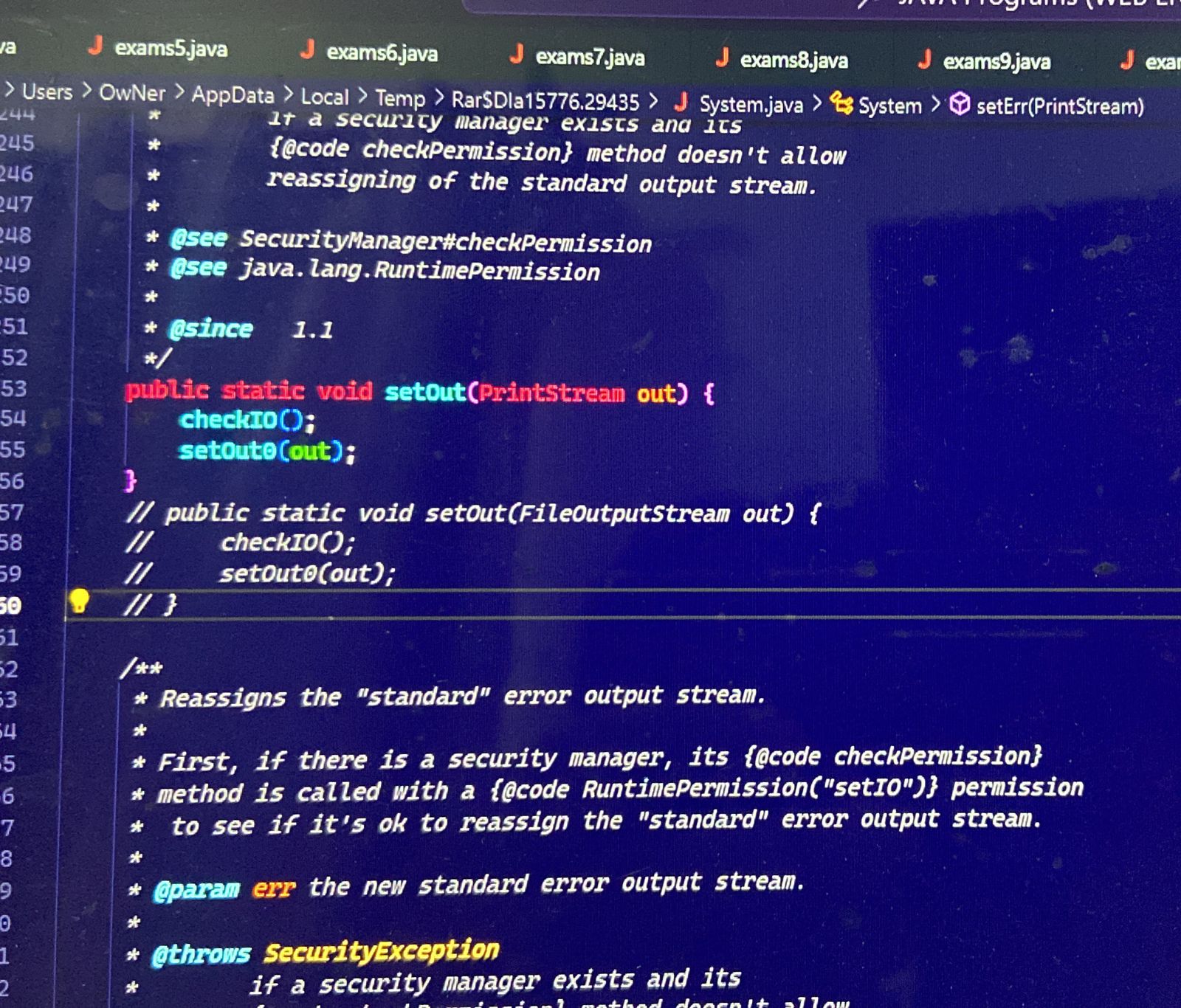
When an object implements Serializable interface, is serialized or deserialized, no constructor of object is called and hence any initialization which is implemented in constructor can’t be done.

**Q5:** **System.setOut(new FileOutputStream("f.txt"));**

**System.out.println("Hello ok");**

**What will be the result when we execute System.out.println("Hello ok"); ?**

****

****

The `System.setOut()` method is used to redirect the standard output stream (`System.out`) to a different output stream. In your example, you are redirecting the standard output stream to a `PrintStream` object that writes to a file named "f".

After calling `System.setOut()`, any subsequent calls to `System.out.println()` will write the output to the file instead of the console. For example, the line `System.out.println("Hello Ok")` will write the string "Hello Ok" to the file "f" instead of printing it to the console.

**Lecture #10:**

**Q1: In TCP/IP, how many max connections (communication sockets) can be opened in one computer?**

For TCP/IP, 64K connections can be opened on one computer.

For UDP, we can also open 64K connections.

'1' computer can have a total 128K {64K(TCP/IP) + 64K(UDP)} open connections at a time.