Programming 2

Tutorial 10

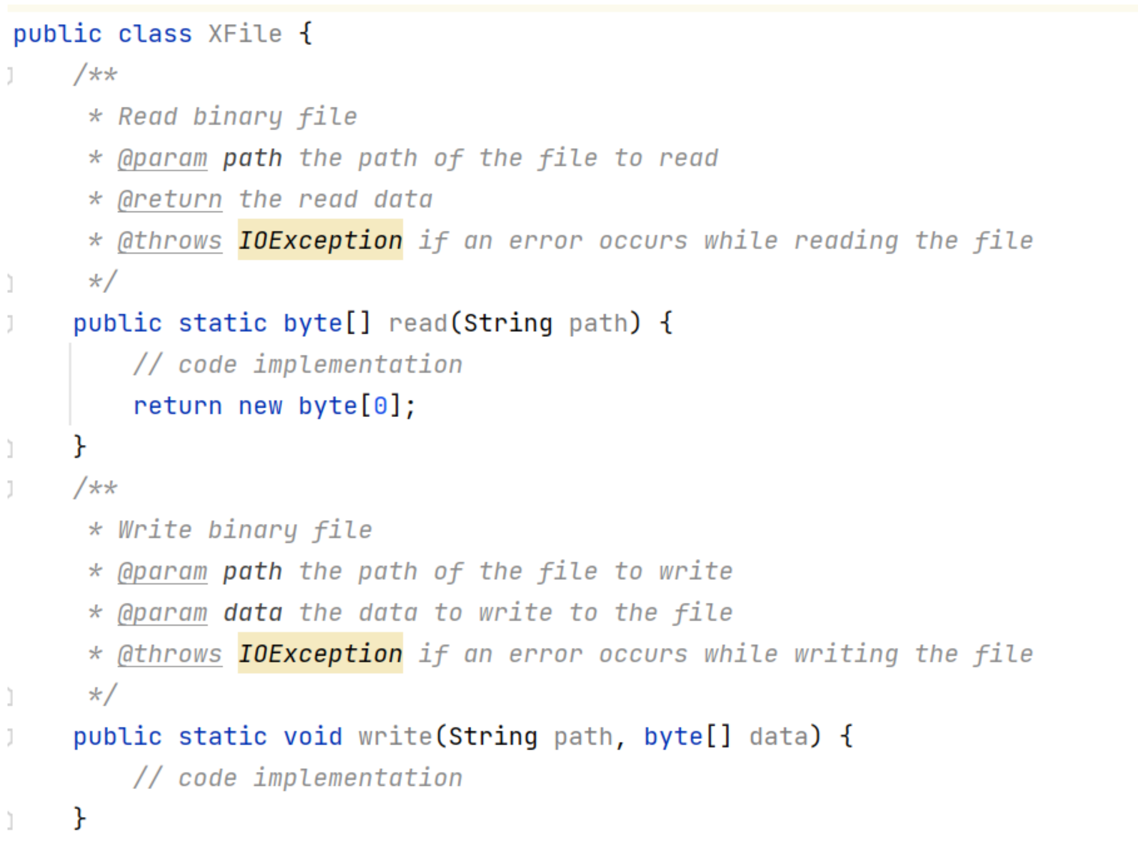
Exercise 1: (Required)

Build a utility library XFile with read() and write() functions allowing reading and writing binary files.

Utilize the above library to copy one file to another.

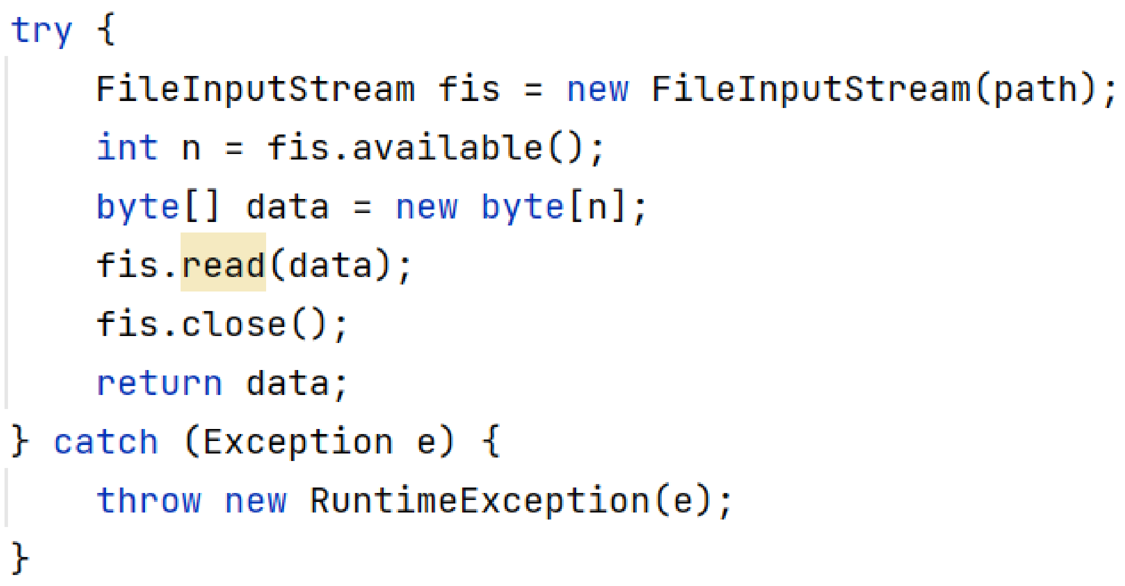
GUIDELINES:

- Create the XFile class



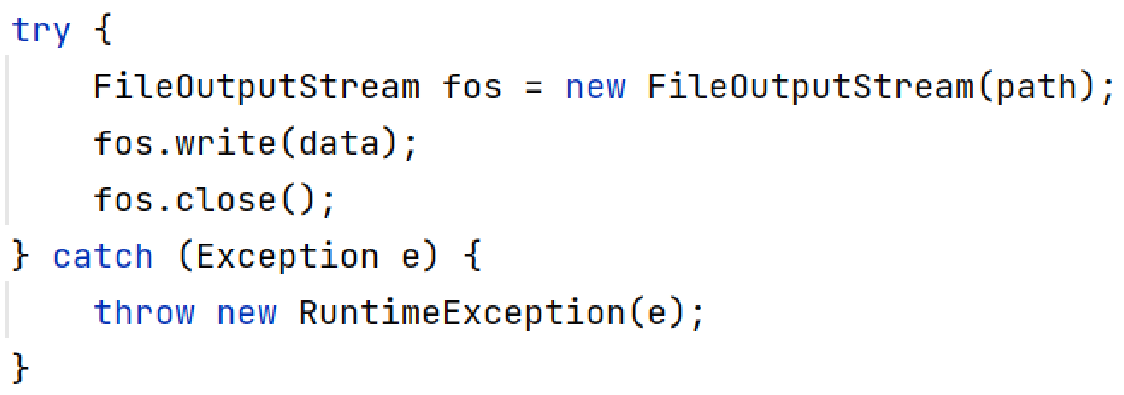
```java
public class XFile {
    /**
     * Read binary file
     * @param path the path of the file to read
     * @return the read data
     * @throws IOException if an error occurs while reading the file
     */
    public static byte[] read(String path) {
        // code implementation
        return new byte[0];
    }
    /**
     * Write binary file
     * @param path the path of the file to write
     * @param data the data to write to the file
     * @throws IOException if an error occurs while writing the file
     */
    public static void write(String path, byte[] data) {
        // code implementation
    }
```

- Write code for the read() function



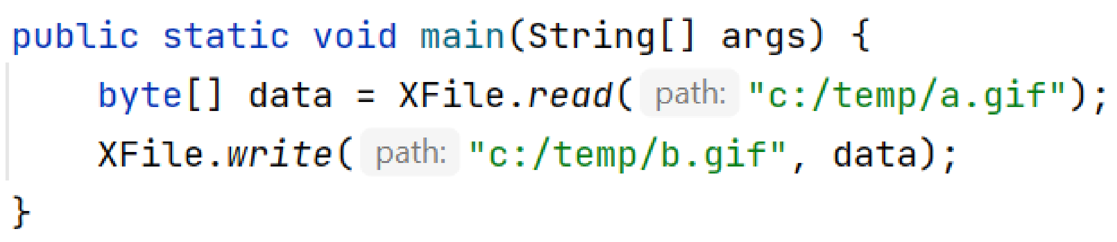
```
try {
    FileInputStream fis = new FileInputStream(path);
    int n = fis.available();
    byte[] data = new byte[n];
    fis.read(data);
    fis.close();
    return data;
} catch (Exception e) {
    throw new RuntimeException(e);
}
```

- Write code for the write() function



```
try {
    FileOutputStream fos = new FileOutputStream(path);
    fos.write(data);
    fos.close();
} catch (Exception e) {
    throw new RuntimeException(e);
}
```

- Create the XFileDemo class containing the main() method and utilize the XFile library as follows:



```
public static void main(String[] args) {
    byte[] data = XFile.read( path: "c:/temp/a.gif");
    XFile.write( path: "c:/temp/b.gif", data);
}
```

Exercise 2: (Required):

Extend the XFile library with two functions allowing reading and writing objects from/to a file.

```java
/**
 * Read object file
 * @param path the path of the file to read
 * @return the read object
 * @throws IOException if an error occurs while reading the file
 */
public static Object readObject(String path) {
    // code implementation
    return null;
}

/**
 * Write object file
 * @param path the path of the file to write
 * @param object the object to write to the file
 * @throws IOException if an error occurs while writing the file
 */
public static void writeObject(String path, Object object) {
    // code implementation
}
```

- Write code for the readObject() function

```java
try {
    ObjectInputStream ois = new ObjectInputStream(new FileInputStream(path));
    Object object = ois.readObject();
    ois.close();
    return object;
} catch (Exception e) {
    throw new RuntimeException(e);
}
```

- Write code for the writeObject() function

```java
try {
    ObjectOutputStream oos = new ObjectOutputStream(new FileOutputStream(path));
    oos.writeObject(object);
    oos.close();
} catch (Exception e) {
    throw new RuntimeException(e);
}
```

- Utilize the readObject() and writeObject() functions to read and write List<Student>. Note that the Student class must implement the Serializable interface. Fill the domain constraint table and apply to your code.

Attribute	Type	Mutable	Optional	Length	Min	max
name						
mark						
faculty						

```java
public class Student implements Serializable {
    // Student class definition
}
```

- In the main method

```java
List<Student> list = new ArrayList<>();
list.add(new Student("Tuấn", 5, "CNTT"));
list.add(new Student("Cường", 7.5, "TKTW"));
list.add(new Student("Hạnh", 8.5, "CNTT"));

XFile.writeObject("c:/temp/students.dat", list);
```

Exercise 3: (Required):

Create a Java class `Matrix` to represent bidimensional matrices of real numbers. The class should export the following methods:

- `Matrix(int n, int m)` : constructor that creates a matrix of size nxm, with all values initially set to 0;
- `void save(String filename)` : that saves the content of the matrix on the file specified by filename;

- `static Matrix read(String filename)` : that reads the data about a matrix from the file specified by filename, creates the matrix, and returns it;
- `Matrix sum(Matrix m)` : that returns the matrix that is the sum of the object and of m, if the two matrices have the same dimensions, and null otherwise;
- `Matrix product(Matrix m)` : that returns the matrix that is the product of the object and of m, if the two matrices have compatible dimensions, and null otherwise

Exercise 4: (Required):

Create a class `IOFile` that exports some functionalities on text files. The class should have a constructor with one parameter of type **String**, representing the name of the file on which to operate, and should export the following methods:

- `int countLines()` : that returns the number of lines of the file;
- `void write(OutputStream os)` : that writes the content of the file to os;
- `void print()` : that prints the content of the file to the screen;
- `void copy(String filename)` : that copies the content of the file to the file specified by filename;
- `void delete()` : that deletes the file.