**File Handling in Java**

**1. Reading and Writing Files**

Java offers multiple classes for reading and writing files, depending on whether you are working with text or binary data.

**FileWriter and FileReader**: These are used for working with text files, using character-based streams.

* **FileWriter**: Writes characters to a file.
* **FileReader**: Reads characters from a file.

**Writing to a File** (Using FileWriter):

java

CopyEdit

import java.io.\*;

public class Main {

public static void main(String[] args) {

try {

FileWriter writer = new FileWriter("example.txt");

writer.write("Hello, Java!");

writer.close();

System.out.println("File written successfully.");

} catch (IOException e) {

System.out.println("An error occurred.");

e.printStackTrace();

}

}

}

**Explanation**:

* FileWriter writes the string "Hello, Java!" to the file.
* The write() method writes the text, and close() ensures the file is saved properly.

**Reading from a File** (Using FileReader):

java

CopyEdit

import java.io.\*;

public class Main {

public static void main(String[] args) {

try {

FileReader reader = new FileReader("example.txt");

int character;

while ((character = reader.read()) != -1) {

System.out.print((char) character);

}

reader.close();

} catch (IOException e) {

System.out.println("An error occurred.");

e.printStackTrace();

}

}

}

**Explanation**:

* FileReader reads characters from the file.
* The read() method returns one character at a time, and -1 indicates the end of the file.

**2. Byte Streams vs Character Streams**

* **Byte Streams**: Handle raw binary data (e.g., images, audio files). Examples: FileInputStream, FileOutputStream.
* **Character Streams**: Handle text data (e.g., characters and strings). Examples: FileReader, FileWriter.

**When to Use**:

* **Byte Streams**: For binary files like .jpg, .mp3, etc.
* **Character Streams**: For text-based files like .txt, .csv, .xml.

**3. The File Class**

The **File** class represents file and directory pathnames and provides methods for creating, deleting, and checking file properties.

**Common Methods of the File Class**:

* exists(): Checks if the file or directory exists.
* createNewFile(): Creates a new file.
* delete(): Deletes the file or directory.
* length(): Returns the size of the file in bytes.
* isDirectory(): Checks if the file object is a directory.
* isFile(): Checks if the file object is a file.

**Example**:

java

CopyEdit

import java.io.\*;

public class Main {

public static void main(String[] args) {

File file = new File("example.txt");

if (file.exists()) {

System.out.println("File exists.");

System.out.println("File size: " + file.length() + " bytes");

} else {

System.out.println("File does not exist.");

}

}

}

**Explanation**:

* The File class checks if the file exists and prints its size.

**4. BufferedReader and BufferedWriter**

**BufferedReader** and **BufferedWriter** provide efficient reading and writing by buffering data, reducing the number of disk reads and writes.

* **BufferedReader**: Reads text from a file more efficiently.
* **BufferedWriter**: Writes text to a file more efficiently.

**Reading with BufferedReader**:

java

CopyEdit

import java.io.\*;

public class Main {

public static void main(String[] args) {

try {

BufferedReader reader = new BufferedReader(new FileReader("example.txt"));

String line;

while ((line = reader.readLine()) != null) {

System.out.println(line);

}

reader.close();

} catch (IOException e) {

System.out.println("An error occurred.");

e.printStackTrace();

}

}

}

**Explanation**:

* readLine() reads one line at a time, making it more efficient for reading large text files.

**Writing with BufferedWriter**:

java

CopyEdit

import java.io.\*;

public class Main {

public static void main(String[] args) {

try {

BufferedWriter writer = new BufferedWriter(new FileWriter("example.txt"));

writer.write("Hello, BufferedWriter!");

writer.newLine(); // Adds a new line

writer.write("This is efficient file writing.");

writer.close();

} catch (IOException e) {

System.out.println("An error occurred.");

e.printStackTrace();

}

}

}

**Explanation**:

* write() writes the text to the file, and newLine() adds a new line.

**5. Serialization and Deserialization**

**Serialization**: Converts an object into a byte stream for storage or transmission.  
**Deserialization**: Converts the byte stream back into an object.

* To make a class serializable, it must implement the **Serializable** interface.

**Serialization Example**:

java

CopyEdit

import java.io.\*;

class Person implements Serializable {

String name;

int age;

public Person(String name, int age) {

this.name = name;

this.age = age;

}

}

public class Main {

public static void main(String[] args) {

Person person = new Person("John", 25);

try {

// Serialize object

ObjectOutputStream out = new ObjectOutputStream(new FileOutputStream("person.ser"));

out.writeObject(person);

out.close();

System.out.println("Object serialized successfully.");

} catch (IOException e) {

e.printStackTrace();

}

}

}

**Explanation**:

* The Person class implements Serializable to make it serializable.
* ObjectOutputStream writes the object to a file.

**Deserialization Example**:

java

CopyEdit

import java.io.\*;

class Person implements Serializable {

String name;

int age;

public Person(String name, int age) {

this.name = name;

this.age = age;

}

}

public class Main {

public static void main(String[] args) {

try {

// Deserialize object

ObjectInputStream in = new ObjectInputStream(new FileInputStream("person.ser"));

Person person = (Person) in.readObject();

in.close();

System.out.println("Object deserialized: " + person.name + ", " + person.age);

} catch (IOException | ClassNotFoundException e) {

e.printStackTrace();

}

}

}

**Explanation**:

* ObjectInputStream reads the object from the file, and readObject() retrieves it.

**Summary of Key Topics:**

* **Reading and Writing Files**:
  + Use FileWriter/FileReader for text files and FileInputStream/FileOutputStream for binary files.
* **Byte Streams vs Character Streams**:
  + Byte streams handle binary data; character streams handle text data.
* **The File Class**:
  + Provides methods for file operations such as creating, deleting, and checking properties.
* **BufferedReader and BufferedWriter**:
  + Provide efficient reading and writing by buffering data.
* **Serialization and Deserialization**:
  + Convert objects to/from byte streams for persistent storage or network transmission.

4o mini