Demo 1:FileInputStream

1.Open file

2.Read File

3.Close File.

```
package chapter12;

// Import packages to read file and handle exceptions / 导入读取文件和异常处理的包
// FileInputStream - for reading bytes from files / 用于从文件读取字节
// FileNotFoundException - when file doesn't exist / 当文件不存在时
// IOException - for general input/output errors / 用于一般输入/输出错误


public class FileInputStreamDemo {
    public static void main(String[] args) {
         int i;
// Stores byte value (0-255) returned by read() method
// 存储 read()方法返回的字节值(0-255)
        char c;
// Used to convert byte to character for display / 用于将字节转换为字符进行显示

// Try-with-resources: Automatically closes FileInputStream / 自动关闭 FileInputStream
// This is better than manual closing in finally block / 这比在 finally 块中手动关闭更好


 // Read bytes until End Of File (EOF) indicated by -1 / 读取字节直到文件末尾(返回-1)
 // f.read() reads single byte and returns as int / f.read()读取单个字节并以 int 形式返回
            // Returns -1 when no more bytes to read / 当没有更多字节可读时返回-1


          // Convert the byte (0-255) to character / 将字节(0-255)转换为字符
         // WARNING: This works only for ASCII characters / 警告：这仅适用于 ASCII 字符


                // Print character without newline / 打印字符（不换行）
```

```
            }

            // FileInputStream 'f' automatically closed here due to try-with-resource
s
            // 由于try-with-resources，FileInputStream 'f' 在这里自动关闭

        } catch (FileNotFoundException e) {
// Exception when file path is wrong or file doesn't exist / 当文件路径错误或文件不存在
时的异常
            System.out.println("File not found");
 // Consider printing the actual file path for debugging / 考虑打印实际文件路径用于调试
        } catch (IOException e) {
      // Exception for read errors, disk problems, etc. / 读取错误、磁盘问题等的异常
            System.out.println("I/O Error");
       // I/O error during reading operation / 读取操作期间的I/O 错误
        }

        // Program continues here after try-catch block / 程序在try-catch 块后继续执行
    }
}
```

Demo 2:BufferedInputStream

```
package chapter12;

import java.io.BufferedInputStream;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.IOException;

public class BufferedInputStreamDemo {
    public static void main(String[] args) {
        int i;  // Stores byte value read from stream / 存储从流读取的字节值
        char c; // Converts byte to character for display / 将字节转换为字符用于
显示

       // Try-with-resources: Automatically closes both streams / 自动关闭两个流

            // Check if mark/reset is supported / 检查是否支持标记/重置功能
```

```
            // Read first 10 characters / 读取前 10 个字符
            System.out.println("Reading first 10 characters:");




    System.out.println(); // New line after printing characters / 打印字符后换行

// Mark current position with read limit of 10 bytes / 标记当前位置，读取限制为 10 字节
// This allows us to reset back to this position later / 这允许我们稍后重置回此位置..user mark()

            System.out.println("\nMark set at current position.");

            // Skip 2 characters / 跳过 2 个字符---use skip()

            System.out.println("Skipped 2 characters.");

            // Read next 7 characters after skipping / 跳过之后读取接下来的 7 个字符




            // Reset stream back to marked position / 将流重置回标记的位置


        // Read 6 characters from reset position / 从重置位置读取 6 个字符




        // Final new line / 最后换行

            System.out.println("\n=== Demo Completed ===");

        } catch (FileNotFoundException e) {
            // Handle file not found exception / 处理文件未找到异常
```

```
            System.out.println("File not found: " + e.getMessage());
            System.out.println("Please check the file path:
C:/Users/jobin/Desktop/hii.txt");

        } catch (IOException e) {
            // Handle general I/O exceptions / 处理一般 I/O 异常
            System.out.println("I/O Error: " + e.getMessage());
            e.printStackTrace(); // Print stack trace for debugging / 打印堆栈
跟踪用于调试
        }
    }
}
```

Demo 3:Filereader Class

```
package chapter12;
import java.io.FileReader;
import java.io.IOException;

/**
 * FileReader Demonstration Program
 * FileReader 演示程序
 */
public class FileReaderDemo {

    public static void main(String[] args) {

        // Using try-with-resources to automatically close the FileReader
        // 使用 try-with-resources 自动关闭 FileReader

      // Create a character array buffer of size 50 to store read characters
            // 创建大小为 50 的字符数组缓冲区来存储读取的字符


            // Read characters from file into the array
            // Returns the number of characters read, or -1 if end of file
            // 从文件中读取字符到数组中
            // 返回读取的字符数，如果到达文件末尾则返回-1


            // Iterate through each character in the array and print it
            // 遍历数组中的每个字符并打印
```

```
        } catch (IOException e) {
            // Handle IOException if file reading fails
            // 处理文件读取时的 IOException
            System.out.println("Error while reading file");
        }

    }
}
```

Demo 4:BufferdReader

```
package chapter12;
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;

/**
 * BufferedReader Demonstration Program
 * BufferedReader 演示程序
 */
public class BufferedReaderDemo {

    public static void main(String[] args) {

        // Using try-with-resources to automatically close both FileReader
and BufferedReader
        // 使用 try-with-resources 自动关闭 FileReader 和 BufferedReader

            // Read line by line until end of file (returns null)
            // 逐行读取直到文件末尾（返回 null）

                // Print each line
                // 打印每一行

            }

        } catch (IOException e) {
            // Handle IOException if file reading fails
            // 处理文件读取时的 IOException
            System.out.println("Error while reading file: " +
e.getMessage());
        }
    }
}
```

Demo 5: FileOutputStream

1.Open file

2.Write file

3.Close File.

```
package chapter12;

import java.io.FileOutputStream;
import java.io.IOException;

/**
 * FileOutputStream Demonstration Program
 * FileOutputStream 演示程序
 */
public class FileinOutputStreamdemo {

    public static void main(String[] args)
    {
        // Define the string to be written to file
        // 定义要写入文件的字符串
        String message = "Welcome to Hainan Normal university";

        // Convert the string to byte array using platform's default charset
        // 使用平台默认字符集将字符串转换为字节数组
        byte[] messageBytes = message.getBytes();

        // Using try-with-resources to automatically close FileOutputStream
        // 使用 try-with-resources 自动关闭 FileOutputStream
        //if it is true file will be written to end of the file.




            // Write each byte individually to the file
            // 将每个字节单独写入文件


            // Note: The file will be created if it doesn't exist, or
overwritten if it exists
            // 注意：如果文件不存在将会创建，如果存在将会被覆盖

        }
        catch(IOException e)
        {
            // Handle IOException if file writing fails
            // 处理文件写入时的 IOException
            System.out.println("Problem Writing to File");
        }
```

```
    }
}
```

Demo 6:BufferedOutputStream

```
package chapter12;

import java.io.BufferedOutputStream;
import java.io.FileOutputStream;
import java.io.IOException;

/**
 * BufferedOutputStream Demonstration Program
 * BufferedOutputStream 演示程序
 */
public class BufferedOutputStreamDemo {
    /**
     * Main method - program entry point
     * 主方法 - 程序入口点
     */
    public static void main(String[] args) {
        String message = "Welcome to Hainan Normal University";

        // Using try-with-resources to automatically close both
FileOutputStream and BufferedOutputStream
        // 使用 try-with-resources 自动关闭 FileOutputStream 和
BufferedOutputStream


            // Convert string to byte array
            // 将字符串转换为字节数组


            // Write bytes to buffered output stream
            // 将字节写入缓冲输出流


            // Note: Data is buffered in memory until buffer is full or
flush() is called
            // 注意：数据会在内存中缓冲，直到缓冲区满或调用 flush()方法

            // ============================================
            // FLUSH METHOD - 刷新方法
            // ============================================

            // The flush() method forces any buffered output bytes to be
written out
            // flush()方法强制将所有缓冲的输出字节立即写入底层输出流

            // Why use flush():
```

```
            // - Ensures data is physically written to disk immediately
            // - Important for critical data that cannot be lost
            // - Useful when you need to make data available to other
processes
            // - Prevents data loss in case of program termination

            // 为什么使用 flush():
            // - 确保数据立即物理写入磁盘
            // - 对于不能丢失的关键数据很重要
            // - 当需要让其他进程能够访问数据时很有用
            // - 防止程序终止时数据丢失

            bos.flush(); // Force immediate write to disk 强制立即写入磁盘

            // After flush(), the buffer is empty and all data is safely on
disk
            // 调用 flush()后，缓冲区被清空，所有数据安全地存储在磁盘上

            System.out.println("Data written and flushed successfully using
BufferedOutputStream!");

        } catch(IOException e) {
            System.out.println("Problem Writing to File: " + e.getMessage());
        }

        // Note: The close() method automatically calls flush() before
closing the stream
        // So in this case, the explicit flush() is redundant but
demonstrates the concept
        // 注意：close()方法在关闭流之前会自动调用 flush()
        // 因此在这种情况下，显式调用 flush()是冗余的，但演示了这个概念
    }
}
```

Demo 7:FileWritter

```
package chapter12;

import java.io.File;
import java.io.FileOutputStream;
import java.io.FileWriter;
import java.io.IOException;

public class FileWriterdemo {
    public static void main(String[] args){
//open file to write using Filewriter class
//write meesage into the text file.

        }
        catch(IOException e){
```

```
            System.out.println("File write error");
        }

    }
}
```

Demo 8:BufferedWriter

```
package chapter12;

import java.io.BufferedWriter;
import java.io.FileWriter;
import java.io.IOException;

public class BufferedWriterDemo {
    public static void main(String[] args) {
        System.out.println("Different fruits:");

        // Using try-with-resources to automatically close BufferedWriter
        // 使用 try-with-resources 自动关闭 BufferedWriter
        try(BufferedWriter bw = new BufferedWriter(new
FileWriter("C:/Users/jobin/Desktop/Output.txt"))) {
            String[] fruits = {"apple", "banana", "orange", "grape", "grape
fruit"};

            // Write each fruit to the file
            // 将每个水果写入文件
            for(String f : fruits) {
                bw.write(f);        // Write fruit name 写入水果名称
                bw.newLine();       // Add new line after each fruit 每个水果后
添加新行

                System.out.println("Written: " + f); // Display in console 在
控制台显示
            }

            // Optional: Flush to ensure data is written immediately
            // 可选：刷新确保数据立即写入
            bw.flush();

            System.out.println("All fruits written to file successfully!");

        } catch(IOException e) {
            System.out.println("Error writing to file: " + e.getMessage());
        }
    }
}
```

Demo 9:Object Serialization and Deserialization:

1.Create User class implements Serializable interface.

2.Create a class Perform Serialization

3.Create a class to Perform Deserialization

Demo:User Class

```
package chapter12;

import java.io.Serializable;

/**
 * User class representing a user with account credentials
 * 用户类，表示具有账户凭证的用户
 *
 * Implements Serializable to allow object serialization
 * 实现Serializable接口以支持对象序列化
 */
public class User implements Serializable {
    // Fixed serialVersionUID to prevent version mismatch during deserialization
    // 固定的序列化版本ID，防止反序列化时的版本不匹配问题
    private static final long serialVersionUID = -7364037216122078291L;

    private String account;    // User account name 用户账户名
    private String password;   // User password 用户密码
    private String email;      // User email address 用户邮箱地址

    /**
     * Constructor to initialize User object
     * 构造函数，初始化用户对象
```

```
     *
     * @param account  user account name 用户账户名
     * @param password user password 用户密码
     * @param email    user email address 用户邮箱地址
     */
    public User(String account, String password, String email) {
        this.account = account;
        this.password = password;
        this.email = email;
    }

    // Getter methods 获取方法
    public String getAccount() {
        return account;
    }

    public String getpassword() {
        return password;
    }

    public String getemail() {
        return email;
    }

    // Setter method for password 密码设置方法
    public void setPassword(String password) {
        this.password = password;
    }

    public String getEmail() {
        return email;
    }

    // Note: Consider overriding toString() method for better object representation
    // 注意：建议重写toString()方法以便更好地表示对象
}
```

Demo Serialization

```
package chapter12;
```

```java
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.ObjectOutputStream;

/**
 * SerializationDemo class demonstrates object serialization
 * 序列化演示类，展示对象序列化过程
 *
 * Serialization: Converting Java objects to byte stream for storage/transmission
 * 序列化：将 Java 对象转换为字节流以便存储或传输
 */
public class SerializationDemo {
    public static void main(String[] args) {
        // Create a User object 创建用户对象
        User user1 = new User("tom123", "hello@123", "xyz@live.com");

        try {
            // Create FileOutputStream to write to file 创建文件输出流以写入文件

     // Create ObjectOutputStream for object serialization 创建对象输出流进行对象序列化


            // Serialize and write the User object to file 序列化并将用户对象写入文件


         // Important: Close streams to release resources and ensure data is flushed
            // 重要：关闭流以释放资源并确保数据被刷新


            System.out.println("Object serialized successfully!");
            System.out.println("对象序列化成功！");

        } catch(IOException e) {
            System.out.println("File write error: " + e.getMessage());
            System.out.println("文件写入错误: " + e.getMessage());
        }
    }
```

```
}
```

Deserialization Demo

```
package chapter12;

import java.io.FileInputStream;
import java.io.IOException;
import java.io.ObjectInputStream;

/**
 * DeserializationDemo class demonstrates object deserialization
 * 反序列化演示类，展示对象反序列化过程
 *
 * Deserialization: Reconstructing Java objects from byte stream
 * 反序列化：从字节流重建Java对象
 */
public class DeserializationDemo {
    public static void main(String[] args) {
        try {
            // Create FileInputStream to read from file 创建文件输入流以读取文件


            // Create ObjectInputStream for object deserialization 创建对象输入流进行
对象反序列化

            // Deserialize and read the User object from file 反序列化并从文件读取用户
对象
                // Explicit casting required 需要显式类型转换

            // Display user information 显示用户信息
            System.out.println("User object deserialized successfully!");


            // Important: Close streams to release resources
            // 重要：关闭流以释放资源
```

```
        } catch (IOException e) {
            // Improved error message 改进的错误信息
            System.out.println("File read error: " + e.getMessage());

        } catch (ClassNotFoundException e) {
            System.out.println("Class not found: " + e.getMessage());

        }
    }
}
```