3.11 Reading and Writing files

Java provides a number of classes and methods that allow you to read and write files.
There are two stream classes

1. **FileInputStream**
2. **FileOutputStream**

These above classes are used to create byte streams linked to files.

FileInputStream(String *fileName*) throws FileNotFoundException
FileOutputStream(String *fileName*) throws FileNotFoundException

Where
fileName specifies the name of the file that want to open. When you create an input stream, if the file does not exist, then **FileNotFoundException** is thrown. For output streams, if the file cannot be opened or created, then **FileNotFoundException** is thrown. **FileNotFoundException** is a subclass of **IOException**. When an output file is opened, any preexisting file by the same name is destroyed.

2.1.1 Java FileInputStream Class

Java FileInputStream class obtains input bytes from a file. It is used for reading byte-oriented data (streams of raw bytes) such as image data, audio, video etc. You can also read character-stream data.

Java FileInputStream class declaration

public class FileInputStream extends InputStream

Java FileInputStream class methods

Method	**Description**
int available()	It is used to return the estimated number of bytes that can be read from the input stream.
int read()	It is used to read the byte of data from the input stream.
int read(byte[] b)	It is used to read up to **b.length** bytes of data from the input stream.
int read(byte[] b, int off, int len)	It is used to read up to **len** bytes of data from the input stream.

long skip(long x)	It is used to skip over and discards x bytes of data from the input stream.
FileChannel getChannel()	It is used to return the unique FileChannel object associated with the file input stream.
FileDescriptor getFD()	It is used to return the FileDescriptor object.
protected void finalize()	It is used to ensure that the close method is call when there is no more reference to the file input stream.
void close()	It is used to closes the stream.

Table 3.6 Java FileInputStream Class Methods

Example Program1:

```
import java.io.FileInputStream;
public class DataStreamExample
{
public static void main(String args[])
{
try
{
FileInputStream fin=new FileInputStream("D:\\testout.txt");
int i=fin.read();
System.out.print((char)i);
fin.close();
}
catch(Exception e)
{
System.out.println(e);
}
}
}
```

Before running the code, a text file named as **"testout.txt"** is required to be created. In this file, we are having following content:

testout.txt

Welcome to Java Stream Classes.

Output:

```
W
```

Example Program2

```
import java.io.FileInputStream;
public class DataStreamExample
{
public static void main(String args[])
{
try
{
FileInputStream fin=new FileInputStream("D:\\testout.txt");
int i=0;
while((i=fin.read())!=-1)
{
System.out.print((char)i);
}
fin.close();
}
catch(Exception e)
{
System.out.println(e);
}
}
}
```

testout.txt
Welcome to Java Stream Classes.

Output:

```
Welcome to Java Stream Classes.
```

2.1.2 Java FileOutputStream Class

Java FileOutputStream is an output stream used for writing data to a file. You can write byte-oriented as well as character-oriented data through FileOutputStream class. But, for character-oriented data, it is preferred to use FileWriter than FileOutputStream.

FileOutputStream class declaration
public class FileOutputStream extends OutputStream

FileOutputStream class methods

Method	Description
protected void finalize()	It is used to clean up the connection with the file output stream
void write(byte[] ary)	It is used to write **ary.length** bytes from the
	byte array to the file output stream.
void write(byte[] ary, int off, int len)	It is used to write **len** bytes from the byte array starting at offset **off** to the file output stream.
void write(int b)	It is used to write the specified byte to the file output stream.

FileChannel getChannel()	It is used to return the file channel object associated with the file output stream.
FileDescriptor getFD()	It is used to return the file descriptor associated with the stream.
void close()	It is used to close the file output stream.

Table 3.7 Java FileOutputStream Class Methods

Example Program1:

```
import java.io.FileOutputStream;
public class FileOutputStreamExample
{
public static void main(String args[])
{
try
{
FileOutputStream fout=new FileOutputStream("D:\\testout.txt");
fout.write(65);
fout.close();
System.out.println("success...");
}
catch(Exception e)
{
System.out.println(e);
}
}
}
```

Output:

```
success...

textout.txt
A
```

Example Program2:

```
import java.io.FileOutputStream;
public class FileOutputStreamExample
{
public static void main(String args[])
{
try
{
FileOutputStream fout=new FileOutputStream("D:\\testout.txt");
String s="Welcome to javaTpoint.";
byte b[]=s.getBytes();//converting string into byte array fout.write(b);
fout.close();
System.out.println("success...");
}
catch(Exception e)
{
System.out.println(e);
}
}
}
```

Output:

```
success...
```

testout.txt

```
Welcome to Java Stream Classes.
```

2 Mark Questions and Answers

1. What is an Exception?

A Java exception is an object that finds an exceptional condition occurs from a piece of code. An exception object is created and thrown to the method from the code where an exception is found.

2. Write down the purpose of exception handling mechanism.

The main purpose of exception handling mechanism is used to detect and report an "exceptional circumstance" so that necessary action can be taken. It performs the following tasks

5. Find the problem(Hit the exception)
6. Inform that an error occurred(throw the exception).
7. Receive the error information(Catch the exception)
8. Take corrective actions(Handle the exception)

3. What are the types of exceptions?

There are two types of exceptions

1. Predefined Exceptions-The Exceptions which are predefined are called predefined exceptions
2. Userdefined Exceptions- The Exceptions which are defined by the user are called userdefined exceptions

4. How the exception handling is managed?

Java exception handling is managed via five keywords.

- try
- catch
- throw
- throws and
- finally

5. Write down the general form of an exception-handling block.

The general form of an exception-handling block

```
try
{
// block of code to monitor for errors
}
catch (ExceptionType1 exOb)
{
// exception handler for ExceptionType1
}
catch (ExceptionType2 exOb)
{
// exception handler for ExceptionType2
}
// ...
finally
{
// block of code to be executed after try block ends
}
```

6. What are the two subclasses under Throwable class?

Throwable is a superclass for all exception types. Thus, Throwable is at top of the exception hierarchy.
There are two subclasses under Throwable class.

3. Exception
4. Error

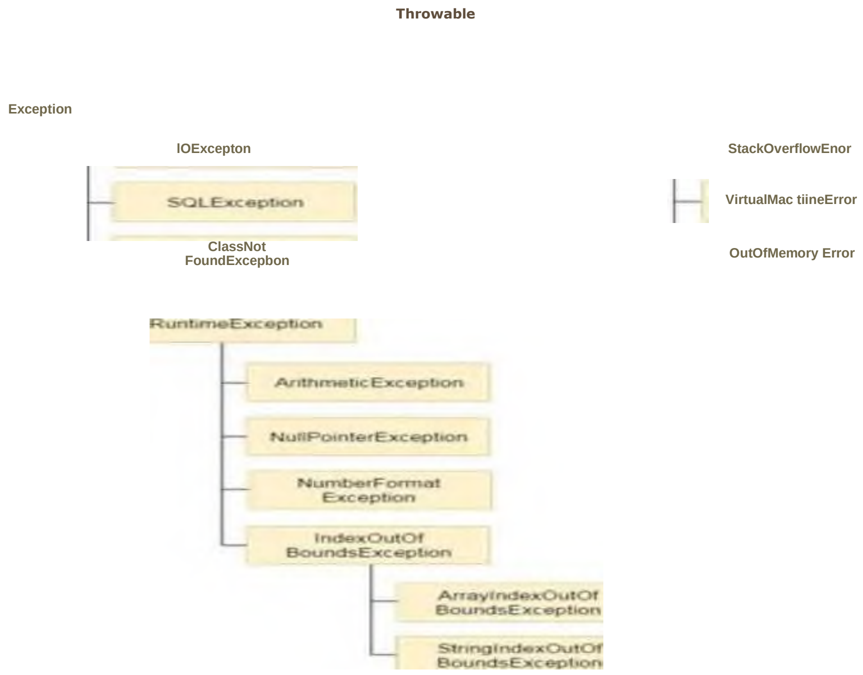
7. What is an Error?

The another subclass is by **Error**, which defines exceptions that are not expected to be caught under normal circumstances by your program.

Exceptions of type **Error** are used by the Java run-time system to indicate errors having to do with the run-time environment, itself.
Eg: Stack overflow is an example of such an error.

8. What is an Exception?
This class is used for exceptional conditions that user programs should catch. We can also subclass this class to create own custom exception types.
The important subclass of this class is **RuntimeException**.
RuntimeException
Exceptions of this type are automatically defined for the programs that you write and include thing such as division by zero and invalid array indexing.

9. Draw the diagrammatical representation for Exception Hierarchy



10. Write down the use of try and catch block in exception handling.
The try block allows us to fix the errors. Catch block prevents the program from automatically terminating. To handle a run-time error, enclose the code to be monitored inside a try block. After the try block, include a catch block that specifies the exception type that needs to be caught.
Syntax:
try
{
statement;
}
catch(Exception-type exOb)
{
statement;

}

11. Explain the situation where we need to use multiple catch clauses.

In some situation, more than one exception can occur by a single piece of code. To handle this situation, we can use two or more catch clauses, each catching a different type of exception. When an exception is thrown, each catch block is executed in order, and the first one whose type matches that exception is executed.After one catch block executes, the others are bypassed, and continues after the try/catch block.

12. Write down the syntax for multiple catch clauses.

The syntax for multiple catch clauses

```
try
{
// block of code to monitor for errors
}
catch (ExceptionType1 exOb)
{
// exception handler for ExceptionType1
}
catch (ExceptionType2 exOb)
{
// exception handler for ExceptionType2
}
```

13. Explain the situation where we need to use nested try statements.

The **try** statement can be nested. That is, a **try** statement can be inside the block of another **try**. Each time a **try** statement is entered, the context of that exception is pushed on the stack. If an inner **try** statement does not have a **catch** handler for a particular exception, the stack is unwound and the next **try** statement's **catch** handlers are inspected for a match. This continues until one of the **catch** statements succeeds, or until all of the nested **try** statements are exhausted. If no **catch** statement matches, then the Java run-time system will handle the exception.

14. Write down the syntax for nested try statement.

The syntax for nested try statement:

```
//Main try block
try
{
statement 1;
statement 2;
//try-catch block inside another try block
try
{
```

```
statement 3;
statement 4;
//try-catch block inside nested try block
try
{
statement 5;
statement 6;
}
catch(Exception e2)
{
//Exception Message
}
}
catch(Exception e1)
{
//Exception Message
}
}
//Catch of Main(parent) try block
catch(Exception e3)
{
//Exception Message
}
```

15. Write down the use of throw statement.

The **throw** keyword in **Java** is used to explicitly **throw** an exception from a method or any **block** of code. We can **throw** either checked or unchecked exception.

Syntax:

```
throw new exception_class("error message");
```

For example:

```
throw new ArithmeticException("dividing a number by 5 is not allowed in this program");
```

16. Write down the use of throws clause.

Using throws clause, We can list the types of exceptions that a method might throw.The exceptions which are thrown in a method might be using **throws** clause. If they are not, a compile-time error will result.

Syntax:

```
type method-name(parameter-list) throws exception-list
{
// body of method
}
```

Exception-list is a comma-separated number of exceptions that a method can throw.

17. Write down the use of finally clause.

finally creates a block of code that is to be executed after a try/catch block has completed its execution.The finally block will execute if an exception is thrown or not thrown. The finally clause is optional.Each try block requires either one catch or a finally clause

Syntax:

```
try
{
   //Statements that may cause an exception
}
catch
{
  //Handling exception
}
finally
{
  //Statements to be executed
}
```

18. What is Unchecked Exception?

These are the exceptions that are not checked at compiled time. In C++, all exceptions are unchecked, so it is not forced by the compiler to either handle or specify the exception. It is up to the programmers to be civilized, and specify or catch the exceptions.

19. What is Checked Exception?

These are the exceptions that are checked at compile time. If some code within a method throws a checked exception, then the method must either handle the exception or it must specify the exception using *throws* keyword.

20. Write down some of the unchecked exceptions in RuntimeException?

The unchecked exceptions in RuntimeException are ArithmeticException, ArrayIndexOutOfBoundsException, ArrayStoreException, ClassCastException, EnumConstantNotPresentException, IllegalArgumentException, IllegalMonitorStateException, IllegalStateException, IllegalThreadStateException, IndexOutOfBoundsException, NegativeArraySizeException, NullPointerException, NumberFormatException, SecurityException, StringIndexOutOfBoundsException, TypeNotPresentException, UnsupportedOperationException

21. Write down some of the checked exceptions in RuntimeException?

The checked exceptions in RuntimeException ClassNotFoundException, CloneNotSupportedException, IllegalAccessException, InstantiationException, InterruptedException, NoSuchFieldException, NoSuchMethodException, ReflectiveOperationException

22. Explain the way to create own exceptions.

We can throw our own exceptions using throw keyword.

Syntax:

throw new Throwable_subclass;

Eg:

throw new ArithmeticException;

23. Define Stack Trace Elements.

The StackTraceElement is a class that describes a single stack frame,which is an element of a stack trace when an exception occurs.The getStackTrace() method is used to return an array of StackTraceElements.

Each stack frame contains the following

5. the class name
6. the method name
7. The file name
8. And the source-code line number

24. List out the methods in StackTraceElements

The methods in StackTraceElements are

- boolean equals(Object *ob*)
- String getClassName( )
- String getFileName( )
- int getLineNumber( )
- String getMethodName( )
- int hashCode( )
- boolean isNativeMethod( )
- String toString( )

25. What is meant by a stream?

A stream is an abstraction that either produces or consumes information.

A stream is linked to a physical device by the java I/O system.The input stream may abstract many different kinds of input: from a disk file,a keyboard,or a network socket. Likewise,an output stream may refer to the console such as a disk file, or a network connection.

26. What are the two types of streams?

There are two types of streams

3. Byte streams
4. Character streams

27. What are the predefined stream variables in System class?

System class contains three predefined stream variables:

4. in
5. out
6. err

System.in refers to the standard input stream. System.out refers to the standard output stream.System.err refers to the standard error stream.System.**System.in** is an object of type **InputStream**; **System.out** and **System.err** are objects of type **PrintStream**. These are byte streams, they are typically used to read and write characters from and to the console.

28. What is meant by a byte stream?

Byte Streams provides a convenient way for handling input and output of bytes. When reading or writing binary data, byte streams are used.

29. What are abstract classes in byte streams?

There are two abstract classes defined in byte streams

3. InputStream
4. OutputStream

30. List out some of the Byte Stream Classes.

BufferedInputStream, BufferedOutputStream, ByteArrayInputStream, ByteArrayOutputStream, DataInputStream, DataOutputStream, FileInputStream, FileOutputStream, InputStream, OutputStream

31. What is meant by a Character stream?

Character Streams provides a convenient way for handling input and output of characters.

32. What are abstract classes in Character streams?

There are two abstract classes defined in byte streams

3. Reader
4. Writer

33. List out some of the Character Stream Classes.

BufferedReader, BufferedWriter, CharArrayReader, CharArrayWriter, FileReader, FileWriter, Reader, Writer

34. How to read a character from BufferedReader Class?

To read a character from a BufferedReader,We can use read() method.

Syntax:
int read( ) throws IOException

Whenever read() method is called, it reads a character from the input stream and returns an integer value.It returns -1 when the end of the stream is encountered.

35. How to read the string from BufferedReader Class?
To read a string from the keyboard,we can use readLine() method.readLine() is a member of the BufferedReader class.
Syntax:
String readLine( ) throws IOException

36. What is the use of PrintWriter class?
PrintWriter is one of the character-based classes.
Constructor:
PrintWriter(OutputStream *outputStream*, boolean *flushOnNewline*)

ouputStream is an object of OutputStream class.flushOnNewline controls when Java flushes the output stream every time a println() method is called. If *flushOnNewline* is **true**, flushing automatically takes place. If **false**, flushing is not automatic.
To write to the console by using a **PrintWriter**, specify **System.out** for the output stream and flush the stream after each newline.

37. Write the use of FileInputStream class
Java FileInputStream class obtains input bytes from a file. It is used for reading byte-oriented data (streams of raw bytes) such as image data, audio, video etc. You can also read character-stream data.

38. Write down the methods in FileInputStream Class.
The methods in FileInputStream class are
- int available()
- int read()
- long skip(long x)
- FileChannel getChannel()
- FileDescriptor getFD()
- protected void finalize()
- void close()

39. Write the use of FileOutputStream class.
Java FileOutputStream is an output stream used for writing data to a file. You can write byte-oriented as well as character-oriented data through FileOutputStream class. But, for character-oriented data, it is preferred to use FileWriter than FileOutputStream.

40. Write down the methods in FileOutputStream Class.

The methods in FileOutputStream class are

- protected void finalize()
- void write(byte[] ary)
- void write(int b)
- FileChannel getChannel()
- FileDescriptor getFD()
- void close()

13 Marks Questions

1. Explain the Throwing and Catching Exception.
2. What is exception? How to throw an exception? Give an example.
3. What is finally class? How to catch exceptions? Write an example.
4. What is meant by exceptions? Why it is needed? Describe the exception hierarchy.
5. Write note on Stack Trace Elements. Give example.
6. Define Exception and explain its different types with example.
7. Discuss about character stream classes.
8. With suitable coding discuss all kinds of exception handling.
9. Write a note on java.io package with its stream classes and methods in it.
10. Write a Java program to demonstrate I/O character stream classes.
11. Write short notes on PrintStream class.
12. Explain how user-defined exception subclasses are created in Java.

15 Marks Questions

1. What is the necessity of exception handling? Explain exception handling taking "Divide -by Zero" as an example.
2. What is an exception? Create your own exception for "Temperature>40" in an "Atomic Reactor Based Application" and write appropriate exception handling code in the main program.
3. Write a Java program to copy the data from one file to another file.
4. Write a program that uses a sequence input stream to output the contents of two files.
5. List the five keywords in Java exception handling. Describe the use of the keywords.