I/O Streams (Contd.).

- Java's stream classes are defined in the **java.io** package.
- Java 2 defines two types of streams:
  - **byte streams**
  - **character streams**
- Byte streams:
  - provide a convenient means for handling input and output of bytes
  - are used for reading or writing binary data
- Character streams:
  - provide a convenient means for handling input and output of characters
  - use **Unicode**, and, therefore, can be internationalized

The Predefined Streams

- System class of the java.lang package contains three predefined stream variables, in, out and err.
- These variables are declared as public and static within System:
  - System.out refers to the standard output stream which is the console.
  - System.in refers to standard input, which is the keyboard by default.
  - System.err refers to the standard error stream, which also is the console by default.

Difference between System.out and System.err

- System.out sends the output to the standard output stream, which is console by default.
- System.err sends the output to the standard error stream, which also happens to be console by default
- The reason behind having two separate streams for output and error is that the standard o utput should be used for regular program outputs while standard error should be used for error messages.

Demonstration of System.out and System.err

```
class StreamDemo {
    public static void main(String[] args) {
        try {
    System.out.print("Writing program output to the output file ");
    int i=0;
    int z=100/i;
        }
        catch(Exception e) {
    System.err.print("ArithmeticException has occured");
        }
    }
}
```

System Properties

- System Properties provide information about local system configuration.
- When the Java Virtual Machine starts, it inserts local System Properties into a System properties list.

- We can use methods defined in System class to access or change the values of these properties.

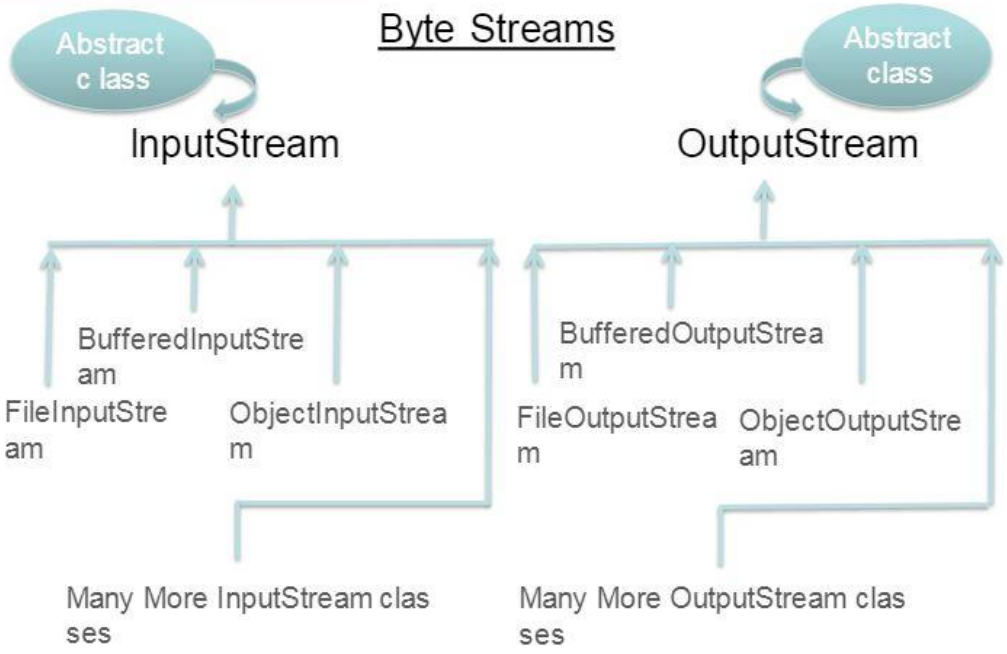
```
public static Properties getProperties()
public static String getProperty(String key)

public static void setProperties(Properties prp)
```

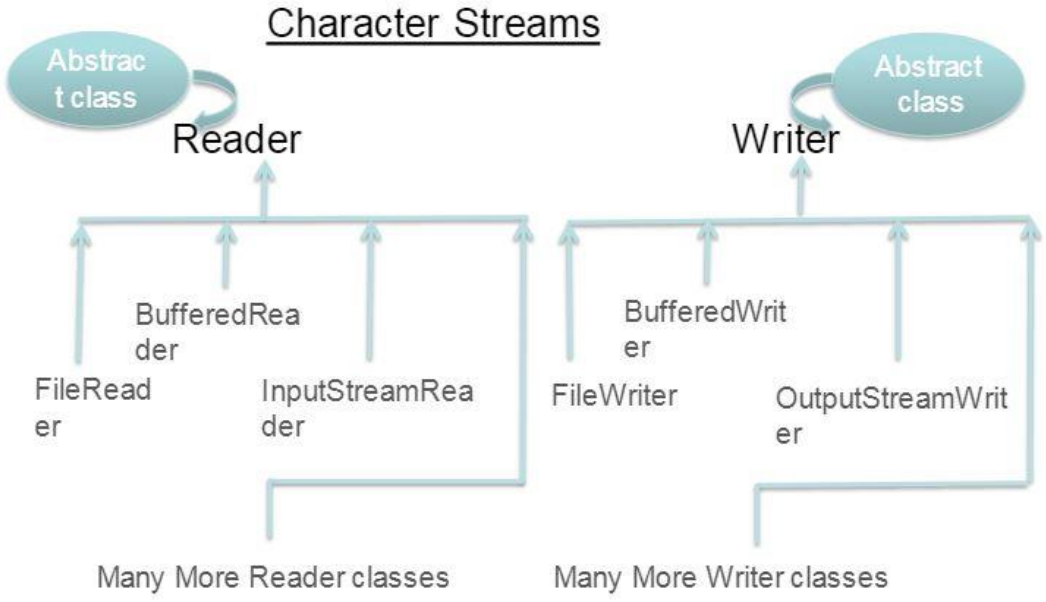
Some of the Important Properties are listed below :

Key	Description of Associated Value
java.version	Java Runtime Environment version
java.home	Java installation directory
java.class.path	Java class path
os.name	Operating system name
user.name	User's account name
user.home	User's home directory
user.dir	User's current working directory

I/O Streams hierarchy



I/O Streams hierarchy (Contd.).



Byte Stream classes

To read & write data into buffer

To read & write object into secondary device (serialization)

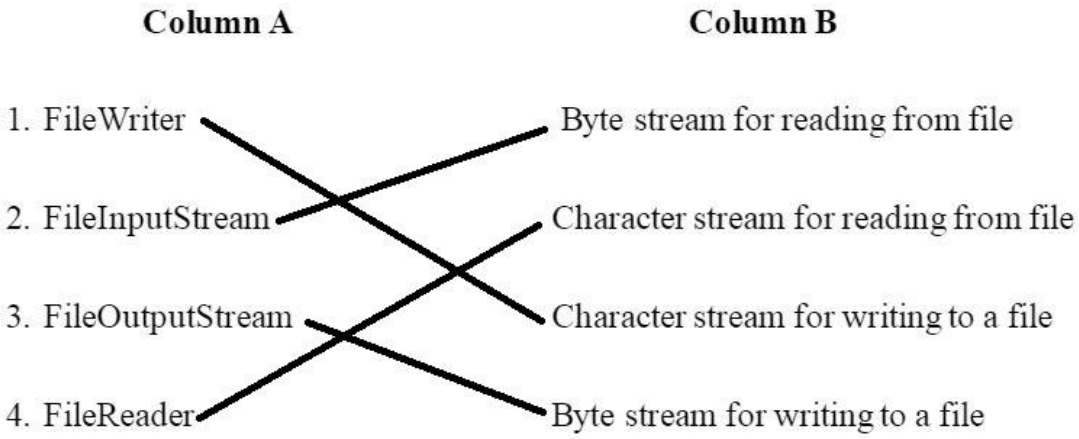
Character Stream classes

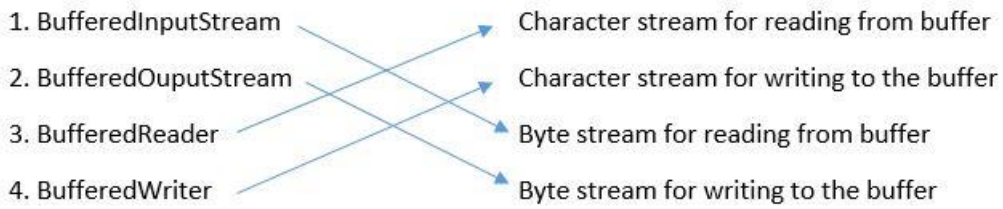
To read & write data into buffer

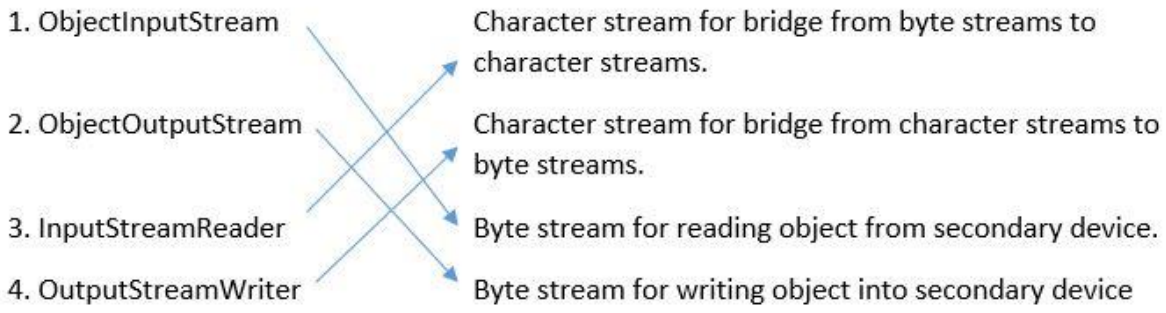
Bridge from character stream to byte stream

Match the following

- Match the streams with the appropriate phrases in column B







Reading Console Input - Stream Wrapping

- The preferred method of reading console input in Java 2 is to use a character stream
- *InputStreamReader* class acts as a bridge between byte and character streams
- Console input is accomplished by reading from System.in
- To get a character-based stream, you wrap ***System.in*** in a BufferedReader object

Reading Console Input - Stream Wrapping

- The **BufferedReader** class supports a buffered input stream. Its most commonly used constructor is shown as follows:

- **BufferedReader(Reader *inputReader*)**

- Here *inputReader* is the stream that is linked to the instance of **BufferedReader** that is being created. **Reader** is an abstract class. One of its concrete subclasses is **InputStreamReader**, which converts bytes to characters. To obtain an **InputStreamReader** object that is linked to **System.in**, use the following constructor:

- **InputStreamReader(InputStream *inputStream*)**

Reading Console Input - Stream Wrapping

Because **System.in** refers to an object of type **InputStream**, it can be used for *inputStream*. Putting it all together, the following line of code creates a **BufferedReader** that is connected to the keyboard, and which in turn enables character input from a byte stream InputStream that is System.in).

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

Reading Characters

```
package m10.io;
import java.io.*;

public class BRRead{

  public static void main (String args[ ]) throws IOException {
        char c;
        BufferedReader br = new BufferedReader(new
                        InputStreamReader(System.in));
        System.out.println("Enter Characters, 'q' to quit");
        do {
                    c = (char) br.read( );
                  System.out.println( c );
        }while (c != 'q');
    }
}
```

Refer documentation for BufferedReader and InputStreamReader

Reading Characters

- **int read( ) throws IOException**

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

Reading Strings

```
package m10.io;
import java.io.*;

public class BRReadLine{

  public static void main (String args[]) throws IOException {
        String str;
        BufferedReader br = new BufferedReader(new
                        InputStreamReader(System.in));
        System.out.println("Enter Characters, 'stop' to quit");
                do {
                    str = br.readLine( );
                  System.out.println ( str );
        }while (!str.equals( "stop"));
  }
}
```

*The above program reads and displays lines of text until you enter the word "**stop**".*

Writing Console Output

- **print( )** and **println( )** are console output methods defined in PrintStream class
- **System.out** is a byte stream used to write bytes

Reading & Writing to File using FileReader & FileWriter

```
package m10.io;
import java.io.*;

public class Copy {

public static void main(String[] args) throws  IOException {

                File inputFile = new File("Source.txt");
                File outputFile = new File("Target.txt");
                FileReader in = new FileReader(inputFile);
                FileWriter out = new FileWriter(outputFile);
                int c;
                while ((c = in.read()) != -1)
                    out.write(c);

                in.close();
                out.close();
    }
}
```

Refer documentation for FileReader and FileWriter