Object Oriented Development with Java

(CT038-3-2 and Version VC1)

Managing File IO

Java File Operation

Topic & Structure of the lesson

- File Class
  - Reading & Writing Data from & to Files
  - I/O Exception
- I/O Stream
  - Text Files & Binary Files
  - Text I/O
    - FileReader & FileWriter
    - BufferedReader & BufferedWriter
    - PrintWriter & PrintStream
    - Scanner
  - Binary I/O
    - FileInputStream & FileOutputStream
    - DataInputStream & DataOutputStream
    - BufferedInputStream & BufferedOutputStream

Learning outcomes

- At the end of this lecture you should be able to:
  - Understand IO processing
  - Explain text I/O and binary I/O
  - Work with various Stream class for IO operations

Key terms you must be able to use

If you have mastered this topic, you should be able to use the following terms correctly in your assessments:

- FileReader & FileWriter
- BufferedReader & BufferedWriter
- PrintWriter & PrintStream
- FileInputStream & FileOutputStream
- DataInputStream & DataOutputStream
- BufferedInputStream & BufferedOutputStream

Introduction

- Data stored in variables, arrays, and objects are temporary
- To save data permanently, store the data in a file on a disk or CD
- The files can be transported or read by other programs

`File` **class**

- Every file is placed in a directory in the file system
- Complete filename contains: *directory path* and *filename*
- *E.g. C:book/Welcome.java*
- It contains useful operations/methods
- E.g. obtain file properties, rename file, delete file, etc.

How is I/O handled in Java?

- `File` object encapsulates the properties of a file or a path, but doesn’t contain methods for reading/writing data from/to a file
- To perform I/O, we need to use Java I/O classes
- E.g. FileWriter and FileReader

Writing & Reading Data to the File

```
FileWriter output = new
  FileWriter("temp.txt");
output.write("Java IO operations");
output.close();

FileReader input = new
  FileReader("temp.txt");
int code = input.read();
System.out.println((char)code);
input.close();
```

I/OException

- All methods in the I/O classes throw java.io.IOException
- Therefore you have to declare to throw this exception in the method or place the code in a try-catch block

IOException

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

  FileWriter output = new FileWriter("temp.txt");
  output.write("Java 1 2 3");
  output.close();

  FileReader input = new FileReader("temp.txt");
  int code = input.read();
  System.out.println((char)code);
  input.close();
}
```

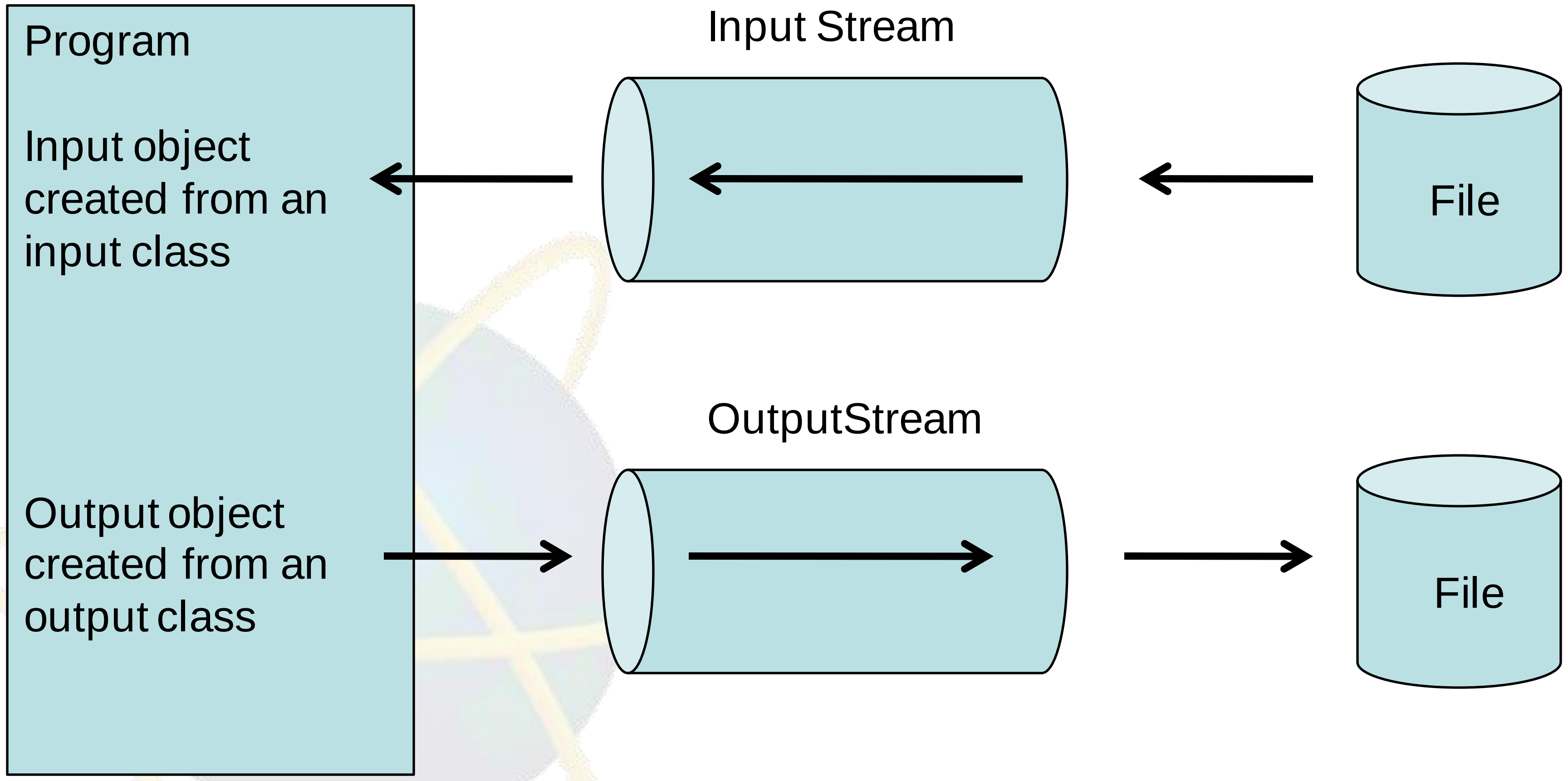
IOException - Alternatively

```
public static void main(String []args){
 try {
  FileWriter output = new FileWriter("temp.txt");
  output.write("Java 1 2 3");
  output.close();
  FileReader input = new FileReader("temp.txt");
  int code = input.read();
  System.out.println((char)code);
  input.close();
 } catch(IOException e) {
     e.printStackTrace();
 }
}
```

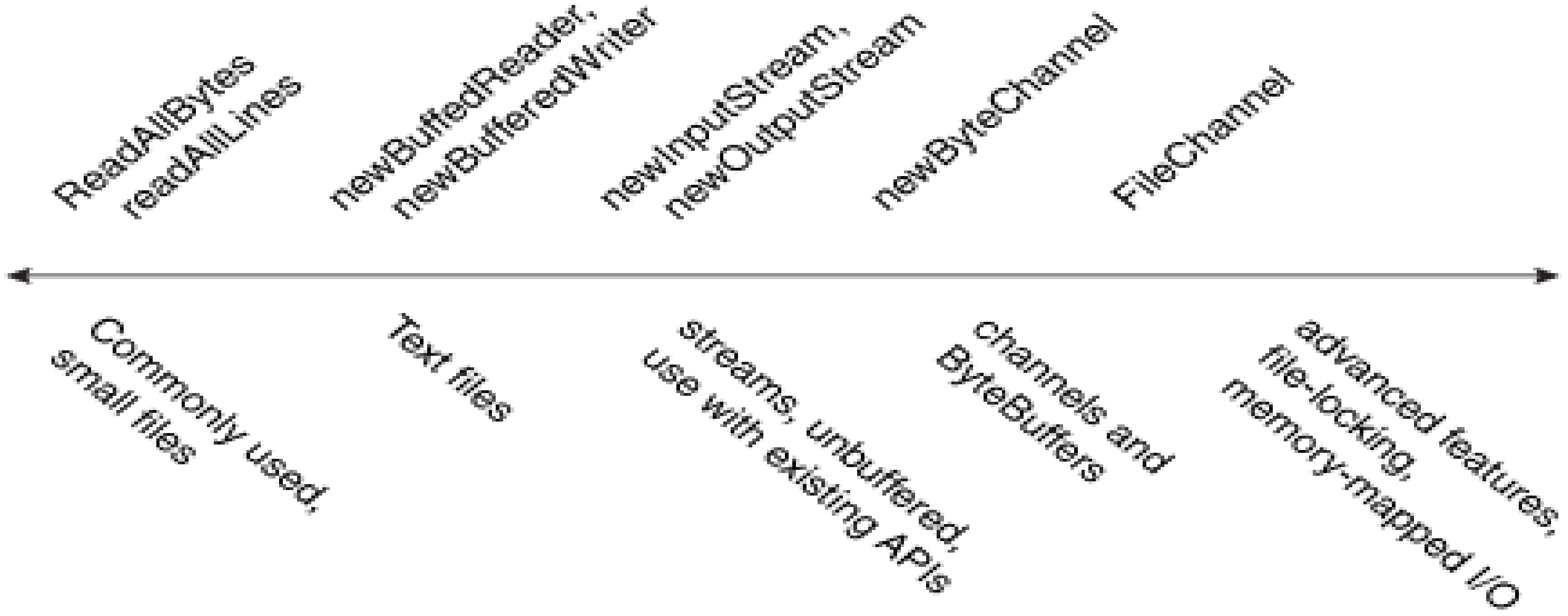
I/O Stream

- Sequence of data
- InputStream (read data from source) and OutputStream (write data to destination)
- Common streams:
  - Byte Streams
    - FileInputStream and FileOutputStream
  - Character Streams
    - FileReader and FileWriter
  - Standard Streams
    - System.in, System.out & System.err
- 3 operations: open, read & close

I/O Stream


Program
Input object created from an input class
Output object created from an output class
Input Stream
OutputStream
File
File

Complexity of File I/O Methods



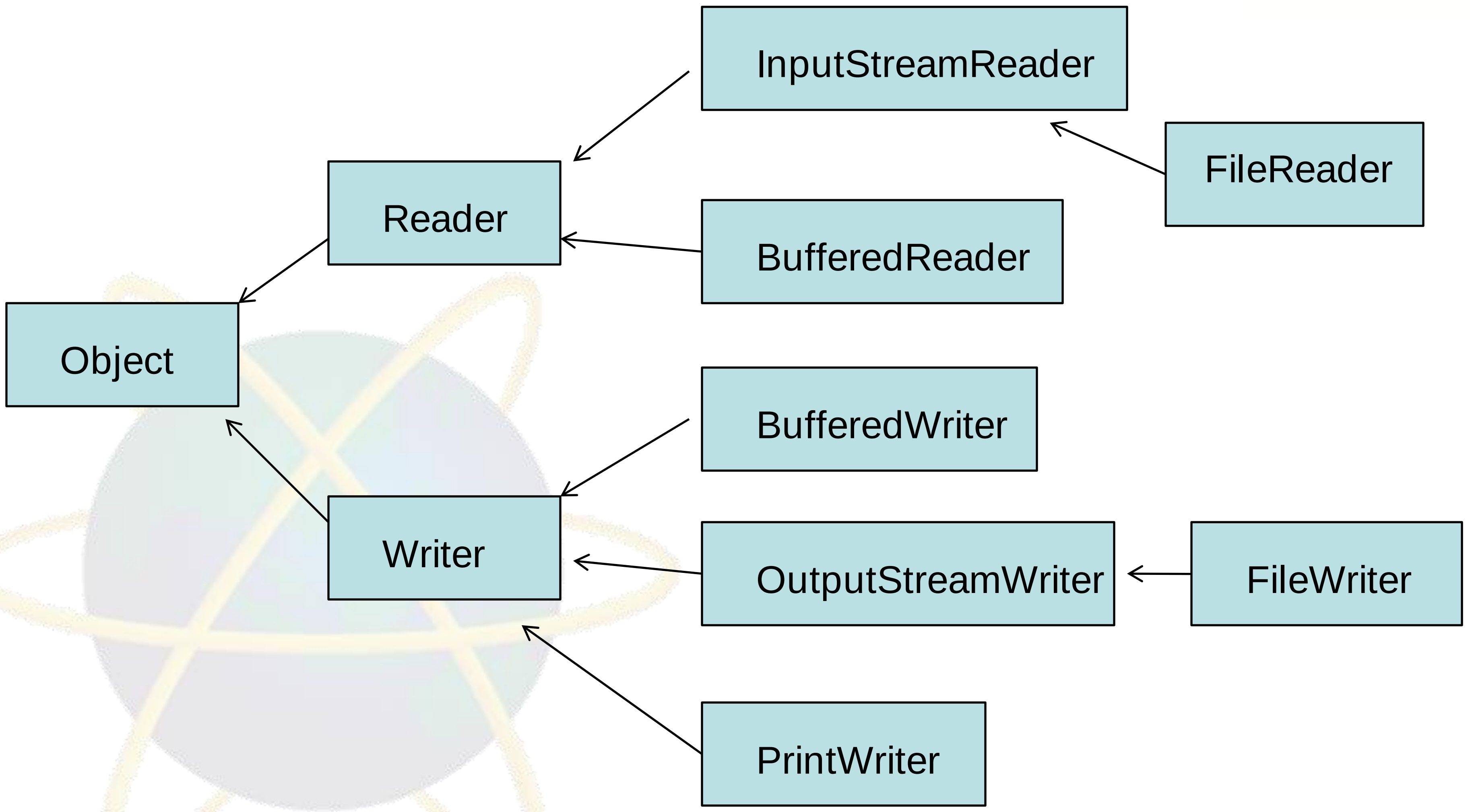
File I/O Methods Arranged from Less Complex to More Complex

Source: https://docs.oracle.com/javase/tutorial/essential/io/file.html

Text Files and Binary Files

- There are text I/O classes and binary I/O classes
- **Text I/O**: data stored in a text file are represented in human-readable form ie. a sequence of character data
- **Binary I/O**: data stored in a binary file are represented in binary form (ie. does not use character encoding)
- Use depends on:
  - – Importance of size/speed (binary usually smaller & faster)
  - – Portability (text files transfer easily)

Text I/O



Object
Reader
Writer
InputStreamReader
FileReader
BufferedReader
BufferedWriter
OutputStreamWriter
FileWriter
PrintWriter

FileReader / FileWriter

- For reading / writing characters from / to files
- They associate an input/output stream with an external file
- Constructor:

```
public FileReader(String filename)
public FileReader(File file)
public FileWriter(String filename)
public FileWriter(File file)
public FileWriter(String filename,
  boolean append)
public FileWriter(File, boolean
```

FileReader

```
public static void main(String [] args){
 FileReader input = null;
 try{
  //create an input stream
  input = new FileReader("temp.txt");
  int code;
  //repeatedly read a character and display it
  on the console
  while((code = input.read()) != -1){
   System.out.println((char)code);
  }
 }
```

contd. to next slide

FileReader

```
catch(FileNotFoundException ex){
   System.out.println("File temp.txt does not
       exist"); }
catch(IOException e) {
   e.printStackTrace();
}
finally {
   try {
    input.close();//close the stream
   }
catch(…){}
  }
}
```

FileWriter

```
public static void main(String []args){
 //create an output stream to the file
 FileWriter output = new
  FileWriter("temp.txt", true);

 //output a string to the file
 output.write("This is a sample line.");

 //close the stream
 output.close();
}
```

BufferedReader / BufferedWriter

- To speed up input and output by reducing the number of reads and writes
- Buffered stream employ a buffered array of characters that acts as a cache
- E.g.: the array reads a chunk of characters into the buffer before the individual characters are read

BufferedReader / BufferedWriter

- Constructors:

```
//create a BufferedReader
public BufferedReader(Reader in)
public BufferedReader(Reader in, int
  bufferSize)


//create a BufferedWriter
public BufferedWriter(Writer out)
public BufferedWriter(Writer out, int
  bufferSize)
```

BufferedReader / BufferedWriter

- The default buffer size is 8192 characters
- Read as many data as possible into its buffer in a single read call
- By contrast, buffered output stream calls the write method only when its buffer fills up or when `flush()` method is called

BufferedReader / BufferedWriter

```
public static void main(String []args) throws
   IOException{
  //create an input stream
  BufferedReader input = new BufferedReader(new
   FileReader("temp.txt"));

  //create an output stream
  BufferedWriter output = new
   BufferedWriter(new FileWriter("temp2.txt"));
```

contd. to next slide

BufferedReader / BufferedWriter

```
 //repeatedly read a line and display it on the
  console
 String line;
 while((line = input.readLine()) != null){
   System.out.println(line);
   output.write(line);
   output.newLine(); //write a line separator
 }//while

input.close();
 output.close();
}
```

PrintWriter and PrintStream

- BufferedWriter is used to output characters and strings
- PrintWriter and PrintStream can be used to output objects, strings, and numeric values as text
- PrintWriter (Java 2) replaced PrintStream
- Both classes are identical
- PrintWriter is more efficient then PrintStream and is recommended to use

PrintWriter and PrintStream

- Both provide:

```
public void print(Object o)
public void print(String s)
public void print(char c)
public void print(char[] array)
public void print(int i)
public void print(long l)
public void print(float f)
public void print(double d)
public void print(boolean b)
```

PrintWriter and PrintStream

- Both provide:

```
public void println(Object o)
public void println(String s)
public void println(char c)
public void println(char[] array)
public void println(int i)
public void println(long l)
public void println(float f)
public void println(double d)
public void println(boolean b)
```

PrintWriter and PrintStream

- Also contain the `printf` method
- PrintWriter constructor:

```
public PrintWriter(Writer out)
public PrintWriter(Writer out,
  boolean autoFlush)
```

PrintWriter and PrintStream

```
public static void main(String [] args)
   throws IOException {
 //check if file temp.txt already exits
 File f = new File("temp.txt");
 if(f.exists()){
   System.out.println("File temp.txt already
   exist.");
   System.out.exit(0);
 }
```

contd. to next slide

PrintWriter and PrintStream

```
//create an output stream
PrintWriter output = new PrintWriter(new
 FileWriter(f));

//generate ten integers and write them to a file
for(int i = 0; i < 10; i++){
 output.print((int)(Math.random() * 100) + “ ");
}
output.close();
```

PrintWriter and PrintStream

```
//open an input stream
 BufferedReader input = new BufferedReader(new
  FileReader("temp.txt"));
 int total = 0;
 String line;
 while((line = input.readLine()) != null){
  //extract numbers using string tokenizer
  StringTokenizer tokens = new
  StringTokenizer(line);
```

contd. to next slide

PrintWriter and PrintStream

```
while(tokens.hasMoreTokens()){
   total +=
  Integer.parseInt(tokens.nextToken());
  }//while
 }//while

 System.out.println("Total is " + total);

 input.close();
}
```

Scanner

- It can be used to scan the content of the file
- Objects of type Scanner are useful for breaking down formatted input into tokens and translating individual tokens according to their data type
- **Breaking Input into Tokens**
  - By default, a scanner uses white space to separate tokens.

Scanner

```
public static void main(String[] args)
   throws IOException {
   Scanner s = null;
      try {
         s = new Scanner(new
   BufferedReader(new
   FileReader("xanadu.txt")));
     while (s.hasNext()) {
        System.out.println(s.next());}
        } finally {
           if (s != null) {
               s.close();
               } } }
```

```
In
Xanadu
did
Kubla
Khan
...
```

Scanner

- **Translating Individual Tokens**
  - Scanner also supports tokens for all of the Java language's primitive types (except for char), as well as BigInteger and BigDecimal

Scanner

```
public static void main(String[] args)
  throws IOException {
        Scanner s = null;
        double sum = 0;

 try {
 s = new Scanner(new BufferedReader(new
  FileReader("usnumbers.txt")));
      s.useLocale(Locale.US);
```

contd. to next slide

Scanner

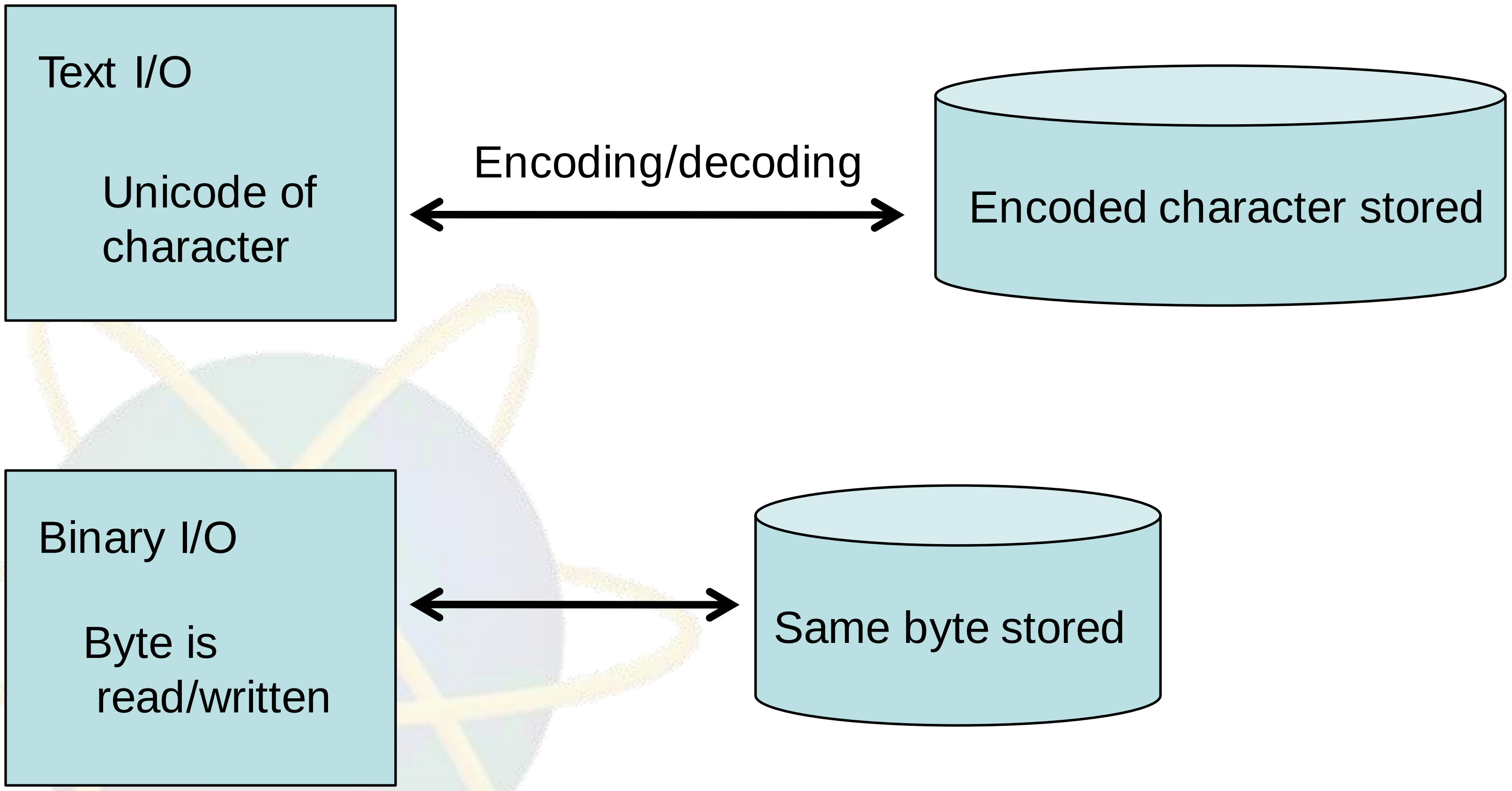
```
while (s.hasNext()) {
         if (s.hasNextDouble()) {
             sum += s.nextDouble();
         } else {
                    s.next();
                }
            }
        } finally {
            s.close();
        }
        System.out.println(sum);
    }
```

```
8.5
32,767
3.14159
1,000,000.1
```

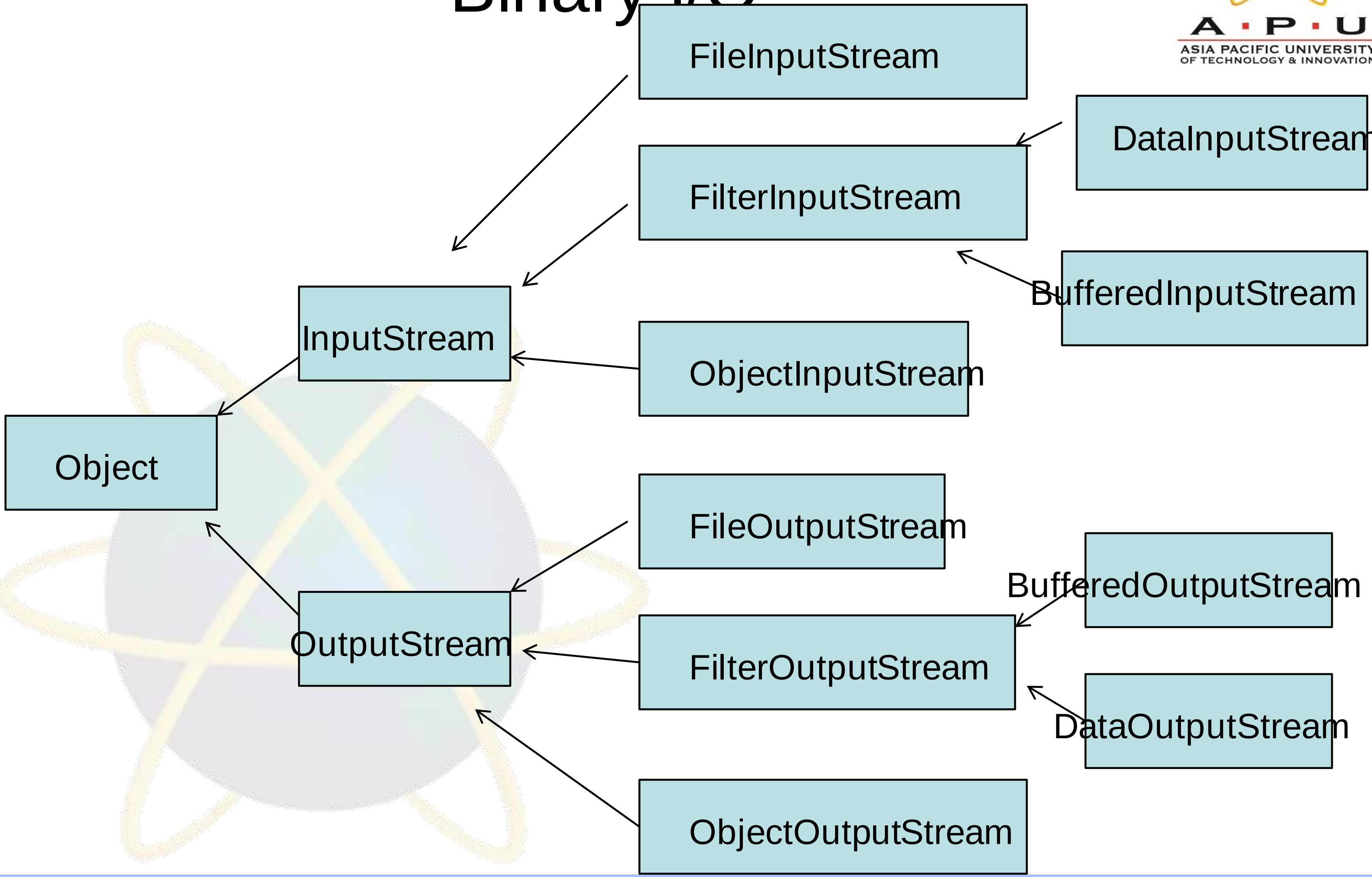
Binary I/O

- Text I/O requires encoding and decoding
- JVM does this conversion between Unicode to file-specific and vise versa
- Binary I/O does not require conversions
- Writing bytes to the file, the original bytes is copied into the file

Binary I/O


Text I/O
Unicode of character
Encoding/decoding
Encoded character stored
Binary I/O
Byte is read/written
Same byte stored

Binary I/O


FileInputStream
DataInputStream
FilterInputStream
BufferedInputStream
InputStream
ObjectInputStream
Object
FileOutputStream
BufferedOutputStream
OutputStream
FilterOutputStream
DataOutputStream
ObjectOutputStream

FileInputStream and FileOutputStream

- For reading/writing bytes from/to files.
- Constructors:

```
public FileInputStream(String
  filename)
public FileInputStream(File file)


public FileOutputStream(String
  filename)
public FileOutputStream(File file)
public FileOutputStream(String
```

FileInputStream and FileOutputStream

```
public static void main(String []args)
  throws IOException {
 //create an output stream to the file
 FileOutputStream output = new
  FileOutputStream("temp.dat");

 //output values to the file
 for(int i = 1; i <= 10; i++){
  output.write(i);
 }//for
 output.close();
```

contd. to next slide

FileInputStream and FileOutputStream

```
//create an input stream to the
 file
FileInputStream input  = new
 FileInputStream("temp.dat");


//read values from the file
int value;
while((value = input.read()) != -
 1){
  System.out.print(value + " ");
}//while
```

DataInputStream and DataOutputStream

- DataInputStream reads bytes from the stream and converts them into appropriate primitive type values or strings
- DataOutputStream converts primitive type values or strings into bytes and output the bytes to the stream
- Constructors:

```
public DataInputStream(InputStream
  instream)
public
```

DataInputStream and DataOutputStream

```
public static void main(String [] args) throws
   IOException {
  //create an output stream for the file
   temp.dat
  DataOutputStream output = new
   DataOutputStream(new
   FileOutputStream("temp.dat"));
```

contd. to next slide

DataInputStream and DataOutputStream

```
//write student test scores
output.writeUTF("John");
output.writeDouble(86.5);
output.writeUTF("Jim");
output.writeDouble(95.5);
output.writeUTF("George");
output.writeDouble(100.0);
 //close output stream
output.close();
```

contd. to next slide

DataInputStream and DataOutputStream

```
//create an input stream for file temp.dat
 DataInputStream input = new
  DataInputStream(new
  FileInputStream("temp.dat"));
  //read student test scores
 System.out.println(input.readUTF() + " " +
  input.readDouble());
 System.out.println(input.readUTF() + " " +
  input.readDouble());
 System.out.println(input.readUTF() + " " +
  input.readDouble());
 input.close(); }
```

BufferedInputStream and BufferedOutputStream

- Can be used to speed up input and output by reducing the number of reads and writes, just like BufferedReader/BufferedWriter
- BufferedReader/BufferedWriter is for reading/writing characters
- BufferedInputStream/BufferedOutputStream is for reading/writing bytes

BufferedInputStream and BufferedOutputStream

- Constructors:

```
//create a BufferedInputStream
public BufferedInputStream(InputStream
  in)
public BufferedInputStream(InputStream
  in, int bufferSize)

//create a BufferedOutputStream
public
  BufferedOutputStream(OutputStream in)
public
```

BufferedInputStream and BufferedOutputStream

- You can improve the previous program by:

```
DataOutputStream output = new
  DataOutputStream(new
  BufferedOutputStream(new
  FileOutputStream("temp.dat")));


DataInputStream input = new
  DataInputStream(new
  BufferedInputStream(new
  FileInputStream("temp.dat")));
```

Quick Review Questions

- What is Text I/O?
- What is Binary I/O?
- List all Text I/O and Binary I/O classes

Summary of Main Teaching Points

- File Class
- I/O Stream
  - Text Files & Binary Files
  - Text I/O
  - Binary I/O

Q & A

Next Session

- Introduction to JDBC
- JDBC Architecture
- Seven Steps in JDBC connection