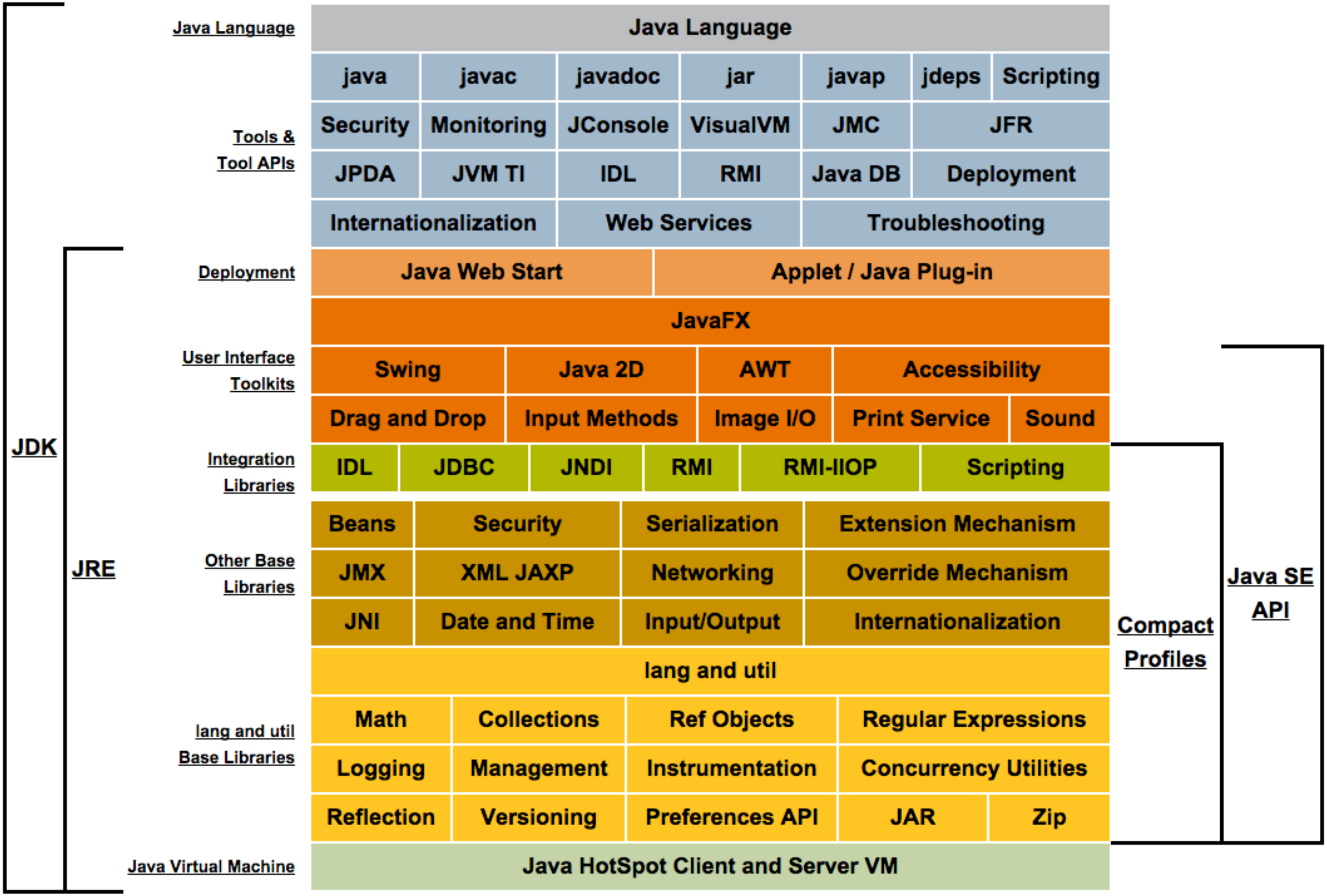
Mobile Application Development

Produced by

Eamonn de Leastar (edeleastar@wit.ie)

Waterford Institute of Technology
INSTITIÚID TEICNEOLAÍOCHTA PHORT LÁIRGE

Streams



JDK

JRE

Java Language
Java Language

Tools & Tool APIs
java | javac | javadoc | jar | javap | jdeps | Scripting
Security | Monitoring | JConsole | VisualVM | JMC | JFR
JPDA | JVM TI | IDL | RMI | Java DB | Deployment
Internationalization | Web Services | Troubleshooting

Deployment
Java Web Start | Applet / Java Plug-in

User Interface Toolkits
JavaFX
Swing | Java 2D | AWT | Accessibility
Drag and Drop | Input Methods | Image I/O | Print Service | Sound

Integration Libraries
IDL | JDBC | JNDI | RMI | RMI-IIOP | Scripting

Other Base Libraries
Beans | Security | Serialization | Extension Mechanism
JMX | XML JAXP | Networking | Override Mechanism
JNI | Date and Time | Input/Output | Internationalization

lang and util Base Libraries
lang and util
Math | Collections | Ref Objects | Regular Expressions
Logging | Management | Instrumentation | Concurrency Utilities
Reflection | Versioning | Preferences API | JAR | Zip

Java Virtual Machine
Java HotSpot Client and Server VM

Compact Profiles

Java SE API

http://www.oracle.com/technetwork/java/javase/tech/index.html

JDK vs Android SDK

Included in ADK

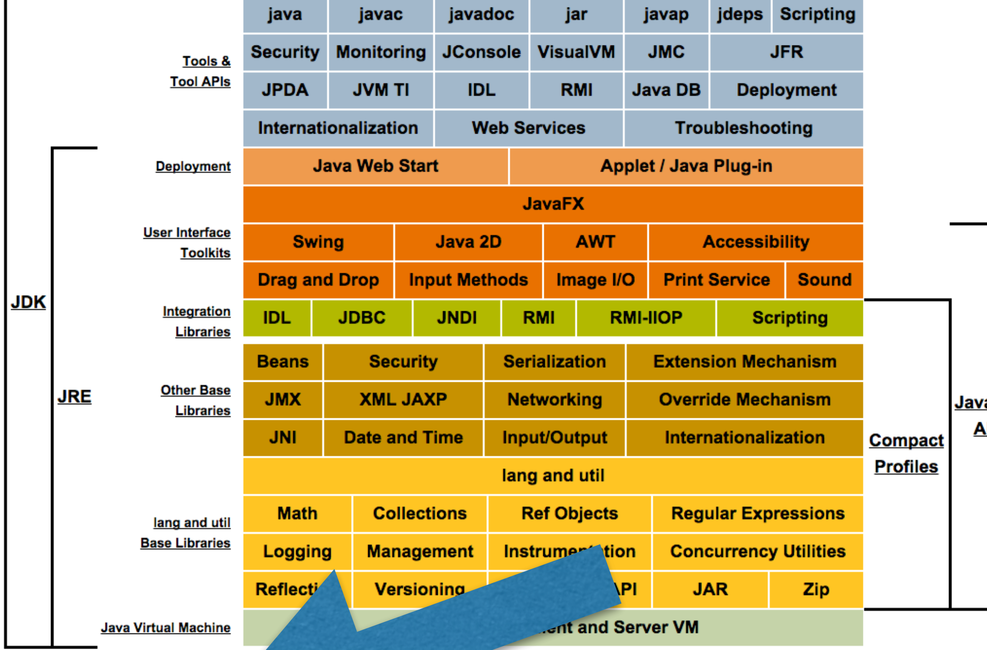
- java.io - *File and stream I/O*
- java.lang (except java.lang.management) - *Language and exceptions*
- support
- java.math - *Big numbers, rounding, precision*
- java.net - *Network I/O, URLs, sockets*
- java.nio - *File and channel I/O*
- java.sql - *Database interfaces*
- java.text - *Formatting, natural language, collation*
- java.util (*including java.util.concurrent) - Lists, maps, sets, arrays, collections*

- java.security - *Authorization, certificates, public keys*
- javax.security (except javax.security.auth.kerberos, javax.security.auth.spi, and javax.security.sasl)
- javax.sound - Music and sound effects
- javax.sql (except javax.sql.rowset) - More database interfaces
- javax.xml.parsers - XML parsing
- org.w3c.dom (but not sub-packages) - DOM nodes and elements
- org.xml.sax - Simple API for XML

JDK vs Android SDK

Excluded from ADK

- java.applet
- java.awt
- java.beans
- java.lang.management
- java.rmi
- javax.accessibility
- javax.activity
- javax.imageio
- javax.management
- javax.naming
- javax.print
- javax.rmi
- javax.security.auth.kerberos
- javax.security.auth.spi
- javax.security.sasl
- javax.swing
- javax.transaction
- javax.xml (except javax.xml.parsers)
- org.ietf.*
- org.omg.*
- org.w3c.dom.* (sub-packages)



JDK

JRE

Tools & Tool APIs
java | javac | javadoc | jar | javap | jdeps | Scripting
Security | Monitoring | JConsole | VisualVM | JMC | JFR
JPDA | JVM TI | IDL | RMI | Java DB | Deployment
Internationalization | Web Services | Troubleshooting

Deployment
Java Web Start | Applet / Java Plug-in

User Interface Toolkits
JavaFX
Swing | Java 2D | AWT | Accessibility
Drag and Drop | Input Methods | Image I/O | Print Service | Sound

Integration Libraries
IDL | JDBC | JNDI | RMI | RMI-IIOP | Scripting

Other Base Libraries
Beans | Security | Serialization | Extension Mechanism
JMX | XML JAXP | Networking | Override Mechanism
JNI | Date and Time | Input/Output | Internationalization

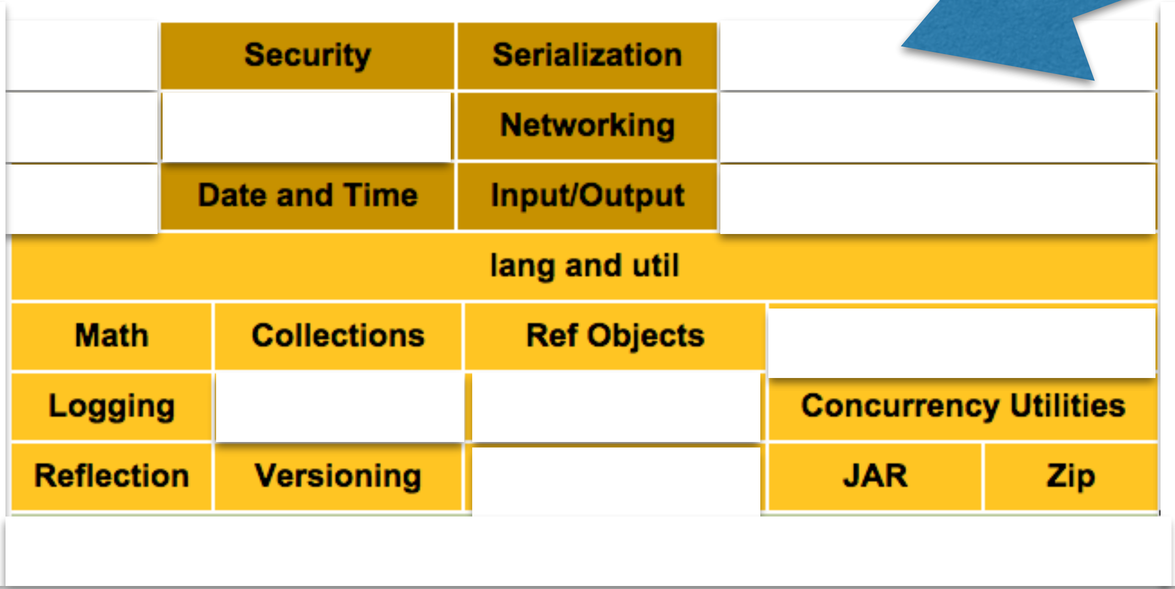
lang and util Base Libraries
lang and util
Math | Collections | Ref Objects | Regular Expressions
Logging | Management | Instrume tion | Concurrency Utilities
Reflecti | Versioning | PI | JAR | Zip

Java Virtual Machine
ent and Server VM

Compact Profiles

Java
A

ADK

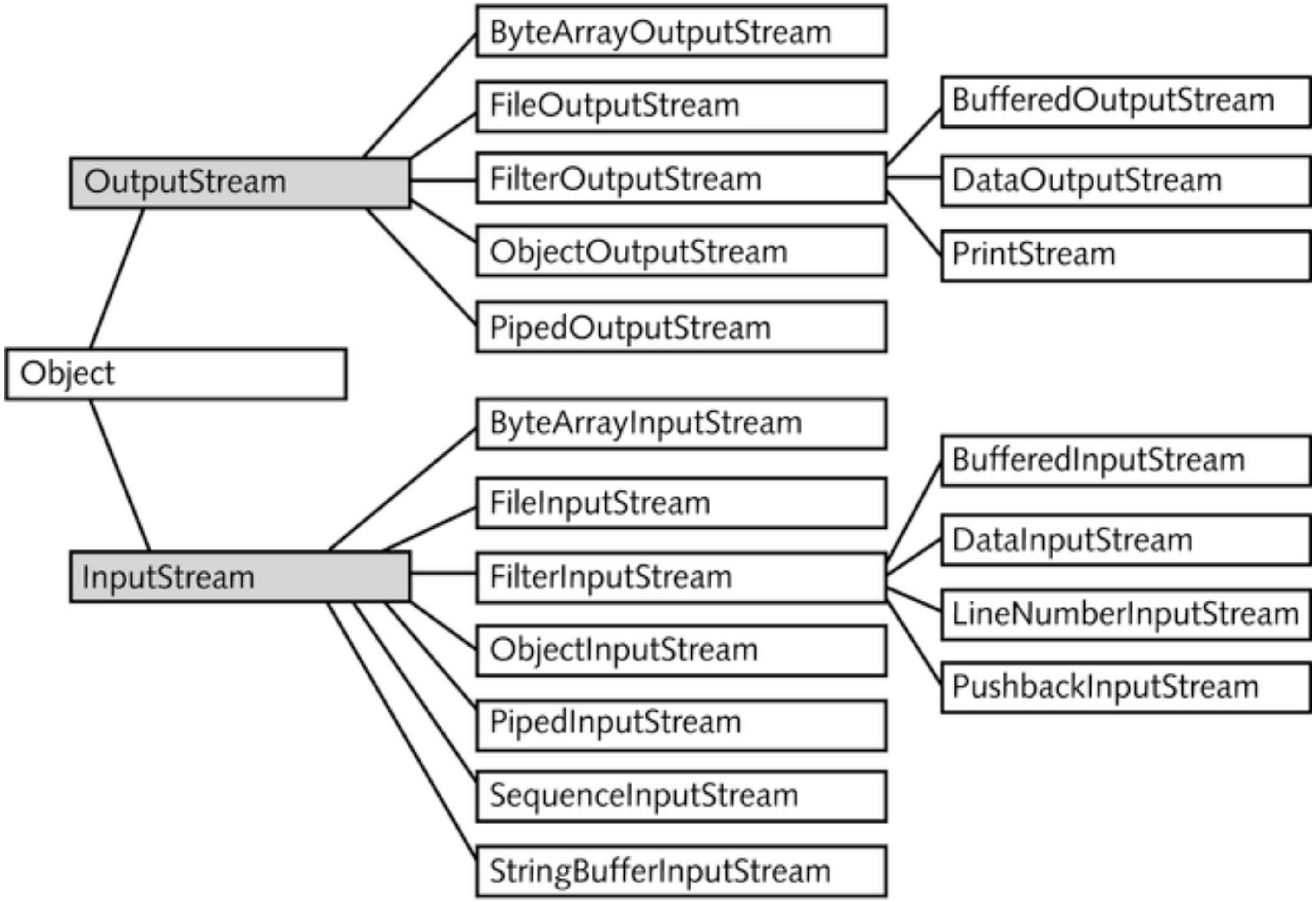


Security | Serialization
Networking
Date and Time | Input/Output
lang and util
Math | Collections | Ref Objects
Logging | Concurrency Utilities
Reflection | Versioning | JAR | Zip

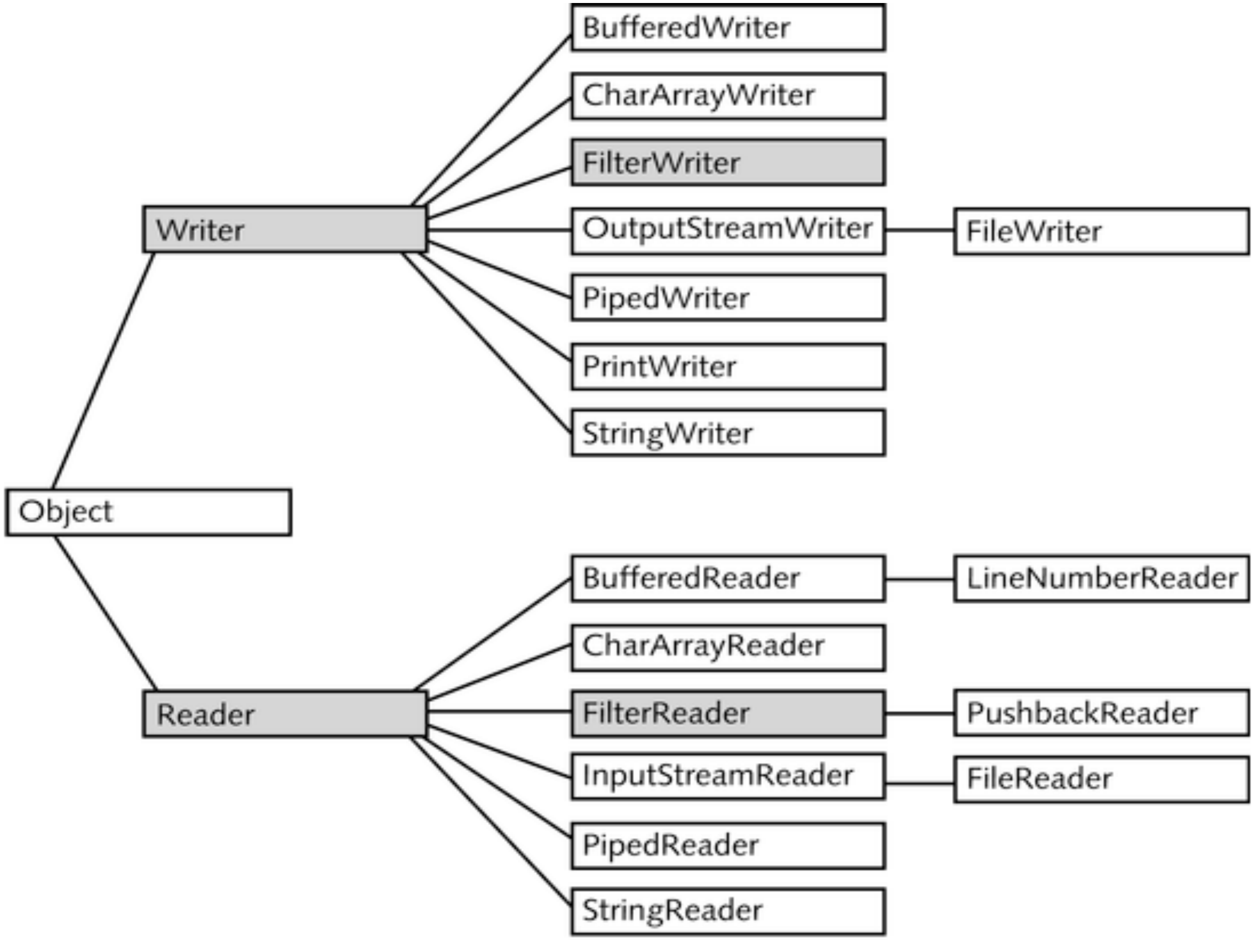
Streams

- An I/O Stream represents an input source or an output destination.
- A stream can represent
  - disk files
  - devices
  - other programs
- Streams support
  - simple bytes
  - primitive data types
  - localized characters
  - objects.
- Some streams simply pass on data, others manipulate and transform the data in useful ways.

Byte-Oriented Streams



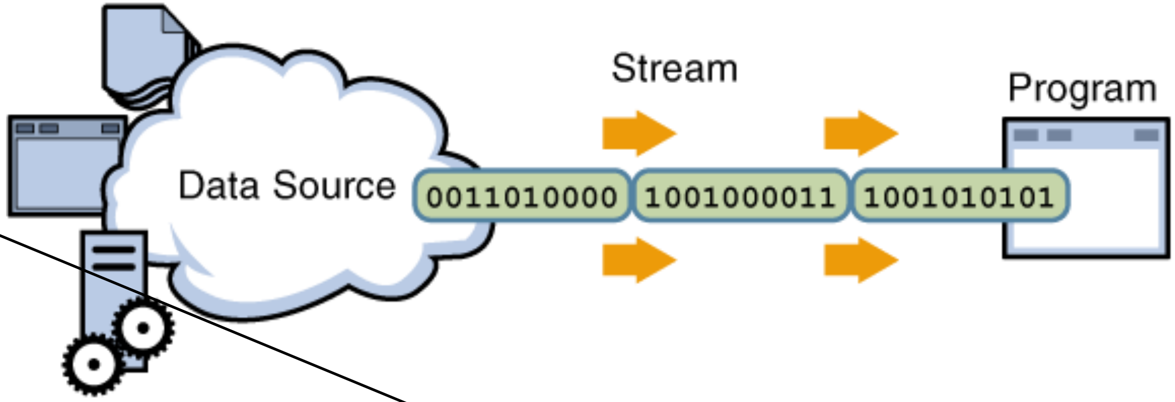
Text Oriented Streams



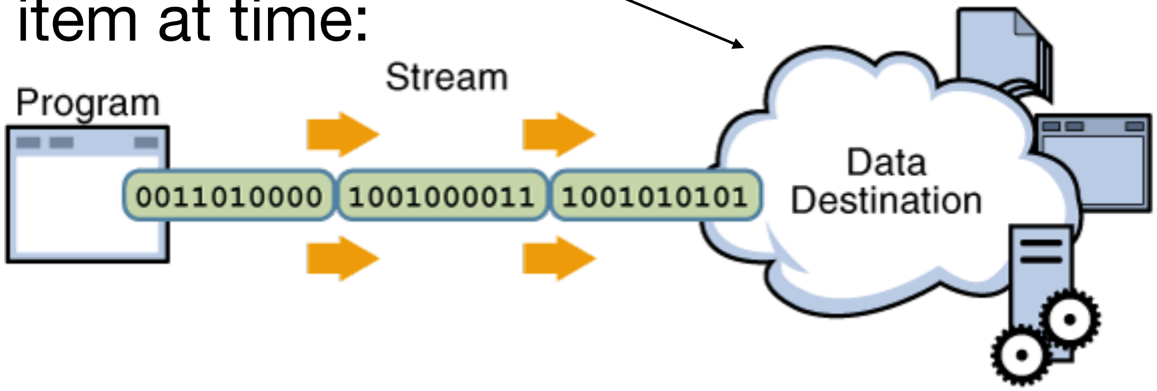
Input/Output Streams

- A stream is a sequence of data.
- A Java program uses an input stream to read data from a source, one item at a time:

disk files
program
device
socket
array

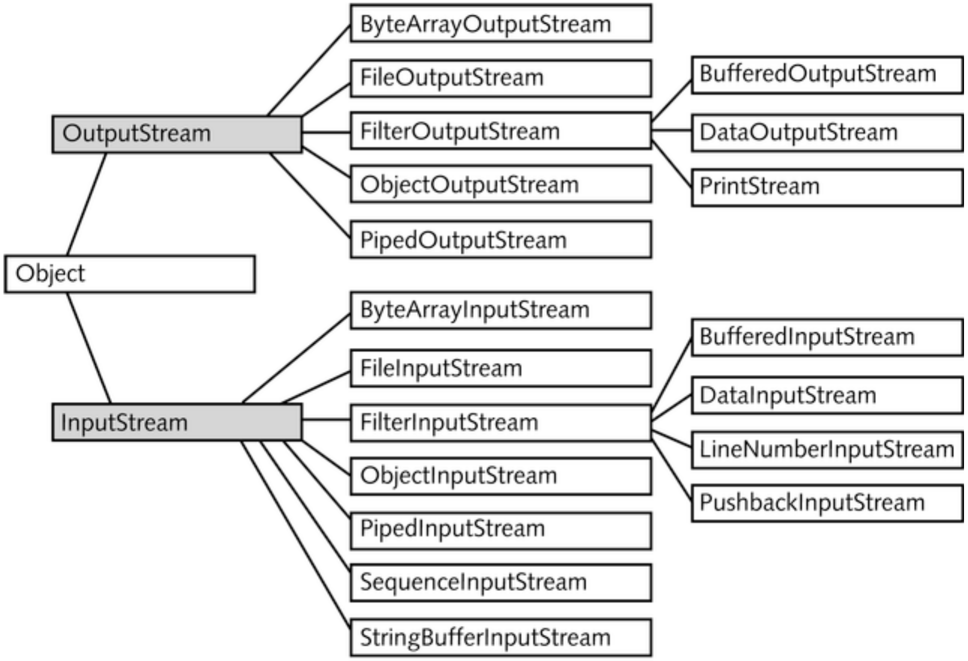


- A Java program uses an output stream to write data to a destination, one item at time:



Byte Streams

- Byte streams perform I/O of 8-bit bytes.
- All byte stream classes are descended from InputStream & OutputStream.
- To read/write from files, use FileInputStream and FileOutputStream.
- Other kinds of byte streams are used much the same way; they differ mainly in the way they are constructed.

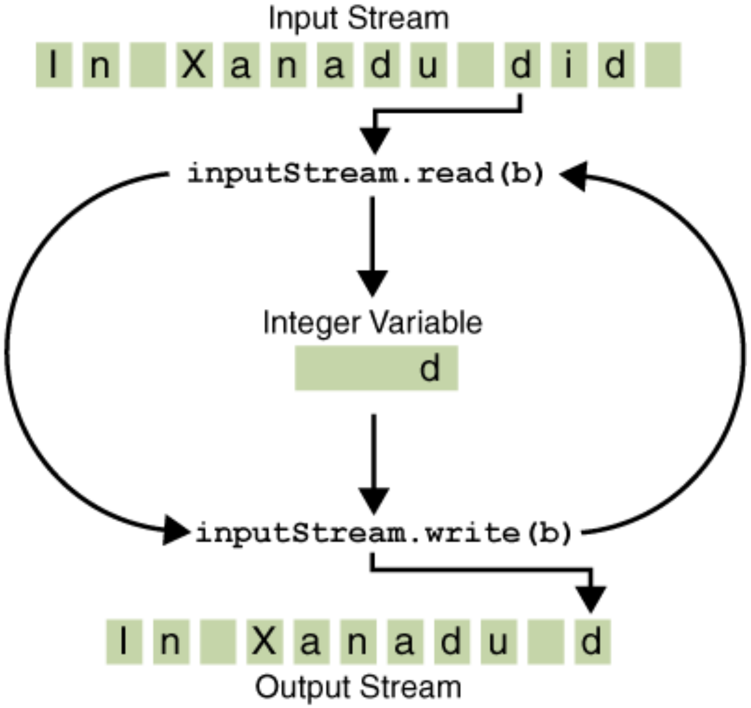


CopyBytes

```
public class CopyBytes
{
  public static void main(String[] args) throws IOException
  {
    FileInputStream in = null;
    FileOutputStream out = null;
    try
    {
      in = new FileInputStream("input.txt");
      out = new FileOutputStream("final.txt");
      int c;
      while ((c = in.read()) != -1)
      {
        out.write(c);
      }
    }
    finally
    {
      if (in != null)
      {
        in.close();
      }
      if (out != null)
      {
        out.close();
      }
    }
  }
}
```

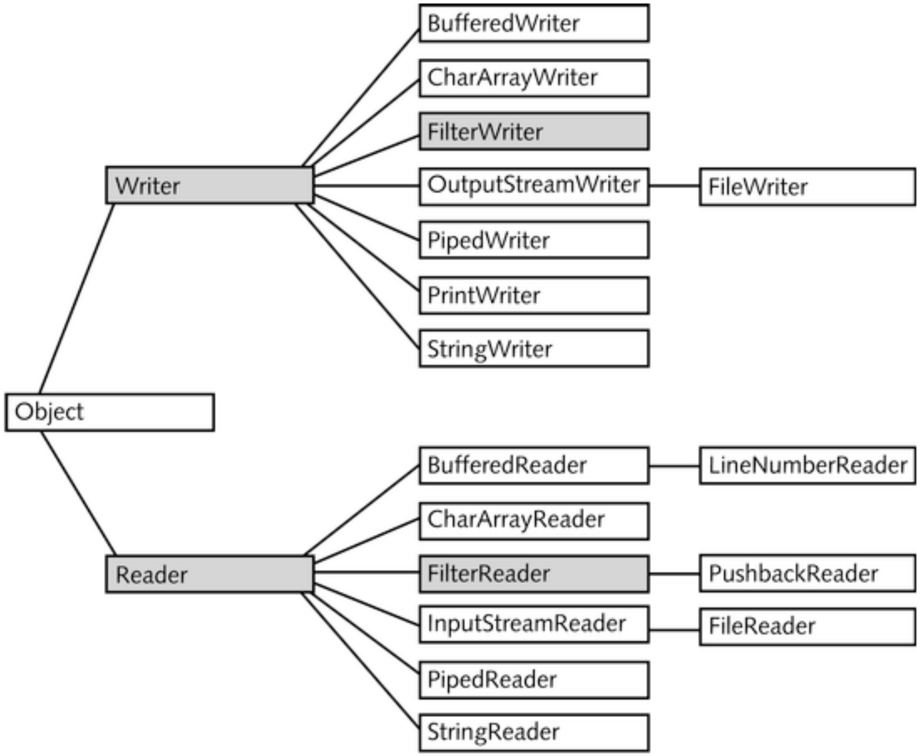
CopyBytes

- An int return type allows read() to use -1 to indicate end of stream.
- CopyBytes uses a finally block to guarantee that both streams will be closed even if an error occurs. this helps avoid resource leaks.
- If CopyBytes was unable to open one or both files the stream variable never changes from its initial null value.
- Byte streams should only be used for the most primitive I/O.
- However, all other stream types are built on byte streams.



Character Streams

- Java stores character values using Unicode
- Character stream I/O automatically translates this to and from the local character set.
- In Western locales, the local character set is usually an 8-bit superset of ASCII.
- I/O with character stream classes automatically translates to/from the local character set.



CopyCharacters

```
public class CopyCharacters
{
  public static void main(String[] args) throws IOException
  {
    FileReader inputStream = null;
    FileWriter outputStream = null;
    try
    {
      inputStream = new FileReader("input.txt");
      outputStream = new FileWriter("final.txt");
      int c;
      while ((c = inputStream.read()) != -1)
      {
        outputStream.write(c);
      }
    }
    finally
    {
      if (inputStream != null)
      {
        inputStream.close();
      }
      if (outputStream != null)
      {
        outputStream.close();
      }
    }
  }
}
```

CopyCharacters vs CopyBytes

- CopyCharacters is very similar to CopyBytes.
  - CopyCharacters uses FileReader and FileWriter
  - CopyBytes uses FileInputStream and FileOutputStream.
- Both use an int variable to read to and write from.
  - CopyCharacters int variable holds a character value in its last 16 bits
  - CopyBytes int variable holds a byte value in its last 8 bits
- Character streams are often "wrappers" for byte streams.
  - A byte stream to perform the physical I/O
  - The character stream handles translation between characters and bytes.
- E.g. FileReader uses FileInputStream, while FileWriter uses FileOutputStream.

Buffered IO

- So far we have used unbuffered I/O:
  - Each read or write request is handled directly by the underlying OS.
  - Can be less efficient, since each such request often triggers disk or network access.
- To reduce this kind of overhead use buffered I/O streams.
  - Read data from a memory area known as a buffer
  - Native input API is called only when the buffer is empty.
  - Buffered output streams write data to a buffer
  - Native output API is called only when the buffer is full.

Line-Oriented IO

- Character I/O usually occurs in bigger units than single characters.
- One common unit is the line:
  - a string of characters with a line terminator at the end.
- A line terminator can be
  - a carriage-return/line-feed sequence ("\r\n")
  - a single carriage-return ("\r"), or a single line-feed ("\n").
- Supporting all possible line terminators allows programs to read text files created on any of the widely used operating systems.

CopyLines

```
public class CopyLines
{
  public static void main(String[] args) throws IOException
  {
    BufferedReader inputStream = null;
    PrintWriter outputStream = null;
    try
    {
      inputStream = new BufferedReader(new FileReader("xanadu.txt"));
      outputStream = new PrintWriter(new FileWriter("characteroutput.txt"));
      String l;
      while ((l = inputStream.readLine()) != null)
      {
        outputStream.println(l);
      }
    }
    finally
    {
      if (inputStream != null)
      {
        inputStream.close();
      }

      if (outputStream != null)
      {
        outputStream.close();
      }
    }
  }
}
```

BufferedReader

- An unbuffered stream can be converted into a buffered stream using the wrapper idiom:
- The unbuffered stream object is passed to the constructor for a buffered stream class.

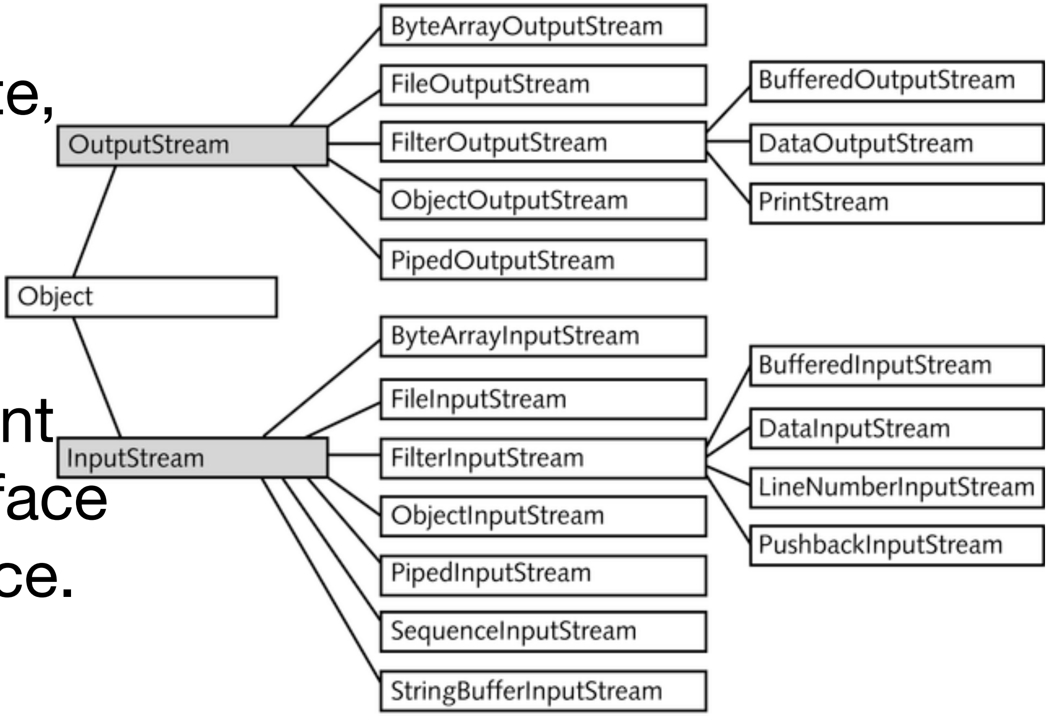
```
try
 {
   inputStream = new BufferedReader(new FileReader("input.txt"));
   outputStream = new PrintWriter(
                       new BufferedWriter(
                         new FileWriter("characteroutput.txt")));

   String l;

   while ((l = inputStream.readLine()) != null)
   {
     outputStream.println(l);
   }
 }
```

Data Streams

- Data streams support binary I/O of primitive data type values (boolean, char, byte, short, int, long, float, and double) as well as String values.
- All data streams implement either the DataInput interface or the DataOutput interface.
- The most widely-used implementations of these interfaces are DataInputStream and DataOutputStream.



DataStream (1)

```
public class DataStream
{
  static final String dataFile = "invoicedata";
  static final double[] prices = { 19.99, 9.99, 15.99, 3.99, 4.99 };
  static final int[] units     = { 12, 8, 13, 29, 50 };
  static final String[] descs = { "Java T-shirt", "Java Mug",
                                  "Duke Juggling Dolls",
                                  "Java Pin", "Java Key Chain"};

  public static void main(String[] args) throws IOException
  {
    DataOutputStream out = new DataOutputStream(
             new BufferedOutputStream(new FileOutputStream(dataFile)));

    for (int i = 0; i < prices.length; i++)
    {
      out.writeDouble(prices[i]);
      out.writeInt(units[i]);
      out.writeUTF(descs[i]);
    }
    out.close();

    //…continued
```

DataStream (2)

```
    DataInputStream in = new DataInputStream(
                           new BufferedInputStream(
                             new FileInputStream(dataFile)));
    double price;
    int unit;
    String desc;
    double total = 0.0;
    try
    {
      while (true)
      {
        price = in.readDouble();
        unit = in.readInt();
        desc = in.readUTF();
        System.out.format("You ordered %d units of %s at $%.2f%n",
                                                          unit, desc, price);
        total += unit * price;
      }
    }
    catch (EOFException e)
    {
      System.out.println("End of file");
    }
  }
}
```

Data Streams Observations

- The writeUTF method writes out String values in a modified form of UTF-8.
  - A variable-width character encoding that only needs a single byte for common Western characters.
- Generally, we detects an end-of-file condition by catching EOFException, instead of testing for an invalid return value.
- Each specialized write in DataStreams is exactly matched by the corresponding specialized read.
- Floating point numbers not recommended for monetary values
  - In general, floating point is bad for precise values.
  - The correct type to use for currency values is java.math.BigDecimal.
- Unfortunately, BigDecimal is an object type, so it won't work with data streams – need Object Streams.

Waterford Institute *of* Technology
INSTITIÚID TEICNEOLAÍOCHTA PHORT LÁIRGE

eLearning
support unit