Java - Input/Output

CS 4354
Fall 2012

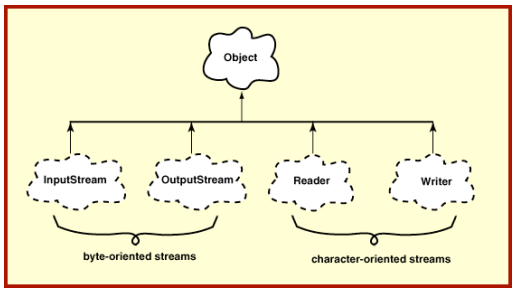
Jill Seaman

Streams

- Java I/O is defined in terms of streams
- A stream in Java is an ordered sequence of data that has a source or a destination.
- A data source is generally known as an input stream
- A data destination is generally known as an output stream or sink

Streams

- Java supports two major types of I/O streams:
- Byte streams (8-bit bytes):
  - ✦These classes are subclasses of InputStream and OutputStream.
- Character streams (16-bit Unicode characters):
  - ✦These classes are subclasses of Reader and Writer



Standard I/O streams

- System.in - an InputStream used to read bytes from the keyboard
  - ✦java.lang.Object
    -->java.io.InputStream
- System.out - a PrintStream used to display bytes to the screen.
  - ✦java.lang.Object
    -->java.io.OutputStream
      -->java.io.FilterOutputStream
        -->java.io.PrintStream
- System.err - another PrintStream used to display error messages in the byte format to the screen

Major character stream input classes (Readers)

- BufferedReader buffers character input stream to improve character input efficiency.
  - ✦LineNumberReader supports line number identification.
- CharArrayReader reads characters from a character array.
- FilterReader an abstract class provides generic input filtering functions.
  - ✦PushbackReader.
- InputStreamReader reads characters from a byte stream.
  - ✦FileReader reads characters from a file.
- PipedReader reads characters from a pipe.
- StringReader reads characters from a string.

Major character stream output classes (Writers)

- BufferedWriter buffers character output stream to improve output efficiency.
- CharArrayWriter writes characters into a character array.
- FilterWriter an abstract class provides generic output filtering functions.
- OutputStreamWriter writes characters into a byte stream.
  - ✦FileWriter writes characters to a file.
- PipedWriter writes characters to a pipe.
- PrintWriter writes textual representations of primitive values and objects.
- StringWriter writes characters into a String.

Reading from the keyboard

- imports
  - ✦java.io.* for the Reader classes
  - ✦java.util.* for the StringTokenizer
- readLine: reads a line of text from the Reader (up to newline)

```
import java.io.*;
import java.util.*;

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

String str = br.readLine();
StringTokenizer st = new StringTokenizer(str);
```


Reading from the keyboard: exceptions

- I/O exceptions
  - ✦readLine() throws an IOException when it fails
  - ✦It must be wrapped in an exception handler (try/catch) that catches the exception.

```
import java.io.*;
import java.util.*;

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

try {
    String str = br.readLine();
  } catch (IOException e) {
    //do something here to process e
  }
```

Reading from the keyboard: parsing

- The Reader reads only strings or single characters
  - ✦programmer must parse the string to find any other kind of data
  - ✦the parsing functions we will be using do not allow any leading or trailing whitespace.
  - ✦StringTokenizer provides a facility for skipping whitespace in a given string
  - ✦StringTokenizer is initialized with a string.
  - ✦The nextToken() method skips whitespace and returns a string containing the next sequence of non-whitespace characters.

```
StringTokenizer st = new StringTokenizer(str);
String data = st.nextToken();
```


Reading from the keyboard: parsing

- These methods parse values from a string

boolean	boolean b = new boolean(data).booleanValue;
byte	byte by = Byte.parseByte(data);
char	int ic = isr.read(); char c = (char)ic;
double	double d = new Double(data).doubleValue();
float	float f = new Float(data).floatValue();
int	int i = Integer.parseInt(data);
long	long l = Long.parseLong(data);
String	just use str (from br.readLine())

Reading from the keyboard: EasyIn

- Java class with helper methods to input each of the primitive data types, using the method shown earlier.
  - ✦Header comment below, an excerpt from the class on next slide
  - ✦Complete class is available on TRACS under resources
  - ✦Put in same directory as your code (compile), add `import easyInPackage;`

```
package easyInPackage
// Copyright (c) Peter van der Linden,  May 5 1997.
// Feel free to use this in your programs, as long as this
// comment stays intact.
//
// This is not thread safe, not high performance, and doesn't tell EOF.
// It's intended for low-volume easy keyboard input.
// An example of use is:
//     int i = EasyIn.readInt();   // reads an int from System.in
```


Reading from the keyboard: EasyIn

```
class EasyIn {
    static InputStreamReader is = new InputStreamReader( System.in );
    static BufferedReader br = new BufferedReader( is );
    private static StringTokenizer st;

    private static StringTokenizer getToken() throws IOException {
       String s = br.readLine();
       return new StringTokenizer(s);
    }

    public static int readInt() {
       try {
         st = getToken();
         return Integer.parseInt(st.nextToken());
       } catch (IOException ioe) {
          System.err.println("IO Exception in EasyIn.readInt");
          return 0;
       }
    }
}
```

Writing to the screen

- System.out
  - ✦System.out is a PrintStream, used to print characters.
  - ✦A PrintStream adds functionality to another output stream, namely the ability to print **representations of various data values** conveniently.
  - ✦All characters printed by a PrintStream are converted into bytes using the platform's default character encoding.
- println(x) (also print(x))
  - ✦Methods of PrintStream (see API website for details)
  - ✦Overloaded to print various data types.
  - ✦Often uses the default toString() method of the wrapper classes.
  - ✦In other words, it's not custom formatted.

Writing to the screen: Formatting

- DecimalFormat class
  - ✦DecimalFormat is a concrete subclass of NumberFormat that formats decimal numbers.
  - ✦`DecimalFormat(String pattern)` Creates a DecimalFormat using the given pattern and the symbols for the default locale.
  - ✦format(x) is a method that formats an item (x) to produce a string.

0	digit (left-padded with zeros)
#	digit, zero shows as absent (no 0 padding)
.	decimal separator
,	Grouping separator
;	Separates positive and negative subpatterns
–	minus sign
%	Multiply by 100, show as percent

Formatting example

```
import java.io.*;
import java.text.*;

class FormatOut {
     public static void main(String args[])  {
        int []  iArray = {1, 12, 123};
        float []  fArray = {1.1F, 10.12F, 100.123F};
        double []  dArray = {1.1, 10.12, 100.1234, 1000.1239};

        DecimalFormat dfi = new DecimalFormat("#00");
        DecimalFormat dff = new DecimalFormat("#00.00 float");
        DecimalFormat dfd = new DecimalFormat("#000.000");

        for (int i = 0; i < iArray.length; i++)
            System.out.println(dfi.format(iArray[i]));

        for (int i = 0; i < fArray.length; i++)
            System.out.println(dff.format(fArray[i]));

        for (int i = 0; i < dArray.length; i++)
            System.out.println(dfd.format(dArray[i]));
    }
}  //  FormatOut
```


Formatting example

- Output from the example:

```
01
12
123
01.10 float
10.12 float
100.12 float
001.100
010.120
100.123
1000.124
```

Object serialization

- A process of transforming an object into a stream of bytes.
- The object must implement the `Serializable` interface.
  - ✦If not, you get an exception: java.io.NotSerializableException: theClass

```
public class Circle implements Serializable {  ...
```

  - ✦Note: there are no required methods to override
- Object serialization allows you to implement persistence.
- Persistence: when an object's lifetime is not determined by whether a program is executing; the object lives in between invocations of the program.

Object serialization: streams

- Java provides two object streams for serialization.
- `ObjectOutputStream`
  - ✦Provides methods for writing object graphs into a byte stream.
  - ✦An object graph refers to all bytes representing an object, including any other objects this object references.
  - ✦The `writeObj()` method writes an object graph to an output stream, given the root of the graph.
- `ObjectInputStream`
  - ✦It provides methods for reading object graphs from a byte stream.
  - ✦The `readObj()` method reads an object graph from an input stream.
  - ✦You must cast the result to the correct object (at the root of the graph).

Serialization example: ObjFIO.java

```
import java.io.*;

class ZStudent implements Serializable {
    int no;
    String first, mid, last;
    float ave;

    ZStudent() {};   // default constructor
    ZStudent(int no, String first, String mid, String last, float ave) {
        this.no = no;
        this.first = first;
        this.mid = mid;
        this.last = last;
        this.ave = ave;
    }

    // to stringize a record; don't want to override toString()
    public String display() {
        return (no + " " + first + " " + mid + " " + last + " " + ave);
    }
}
```


Serialization example: ObjFIO.java cont.

```
class ObjFIO {
    public static void main(String[] args) {
        ZStudent[] zstudents = {
            new ZStudent(50,  "Blue ", "M", "Monday   ", 50.0F),
            new ZStudent(100, "Gray ", "G", "Tuesday  ", 60.0F),
            new ZStudent(150, "Green", "G", "Wednesday", 70.0F),
            new ZStudent(200, "Pink ", "P", "Thursday ", 80.0F),
            new ZStudent(300, "Red  ", "R", "Friday   ", 90.0F)};

        try {
            FileOutputStream fos = new FileOutputStream("zStudentFile");
            ObjectOutputStream oos = new ObjectOutputStream(fos);

            // to write out student records to a file
            for (int i = 0; i < 5; i++) {
                oos.writeObject(zstudents[i]);// to write 1 obj at a time
            }
            fos.close();
```

Serialization example: ObjFIO.java cont.

```
            FileInputStream fis = new FileInputStream("zStudentFile");
            ObjectInputStream ois = new ObjectInputStream(fis);
            ZStudent sbuf;

            // to read out student records from a file
            for (int i = 0; i < 5; i++) {
                sbuf = (ZStudent)ois.readObject();  // explicit cast reqd
                System.out.println(sbuf.display());
            }
            fis.close();
        } catch (IOException ioe) {
            System.out.println("Error: " + ioe);
        } catch (ClassNotFoundException cnfe) {
            System.out.println("Error: " + cnfe);
        }
    }
}
```


Serialization example

- Output from the example:

```
50 Blue  M Monday    50.0
100 Gray  G Tuesday   60.0
150 Green G Wednesday 70.0
200 Pink  P Thursday  80.0
300 Red   R Friday    90.0
```

- Note: Arrays are objects, and may be serialized as a whole:

```
oos.writeObject(zstudents);
```

```
ZStudent [] newStudents = (ZStudent[])ois.readObject();
for (int i=0; i<5; i++) {
    System.out.println(newStudents[i].display());
}
```


General File I/O (characters)

- FileReader - a Reader to read character files
  - ✦java.lang.Object
    -->java.io.Reader
      -->java.io.InputStreamReader
        -->java.io.FileReader

```
FileReader fr = new FileReader("dataFile");
int inChar = fr.read();
if (inChar != -1) System.out.print((char)inChar + " ");
```

- FileWriter - a Writer to write characters to a file
  - ✦java.lang.Object
    -->java.io.Writer
      -->java.io.OutputStreamWriter
        -->java.io.FileWriter

```
FileWriter  fw = new FileWriter("dataFile");
fw.write('x');
```