1.00/1.001/1.002
Introduction to Computers and Engineering Problem Solving

Recitation 9
Stream and Phidget

April 23-24, 2012

Outline

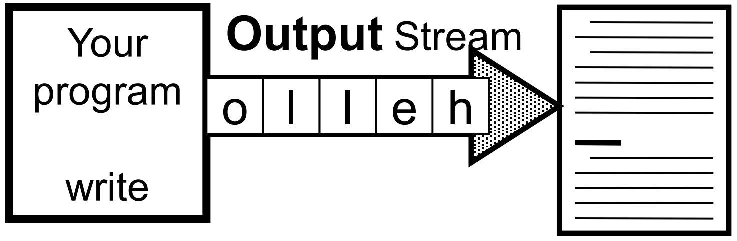
- Streams

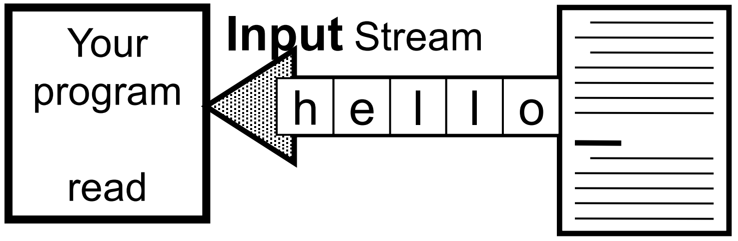
- Phidgets

Streams Overview

- Java programs communicate with the outside world using streams
- I/O streams: work in one direction only
  - Input stream: control data coming into the program
  - Output stream: control data leaving the program
- Streams: **FIFO** queues
- Streams have many uses:
  - Music and videos "stream" from online providers
  - This recitation focuses on: writing to and reading from a text file

Notice: Streams are labeled with respect to your program:



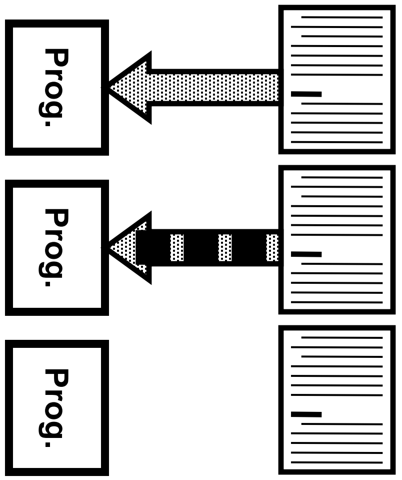


write = „push" data into a file

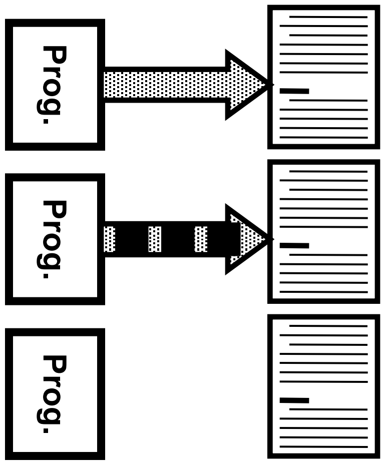
read = „pull" data from a file

General Strategy for reading from and writing to a Stream

- Reading
  - Open an **INPUT** stream
  - while more information in the stream
    - read information
  - close the stream



- Writing
  - Open an **OUTPUT** stream
  - while more information in the program
    - write information
  - close the stream



Connecting Streams

- Each stream class has a specific functionality.
- Streams can be connected to get more functionality

- Example: **`class BufferedReader`**
  - Buffers the character stream from `FileReader` for efficiency
  - allows you to read line by line

```
FileReader fileInput = new
 FileReader("file.txt");
BufferedReader bufferedInput = new
 BufferedReader(fileInput);
```

Example: Reading a Text File

```
try{
    // Step1: Create class that gets the File
    FileReader fr = new 
FileReader(“file.txt”);
    // Step2: Create class to read data from File
    BufferedReader br = new 
BufferedReader(fr);
    // Step3: Read the data
    String line = br.readLine();
    while(line!=null){
        System.out.println(line);
        line = br.readLine();
    }
    // Step4: Close the File! (so others can use it)
    br.close();
} catch (IOException ioe) {
    ioe.printStackTrace();
}
```

Example: Writing a Text File

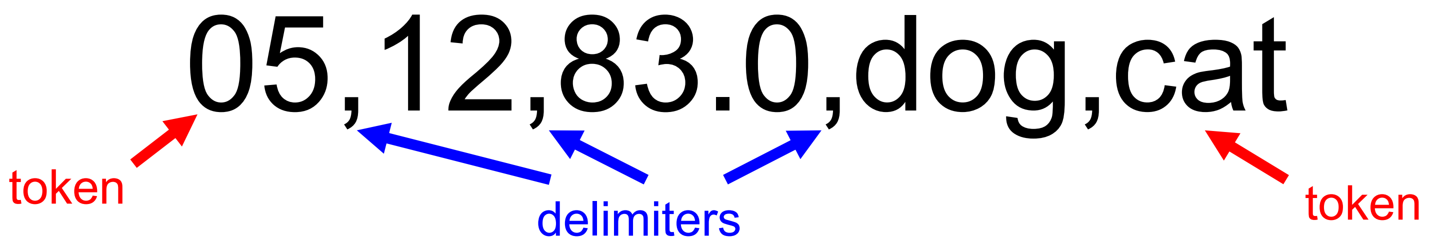
```
try{
    // Step1: Create class that gets the File
    FileWriter fw = new
                    FileWriter("file1.txt");
    // Step2: Create class to write data into File
    PrintWriter pw = new  PrintWriter(fw);
    // Step3: Write the data
    pw.println("How are you doing?");
    // Step4: Close the File! (so others can use it)
    pw.close();
} catch (IOException ioe) {
    ioe.printStackTrace();
}
```

File Exercise

- Write a program that copies, line by line, a text file file.txt to fileCopy.txt

Text Files & Delimiters

A **delimiter** is a character used to separate information (a.k.a. **tokens**).



There are two common delimiters for text files: **commas** and **tabs**

File w/ Comma Separated Values

data.csv

```
05,12,83
```

File w/ Tab Separated Values

data.tsv

```
05	12	83
```

Writing: Why use a delimiter?

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

```
try{
      FileWriter fw = new FileWriter(“data.csv”);
      PrintWriter pw = new PrintWriter(fw);
      pw.print(87);
      pw.print(56);
      pw.close();
} catch (IOException ioe) {
      ioe.printStackTrace();
}
```

data.csv

```
87 56
```

PROBLEM!
What are the two numbers?
Are they 875 and 6? 8 and 756?

```
try{
      FileWriter fw = new FileWriter(“data.csv”);
      PrintWriter pw = new PrintWriter(fw);
      pw.print(87+”,”);
      pw.print(56+”,”);
      pw.close();
} catch (IOException ioe) {
      ioe.printStackTrace();
}
```

data.csv

```
87, 56,
```

SOLUTION!
We can use a delimiter to separate the values

Reading: Skipping delimiters

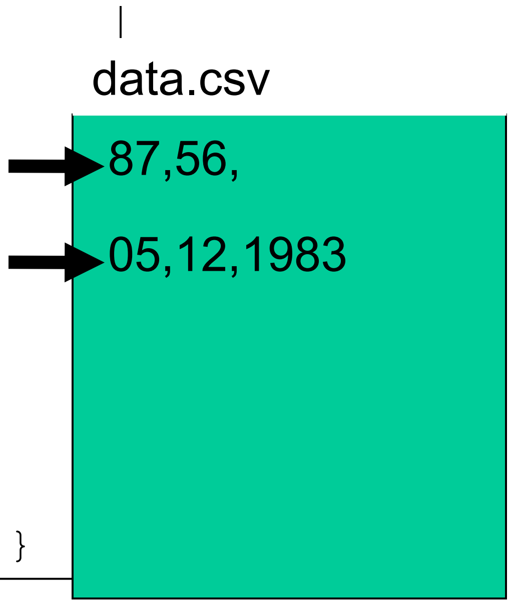
Reading the data now becomes tricky! We need to skip the delimiters.

Fortunately, the `String` class has a method called split()

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

try{
FileReader fr = new FileReader("data.csv");
BufferedReader br = new BufferedReader(fr);
String line;
 while((line = br.readLine()      ) != null){
    // Use String's split() method
    String[ ] svalues = line.split(",");
        // loop over and parse each String
    for(String svalue: svalues){
        int value = Integer.parseInt(svalue);
     }
    }
      br.close();
} catch (IOException ioe) { ioe.printStackTrace(); }
```

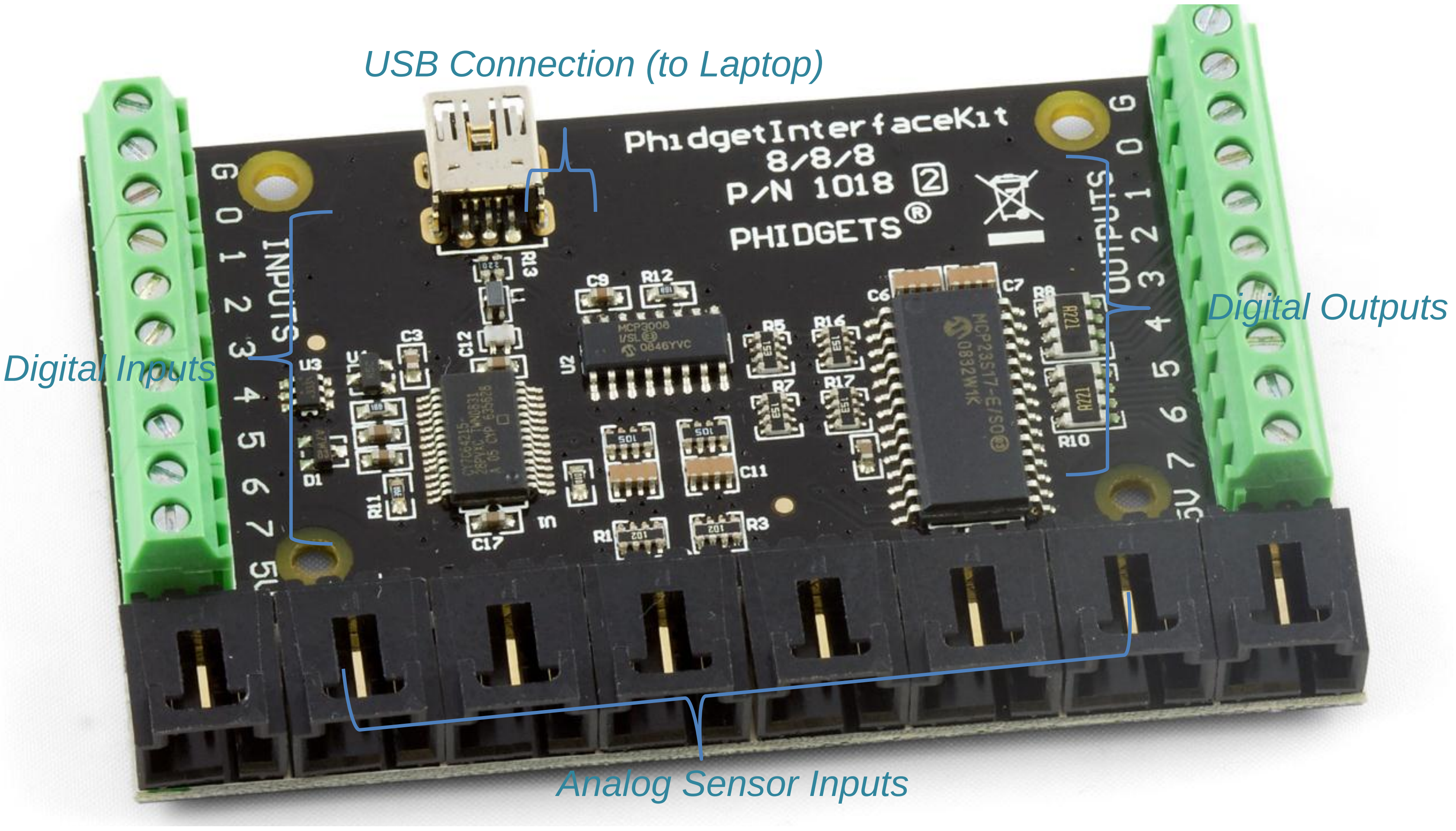
- Get a String with delimited data: "87,56,"
- Call the split() method, telling it to divide the String when it sees ","



Outline

- Streams
- Phidgets

Phidget Interface Anatomy



Characteristics

- Phidgets provide the hardware interface to attach sensors to your computer
- The com.phidget.* class files provide the software needed to connect your Java program to the Phidget hardware
- Sensor input is noisy – do not rely too much on the exact values
- Sensor change events will arrive unpredictably – do not rely on them to indicate that time has passed
- Use try/catch blocks around all phidget code that uses your InterfaceKitPhidget object

Phidget Sensors

- The Phidget board does not determine which sensor is attached, it only knows which port the sensor is connected to
- If you plug your sensors into the wrong ports, your program won't identify them correctly
- You must have Phidget kit software closed when running Eclipse
- You must close all previous application window when developing code for problem sets (e.g. you cannot have more than one Swing application running)

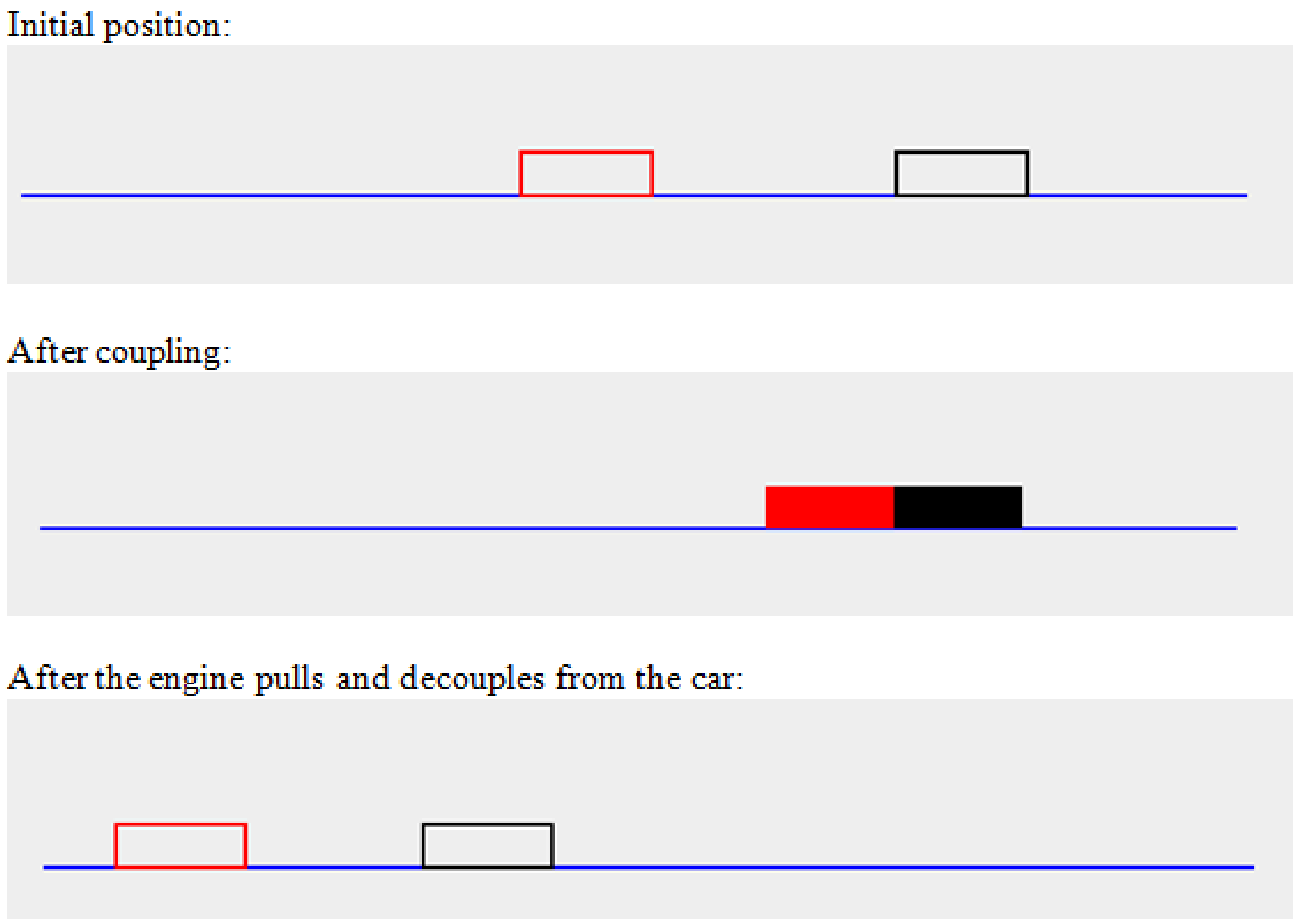
Example: Light Sensor Lamp

Street lamps turn on when it gets dark outside!

Model using phidget

- Use precision light sensor
- Threshold: 200
- Attach LED to digital output 0
- When sensor value is below threshold turn LED on, otherwise keep it off
- Default value off

Problem Set 8



- Use sensors to couple and uncouple engine and a car
- Write event log to a text file

Problem Set 8

- You will use slider sensor to control the movement of engine and car, in either direction
- You need to keep moving slider to keep generating events so the engine keeps moving
- Use a pressure sensor to couple or uncouple engine and car
- A light sensor is used for emergency stop. Cover it while testing(Train moves only when it's darker)

Start Early!