Object Oriented Programming

Week 8 Part 1
Input and Output

Lecture

- Overview of I/O
- Monitor Output
- Keyboard Input

I/O Overview

Input and Output

- Our programs get input and output from a number of sources.
- To simplify, input and output are handled by streams.

Streams Defined

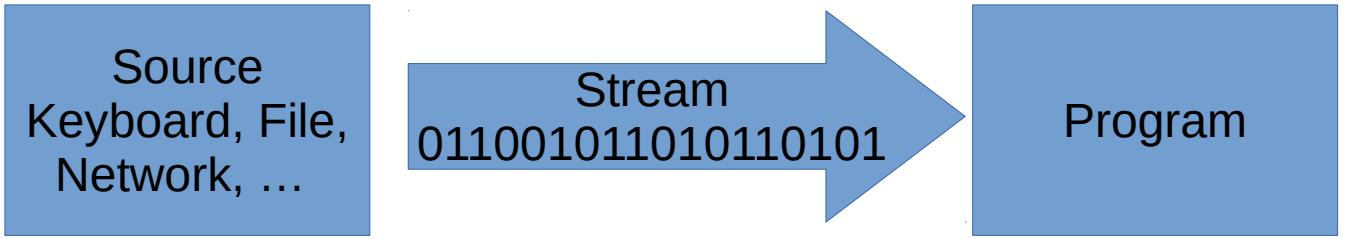
- A *stream* is a sequence of bits that flows from a writer (source) to a reader (sink).
  - Streams abstract input and output so that you can
    - Read from the keyboard, files, a network, or any other device in the same way.
    - Write to the monitor, files, a network, or any other device in the same way
  - Streams abstract the process of reading a writing away from the source and the sink.

Why Streams

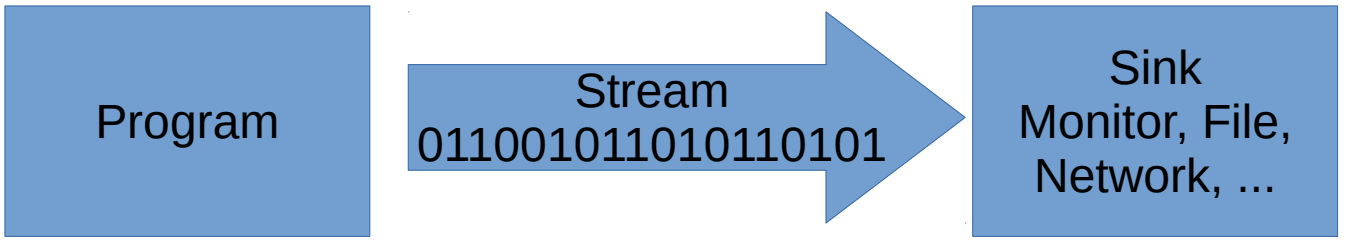
- Streams reduce *coupling* (low coupling is good)
  - Reducing coupling reduces the knowledge one object needs about another.
  - Streams do not need to know where the bits come from or go to.
- Streams increase *cohesion* (high cohesion is good)
  - Cohesion is the degree to which elements belong together
  - Steam operations focus only on the interpretation of the bits moving from the source to the sink

Input and Output Streams

- Input Stream



- Output Stream



System

- System: Constant class defines program elements
  - “public final class System extends Object”
- Defines standard I/O
  - “in”: the standard input, usually the keyboard
  - “out”: the standard output, usually the monitor
  - “err”: the standard error, usually also the monitor
- System is automatically part of your program

Standard Input and Output

- Standard output
  - “static PrintStream in”
- Standard input
  - “static InputStream out”
- Standard error
  - “static PrintStream err”

Monitor Output

Using `System.out` and `System.err`

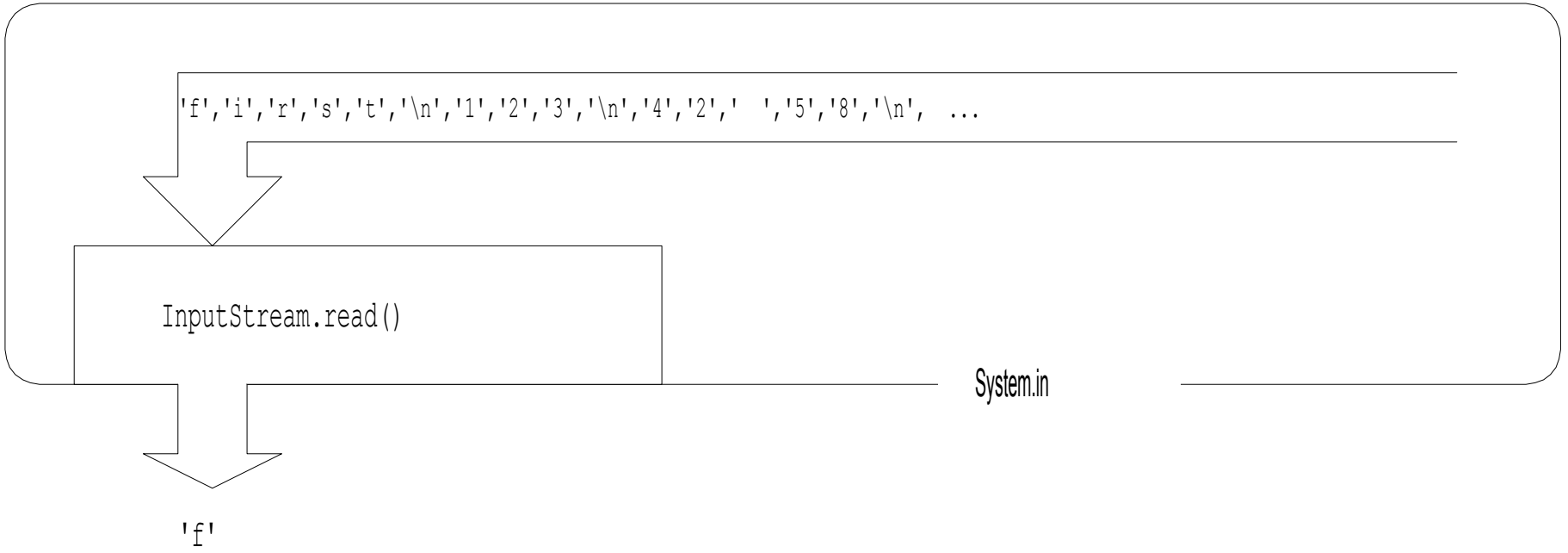
- Output Streams, such as System.out and System.err are easier to use than System.in
- Methods
  - print(String x) and println(String x)
    - Prints a single string to the output stream
      - Println follows the string with a newling
  - format(String format, args …)
    - Like C, printf()
    - The format string is a string and the string representations of the args replace characters preceded by %
      - e.g., %c for character, %d for integer, %s for string
    - May use '%n' for newline, '%t' for tab, etc.

Keyboard Input

Using `System.in`

- `System.in` is an `InputStream`
- Can use `System.in` many ways
  - Directly (low-level access)
  - Through layers of abstraction (high-level access)

Using `System.in`



- `read()` method reads one character at a time.
- Returns an `int`
- Returns –1 for EOF (End of File = Ctrl-Z)

Using `System.in`

```
public static void main(String[] args)
  throws java.io.IOException
  {
    char character;

    // Prompt for a character and read it
    System.out.print("Enter a character: ");
    System.out.flush();
    character = (char) System.in.read();

    // Display the character typed
    System.out.println();
    System.out.println("You typed " + character);
  }
```

Using `System.in`

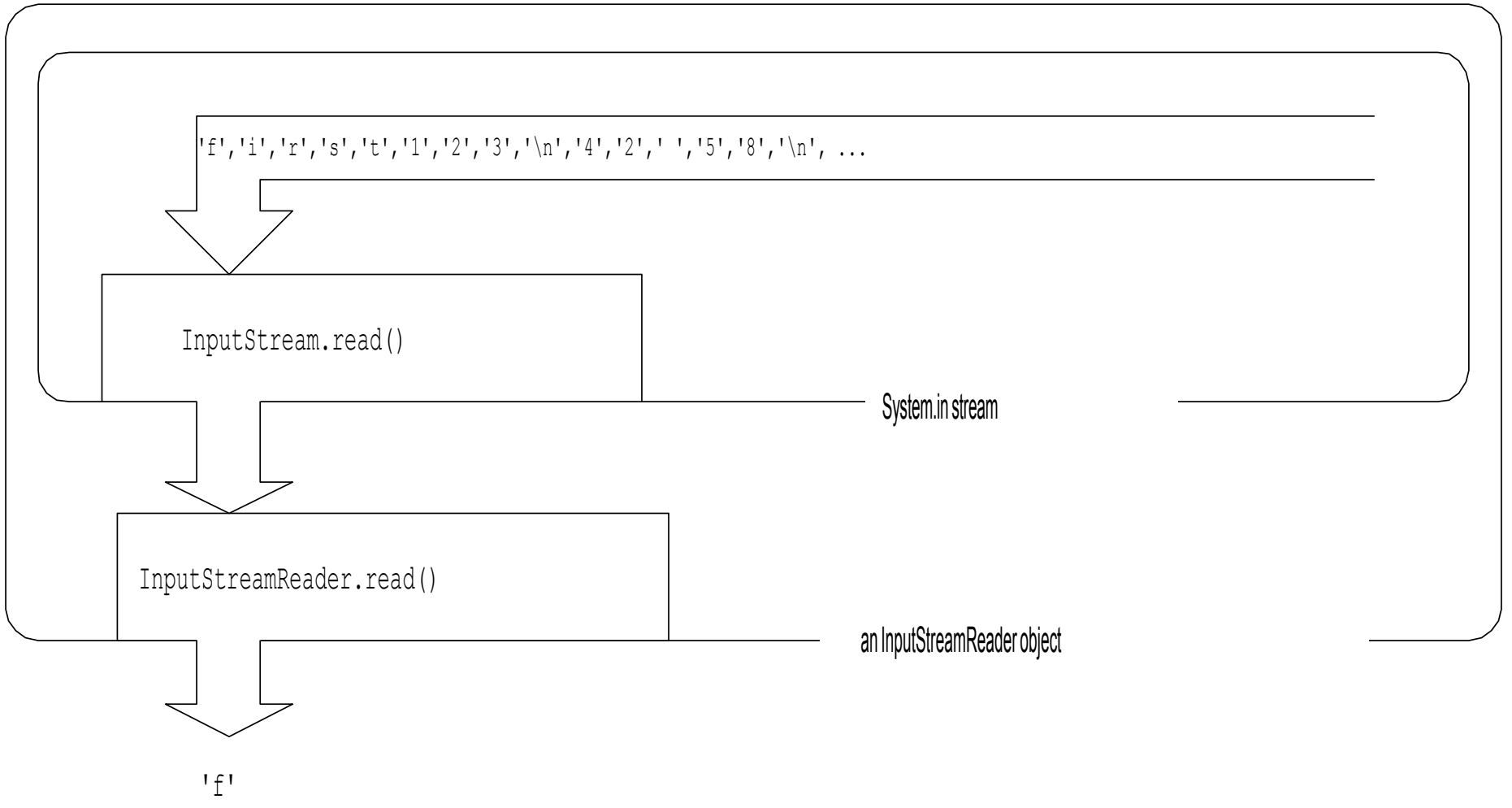
```
int intChar = System.in.read();

while (intChar != -1)
{
  // Convert to character
  char character = (char) intChar;

  System.out.println("Next character is " +
  character);

  // Get next one
  intChar = System.in.read();
}
```

Reading Strings

- No String-reading methods in `System.in`
- To read strings from keyboard, first wrap `System.in` inside `InputStreamReader` object:

```
InputStreamReader ISReader
      = new InputStreamReader(System.in);
```

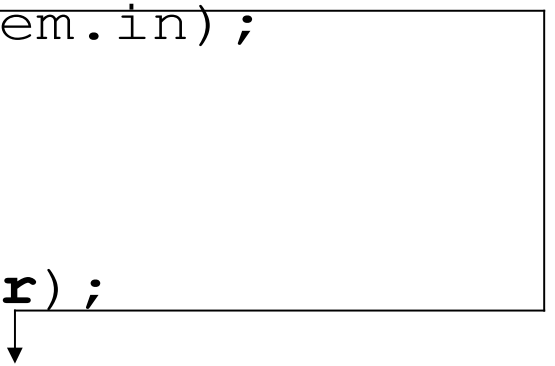
Reading Strings



Reading Strings

- Next, wrap `InputStreamReader` object in `BufferedReader` object:

```
InputStreamReader ISReader
  = new InputStreamReader(System.in);

BufferedReader BufReader
  = new BufferedReader(ISReader);
```



Reading Strings

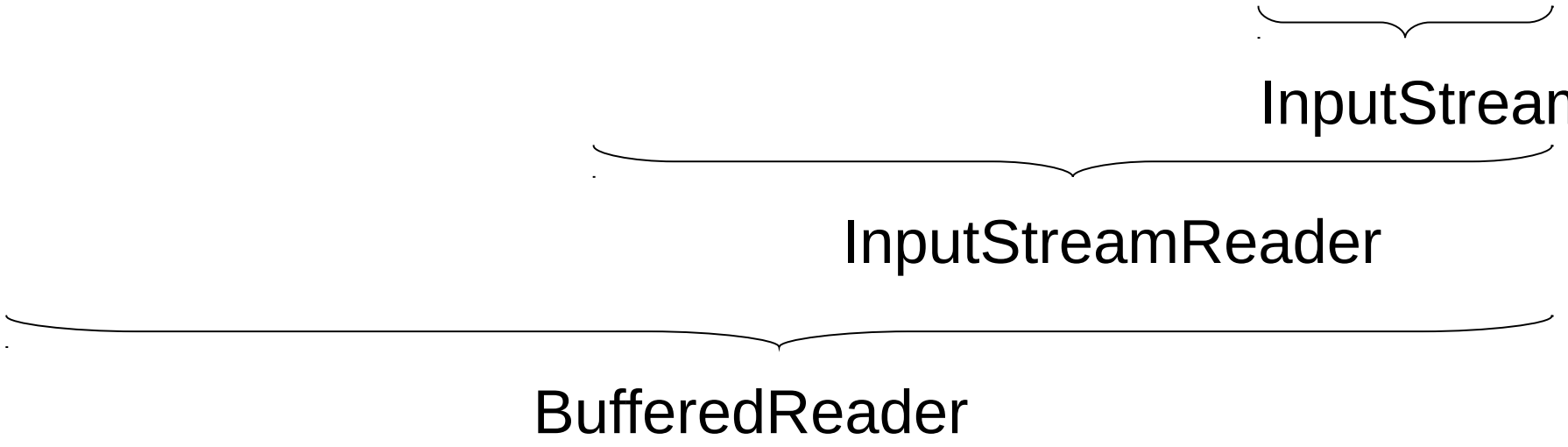
- Can combine these into a single statement:

```
BufferedReader BufReader
  = new BufferedReader(new InputStreamReader(System.in));
```

Reading Strings

- Can combine these into a single statement:

```
BufferedReader BufReader
  = new BufferedReader(new InputStreamReader(System.in));
```



Reading Strings

- Methods in BufferedReader
  - `read()`
  - `readLine()`

- Read a string by using `readLine()` method:

```
String str = BufReader.readLine();
```

Reading Strings

Reading Strings

- `InputStreamReader`, `BufferedReader` in `java.io.*`

- Skeleton for reading:

```
import java.io.*;

class ClassName
{
  public static void main(String[] args)
   throws java.io.IOException
  {
    // Create a buffered input stream and attach it to standard
    // input
    BufferedReader BufReader
      = new BufferedReader(new InputStreamReader(System.in));
    ...
  }
}
```

Example: Reading User's Name

```
// Reads a user's first name and last name.
// Demonstrates use of InputStreamReader,
// BufferedReader and the readLine() method.

import java.io.*;

class ReadInputAsString
{
  public static void main(String[] args)
   throws java.io.IOException
  {
    String firstName, lastName;

    // Create an input stream and attach it to the standard
    // input stream
    BufferedReader BufReader
      = new BufferedReader(new InputStreamReader(System.in));
```

Example: Reading User's Name

```
    // Read a line from the user as a String
    System.out.print("Enter your first name: ");
    System.out.flush();
    firstName = BufReader.readLine();


    // Read a line from the user as a String
    System.out.print("Enter your last name: ");
    System.out.flush();
    lastName = BufReader.readLine();

    // Display the strings
    System.out.println();
    System.out.println("Your name is" + firstName + lastName);
  }
}
```

Example: Reading User's Name

```
    // Read a line from the user as a String
    System.out.print("Enter your first name: ");
    System.out.flush();
    firstName = BufReader.readLine();


    // Read a line from the user as a String
    System.out.print("Enter your last name: ");
    System.out.flush();
    lastName = BufReader.readLine();

    // Display the strings
    System.out.println();
    System.out.println("Your name is" + firstName + lastName);
  }
}
```

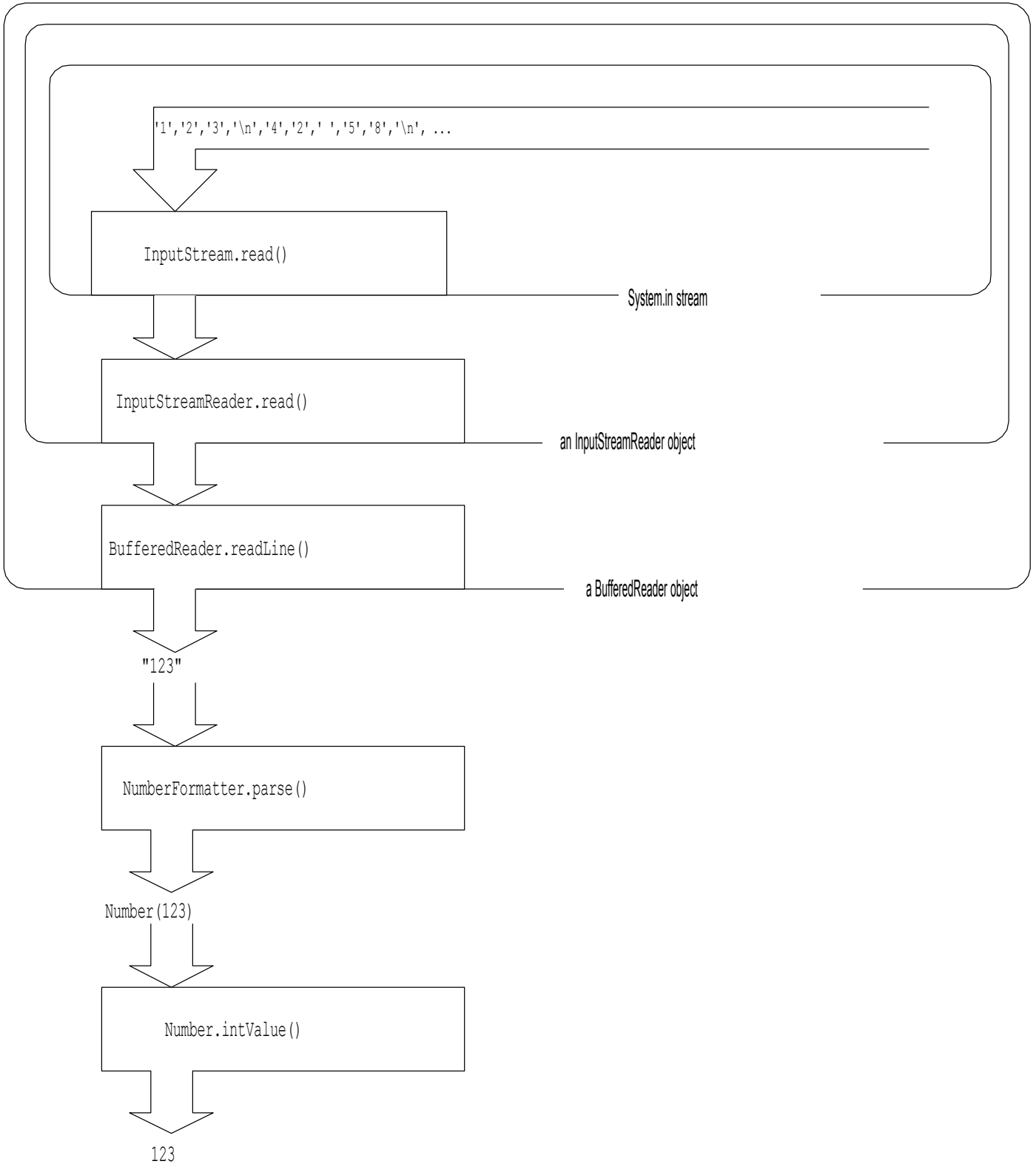
```
Enter your first name: Linda
Enter your last name: Jones

Your name is Linda Jones
```

Formatting Numbers

- Alternative to `Integer.parseInt()` etc.

- `NumberFormat`
  - Object factory: use `getInstance()`
  - `parse()` takes a string and returns a `Number`
  - `parse()` throws `java.text.ParseException`

- General `Number` class
  - Return value using `intValue()`, `doubleValue()` etc.


'1','2','3','\n','4','2',' ','5','8','\n', ...
InputStream.read()
System.in stream
InputStreamReader.read()
an InputStreamReader object
BufferedReader.readLine()
a BufferedReader object
"123"
NumberFormatter.parse()
Number(123)
Number.intValue()
123

Formatting Numbers from Kbd

```
NumberFormat formatter = NumberFormat.getInstance();
BufferedReader BufReader
      = new BufferedReader(new InputStreamReader(System.in));

System.out.print("Enter an integer: ");
System.out.flush();
String response = BufReader.readLine();

Number numberObj = formatter.parse(response);

int numberInt = numberObj.intValue();
```

Formatting Numbers from Kbd

```
NumberFormat formatter = NumberFormat.getInstance();
BufferedReader BufReader
      = new BufferedReader(new InputStreamReader(System.in));

System.out.print("Enter an integer: ");
System.out.flush();
String response = BufReader.readLine();

Number numberObj = formatter.parse(response);

int numberInt = numberObj.intValue();
```

Combine

Formatting Numbers from Kbd

```
NumberFormat formatter = NumberFormat.getInstance();
BufferedReader BufReader
      = new BufferedReader(new InputStreamReader(System.in));

System.out.print("Enter an integer: ");
System.out.flush();

int numberInt
      = formatter.parse( BufReader.readLine() ).intValue();
```

Formatting Numbers from Kbd

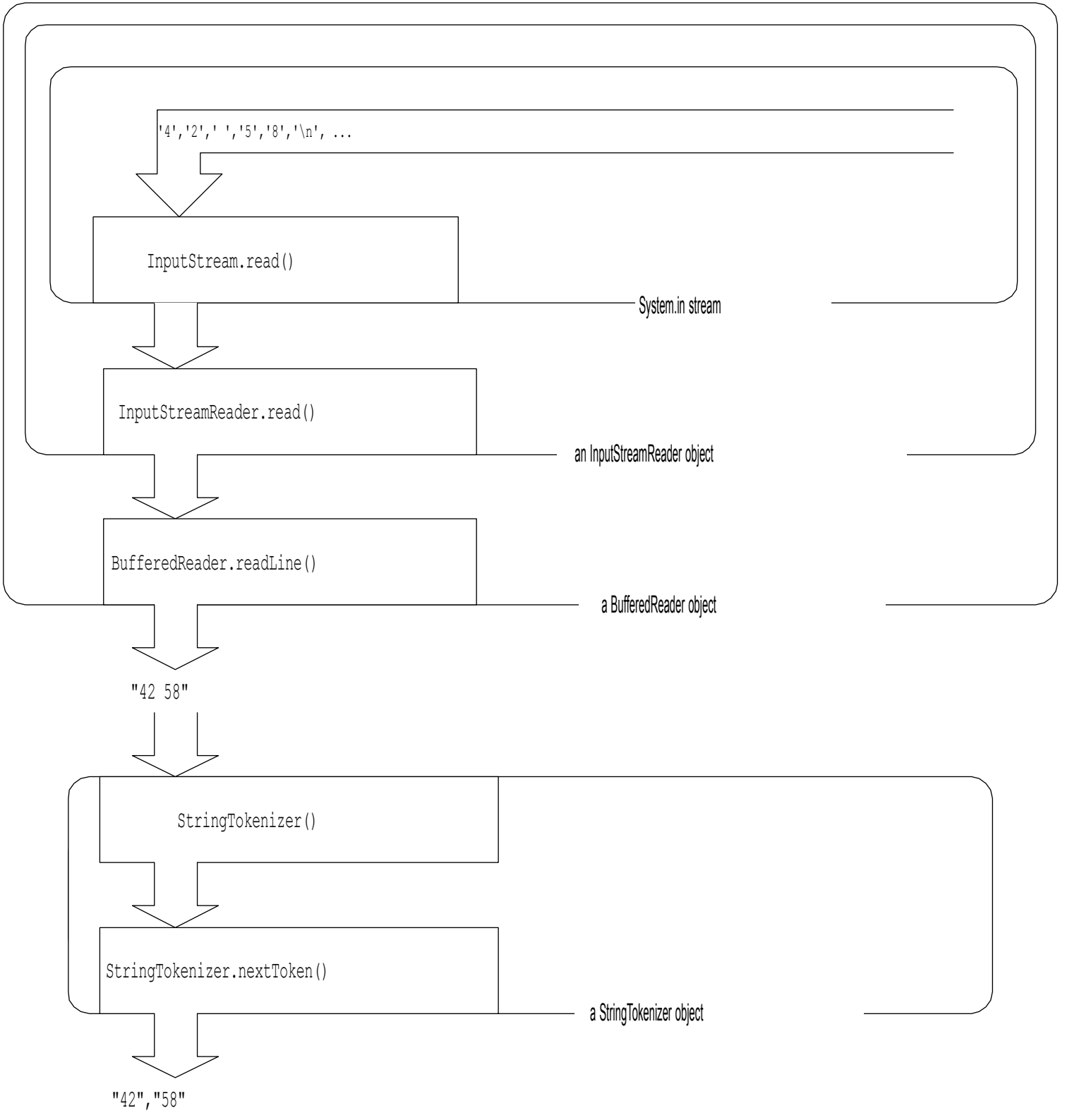
```
NumberFormat formatter = NumberFormat.getInstance();
BufferedReader BufReader
      = new BufferedReader(new InputStreamReader(System.in));

System.out.print("Enter an integer: ");
System.out.flush();

int numberInt
      = formatter.parse( BufReader.readLine() ).intValue();
```

Don’t forget to throw or catch java.text.ParseException

Input Streams: Multiple Values

- To read several values on one line
  - Use `StringTokenizer` object
  - Breaks one string into component parts
  - Must still convert numbers (if necessary)

- StringTokenizer
  - Constructor takes string
  - `nextToken()` method
  - `hasMoreTokens()` method


'4','2',' ','5','8','\n', ...
InputStream.read()
System.in stream
InputStreamReader.read()
an InputStreamReader object
BufferedReader.readLine()
a BufferedReader object
"42 58"
StringTokenizer()
StringTokenizer.nextToken()
a StringTokenizer object
"42","58"

Input Streams: Multiple Values

```
StringTokenizer tokenizer = new StringTokenizer("42 58");

String token;
token = tokenizer.nextToken();
System.out.println(token);    // Displays 42
token = tokenizer.nextToken();
System.out.println(token);    // Displays 58
```

- BETTER…

```
StringTokenizer tokenizer = new StringTokenizer("42 58");
while (tokenizer.hasMoreTokens())
{
  System.out.println(tokenizer.nextToken());
}
```