Inf1-OP

Creating Classes

Volker Seeker, adapting earlier version by Perdita Stevens and Ewan Klein

School of Informatics

February 6, 2019

Custom Libraries

Standard I/O library[1]

[1]Thanks to Sedgewick&Wayne for much of this content

Command-Line Input vs. Standard Input

Command-line inputs

- Useful for reading in a few user values.
- Not practical for large amount of input data.
- Input entered before program begins execution.
- In Eclipse, requires changing `Run Configurations... > Arguments`

Standard input

- Flexible OS abstraction for any input.
- By default, standard input is received from terminal window.
- Input entered while program is executing.
- In Eclipse, input can be entered via the Console window.

Standard Input and Output

Standard input: `StdIn` is Sedgewick&Wayne-specific library for reading text input.

`public class StdIn`

`boolean`	`isEmpty()`	`true` *if no more values,* `false` *otherwis*
`int`	`readInt()`	*read a value of type* `int`
`double`	`readDouble()`	*read a value of type* `double`
`long`	`readLong()`	*read a value of type* `long`
`boolean`	`readBoolean()`	*read a value of type* `boolean`
`char`	`readChar()`	*read a value of type* `char`
`String`	`readString()`	*read a value of type* `String`
`String`	`readLine()`	*read the rest of the line*
`String`	`readAll()`	*read the rest of the text*
`String`	`redirectInput(String fn)`	*read the contents of file named* `fn`

Standard Input and Output

Standard output: `StdOut` is Sedgewick&Wayne-specific library for writing text output.

`public class StdOut`

`void`	`print(String s)`	*print* `s`
`void`	`println(String s)`	*print* `s`*, followed by newline*
`void`	`println()`	*print a newline*
`void`	`printf(String f, ...)`	*formatted print*

Standard Input and Output

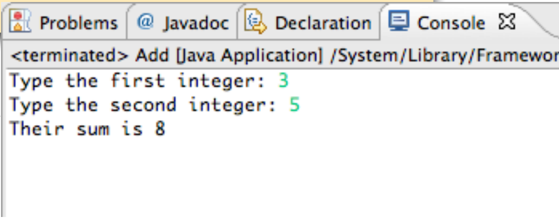
`StdIn.java`, `StdOut.java`, `StdDraw.java` and `StdRandom.java` have been bundled together as a zip archive: `http://www.inf.ed.ac.uk/teaching/courses/inf1/op/2019/labs/resources/stdlib.zip`.

- Get the libraries from above URL.
- Unzip and place the files in the same directory as your other `.java` source files.
- In this case, you do not to have to explicitly `import` the libraries.
- Full set of Sedgewick&Wayne libraries are also available for download from their booksite: `http://introcs.cs.princeton.edu/stdlib/`

Standard Input and Output

```
public class Add {
    public static void main(String[] args) {
        StdOut.print("Type the first integer: ");
        int int1 = StdIn.readInt();
        StdOut.print("Type the second integer: ");
        int int2 = StdIn.readInt();
        int sum = int1 + int2;
        StdOut.print("Their sum is " + sum);
    }
}
```

result of running this code in Eclipse



```
<terminated> Add [Java Application] /System/Library/Framewor
Type the first integer: 3
Type the second integer: 5
Their sum is 8
```

Standard Drawing API, 1

Standard Drawing: `StdDraw` is a library for producing graphical output.

```
 public class StdDraw
------------------------------------------------------------
   void point(double x, double y)
   void line(double x0, double y0, double x1, double y1)
   void text(double x, double y, String s)
   void circle(double x, double y, double r)
   void filledCircle(double x, double y, double r)
   void square(double x, double y, double r)
   void filledSquare(double[] x, double[] y)
   void polygon(double x, double y, double r)
   void filledPolygon(double x, double y, double r)
------------------------------------------------------------
```

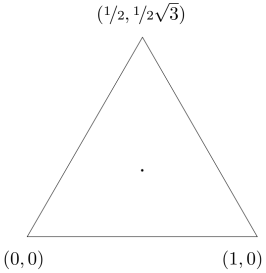
Standard Drawing API, 2

public class StdDraw	
`void setXscale(double x0, double x1)`	*reset* `x` *range to* `(x0, x1)`
`void setYscale(double y0, double y1)`	*reset* `y` *range to* `(y0, y1)`
`void setPenRadius(double r)`	*set pen radius to* `r`
`void setPenColor(Color c)`	*set pen colour to* `c`
`void setFont(Font f)`	*set font to* `f`
`void setCanvasSize(int w, int h)`	*set canvas to* `w`*-by-*`h` *windov*
`void clear(Color c)`	*clear the canvas; colour it* `c`
`void show(int dt)`	*show all; pause* `dt` *millisecoı*
`void save(String fn)`	*save to a file named* `fn` *(wit*

NB Calling these functions with the same names but no arguments resets to default values.

Standard Draw

```
public class Triangle {
    public static void main(String[] args) {
        double t = Math.sqrt(3.0) / 2.0;
        StdDraw.line(0.0, 0.0, 1.0, 0.0);
        StdDraw.line(1.0, 0.0, 0.5, t);
        StdDraw.line(0.5,   t, 0.0, 0.0);
        StdDraw.setPenRadius(.01); // make point bigger
        StdDraw.point(0.5, t/3.0);
        StdDraw.save("triangle.png");
    }
}
```



Exceptions

A way to handle errors.

Things can go wrong

Not only because of bugs in your code, also because of bugs in library code you might use or inaccessible resources.

source:

https://4pyz335b69-flywheel.netdna-ssl.com/wp-content/uploads/2014/05/things-go-wrong.png

Things can go wrong

A systematic and effective way of handling errors is needed.

Error Handling Example

```
readFile
    open the file;
    determine its size;
    allocate that much memory;
    read the file into memory;
    close the file;
```

Error Handling Example

```
errorCodeType readFile {
    initialize errorCode = 0;
    open the file;
    if (theFileIsOpen) {
        determine the length of the file;
        if (gotTheFileLength) {
            allocate that much memory;
            if (gotEnoughMemory) {
                read the file into memory;
                if (readFailed) errorCode = -1;
            else errorCode = -2;
        else errorCode = -3;
        close the file;
        if (theFileDidntClose && errorCode == 0)
            errorCode = -4;
        else errorCode = errorCode and -4;
    else errorCode = -5;
    return errorCode;
}
```

Error Handling Example

```
readFile {
    try {
        open the file;
        determine its size;
        allocate that much memory;
        read the file into memory;
        close the file;
    } catch (fileOpenFailed) {
       doSomething;
    } catch (sizeDeterminationFailed) {
        doSomething;
    } catch (memoryAllocationFailed) {
        doSomething;
    } catch (readFailed) {
        doSomething;
    } catch (fileCloseFailed) {
        doSomething;
    }
}
```

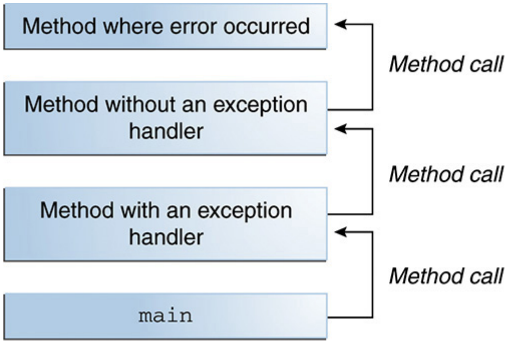
Error Handling with Exceptions

Exceptional Events

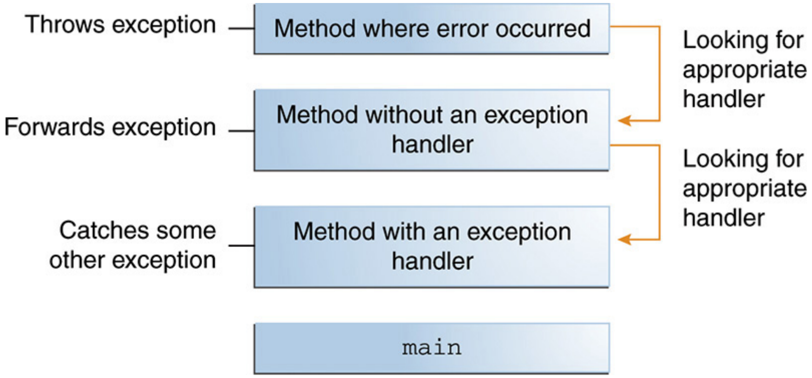
Exceptions allow a clear separation of the **program logic** from its **error handling** code.

They encapsulate **type**, **message** and **location** of an error.

Error Handling with Exceptions


Method where error occurred
Method call
Method without an exception handler
Method call
Method with an exception handler
Method call
main

Error Handling with Exceptions


Throws exception
Method where error occurred
Looking for appropriate handler
Forwards exception
Method without an exception handler
Looking for appropriate handler
Catches some other exception
Method with an exception handler
main

Exceptions are passed up the call stack until a handler is found.

Try-Catch

```
try {
   Scanner s = new Scanner(new File("sample.txt"));
} catch (FileNotFoundException e) {
   System.out.println("Specified file not found: " + e);
   System.exit(1);
}
```

- Try block surrounds method which throws exception
- Catch block handles it
- Multiple catch blocks are possible for different exceptions

Passing Exceptions On

```
public void readNumbers(String file) throws FileNotFoundException {
    Scanner sc = new Scanner(file);

    while (sc.hasNextLine()) {
        int i = sc.nextInt();
        System.out.println(i);
    }
    sc.close();
}
```

- The throws keyword added to the method header with the corresponding exception type can pass on exceptions to the calling code
- No all exceptions need to be indicated by throws, e.g. IllegalArgumentException, IndexOutOfBoundsException

Throw your own

```
public void happyBirthday(int age) {
    if (age < 0) {
         throw new IllegalArgumentException(
                "Age must be positive, but is: " + age);
    }

    for (int i = 0; i < age; i++)
        System.out.println("Hip Hip - Horay!");
}
```

- The throw keyword together with a new exception object can be used to raise your own exceptions.
- This is useful when protecting the API of your class against invalid input.

Summary

- Exceptions are Java's way of handling exceptional events, i.e. error cases
- They encapsulate type, message and location of the error
- They are handled using **try-catch**
- They are forwarded using **throws**
- They are raised using **throw**

Reading

Java Tutorial

Chapter 10 *Exceptions*

Chapter 11 *Basic I/O and NIO*

Those two chapters contain more than what is required in this course, so read what you need and remember where you can look up the rest for later.