6.092: Introduction to Java

6: Interfaces, Input/Output, Understanding Exceptions

Final Lecture

- Review
- Interfaces
- Input/Output (I/O)
- Understanding Exceptions
- Administrivia

```
public class DrawGraphics{
        BouncingBox box;

        public DrawGraphics(){
                box = new BouncingBox(200,50,Color.RED);
        }

        public void draw(Graphicssurface){
                surface.drawLine(50,50,250,250);
                box.draw(surface);
        }
}
```

```
public class DrawGraphics{
        BouncingBox box;

        public DrawGraphics(){
                box = new BouncingBox(200,50,Color.RED);
        }

        public void draw(Graphicssurface){
                surface.drawLine(50,50,250,250);
                box.draw(surface);
        }
}
```

```
public class DrawGraphics{
        BouncingBox box;

        public DrawGraphics(){
                box = new BouncingBox(200,50,Color.RED);
        }

        public void draw(Graphicssurface){
                surface.drawLine(50,50,250,250);
                box.draw(surface);
        }
}
```

Field

```
public class DrawGraphics{
        BouncingBox box;

        public DrawGraphics(){
                box = new BouncingBox(200,50,Color.RED);
        }

        public void draw(Graphicssurface){
                surface.drawLine(50,50,250,250);
                box.draw(surface);
        }
}
```

```
public class DrawGraphics{
        BouncingBox box;

        public DrawGraphics(){
                box = new BouncingBox(200,50,Color.RED);
        }
```

Constructor

```
        public void draw(Graphicssurface){
                surface.drawLine(50,50,250,250);
                box.draw(surface);
        }
}
```

```
public class DrawGraphics{
        BouncingBox box;

        public DrawGraphics(){
                box = new BouncingBox(200,50,Color.RED);
        }

        public void draw(Graphicssurface){
                surface.drawLine(50,50,250,250);
                box.draw(surface);
        }
}
```

```
public class DrawGraphics{
        BouncingBox box;

        public DrawGraphics(){
                box = new BouncingBox(200,50,Color.RED);
        }

        public void draw(Graphicssurface){
                surface.drawLine(50,50,250,250);
                box.draw(surface);
        }
}
```

Method

```
public class DrawGraphics{

        public DrawGraphics(){
                BouncingBox box =
                        new BouncingBox(200,50,Color.RED);
        }

        public void draw(Graphicssurface){
                surface.drawLine(50,50,250,250);
                box.draw(surface);
        }
}
```

What’s wrong here?

```
public class DrawGraphics{
        ArrayList<BouncingBox>boxes=newArrayList<BouncingBox>();

        public DrawGraphics(){
            boxes.add(new BouncingBox(200,50,Color.RED));
            boxes.add(new BouncingBox(10,10,Color.BLUE));
            boxes.get(0).setMovementVector(1,0);
            boxes.get(1).setMovementVector(-3,-2);
        }

        public void draw(Graphics surface) {
                for(BouncingBox box : boxes) {
                        box.draw(surface);
                }
        }
}
```

Interfaces

Java Interfaces

Manipulate objects, without knowing how they work

Useful when you have similar but not identical objects

Useful when you want to use code written by others

Interface Example: Drawing

```
public class BouncingBox {
    public void draw(Graphics surface) {
        // … code to draw the box …
    }
}

// … draw boxes …
for(BouncingBox box : boxes){
    box.draw(surface);
}
```

Things that are cooler than BouncingBox

- Flowers
- Hello Kitty
- Monkey Faces
- 3D cube
- MIT Logo
- …

Draw Something Cool!

```
public class Flower {
    public void draw(Graphics surface) {
        // … code to draw the flower…
    }
}

// … draw flowers…
for(Flower f : flowers){
    f.draw(surface);
}
```

```
public class DrawGraphics {
        ArrayList<BouncingBox> boxes = newArrayList<BouncingBox>();
        ArrayList<Flower> flowers = newArrayList<Flower>();
        ArrayList<Car> cars = newArrayList<Car>();

        ...

        public void draw(Graphics surface) {
            for(BouncingBox box : boxes){
                box.draw(surface);
            }
            for(Flower flower : flowers){
                flower.draw(surface);
            }
            for(Car car : cars){
                car.draw(surface);
            }
        }
}
```

```
public class DrawGraphics {
        ArrayList<Drawable> shapes = newArrayList<Drawable>();
        ArrayList<Flower> flowers = newArrayList<Flower>();
        ArrayList<Car> cars = newArrayList<Car>();

        …

        public void draw(Graphics surface) {
            for(Drawable shape : shapes){
                shape.draw(surface);
            }
            for(Flower flower : flowers){
                flower.draw(surface);
            }
            for(Car car : cars){
                car.draw(surface);
            }
        }
}
```

Interfaces

Set of classes that share methods

Declare an *interface* with the common methods

Can use the interface, without knowing an object’s specific type

Interfaces: Drawable

```
import java.awt.Graphics;
import java.awt.Color;

interface Drawable{
    void draw(Graphics surface);
    void setColor(Color color);
}
```

Implementing Interfaces

Implementations provide complete methods:

```
import java.awt.Graphics;
class Flower implements Drawable {
   // ... other stuff ...
   public void draw(Graphics surface){
        // ... code to draw a flower here ...
   }
}
```

Interface Notes

Only have methods (mostly true)

Do not provide code, only the definition (called *signatures*)

A class can implement any number of interfaces

Using Interfaces

Can only access stuff in the interface.

```
Drawable d = new BouncingBox(…);
d.setMovementVector(1, 1);
```

Using Interfaces

Can only access stuff in the interface.

```
Drawable d = new BouncingBox(…);
d.setMovementVector(1, 1);
```

The method setMovementVector(int, int) is undefined for the type Drawable

Casting

If you know that a variable holds a specific type, you can use a cast:

```
Drawable d = new BouncingBox(…);
BouncingBox box = (BouncingBox) d;
box.setMovementVector(1, 1);
```

Input/Output (I/O)

We’ve seen Output

```
System.out.println(“some string”);
```

What About Input?

```
public class FooCorporation {

        public static void payment(double pay, double hours) {
                if (pay < 8) {
                …
        }

        public static void main(String[] args) {
                payment(7.5, 35);
                payment(8.2,42);
                …
        }
}
```

What About Input?

```
public class FooCorporation {

    public static void payment(double pay, double hours) {
        if (pay < 8) {
        …
    }

    public static void main(String[] args) {
        payment(7.5, 35);
        payment(8.2,42);
        …
    }
}
```

Hire new employee?

The Full Picture

10010101010101000101 ...

InputStream

`System.in`

‘O’ ‘k’ ‘a’ ‘y’ ‘ ‘ ‘a’ ‘w’ ‘e’ …

InputStreamReader

“Okay awesome, cool\n” …

BufferedReader

InputStream

- InputStream is a stream of bytes
  - Read one byte after another using `read()`
- A byte is just a number
  - Data on your hard drive is stored in bytes
  - Bytes can be interpreted as characters, numbers..

InputStream

- InputStream is a stream of bytes
  - Read one byte after another using `read()`
- A byte is just a number
  - Data on your hard drive is stored in bytes
  - Bytes can be interpreted as characters, numbers..

```
InputStream stream = System.in;
```

InputStreamReader

- Reader is a stream of characters
  - Read one character after another using `read()`
- InputStreamReader takes an InputStream and converts bytes to characters

InputStreamReader

- Reader is a stream of characters
  - Read one character after another using `read()`
- InputStreamReader takes an InputStream and converts bytes to characters

```
new InputStreamReader(stream)
```

BufferedReader

- BufferedReader buffers a character stream so you can read line by line
  - `String readLine()`

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

```
InputStreamReader ir = new
   InputStreamReader(System.in);
BufferedReader br = new
   BufferedReader(ir);
```

```
InputStreamReader ir = new
   InputStreamReader(System.in);
BufferedReader br = new
   BufferedReader(ir);

System.out.println("Pay? ");
String pStr = br.readLine();
System.out.println("Hours? ");
String hStr = br.readLine();
```

```
InputStreamReader ir = new
   InputStreamReader(System.in);
BufferedReader br = new
   BufferedReader(ir);

System.out.println("Pay? ");
String pStr = br.readLine();
System.out.println("Hours? ");
String hStr = br.readLine();

double pay = Double.parseDouble(pStr);
double hours = Double.parseDouble(hStr);
```

FileReader

- FileReader takes a text file
  - converts it into a character stream
  - FileReader(“PATH TO FILE”);

- Use this + BufferedReader to read files!

```
FileReader fr = new
   FileReader(“readme.txt”);
BufferedReader br = new
   BufferedReader(fr);
```

FileReader Code

```
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;

public class ReadFile {

  public static void main(String[] args)  throws IOException {
    FileReader fr = new FileReader("readme.txt");
    BufferedReader br = new BufferedReader(fr);
    String line = null;
    while ((line = br.readLine()) != null) {
      System.out.println(line);
    }
    br.close();

  }
}
```

More about I/O

- http://java.sun.com/docs/books/tutorial/essential/io/

Understanding Exceptions

Exceptions

- NullPointerException
- ArrayIndexOutOfBoundsException
- ClassCastException
- RuntimeException

What is an “Exception”?

- Event that occurs when something “unexpected” happens

```
Exception in thread "main"
java.lang.ArrayIndexOutOfBoundsException: 5
   at SimpleDraw.main(SimpleDraw.java:8)
```

How do exceptions “happen”?

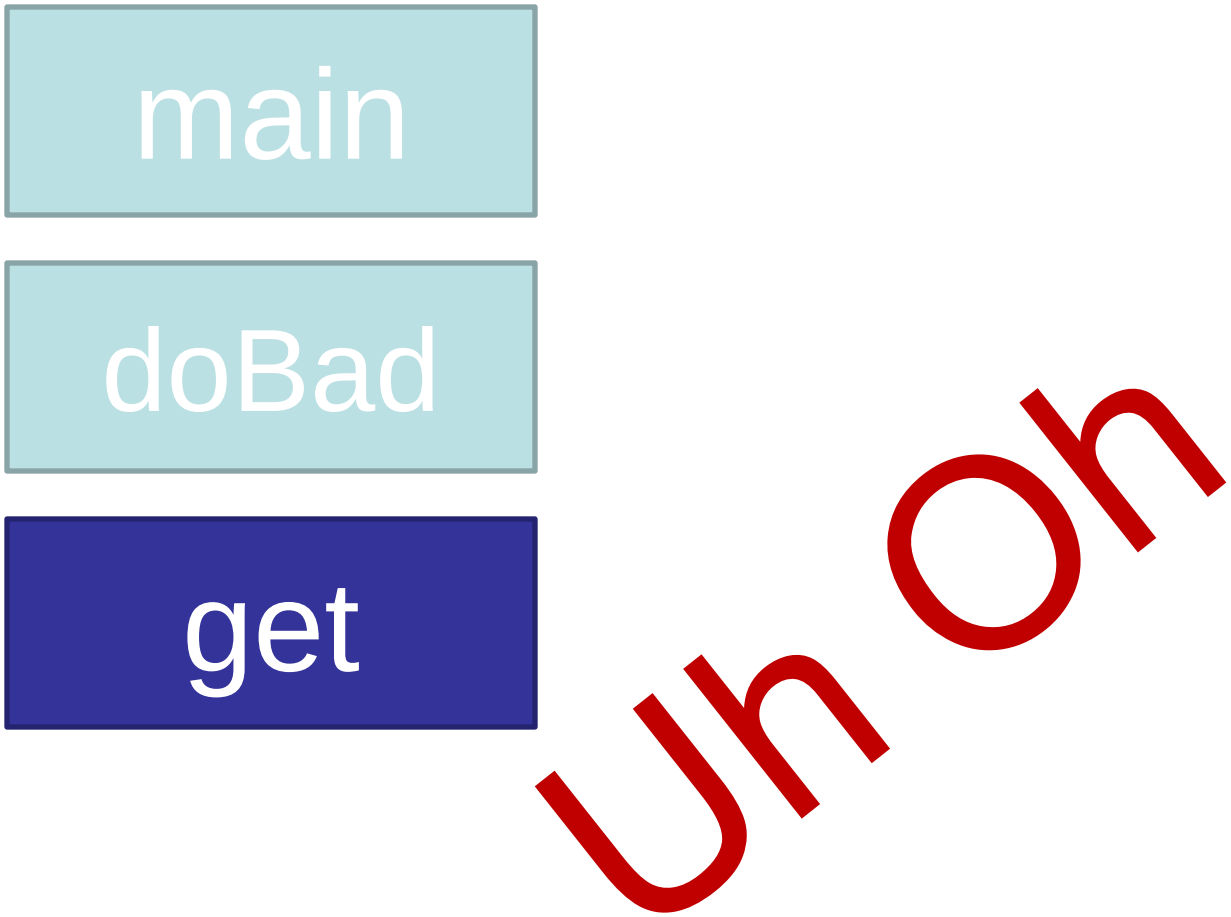
- Java doesn’t know what to do, so it
  - Creates an Exception object
  - Includes some useful information
  - “throws” the Exception

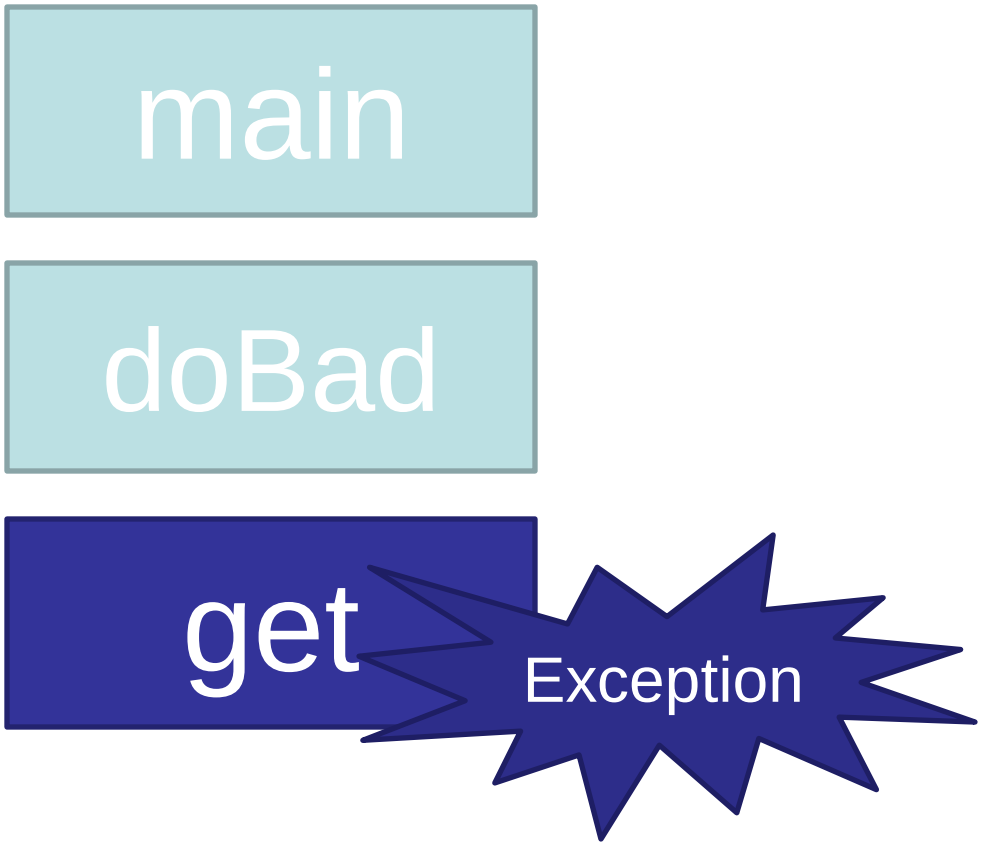
```
public class YourClass {
        ArrayList<BouncingBox> boxes;

        public static BouncingBox get() {
                return boxes.get(1);
        }

        public static void doBad() {
                BouncingBox b = get();
        }

        public static void main(String[] args) {
                doBad();
        }
}
```

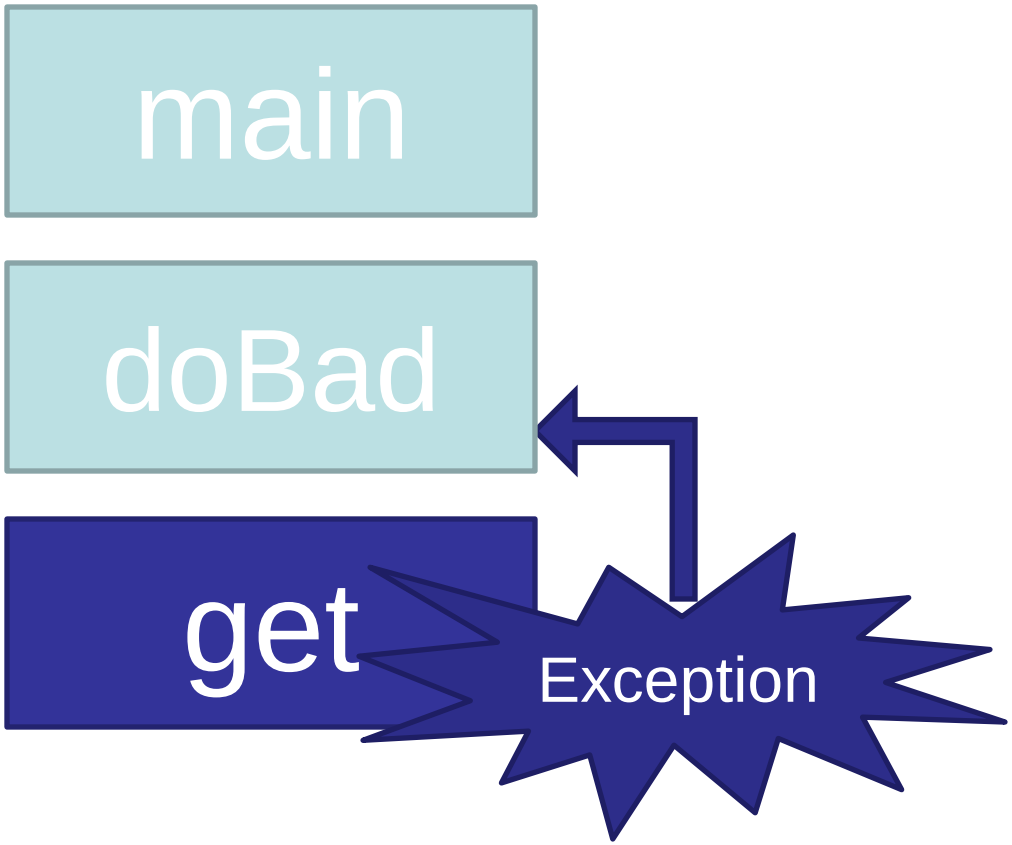
main

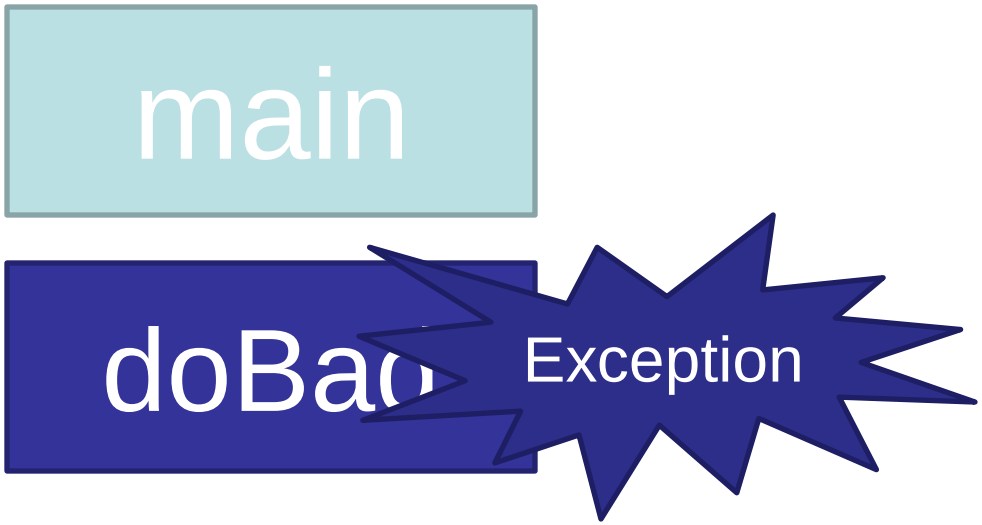

main
doBad

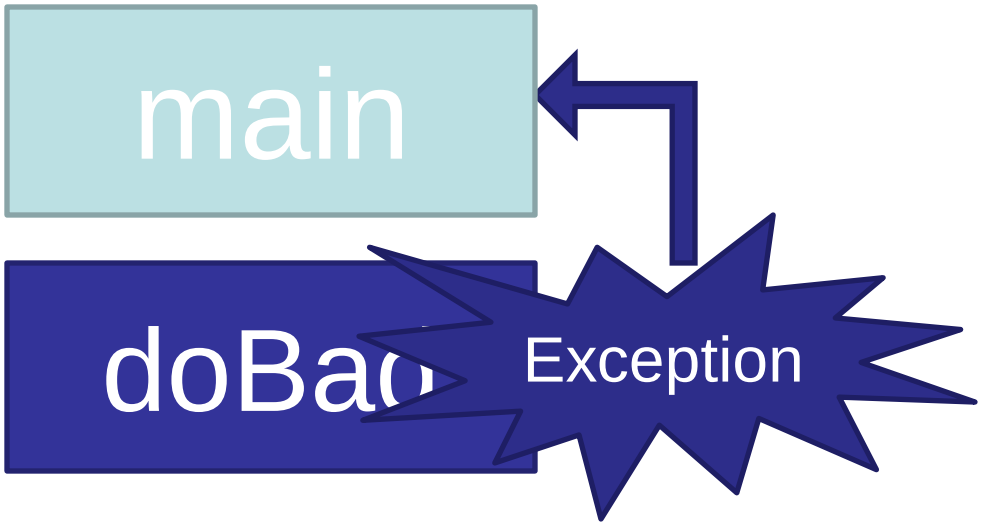

main
doBad
get


main
doBad
get
Uh Oh

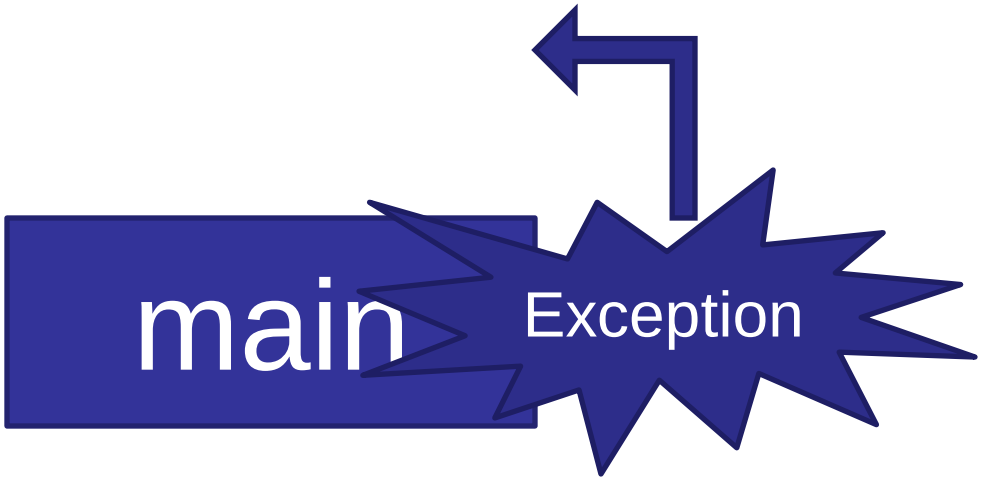

main
doBad
get
Exception


main
doBad
get
Exception


main
doBad
Exception


main
doBad
Exception


main
Exception


main
Exception

What will you see?

```
Exception in thread "main"
java.lang.NullPointerException
    at YourClass.get(YourClass.java:10)
    at YourClass.doBad(YourClass.java:14)
    at YourClass.main(YourClass.java:18)
```

What will you see?

```
Exception in thread "main"
java.lang.NullPointerException
    at YourClass.get(YourClass.java:10)
    at YourClass.doBad(YourClass.java:14)
    at YourClass.main(YourClass.java:18)
```

The line number
is your biggest hint

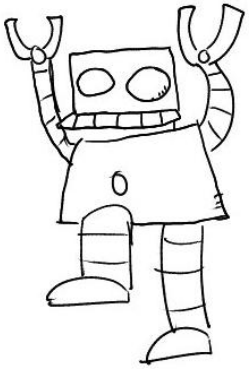
Exceptions

- http://java.sun.com/docs/books/tutorial/essential/exceptions
- http://en.wikipedia.org/wiki/Exceptions

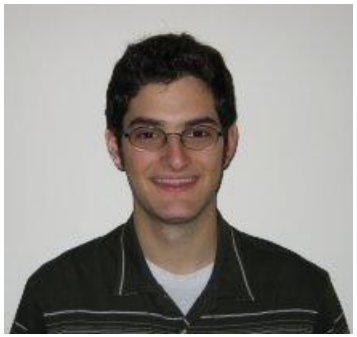
Thanks!

Evan

Dina



Eugene



Adam

Questions?

- Interfaces, I/O, Exceptions
- Bigger-picture: where to go from here?

Grades

- Please verify your assignment grades are what you expect
- You can drop one assignment

Course Evaluation

Please evaluate the course so we can improve!

Feedback from people that dropped is very useful!

http://sixweb.mit.edu

Thanks For Attending!

Assignment:
More graphics, I/O

Start a new project: code has changed.