CSI201 Week 6-1 is 5

- Chapter 3 Concepts Summary
- Project 03 Introduction
- Programming Last Weeks Computation Example as a Method
  - make and use “class methods” G&E 3.3.1
- Making a GCD (Greatest Common Divisor) Space Telescope.

Summary after G&E sec. 3.7

- 3.7.1 Invoking Object Methods

objectRef . *methodName* (*paramList*) ;

IMPOSSIBLE without an object to invoke the object method ON, WITH, or FOR!!

Creating Objects after GE 3.7.3

- **`new`** *ClassName* **`(`** *paramList* **`);`**

(usually) USELESS w/o FIRST making a reference variable so the shiny new object can be referred to AFTER it was built!

- Example:

```
Turtle tu;

tu = new Turtle( new World( ));
```

GE 3.7.1 and 3.7.3 together

```
Turtle tu;
tu = new Turtle( new World( ));
tu.forward( 197 );
```

ClassName refVariable **;**

refVariable **=** **new** *ClassName* **(** *params1* **);**

refVariable.*methName* **(** *params2* **);**

GE 3.7.1 and 3.73 together

```
Turtle tu;
tu = new Turtle( new World( ));
tu.forward( 197 );
```

Example

ClassName refVariable **;**

refVariable **=** **new** *ClassName* **(** *params1* **);**

refVariable **.** *methName* **(** *params2* **);**

Generic Description

What's Happening???

```
Turtle tu;
```

Make a variable good for referring to a **`Turtle`** object.

```
tu = new Turtle( new World( ));
```

First, make a new **`World`**. Second, make a new **`Turtle`**. The **`Turtle`** maker gets the reference to that **`World`** as a parameter.

```
tu.forward( 197 );
```

Call the **`forward`** method on, with, for THAT **`Turtle`**, with param. value 197.

(There is no) Dumb Question.

```
Turtle tu;

World worldObjRef;
```

Don't you need to declare a World variable?

This is what the textbook shows!

```
worldObjRef = new World( );

tu = new Turtle( worldObjRef );

tu.forward( 197 );
```

You could if you like, but DON'T NEED TO.

```
Turtle tu;
```

```
World worldObjRef;
```

Zero-th, make an (unnecessary) variable good for referring to a **`World`**.

```
worldObjRef = new World( );
```

First, make a new **`World`**.

```
tu = new Turtle( worldObjRef );
```

Second, make a new **`Turtle`**. The **`Turtle`** maker gets the reference to that **`World`** as a parameter.

```
tu.forward( 197 );
```

(There is no) Dumb Question.

```
Turtle tu;

World worldObjRef;
```

Instead of the above,

IS IT OK to REVERSE THE ORDER?

```
World worldObjRef;

Turtle tu;
```

(A) Yes (B) NO!!!

```
.............

worldObjRef = new World( );

tu = new Turtle( worldObjRef );
```

(There is no) Dumb Question.

```
World worldObjRef;

worldObjRef = new World( );

Turtle tu;

tu = new Turtle( worldObjRef );
```

What about this??

(A) OK (B) BAD!!:(

(There is no) Dumb Question.

```
World worldObjRef;
worldObjRef = new World( );
Turtle tu;
tu = new Turtle( worldObjRef );
```

That's fine. But what about

```
tu = new Turtle ( worldObjRef );
```

```
Turtle tu;
```

(A) OK (B) BAD!!:(

```
1  public class BadExample {
2    public static void main(String[] a)
3    {
4      World worldObjRef;
5      worldObjRef = new World();
6      tu = new Turtle( worldObjRef );
7      Turtle tu;
8    }
9  }
```

```
1 error found:
File: /home/seth/Courses/CSI201 8/BadExample.java  [line: 6]
Error: /home/seth/Courses/CSI2( 08/BadExample.java:6: cannot find symbol
symbol  : variable tu
location: class BadExample
```

```
1  public class BadExample {
2    public static void main(String[] a)
3    {
4      World worldObjRef;
5      worldObjRef = new World();
6      tu = new Turtle( worldObjRef );
7      Turtle tu;
8    }
9  }
```

Computers generally do commands
IN THE ORDER YOU WRITE THEM.

Line 6 is BEFORE Line 7.
Line 6 tries to USE a variable named tu
BEFORE that variable is declared
in Line 7. So it fails AT 6.
“Declared” means the variable and its name
are put into the computer's memory.

Creating new (YOUR OWN) methods 3.7.4

- Decide what kind of objects you want to give YOUR new potential behavior to. Suppose that kind is *ClassName*
- Open that class's definition file *ClassName*`.java`
- Put ***directly*** within the outermost `{ ... }` your method's definition:

`public` *returnType methodName*`(`

parmType1 parmName1`,` *parmType2 parmName2*`)`

`{` *code that programs your behavior*

`}`

Calling your own methods on objects like Turtles

- EASY! Just like calling the methods that CAME (from G&E) with **`Turtles`**.

Project 03

Add new image processing methods to G&E **`Picture`**s,

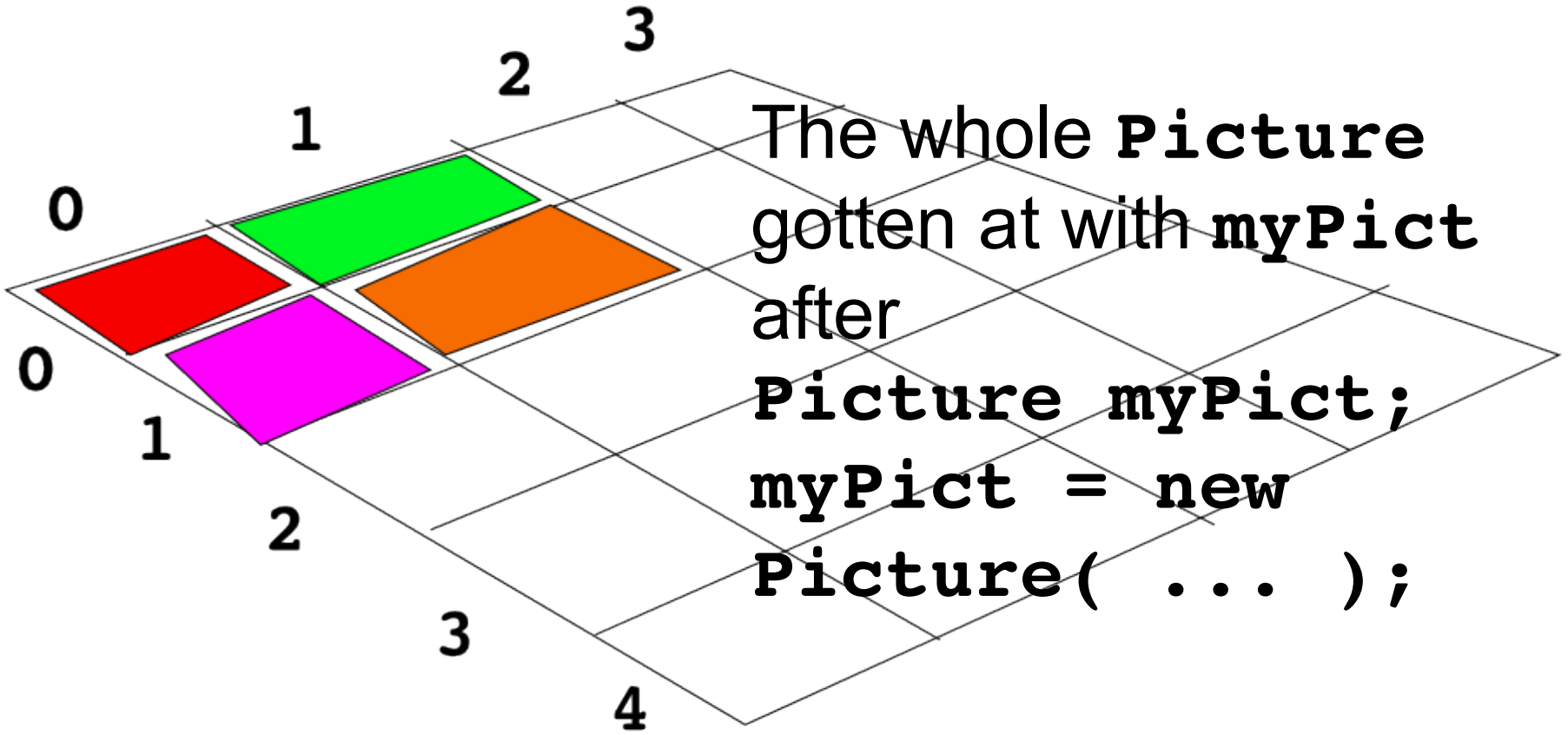
your new methods must be parametrized by
(1) what locations in the **`Picture`** to modify
(2) some details to control the modification

Programming Topics to learn:
Array idea
Using a return value returned by a method call
Scope of names of variables
Specifying packages sometimes
Files
Various kinds of loop statements

Project 03

READING:
ALL OF CHAPTER 4
(but at Albany, we make complete programs that do real things programmed to begin in the main method!)

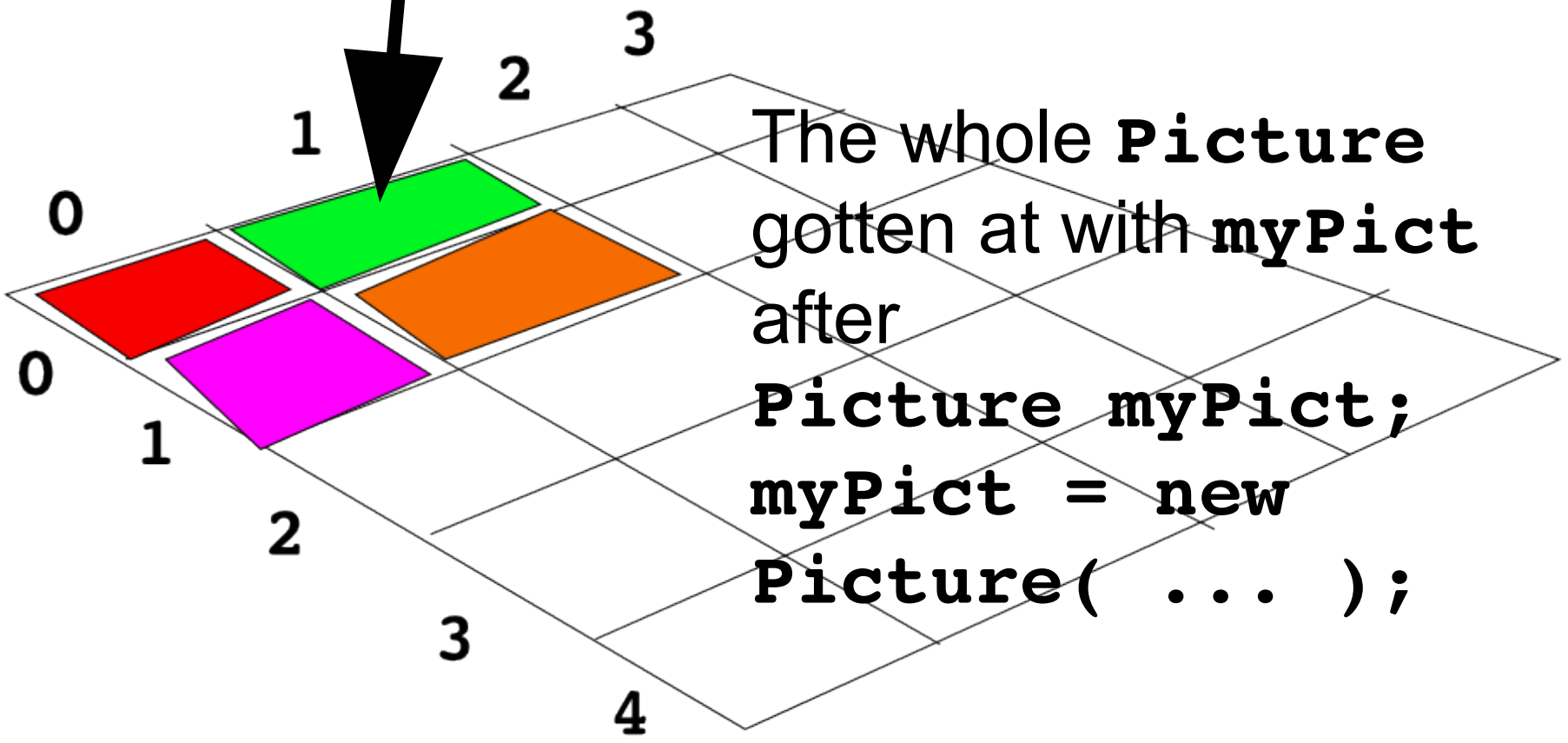
CHAPTER 5
Just pages 131, 132, 133, 134
plus today's demos



The whole **`Picture`** gotten at with **`myPict`** after

```
Picture myPict;
myPict = new
Picture( ... );
```

`Pixel` gotten at with
`myPict.getPixel( 1, 0 )`



The whole **`Picture`** gotten at with **`myPict`** after

```
Picture myPict;
myPict = new
Picture( ... );
```

Using G&E's stuff for digital Pictures

- **`Picture pict;`**//Make a variable to refer to a Picture
- **`pict = new Picture( ... );`** //Make the Picture object
- **`Pixel pix = pict.getPixel(`** SOME x-location, SOME y-location**`);`**
- **`pix.setColor( new Color(`** some red intensity, some green**`,`** some blue intensity**`) );`**
- **`pict.explore( );`** //Show a nice window for enlarging and viewing locations and RGB intensities of its Pixels one by one.