To program operations to be done in a particular order

A) You code them in the order you want them to be done.

B) You can code them in the order you think of them, and they computer will do them in the right order.

Project 03 topics

- Add AND Test public methods to Picture.java

```
void changeWhole(
        double amount )
{ /*YOU WRITE*/ }
boolean scribble(
      int xPos, int yPos,
      double scale)
{ /*YOU WRITE*/ }
```

Project 03 topics

- Add AND Test public methods to Picture.java

```
boolean ManipBoxUniformly(
   int xMin, int yMin,
   int xMax, int yMax,
   double amount )
{ /*YOU WRITE*/ }
```

Project 03 topics

- Add AND Test public methods to Picture.java

```
boolean ManipBoxPatterned(
   int xMin, int yMin,
   int xMax, int yMax,
   double amount )
{ /*YOU WRITE*/ }
```

Project 03 topics

- Add AND Test methods to Picture.java

Part1's main must TEST `changeWhole`

Part12's main must TEST `scribble`

Part123's main must TEST `ManipBoxUniformly`

Part1234's main must TEST `ManipBoxPatterned`

How we'll grade Proj03

- We will try to COMPILE your Part1234 Picture.java file (0 if it doesn't compile!!!)
- We will UNIT TEST **`changeWhole, scribble, ManipBoxUniformly`** and **`ManipBoxPatterned`**
- **UNIT TEST: We'll run a (secret) main that CALLS your 4 added methods with parameters we choose, on our Picture.**
- **We will also spot-check that your 4 versions add the methods one by one, AND TEST the added one.**

Project 03 Topics

1) Parameters to specify

- amount of change
- ONE x,y LOCATION (two integers of data LOCATE a Pixel and a spot where the Turtle can go)
- Four integers specify the RANGE OF x,y LOCATIONS that define a RECTANGLE or BOX

2) boolean return value: Extra credit for (successfully) using if operations to return whether or not the method was called with parameters so the modifications are tried ONLY within the **`Picture`**.

(regular cr. if it crashes on bad parameters).

Project 03 Topics

3) Doubly nested loops to process (part 1) ALL and (parts 3&4) just a sub-box/rectangle full of Pixels in **`this Picture`**. (LIKE IN THE GCD Visualizer of last lecture!)

4) Get at a Pixel first, 2nd, retrieve and compute with its original color to calculate a change, 3rd change the original color.

5) Same as 4) except the change must depend on the Pixel's LOCATION. (so a pattern appears!)

Project 03 Topics

6)Intro to Java arrays (GE pages 76-81)

7)The most important skill is tracing (p.99)

8)Scope (p.101 and 113)

Project 03 Topics

9)Intro to Java arrays (GE pages 76-81)

10)Digital image colors (GE pages 81-85)

13)Use fully qualified names when necessary (GE page 87 sidebar) import java.awt.Color;

13)Technology details: need to repaint after changing colors and end filenames with .jpg (page 88)

14) Know where your files are! (p. 89 DrJava's default dir. is pretty useless.)

Project 03 Topics

10)Flowchart to understand the logic of while(){ }

11)Stopping an infinite loop: Handy RESET button! (p.95)

12)Declare variable(s) tested by while BEFORE the while (p.97)

13)CAN'T define AGAIN a method w/ the SAME NAME and SAME parameter types! (p.98)

14)Loops can take a long time(p.99)

15)Memory and files are DIFFERENT(p.99)

16)Intro to Java arrays (GE pages 76-81)

13)Digital image colors (GE pages 81-85)

```
    int x; int y;

    y = yMin; /*first row to blacken*/;
    while( y <= yMax )
    {
      x = xMin;
      while( x <= xMax )
      {
        this.blackenOne( x, y );
        x =  x + 1;
      }
      y = y + 1;
    }
}
```

```
int x; int y;

y = yMin; /*first row to blacken*/;

  //SOMEHOW, value(s) of y are set!
  x = xMin;
  while( x <= xMax )
  {
    this.blackenOne( x, y );
    x =  x + 1;
  }
```

SINGLE LOOP Flowchart

```
    int x; int y;

    y = yMin; /*first row to blacken*/;
    while( y <= yMax )
    {
      x = xMin;
      while( x <= xMax )
      {
        this.blackenOne( x, y );
        x =  x + 1;
      }
      y = y + 1;
    }
}
```

DOUBLE LOOP Flowchart

BlackenRect coded with RELATIVE LOCATION xOffset, yOffset

Live coding!
Ideas:
as
`x=xMin; while(x<=xMax){ ...; x=x+1;}`
runs, x('s value) ranges
FROM xMin('s value) TO xMax
and
xOffset = x – xMin;
ranges
FROM 0 to xMax-xMin

Pick your style! Voting gives iClicker points; both choices are good.

(A)Compute xOffset by making the computer run the same while loop to set x as before, and, during each repetition, make it/he/she compute:

xOffset = x – xMin;

(B)Make the while loop code compute

xOffset from 0 to xMax-xMin FIRST,

then make it/he/she compute

x = xMin + xOffset

Make the lightness of the grey color depend on xOffset

- Live coded! Try getPixel(x,y).setRed(xOffset) etc.
- Introduce normalization..

Project 03 topics

- Add AND Test public methods to **`Picture.java`** ***(ONLY!!!)***

```
void changeWhole(
        double amount )
{ /*YOU WRITE*/ }
boolean scribble(
      int xPos, int yPos,
      double scale)
{ /*YOU WRITE*/ }
```

Picture.java

```
//Various imports of packages of classes or
//classes
import java.awt.image.BufferedImage;
// more imports.
public class Picture extends SimplePicture
{
  //G&E's writing ...
  public void changeWhole( double amount )
  {
    //YOU WRITE CODE LIKE
    //Program 10, page 114
    //Make any modification EXCEPT changeRed
  }

  //AND YOU MUST WRITE your main
  //to TEST YOUR WORK!!
```

Digression for people who finished Project03

What is inside a **`Picture`**, really?

G&E made a **`Picture`** to look like and be worked with like a 2-dim array of **`Pixel`**s.

REALLY, a **`Picture`** object is implemented by an object of the type (class)

`java.awt.image.BufferedImage`

"Buffered" means it's stored in memory (so it can be edited.)

(secret: G&E's getPixel method makes a new Pixel each time you ask to get a Pixel!)

```
  //to TEST YOUR WORK!!!

  public static void main(String[] a)
  { String filename = FileChooser.pickAFile();
    Picture pOne = new Picture(filename);
      //2 Pictures made
    Picture pTwo = new Picture(filename);
      //from ONE FILE!!!
    pOne.changeWhole( 0.2 );
    pTwo.changeWhole( 1.0);
    pOne.explore( );
       //You should see that the two copies
    pTwo.explore( );
       //have been changed differently!!
 }

}// this } is the end of the Picture class.,
```

Part2 scribble method.

```
public boolean scribble
  (int xPos, int yPos, double scale)
```

is a method to give every **`Picture`** object a new potential behavior.

What is that behavior? (no right answr)
(A) I implemented and tested it.
(B) I've a pretty good idea of it.
(C) Still clueless :(

```
public boolean scribble
  (int xPos, int yPos, double scale)
{ //The code YOU WRITE HERE
  // will (in the future!!) direct an artist
  // (computer person or robot) to
  // MAKE a Turtle object and USE it
  // (that Turtle) to draw the scribble
```

What kind of object has the following potential behaviors?

- moveTo – to move itself to a given xPos, yPos
- penUp – not draw until further notice
- forward – move forward a given distance

(A) A **`Turtle`** kind of object
(B) A **`Picture`** kind of object
(C) A **`World`** kind of object
(D) A **`String`** kind of object

```
public boolean scribble( int xPos, int yPos, double scale )
{
  Turtle tu = new Turtle( this );
  //this refers to the Picture on, for or with scribble() is called.
  //A GE Turtle can live in a Picture as well as a World object.
  tu.penUp( );
  tu.moveTo( xPos, yPos );
  //Purpose of the last 2 statements:
  //They make the Turtle start at location (xPos, yPos) without
  //making any marks on the Picture.
  tu.penDown( );
  tu.forward( (int) ( scale ) );
  //Your code to program making the scribble goes here.
  //Must be more complex than this:Must make 3 or more strokes!
  //It CAN use tu.moveTo(), not just tu.forward().
  //Finally, the size of the scribble MUST be controlled linearly
  //by the value of scale.
  return true;
  //If you do the extra credit version, return may be
  //coded in other places, and the value it returns must be false
  //when the scribbling would get messed up because the Turtle would
  //be commanded to move outside the Picture. You'll need if
  //statements and carefully thought out math to do that.
  //Trying to move outside is fine for regular credit.
}
```

```
public boolean scribble
    ( int xPos, int yPos, double scale )
//Make the method.
{


}
```

```
public boolean scribble
    ( int xPos, int yPos, double scale )
//Make the method.
{
  //The code below DIRECTS the artist
  //(computer person) to MAKE A Turtle.

  Turtle tu = new Turtle( this );

  //this refers to the Picture on, for or with
  // scribble() is called.
 .....
}
```

```
public boolean scribble
    ( int xPos, int yPos, double scale )
{
  Turtle tu = new Turtle( this );
  tu.penUp( );
  tu.moveTo( xPos, yPos );
  //Purpose of the last 2 statements:
  //Make the Turtle start at location
  // (xPos, yPos) without
  //making any marks on the Picture.
  .....
}
```

```
public boolean scribble
    ( int xPos, int yPos, double scale )
{
  Turtle tu = new Turtle( this );
  tu.penUp( );
  tu.moveTo( xPos, yPos );
  tu.penDown( );
  tu.forward( (int) ( scale ) );
          //or
  tu.forward( (int) ( scale*(34.76)) );
  //Your code to program making the
  //   scribble goes here.  For credit, it
  // MUST be more interesting than 1 line!
}
```

```
public boolean scribble( int xPos, int yPos, double scale )
{
  Turtle tu = new Turtle( this );
  //this refers to the Picture on, for or with scribble() is called.
  //A GE Turtle can live in a Picture as well as a World object.
  tu.penUp( );
  tu.moveTo( xPos, yPos );
  //Purpose of the last 2 statements:
  //They make the Turtle start at location (xPos, yPos) without
  //making any marks on the Picture.
  tu.penDown( );
  tu.forward( (int) ( scale ) );
  //Your code to program making the scribble goes here.
  //Must be more complex than this:Must make 3 or more strokes!
  //It CAN use tu.moveTo(), not just tu.forward().
  //Finally, the size of the scribble MUST be controlled linearly
  //by the value of scale.
  return true;
  //If you do the extra credit version, return may be
  //coded in other places, and the value it returns must be false
  //when the scribbling would get messed up because the Turtle would
  //be commanded to move outside the Picture. You'll need if
  //statements and carefully thought out math to do that.
  //Trying to move outside is fine for regular credit.
}
```

Proj03 Requirements

- Part 1: amount parameter MUST affect the AMOUNT OF CHANGE! AND your main

MUST test it with at least 2 different amounts!

- Part 2: the xPos, yPos and scale params MUST affect the LOCATION and the SIZE

MUST test it with at least 2 different locations AND 2 different scales (sizes)

HOW to TEST??

Put and leave your testing code in your main method, so the TAs can see what testing you did!

Proj03 Requirements

- Part 3: xMin, yMin, xMax, yMax params MUST determine the exact “Box” to change.

  amount MUST affect the amount of change.

  MUST test with at least 2 different amounts

  Just blackening or coloring a box is NOT ACCEPTABLE: Some of the original color intensities must affect the result.

Proj03 Requirements

- Part 4: Same as part 3 with ONE MORE REQUIREMENT:

  The LOCATION of each Pixel (preferably normalized to the Box) MUST affect how its color is changed.

so AS A RESULT: Some sort of pattern (grid, dots, lines, wiggles, circles, random, ripples, use your imagination is superposed on the image.)

Remember: amount must affect the change too, and you must test with at least 2 amounts!

To program operations to be done in a particular order

A) You code them in the order you want them to be done.

B) You can code them in the order you think of them, and they computer will do them in the right order.

Project 03 topics

- Add AND Test public methods to Picture.java

```
void changeWhole(
        double amount )
{ /*YOU WRITE*/ }
boolean scribble(
      int xPos, int yPos,
      double scale)
{ /*YOU WRITE*/ }
```

Project 03 topics

- Add AND Test public methods to Picture.java

```
boolean ManipBoxUniformly(
   int xMin, int yMin,
   int xMax, int yMax,
   double amount )
{ /*YOU WRITE*/ }
```

Project 03 topics

- Add AND Test public methods to Picture.java

```
boolean ManipBoxPatterned(
   int xMin, int yMin,
   int xMax, int yMax,
   double amount )
{ /*YOU WRITE*/ }
```

Project 03 topics

- Add AND Test methods to Picture.java

Part1's main must TEST `changeWhole`

Part12's main must TEST `scribble`

Part123's main must TEST `ManipBoxUniformly`

Part1234's main must TEST `ManipBoxPatterned`

How we'll grade Proj03

- We will try to COMPILE your Part1234 Picture.java file (0 if it doesn't compile!!!)
- We will UNIT TEST **`changeWhole, scribble, ManipBoxUniformly`** and **`ManipBoxPatterned`**
- **UNIT TEST: We'll run a (secret) main that CALLS your 4 added methods with parameters we choose, on our Picture.**
- **We will also spot-check that your 4 versions add the methods one by one, AND TEST the added one.**

Project 03 Topics

1) Parameters to specify
   - amount of change
   - ONE x,y LOCATION (two integers of data LOCATE a Pixel and a spot where the Turtle can go)
   - Four integers specify the RANGE OF x,y LOCATIONS that define a RECTANGLE or BOX

2) boolean return value: Extra credit for (successfully) using if operations to return whether or not the method was called with parameters so the modifications are tried ONLY within the **`Picture`**.

   (regular cr. if it crashes on bad parameters).

Project 03 Topics

3) Doubly nested loops to process (part 1) ALL and (parts 3&4) just a sub-box/rectangle full of Pixels in **`this Picture`**. (LIKE IN THE GCD Visualizer of last lecture!)

4)Get at a Pixel first, 2nd, retrieve and compute with its original color to calculate a change, 3rd change the original color.

5)Same as 4) except the change must depend on the Pixel's LOCATION. (so a pattern appears!)

Project 03 Topics

6)Intro to Java arrays (GE pages 76-81)

7)The most important skill is tracing (p.99)

8)Scope (p.101 and 113)

Project 03 Topics

9)Intro to Java arrays (GE pages 76-81)

10)Digital image colors (GE pages 81-85)

13)Use fully qualified names when necessary (GE page 87 sidebar) import java.awt.Color;

13)Technology details: need to repaint after changing colors and end filenames with .jpg (page 88)

14) Know where your files are! (p. 89 DrJava's default dir. is pretty useless.)

Project 03 Topics

10)Flowchart to understand the logic of while(){ }

11)Stopping an infinite loop: Handy RESET button! (p.95)

12)Declare variable(s) tested by while BEFORE the while (p.97)

13)CAN'T define AGAIN a method w/ the SAME NAME and SAME parameter types! (p.98)

14)Loops can take a long time(p.99)

15)Memory and files are DIFFERENT(p.99)

16)Intro to Java arrays (GE pages 76-81)

13)Digital image colors (GE pages 81-85)

```
    int x; int y;

    y = yMin; /*first row to blacken*/;
    while( y <= yMax )
    {
      x = xMin;
      while( x <= xMax )
      {
        this.blackenOne( x, y );
        x =  x + 1;
      }
      y = y + 1;
    }
  }
```

```
int x; int y;

y = yMin; /*first row to blacken*/;

  //SOMEHOW, value(s) of y are set!
  x = xMin;
  while( x <= xMax )
  {
    this.blackenOne( x, y );
    x =  x + 1;
  }
```

SINGLE LOOP Flowchart

```
  int x; int y;

  y = yMin; /*first row to blacken*/;
  while( y <= yMax )
  {
    x = xMin;
    while( x <= xMax )
    {
      this.blackenOne( x, y );
      x =  x + 1;
    }
    y = y + 1;
  }
}
```

DOUBLE LOOP Flowchart

BlackenRect coded with RELATIVE LOCATION xOffset, yOffset

Live coding!
Ideas:
as
`x=xMin; while(x<=xMax){ ...; x=x+1;}`
runs, x('s value) ranges
FROM xMin('s value) TO xMax
and
xOffset = x – xMin;
ranges
FROM 0 to xMax-xMin

Pick your style! Voting gives iClicker points; both choices are good.

(A)Compute xOffset by making the computer run the same while loop to set x as before, and, during each repetition, make it/he/she compute:

xOffset = x – xMin;

(B)Make the while loop code compute

xOffset from 0 to xMax-xMin FIRST,

then make it/he/she compute

x = xMin + xOffset

Make the lightness of the grey color depend on xOffset

- Live coded! Try getPixel(x,y).setRed(xOffset) etc.
- Introduce normalization..

Project 03 topics

- Add AND Test public methods to **`Picture.java`** ***(ONLY!!!)***

```
void changeWhole(
        double amount )
{ /*YOU WRITE*/ }
boolean scribble(
      int xPos, int yPos,
      double scale)
{ /*YOU WRITE*/ }
```

Picture.java

```
//Various imports of packages of classes or
//classes
import java.awt.image.BufferedImage;
// more imports.
public class Picture extends SimplePicture
{
  //G&E's writing ...
 public void changeWhole( double amount )
 {
   //YOU WRITE CODE LIKE
   //Program 10, page 114
   //Make any modification EXCEPT changeRed
 }

 //AND YOU MUST WRITE your main
 //to TEST YOUR WORK!!
```

Digression for people who finished Project03

What is inside a **`Picture`**, really?
G&E made a **`Picture`** to look like and be worked with like a 2-dim array of **`Pixel`**s.
REALLY, a **`Picture`** object is implemented by an object of the type (class)

`java.awt.image.BufferedImage`

“Buffered” means it's stored in memory (so it can be edited.)

(secret: G&E's getPixel method makes a new Pixel each time you ask to get a Pixel!)

```
 //to TEST YOUR WORK!!!

 public static void main(String[] a)
 { String filename = FileChooser.pickAFile();
   Picture pOne = new Picture(filename);
     //2 Pictures made
   Picture pTwo = new Picture(filename);
     //from ONE FILE!!!
   pOne.changeWhole( 0.2 );
   pTwo.changeWhole( 1.0);
   pOne.explore( );
      //You should see that the two copies
   pTwo.explore( );
      //have been changed differently!!
 }

}// this } is the end of the Picture class.,
```

Part2 scribble method.

```
public boolean scribble
 (int xPos, int yPos, double scale)
```

is a method to give every **`Picture`** object a new potential behavior.

What is that behavior? (no right answr)
(A) I implemented and tested it.
(B) I've a pretty good idea of it.
(C) Still clueless :(

```
public boolean scribble
  (int xPos, int yPos, double scale)
{ //The code YOU WRITE HERE
  // will (in the future!!) direct an artist
  // (computer person or robot) to
  // MAKE a Turtle object and USE it
  // (that Turtle) to draw the scribble
```

What kind of object has the following potential behaviors?

- moveTo – to move itself to a given xPos, yPos
- penUp – not draw until further notice
- forward – move forward a given distance

(A) A **`Turtle`** kind of object
(B) A **`Picture`** kind of object
(C) A **`World`** kind of object
(D) A **`String`** kind of object

```
public boolean scribble( int xPos, int yPos, double scale )
{
  Turtle tu = new Turtle( this );
  //this refers to the Picture on, for or with scribble() is called.
  //A GE Turtle can live in a Picture as well as a World object.
  tu.penUp( );
  tu.moveTo( xPos, yPos );
  //Purpose of the last 2 statements:
  //They make the Turtle start at location (xPos, yPos) without
  //making any marks on the Picture.
  tu.penDown( );
  tu.forward( (int) ( scale ) );
  //Your code to program making the scribble goes here.
  //Must be more complex than this:Must make 3 or more strokes!
  //It CAN use tu.moveTo(), not just tu.forward().
  //Finally, the size of the scribble MUST be controlled linearly
  //by the value of scale.
  return true;
  //If you do the extra credit version, return may be
  //coded in other places, and the value it returns must be false
  //when the scribbling would get messed up because the Turtle would
  //be commanded to move outside the Picture. You'll need if
  //statements and carefully thought out math to do that.
  //Trying to move outside is fine for regular credit.
}
```

```
public boolean scribble
    ( int xPos, int yPos, double scale )
//Make the method.
{


}
```

```
public boolean scribble
    ( int xPos, int yPos, double scale )
//Make the method.
{
  //The code below DIRECTS the artist
  //(computer person) to MAKE A Turtle.

  Turtle tu = new Turtle( this );

  //this refers to the Picture on, for or with
  // scribble() is called.
 .....
}
```

```
public boolean scribble
    ( int xPos, int yPos, double scale )
{
  Turtle tu = new Turtle( this );
  tu.penUp( );
  tu.moveTo( xPos, yPos );
  //Purpose of the last 2 statements:
  //Make the Turtle start at location
  // (xPos, yPos) without
  //making any marks on the Picture.
 .....
}
```

```
public boolean scribble
    ( int xPos, int yPos, double scale )
{
  Turtle tu = new Turtle( this );
  tu.penUp( );
  tu.moveTo( xPos, yPos );
  tu.penDown( );
  tu.forward( (int) ( scale ) );
          //or
  tu.forward( (int) ( scale*(34.76)) );
  //Your code to program making the
  //   scribble goes here.  For credit, it
  // MUST be more interesting than 1 line!
}
```

```
public boolean scribble( int xPos, int yPos, double scale )
{
  Turtle tu = new Turtle( this );
  //this refers to the Picture on, for or with scribble() is called.
  //A GE Turtle can live in a Picture as well as a World object.
  tu.penUp( );
  tu.moveTo( xPos, yPos );
  //Purpose of the last 2 statements:
  //They make the Turtle start at location (xPos, yPos) without
  //making any marks on the Picture.
  tu.penDown( );
  tu.forward( (int) ( scale ) );
  //Your code to program making the scribble goes here.
  //Must be more complex than this:Must make 3 or more strokes!
  //It CAN use tu.moveTo(), not just tu.forward().
  //Finally, the size of the scribble MUST be controlled linearly
  //by the value of scale.
  return true;
  //If you do the extra credit version, return may be
  //coded in other places, and the value it returns must be false
  //when the scribbling would get messed up because the Turtle would
  //be commanded to move outside the Picture. You'll need if
  //statements and carefully thought out math to do that.
  //Trying to move outside is fine for regular credit.
}
```

Proj03 Requirements

- Part 1: amount parameter MUST affect the AMOUNT OF CHANGE! AND your main

MUST test it with at least 2 different amounts!

- Part 2: the xPos, yPos and scale params MUST affect the LOCATION and the SIZE

MUST test it with at least 2 different locations AND 2 different scales (sizes)

HOW to TEST??

Put and leave your testing code in your main method, so the TAs can see what testing you did!

Proj03 Requirements

- Part 3: xMin, yMin, xMax, yMax params MUST determine the exact “Box” to change.

  amount MUST affect the amount of change.

  MUST test with at least 2 different amounts

  Just blackening or coloring a box is NOT ACCEPTABLE: Some of the original color intensities must affect the result.

Proj03 Requirements

- Part 4: Same as part 3 with ONE MORE REQUIREMENT:

  The LOCATION of each Pixel (preferably normalized to the Box) MUST affect how its color is changed.

so AS A RESULT: Some sort of pattern (grid, dots, lines, wiggles, circles, random, ripples, use your imagination is superposed on the image.)

Remember: amount must affect the change too, and you must test with at least 2 amounts!