Relativity
(classical computer programming)
Sequential processing you have programmed into your applications' main methods:

Making (instantiating) objects:

```java
EditablePicture bigP =
    new EditablePicture(
        FileChooser.PickAFile()
    );

Picture littleP =
    new Picture(
        FileChooser.PickAFile()
    );
```
Sequential processing you have programmed into your applications' main methods:

Calling methods on those objects:

```java
bigP.copyIn( 20, 100, littleP);
//Purpose: Make the code defined in
//copyIn copy the little Picture
//into the big Picture.

bigP.explore();
//Purpose: Make you see the result
//of the copyIn operation.
```
iClicker: Interchange 2 lines of code. What would happen?

```java
main(..) { ...
    //as before, make bigP and
    // littleP ONLY.
    bigP.explore();
    bigP.copyIn( 20, 100, litP);
}
```

(A) You see nothing.
(B) You see the original big picture.
(C) You see the big picture with the little picture copied into it.
(D) You see the little picture.
(E) You see the big picture twice: 1. The original
    2. Original with the little picture copied into it.
The programs below do different things!

```java
main(..) { ... //as before
    //make bigP and littleP ONLY.
    bigP.explore();
    bigP.copyIn(20, 100, litP);
}

main(..) { ... //as before
    //make bigP and littleP ONLY.
    bigP.copyIn(20, 100, litP);
    bigP.explore();
}
```
Which version can you use to TEST your Lab8 copyIn method?

A

```
main(..) { ...//as before
    //make bigP and littleP ONLY.
    bigP.explore();
    bigP.copyIn( 20, 100, litP);
}
```

B

```
main(..) { ...//as before
    //make bigP and littleP ONLY.
    bigP.copyIn( 20, 100, litP);
    bigP.explore();
}
```
DEFINING METHODS
Sequential processing you have programmed into METHOD DEFINITIONS that EXTEND 10-year old preprogrammed behaviors

```java
class EditablePicture extends Picture {
    public void copyIn(int originX,
                        int originY,
                        Picture source )
    {
        //You coded in Lab8
        return;  //Just return to caller
        //with no return value.
    }
}
```
Methods that MODIFY, COPY IN or COPY OUT image information of Pixels in Pictures

DEFINING THEM: USING parameter variables, LOOPS and CONDITIONALS

CALLING THEM: Using parameter values
Lab 507 First Half

Before

After

Second Half

New Paintable Picture
Lab 607 First Half

Before  

AFTER

Second Half

origin X, origin Y
Calculations done inside copy in to locate origin X + xRel
origin Y + yRel
copyIn solution

- copyIn method programs doing stuff ON (this) EditablePicture object

- Parameter variables:
  - originX
  - originY
  - source (Picture)

- It uses two return values from
  
  \[ \text{source.getWidth()} \]
  \[ \text{source.getHeight()} \]

- copyIn's loop control variables are
  - xRel
  - yRel
public class EditablePicture extends Picture
{
    public void copyIn(int originX, int originY, Picture source)
    {
        for(int yRel=0; yRel<source.getHeight(); yRel=yRel+1)
        {
            for(int xRel=0; xRel<source.getWidth(); xRel=xRel+1)
            {
                Pixel srcP = source.getPixel(xRel,yRel);
                this.getPixel(
                        originX+xRel,
                        originY+yRel)
                    .setColor(srcP.getColor());
            }
        }
    }
}
Other purposes for xRel and yRel are:

(1) To track a Pixel location ACTUALLY IN the little Picture.
(2) To help track and compute a Pixel location in the big Picture RELATIVE TO (originX, originY)
public class EditablePicture extends Picture {
    public void copyIn(int originX, int originY, Picture source) {
        for(int yRel=0; yRel<source.getHeight(); yRel=yRel+1) {
            for(int xRel=0; xRel<source.getWidth(); xRel=xRel+1) {
                Pixel srcP = source.getPixel(xRel,yRel);
                this.getPixel(
                        originX+xRel,
                        originY+yRel)
                        .setColor(
                                srcP.getColor());
            }
        }
    }
}
Homework practice quiz: In what order (1st, 2nd, 3rd, 4th, 5th) does the computer do these 5 things?

1. Get a Pixel from this, big Picture?
2. Get a Color from a Pixel of the source, little Picture?
3. Compute originX+xRel and originY+yRel?
4. Get a Pixel from the source, little Picture?
5. Set the big Picture's Pixel to a usually different Color?
public class EditablePicture extends Picture {
    public void copyIn(int originX, int originY, Picture source) {
        for(int yRel=0; yRel<source.getHeight(); yRel=yRel+1) {
            for(int xRel=0; xRel<source.getWidth(); xRel=xRel+1) {
                Pixel srcP = source.getPixel(xRel, yRel);
                this.getPixel(
                        originX+xRel,
                        originY+yRel)
                    .setColor(
                            srcP.getColor());
            }
        }
    }
}
public class EditablePicture extends Picture {
    public void copyIn(int originX, int originY, Picture source) {
        for (int yRel = 0; yRel < source.getHeight(); yRel++) {
            for (int xRel = 0; xRel < source.getWidth(); xRel++) {
                Pixel srcP = source.getPixel(xRel, yRel);
                this.getPixel(originX + xRel, originY + yRel).setColor(srcP.getColor());
            }
        }
    }
}
A) yRel is used to control a loop.

B) yRel is used to help locate a Pixel within the little source Picture absolutely (not relatively).

C) yRel is used to help locate a Pixel within this big Picture relative to (originX, originY).
public class EditablePicture extends Picture {
    public void copyIn(int originX, int originY, Picture source) {
        for (int yRel = 0; yRel < source.getHeight(); yRel = yRel + 1) {
            for (int xRel = 0; xRel < source.getWidth(); xRel = xRel + 1) {
                Pixel srcP = source.getPixel(xRel, yRel);
                this.getPixel(originX + xRel, originY + yRel)
                    .setColor(srcP.getColor());
            }
        }
    }
}
iClicker

A) yRel is used to control a loop.

B) yRel is used to help locate a Pixel within the little source Picture absolutely (not relatively).

C) yRel is used to help locate a Pixel within this big Picture relative to (originX,originY).
public class EditablePicture extends Picture {
    public void copyIn(int originX, int originY, Picture source) {
        for(int yRel=0; yRel<source.getHeight(); yRel=yRel+1) {
            for(int xRel=0; xRel<source.getWidth(); xRel=xRel+1) {
                Pixel srcP = source.getPixel(xRel, yRel);
                this.getPixel(originX+xRel, originY+yRel).setColor(srcP.getColor());
            }
        }
    }
}
iClicker

A) yRel is used to control a loop.

B) yRel is used to help locate a Pixel within the little source Picture absolutely (not relatively).

C) yRel is used to help locate a Pixel within this big Picture relative to (originX,originY).
Purposes for xRel and yRel are:
(1) To track an ABSOLUTE Pixel location ACTUALLY IN the little Picture.

(2) To help track and compute a Pixel location in the big Picture RELATIVE TO (originX, originY)
Calculations done inside copy in to locate
origin X + xRel
origin Y + yRel
You can program anything you want (like in Alice's restaurant)

I programmed copyOut to STOP about 1/3 through its original task.

HOW? I made another variable for COUNTING each Pixel when it is re-colored. Also code to compute HOW MANY Pixels in the little Picture.
public void copyIn(int originX, int originY, Picture source) {
    int nPixCopied = 0;
    for(int yRel=0; yRel<source.getHeight(); yRel=yRel+1) {
        for(int xRel=0; xRel<source.getWidth(); xRel=xRel+1) {
            Pixel srcP = source.getPixel(xRel, yRel);
            this.getPixel(
                    originX+xRel,
                    originY+yRel).setColor(srcP.getColor());
            nPixCopied = nPixCopied+1;
            if(nPixCopied == (int)(0.3251*source.getHeight()*source.getWidth())) {
                return;
            }
        }
    }
}
About 0.3 of the full rows plus one partial row.
Sudoku too hard for now!

Sudoku BOARD computing support CURRENT LAB 10!
Sudoku board

- 81 squares or boxes for the maker and puzzle solver to write in numbers 1, 2, ... 9.
- 9 rows of 9 boxes each. In other words, 9 columns of 9 boxes each. Just like a digital Picture.
- 3 rows of 3 BLOCKS each—A BLOCK is a 3x3 grid of boxes!
- In other words, 3 columns of 3 BLOCKS each. (Confusing!)
3 kinds of Sudoku rules (too hard for now)

- **Row rule:** Each of the 9 boxes in a row must contain exactly one of each of the 1-9 numbers.

- **Column rule:** Each of the 9 boxes in a column must contain exactly one of each of the 1-9 numbers.

- **BLOCK rule:** Each of the 9 boxes in a (3x3) block must contain exactly one of each of the 1-9 numbers.

**THERE ARE 27 rules altogether:** 9 different row rules, 9 different column rules and 9 different block rules!
The lecture now and this week's lab 10 is about LOCATING board boxes or squares within rows within columns within blocks.
READ THE JAVADOC

Javadoc web site for Sudoku and other classes
Live coding example 1: Write a 7 into EACH of the 9 upper left boxes of each of the 9 BLOCKS

```
7 0 0 7 0 0 7 0 0
0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0

7 0 0 7 0 0 7 0 0
0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0

7 0 0 7 0 0 7 0 0
0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0

7 0 0 7 0 0 7 0 0
0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0
```
public void sevensInULsOfBlocks()
{
    for( int rb = 0; rb < 3; rb = rb + 1)
    {
        for( int cb = 0; cb < 3; cb = cb + 1)
        {
            this.board[3*cb][3*rb] = 7;
        }
    }
    return ;
}
Live coding example 2: Write 1-9 into EACH of the 9 upper left boxes of each of the 9 BLOCKS

<table>
<thead>
<tr>
<th>1 0 0</th>
<th>2 0 0</th>
<th>3 0 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 0</td>
<td>0 0 0</td>
<td>0 0 0</td>
</tr>
<tr>
<td>0 0 0</td>
<td>0 0 0</td>
<td>0 0 0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>4 0 0</th>
<th>5 0 0</th>
<th>6 0 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 0</td>
<td>0 0 0</td>
<td>0 0 0</td>
</tr>
<tr>
<td>0 0 0</td>
<td>0 0 0</td>
<td>0 0 0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>7 0 0</th>
<th>8 0 0</th>
<th>9 0 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 0</td>
<td>0 0 0</td>
<td>0 0 0</td>
</tr>
<tr>
<td>0 0 0</td>
<td>0 0 0</td>
<td>0 0 0</td>
</tr>
</tbody>
</table>
public void countsInULsOfBlocks()
{
    int blkCounter = 1;
    for( int rb = 0; rb < 3; rb = rb + 1)
    {
        for( int cb = 0; cb < 3; cb = cb + 1)
        {
            this.board[3*cb][3*rb] = blkCounter;
            blkCounter = blkCounter + 1;
        }
    }
    return ;
}
Live coding example 3: Write 1-9 into EACH of the 9 second-row, third-column boxes RELATIVE TO EACH of the 9 BLOCKS

<p>| | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>4</td>
<td>0</td>
<td>0</td>
<td>5</td>
<td>0</td>
<td>0</td>
<td>6</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>7</td>
<td>0</td>
<td>0</td>
<td>8</td>
<td>0</td>
<td>0</td>
<td>9</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
public void countsIn23 RelativesOfBlocks()
{
    int blkCounter = 1;
    for( int rb = 0; rb < 3; rb = rb + 1)
    {
        for( int cb = 0; cb < 3; cb = cb + 1)
        {
            this.board[3*cb+2][3*rb+1] = blkCounter;
            blkCounter = blkCounter + 1;
        }
    }
    return ;
}
Back to copyIn
I programmed copyOut to TEST if 
(originX+yRel,originY+yRel) is a 
valid Pixel array location BEFORE 
trying to get a Pixel from a location 
where there might be NO Pixel. 
Purpose: Avoid a CRASH.

(You can program 
anything you want 
like in Alice's restaurant.)
This photo of Theresa's Stockbridge Cafe is courtesy of TripAdvisor
public void copyIn(int originX, int originY, Picture source) {
    for(int yRel=0; yRel<source.getHeight(); yRel=yRel+1) {
        for(int xRel=0; xRel<source.getWidth(); xRel=xRel+1) {
            Pixel srcP = source.getPixel(xRel, yRel);
            if( originX+xRel < this.getWidth() &&
                &&
                originY+yRel < this.getHeight() ) {
                this.getPixel(
                        originX+xRel,
                        originY+yRel).setColor(srcP.getColor());
            }
        }
    }
}
Result

The copied in (small) Picture is “cropped” on the left and/or on the bottom if it would otherwise go outside this (big) Picture.

It led me to think of an INNOVATION!

The SAME if conditional statement technique can be used to crop the little Picture on the right and/or top also!
public void copyIn(int originX, int originY, Picture source) 
{
    for(int yRel=0; yRel<source.getHeight(); yRel=yRel+1) 
    { 
        for(int xRel=0; xRel<source.getWidth(); xRel=xRel+1) 
        { 
            Pixel srcP = source.getPixel(xRel,yRel); 
            if( originX+xRel < this.getWidth() \&\& originX+xRel\geq0 
                \&\& originY+yRel < this.getHeight() \&\& originY+yRel\geq0 ) 
            { 
                this.getPixel(
                    originX+xRel,
                    originY+yRel).setColor(srcP.getColor()); 
            } 
        } 
    }
}