Lecture 14-pre-midterm
Midterm next lecture!
Right here. 80+10 overtime min.
Closed book/note/etc except for 1
std size sheet of paper
70% multiple choice, Scantron

30% WRITING JAVA CODE
iclicker: What best describes the “sequential processing” concept?

A). The coded instructions you wrote are performed by the computer in exactly the sequence you wrote (modified by calls, whiles, fors, ifs and returns) even if the results are wrong.

B). After you pick and code useful instructions, the computer will figure out the sequence and performs that sequence to always make the results right.
iClicker: Suppose the very last time you clicked *Compile* for your program, there were syntax errors. When you then click *Run*, the computer will follow some of that program's instructions:

A) Always
B) Sometimes
C) Never
Three kinds of error and their times

• Compile time errors
  – Syntax errors are detected at this time, and the Java system will not let a program with syntax errors run.

• Runtime errors, also called crashes
  – A program with no syntax errors is successfully compiled, but when it takes control of the computer, it commands the computer to do some impossible operation, for example (1) divide by zero or (2) read or copy data into an array element or Pixel that doesn't exist because the number(s) that locate it is (or are) out of range.

• Logic errors (also occur at run time): No failure is detected but the programmer programmed the computer to do things he or she didn't want. (Example: A banker programming add for the withdrawal operation.)
maybe where is the code that defines the setColor method?

(it's in the book Classes)
Sequential processing you have programmed into your applications' main methods:

Making (instantiating) objects:
World wref = new World( );
ArtisticTurtle tref = new ArtisticTurtle( wref );

Calling methods on those objects:
    tref.forward( 100 );
    tref.turn(90);
    tref.forward( 50 );
DEFINING METHODS
Sequential processing you have programmed into METHOD DEFINITIONS that EXTEND 10-year old preprogrammed behaviors

class ArtisticTurtle extends Turtle
{
    void club( int size )
    {
        this.forward( size*10 );
        this.turn( 75 );
        this.forward( size*2);
        return;
    }
}
CALLING METHODS
Sequential processing you have programmed into main (other places too) to CALL a METHOD you have ALREADY DEFINED.

class App
{
  public static void main(..)
  {
    World wref = new World();
    ArtisticTurtle tref=new ArtisticTurtle(wref);
    tref.club( 40 );
    tref.club( 30 );
  }
}
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 Text
Declare variable

\[ \text{Picture pictR} = \text{new Picture} \]

\[ \text{taken1, taken2} \]

Data type: Variables "Name of a JPG file"

The pictR

whole refers to picture the whole picture

So \[ \text{int count = pictR.getWidth()}; \]

returns the width widths of the whole picture

Pixel pXR:

\[ \text{pXR = pictR.getPixel(x, y)} \]

\[ \text{get one pixel location} \]

\[ \text{from} \]

\[ \text{get one pixel location} \]

\[ \text{entire picture color i + Purple} \]

\[ \text{pXR.setColor(new InvariantColor(175, 0, 15))}; \]
One Picture, \( \mathbf{X} \Omega \)
Many (different) Pixels, \( \mathbf{Y} \)
It has \( \mathbf{W} \) with different Pixels to one Pixel
Declare variable

Picture pictR = new Picture

Data type + Variable's name of a jpg file

Declaration the pictR

whole refers to picture the whole picture

so int count = pictR.getWidth();

gets the width width is of the whole picture

Pixel pXR;
pXR = pictR.getPixel((x, y))

get one pixel location

entire picture at color it + purple

pXR.setColor(new java.awt.Color(175, 0, 125));
public void makeAllWhite()
{
    int x;
    x = 0;
    while( x < this.getWidth() )
    {
        //whiten a whole column..
        int y;
        y = 0;
        while( y < this.getHeight() )
        {
            Pixel pRef;
            pRef = this.getPixel(x,y);
            pRef.setColor(new java.awt.Color(255,255,255));
            y = y + 1;
        }
        x = x + 1;
    }
    return ;
}
```java
public void makeRedWhite()
{
    int x;
    x = 0;
    while( x < this.getWidth() )
    {
        // whiten a whole column..
        int y;
        y = 0;
        while( y < this.getHeight() )
        {
            Pixel pRef;
            pRef = this.getPixel(x,y);
            if ( pRef.getRed() > (pRef.getGreen() + pRef.getBlue()) )
            {
                pRef.setColor(new java.awt.Color(255,255,255));
            }
            y = y + 1;
        }
        x = x + 1;
    }
    return ;
}
```
my fashion goal...
public void makeRedCheckered()
{
    int x;
    x = 0;
    while( x < this.getWidth() )
    {
        //whiten a whole column..
        int y;
        y = 0;
        while( y < this.getHeight() )
        {
            Pixel pRef;
            pRef = this.getPixel(x,y);
            if ( pRef.getRed() > (pRef.getGreen() + pRef.getBlue()) )
            {
                if( x,y should be made white)
                {
                    pRef.setColor(new java.awt.Color(255,255,255));
                }
                else
                {
                    pRef.setColor(new java.awt.Color(0,0,0));
                }
            }
            y = y + 1;
        }
        x = x + 1;
    }
    return ;
}
Choose an integer \( e \) such that 
\[1 < e < \phi(n) \text{ and } \gcd(e, \phi(n)) = 1;\]
i.e., \( e \) and \( \phi(n) \) are coprime.

* \( e \) is released as the public key exponent.
GCD

GCD calculation on paper sheet.
public class GCDApp {

    public static void main(String[] args) {
        java.util.Scanner sc = new java.util.Scanner(System.in);
        System.out.println("Input two numbers to test for GCD being 1:");
        int inValOne;
        int inValTwo;
        inValOne = sc.nextInt();
        inValTwo = sc.nextInt();
        MathMasher mm = new MathMasher();
        int result = mm.gcd(inValOne, inValTwo);
        if (result == 1) {
            System.out.println("Hooray, the gcd is 1! ");
        } else {
            System.out.println("Sorry, the gcd is " + result);
            System.out.println("Maybe try again.");
        }
    }
}
public class MathMasher extends Object {
    public int gcd( int redVar, int greenVar ) {
        while( redVar > 0 && greenVar > 0 ) //Repeat the subtract-like-crazy algorithm.
        {
            if( redVar > greenVar ) {
                redVar = redVar - greenVar;
                System.out.println( redVar );
            } else {
                greenVar = greenVar - redVar;
                System.out.println( greenVar );
            }
        }
        if( redVar != 0 ) //Determine which value to return by choosing the non-zero one.
        {
            return redVar;
        } else {
            return greenVar;
        }
    }
}
Application to use our gcd computing method.
Our gcd computing method demonstrates the electronic computer doing the computing on variables that you did by hand.
Many fewer decision and re-writing steps are done when repeated subtraction is replaced by divide-and-return-remainder.
public class MathMasher extends Object
{
    public int gcd( int redVar, int greenVar )
    {
        while( redVar > 0 && greenVar > 0 ) //Repeat the subtract-like-crazy algorithm.
        {
            if( redVar > greenVar )
            {
                redVar = redVar % greenVar;
                System.out.println( redVar );
            } else
            {
                greenVar = greenVar % redVar;
                System.out.println( greenVar );
            }
        }
        if( redVar != 0 )  //Determine which value to return by choosing the non-zero one.
        {
            return redVar;
        } else
        {
            return greenVar;
        }
    }
}

It's much faster to find the remainder after division than to find it after repeated subtraction!
public void makeRedCheckered()
{
    int x;
    x = 0;
    while( x < this.getWidth() )
    {
        //whiten a whole column..
        int y;
        y = 0;
        while( y < this.getHeight() )
        {
            Pixel pRef;
            pRef = this.getPixel(x,y);
            if ( pRef.getRed() > (pRef.getGreen() + pRef.getBlue()) )
            {
                if( \textit{x,y should be made white} )
                {
                    pRef.setColor(new java.awt.Color(255,255,255));
                }
                else
                {
                    pRef.setColor(new java.awt.Color(0,0,0));
                }
            }
            y = y + 1;
        }
        x = x + 1;
    }
    return ;
}
Make each green pixel white if \( \lfloor x/50 \rfloor \leq 25 \) and \( \lfloor y/50 \rfloor < 25 \) or \( \lfloor x/50 \rfloor > 25 \) and \( \lfloor y/50 \rfloor \geq 25 \). Else, color it black.
How to decide black or white to make a checkboard

- Decide the width and height of the fundamental domain (for example, 50 pixels by 50 pixels)
- Get the x (and y) coordinates of the Pixel to decide about.
- Calculate $x \% 50$ (for example) which locates where WITHIN THE FUNDAMENTAL DOMAIN (not the whole Picture) where the Pixel is
- Similarly for $y \% 50$
- Use $x \% 50$ and $y \% 50$ to make the black/white decision.
Make each green pixel white if
\[ x \% 50 < 25 \]
and
\[ y \% 50 < 25 \]
or
\[ x \% 50 \geq 25 \]
and
\[ y \% 50 \geq 25 \]
else color it black.
public void makeRedCheckered()
{
    int x;
    x = 0;
    while( x < this.getWidth() )
    {
        //PROCESS a whole column..
        int y;
        y = 0;
        while( y < this.getHeight() )
        {
            Pixel pRef;
            pRef = this.getPixel(x,y);
            if (pRef.getRed() > (pRef.getGreen() + pRef.getBlue())) // recolor ONLY very red Pixels
            {
                int xRem;
                int yRem;
                xRem = x % 50; // calculate to use soon the x location WITHIN THE FUNDAMENTAL SQUARE
                yRem = y % 50; // calculate to use soon the y location WITHIN THE FUNDAMENTAL SQUARE
                if( /* test for making white versus black coded FROM THE HANDWRITTEN PLAN*/
                    ((xRem < 25) && (yRem < 25))
                    ||
                    ((xRem >= 25) && (yRem >= 25))
                )
                {
                    pRef.setColor(new java.awt.Color(255,255,255));
                }
                else
                {
                    pRef.setColor(new java.awt.Color(0,0,0));
                }
            }
            y = y + 1;
        }
        x = x + 1;
    }
    return ;
}