Making a Ball

• Adding a parametrized method “ball” to a class “Picture”
• Looping to get at many Pixels
• Our 2\textsuperscript{nd} if statement example: if a Pixel is in the ball then color it.
• Use of weighted average
• Method \text{weAv} that returns a (double) value.
• Method \text{weAv} is “static”
• Cast (int) required because truncating a double to an int LOSES INFORMATION.
Making a Ball

• Adding a parametrized method “ball” to a class “Picture”
  – pict.ball(xC,yC,radius) makes the computer work ON this Picture (referred to by the value of pict) to mess this Picture up by painting a ball onto it.
  – So, we say “ball” is an INSTANCE METHOD to mean ball works ON ONE (given) instance of the Picture class. That instance is “a Picture object”
Making a Ball

- Looping to get at many Pixels
  - The loop variables int x and int y hold coordinates, or positions or locations at which Pixel data is stored and changed.
Making a Ball

• Our 2\textsuperscript{nd} if statement example: if a Pixel is in the ball then color it.
  – The for loops restrict this testing and coloring if the Pixel is inside the ball to the smallest square region that contains the ball. This saves the computer time.
Making a Ball

• Use of weighted average
  – Makes the ball look round by coloring the middle white, the circumference a saturated red, and between, a smooth gradation of shades of red.

• Method weAv that returns a (double) value.
  – We wrote it ONCE AND FOR ALL, so if we need a weighted average calculated for some other reason, we do not have to bother coding the formula $N1*W1+N2*W2$ ever again.
Caller and Callee action for a method with a return value.

- Like before, the caller writes parameters on the calling card.
- Instead of doing something on say the caller's Picture, the caller computes some numbers and SHOUTS (communicates) a computed value back to the callee.
- Dramatized: Prof. calls weAv through a cell-phone, tells the callee 4.0, 0.5, 3.0, 0.5 The callee computes 3.5 and tells it to the Prof before she hangs up.
Making a Ball

- Method `weAv` is “static”
- Cast `(int)` required because truncating a double to an int LOSES INFORMATION.
• Method `weAv` that returns a (double) value.
  - We wrote it ONCE AND FOR ALL, so if we need a weighted average calculated for some other reason, we do not have to bother coding the formula `N1*W1+N2*W2` ever again.

• It's unnatural, distracting, misleading, and bad software expression to be forced to always have a Picture ON WHICH TO CALL `weAv(N1,W1,N2,W2)` So, we make it static method. (static is a Java keyword.)
  - That means `weAv` is always and only called ALL BY ITSELF, with 4 double numeric values.
  - “static” is a rather arbitrary word for the concept; it comes from Java's historical predecessors C and C++. 
The Math class

• Google it: Java API Math You can get accurate estimates of pi and e from Math.PI and Math.E. We looked at the info about Math.sin().

• The Math class is a shopping bag holding a lot of methods that are called with numeric parameters, do popular mathematical calculations (for scientists, economists, sometimes business people).

• ALL the methods in Java's Math class are static
  – It makes no sense to force people to call them ON an object.
Instance methods

- Almost all the methods we covered: `forward()` (of Turtle), `show()` (of Picture and of Turtle), `setWidth()`, `getHeight()` and `getPixel` (of Picture), `setColor()` (of Pixel) are **instance methods**

They are ONLY called ON some object (like the Turtle, Picture, or Pixel) on which calling the method makes sense.
Arrays Again beginning Ch.4

double gradeArray[ ];
gradeArray = {80, 90.5, 88, 92, 94.5};
System.out.println(gradeArray.length);
//What's printed?
//(A) 0  (B) 80  (C) 4  (D) 5  (E) 6
System.out.println(gradeArray[4]);
  //(A) 80.0  (B) 92.0
  //(C) 88.0  (D) 94.5
  //(E) nothing, there's a crash.
Before there were arrays, we had to code:

double grade0;
grade0 = 80.0;
double grade1;
grade1 = 90.5
double grade2;
grade2 = 88;
double grade3;
grade3 = 92;
double grade4;
grade4 = 94.5;
Which grade would you like to change?

Input course number and chGrade;
if( course == 0)
{ grade0 = chGrade; }
else if ( course == 1 )
{ grade1 = chGrade; }
else if (course == 2 )
{ grade2 = chGrade; }
else if (course == 3 )
{ grade3 = chGrade; }
else if (course == 4)
{ grade4 = chGrade; }
else
{ System.out.println(“You didn't take that course!”);}
Blurring-text reading's expected!

- Example of “combining pixels”
- One use: Reduce pixelation of enlargements.
- UAlbany 201 extra: Might improve edge detection by removing tiny (technically, high-frequency) features.
- “set each pixel to an average of pixels around it”
- YOU will set average colors into a DIFFERENT PICTURE...so you can do research by comparing things.
Blurring—see textbook

• “set each pixel to an average of pixels around it”
  – Misprint? The target pixel is included too!

• “large loop that surrounds everything”
  PURPOSE: Make its body compute the blurred color for each pixel, one at a time.

• The body: (A) nested loops to accumulate 3 sums

  AND ONE count ;

  (B) 3 divisions by count ;

  PURPOSE: compute the average of the pixels to either side the pixel, up to size pixels away
The counting pattern

Pseudo code will be shown for the textbook's Java code.
Possible exam questions:
We give you Java code, maybe right from the textbook.
YOU EXPLAIN IT by writing pseudo code!!
```c
int count = 0;
Initialize redT, greenT, blueT;//not V!
for each xS, yS near x,y
{
    if( (xS,yS) is in the Picture)
    {
        count = count+1;
        add R,G and B numbers from pixel at (xS,yS) to redT, greenT, blueT
    }
}
Put averages into pix at (x,y);
```
We now zoom in on the Pixel coordinates near (x,y), the if, and the count=count+1 operation to EXPLAIN THEIR PURPOSES.
for each xS, yS near x,y (ALL xS,yS near x,y (bad!))
{
    if( (xS,yS) is in the Picture)
        //NEED the if() to prevent a CRASH
        {
            count = count+1;
            //So..the number of pixels
            //averaged is
            //not always (2*size+1)*(2*size+1)
            //SO..the program counts them.
            add R,G and B numbers
            from pixel at (xS,yS)
            to redT, greenT, blueT
        }
}
What does “near” mean? size is a parameter for how much blurring.

//The “big” loop
for all (x,y) pixel coords in the Picture {
  for(xS = x–size; xS <= x+size; xS++)
    for(yS = y–size; yS <= y+size; yS++)
      { //BODY to compute colors for
        //ONE pixel, at (x,y) in the blurred
        //Picture
      }
}
Here's a 9x4 pixel image. Suppose size is 1

In computing the blurred color of pixel A, how many pixels are averaged? That's the final value of count.

(A) 1  (B) 4  (C) 8  (D) 9  (E) 6
Here's a 9x4 pixel image. Suppose size is 1.

In computing the blurred color of pixel A, how many pixels are averaged? That's the final value of `count`.

(A) 1  (B) 4  (C) 8  (D) 9  (E) 6
Here's a 9x4 pixel image. Suppose size is 1.

In computing the blurred color of pixel B, how many pixels are averaged?
That's the final value of count.

(A) 1  (B) 4  (C) 8  (D) 9  (E) 6
Let size be 1. Two averages are shown. YOU: calculate three more, for the pixels marked?
(size is 1) What's the average in pix. ??

(A)0  (B)1  (C)2  (D)3  (E)6
Let size be 1. What are the 6 averages in the pixels marked with ?s?
Averages? size is 1, so the averages are over 3x3 regions (except near boundaries.)

(A) 

(B) 

(C) 

(D)
Averaging for Proj 3

- Book code REPLACES each pixel by an average
  but YOU must put the averages into A NEW PICTURE!

- Your choice: (iClick your preference, both right.)
  - (A) Put the blurred colors directly in the 3x2 expanded answer Picture (make a method to do this!)
  - (B) Make a new blurred Picture; copy it into the 3x2 expanded answer Picture
    (make 2 methods: one to make a new blurred Picture, second to copy one Picture into another.)
Edge Detection

• Do it on average intensity values.
• Break out computation in the book into separate steps:
  – Compute \( \text{int} \; \text{diff} = \text{Math.abs(topAverage-bottomAverage)}; \)
  – make the color of the top Pixel be \( \text{new Color(diff, diff, diff)} \) to see a gray scale image of absolute differences.
  – Color the 3\(^{rd}\) Picture's Pixels WHITE OR BLACK depending on the test of \text{diff} against amount on page 184.
Instance methods

- Almost all the methods we covered: `forward()` (of Turtle), `show()` (of Picture and of Turtle), `getWidth()`, `getHeight()` and `getPixel` (of Picture), `setColor()` (of Pixel) are instance methods.

They are ONLY called ON some object (like the Turtle, Picture, or Pixel) on which calling the method makes sense.
static methods

- FileChooser.pickAFile(), Math.sqrt(), main(), weAv(N1, W1, N2, W2)

static or class methods

They are ONLY called BY THEMSELVES, NEVER ON object (like the Turtle, Picture, or Pixel) on which calling the method makes sense.
class MyApp {
    public static void main(String a[])
    {
        World w = new World();
        Turtle t = new Turtle(w);
        t.drawL(3.0);
    }
}

class Turtle {
    //LOTS OMITTED..You'll write:
    public void drawL(double scale)
    {
        this.forward((int)(100*scale));
    }
}
class Turtle {
    public void drawL(double scale) {
        this.forward((int)(100*scale));
    }
}

makeTurtleDrawL(t, 3.0);//Caller
SAME thing done with a static method!!
public static void makeTurtleDrawL(Turtle theT, double scale) {
    theT.forward((int)(100*scale));
}
In BOTH the familiar instance way AND the static method way

The caller writes 3 things on the calling card.

Calling Card

(1) where to get at the Turtle

Prof. Chaiken's desk

(2) number for how big to draw

3.0

(3) whom to tell when you're done

Java code just after ...3.0);
In BOTH the instance method way AND the static method way

Calling Card

(1) where to get at the Turtle Prof. Chaiken's desk
(2) number for how big to draw 3.0
(3) whom to tell when you're done Java code just after ...3.0);

(Java system doesn't let programmers see the 3rd box)
class Turtle {
    public void drawL(double scale) {
        this.forward((int)(100*scale));
    }
}

makeTurtleDrawL(t, 3.0); // Caller static method

static void makeTurtleDrawL(Turtle theT, double scale) {
    theT.forward((int)(100*scale));
}
t.drawL(3.0); //Caller code.

class Turtle {
    public void void drawL (double scale) {
        this.forward((int)(100*scale));
    }
}

makeTurtleDrawL (t, 3.0); //Caller

old fashioned kind of method

static void makeTurtleDrawL (Turtle theT, double scale) {
    theT.forward((int)(100*scale));
}
static means: If you want the method to do something involving an object like a Turtle, you must code into the method

AN EXPLICIT PARAMETER
(like theT I made up)
referring to it (the object like a Turtle.)

“not-static” means: The method has an IMPLICIT PARAMETER named this
for the above purpose, so you don't have to make one up.
Object Oriented Programming
One of it's ideas.

Suppose a method like
\texttt{drawT( int size )} makes sense ONLY when applied to an object of a particular kind, like a Turtle.
Suppose a method like `drawT(int size)` makes sense ONLY when applied to an object of a particular kind, like a Turtle.

Object Oriented Programming
One of it's ideas.
IDEA: It's BETTER to force people (professional software developers) to
(A) code CALLERS
   `someTurt.drawT(24)` instead of
   `drawT(someTurt, 24)`
(B) put `drawT` IN the def. of Turtles.
When a CALLER calls an instance (non-static) method, an object to CALL it ON is:

(A) Always required.
(B) Can be there sometimes, but not always.
(C) Is Never Allowed.
class Turtle {
    public void drawI(int size) {
        this.forward((int)2.7*size);
    }
    public void drawT(int size) {
        this.drawI(size);  //A
        drawI(size);       //B
    }
}

Line A and Line B mean exactly the same thing! this refers to the object on which the instance method is called. It is implicit when an instance method of one class (Turtle) calls another instance method of the same class (Turtle).
When a CALLER calls an static (class) method, an object to CALL it ON is:

(A) Always required.
(B) Can be there sometimes, but not always.
(C) Is Never Allowed.
What about the method call
\[
\text{Math.abs}(-34.2)
\]?
This looks like calling a static method ON the Math object!

No! Syntactically and superficially it looks like \texttt{Math} refers to an object on which \texttt{abs(-34.2)} is called, but \texttt{Math} is the name of a class, not a variable referring to an object.

\texttt{abs( )} is a static method in the \texttt{Math} class.

It's a static (also called class) method, so calling it on an object is impossible!
New Topic

A popular Program Control Pattern
The

“do something, see if it's last stop if so, otherwise repeat”

control pattern
Scan your item, or press PAY when you're done.
result = getScanOrPayPress()

if(result is a scan)

put result in the bill

true

false
total up the bill. Take the money.
Flowchart Language

- Box means do something; arrow means “GO TO”
- Diamond means make a decision and branch, based on the outcome (usually an if( ) statement).
- Circle means end or beginning of the computing that this flowchart describes.
- Old-fashioned languages AND TODAY'S MACHINE, ASSEMBLY, and VIRTUAL MACHINE languages do not have loop statements: Only if, method call & return, and GOTO. Flowcharts express loops with ifs and GOTOs.
result = getScanOrPayPress()
while (result is a scan)
{
    put result in the bill;
    result = getScanOrPayPress();
}
Total up the bill;
Take the money;
boolean scanningInProgress = true;
while(scanningInProgress)
{
    result = getInput();
    if(result is "Pay")
        scanningInProgress=false;
    else
        add result to bill;
}
Total up the bill; Take the money;
STATE-based control

• A really good idea for programming user interfaces.
• New states are easily added as features are added.
• Programmers can think:
  – “What should it do when it's in one particular state?
  – “What situation in one particular state should make it CHANGE state?
boolean scanningInProgress = true;

User interface STATE TRANSITION DIAGRAM

while(scanningInProgress) {
    result = getInput();
    if(result is "Pay")
        scanningInProgress=false;
    else
        add result to bill;
}

Total up the bill; Take the money; return;