Project 02 is 3 Files
(3 Java Class Definitions)

- **GraffitiApp.java**
  - contains only:
    ```java
    public static void main(String[] a) { ... }
    ```
    (GraffitiApp is our application container)

- **ArtisticTurtle.java**
  - public class ArtisticTurtle extends Turtle
    ```java
    { ... }
    ```

- **PaintablePicture.java**
  - import java.awt.Color;
    public class PaintablePicture extends Picture
    ```java
    { ... }
    ```
import java.awt.Color;
public class GraffitiApp {

    public static void main(String[] a) {
        FileChooser.pickMediaPath();
        PaintablePicture pRef;
        File file = FileChooser.pickAFile();
        pRef = new PaintablePicture(file);
        ArtisticTurtle tRef = new ArtisticTurtle(pRef);
        // Code to move the Turtle to some spot.
        tRef.behaviorONE(<SOME SCALE VALUE>, ...);
        // Code to move the Turtle to another spot.
        tRef.behaviorONE(<DIFFERENT SCALE VALUE>, ...);
        // Code to move the Turtle to another spot.
        tRef.behaviorTWO(<SCALE VALUE V>, ...);
        // Code to move the Turtle to another spot.
        tRef.behaviorTWO(<DIFFERENT FROM V>, ...);
        // Code for painting.
        pRef.explore();
    }
}
Why is it critical `pRef.explore()`; which commands the computer to put up the explorer, be coded AFTER your 4 method calls?

A) So you and your TA can SEE the results of your 8 (at least) method calls.

B) Exploring a world can't be done until after you finish traveling to it.
public class GraffitiApp
{
    public static void main(String[] a)
    {
        FileChooser.pickMediaPath();
        PaintablePicture pictRef = new PaintablePicture("sdc.JPG");
        ArtisticTurtle tRef = new ArtisticTurtle( pictRef );
        tRef.penUp();
        tRef.moveTo(262,230);
        tRef.penDown();
        tRef.brush(100);
        tRef.penUp();
        tRef.moveTo(275,183);
        tRef.penDown();
        tRef.brush(40);

        my ArtisticTurtle's behavior One called TWO (2) times.
tRef.penUp(); //extra art is

tRef.moveTo(361,186); // ok for fun

tRef.turn(180); //no extra credit:(

(tRef.penDown();

tRef.brush(45);

tRef.penUp();

my

ArtisticTurtle's

behavior Two
called TWO (2)
times.

Different
parameter
values: 10
in first call
and 5 in
second
call
Figure complexity requirement: Turtle figures must require the pen be lifted or retraced (for full credit).
public class ArtisticTurtle extends Turtle {
    public ArtisticTurtle(Picture pRef) {
        super( pRef );
    }
    public void brush( int length ) {
        this.setPenWidth( 5 );
        this.turn(170);
        this.forward(length);
        //The rest is my secret!
    }
    public void dollar( int length ) {
        this.setPenWidth(5);
        this.forward( 10*length );
        this.turn( 90 );
        //The rest is my secret!
    }
}

The parameter MUST be used to control the figure size!!
My `ArtisticTurtle` class defines a third method!

It simply draws a Vee

It is HELPFUL helper method, and an example too simple for YOUR Proj02 assignment.
public void vee(int length) {
    // TOO SIMPLE: WON'T COUNT
    // IF YOU COPY THIS.
    this.turn(150);
    this.forward(length/2);
    this.turn(-120);
    this.forward(length/2);
    this.forward(-length/2);
    this.turn(120);
    this.forward(-length/2);
    this.turn(-150);
} // MATCHES, ends method def.

} // MATCHES public class ...
// to END the class definition
Lab 05: No assignment besides making and starting a strategy for beginning or continuing your Project02 homework assignment.

The ArtisticTurtle part: Don't waste your lab time on that! Sample code that solves the problem is on the Web in and under Lecture 08. You just have to, individually, make the figure complex enough for credit (not just a few connected lines!) and do a second method.

Lab05 is for the PaintablePicture part. First, pick from the Plan X or Plan Y (NOT BOTH!) whichever you like best. Try to complete the step 1) of your chosen plan, and do what you can on step 2). Foils for a lecture on programming of loops follow the plans.
In the next slide, we jump ahead to show you a too-simple example of what is required for the Step 8 Original problem solving and coding. After that are slides about loops.
import java.awt.Color;
public class PaintablePicture extends Picture{
    public PaintablePicture( String fileName )
    { super( fileName ); }
    //The empty space below is good for you to put in the
    //methods you will write.
    public void tooThinSplotch( int midX, int yParm )
    {
        Color purple = new Color( 175, 0, 175 );
        Pixel pixRef;
        int X;
        X = midX - 50;
        int rightX;
        rightX = midX + 50;
        while( X <= rightX )
        {
            pixRef = this.getPixel( X, yParm );
            pixRef.setColor( purple );
            X = X + 1;
        }
    }
} //This brace matches the brace on the second line.
tRef.moveTo(230, 323);
tRef.penDown();
tRef.dollar(10);
tRef.penUp();
tRef.moveTo(350, 319);
tRef.penDown();
tRef.dollar(5);

pictRef.tooThinSplotch(321, 205);
pictRef.tooThinSplotch(421, 205);

// The 2nd step 8 method must be called at least twice too!
Lessons on LOOPS
Loopy program in English to tell a computer (he or she) to print 15 stars.

1. Set up a variable named nStarsPrinted to help you remember how many stars you finished printing.

2. Initialize nStarsPrinted to 0. In plain English, copy 0 into the ticket or box on your calculation sheet named nStarsPrinted.

3. Look at the current value in nStarsPrinted. See if it is still < 15. If not (it is >= 15), skip the three steps below, and follow the instructions after those three steps. (A smart human computer would skip this step for the first star, BUT robotic computers ARE NOT SMART, and your have to program stupid instructions like this when you write programs.)

4. Print one more (or the first) star.

5. Add 1 (one) to the number you see in nStarsPrinted. Overwrite the old value nStarsPrinted with the answer from your addition.

6. Continue computing at step 3. As usual, after each step 3, go on to the next step except if step 3 tells you not to.

7. Report to your supervisor that you are finished, and take a break until your supervisor commands you to run another program.

You have gotten to the end of the program. There is no more after this.
“Loop” means something you might get into, and if you do, you'll go round and round coming back to the same spot until you stop and get off.

Here you decide between whether: (A) Go around (the first time OR again) versus (B) Get off.
Loopy program in English to tell a computer (he or she) to print 15 stars. I've followed each English step by its Java equivalent. (These lines of Java introduce the remainder as a complete application.)

public class FifteenStarPrinter {

    public static void main(String[] a) {

        1. Set up a variable named nStarsPrinted to help you remember how many stars you finished printing.

        int nStarsPrinted;

        2. Initialize nStarsPrinted to 0. In plain English, copy 0 into the ticket or box on your calculation sheet named nStarsPrinted.

        nStarsPrinted = 0;

    }

}
What does a Java declaration of a variable, like `int nStarsPrinted;` do?

A) Set up a memory location to be used for that variable, so you can give it a name later.

B) Set up a memory location to be used for that variable and give it a name at the same time.

C) Store or save a well defined value, like 0, into a variable.
3. Look at the current value in nStarsPrinted. See if it is still < 15. If not (it is < 15), skip the three steps below, and follow the instructions after those three steps. (A smart human computer would skip this step for the first star, BUT robotic computers ARE NOT SMART, and your have to program stupid instructions like this when you write programs.)

```java
while ( nStarsPrinted < 15 )
{
    //This { will enclose the
    //next 2 Java statements.

    Here you decide between whether:
    (A) Go around (the first time OR again) versus
    (B) Get off.
```
3. Look at the current value in nStarsPrinted. See if it is still < 15. If not (it is < 15), skip the three steps below, and follow the instructions after those three steps. (A smart human computer would skip this step for the first star, BUT robotic computers ARE NOT SMART, and your have to program stupid instructions like this when you write programs.)

\[
\text{while ( nStarsPrinted } < 15 \text{ )} \\
\{ \\
\text{//This } \{ \text{ will enclose the} \\
\text{//next 2 Java statements.} \\
\}
\]

4. Print one more (or the first) star.

\[
\text{System.out.print( “*” );}
\]

5. Add 1 (one) to the number you see in nStarsPrinted. Overwrite the old value in nStarsPrinted with the answer from your addition.

\[
n\text{StarsPrinted}=\text{nStarsPrinted+1;}
\]

6. Continue computing at step 3. As usual, after each step 3, go on to the next step except if step 3 told you not to.

\[
\}
\]

Yes, that one single } (matching the \{ under step 3) really means all that! (Hmm, maybe robotic computers aren't so dumb after all.)
3. Look at the current value in nStarsPrinted. See if it is still < 15. If not (it is >= 15), skip the three steps below, and follow the instructions after those three steps. (A smart human computer would skip this step for the first star, BUT robotic computers ARE NOT SMART, and you have to program stupid instructions like this when you write programs.)

```
while ( nStarsPrinted < 15 )
{
   //This { will enclose the
   //next 2 Java statements.
   Step 5. Add 1 and
   rewrite.

   +

   ★ Step 4. Print *

   Step 3. Here you decide between whether:
(A) Go around (the first time OR again)
versus (B) Get off.
```
3. Look at the current value in nStarsPrinted. See if it is still < 15. If not (it is >= 15), skip the three steps below, and follow the instructions after those three steps. (A smart human computer would skip this step for the first star, BUT robotic computers ARE NOT SMART, and your have to program stupid instructions like this when you write programs.)

   while ( nStarsPrinted < 15 )
   {
      //This { will enclose the
      //next 2 Java statements.

4.     Print one more (or the first) star.

      System.out.print( "*" );

5.     Add 1 (one) to the number you see in nStarsPrinted.
      Overwrite the old value in nStarsPrinted with the answer
      from your addition.

      nStarsPrinted=nStarsPrinted+1;

6.     Continue computing at step 3. As usual, after each step 3, go on to the next step except if step 3 told you not to.

   }

   Yes, that one single } (matching the { under step 3) really means all that! (Hmm, maybe robotic computers aren't so dumb after all.)
7. Report to your supervisor that you are finished, and take a break until your supervisor commands you to run another program.

Sometimes, the “operating system” (Linux, Windows, MacOS..) is called the “supervisor”

```
return ;  //It's a rule of Java that
//returning from main leads to an implicit
//system exit operation. The operating system
//is like a supervisor. Also, you can omit the
//return statement at the end of a method body
//(except if the method returns a value, like int
//instead of void.
```

```}
```
```
You have gotten to the end of the program. There is no more after this.
```
```
```}
```
```
```
```
```}
```
```
```
```
```
```
```
Tip: If your braces don't match, you will get compiler error messages. If the } that is supposed to end the class definition is omitted, the error message will be: ”reached end of file while parsing”
public class FifteenStarPrinter
{
    public static void main(String[] a)
    {
        int nStarsPrinted;
        nStarsPrinted = 0;
        while( nStarsPrinted < 15 )
        {
            System.out.print( "*" );
            nStarsPrinted = nStarsPrinted + 1;
        }
        return ;
    }
}
public class FifteenStarPrinter
{
    public static void main(String[] a)
    {
        int nStarsPrinted;
        nStarsPrinted = 0;
        while( nStarsPrinted < 15 )
        {
            System.out.print( "*" );
            nStarsPrinted = nStarsPrinted + 1;
        }
        return ;
    }
}

I like to think of the code that sets up the variables and data to control the loop is part of the loop idea.

This box encloses a loop.
Suppose you want to do 15 push-ups for exercise. How would you prefer to count them?

A) Count up, 1, 2, 3, 4, ..... , 12, 13, 14, 15 got to 15 STOP.

B) Count down, 15, 14, 13, 12, ..... , 3, 2, 1 (one more...), 0 STOP.

(Opinion question: no right answer.)
public class TenStarPrinter {
    public static void main(String[] a) {
        int nStarsIWillPrint;
        nStarsIWillPrint = 15;
        while (nStarsIWillPrint > 0) {
            System.out.print( "*" );
            nStarsIWillPrint = nStarsIWillPrint - 1;
        }
        return ;
    }
}

Here is a count-down style loop. [3 code changes + renaming a variable]
import java.util.Scanner;
public class InteractiveStarPrinter
{
    public static void main(String[] a)
    {
        Scanner sc = new Scanner(System.in);
        int nStarsPersonWants;
        nStarsPersonWants = sc.nextInt();
        int nStarsIWillPrint;
        nStarsIWillPrint = nStarsPersonWants;
        while( nStarsIWillPrint > 0 )
        {
            System.out.print( "*" );
            nStarsIWillPrint = nStarsIWillPrint - 1;
        }
        return ;
    }
}
import java.util.Scanner;

public class InteractiveStarPrinter
{
    public static void main(String[] a)
    {
        Scanner sc = new Scanner(System.in);
        int nStarsPersonWants;
        nStarsPersonWants = sc.nextInt();
        int nStarsPrinted;
        nStarsPrinted = 0;
        while( nStarsPrinted < nStarsPersonWants )
        {
            System.out.print( "*" );
            nStarsPrinted = nStarsPrinted + 1;
        }
        return;
    }
}

This box encloses a loop.
public class Internal250To350Generator
{
    public static void main(String[] a)
    {
        int numberBox;
        numberBox = 250;
        while( numberBox <= 350 )
        {
            System.out.println( numberBox );
            numberBox = numberBox + 1;
        }
        return ;
    }
}

SAME computing idea, but for a bigger range of bigger numbers. Start at 250, do 251, 252, 253, .... 347, 348, 349, 350. That's 101 numbers. The last number printed is 350. After the print, numberBox contains 351.
import java.awt.Color;
public class PictureLinePurpler {
    public static void main(String[] a)
    {
       FileChooser.pickMediaPath();
        Picture pRef;
pRef = new Picture(FileChooser.pickAFile());
    Color purple = new Color( 175, 0, 175 );
    Pixel pixRef;
    int X;  //People: Be smart/lazy: Make the
            //computer do subtracting to get 271.
    X = 321-50;  //computer do do subtracting to get 271.
    int rightX;  //A row of pixels from (271,205)
    rightX = 321+50;  //to (371,205) made purple shows up
    while( X <= rightX )  //as a thin purple line across
    {
        //Chaiken's nose.
        pixRef = pRef.getPixel( X, 205 );
pixRef.setColor( purple );
        X = X + 1;
    }
    pRef.explore();
}

Only 12 lines command the computer to recolor 101 Pixels!
IntGeneratorAndPrinter objects will soon get a method that uses a loop (the loop is inside the method) to implement int printing behavior.

```java
public void printToFrom(int numberBox, int maxWanted) {
    //Here is the body. The
    //LOOP is inside
    the body.
}
```
public class IntGeneratorAndPrinter {
    public void printFromTo(int numberBox, int maxWanted) {
        while (numberBox <= maxWanted) {
            // Comment: One of the ints (first param. value)
            // up to (2nd param. value) inclusively is
            // now available in numberBox, hidden in memory.
            // Put code below to read and use that number.
            System.out.println(numberBox);
            // Put useful code here. Comment out the println
            // when debugging and testing are finished.
            numberBox = numberBox + 1;
        }
        return;
    }
    
    public static void main(String[] a) {
        IntGeneratorAndPrinter objRef = new IntGeneratorAndPrinter();
        objRef.printFromTo(250, 350);
    }

    // A main method like this is a good place to put
    // code for testing a class you created for general use.
}
The computer copies two parameter values 250, 350 into two parameter variables `numberBox`, `maxWanted`. When?
(A) Before the method call.
(B) During the method call but before the `while` loop begins.
(C) After the `while` loop begins but before the method returns or finishes.
(D) After the method call returns (or finishes).
(E) Never. Variables are initialized ONLY by = (assignment), NOT by method calls. (specious non-fact)