Unit 3 ~ Input and Methods

Unit 3A

User Input

Input can come from a variety of sources. Mostly we will deal with input from the keyboard. File input is another common input source.

As a user types, the characters go into the *input stream*. Java has a **Scanner** class that we can use for reading in from this stream.

First, you must import the Scanner package.

```
import java.util.Scanner;
```

Then you must *instantiate* (create) a Scanner object and *initialize* (load with values) it with an input stream. We can do this in one line of code.

```
Scanner input = new Scanner(System.in);     //watch for case sensitivity
```

Or we can do it in two lines of code:

```
Scanner input;
input = new Scanner(System.in);
```

Once you have created a Scanner object, we can use the Scanner methods to do things.

Some Scanner Methods	
Method	**Use**
next()	returns the next one word String
nextLine()	returns the String to the End of Line (EOL)
nextInt()	Returns the next int value
nextDouble()	Returns the next double value
nextFloat()	Returns the next float value
etc.	

LOOK UP

Don't forget to check the **api** for more information on java methods.
http://docs.oracle.com/javase/7/docs/api/java/util/Scanner.html

Scanner tutorial: http://www.java-made-easy.com/java-scanner.html

LOOK UP

Parts of a Java Program
http://mathbits.com/MathBits/Java/Introduction/ProgramParts.htm

Reading in Integers

Here is some sample code:

```
Scanner keyboard = new Scanner (System.in);
System.out.print("Enter an integer: ");
int num = keyboard.nextInt();
```

Things to note:

- print(), not println() was used to make your prompts cleaner.
- a space after the : in the output line to also keep your prompts cleaner
- num was declared and initialized on one line
- nextInt () will read up to the first whitespace value entered (enter, tab, space)

Now you can have your program use *num* however you want.

Open ScannerInts.java and ScannerReals.java in Dr. Java and run. Look through the code. Ask questions about anything you don't understand.

Reading in Strings

You can read in Strings with *next()* and *nextLine()*.

- next() reads in the next text value entered up to the white space.
- nextLine() reads in the text until the end of line, including the EOL character

Open ScannerStrings.java in Dr. Java and run. Look through the code. Ask questions about anything you don't understand.

Problems with nextLine()

If you read in an integer with nextInt(), then the stream pointer is sitting at the *enter* (\n) character that you would have used after typing your number. Then, when you read in with *nextLine*(), the only thing it reads is the EOL character.

Open NextLineIssues.java and MultiRead.java in Dr. Java and run. Look through the code. Ask questions about anything you don't understand.

Close() You should close the input stream when you are finished reading in from it.

GUI Input

GUI is Graphical User Interface. If you studied Visual Basic, then you used GUI all the time. Java is different. We can still write the same programs that we did in VB, but you have to do a little more work. However, the work is very repetitive and easy once you get the hang of it.

Open GuiHelp.java in Dr. Java and run. Look through the code. Make a few simple changes. Ask questions about anything you don't understand.

Unit 3A Lab [1]

Lab 3 - INPUT

Lab Goal : This program was designed to teach you how to define, input, and output a variable.

Lab Description : Define, input, and print some variables and their values.

Lab Shell :

```
public class Lab0c
{
  public static void main (String[] args)
  {
      Scanner keyboard = new Scanner(System.in);
      int intOne, intTwo;

      System.out.print("Enter an integer :: ");
      intOne = keyboard.nextInt();

      System.out.print("Enter an integer :: ");
      intTwo = keyboard.nextInt();

      System.out.println("integer one = " + intOne);
      System.out.println("integer two = " + intTwo);
   }
}
```

Files Needed ::
Lab0c.java

Sample Input:

```
Enter an integer :: 2
Enter an integer :: 3

Enter a double :: 4
Enter a double :: 5

Enter a float :: 6
Enter a float :: 7

Enter a short :: 8
Enter a short :: 9
```

BONUS ::
Read in byte and long values.

Sample Output :

```
integer one = 2
integer two = 3

double one = 4.0
double two = 5.0

float one = 6.0
float two = 7.0

short one = 8
short two = 9
```

Bonus Sample Output :

```
SAME INPUT AS ABOVE
SAME OUTPUT AS ABOVE

intOne + intTwo = 5
```

You need to make some new variables.

```
int intTotal = 0;
double doubleTotal = 0.0;
```

[1] Lab0c.doc

Unit 3B

Type Casting

Type casting converts a number from one type to another compatible type. For example, you might want to convert an integer to a double so that you don't lose accuracy in your results.

```
int x = 9;
double y;
y = x / 4;              //results in 2
y = double(x)/4;        //results in 2.5
```

Type casting a double to an int *truncates* the result. ie 4.7 would become 4, not 5

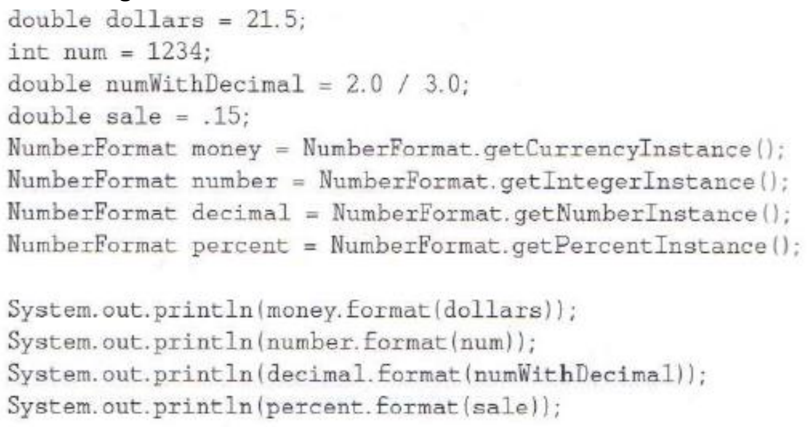
Formatting Numeric Output

NumberFormat()

There are a few ways to format numbers. The first is the NumberFormat class.

```
import java.text.NumberFormat;
```

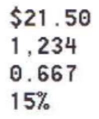
Given the following code:



```
double dollars = 21.5;
int num = 1234;
double numWithDecimal = 2.0 / 3.0;
double sale = .15;
NumberFormat money = NumberFormat.getCurrencyInstance();
NumberFormat number = NumberFormat.getIntegerInstance();
NumberFormat decimal = NumberFormat.getNumberInstance();
NumberFormat percent = NumberFormat.getPercentInstance();

System.out.println(money.format(dollars));
System.out.println(number.format(num));
System.out.println(decimal.format(numWithDecimal));
System.out.println(percent.format(sale));
```

the results would be



```
$21.50
1,234
0.667
15%
```

The DecimalFormat class also offers formatting options for decimals. Look online for more info.

Format()

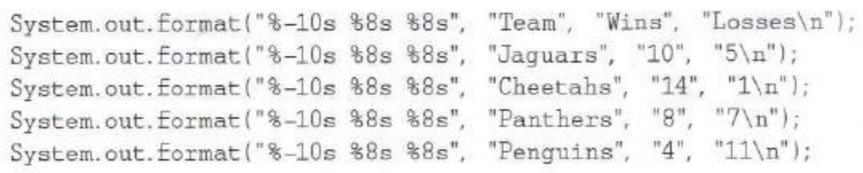
To format output into a nice looking table, you can use the **format()** method. You give the method a format string that describes what your output will look like and your output. The format string takes the form

%[alignment][width][.decimals]f

where

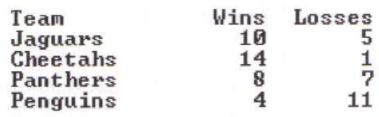
%	indicates the start of the specifier
[alignment]	skip this for right alignment, minus sign for left alignment
[width]	width of column
[decimals]	indicates the number of decimals for a float
f	indicates the data is a floating point number
d	put where f is, indicates the data is an integer
s	put where f is, indicates the data is a String

For example:



```
System.out.format("%-10s %8s %8s", "Team", "Wins", "Losses\n");
System.out.format("%-10s %8s %8s", "Jaguars", "10", "5\n");
System.out.format("%-10s %8s %8s", "Cheetahs", "14", "1\n");
System.out.format("%-10s %8s %8s", "Panthers", "8", "7\n");
System.out.format("%-10s %8s %8s", "Penguins", "4", "11\n");
```

results in



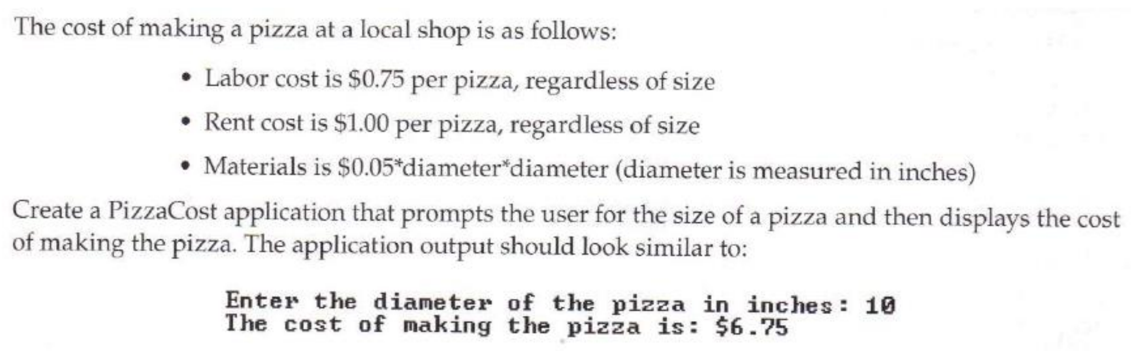
```
Team           Wins   Losses
Jaguars          10        5
Cheetahs         14        1
Panthers          8        7
Penguins          4       11
```

Unit 3B Labs

GradeAvg

Create a GradeAvg app that prompts the user for 5 grades and then displays the average of the grades. Assume the grades are entered as integers. Real division should always be performed when calculating average so type cast as required.

PizzaCost



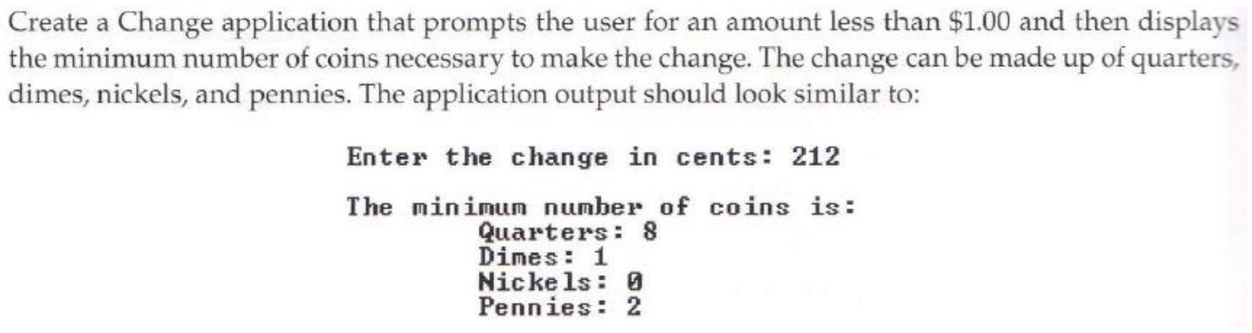
The cost of making a pizza at a local shop is as follows:

- Labor cost is $0.75 per pizza, regardless of size
- Rent cost is $1.00 per pizza, regardless of size
- Materials is $0.05*diameter*diameter (diameter is measured in inches)

Create a PizzaCost application that prompts the user for the size of a pizza and then displays the cost of making the pizza. The application output should look similar to:

```
Enter the diameter of the pizza in inches: 10
The cost of making the pizza is: $6.75
```

Change



Create a Change application that prompts the user for an amount less than $1.00 and then displays the minimum number of coins necessary to make the change. The change can be made up of quarters, dimes, nickels, and pennies. The application output should look similar to:

```
Enter the change in cents: 212

The minimum number of coins is:
        Quarters: 8
        Dimes: 1
        Nickels: 0
        Pennies: 2
```

Unit 3A and 3B Worksheet

[2]Fill in each blank below.

1. java.util.Scanner; is the import needed to use Scanner.
2. nextInt(); is the Scanner method used to input an integer.
3. nextDouble(); is the Scanner method used to input a double.
4. nextString(); is the Scanner method used to input a one word String.
5. next(); is the Scanner method used to input a one line String.

[3]Part 2

```
System.out.print( "one" );                        // LINE 1
System.out.print( "o\tne" );                      // LINE 2
System.out.print( "on\te" );                      // LINE 3
System.out.print( "on\\e" );                      // LINE 4
System.out.print( 4 + 3);                         // LINE 5
System.out.print( 3 + " " + 4 );                  // LINE 6
System.out.print("\" quotes  \"");                // LINE 7
System.out.print( "\\\\t\\a\\");                  // LINE 8
System.out.print( "\\a\\b\\c" );                  // LINE 9
System.out.print( 3 + " " + 4 + 5);               // LINE 10
System.out.print( 3 + 4 + " " + 5 + 6);           // LINE 11
```

1.	one
2.	o ne
3.	on e
4.	on\e
5.	7
6.	34
7.	" quotes "
8.	\\t\a\
9.	\a\b\c
10.	345
11.	756

[2] inputworksheet.doc
[3] syntaxoutputworksheet.doc

Unit 3C

Methods

[4]It was recognized long ago that programming is best accomplished by working with smaller sections of code that are connected in very specific and formal ways. Programs of any significant size should be broken down into smaller pieces. Classes can be used to create objects that will solve those smaller pieces. We determine what behaviors these objects will perform. These behaviors of the objects are called **methods**.

Think back to your first program in this course…we used

```
        myPencil.forward(100);
        myPencil.turnLeft();
```

You have already frequently used the println() method, as well.

Sometimes methods have parameters. Sometimes they do not. Sometimes it is optional. In the example above,

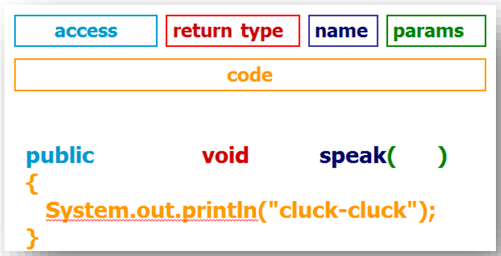
```
        myPencil.forward(100);
```

the call to *forward* passes the **argument** 100 to the forward **parameter** expecting it.

Arguments → the actual values we pass
Parameters →the description of the arguments we will be passing

Method Signature

A method has a signature that provides information about the method.



```
public          void        speak(    )
{
  System.out.println("cluck-cluck");
}
```

In the figure above, the

- access is **public** so it can be called from any location
- return type is **void** so it isn't returning any value
- name is **speak** so that's how we will call it
- parmater list is empty so when we call it, we don't include an argument

To use this method, we declare a Chicken (don't worry about that part for now), and then the chicken 'speaks'.

```
        Chicken red=new Chicken();
        red.speak();                         // "cluck-cluck" appears in the console
```

[4] http://staff.fcps.net/scombs/apcompsci/LessonA4/index.html

Open Chicken.java in Dr. Java and run. Look through the code. Run and experiment.
Open Turkey.java and TurkeyRunner.java. Look through the code. Run and experiment. Ask questions if you have them.

Unit 3C Labs

Lab 01A – STARS AND STRIPES

Lab Goal : The lab was designed to teach you how to write a class and to write methods for that class.

Lab Description : Write methods for class Stars and Stripes. Then, call the methods to create the stars and stripes patterns shown below. Make your own pattern for extra credit.

Lab Shell :

```
public class StarsAndStripes
{
   public StarsAndStripes()
   {
      out.println("StarsAndStripes");
      printTwoBlankLines();
   }

   public void printTwentyStars()
   {
   }

   public void printTwentyDashes()
   {
   }

   public void printTwoBlankLines()
   {
   }

   public void printASmallBox()
   {
   }

   public void printABigBox()
   {
   }
}
```

Sample Output :

```
StarsAndStripes


--------------------
********************
--------------------
********************
--------------------
********************
--------------------


--------------------
********************
--------------------
********************
--------------------
********************
--------------------
--------------------
********************
--------------------
********************
--------------------
********************
--------------------
```

Files Needed ::
StarsAndStripes.java
Lab01a.java

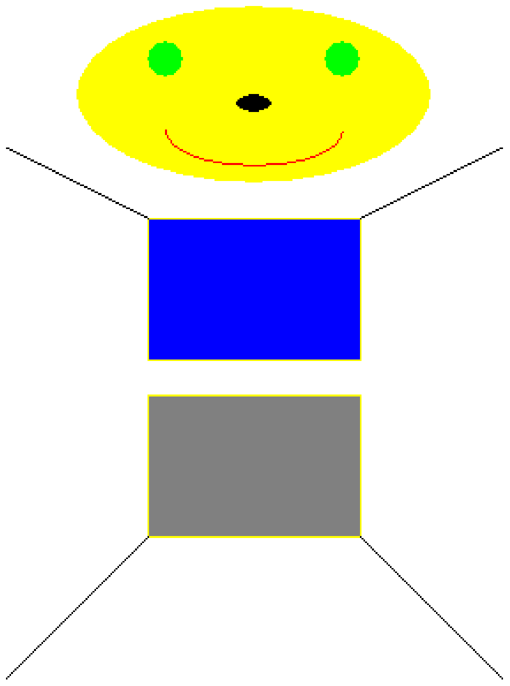
Lab 01D – ROBOT

Lab Goal : The lab was designed to teach you how to write a class and to write methods for that class. It was also designed to teach you how to call methods that have parameters. You will be calling graphics methods with parameters and methods where you define the parameters.

Lab Description : Draw a robot to the best of your ability.

Sample Output :

Files Needed ::
`GraphicsRunner.java`
`Robot.java`



✓ Submit your labs for marking as per instructions.

Unit 3D

Constructors

Constructors are special methods used to create an instance of your class, that is, they *construct the object*. They set up the initial values (field/instance variables) for the object.

- The constructor always has the *exact same name* as the class.
- They never have a return type, such as void.
- They may take parameters.
- There may be more than 1 as long as the parameter list is different in each one.

For example, if you have a class Dog:

```
public class Dog
{
        //field variables aka instance variables
        private int age;
        private int weight;

//constructor with 2 parameters
public Dog (int howOld, int howBig)
{
        age = howOld;
        weight=howBig;
}
}
```

In another class:

```
Dog milo = new Dog (3, 62);
```

In this example, age and weight are **field variables**. That is, they are *global* to the *class*. They are **private**, though, as all field variables are, and cannot be directly changed from outside the class. When this Dog is created from another class, we set the age and weight by passing it that information through parameters and the constructor constructs the new Dog with that information, setting up the field variables.

Dog milo
-age = 3 -weight = 62
methods go here

This is a basic UML diagram. The minus signs mean private access.

Overloaded Constructors

As mentioned, you can have several constructors, as long as the parameter lists are distinguishable from one another. For example,

```
public Dog (int howOld, int howBig, String name);
```

A call using this constructor would look like:

```
        Dog milo =  new Dog (3, 62, "Milo");
```

Note that if 3 and 62 were swapped here, the dog would have age=62 and weight=3.

Unit 3D Lab

Circle

Circle
-radius
+calcArea +calcCircumference +displayInfo

Write a Circle2 class that has 2 constructors
- one with no parameters and that sets the radius of the circle to 0
- one with a parameter that accepts the radius

Include a method for calculating the area.
Include a method for calculating the circumference.
Include a method that prints both area and circumference to the console.

Lab 02B – SUM

Lab Goal : This lab was designed to teach you how to instantiate an object, pass parameters, calculate values, and display the results.

Lab Description : Given two numbers, calculate the sum and print the sum.

sum = one + two   sum is equal to the value of one added to the value of two

Class Shell:

```
class Sum
{
   private double one, two, sum;

   public void setNums(double num1, double num2)
   {
      one=num1;
      two=num2;
   }

   public void sum( )
   {


   }

   public void print( )
   {


   }
}
```

Files Needed ::
Sum.java
Lab02b.java

Sample Data:

```
5 5
90 100
100.5 85.8
-100 55
15236 5642
1000 555
```

Sample Output :

```
5.0 + 5.0 == 10.00

90.0 + 100.0 == 190.00

100.5 + 85.8 == 186.30

-100.0 + 55.0 == -45.00

15236.0 + 5642.0 == 20878.00

1000.0 + 555.0 == 1555.00
```

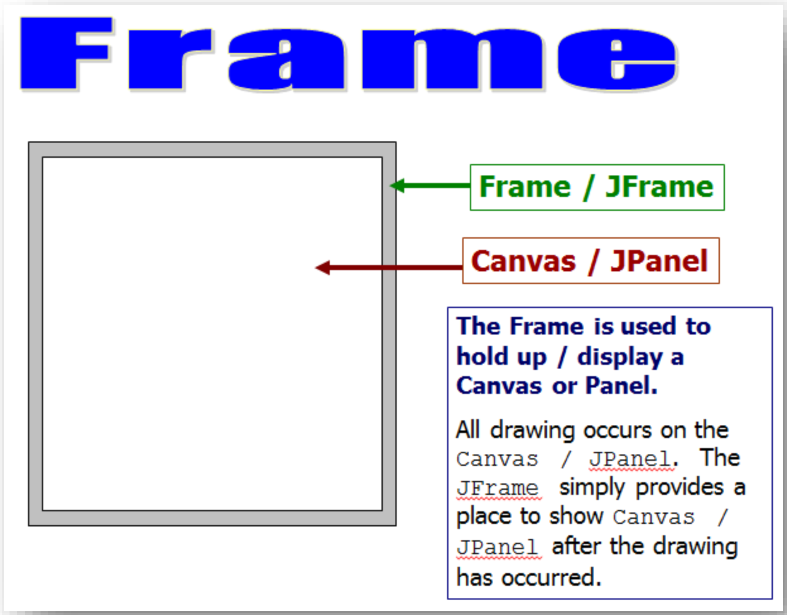
FORMATTING OUTPUT

```
double dec = 9.541724;

out.printf("%.3f\n",dec);                  //outs 9.542
      //printf is a void method


out.println(String.format("%.3f",dec));    //outs 9.542
      //format is a String return method
      //format is useful when writing toString() methods
```

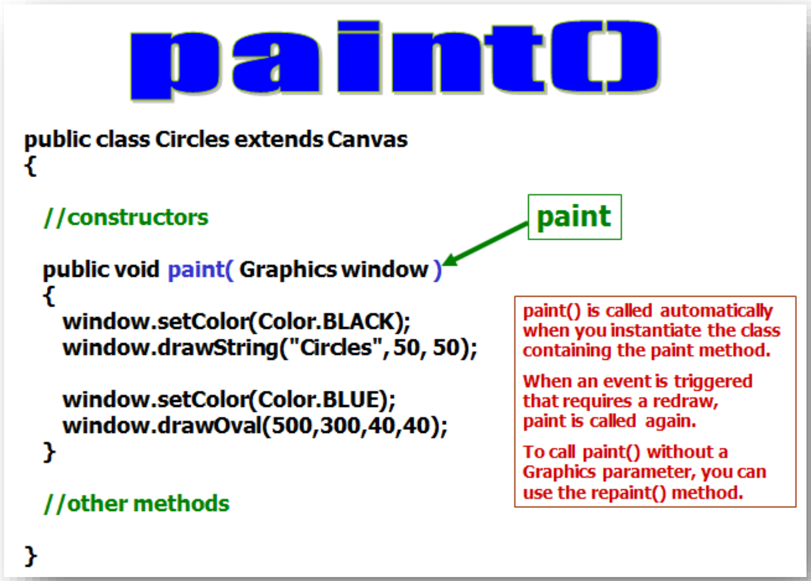
More GUI



The Frame is used to hold up / display a Canvas or Panel.

All drawing occurs on the `Canvas / JPanel`. The `JFrame` simply provides a place to show `Canvas / JPanel` after the drawing has occurred.

The method **paint()** is typically used to draw Graphics on the window. paintComponent() is another method for drawing/redrawing the window.


paint()
paint

```
public class Circles extends Canvas
{
  //constructors

  public void paint( Graphics window )
  {
    window.setColor(Color.BLACK);
    window.drawString("Circles",50, 50);

    window.setColor(Color.BLUE);
    window.drawOval(500,300,40,40);
  }

  //other methods

}
```

paint() is called automatically when you instantiate the class containing the paint method.

When an event is triggered that requires a redraw, paint is called again.

To call paint() without a Graphics parameter, you can use the repaint() method.

Your Turn Open GraphicsRunner.java and Circles.java. Run the program. Explore anad experiment with the code.

Graphics Methods

```
import java.awt.Graphics;
import java.awt.Color;
import javax.swing.JFrame;
```

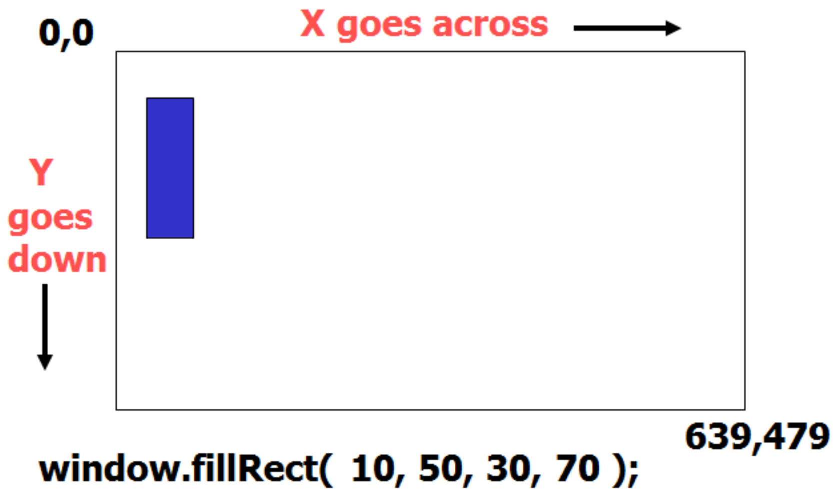
Frequently used Graphics methods	
Method	**Use**
setColor(x)	Sets the current drawing color to x
drawstring(s,x,y)	Draws String s at point x,y
drawOval(x,y,w,h)	Draws an unfilled oval at point x,y that is w wide and h tall
fillOval(x,y,w,h)	Same as drawOval but filled with current Color
drawLine(a,b,c,d)	Draws a line starting at point a,b and going to point c,d
drawRect(x,y,w,h)	Draws an unfilled rectangle at spot x,y that is w wide and h tall
fillRect(x,y,w,h)	Same as drawRect() except filled with the current Color

LOOK UP The Graphics api contains a full list of methods.

Parameters

Parameters are the data that is passed to the method in the parenthesis () following the method name. Most, if not all of the Graphics class methods require parameters. These communicate information about what needs to be done. For example, setColor() requires a Color parameter. Without it, the color cannot be set.

The Graphics Screen



LOOK UP drawArc() and fillArc()

Your Turn Open Rectangles.java, Lines.java, Arcs.java, Fonts.java and Colors.java and experiment. Polygons.java will demonstrate how to create a polygon. It uses arrays, however.

Lab 0123 – CREATE YOUR OWN SHAPE

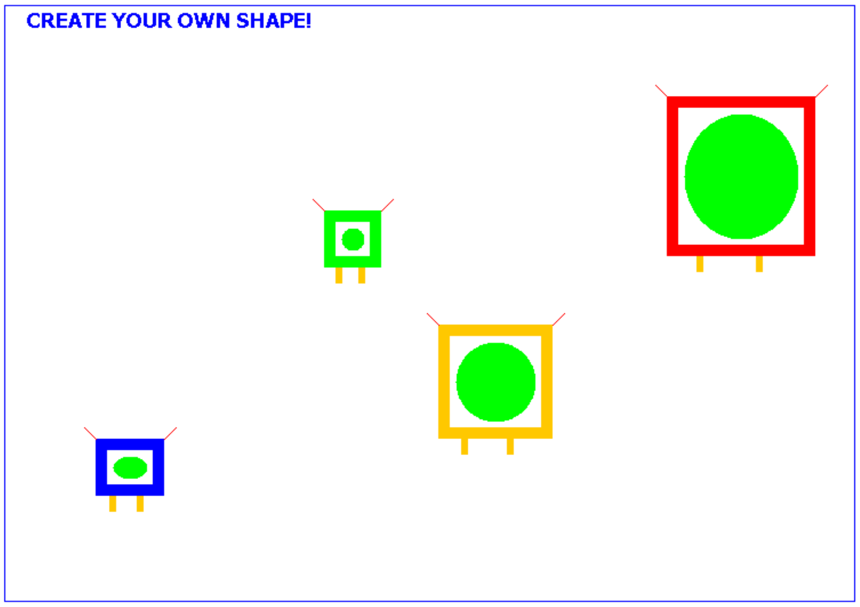
Lab Goal : This lab was designed to teach you how to design and use classes, instantiate objects, and use graphic methods.

Lab Description : Write a program to create a unique shape using graphics, instance variables, constructors, and classes.

Part One :: Use the `Shape.java` file to create your own shape. Complete the Shape constructor and add code to the draw method to draw your own shape. Use `ShapePanel.java` and `GraphicsRunner.java` to test your `Shape` class. Be creative and come up with something really cool.

Files Needed ::
`Shape.java`
`ShapePanel.java`
`GraphicsRunner.java`

Output :

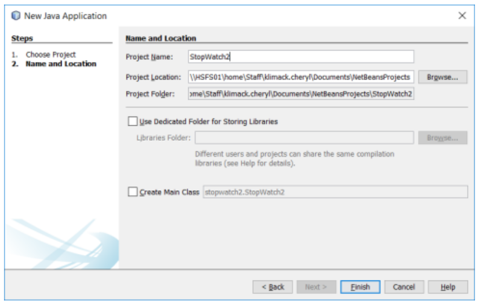


Unit 3F

StopWatch Project

Using Netbeans makes this process much more simple.

1. Create a new NB project named **StopWatch**. Do NOT create a main class.



New Java Application

Steps
1. Choose Project
2. **Name and Location**

Name and Location

Project Name: StopWatch2

Project Location: \\HSFS01\home\Staff\klimack.cheryl\Documents\NetBeansProjects

Project Folder: ›me\Staff\klimack.cheryl\Documents\NetBeansProjects\StopWatch2

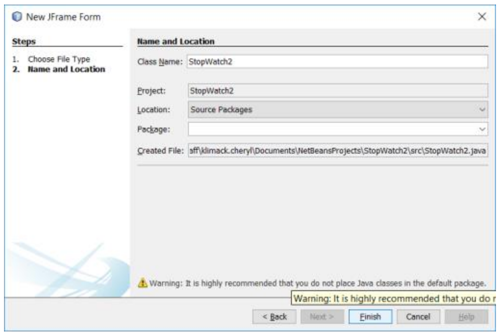
☐ Use Dedicated Folder for Storing Libraries

Libraries Folder:

Different users and projects can share the same compilation libraries (see Help for details).

☐ Create Main Class stopwatch2.StopWatch2

2. Add a JFrame Form to the project by right clicking on the project.



New JFrame Form

Steps
1. Choose File Type
2. **Name and Location**

Name and Location

Class Name: StopWatch2

Project: StopWatch2

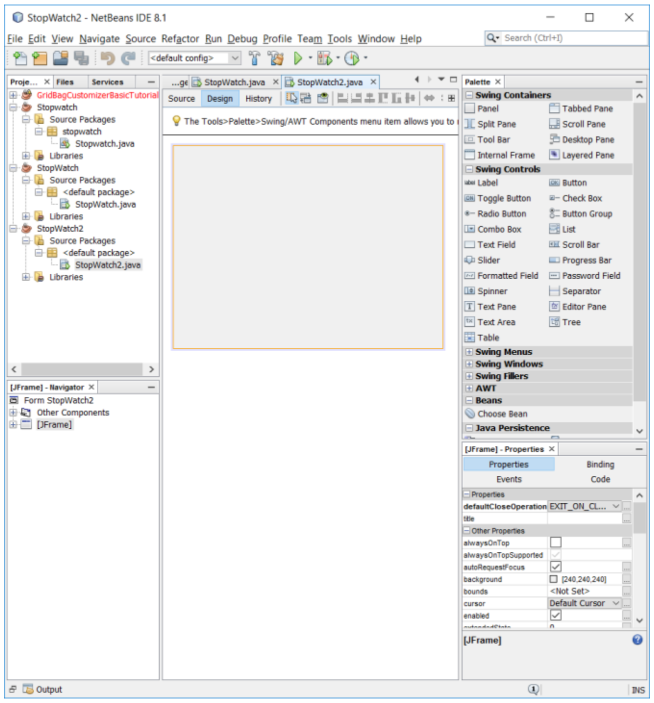
Location: Source Packages

Package:

Created File: aff\klimack.cheryl\Documents\NetBeansProjects\StopWatch2\src\StopWatch2.java

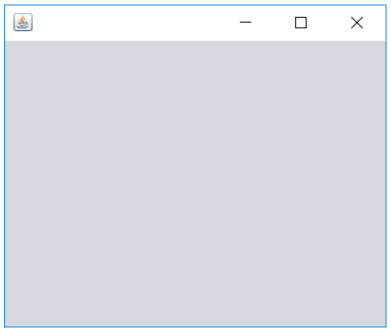
Warning: It is highly recommended that you do not place Java classes in the default package.

You will see the following screen, indicating a form with no controls. You may run it, and see a blank form running. Choose this class for the **main class** when prompted.

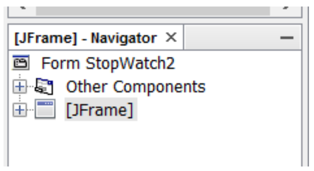


StopWatch2 - NetBeans IDE 8.1

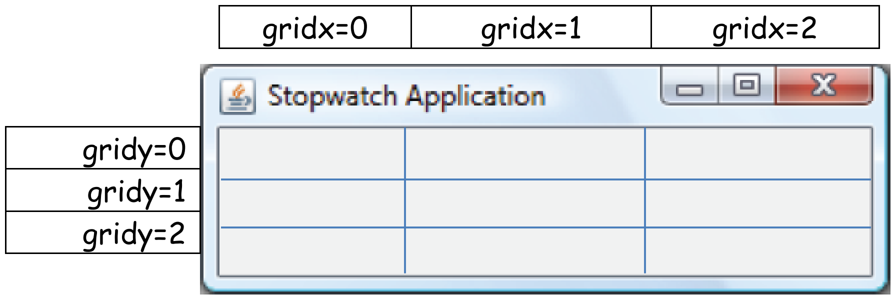
The Tools>Palette>Swing/AWT Components menu item allows you to



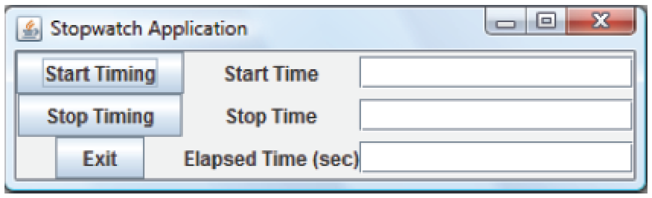
3. Next, right click on the JFrame in the Navigator pane, and set the Layout to **GridBagLayout**. This basically will allow you to work in a flexible, table format.



GridBagLayout



Envision the layout as a grid. We will be using **GridBagLayout** for most of our apps, which will allow us to position objects as though in a table.

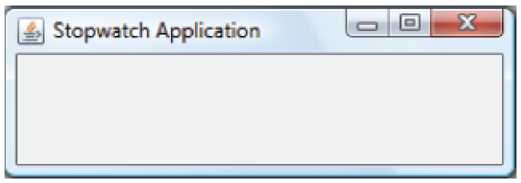


Constructor

If you click the Source tab, you can view the code that has already been generated for you. You can write all this code manually, if you desire. Note your **constructor** code.

4. Go back to the Design tab. Choosing your JFrame, set the title to "Stopwatch Application" and the width to 300, the height to 100.

5. Run, and view your constructor again. View the **main method,** as well. The main could have been simpler, but this IDE generates if for us so we will leave it alone.



Main Method

```
public static void main (String args[]) {
        //construct the frame
        new StopWatch().setVisible(true);
}
```

Creating Controls
Using this IDE, we can drag the appropriate controls onto the JFrame. You will note that when we use the GridBagLayout, we can't drag them around on the form…yet.

6. Add the following controls, and name them as indicated.

   Button startButton, Label startLabel, TextField startTextField, Button stopButton

7. After you get the above 4 controls in, choose JFrame, Customize Layout
   From here you can drag the controls around.
   Do this again after adding more labels.

8. Textfields for this have **columns** set to 15, and no text.

9. Add the rest of the controls

   Button: exitButton
   Label: stopLabel
   Label elapsedLabel
   TextField stopTextField
   TextField elapsedTextField

10. Check out your code again. Note the pack(); in the constructor.

    pack(); //this command must come AFTER the control(s) are added. Make sure you insert the rest of the controls ABOVE it. Your project should have a button in it now.

Event Methods and their Listeners

If we double click a control, the appropriate **event listeners and corresponding methods** are created in the code. We will code the **event methods** (this is what happens when the user interacts with the control). The **listener** is automatically generated to listen for the user to do something to the control.

11. Start with the Exit button. The code is

```
System.exit(0);
```

    Test it.

Variables

12. Declare 3 field variables:

```
long startTime;
long stopTime;
double elapsedTime;
```

13. In the event method startButtonActionPerformed() add this code:

```
startTime = System.currentTimeMillis();
startTextField.setText(String.valueOf(startTime));
stopTextField.setText("");
elapsedTextField.setText("");
```

14. In the stopButtonActionPerformed() method add this code:

```
// click of stop timing button
stopTime = System.currentTimeMillis();
stopTextField.setText(String.valueOf(stopTime));
elapsedTime = (stopTime - startTime) /1000.0;
elapsedTextField.setText(String.valueOf(elapsedTime));
```

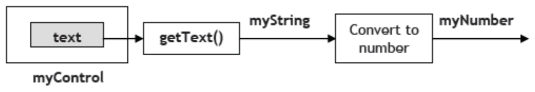
Try some of these things:

- Try changing the frame background color.
- Notice you can press the ‘**Stop Timing’** button before the ‘**Start Timing’** button. This shouldn’t be so. Change the application so you can’t do this. And make it such that you can’t press the ‘**Start Timing’** until ‘**Stop Timing’** has been pressed. Hint: Look at the button **enabled** property.
- Can you think of how you can continuously display the ‘**End Time’** and ‘**Elapsed Time’**? This is a little tricky because of the event-driven nature of Java. Look at the **Timer** class (do a little Java research). By setting the **delay** property of this class to **1000**, it will generate its own events every one second. Put code similar to that in the event method for the **stopButton** in the **Timer** class’ **actionPerformed** method and see what happens.

Other (from Learn Java GUI – Kidware)

Strings to Numbers to Strings

In Java GUI applications, string variables are used often. The **text** displayed in the label control and the text field control are string types. You will find you are constantly converting string types to numeric data types to do some math and then converting back to strings to display the information. Let’s look at each of these operations. First, from **string to number** - the process is:



To retrieve the text value in a control, use the **getText** method. In this example,

```
myString = myControl.getText();
```

To convert a string type to a numeric value, use the **valueOf** function. We will look at two examples. To convert a string (**myString**) to an **int** (**myInt**), use:

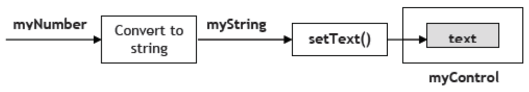
```
myInt = Integer.valueOf(myString).intValue();
```

To convert a string (**myString**) to a **double** type (**myDouble**), use:

```
myDouble = Double.valueOf(myString).doubleValue();
```

You need to be careful with these methods - if the string is empty or contains unrecognizable characters, an error will occur.

Now, the conversion process from **number to string** is:



There are two ways to convert a numeric variable to a string. The **valueOf** function does the conversion with no regard for how the result is displayed. This bit of code can be used to convert the numeric variable **myNumber** to a string (**myString**):

```
myNumber = 3.1415926;
myString = String.valueOf(MyNumber);
```

In this case, **myString** will be “**3.1415926**” - if you need to control the number of decimal points, the **format** function is used. As an example, to display **myNumber** with no more than two decimal points, use:

```
myNumber = 3.1415926;
myString = new DecimalFormat("0.00").format(MyNumber);
```

In the display string (“0.00”), the pound signs represent place holders. **myString** is now “**3.14**” Using this format function requires that the **java.text.*** package be imported into your application.

To set the text value displayed in a control, use the **setText** method. If you want to display **myString** in **myControl**, use:

```
myControl.setText(myString);
```

Unit 3F Labs

Beep Problem. Build an application with a single button. When the button is clicked, make the computer beep (use the **Toolkit.getDefaultToolkit().beep()** method).

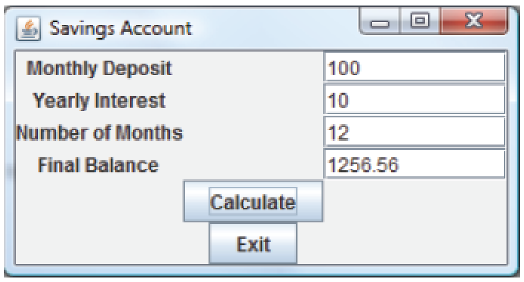
Text Problem. Build an application with a single button. When the button is clicked, change the button's **text** property. This allows a button to be used for multiple purposes. If you want to change the button caption back when you click again, you'll need an **if** statement. We'll discuss this statement in the next class, but, if you're adventurous, give it a try.

Enabled Problem. Build an application with two buttons. When you click one button, make it disabled (**enabled** = **false**) and make the other button enabled (**enabled** = **true**).

Date Problem. Build an application with a button. When the button is clicked, have the computer display the current date in a text field control. You'll need to study the **Date** class.

Savings Account The idea of this project is to determine how much you save by making monthly deposits into a savings account. The formula is $F = D\,[\,(1 + I)^M - 1]\,/\,I$ where

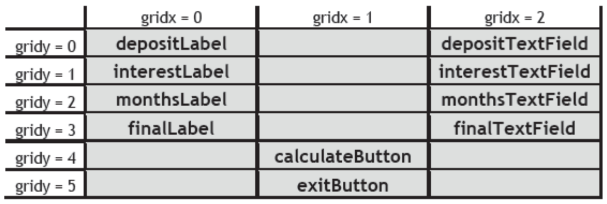
F - Final amount
D - Monthly deposit amount
I - Monthly interest rate
M - Number of months



Savings Account

Monthly Deposit	100
Yearly Interest	10
Number of Months	12
Final Balance	1256.56

We will place 4 labels, 4 text fields, and 2 buttons on the frame. The arrangement in the **GridBagLayout** will be.



	gridx = 0	gridx = 1	gridx = 2
gridy = 0	depositLabel		depositTextField
gridy = 1	interestLabel		interestTextField
gridy = 2	monthsLabel		monthsTextField
gridy = 3	finalLabel		finalTextField
gridy = 4		calculateButton	
gridy = 5		exitButton	

1. Create your form as shown above. Text Fields are 10 columns wide. Make sure it runs.
2. Code the **exitButton.**
3. Code the calculateButtonActionPerformed with the following:

```
private void calculateButtonActionPerformed(ActionEvent e)
{
  double deposit;
  double interest;
  double months;
  double finalBalance;
  double monthlyInterest;
  // read values from text fields
  deposit =
Double.valueOf(depositTextField.getText()).doubleValue();
  interest =
Double.valueOf(interestTextField.getText()).doubleValue();
  monthlyInterest = interest / 1200;
  months =
Double.valueOf(monthsTextField.getText()).doubleValue();
  // compute final value and put in text field;
  finalBalance = deposit * (Math.pow((1 +
monthlyInterest), months) - 1) / monthlyInterest;
  finalTextField.setText(new
DecimalFormat("0.00").format(finalBalance));
}
```

4. Remove the finalTextField and exitButton from the tab order with:
   finalTextField.setFocusable(false);
   exitButton.setFocusable(false);

Unit 3 Worksheet 1[5]

Show the output of each block of code below.

1. What is the output?

```
public class Quiz{
   public void one()
   {
      System.out.println("one");
   }
}

//code in the main of another class
Quiz test = new Quiz();
test.one();
```

```
one
```

2. What is the output?

```
public class Quiz{
   public void one()
   {
      System.out.println("one");
   }
}

//code in the main of another class
Quiz test = new Quiz();
test.one();
test.one();
```

```
one
one
```

3 What is the output?

```
public class QuizTwo{
   public void one()
   {
      System.out.println("one");
   }
   public void two()
   {
      System.out.println("two");
   }
}

//code in the main of another class
QuizTwo test = new QuizTwo();
test.two();
test.one();
test.one();
```

```
two
one
one
```

4. What is the output ?

```
public class QuizThree{
   public void one()
   {
      System.out.println("one");
   }
   public void two()
   {
      System.out.println("two");
      one();
   }
}

//code in the main of another class
QuizThree test = new QuizThree();
test.two();
test.one();
test.two();
```

```
two
one
one
two
one
```

[5] methodsworksheet.doc

Unit 3 Worksheet 2[6]

1. What is the output?

11

```
public class Quiz1{
   private int one=5, two=6, total;
   public void add(){  total = one + two;  }
   public void print(){  out.println(total);  }

   public static void main(String args[]){
      Quiz1 test = new Quiz1();
      test.add();
      test.print();
   }
}
```

2. What is the output?

0

```
public class Quiz2 {
   private int one=5, two=6, total;
   public void add(){  int total = one + two;  }
   public void print(){  out.println(total);  }

   public static void main(String args[]){
      Quiz2 demo = new Quiz2();
      demo.add();
      demo.print();
   }
}
```

3 What is the output?

26

```
public class Quiz3 {
   private int one=15, two=22, total;
   public void add(){
      one=4;
      total = one + two;
      two=8;
   }
   public void print(){  out.println(total);  }

   public static void main(String args[]){
      Quiz3 fun = new Quiz3();
      fun.add();
      fun.print();
   }
}
```

4. What is the output ?

44

```
public class Quiz4 {
   private int one,two,total;

   public void setNums(int n1, int n2){
      one=n1;
      two=n2;
   }
   public void add(){
      total = one + two;
   }
   public void print(){  out.println(total);  }

   public static void main(String args[]){
      Quiz4 run = new Quiz4();
      run.setNums(19,25);
      run.add();
      run.print();
   }
}
```

[6] methodsinstancevarsab.doc (quiz)

Unit 3 Worksheet 3[7]

1. What is the output?

```
public class Dog
{
   public void speak() {
      System.out.println("whoof");
   }
}

//code in the main of another class
Dog pup = new Dog();
pup.speak();
pup.speak();
pup.speak();
```

whoof
whoof
whoof

2 What is the output?

```
public class Cat
{
   public void speak() {
      System.out.println("meow");
   }
   public void sayName() {
      System.out.println("kitty");
   }
}

//code in the main of another class
Cat kit = new Cat();
kit.sayName();
kit.speak();
kit.speak();
kit.sayName();
```

kitty
meow
meow
kitty

3. What is the output ?

```
public class Alligator
{
   public void speak() {
      System.out.println("grrrr");
   }
   public void sayName() {
      System.out.println("gatuh");
      speak();
   }
}

//code in the main of another class
Alligator all = new Alligator();
all.speak();
all.sayName();
all.speak();
all.sayName();
```

grrrr
gatuh
grrrr
grrrr
gatuh
grrrr

4. Write a method named `printNameAddressCityState` that will print out your name, address, and city/state. Your name should appear on the first line, you address on the second line, and your city/state on the last line.

[7] methodsquizab.doc

Unit 3G

The Turtle
Let's play some more! Open Dr. Java, but you don't need a file open. You DO need to make sure that you have an **Extra Classpath** set to the *bookClasses* folder that you copied, and that the **Interactions Working Directory** is pointed directly to h:\java\projects or similar.

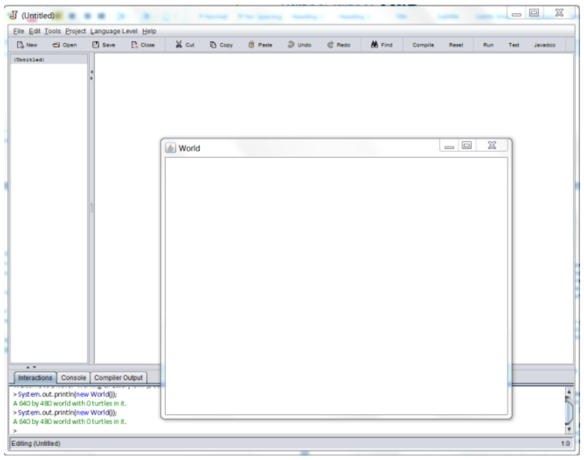
In the bookClasses folder are some classes that we can use.

In the *interactions* pane, enter

- System.out.println(new World());

The following will appear, along with a new 640 x 480 window.

A 640 by 480 world with a 0 turtles in it.


(Untitled)
File Edit Tools Project Language Level Help
New Open Save Close Cut Copy Paste Undo Redo Find Compile Reset Run Test Javadoc
World
Interactions Console Compiler Output
Editing (Untitled)

```
> System.out.println(new World());
A 640 by 480 world with 0 turtles in it.
> System.out.println(new World());
A 640 by 480 world with 0 turtles in it.
>
```

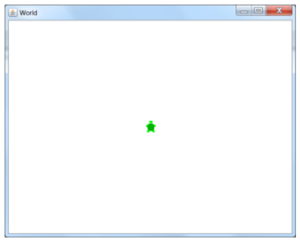
If you look in your bookClasses folder, there is a World.java and a World.class file. The class file is a result of compiling the java file. You can open the java file and have a look at it.

If we want to be able to manipulate our World, we need to store it in a variable.

- World worldOjb = new World(); //this is an object variable declaration
- System.out.println(worldObj);
  A 640 by 480 world with 0 turtles in it.

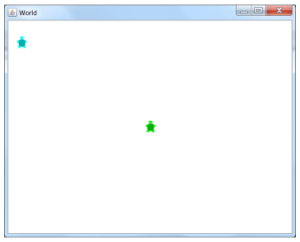
Now we will create a Turtle

- Turtle t1= new Turtle(worldObj);
- System.out.println(t1);
  No name turtle at 320, 240 heading 0.



The settings that describe the turtle are *default*. Unless you tell it otherwise, this is how it is created. Open Turtle.java and note the constructors. There are 4 constructors which means we can create a turtle 4 different ways.

- Turtle t2=new Turtle(30,50,worldObj);
- System.out.println(t2);
  No name turtle at 30,50, heading 0



When you opened Turtle.java up, you may also note that Turtle **extends** *SimpleTurtle.* This means that whatever a SimpleTurtle can do, so can a Turtle. Turtle has no methods for movement, but SimpleTurtle does.

- t1.foward(20);
- t1.turnLeft();
- tr.forward(30);
- t1.turnRight();
- t1.forward(40);
- t1.turn(-45);
- t1.forward(30);

Look at SimpleTurtle.java.

What other behaviors (methods) can your turtles perform?

Unit 3H

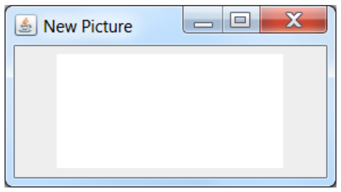
Working with JPG Images

We can create and manipulate media like pictures and sound.

- ➢ System.out.println(new Picture());
  Picture, filename null height 100 width 200

We have to ask pictures to show themselves using **show**().

- ➢ Picture p1 = new Picture();
- ➢ p1.show();



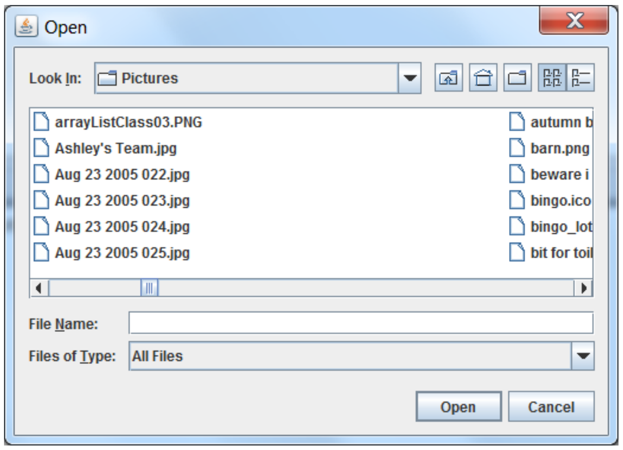
The default new Picture is 200 x 100 and white.

Creating a picture from a file

- ➢ System.out.println(FileChooser.pickAFile()):

An Open dialog box will appear. Locate your image, starting with Computer. A String will be returned with the path and filename of the image of choice (or sound, later on.)

H:\Pictures\Aug 23 2005 023.jpg



Caution!

Make sure the start of the returned filename path is a drive letter, and not a network mapping such as \\hsfs01.

Choose Computer and then follow the path from there.

We have to create a new Picture from our filename.

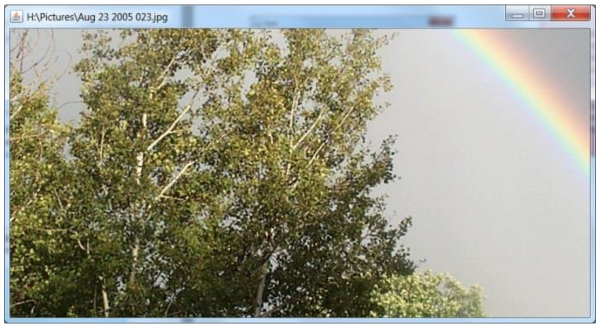
- ➢ System.out.println(new Picture(FileChooser.pickAFile()));
  Picture, filename H:\Pictures\Aug 23 2005 023.jpg height 1168 width 1760

This did create the picture, but we still can't see it.

- ➢ new Picture(FileChooser.pickAFile()).show();

or

- ➢ String picName=FileChooser.pickAFile();
- ➢ Picture picObject =new Picture(picName);
- ➢ picObject.show();



Unit 3I

Playing a Wav File
This is very similar to showing a picture. Try each of these. The first 2 won't produce any sound, however.

- System.out.println(FileChooser.pickAFile());

```
H:\Music\Sound Effects\asteroids\asteroids_tonehi.wav
```

- System.out.println(new Sound(FileChooser.pickAFile()));

```
Sound file: H:\Music\Sound Effects\asteroids\asteroids_tonehi.wav number of samples: 1915
```

- new Sound(FileChooser.pickAFile()).play();

Naming Your Media

```
String filename = FileChooser.pickAFile();
Sound soundObj = new Sound(filename);
soundObj.play();
```

Unit 3G Labs

Create a World object and a Turtle object and use the Turtle to draw your choice of:

a. a star
b. an arrow
c. a pyramid
d. a flower
e. your first name

Call the teacher over for marking.

Unit 3 Assignment 1

Easter

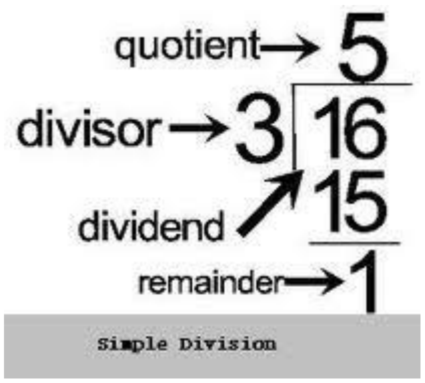
The naming of variables is very important. The names should be descriptive and to the point. However, when working with any type of algorithm or formula that may be presented to you, it is always better to use the variable names used by the client. This reduces potential confusion that may arise if the client named something "a" and you renamed it "netIncome". This becomes quite clear in this assignment because if you try to change the names that are given, you would have to remember to translate from one variable name to the other.

A convenient algorithm for determining the date of Easter in a given year was devised in 1876 and first appeared in Butcher's Ecclesiastical Handbook. This algorithm holds for any year in the Gregorian calendar, which means years including and after 1583. Subject to minor adaptations, the algorithm is as follows:

1. Let y be the year (such as 1583 or 2003).
2. Divide y by 19 and call the remainder a. Ignore the quotient.
3. Divide y by 100 and get a quotient b and a remainder c.
4. Divide b by 4 and get a quotient d and a remainder e.
5. Divide b + 8 by 25 and get a quotient f. Ignore the remainder.
6. Divide b - f + 1 by 3 and get a quotient g. Ignore the remainder.
7. Divide 19 * a + b - d - g + 15 by 30 and get a remainder h. Ignore the quotient.
8. Divide c by 4 and get a quotient i and a remainder k.
9. Divide 32 + 2 * e + 2 * i - h - k by 7 and get a remainder r. Ignore the quotient.
10. Divide a + 11 * h + 22 * r by 451 and get a quotient m. Ignore the remainder.
11. Divide h + r - 7 * m + 114 by 31 and get a quotient n and a remainder p.

The value of `n` gives the month (3 for March and 4 for April) and the value of `p + 1` gives the day of the month. For example, if `y` is 2003:

```
a = 8
b = 20
c = 3
d = 5
e = 0
f = 1
g = 6
h = 26
i = 0
k = 3
r = 3
m = 0
n = 4
p = 19
```



Therefore, in 2003, Easter fell on April 20 (month = n = 4 and day = p + 1 = 20).

Assignment:

1. Write a program to solve for the day that Easter falls on for a given year.
2. The program should display the values for all of the variables and the date for Easter. A Sample run output for the year 2003 would be:

```
a = 8
b = 20
c = 3
d = 5
e = 0
f = 1
g = 6
h = 26
i = 0
k = 3
r = 3
m = 0
n = 4
p = 19

Easter in 2003 falls on 4/20
```

3. Verify that the program gives the correct date of Easter for the current year.

Instructions:

1. After completing the program, print your source code and a run output to the printer.
2. Make sure your name is documented near the top of your source code.

This program has been started for you.

```
public class Easter{
private int y;

public Easter(int year){
y = year;
}

public void calculate(){
int a = y % 19;
System.out.println("a = " + a);
int b = y / 100;
System.out.println("b = " + b);
...
```

More help on this assignment is available in the **chapter 4 case study** of the LVP book available to you in hard copy and on the shared drive.

Test on Unit 3

Check for Theory Review on Blackboard