CS415
INTRODUCTION TO COMPUTER SCIENCE
FALL 2017

5-METHODS AND VARIABLES
CHAPTER 1

IMPORTS

- In Java a type of object is defined by a Class.
- At first we will use objects define by classes that someone else wrote.
- The first line imports all the classes from "wheelsunh.users"
- The second line imports only the class "Color" from the library "java.awt"
- wheelsunh is used only in this class whereas java.awt is a standard java library

```
import wheelsunh.users.*;
import java.awt.Color;
```


LIBRARY API'S

- A libraries provides an Application Programming Interface (API).
- An API is a web page that describes how to use the classes in the library
- The first line imports all the classes from "wheelsunh.users
- The second line imports only the class "Color" from the library "java.awt"
- wheelsunh is used only in this class whereas java.awt is a standard java library

```
import wheelsunh.users.*;
import java.awt.Color;
```


ANATOMY REVIEW

imports
class comment
class header
class body

```
import wheelsunh.users.*;
import java.awt.Color;

/**
 * Lab1.java:  Displays a red circle.
 * @author Mark Bochert
 */
public class Lab1
{
    private Ellipse _circle;

    /**
     *   Constructor for the Lab1 class.
     */
    public Lab1( )
    {
        _circle = new Ellipse( Color.RED );
    }

    /** main program creates a Frame and
      *     invokes the class constructor.
      *
      * @param  args  the command line.
      */
    public static void main( String[] args )
    {
        Frame f = new Frame();
        Lab1 app = new Lab1();
    }
} //End of Class Lab1
```

instance variable
constructor comment
constructor header
constructor body
main method comment
main method header
main method body

VARIABLES

- Remember, a program consists of Data (things to be manipulated) and Algorithms (instructions on how to manipulate the data)
- We represent the data with Variables which store the data in memory.
- RULE: Variables need to be declared before they are used.
- The declaration needs to specify the type of data (Ellipse, Rectangle, int)
- And the name (rightArm, head, width).
- Thats why we write:

```
private Ellipse _circle;
```


VARIABLE ASSIGNMENT

```
private Ellipse _circle;
```

- Once we create the variable we still have no object.
- We still need to create an object and associate the variable with it.

```
_circle = new Ellipse( );
```

- The "=" token is called the assignment operator.
- The variable must be on the left and the object on the right.
- The assignment changes the variable so that it now refers to the object.

INSTANCE VARIABLES

```
import wheelsunh.users.*;
import java.awt.Color;

/**
 * Lab1.java:  Displays a red circle.
 * @author Mark Bochert
 */
public class Lab1
{
    private Ellipse _circle;

    /**
     *   Constructor for the Lab1 class.
     */
    public Lab1( )
    {
      _circle = new Ellipse( Color.RED );
    }

    /** main program creates a Frame and
      *  invokes the class constructor.
      *
      * @param  a the command line.
      */
    public static void main( String[] a)
    {
       Frame f = new Frame();
       Lab1 app = new Lab1();
    }
} //End of Class Lab1
```

- A type of variable that is available in all parts of a class is called an instance variable.
- Notice that it is declared inside, at the beginning of the class body but outside any method.
- The private key word means it is not available outside the class.
- The type is: Ellipse
- The name is _circle
- The declaration ends with ;

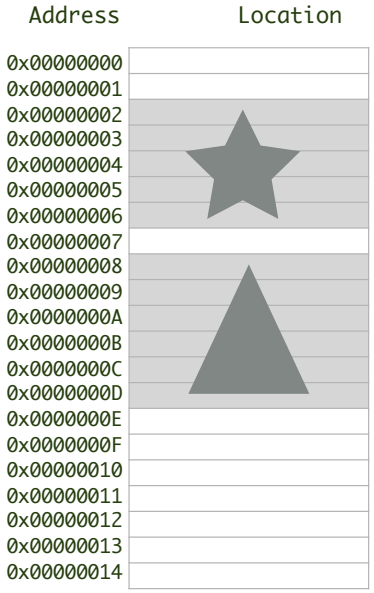
VARIABLES IN MEMORY

- Objects are stored in primary memory.
- Programmers need models of memory that will help us understand the program's behavior.
- Memory consists of a set of consecutively numbered storage locations, each containing one byte.
- The number is called the address of the location

Address	Location
0x00000000	
0x00000001	
0x00000002	
0x00000003	
0x00000004	
0x00000005	
0x00000006	
0x00000007	
0x00000008	
0x00000009	
0x0000000A	
0x0000000B	
0x0000000C	
0x0000000D	
0x0000000E	
0x0000000F	
0x00000010	
0x00000011	
0x00000012	
0x00000013	
0x00000014	

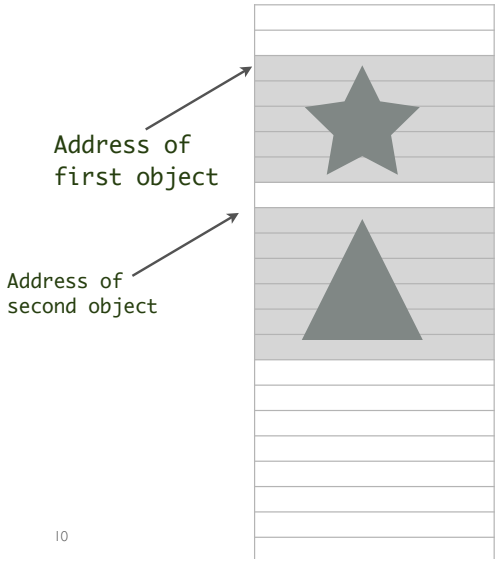
OBJECTS IN MEMORY

- Objects stored in memory
- 0x00000002 is the address of one object
- 0x00000008 is the address of the other.



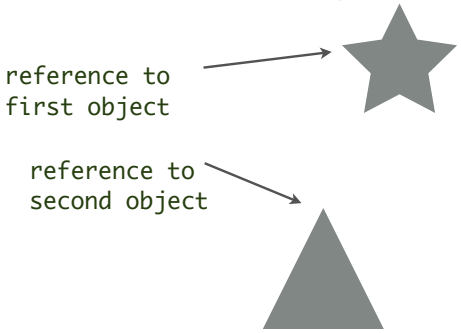
OBJECTS IN MEMORY (MORE ABSTRACTLY)

- We usually don't need to know the value of the address,
- but only that it refers (points) to a particular object
- The numbers have been simplified into arrows



OBJECTS IN MEMORY (EVEN MORE ABSTRACTLY)

- We don't always need to know the organization of memory,
- but simply that they are somewhere in memory
- and we have references to them



VARIABLE DECLARATION

```
private Ellipse _circle;
```

_circle

null

- A variable declaration allocates a memory location.
- The name is associated with the location
- the variable is "null"
- "We have created a box in memory named _circle"

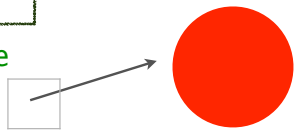
CONSTRUCTOR CALL

```
new Ellipse();
```

- The constructor creates the object in memory.
- We have no idea where in memory!

VARIABLES AND REFERENCES

```
circle = new Ellipse();
```



- The assignment assigns the location to our variable.
- We say that the variable contains a reference to the object.
- The variable does not contain the object.
- This has implications as to how variable assignment works

METHODS

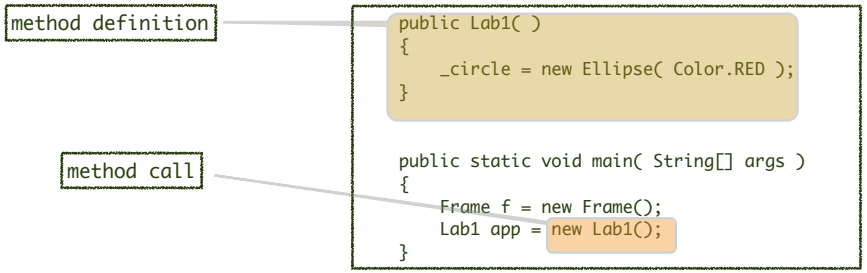
- Think of the methods as the "chapters" in the Class.
- Our first few classes have only 2 methods.
- To write a simple program we might simply add new instance variables and new code to the body of the constructor.
- At some point this will make the constructor method too long and complicated. (A rule of thumb says that you should be able to see the entire method without scrolling).
- Programming languages provide ways to break large complicated programs into independent modules or "chunks".
- In Java, if we have a method that is too long we can break it down into several smaller methods.

METHODS

- There are three categories of methods
  - Constructors: Used to create a new object and initialize its state
  - mutators (setters): Used to change the state of an object
  - accessors (getters): Used to access the state

METHODS

- There are two aspects of a method that interact
  - method definition: Specifies how the method is used and what it does
  - method call (invocation): Cause the method to execute



PARAMETERS

- Method parameters specify extra information that can be added to a message.
- Take a look at some of the methods available for the shapes in wheelsunh (page 71). Some have extra specifications inside the parentheses.
- These are parameters.

```
public void setColor( Color aColor )
public void setSize( int x, int y )
public void setRotation( int r )
```


PARAMETERS

```
public void setColor( Color aColor )
public void setSize( int x, int y )
public void setRotation( int r )
```

- setColor has one parameter with type Color
- setSize has two parameters, both of type int (integers)
- setRotation has one int parameter
- What is the syntax for calling a method with parameters?

PARAMETERS

```
public void setColor( Color aColor )
public void setSize( int x, int y )
public void setRotation( int r )
```

- The definition specifies the name and type and number of formal parameters
- the call must use the correct name and supply the correct number and type of actual parameters

```
_rectangle.setColor( Color.RED )
_rectangle.setSize( 50, 25 )
_rectangle.setRotation( 180 )
```

PARAMETERS

receiver

```
public void setColor( java.awt.Color aColor )
{
...
}
```

f(x) = 2x + 3

Formal parameter

- The formal parameters that are specified in the method definition (receiver)...
- Must be matched by the actual parameters provided in the method call (sender)

Actual parameter

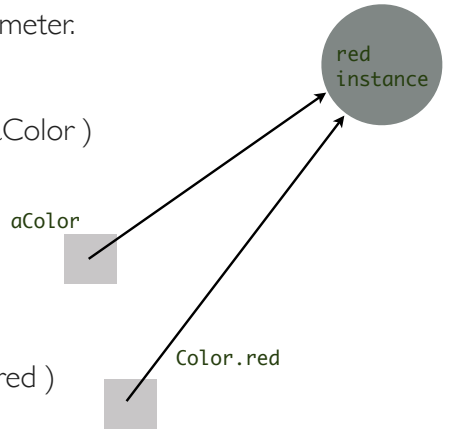
```
myRectangle.setColor( java.awt.Color.blue );
```

f(5)

sender

MEMORY REVEALED

- After the method call, the formal parameter is a variable that refers to the same instance as the actual parameter.
- Formal Parameter (receiver)
  public void setFrameColor(Color aColor)
  { .....
- Actual parameter (sender).
  myRectangle.setFrameColor(Color.red)



CONSTRUCTORS

- There are three types of methods: Constructors, mutators, and accessors
- Constructors initialize objects

constructor definition

constructor call

```
public Lab1( )
{
    _circle = new Ellipse( Color.RED );
}

public static void main( String[] args )
{
    Frame f = new Frame();
    Lab1 app = new Lab1();
}
```


WRITING A METHOD

- Write a program that draws a white square in a blue circle

```
public class Example
{
   private Ellipse _circle;
   private Rectangle _square;

   //Magic Numbers
   private int _x = 100;
   private int _y = 200;
   private int _circleSize = 40;
   private int _squareSize = 20;
   private int _offset = ( _circleSize - _squareSize ) / 2;

  public Example( )
    {
       _circle = new Ellipse( Color.blue );
       _square = new Rectangle( Color.white );
       _circle.setSize( _circleSize, _circleSize );
       _square.setSize( _squareSize, _squareSize );
       _circle.setLocation( _x, _y );
       _square.setLocation( _x + _offset, _y + _offset );
    }


    public static void main( String[] a)
    {
       Frame f = new Frame();
       Example app = new Example();
    }
}
```

MAGIC NUMBERS?

```
//Magic Numbers
private int _x = 100;
private int _y = 200;
private int _circleSize = 40;
private int _squareSize = 20;
// offset needed to center the square in the circle
private int _offset = ( _circleSize - _squareSize ) / 2;
```

- Many numbers in your code are more than just numbers.
- In this example, 40 is not just a number, it is the size of the circle. We call it a magic number.
- Creating a variable for each magic number makes the code easier to read, less error prone, and easier to modify.

INTEGERS

- Numbers are treated differently than objects in Java; we call them primitive types.
- The type of integer that we use most often is `int`
- Note the lower case; this is not a class it is a primitive type.
- Primitive types have no methods and no properties.

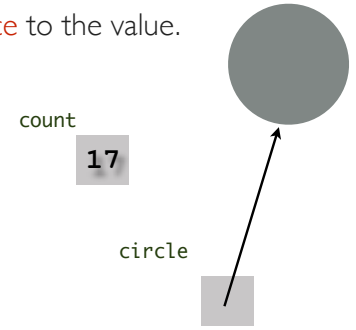
SOME JAVA INT OPERATIONS

- We can add two integers
  2 + 2 is 4
- We can subtract one integer from another
  4 - 3 is 1
- We can multiply two integers
  2 * 3 is 6
- We can divide one integer by another
  5 / 2 is 2 (the integer part of the real number division)

MEMORY REVEALED

- Primitive types and Class types are handled differently in memory.
- A primitive type variable contains the value.
- An object type variable contains a reference to the value.

```
count = 17;
circle = new Ellipse();
```



WRITING A METHOD

- Suppose we want two shapes?

```
public class Example
{
   private Ellipse _circle;
   private Rectangle _square;

   //Magic Numbers
   private int _x = 100;
   private int _y = 200;
   private int _circleSize = 40;
   private int _squareSize = 20;
   private int _offset = ( _circleSize - _squareSize ) / 2;

  public Example( )
    {
       _circle = new Ellipse( Color.blue );
       _square = new Rectangle( Color.white );
       _circle.setSize( _circleSize, _circleSize );
       _square.setSize( _squareSize, _squareSize );
       _circle.setLocation( _x, _y );
       _square.setLocation( _x + _offset, _y + _offset );

       _x = 300;
       _y = 400;

       _circle = new Ellipse( Color.blue );
       _square = new Rectangle( Color.white );
       _circle.setSize( _circleSize, _circleSize );
       _square.setSize( _squareSize, _squareSize );
       _circle.setLocation( _x, _y );
       _square.setLocation( _x + _offset, _y + _offset );
}

    public static void main( String[] a)
    {
       Frame f = new Frame();
       Example app = new Example();
    }
}
```


WRITING A METHOD

- Move common code into a method

```
public class Example
{
   private Ellipse _circle;
   private Rectangle _square;

   //Magic Numbers
   private int _x = 100;
   private int _y = 200;
   private int _circleSize = 40;
   private int _squareSize = 20;
   private int _offset = ( _circleSize - _squareSize ) / 2;

   public Example( )
   {
        makeShape( _x, _y );
        makeShape( 300, 400 );
   }

   private void makeShape( int x, int y )
   {
       _circle = new Ellipse( Color.blue );
       _square = new Rectangle( Color.white );
       _circle.setSize( _circleSize, _circleSize );
       _square.setSize( _squareSize, _squareSize );
       _circle.setLocation( x, y );
       _square.setLocation( x + _offset, y + _offset );
    }

    public static void main( String[] a)
    {
       Frame f = new Frame();
       Example app = new Example();
    }
}
```

- private: This method is not needed outside this class
- void: this is a mutator (setter or procedure)
- int x, int y: Formal parameters this can only be used in this function
- Method comment has been left out to fit on slide

WRITING A METHOD

```
/**
*   Constructor.
*/
 public Example( )
 {
      makeShape( _x, _y );
      makeShape( 300, 400 );
 }


/**
* Makes a shape at x,y.
*
* @param  x int
* @param y int
*/
 private void makeShape( int x, int y )
 {
     _circle = new Ellipse( Color.blue );
     _square = new Rectangle( Color.white );
     _circle.setSize( _circleSize, _circleSize );
     _square.setSize( _squareSize, _squareSize );
     _circle.setLocation( x, y );
     _square.setLocation( x + _offset, y + _offset );
  }
```

- Remember you need comments on methods

LOCAL VARIABLES

```
/**
* Makes a shape at x,y.
*
* @param  x int
* @param y int
*/
 private void makeShape( int x, int y )
 {
     int offset  = ( _circleSize - _squareSize ) / 2;

     _circle = new Ellipse( Color.blue );
     _square = new Rectangle( Color.white );
     _circle.setSize( _circleSize, _circleSize );
     _square.setSize( _squareSize, _squareSize );
     _circle.setLocation( x, y );
     _square.setLocation( x + offset, y + offset );
  }
```

- x and y are formal parameters
- They can only be used in this method.
- We say they are local to the method.
- Other local variables can be created.
- In general, variables that aren't needed in more than one method should be local rather than instance.