

Object Oriented Programming

- ❖ **Procedural design and programming**
 - ◆ The focus of the class so far
 - ◆ Primarily used for sequential event programming and methods
- ❖ **Object oriented design and programming**
 - ◆ **Graphical User Interface (GUI)** uses this approach
 - ◆ **Unified Modeling Language (UML)** diagrams for design
 - ◆ Program Design Phase to describe OOD Structure of Program
 - ◆ Program modeling language has its own set of rules and notations
 - ◆ Programming language independent so use with any OOP language
 - ◆ Java is a Object Oriented Programming Language
 - ◆ Classes can be considered templates for an object
 - ◆ Objects can be created from classes and contain data
 - ◆ Methods are procedural processes that can act upon objects
- ❖ **But first some loose ends...**

Copyright © 2012 R.M. Laurie 1

Java Fully Supports Unicode

- ❖ Java Defaults to encoding of **source file** filename.java
 - ◆ To utilize unicode change encoding to **UTF-8** from cp1252
 - ◆ Eclipse encoding can be changed to go to UTF-8
Window-->Preferences-->General-->Workspace
Change **Text File Encoding** to UTF-8
 - ◆ Works in Swing but not in DOS console

```

1. import javax.swing.*;
2. public class Unicode
3. {
4.     public static void main(String[] args)
5.     {
6.         char[] cSuit = {'\u2660', '\u2665', '\u2666', '\u2663'};
7.         String sDisplay = "日本国 山口県 岩国市\n";
8.         int 日 = 1, 月 = 30, 年 = 365;
9.         for(int nI=0; nI < cSuit.length; nI++)
10.            sDisplay += cSuit[nI] + " ";
11.            日 = 10 * 日 + 6 * 月 + 1 * 年;
12.            sDisplay += "\n日 = 10 日 + 6 月 + 1 年 = " + 日;
13.            JOptionPane.showMessageDialog(null, sDisplay);
14.        }
15.    }
    
```



Copyright © 2012 R.M. Laurie 2

```

1. import javax.swing.*;
2. public class NichiCalcJA
3. {
4.     public static void main(String[] args)
5.     {
6.         String sOutput = "", sHome = "\n日本国・山口県・岩国市";
7.         String sEntry, sAuthor = "\n著者: ラウリ・ロベト";
8.         int 日, 月, 年, 合計日, nYorN;
9.         JOptionPane.showMessageDialog(null, "このプログラムは日が経過計算"
10.            + sAuthor + sHome);
11.         do
12.         {
13.             sEntry = JOptionPane.showInputDialog("何年", '0');
14.             年 = Integer.parseInt(sEntry);
15.             sEntry = JOptionPane.showInputDialog("何ヶ月", '0');
16.             月 = Integer.parseInt(sEntry);
17.             sEntry = JOptionPane.showInputDialog("何日", '0');
18.             日 = Integer.parseInt(sEntry);
19.             合計日 = 日 + 30 * 月 + 365 * 年;
20.             sOutput = 年 + "年" + " + " + 月 + "月" + " + " + 日 + "日 = " + 合計日
21.                + "合計日";
22.             JOptionPane.showMessageDialog(null, sOutput);
23.             nYorN = JOptionPane.showConfirmDialog(null, "再度実行したいですか",
24.                "オプションの選択", JOptionPane.YES_NO_OPTION);
25.             }while(nYorN == JOptionPane.YES_OPTION);
26.         }
27.     }
    
```

Kanji can be used for Identifiers but not recommended

Are those parameters in main()?

- ❖ **public static void main(String[] args)**
 - ◆ Contains a parameter list which can be utilized when running in command line or calling main method from another method
 - ◆ String arguments can be passed to main method because parameter is array of strings
 - ◆ Note Strings must be converted to numbers in Example below

```

1. public class CylinderVolume
2. {
3.     public static void main(String[] args)
4.     {
5.         double dDiameter = Double.parseDouble(args[0]);
6.         double dHeight = Double.parseDouble(args[1]);
7.         double dVolume = Math.PI * Math.pow(dDiameter/2, 2)
8.            * dHeight;
9.         System.out.println("Cylinder Volume = " + dVolume );
10.    }
11. }
    
```

C:_java\CmdLineArgs\bin>java CylinderVolume 6 2
Cylinder Volume = 56.548667764616276

Copyright © 2012 R.M. Laurie 4

main() call from another class method

```

1. public class CircleArea {
2.     public static double main(String[] args) {
3.         double dRadius = Double.parseDouble(args[0]);
4.         double dArea = Math.PI * Math.pow(dRadius , 2);
5.         return dArea;
6.     }
7. }

```

```

1. public class CylinderVolume2 {
2.     public static void main(String[] args) {
3.         double dDiameter = Double.parseDouble(args[0]);
4.         double dHeight = Double.parseDouble(args[1]);
5.         String[] sRadius = {Double.toString(dDiameter/2)};
6.         double dVolume = CircleArea.main(sRadius) * dHeight;
7.         System.out.println("Cylinder Volume = " + dVolume );
8.     }
9. }

```

C:_java\CmdLineArgs\bin>java CylinderVolume2 6 2
Cylinder Volume = 56.548667764616276

Copyright © 2012 R.M. Laurie 5

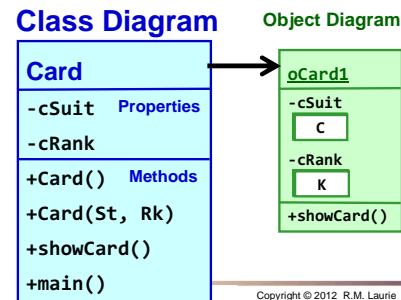
Object Oriented Program Development

- ❖ Object-oriented programming
 - ◆ Emphasis on object attributes and object behaviors
- ❖ Object Model objects are real world entities (nouns)
 - ◆ **Data Fields** (Properties, Attributes, **Instance variables**)
 - ◆ Identifier is usually a noun
 - ◆ Access using Object.Variable = . Object member access operator
 - ◆ **Behaviors** (Actions, **Methods**)
 - ◆ Identifier is verbNoun combination
 - ◆ Access using Object.Method = . Object member access operator
 - ◆ **Get methods** read from instance variables (accessor or getter)
 - ◆ **Set methods** write to instance variables (mutator or setter)
 - ◆ **Constructor methods** create new objects (Same name as class)
- ❖ **Classes**
 - ◆ Template for objects sometimes called contracts (noun)
 - ◆ Like blueprint or recipe from which objects are created
 - ◆ Objects are created or **instantiated** from class
 - ◆ Objects are considered **instances** of a class

Copyright © 2006 R.M. Laurie 6

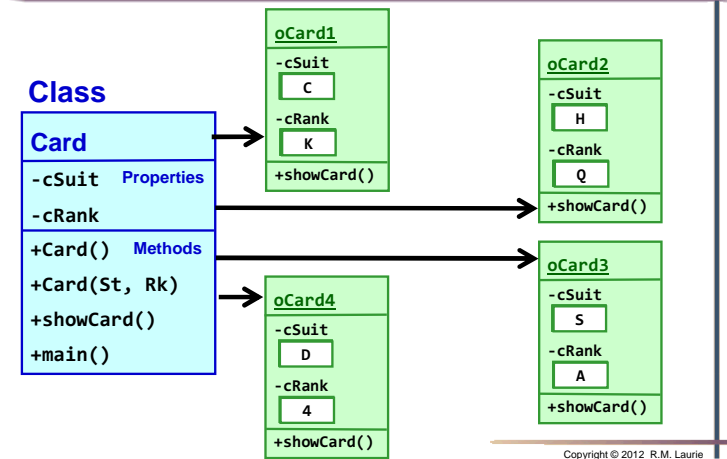
Introduction to UML

- ❖ **Unified Modeling Language (UML)** class diagrams
 - ◆ Graphical design tool for Object Oriented Design
 - ◆ Models and describes OOD Structure of Program
 - ◆ Should do before coding can also help reverse-engineering
- ❖ **Object** represents an entity with noun identifier
 - ◆ **Data Fields** (noun)
 - ◆ Properties
 - ◆ Attributes
 - ◆ Instance variables
 - ◆ **Behaviors** (verbNoun)
 - ◆ Actions
 - ◆ Methods
 - ▶ Constructor?
 - ▶ Get methods?
 - ▶ Set methods?
- ❖ **Class**
 - ◆ Template for objects
 - ◆ Objects **instantiated** from class



Copyright © 2012 R.M. Laurie 7

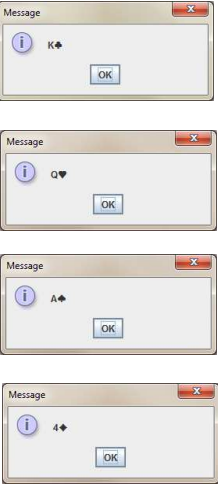
UML Diagram of Instantiated Objects



Copyright © 2012 R.M. Laurie 8

```

1. import javax.swing.JOptionPane;
2. public class Card {
3.     // PROPERTIES IN CLASS ARE INSTANCE VARIABLES
4.     private char cSuit;
5.     private char cRank;
6.     // METHODS IN CLASS
7.     // Application Launcher main Method
8.     public static void main(String[] args) {
9.         Card oCard1 = new Card('K', 'C');
10.        Card oCard2 = new Card('Q', 'H');
11.        Card oCard3 = new Card('A', 'S');
12.        Card oCard4 = new Card('4', 'D');
13.        oCard1.showCard(); // Object Method call
14.        oCard2.showCard(); // Object Method call
15.        oCard3.showCard(); // Object Method call
16.        oCard4.showCard(); // Object Method call
17.    }
18.    // Constructor Method has same name as class
19.    public Card( char Rank, char Suit ){
20.        cRank = Rank;
21.        cSuit = Suit;
22.    }
23.    // showCard Method displays card
24.    public void showCard(){
25.        String sOut;
26.        sOut = String.valueOf(cRank);
27.        if(cSuit == 'S') sOut += '\u2660';
28.        else if(cSuit == 'H') sOut += '\u2665';
29.        else if(cSuit == 'D') sOut += '\u2666';
30.        else if(cSuit == 'C') sOut += '\u2663';
31.        JOptionPane.showMessageDialog(null, sOut);
32.    }
33. }
    
```



Class Methods

- ❖ **Classes provide methods to:**
 - ◆ **Get methods** read from instance variables (accessor or getter)
 - ◆ Access instance variable values
 - ◆ Display instance variable values
 - ◆ **Set methods** write to instance variables (mutator or setter)
 - ◆ Change instance variable values
 - ◆ **Constructor methods** create new objects from class
 - ◆ Same name as class
- ❖ **Format of method header:**

```
public returnType methodName(parameterList)
```

 - ◆ **public** = Method can be used in other classes
 - ◆ **returnType**
 - ◆ Method returns a value (output) of specified data type
 - ◆ **void** means nothing returned
 - ◆ **methodName** = Method Identifier
 - ◆ **parameterList** = Method input values

Copyright © 2012 R.M. Laurie | 10

Constructor Methods

- ❖ Method has same name as class
- ❖ Called each time object is created
- ❖ Purpose:
 - ◆ Initialize new object's instance variables

```

public class MethodExample1
{
    // Data declaration section
    private String sMessage;
    // Methods definition section
    public MethodExample1()
    {
        sMessage = "I like Java";
    }
}
    
```

Copyright © 2012 R.M. Laurie | 11

Get Methods and Set Methods

- ❖ **Get methods or Accessors method** read values stored in object's instance variables


```
public void displayMessage()
{
    JOptionPane.showMessageDialog(
        null, sMessage);
}
```
- ❖ **Set Methods or Mutator Methods** modify object's data variables after object is created


```
public void changeMessage(String sNewMsg)
{
    sMessage = sNewMsg;
}
```

Copyright © 2012 R.M. Laurie | 12

```

1. import javax.swing.*;
2. public class Message
3. {
4.     // Data declaration section
5.     private String sMessage;
6.     // Methods definition section
7.     public Message()
8.     {
9.         sMessage = "I like Java";
10.    }
11.    public void displayMessage()
12.    {
13.        JOptionPane.showMessageDialog(null,sMessage);
14.    }
15.    public void changeMessage(String sNewMsg)
16.    {
17.        sMessage = sNewMsg;
18.    }
19.    public static void main(String[] args)
20.    {
21.        Message oMessageOne;
22.        oMessageOne = new Message();
23.        oMessageOne.displayMessage();
24.        oMessageOne.changeMessage("I prefer Pepsi");
25.        oMessageOne.displayMessage();
26.    }
27. }
    
```

Where are the Constructors, Get Methods, Set Methods, and Application Launcher methods?

Visibility Modifier

NONE USED

- ◆ Package-access
- ◆ Class or method can be accessed within package
- ◆ package packageName; // first line in file and same folder

public

- ◆ Any other class method may invoke a *public method*
 - ◆ This includes within package or from imported packages
- ◆ Any other class or method can directly access or change a *public instance variable*
 - ◆ Similar to global variable and should be avoided in program

private

- ◆ Only a method in the same class can invoke a *private method*
- ◆ Only a method in the same class can access or change a *private instance variable*
- ◆ Class and method variables should be **private** to prevent inappropriate changes.

Copyright © 2012 R.M. Laurie 14

Static and Non-Static

- ❖ **Non-static** methods access through objects
 - ◆ Data located in object
 - ◆ Syntax: dataType objectName.methodName(parameters);
 - ◆ Examples:
 - ◆ oCard1.showCard();
 - ◆ nLength = sEntry.length();
 - ◆ c1st = sEntry.charAt(0);
- ❖ **Static** methods accessed through class
 - ◆ Cannot operate on an object only class access
 - ◆ Receives all data as arguments
 - ◆ Syntax: dataType ClassName.methodName(parameters);
 - ◆ Example:
 - ◆ JOptionPane.showMessageDialog(null,"Wakeup");
 - ◆ nScore = Integer.parseInt(sEntry);
- ❖ **Static variables and constants**
 - ◆ Associated with class and not replicated with each object
 - ◆ Shared variable or constant among objects

Copyright © 2012 R.M. Laurie 15

Programming Style

- ❖ Use accepted programming style
 - ◆ Makes programs easy to read
 - ◆ Minimizes mistakes
 - ◆ Consistent identifier naming convention
 - ◆ Java ignores whitespace so add as much as you want so code is easy to read and understand
 - ◆ Write descriptive variable and method identifiers
- ❖ Multiple classes can be created in one java file
 - ◆ Only one **main** method for each Java file
 - ◆ **main** method is the application launcher
 - ◆ **main** method can be placed anywhere in file

Copyright © 2012 R.M. Laurie 16

Creating Objects

- ❖ **Objects**
 - ◆ Contains the *Instance Variables* declared in data declaration section
- ❖ **Reference variable Declaration**
 - ◆ Reference location for object's values and methods
 - ◆ `Card oCard1;`
- ❖ **new Dynamic Memory Allocation operator**
 - ◆ For *creating an instance* or *instantiating an object*
 - ◆ `Card oCard1 = new Card();`
- ❖ **Instance Methods**
 - ◆ Provide operations that can be applied to objects
- ❖ **Static Methods**
 - ◆ Class accessed, object independent, and general purpose functions

Copyright © 2012 R.M. Laurie 17

Object Method Call

- ❖ **Method Call** invoked upon an object
 - ◆ **Arguments** are values passed to method
 - ◆ Must match parameter position and dataType
 - ◆ Variables pass contents of variable called pass-by-value
 - ◆ **Overloaded Methods**
 - ◆ Have same identifier name but different parameter lists
 - ◆ Parameter dataType may also determine method call
 - ◆ **return value**
 - ◆ Output value of the method
 - ◆ Can only return one thing
 - ◆ Method can have multiple return statements
 - ◆ First return statement reach returns control to call location
- ❖ **Examples:**

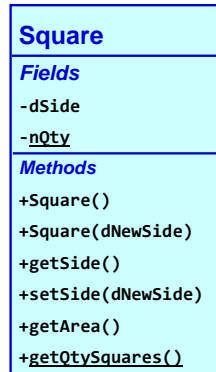
```
oSquare1.setSide(12);
oCard4.showCard();
```

Copyright © 2012 R.M. Laurie 18

Design using UML then Implement

```
1. public class Square {
2.     private double dSide = 1; // default is 1
3.     private static int nQty = 0;
4.     public Square() {
5.         nQty++;
6.     }
7.     public Square(double dNewSide) {
8.         dSide = dNewSide;
9.         nQty++;
10.    }
11.    public double getSide(){
12.        return dSide;
13.    }
14.    public void setSide(double dNewSide){
15.        dSide = dNewSide;
16.    }
17.    public double getArea() {
18.        return dSide * dSide;
19.    }
20.    public static int getQtySquares(){
21.        return nQty;
22.    }
23. }
```

Class UML Diagram



Copyright © 2012 R.M. Laurie 19

```
1. public class Square {
2.     private double dSide = 1; // default is 1
3.     private static int nQty = 0;
4.     public Square() {
5.         nQty++;
6.     }
7.     public Square(double dNewSide) {
8.         dSide = dNewSide;
9.         nQty++;
10.    }
11.    public double getSide(){
12.        return dSide;
13.    }
14.    public void setSide(double dNewSide){
15.        dSide = dNewSide;
16.    }
17.    public double getArea() {
18.        return dSide * dSide;
19.    }
20.    public static int getQty(){
21.        return nQty;
22.    }
23. }
24.
25. class TestSquare {
26.     public static void main(String[] args) {
27.         System.out.println("Objects=" + Square.getQty());
28.         Square oSq1 = new Square();
29.         System.out.println("Objects="+Square.getQty()+" | Area1="+oSq1.getArea());
30.         Square oSq2;
31.         oSq2 = new Square(2);
32.         System.out.println("Objects="+Square.getQty()+" | Area2="+oSq2.getArea());
33.         Square oSq3 = new Square(3);
34.         System.out.println("Objects="+Square.getQty()+" | Area3="+oSq3.getArea());
35.         oSq3 = oSq2;
36.         System.out.println("Objects="+Square.getQty()+" | Area3="+oSq3.getArea());
37.         oSq2.setSide(5);
38.         System.out.println("Objects="+Square.getQty()+" | Area2="+oSq2.getArea());
39.     }
}
```

Launch App from TestSquare

Two classes in one file
 Square.java is public
 Note: TestSquare class not public
 Only one main method per file
 Only one public class per file

Objects=0
 Objects=1 | Area1=1.0
 Objects=2 | Area2=4.0
 Objects=3 | Area3=9.0
 Objects=3 | Area3=4.0
 Objects=3 | Area2=25.0

```

1. public class Poker5Stud {
2.     public static void main(String[] args) {
3.         int[] nDeck = new int[52];
4.         initializeCards(nDeck);
5.         shuffleCards(nDeck);
6.         System.out.print("Dealer Hand: ");
7.         for(int nD = 0; nD < 5; nD++)
8.             Card.display(nDeck[nD]);
9.         System.out.print("\n Your Hand: ");
10.        for(int nD = 5; nD < 10; nD++)
11.            Card.display(nDeck[nD]);
12.    }
13.    public static void initializeCards(int[] nCards) {
14.        for(int nI = 0; nI < nCards.length; nI++)
15.            nCards[nI] = nI;
16.    }
17.    public static void shuffleCards(int[] nCard) {
18.        for(int nI = 0; nI < nCard.length; nI++)
19.        {
20.            int nIndex = (int)(Math.random() * nCard.length);
21.            int nTemp = nCard[nI];
22.            nCard[nI] = nCard[nIndex];
23.            nCard[nIndex] = nTemp;
24.        }
25.    }
26. }
27. class Card {
28.     static char[] cSuits = {'\u2660' /* Spades */, '\u2665' /* Hearts */,
29.        '\u2666' /* Diamonds */, '\u2663' /* Clubs */};
30.     static char[] cRanks = {'A','2','3','4','5','6','7','8','9','T','J','Q','K'};
31.     public static void display(int nCard) {
32.         char cSuit = cSuits[nCard / 13];
33.         char cRank = cRanks[nCard % 13];
34.         System.out.printf("%c%c ", cRank, cSuit);
35.     }
36. }
    
```

Two classes in one file
 Poker5Stud.java
 Note: Card class is not public
 Only one main method per file

Dealer Hand: 2♠ 5♠ 4♠ A♠ 2♠
 Your Hand: 3♥ A♥ J♥ 7♥ J♥

This is a procedural approach
 because all methods are static
 and no objects are created

Passing Objects and Arrays of Objects

❖ Passing reference to objects to a method

```

1. class TestSquare {
2.     public static void main(String[] args) {
3.         Square oSq1 = new Square(6);
4.         printSquareArea(oSq1);
5.     }
6.     public static void printSquareArea(Square oSq) {
7.         System.out.println("Side="+oSq.getSide()+" Area="+oSq.getArea());
8.     }
9. }
    
```

❖ Passing reference to array of object references

```

1. class TestSquare {
2.     public static void main(String[] args) {
3.         Square[] arySquare = new Square[3];
4.         for(int nI=0; nI < 3; nI++)
5.             arySquare[nI] = new Square(3*(nI + 1));
6.         printSquareArea(arySquare);
7.     }
8.     public static void printSquareArea(Square[] oSq) {
9.         for(int nI=0; nI < 3; nI++)
10.            System.out.println("Side="+oSq[nI].getSide()+" Area="
11.                +oSq[nI].getArea());
12.     }
13. }
    
```

Copyright © 2012 R.M. Laurie 22

```

1. public class Card {
2.     private char cSuit;
3.     private char cRank;
4.     static int nCardsDealt;
5.     Card( char Rank, char Suit) {
6.         cRank = Rank;
7.         cSuit = Suit;
8.     }
9.     public String getCard(){
10.        return Character.toString(cRank)
11.            + Character.toString(cSuit);
12.    }
13.    public static void createCards(Card[] aryCards) {
14.        char[] cSuits = {'\u2660' /* Spades */, '\u2665' /* Hearts */,
15.            '\u2666' /* Diamonds */, '\u2663' /* Clubs */};
16.        char[] cRanks = {'A','2','3','4','5','6','7','8','9','10','J','Q','K'};
17.        for(int nI = 0; nI < aryCards.length; nI++) {
18.            aryCards[nI] = new Card(cRanks[nI % 13], cSuits[nI / 13]);
19.        }
20.    }
21.    public static void shuffleCards(Card[] aryCards) {
22.        for(int nI = 0; nI < aryCards.length; nI++) {
23.            int nIndex = (int)(Math.random() * aryCards.length);
24.            Card oTemp = aryCards[nI];
25.            aryCards[nI] = aryCards[nIndex];
26.            aryCards[nIndex] = oTemp;
27.            nCardsDealt++;
28.        }
29.    }
30.    public static String getAllCards(Card[] aryCards) {
31.        String sOutput="";
32.        for(int nI = 0; nI < aryCards.length; nI++)
33.            sOutput += aryCards[nI].getCard() + " ";
34.        return sOutput;
35.    }
36. }
    
```

```

1. import javax.swing.JOptionPane;
2. public class BlackJack {
3.     public static void main(String[] args) {
4.         Card[] aryCardDeck = new Card[52];
5.         String sDisplay = "Created Deck:\n";
6.         Card.createCards(aryCardDeck);
7.         sDisplay += Card.getAllCards(aryCardDeck);
8.         Card.shuffleCards(aryCardDeck);
9.         sDisplay += "\nShuffled Deck:\n";
10.        sDisplay += Card.getAllCards(aryCardDeck);
11.        JOptionPane.showMessageDialog(null, sDisplay);
12.    }
13. }
    
```

Two classes in two files
 BlackJack.java & Card.java
 Note: Both classes are public
 This is an Object Oriented approach
 because objects are created
 Which program is easier to
 understand and utilize?

Homework 4

- ❖ Using UML design a class to represent a sphere. The class must contain appropriate data fields, and the following methods: Constructor, setDiameter, getDiameter, getVolume, getSurfaceArea, getQtySpheres.
- ❖ Implement your design using Java and name your Java file YourName_Sphere.java but Do not include main method in you Sphere class!
- ❖ For this program you will determine the total volume of glass required to make a glass sphere of a certain diameter and wall thickness. For your algorithm you should consider two concentric spheres of different diameters. Determine the volume of these two spheres. Then subtract the volume of the smaller inner sphere from the volume of the outer sphere. Create a flowchart to describe the processing in this one main method showing boxes for calls to sphere object methods.
- ❖ Your program must prompt the user for the outer sphere and inner sphere diameters. Alternatively you could also prompt for wall thickness and make appropriate calculations to determine inner sphere diameter.
- ❖ Create a second class called VolGlassSphere within in the same file which will have the main method. Code your glass sphere volume calculator such that it calls methods in the YourName_Sphere class
- ❖ Upload via WebTycho your *YourName_Sphere.java* program source code file to the Homework 4 assignment folder. Submit on paper: Cover sheet, design to include specifications, UML class diagram, test data, output for test data, and source code.
- ❖ This program is due at the beginning of Class 2 - Week 7.

Copyright © 2012 R.M. Laurie 24