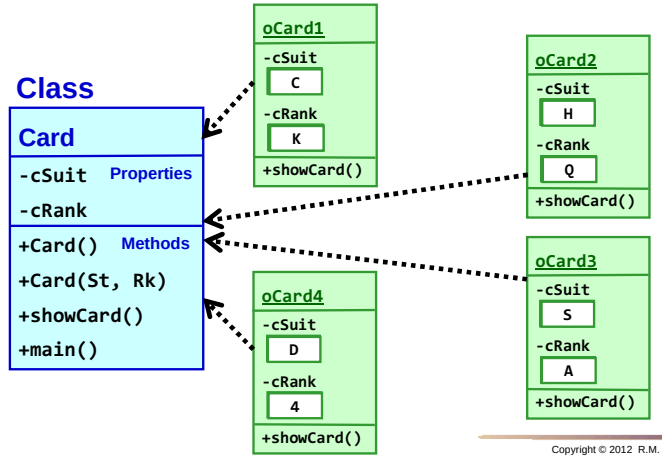
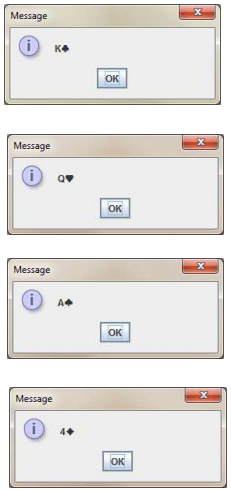
Java Modularity

- **Modularity**
  - Break large problem into smaller pieces
  - *Divide & Conquer* problem solving approach
  - Facilitates design, implementation, and maintenance
- *Packages* contain groups of *reusable* Classes
- *Classes* provide a framework for adding functionality to the Java Language
  - *Defines* Properties (attributes, class variables, fields)
  - *Defines* Methods (functions, procedures, operations)
- *Objects* are self contained instances of a class
  - *Contains* Properties (instance variables, member fields)
  - *Calls* Methods (instance methods, member functions)

UML Diagram of Instantiated Objects




```
1.  import javax.swing.JOptionPane;
2.  public class Card {
3.     // PROPERTIES IN CLASS
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.    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. }
```



Access Specifier

`public`

- Any other class or method can directly access or change a *public instance variable*
- Similar to global variable and should be avoided
- Any other class method invoke a *public method*

`private`

- Only a method in the same class can access or change a *private instance variable*
- only a method in the same class can invoke a *private method*

Class and method variables should be `private` to prevent inappropriate changes.

Static Methods and Non-Static

- **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);
    ```
- Non-static methods access through objects
  - Syntax:
    ```
    dataType objectName.methodName(parameters);
    ```
  - Example:
    ```
    oCard1.showCard();
    nLength = sEntry.length();
    C1st = sEntry.charAt(0);
    ```


Declaration Statements Syntax

- optAccessSpecifier dataType varName;
  - private – Access variables only within class methods
  - public – Access variables from anywhere (Avoid!)
- *Variable Scope* – Specifies visibility of variable
  - *Local* – Only accessible when code block is executed
    ```
    int nSum;
    ```
  - *Instance* – Created for each object (Object Data Field)
    ```
    private int nSum;  // Access only within class methods
    ```
  - *Class* – Within class's body but outside method
    ```
    private static int nSum;
    ```
  - *Parameter* – Within parenthesis of method head
    ```
    public void setCard(char cRank, char cSuit)
    ```
    Parameters are only available in method

Creating Objects

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

Java Method Definition

- Method Definitions are encapsulated in a class
  - Identifier naming convention verbNoun
  - Parameters
    - Input data for the method assigned to parameter variables
    - Requires data type to be specified in method definition
  - Local variables
    - Declared within method declaration
  - Return value data type specified
    - Result value is passed to *method caller*
- Method definition syntax

Method Header

```
return-value-type  methodName( parameter1, parameter2, …)
{                          // Method body start
    declarations and statements

    return nValue;
}                          // Method body end
```


Java Method Call

- *Method Call* invoked in other methods
  - 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:

```
JOptionPane.showMessageDialog(null, sOut);
oCard4.showCard();
```


Java Static Method Call Example

```
public class MethodDemo1
{
  public static void main(String args[])
  {
    int nI;
    for ( nI = 1; nI <= 9; nI++)
      System.out.println("The square of "
        + nI + " is " + squareNum(nI) );
  }
  public static int squareNum(int nY)
  {
     int nX;
     nX = nY * nY;
     return nX;
  }
}
```

Calling method `squareNum` and passing as an argument the value of `nI`.

Parameter `nY` gets the value of argument variable `nI`.

The `return` statement passes the value of `nX` back to the calling function.

```
The square of 1 is 1
The square of 2 is 4
The square of 3 is 9
The square of 4 is 16
The square of 5 is 25
The square of 6 is 36
The square of 7 is 49
The square of 8 is 64
The square of 9 is 81
```


Integer and Double Classes

- Integer class is *wrapper* for int primitive data type
  - .parseInt(String)
  - .toString(int)
- Double class is *wrapper* for double primitive data type
  - .parseDouble(String)
  - .toString(double)
- Static methods invoked from class (not object)
  - int nQuantity = Integer.parseInt(sQuantity);
  - double dPrice = Double.parseDouble(sPrice);
  - String sCounter = Integer.toString(nCounter);
  - String sPrice = Double.toString(dPrice);

Integer Class

Properties

Value

Methods

.toString(int)

.parseInt(String)

Class Method Inputs

Mathematical Class

- Java provides standard preprogrammed methods within class named `Math`
  - Methods are `static` and `public`
  - Considered part of java.lang package
  - Part of Java Language so no import needed
- Each `Math class` method is called by:
  - Listing name of class `Math`
  - A period `.`
  - Method's name `dAnswer = Math.pow(3, 2)`
  - Pass data arguments within parentheses
  - Return value type needs to be considered

http://docs.oracle.com/javase/6/docs/api/java/lang/Math.html

Java's Math Class

- Class constants:
  `PI, E`
- Class methods:
  - Rounding Methods:

`double ceil(double x)`	x rounded up to its integer
`double floor(double x)`	x is rounded down to integer `int`
`round(float x)`	x is rounded to its nearest integer

  - min, max, abs, and random Methods

`max(a, b)` and `min(a, b)`	Return max or min of two values
`abs(a)`	Returns the absolute value
`random()`	Returns random `double` [0, 1]

  - Exponent Methods

`pow(double a, double b)`	Returns a to the power of b
`sqrt(double a)`	Returns the square root of a

  - Trigonometric Methods

`sin(double a)`	`asin(double a)`
`cos(double a)`	`acos(double a)`
`tan(double a)`	`atan(double a)`

 13

Java's Math Class Examples

`Math.max(2, 3)`	returns 3
`Math.max(2.5, 3)`	returns 3.0
`Math.min(2.5, 3.6)`	returns 2.5
`Math.abs(-2)`	returns 2
`Math.abs(-2.1)`	returns 2.1
`Math.ceil(2.1)`	returns 3.0
`Math.ceil(-2.1)`	returns -2.0
`Math.floor(2.1)`	returns 2.0
`Math.floor(-2.1)`	returns -3.0
`Math.round(2.6f)`	returns 3
`Math.round(-2.0f)`	returns -2
`Math.round(-2.6)`	returns -3
`Math.pow(2, 3)`	returns 8.0
`Math.pow(3, 2)`	returns 9.0
`Math.pow(3.5, 2.5)`	returns 22.91765
`Math.sqrt(4)`	returns 2.0
`Math.sqrt(10.5)`	returns 3.24
`(int)(Math.random() * 10)`	Random integer [0 to 9]
`50 + (int)(Math.random() * 50)`	Random integer [50 to 99]

 14

Using a Class Method Library

- Java provides extensive set of tested and reliable classes
  - Increases with introduction of each new version
  - Java Platform, Standard Edition 6 API Specification
  - http://docs.oracle.com/javase/6/docs/api/index.html
- Professional programmers create and share libraries of developed classes
  - Enables code reuse in other programs
  - Minimizes redundant code
  - Code reliability dependent on testing rigor
  - Encapsulation = implementation details hidden
  - Top – Down Design process is general to detail
  - Top – Down or Bottom – Up implementation

 15

Base Converter

```
1.  import java.util.Scanner;
2.  public class BaseConverter {
3.    public static void main(String[] args) {
4.      Scanner kbdInput = new Scanner(System.in);
5.      while(true) {
6.        System.out.print("Enter a decimal number: ");
7.        int nDec = kbdInput.nextInt();
8.        System.out.print("Would you like to convert this to:"
9.          + "\n  [b] = Binary = Base 2\n  [o] = Octal = Base 8"
10.         + "\n  [h] = Hexadecimal = Base 16\n  [q] = Quit\nWhich Base: ");
11.       String sInput = kbdInput.next();
12.       char cBase = sInput.charAt(0);
13.       if(cBase == 'b' || cBase == 'B')
14.         System.out.println(nDec + " decimal = " + toBin(nDec) + " binary");
15.       else if(cBase == 'o' || cBase == 'O')
16.         System.out.println(nDec + " decimal = 0" + toOct(nDec) + " octal");
17.       else if (cBase == 'h' || cBase == 'H')
18.         System.out.println(nDec + " decimal = 0x"
19.           + toHex(nDec).toUpperCase() + " hexadecimal");
20.       else if (cBase == 'q' || cBase == 'Q')
21.         break;
22.       else
23.         System.out.println("\nWrong letter selected");
24.     }
25.     kbdInput.close();
26.   }
27.   public static String toBin(int nDecimal) {
28.     return Integer.toString(nDecimal, 2);
29.   }
30.   public static String toOct(int nDecimal) {
31.     return Integer.toString(nDecimal, 8);
32.   }
33.   public static String toHex(int nDecimal) {
34.     return Integer.toString(nDecimal, 16);
35. } }
```

```
Enter a decimal number: 123
Would you like to convert this to:
  [b] = Binary = Base 2
  [o] = Octal = Base 8
  [h] = Hexadecimal = Base 16
  [q] = Quit
Which Base: h
123 decimal = 0x7B hexadecimal
```