Welcome

Java Programming I
CIS 325

Week 4 – Java Classes II and Review

Christopher K. Burns

Agenda

Tonight's agenda

- Homework Reminder
- Finishing up Java Arrays
- Building a Java App from start to finish
  - Java Classes
  - Review of Chapters 1-8
- Sample Mid-Term

Home Work

(Assigned last week)

Read text Chapters 6 and 7

5.9 on page 226

6.23 on page 282

- Due before start of class on 4 May

Schedule

Week		Content	
1	4/6	Chapter 1 Intro to Computers, the Internet and the Web Chapter 2 Intro to Java Applications Chapter 3 Java Classes and Objects: Part 1 *Homework 1 Assigned*	
2	4/13	Chapter 4 Control Structures: Part 1 Chapter 5 Control Structures: Part 2	
3	4/20	Chapter 6 Methods Chapter 7 Arrays *Homework 2 Assigned*	HW 1 DUE
4	4/27	Chapter 8 Java Classes and Objects: Part 2 Chapter 1-8 Review	
5	5/4	***MID-TERM EXAMINATION***	HW 2 DUE
6	5/11	Chapter 9 Object-Oriented Programming: Inheritance Chapter 10 Object-Oriented Programming: Polymorphism *Homework 3 Assigned*	PROJECT IDEA DUE
7	5/18	Chapter 11 GUI Components: Part 1 Chapter 12 Graphics and Java2D	
8	5/25	Chapter 13 Exception Handling Chapter 29 Strings, Characters and RegEx *Homework 4 Assigned*	HW 3 DUE
9	6/1	Chapter 20: Java Applets Chapter 23 Multithreading	
10	6/8	Files, JDBC, Networking, Servlets, and JSP *Class lab time for review and assistance with final project*	HW 4 DUE
11	6/15	*Alternate class night for any previously cancelled classes* ***FINAL PROJECT DUE***	PROJECT DUE

Last Weeks Topics

- Java API documentation, Math and Random
- Methods
- Static Methods
- Constants (Final)
- Method overloading
- Returning values
- Casting and Promotion
- Declaring and using arrays
- Passing arrays to methods
- Arrays are passed by reference
- JDK5 new for loop and multi params
- Multi-dimensional arrays
- Command line arguments

Declaring and Creating Arrays

- Declaring and Creating arrays
  - Arrays are objects that occupy memory
  - Created dynamically with keyword new

```
int c[] = new int[ 12 ];
```

    - Equivalent to

```
int c[];  // declare array variable
c = new int[ 12 ]; // create array
```

  - We can create arrays of objects too

```
String b[] = new String[ 100 ];
```

Examples Using Arrays

```
import javax.swing.JOptionPane;

public class UseMyMathArray1
{
    public static void main(String[] args)
    {
        int nums [] = new int[ 5 ];
        nums[0] = 10;
        nums[1] = 20;
        nums[2] = 35;
        nums[3] = 5;
        nums[4] = -2;

        System.out.println( "Max is " + MyMath.max( nums[0], nums[1],
           nums[2], nums[3], nums[4] ) );
    }
}
```

Declaring and Creating Arrays

- Using an array initializer
  - Use *initializer list*
    - Items enclosed in braces ({})
    - Items in list separated by commas

```
int n[] = { 10, 20, 30, 40, 50 };
```

      - Creates a five-element array
      - Index values of 0, 1, 2, 3, 4
  - Do not need keyword new

Examples Using Arrays

```
import javax.swing.JOptionPane;

public class UseMyMathArray1
{
    public static void main(String[] args)
    {
        int nums [] = { 10, 20, 35, 5, -2 };

        System.out.println( "Max is " + MyMath.max( nums[0], nums[1],
           nums[2], nums[3], nums[4] ) );
    }
}
```

Passing Arrays to Methods

```
public class MyMathArray
{
    public static int max( int nums[] )
    {
        int max = nums[0];
        for ( int i = 1; i < nums.length; i++ )
        {
            max = Math.max( max, nums[i] );
        }
        return max;
    }
}
```

Passing Arrays to Methods

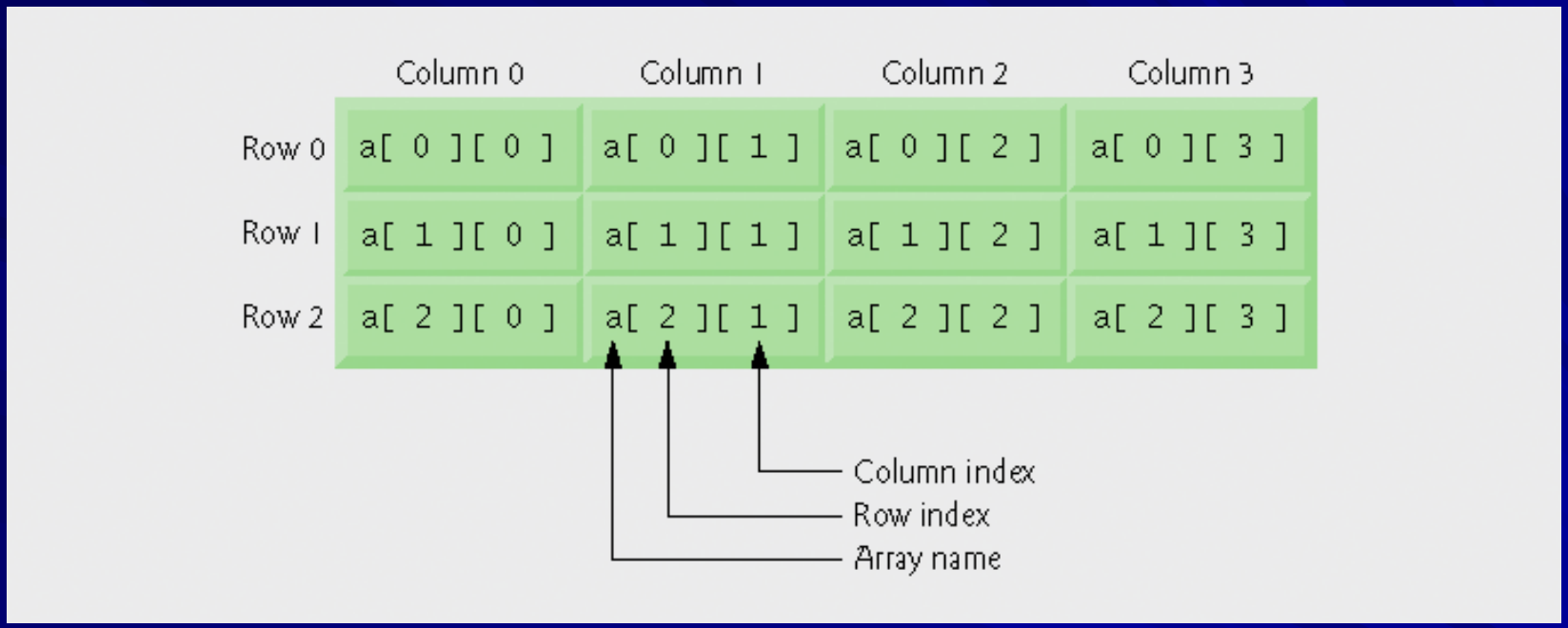
```
import javax.swing.JOptionPane;

public class UseMyMathArray2
{
    public static void main(String[] args)
    {
        int nums [] = { 10, 20, 35, 5, -2 };
        System.out.println( "Max is " +
            MyMathArray.max( nums ) );
    }
}
```

Multidimensional Arrays

- Multidimensional arrays
  - Tables with rows and columns
    - Two-dimensional array
    - m-by-n array

Two-dimensional array with three rows and four columns.



Multidimensional Arrays

- Arrays of one-dimensional array
  - Declaring two-dimensional array b[2][2]

```
int b[][] = { { 1, 2 }, { 3, 4 } };
```

    - 1 and 2 initialize b[0][0] and b[0][1]
    - 3 and 4 initialize b[1][0] and b[1][1]

```
int b[][] = { { 1, 2 }, { 3, 4, 5 }
};
```

    - row 0 contains elements 1 and 2
    - row 1 contains elements 3, 4 and 5

Multidimensional Arrays

- Two-dimensional arrays with rows of different lengths
  - Lengths of rows in array are not required to be the same
    - E.g., `int b[][] = { { 1, 2 }, { 3, 4, 5 } };`

Multidimensional Arrays

- Creating two-dimensional arrays with array-creation expressions
  - Can be created dynamically
    - 3-by-4 array

```
int b[][];
b = new int[ 3 ][ 4 ];
```

    - Rows can have different number of columns

```
int b[][];
b = new int[ 2 ][ ];   // create 2 rows
b[ 0 ] = new int[ 5 ]; // create 5 columns for row 0
b[ 1 ] = new int[ 3 ]; // create 3 columns for row 1
```

Multidimensional Arrays

- Common multidimensional-array manipulations performed with `for` statements
  - Many common array manipulations use `for` statements

E.g.,

```
for ( int column = 0; column < a[ 2 ].length;
column++ )
      a[ 2 ][ column ] = 0;
```

Multidimensional Arrays

```
// Fig. 7.17: InitArray.java
// Initializing two-dimensional arrays.

public class InitArray
{
   // create and output two-dimensional arrays
   public static void main( String args[] )
   {
      int array1[][] = { { 1, 2, 3 }, { 4, 5, 6 } };
      int array2[][] = { { 1, 2 }, { 3 }, { 4, 5, 6 } };

      System.out.println( "Values in array1 by row are" );
      outputArray( array1 ); // displays array1 by row

      System.out.println( "\nValues in array2 by row are" );
      outputArray( array2 ); // displays array2 by row
   } // end main
```

Multidimensional Arrays

```
   // output rows and columns of a two-dimensional array
   public static void outputArray( int array[][] )
   {
      // loop through array's rows
      for ( int row = 0; row < array.length; row++ )
      {
         // loop through columns of current row
         for ( int column = 0; column < array[ row ].length; column++ )
            System.out.printf( "%d  ", array[ row ][ column ] );

         System.out.println(); // start new line of output
      } // end outer for
   } // end method outputArray
} // end class InitArray
```

Using Command-Line Arguments

- Command-line arguments
  - Pass arguments from the command line
    - `String args[]`
  - Appear after the class name in the `java` command
    - `java MyClass a b`
  - Number of arguments passed in from command line
    - `args.length`
  - First command-line argument
    - `args[ 0 ]`

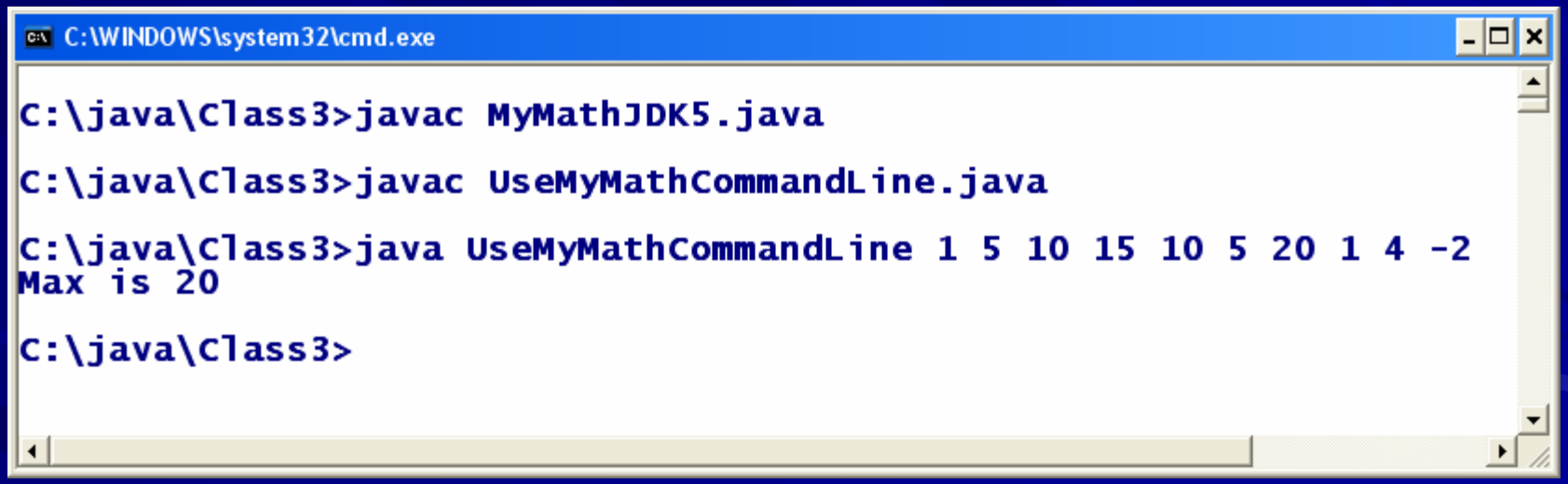
Using Command-Line Arguments

```
public class MyMathJDK5
{
    //public static int max( int nums[] )
    public static int max( int ... nums )
    {
        int max = nums[0];
        for ( int i : nums )
        {
            max = Math.max( max, i );
        }
        return max;
    }

    public static int max( String strings[] )
    {
        int nums[] = new int[ strings.length ];
        for ( int i = 0; i < strings.length; i++ )
        {
            nums[ i ] = Integer.parseInt( strings[ i ] );
        }
        return max( nums );
    }
}
```

Using Command-Line Arguments

```
public class UseMyMathCommandLine
{
    public static void main(String[] args)
    {
        System.out.println( "Max is " + MyMathJDK5.max( args ) );
    }
}
```



Java Classes II

Topics Concerning Classes

- A class versus an object
  - The life of an object
    - Default and Overloaded Constructors
    - Finalize
  - The life of a class
    - static methods and fields
- Accessors – privates, and public set and get
- What's this?
- Enumerated types

Scenario

We are responding to an RFP from a law firm to create a software package that will track their phone usage so clients can be billed appropriately.

We need to prove that this can be done in Java.

Let's create a simple prototype to start with.

We will create two classes. One to control when the calls start and end, and the other to contain the call data.

Scenario

- Need to model
  - Phone Call
    - Properties
      - Name
      - Number
  - Phone Manager
    - Methods
      - StartCall
      - PrintBill

The `PhoneCall` class

a lot of code will be put together over the next few slides. (There is a download of the code at these various stages)

Here is a break down of the steps:

1. **`Basic Class Structure and Usage`**
2. **`Using accessors and this`**
3. **`Turning the PhoneTest class into a call manager`**
4. **`Duration (Time) calculation`**
5. **`Constructors`**
6. **`Using a class static`**
7. **`Using a class static array to create a collection`**
8. **`Enumerated types and overloaded constructors`**

1. The `PhoneCall` class

```
public class PhoneCall
{
    public String name;
    public String number;
}
```

1. Testing our PhoneCall class

```
import javax.swing.JOptionPane;

public class PhoneTest1
{
    public static void main( String args[] )
    {
        PhoneCall1 pc;
        pc = new PhoneCall1();
        pc.name = "Stan";
        pc.number = "540-555-1234";

        String output = "Call made to:\n" + pc.name + "\n" + pc.number;
        JOptionPane.showMessageDialog( null, output );
    }
}
```

2.The `PhoneCall` class – with accessors and *this*

```
public class PhoneCall
{
    private String number;
    public String getName()
    {
        return this.name;
    }
    public void setName( String name )
    {
        this.name = name;
    }

    private String name;
    public String getNumber()
    {
        return this.number;
    }
    public void setNumber( String number )
    {
        this.number = number;
    }
}
```

3. Managing a call (PhoneTest.java)

```
public static void main( String args[] )
{
    phoneTest = new PhoneTest3();
    phoneTest.startCall();
    phoneTest.printLogs();
}

public void startCall()
{
    String number = JOptionPane.showInputDialog( "Please enter number to dial" );
    String name = JOptionPane.showInputDialog( "Please enter account");
    pc = new PhoneCall3();
    pc.setName( name );
    pc.setNumber( number );
}

public void printLogs()
{
    String output = "Call made to:\n" + pc.getName() + "\n" + pc.getNumber();
    JOptionPane.showMessageDialog( null, output );
}
```

4. How long was the call

```
public void startCall()
   {
       String number = JOptionPane.showInputDialog( "Please enter number to dial" );
       String name = JOptionPane.showInputDialog( "Please enter account");

       JOptionPane.showMessageDialog( null,
                "Click Okay to Begin\nCalling " + name + "\nat " + number );
       long startTime = System.currentTimeMillis();

       JOptionPane.showMessageDialog( null, "Click okay when done calling");
       long endTime = System.currentTimeMillis();

       long duration = endTime - startTime;
       duration /= 1000;

       pc = new PhoneCall4();
       pc.setName( name );
       pc.setNumber( number );
       pc.setDuration( duration );
   }
```

5. Adding a constructor

```
public class PhoneCall
{
    private String name;
    private String number;
    private long duration;

    public PhoneCall( String name, String number, long duration )
    {
        this.name = name;
        this.number = number;
        this.duration = duration;
    }

    public String getName()
    {
...
```

6. Static

Using a static member variables, a class can store information about itself, such as how many instances have been created…

```
public class PhoneCall
{
    public static long callCount;
    private String name;
    private String number;
    private long duration;

    public PhoneCall( String name, String number, long duration )
    {
        this.name = name;
        this.number = number;
        this.duration = duration;
        callCount++;
    }
...
```

6. Static

… or how many instances have been destroyed…

```
public class PhoneCall
{
    public static long callCount;
    private String name;
    private String number;
    private long duration;

    protected void finalize()
    {
       callCount--;
    }
 ...
```

7. Using Statics to Contain Ourselves

As we saw in the previous example, a static member variable belongs to the class and not to each object instance…

Thus static members are essentially "global variables" within the context of a class.

So, if we wanted to maintain a collection of some class object, who would know better how to collect itself than the class itself. We will now use an array declared as static to contain all the instances of the PhoneCall class.

7. Using Statics to Contain Ourselves

```
public class PhoneCall
{
    private static PhoneCall calls[];
    ...

    public PhoneCall( String name, String number, long duration )
    {
        ...
        if ( calls == null )
        {
            // no call array has been created yet, so create a new one of size 1
            calls = new PhoneCall7[1];
            // set the first element of the array to us
            calls[ 0 ] = this;
        }
        else
        {
            // make an array that is 1 larger than current call array
            PhoneCall7 callsTemp[] = new PhoneCall7[ calls.length + 1 ];
            for( int i = 0; i < calls.length; i++ )
            {
                callsTemp[i] = calls[i];
            }
            callsTemp[ callsTemp.length - 1 ] = this;
            calls = callsTemp;
        }
    }
```

7. Using Statics to Contain Ourselves

```
    public static String getLogs()
    {
        String output = "";
        for( int i = 0; i < calls.length; i++ )
        {
            output += calls[ i ].getName() + " : " + calls[i].getNumber() + " (" + calls[i].getDuration() + "s)\n";
        }
        return output;
    }
}
```

8. What type of call?

This is a new feature of the JDK 1.5, so will not work on the class compilers; but it is a very important and useful language feature.

Enumerated types allow us to define nicely named constants that represent particular pieces of data:

Colors :

RED = 0;

GREEN = 1;

BLUE = 2;

8. What type of call?

```
{
  public static enum CallType { BUSINESS, PERSONAL,
   EMERGENCY, MAINTENANCE, DONE };

  private CallType callType;

  public CallType getType()
  {
        return callType;
  }

...
}
```

8. What type of call?

```
private PhoneCall.CallType getCallType()
    {
        String prompt = "Enter Call Type:";
        for( int i = 0; i < PhoneCall.CallType.values().length; i++ )
        {
            prompt += "\n " + (i + 1) + ": " +
                PhoneCall.CallType.values()[i].toString();
        }
        String userInput = JOptionPane.showInputDialog( prompt );
        int index = Integer.parseInt( userInput ) - 1;
        return PhoneCall.CallType.values()[ index ];
    }
```

8. Now let's add overload the constructor

```
public class PhoneCall
{
...
    public PhoneCall( String name, String number, long duration )
    {
          ...
    }

    public PhoneCall( String name, String number, long duration, CallType callType )
    {
        this( name, number, duration );
        this.callType = callType;
    }

...
```

Sample Mid Term

(a sample mid-term will be handed out in class)