CS1007 lecture #15 notes

thu 14 mar 2002

- news
- two-dimensional arrays
- formatting output
- keyboard input
- exception handling
- streams
- two-dimensional arrays of objects
- vectors
- reading: ch 2.8, 4.6, 6.5, 8.1-8.3

news.

- midterm #2 changed to: TUE APRIL 9
- homework #4 was posted yesterday, due: THU MAR 28
  - look at it early – it's not so easy
- homework #3 — I'll run the fun tournament over break

two-dimensional arrays (review, 1).

- last time we talked about two-dimensional arrays of characters
- as with one-dimensional arrays, you can have arrays of anything
- you can also have more than 2 dimensions, but we won't go into that...
- you declare a 2D array like this:
  `double[][] a2;`
- instantiate like this (for example for a 4x4 array):
  `a2 = new double[4][4];`
- the first dimension is called *row*
- the second dimension is called *column*
- so the element in the i-th row and the j-th column is accessed like this:
  `a2[`i`][`j`]`

two-dimensional arrays (review, 2).

- indices look like this:

$column \rightarrow$ row $\downarrow$	0	1	2	3
0	$a2[0][0]$	$a2[0][1]$	$a2[0][2]$	$a2[0][3]$
1	$a2[1][0]$	$a2[1][1]$	$a2[1][2]$	$a2[1][3]$
2	$a2[2][0]$	$a2[2][1]$	$a2[2][2]$	$a2[2][3]$
3	$a2[3][0]$	$a2[3][1]$	$a2[3][2]$	$a2[3][3]$

- so typically you use two nested loops to cycle through the $rows$ (outer loop) and $columns$ (inner loop)
- for example:

```
for ( int i=0; i<size; i++ ) {
  for ( int j=0; j<size; j++ ) {
    ...do something with a2[i][j]...
  } // end for j (inner, column loop)
} // end for i (outer, row loop)
```

formatting output.

- `java.text.DecimalFormat` class
- used to format decimal numbers
- construct an object that handles a format
- use that format to output decimal numbers
- methods include:
  - `DecimalFormat( String pattern );`
  - `void applyPattern( String pattern );`
  - `String format( double number );`
- formatting patterns include:
  - 0 used to indicate that a digit should be printed, or 0 if there is no digit in the number (i.e., leading and trailing zeros)
  - # used to indicate that if there is a digit in the number, then it should be printed; indicates rounding if used to the right of the decimal point
  - e.g., `DecimalFormat fmt = new DecimalFormat( "#.00" );`

example using 2D arrays and formatted output.

```
import java.util.Random;
import java.text.DecimalFormat;

public class ex15_1 {

  public static void main( String[] args ) {

    // declare and instantiate 2D array (4x4)
    final int size = 4;
    double[][] a2 = new double[size][size];

    // initialize array elements to random numbers
    Random rndm = new Random();
    for ( int i=0; i<size; i++ ) {
      for ( int j=0; j<size; j++ ) {
        a2[i][j] = rndm.nextDouble();
      } // end for j
```

```
    } // end for i

    // output array using formatted output so that
    // it all lines up nicely
    DecimalFormat fmt = new DecimalFormat( "#.00" );
    for ( int i=0; i<size; i++ ) {
      for ( int j=0; j<size; j++ ) {
        System.out.print( fmt.format( a2[i][j] )+" " );
      } // end for j
      System.out.println();
    } // end for i

  } // end of main()

} // end of ex15_1
```

keyboard input (1).

- `java.io.InputStream` class
- methods:

```
public abstract int read() throws IOException;
```

- reads a single character from the keyboard
- returns its ASCII value
- in UNIX, $< enter >$ key has an ASCII value of 10
- in DOS, $< enter >$ key has two values: carriage return and linefeed, so special handling is requred
- *use UNIX for your homework*, otherwise the TAs will have problems running it

keyboard input (2).

- here's a `Keyboard` class that reads one character from the keyboard and returns it:

```
import java.io.*;
public class Keyboard {
  public static char readchar() {
    int i = 0;
    try {
      i = System.in.read();
    }
    catch ( IOException iox ) {
      System.out.println( "there was an error: " + iox );
    }
    return (char)i;
  } // end of readchar()
} // end class Keyboard
```

keyboard input (3).

- typically program *prompts* the user for input
- e.g., `please type something:`
- then `Keyboard.readchar()` is called
- $< enter >$ is handled differently —
  - it goes into the input buffer
  - but it also is used to terminate the input
- given the statement:
  `char c = Keyboard.readchar()`
  how many characters go into the input buffer?

keyboard input (4).

- will this work?

```
public class ex15_2 {
  public static void main( String[] args ) {
    char c;
    boolean more = true;
    while ( more ) {
      System.out.print( "please type something: " );
      c = Keyboard.readchar();
      System.out.println( "c=" + c );
      more = ( c != 10 );
    }
  } // end of main
} // end class ex15_2
```

- how can we fix it?

exception handling, in brief (review).

- example:

```
try {
  i = System.in.read();
}
catch ( IOException iox ) {
  System.out.println( "there was an error: " + iox );
}
```

- `try` clause contains code which may generate an exception, i.e., an error
- `catch` clause contains code to execute in case the error happens; i.e., where to go if the exception gets *caught*

streams (1).

- here's our standard picture of input and output:
  input $\rightarrow$ $\boxed{\text{CPU}}$ $\rightarrow$ output
- for now, input comes from the keyboard and output goes to the screen
- later in the semester, we'll learn about file I/O (input/output)
- input and output flow from and to *streams*
- a *stream* is an ordered sequence of bytes
- they flow from a source to a destination
- with input, the source is the keyboard and the destination is a program
- with output, the source is a program and the destination is the screen

streams (2).

- thus there are two categories of streams:
  - *input streams*
  - *output streams*
- streams can also be subdivided based on their content:
  - *character streams* (i.e., text)
  - *byte streams* (i.e., binary data)
- or their usage:
  - *data streams* (e.g., String in memory, file on disk)
  - *processing streams* (manipulation of a data stream)

streams (3).

- classes that handle *byte streams*
  - `InputStream` ← `FileInputStream`
  - `OutputStream` ← `FileOutputStream` ← `PrintStream`
- classes that handle *character streams*
  - `Reader` ← `BufferedReader`
  - `Writer` ← `BufferedWriter`
- for example, in `java.lang.System`:
  - `System.in` is an `InputStream`
  - `System.out` is a `PrintStream`

streams (4).

- we can re-write our `Keyboard` class using a `BufferedReader`:

```
import java.io.*;
public class Keyboard2 {
  public static char readchar() {
    InputStreamReader isr = new InputStreamReader( System.in );
    BufferedReader  stdin = new BufferedReader( isr );
    String s = null;
    try {
      s = stdin.readLine();
    }
    catch( IOException iox ) {
      System.out.println( "there was an error: " + iox );
    }
    return s.charAt( 0 );
  } // end of readchar()
} // end of class Keyboard2
```

two-dimensional arrays of objects (1).

- just like with one-dimensional arrays, with 2D arrays you can also have arrays of objects
- here's an example using a class called `Dice`:

```
import java.util.*;

public class Dice {

  private Random random;
  private int    value;

  public Dice( Random r ) {
    random = r;
    roll();
  } // end of Dice() constructor

  public void roll() {
    value = Math.abs( random.nextInt() % 6 ) + 1;
```

```
  } // end of roll() method

  public int getValue() {
    return( value );
  } // end of getValue() method

  public String toString() {
    return( String.valueOf( value ));
  } // end of toSring() method

} // end of Dice class
```

two-dimensional arrays of objects (2).

- here's the main class:

```
import java.util.*;
import java.io.*;

public class ex15_3 {

  public static void main( String[] args ) {

    int      ndice = 0;
    Dice[][] dice;
    Random   random = new Random();

    System.out.print( "how many dice should we have? " );
    InputStreamReader isr = new InputStreamReader( System.in );
    BufferedReader  stdin = new BufferedReader( isr );
    String s = null;
```

```
try {
  s = stdin.readLine();
  ndice = Integer.parseInt( s.trim() );
}
catch( IOException iox ) {
  System.out.println( "there was an error: " + iox );
  System.exit( 1 );
}

dice = new Dice[ndice][ndice];
for ( int i=0; i<ndice; i++ ) {
  for ( int j=0; j<ndice; j++ ) {
    dice[i][j] = new Dice( random );
  }
}

for ( int i=0; i<ndice; i++ ) {
  for ( int j=0; j<ndice; j++ ) {
    System.out.print( dice[i][j] + " " );
  }
```

```
        System.out.println();
      }

    } // end of main()

  } // end of class ex15_3
```

- notice that we are using the `BufferedReader`...
- notice that we use `String.trim()`...
- notice that we use `System.exit( 1 )`...
- notice that we instantiate twice...

vectors (1).

- Java has a nice class which handles arrays dynamically: `java.util.Vector`
- the elements of a `Vector` can be any type of Java `Object`
- note that when you fetch an element from a vector, you have to cast it from a generic object to the specific class type the object should be (see example below)
- some methods:
  - constructor: `Vector();`
  - `public void addElement( Object obj );`
  - `public void insertElementAt( Object obj, int index );`
  - `public void removeElementAt( int index );`
  - `public void removeAllElements();`
  - `public void setElementAt( Object obj, int index );`
  - `public Object elementAt( int index );`
  - `public int size();`

vectors (2).

- let's use the `Dice` class again
- here's a new main class:

```
import java.util.*;
import java.io.*;

public class ex15_4 {

  public static void main( String[] args ) {

    int    ndice = 0;
    Vector dice;
    Random random = new Random();

    System.out.print( "how many dice should we have? " );
    InputStreamReader isr = new InputStreamReader( System.in );
    BufferedReader  stdin = new BufferedReader( isr );
```

```
String s = null;
try {
  s = stdin.readLine();
  ndice = Integer.parseInt( s.trim() );
}
catch( IOException iox ) {
  System.out.println( "there was an error: " + iox );
  System.exit( 1 );
}

dice = new Vector( ndice );
for ( int i=0; i<ndice; i++ ) {
  dice.addElement( new Dice( random ));
}

for ( int i=0; i<ndice; i++ ) {
  Dice tmp = (Dice)dice.elementAt( i );
  System.out.print( tmp + " " );
}
System.out.println();
```

```
    } // end of main()

  } // end of class ex15_4
```

- notice that we instantiate twice...
- notice that we instantiate in the call to `dice.addElement()`
- notice that we *cast* the return from `dice.elementAt()`

$\rightarrow$

Have a great spring break!!