Unit Five Lecture Three:

Topic 1: Copying Arrays

Array variables hold a reference (pointer) to the actual array. If you copy the reference, you get another reference to the same array.

```
double[] data = new double[10];
.
.
.
double[] prices = data;
```

data and prices are both references to the same array, one array - two names.

If you want to make a true copy of an array, call the clone method.

```
double[] prices = (double[])data.clone();
```

data is a reference to the original array and prices is a reference to a different array with the same contents as the array referenced by data.

From time to time you may need to copy elements from one array into another array. You can use the static method System.arraycopy for that purpose.

```
System.arraycopy(data, 0, prices, 0, data.length);

data   -> array you're copying from
0      -> index in data at which copying should start
prices -> array you're copying to
0      -> index in prices where 1st element should be copied
data.length -> number of elements to be copied
```

Topic 2: Resizing an Array

In some situations you will not know how many items your array must hold. To solve this problem, you make an array that you anticipate is larger than the largest possible number of entries, and partially fill it.

However, in some situations the number of entries may still exceed the size of the array. When you run out of space in an array, you can create a new, larger array. To do this, you copy all elements into the new array and have the old array variable reference the new array. APCS recommends that when you create the new array you at least **double** the size of the original array.

```
public class Example
{
      private int myArray[] = new int[50];
      .
      .
      .
      public void Resize()
      {
            int newArray[] = new int[2 * myArray.length];

            System.arraycopy(myArray, 0, newArray, 0, myArray.length);
            myArray = newArray;
      }
}
```

Remember if you do nothing to protect your program from a subscript out of bounds error, this kind of error will crash the program.

Topic 3: Inserting elements into an Array to maintain order

The following code shows you how to do this. The assumptions here are that the array (myArray) is already partially filled in ascending order, the integer to be inserted is stored in num, and the number of elements in the array is stored in dataSize.

```
int j = 0;
boolean found = false;

while ((j<dataSize) && (!found))
{
      if (num > myArray[j])
            j++;
      else
            found = true;
}
for (int k=dataSize; k>j; k--)
{
      myArray[k] = myArray[k-1];
}
myArray[j] = num;
dataSize++;
```

Topic 4: Deleting an element from an Array

The following code shows you how to do this. The assumptions here are that the array (myArray) is already partially filled, the integer to be deleted is stored in num, and the number of elements in the array is stored in dataSize.

```
int j = 0;
boolean found = false;

while ((j<dataSize) && (!found))
{
      if (num != myArray[j])
            j++;
      else
            found = true;
}
dataSize--;
for (int k=j; k<dataSize; k++)
{
      myArray[k] = myArray[k+1];
}
```

Topic 5: Exception Handling

In Java, exception handling provides a flexible mechanism for passing control from the point of error detection to a competent recovery handler.

The try/catch Structure

Suppose you would like to trap a possible subscript out of bounds, Runtime error. You could use the following code.

```
import javax.swing.JOptionPane;
import javax.swing.JApplet;

public class Subscript extends JApplet
{
      public void init()
      {
            int[] data = {8, 12, 20, 29, 32};

            int index = 0, value;
            String input;

            try   // throwing an exception
            {
                  input = JOptionPane.showInputDialog("Enter Index: ");
                  index = Integer.parseInt(input);

                  value = data[index];  // possible subscript out of bounds

                  JOptionPane.showMessageDialog(null, "Value = " + value,
                                    "Example", JOptionPane.INFORMATION_MESSAGE);
            }
```

```
            catch (RuntimeException exception)   // catching an exception
            {
                  JOptionPane.showMessageDialog(null, "Adjusting Bad Index",
                                    "Example", JOptionPane.ERROR_MESSAGE);

                  index = index % 5;
                  value = data[index];

                  JOptionPane.showMessageDialog(null, "Value = " + value,
                                    "Example", JOptionPane.INFORMATION_MESSAGE);
            }
      }
}
```

The programmer encloses in a *try* block any code that may generate an exception (error). The *try* block should be followed by one or more *catch* blocks. Each *catch* block specifies the type of exception it can catch and contains code to handle that exception.

When a *try* block throws an exception, program control leaves that *try* block. The program then searches the catch blocks, in order, looking for an appropriate handler. Note that an exception handler cannot access variables or objects defined in the corresponding *try* block, because the try block expires before the handler begins executing and those variables and objects are out of scope. When a *catch* block has finished executing, control resumes with the first statement after the last exception handler.

If a *try* block executes and no exceptions are thrown, all the exception handlers are skipped and control resumes with the first statement after the last exception handler.

Exceptions included in the AP Java subset

1) ArithmeticException - dividing an integer by zero

```
int n=5;
int m=0;
int ans = n/m;
```

2) NullPointerException - failing to initialize a reference instance field and trying to use this variable in a method of the class.

```
private Month x;
.
.
.
int ans = x.getDays();
```

3) ClassCastException - an attempt to cast an object to a class of which it is not an instance

```
Month x = new Month(4);
String name = (String)x;
```

4) ArrayIndexOutOfBoundsException - using an index less than zero or greater than or equal to the length of the array

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

5) IndexOutOfBoundsException - using an index less than zero or greater than or equal to the size of the arraylist

```
ArrayList list = new ArrayList();
for (int j=0; j<4; j++)
      list.add(new Integer(j*j));   // list = 0, 1, 4, 9
list.add(2, new Integer(7));        // list = 0, 1, 7, 4, 9
list.remove(1);                     // list = 0, 7, 4, 9
list.remove(-1);                    // error
```

Unit 5 Assignment 3: Resizing & Inserting into an Array