CS1007 lecture #11 notes

thu 28 feb 2002

- news
- arguments
- arrays
- reading: ch 5.1, 6.1-6.2

news.

- midterm exam back today
- homework#3 posted, due Thu Mar 7

arguments.

- given the example in chapter 4 on pages 179-180
- we modify the example to use arguments and demonstrate the visibility and scope of these variables

Coin class (modified from book p180).

```
public class Coin {

  // declare public constants
  // -- can be seen outside the class
  // -- have global scope inside the class
  public final int HEADS = 0;
  public final int TAILS = 1;

  // declare private variables
  // -- cannot be seen outside the class
  // -- have global scope inside the class
  private int face;
  private int headcount;
  private int tailcount;

  public Coin() {
    headcount = 0;
    tailcount = 0;
    flip( 1 );
  } // end constructor

  public void flip( int numflips ) {
    for ( int i=0; i<numflips; i++ ) {
      face = (int)(Math.random() * 2 );
      if ( face == HEADS ) {
        headcount++;
      }
```

```
      else {
        tailcount++;
      }
    }
  } // end method flip()

  public int getFace() {
    return face;
  } // end method getFace()

  public int getHeadcount() {
    return headcount;
  } // end method getHeadcount()

  public int getTailcount() {
    return tailcount;
  } // end method getTailcount()

  public String toString() {
    String faceName;
    if ( face == HEADS ) {
      faceName = "heads";
    }
    else {
      faceName = "tails";
    }
    return faceName;
  } // end method toString()

} // end class Coin
```

CountFlips class (modified from book p180).

```
public class ex11 {

  public static void main( String[] args ) {

    Coin mycoin = new Coin();
    int numflips = 1000;

    mycoin.flip( numflips );

    System.out.println( "The number of flips: " + numflips );
    System.out.println( "The number of heads: " + mycoin.getHeadcount() );
    System.out.println( "The number of tails: " + mycoin.getTailcount() );

  } // end of main()

} // end class ex11
```

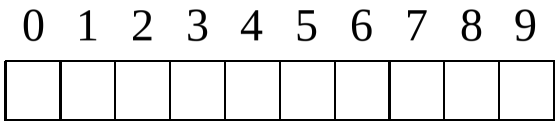
analysis of arguments.

- the variable `numflips` is declared in the main example class (`ex11`) and also as an argument to the `flip()` method in `Coin` class
- in the `main()` method, `numflips` is a local variable, and memory is allocated for it within the scope of `main`
- when `main()` invokes the `flip()` method, the value of main's numflips is used to initialize a newly allocated local variable inside the `flip()` method – the `numflips` argument that is local to flip because it is declared in flip's argument list

arrays (1).

- used to associate multiple instances of the same type of variable
- the "`[]`" indicates it's an *array*
- we can have arrays of anything (i.e., other data types)
- one example we've already used is `String[]`, which is an array of `String`...
- visualize an array as a sequence of boxes, contiguous in the computer's memory, where each box stores one instance of the type of data associated with that array:

- the boxes are numbered, starting with 0 and ending with the length of the array less one; each number is called an *index*
- the *indices* for an array of 10 items can be visualized like this:



arrays (2).

- declaring:

```
int[] A;
```

- instantiating:

```
A = new int[10];
```

- accessing (index=4, which is the 5th item in the array):

```
A[4]
```

  use this accessed item just like any single data element of that type, in this case an `int`

- the number of items in the array is the variable `A.length`

arrays (3).

- let's modify the previous example using an array of `Coin`
- note that when you use an array of a data type other than a primitive data type, you need to instantiate each individual entry in the array as well as the locations of each class element
- you can think of it like a phone book, which contains a list of addresses that give locations of houses, but not the actual houses themselves
- the `Coin[]` variable contains a list of addresses
- first you need to construct locations for the addresses:

```
Coin[] pocket;           // declare
pocket = new Coin[10];   // instantiate
```

or

```
Coin[] pocket = new Coin[10]; // declare and instantiate
```

- then you need to construct each house one at a time:

```
for ( int i=0; i<numcoins; i++ ) {
  pocket[i] = new Coin();
} // end for i
```

array example.

```
public class ex11_2 {

  public static void main( String[] args ) {

    int numcoins = 10;
    Coin[] pocket = new Coin[numcoins];
    int headcount = 0, tailcount = 0;

    // instantiate each of the coins in the array
    for ( int i=0; i<numcoins; i++ ) {
      pocket[i] = new Coin();
    } // end for i

    // count the number of heads and tails in the array
    for ( int i=0; i<numcoins; i++ ) {
      headcount += pocket[i].getHeadcount();
      tailcount += pocket[i].getTailcount();
    } // end for i

    // report results
    System.out.println( "The number of flips: " + numflips );
    System.out.println( "The number of heads: " + headcount );
    System.out.println( "The number of tails: " + tailcount );

  } // end of main()

} // end class ex11_2
```