CS1007 lecture #12 notes

thu 18 oct 2001

*http://www.cs.columbia.edu/~sklar/cs1007*

today:
- news
- `java.lang.System` class
- `java.io` package
- program organization
- writing your own classes
- writing your own methods
- arguments
- reading: *ch 4.1-4.3*



## news.

- strange CUNIX problems... still no answers
- no office hours on Mon Oct 22 – I'll be out of town
- two new TA's – check schedule for more hours on web page
- one less homework — 8 in total
- homework #4 is out, due on WED Oct 24
- reminder: midterm II is on Nov 1



## `java.lang.System` class.

- variables:

```
public static PrintStream err;
public static InputStream in;
public static PrintStream out;
```

- methods:

```
public static long currentTimeMillis();
public static void exit( int num ) throws SecurityException;
```



## `java.io.PrintStream` class.

- methods:

```
public void print( ... );
public void println( ... );
```

- example:

```
public class hello {
  public static void main ( String[] args ) {
    System.out.println( "hello world!\n" );
  } // end main method
} // end hello class
```



## `java.io.InputStream` class.

- methods:

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

- example:

```
import java.lang.*;
import java.io.*;
public class lect12_ex1 {
  public static void main( String[] args ) {
    int i = 0;
    System.out.print( "please type something: " );
    try {
      i = System.in.read();
    }
    catch ( IOException iox ) {
      System.out.println( "there was an error: " + iox );
    }
    System.out.println( "i=" + i );
  } // end of main
} // end class lect12_ex1
```

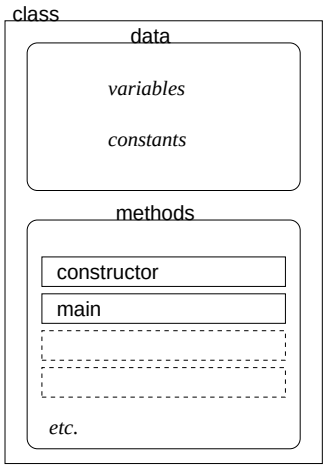


## exception handling, in brief.

```
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*

## program organization.






## a small class.

```
public class Coin {
  // public constants; can be seen outside the class
  public final int HEADS = 0;
  public final int TAILS = 1;
  // private variable; only visible within the class
  private int face;
  // constructor
  public Coin() {
    flip();
  } // end constructor
  // method
  public void flip() {
    face = (int)(Math.random() * 2 );
  } // end method flip()
  // method
  public int getFace() {
    return face;
  } // end method getFace()
  // method
  public String toString() {
    String faceName;
    if ( face == HEADS ) {
      faceName = "heads";
    }
    else {
      faceName = "tails";
    }
    return faceName;
  } // end method toString()
} // end class Coin
```



## writing another class which calls the first class.

```
import java.lang.*;
public class lect12_ex3 {
  public static void main( String[] args ) {
    final int GOAL = 3;
    int count1 = 0, count2 = 0;
    Coin coin1 = new Coin();
    Coin coin2 = new Coin();
    while (( count1 < GOAL ) && ( count2 < GOAL )) {
      coin1.flip();
      coin2.flip();
      System.out.println( "Coin1: " + coin1 + "  Coin2: " + coin2 );
      count1 = ( coin1.getFace() == coin1.HEADS ) ? count1 + 1 : 0;
      count2 = ( coin2.getFace() == coin2.HEADS ) ? count2 + 1 : 0;
    } // end while
    if ( count1 < GOAL )
      System.out.println( "Coin 1 wins!" );
    else if ( count2 < GOAL )
      System.out.println( "Coin 2 wins!" );
    else
      System.out.println( "It's a tie!" );
  } // end main method
// end class lect12_ex3
```



## method arguments, in brief.

- let's modify the `flip()` method to take an *argument* indicating how many times to flip the coin:

```
public void flip( int n ) {
  for ( int i=0; i<n; i++ ) {
    face = (int)(Math.random() * 2 );
  }
} // end method flip()
```

- this method is then *called* or *invoked* by putting in a value for the argument, in this case `int n`

```
...
flip( 7 );
...
```

- inside `flip()`, the int argument n now has the value 7