CS 211 - 3/17/17

Friday, March 17, 2017 9:59 AM

From Wednesday: Constructors

```
class X
{
  private int dataMember;  // initialization NOT done here

  public X ( )
   {
    dataMember = 0;     // initialization done in Constructors
     …
   }

 …..
}
```

If no constructors are written for a class, Java will create the default constructor which will zero out all data members.

Once any constructor is written for a Java class, then this default "default construction" is NOT created by the Java compiler.

```
class Fraction
{
   private int num;
   private int den;

   //    f3 = f1 + f2;      - Mathematical expression
   //    f3 = f1.add (f2) - typical Java Expression
   //         f3 is the return value in the method add()
   //         f1 is the value for the implicit this parameter
   //         f2 is the explicit parameter rhs.
   public Fraction add ( Fraction rhs )
    {
        int local_var_num;
        int iv, jv;

        int num1 = num;
        int num2 = rhs.num;
        int den1= this.den;
        int den2= rhs.den;
```

```
        while ( true )
         {
             double jv;

              iv = 0;
              jv = 0;
        }
      ….
      }
```

```
   // add2 - a different approach to the add method above
  //    Since add2 () is a STATIC method, there is no implicit this parameter
  //    f3 = f1 + f2;
  //     f3 = Fraction.add2 (f1, f2);
  //      f3 = f3.add2 (  f1, f2 );      <===   valid but UNDESIRED way to write the call
  public static Fraction add2 ( Fraction lhs, Fraction rhs )
   { ….. }
```

What is the difference between writing code as

```
  //     f3 = Fraction.add2 (f1, f2);
```

compared to

```
   //    f3 = f1.add (f2); - typical Java Expression, normally preferred over the above expression
```

==========

```
class MyStack
{
  private MyNode head;

  public MyStack()
  {
    head = null;
  }

  public boolean isEmpty()
  {
    if (head == null)
      return true;
    else
```

```
    return false;
  }

  public void push (int v1)
  {
   MyNode tmp = new MyNode (v1, head);
   head = tmp;
  }

  public void pop()
  {
   head = head.getNext();
  }

  public int top()
  {
   return head.getElem();
  }


private class MyNode
{
  int elem;
  MyNode next;

  public MyNode ( )
  {
   elem = -999;
   next = null;
  }

  public MyNode (int v1)
  {
   elem = v1;
   next = null;
  }

  public MyNode (int v1, MyNode n)
  {
   elem = v1;
   next = n;
  }

  public void setElem ( int v1)
```

```
  {
    elem = v1;
  }

  public int getElem ( )
  {
    return elem;
  }

  public void setNext ( MyNode n)
  {
    next = n;
  }

  public MyNode getNext ( )
  {
    return next;
  }
} // end of the MyNode class


} // end of the MyStack class
```