/\*\*

\* A class to provide a simple list of integers.

\* List resizes automatically. Used to illustrate

\* various design and implementation details of

\* a class in Java.

\*

\* Version 1 only contains the instance variables and

\* the constructors

\* @author scottm

\*

\*/

public class IntListVer2{

// class constant for default size

private static final int DEFAULT\_CAP = 10;

//instance variables

// iValues store the elements of the list and

// may have extra capacity

private int[] iValues;

private int iSize;

/\*\*

\* Default add method. Add x to the end of this IntList.

\* Size of the list goes up by 1.

\* @param x The value to add to the end of this list.

\*/

public void add(int x){

// is there extra capacity available?

// if not, resize

if(iSize == iValues.length)

resize();

assert 0 <= iSize && iSize < iValues.length;

iValues[iSize] = x;

iSize++;

}

// resize internal storage container by a factor of 2

private void resize() {

int[] temp = new int[iValues.length \* 2];

System.arraycopy(iValues, 0, temp, 0, iValues.length);

iValues = temp;

}

/\*\*

\* Return a String version of this list. Size and

\* elements included.

\*/

public String toString(){

// we could make this more effecient by using a StringBuffer.

// See alternative version

String result = "size: " + iSize + ", elements: [";

for(int i = 0; i < iSize - 1; i++)

result += iValues[i] + ", ";

if(iSize > 0 )

result += iValues[iSize - 1];

result += "]";

return result;

}

// Would not really have this and toString available

// both included just for testing

public String toStringUsingStringBuffer(){

StringBuffer result = new StringBuffer();

result.append( "size: " );

result.append( iSize );

result.append(", elements: [");

for(int i = 0; i < iSize - 1; i++){

result.append(iValues[i]);

result.append(", ");

}

if( iSize > 0 )

result.append(iValues[iSize - 1]);

result.append("]");

return result.toString();

}

/\*\*

\* Default constructor. Creates an empty list.

\*/

public IntListVer2(){

//redirect to single int constructor

this(DEFAULT\_CAP);

//other statments could go here.

}

/\*\*

\* Constructor to allow user of class to specify

\* initial capacity in case they intend to add a lot

\* of elements to new list. Creates an empty list.

\* @param initialCap > 0

\*/

public IntListVer2(int initialCap) {

assert initialCap > 0 : "Violation of precondition. IntListVer1(int initialCap):"

+ "initialCap must be greater than 0. Value of initialCap: " + initialCap;

iValues = new int[initialCap];

iSize = 0;

}

/\*\*

\* Return true if this IntList is equal to other.<br>

\* pre: none

\* @param other The object to comapre to this

\* @return true if other is a non null, IntList object

\* that is the same size as this IntList and has the

\* same elements in the same order, false otherwise.

\*/

public boolean equals(Object other){

boolean result;

if(other == null)

// we know this is not null so can't be equal

result = false;

else if(this == other)

// quick check if this and other refer to same IntList object

result = true;

else if( this.getClass() != other.getClass() )

// other is not an IntList they can't be equal

result = false;

else{

// other ris not null and refers to an IntList

IntListVer2 otherIntList = (IntListVer2)other;

result = this.iSize == otherIntList.iSize;

int i = 0;

while(i < iSize && result){

result = this.iValues[i] == otherIntList.iValues[i];

i++;

}

}

return result;

}

}