1. **Program to create ASCII frequency table from file and url**

```
   import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.net.MalformedURLException;
import java.net.URL;


public class FreqTableExampleOriginal {

   public static final int NUM_ASCII_CHAR = 128;

   // program to create a frequency table.
   // Example of simple try catch blocks to deal with checked exceptions
   public static void main(String[] args)
      {

      int[] freqs = createFreqTableURL("http://www.utexas.edu/");

      if( freqs.length == 0)
         System.out.println("No frequency table created due to problems when reading from file");
      else{
         for(int i = 0; i < NUM_ASCII_CHAR; i++){
            System.out.println("charcater code: " + i + " ,character: " + (char)i + " ,frequency: " +
freqs[i]);
         }
         System.out.println("Total characters in file: " + sum(freqs));
      }


      freqs = new int[]{};
      try{
         freqs = createTable("ciaFactBook2008.txt");
      }
      catch(FileNotFoundException e){
         System.out.println("File not found. Unable to create freq table" + e);
      }
      catch(IOException e){
         System.out.println("Problem while reading from file. Unable to create freq table" + e);
      }
      if( freqs.length == 0)
         System.out.println("No frequency table created due to problems when reading from file");
      else{
```

```
            for(int i = 0; i < freqs.length; i++){
                System.out.println("charcater code: " + i + " ,character: " + (char)i + " ,frequency: " +
freqs[i]);
            }
            System.out.println("Total characters in file: " + sum(freqs));
        }

    }


    // return sum of ints in list
    // list may not be null
    private static int sum(int[] list) {
        assert list != null : "Failed precondition, sum: parameter list" +
            " may not be null.";
        int total = 0;
        for(int x : list){
            total += x;
        }
        return total;
    }


    // pre: url != null
    // Connect to the URL specified by the String url.
    // Map characters to index in array.
    // All non ASCII character dumped into one element of array
    // If IOException occurs message printed and array of
    // length 0 returned.
    public static int[] createFreqTableURL (String url){
        if(url == null)
            throw new IllegalArgumentException("Violation of precondition. parameter url must not be
null.");

        int[] freqs = new int[NUM_ASCII_CHAR];
        try {
        URL inputURL = new URL(url);
        InputStreamReader in
            = new InputStreamReader(inputURL.openStream());

        while(in.ready()){
            int c = in.read();
            if(0 <= c && c < freqs.length)
                freqs[c]++;
            else
                System.out.println("Non ASCII char: " + c + " " + (char) c);
```

```
        }
        in.close();
        }
        catch(MalformedURLException e){
            System.out.println("Bad URL.");
            freqs = new int[0];
        }
        catch(IOException e){
            System.out.println("Unable to read from resource." + e);
            freqs = new int[0];
        }
        return freqs;
    }



    // Connect to the file specified by the String fileName.
    // Assumes it is in same directory as compiled code.
    // Map characters to index in array.
    public static int[] createTable(String fileName) throws FileNotFoundException, IOException{
        int[] freqs = new int[NUM_ASCII_CHAR];
        File f = new File(fileName);
        FileReader r = new FileReader(f);
        while( r.ready() ){
            int ch = r.read();
//            if( 0 <= ch && ch <= NUM_ASCII_CHAR)
//                freqs[ch]++;
//            else
//                freqs[INDEX_NON_ASCII]++;
            if(0 <= ch && ch < freqs.length)
                freqs[ch]++;
            else
                System.out.println((char) ch);

        }
        r.close();
        return freqs



}
```

2. IntListVer1

```
    /**
 * 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 IntListVer1 {
    // class constant for default size
    private static final int DEFAULT_CAP = 10;

    //instance variables
    private int[] iValues;
    private int iSize;

    /**
     * Default constructor. Creates an empty list.
     */
    public IntListVer1(){
        //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 IntListVer1(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;
    }
}
```

3. IntListTesterVer1

```
public class IntListTesterVer1 {
    public static void main(String[] args){
```

```
        IntListVer1 list1 = new IntListVer1();
        IntListVer1 list2 = new IntListVer1(100);

        //equal when empty?
        System.out.println("list1.equals(list2): " + list1.equals(list2));
        System.out.println("list1: " + list1);
        System.out.println("list2: " + list2);
    }
}
```

4. IntListVer2

```
    /**
 * 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;
    }

}
```

5. IntListTesterVer2

```
public class IntListTesterVer2 {
    public static void main(String[] args){
        IntListVer2 list1 = new IntListVer2();
        IntListVer2 list2 = new IntListVer2(100);

        //equal when empty?
        System.out.println("list1.equals(list2): " + list1.equals(list2));
        System.out.println("list1: " + list1);
        System.out.println("list2: " + list2);

        //add elements
        for(int i = 0; i < 100; i += 5){
            list1.add(i);
            list2.add(i);
        }

        System.out.println("list1.equals(list2): " + list1.equals(list2));
        System.out.println("list1: " + list1);
        System.out.println("list2: " + list2);

        list2.add(200);
        System.out.println("Added 200 to list2.");
        System.out.println("list1.equals(list2): " + list1.equals(list2));
        System.out.println("list1: " + list1);
        System.out.println("list2: " + list2);

        System.out.println("Testing efficieny of StringBuffer versus using String.");
        System.out.println("Increasing list1 size to 10000.");
        Stopwatch s = new Stopwatch();
        list1 = new IntListVer2();
        for(int i = 0; i < 10000; i++)
            list1.add(i);
        s.start();
        list1.toString();
        s.stop();
        System.out.println("Time to build String using String class: " + s.toString() );
        s.start();
```

```
        list1.toStringUsingStringBuffer();
        s.stop();
        System.out.println("Time to build String using StringBuffer class: " + s.toString() );

    }
}
```

6. IntListVer3

```
    /**
 * 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 3 added the insert and remove methods. Changed the
 * add method to rely on insert.
 * @author scottm
 *
 */
public class IntListVer3{
    // 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 constructor. Creates an empty list.
     */
    public IntListVer3(){
        //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 IntListVer3(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;
}

/**
 * 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){
    //example of loose coupling
    insert(iSize, x);
}

/**
 * Retrieve an element from the list based on position.
 * @param pos 0 <= pos < size()
 * @return The element at the given position.
 */
public int get(int pos){
    assert 0 <= pos && pos < size() : "Failed precondition get. " +
                    "pos it out of bounds. Value of pos: " + pos;
    return iValues[pos];
}

/**
 * Insert x at position pos. Elements with a position equal
 * to pos or more are shifted to the right. (One added to their
 * position.)
 * post: get(pos) = x, size() = old size() + 1
 * @param pos 0 <= pos <= size()
 * @param x
 */
public void insert(int pos, int x){
    assert 0 <= pos && pos <= size() : "Failed precondition insert. " +
    "pos is invalid. Value of pos: " + pos;
    ensureCapcity();
    for(int i = iSize; i > pos; i--){
        iValues[i] = iValues[i - 1];
    }
    iValues[pos] = x;
```

```
    iSize++;
}

/**
 * Remove an element from the list based on position.
 * Elements with a position greater than pos
 * are shifted to the left. (One subtracted from their
 * position.)
 * @param pos 0 <= pos < size()
 * @return The element that is removed.
 */
public int remove(int pos){
    assert 0 <= pos && pos < size() : "Failed precondition remove. " +
    "pos it out of bounds. Value of pos: " + pos;
    int removedValue = iValues[pos];
    for(int i = pos; i < iSize - 1; i++)
        iValues[i] = iValues[i + 1];
    iSize--;
    return removedValue;
}

private void ensureCapcity(){
    // is there extra capacity available?
    // if not, resize
    if(iSize == iValues.length)
        resize();
}

/**
 * Returns the size of the list.
 * @return The size of the list.
 */
public int size(){
    return 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();
 }

/**
 * 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 is not null and refers to an IntList
            IntListVer3 otherIntList = (IntListVer3)other;
            result = this.size() == otherIntList.size();
            int i = 0;
            while(i < iSize && result){
                result = this.iValues[i] == otherIntList.iValues[i];
                i++;
            }
        }
        return result;
    }

}
```

7. SortedIntList

```
    public class SortedIntList extends IntListVer3{

    public SortedIntList(int initialCap){
        //call IntList constructor
        super(initialCap);
    }

    public SortedIntList(){
        super();
    }

    //override add
    public void add(int value){
        //search for location to insert value
        int pos = 0;
        while( pos < size() && value > get(pos) ){
            pos++;
        }
        super.insert(pos, value);
    }

}
```

8. GenericList

```
  /**
 * A class to provide a simple list.
 * List resizes automatically. Used to illustrate
 * various design and implementation details of
 * a class in Java.
 *
 * @author scottm
 *
 */
public class GenericList{
    // 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 Object[] 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(Object x){
        insert(iSize, x);
    }

    public Object get(int pos){
        return iValues[pos];
    }

    /**
     * Insert obj at position pos.
     * post: get(pos) = x, size() = old size() + 1
     * @param pos 0 <= pos <= size()
     * @param obj The element to add.
     */
    public void insert(int pos, Object obj){
        ensureCapcity();
        for(int i = iSize; i > pos; i--){
            iValues[i] = iValues[i - 1];
        }
        iValues[pos] = obj;
```

```
    iSize++;
}

public Object remove(int pos){
    Object removedValue = iValues[pos];
    for(int i = pos; i < iSize - 1; i++)
        iValues[i] = iValues[i + 1];
    iValues[iSize - 1] = null;
    iSize--;
    return removedValue;
}

private void ensureCapcity(){
    // is there extra capacity available?
    // if not, resize
    if(iSize == iValues.length)
        resize();
}

public int size(){
    return iSize;
}

// resize internal storage container by a factor of 2
private void resize() {
    Object[] temp = new Object[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].toString() + ", ";
    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 GenericList(){
    //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 GenericList(int initialCap) {
    assert initialCap > 0 : "Violation of precondition. IntListVer1(int initialCap):"
        + "initialCap must be greater than 0. Value of initialCap: " + initialCap;
    iValues = new Object[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 is not null and refers to an IntList
            GenericList otherList = (GenericList)other;
            result = this.size() == otherList.size();
            int i = 0;
            while(i < iSize && result){
                result = this.iValues[i].equals( otherList.iValues[i] );
                i++;
            }
        }
        return result;
    }

}
```

9. Die class

```
    import java.util.Random;

/**
 * Models a playing die with sides numbered 1 to N.
 * All sides have uniform probablity of being rolled.
 *
 * @author Summer CS 307 class
 */
public class Die
{   public static final int DEFAULT_SIDES = 6;

    private static Random ourRandNumGen = new Random();

    private final int iMyNumSides;
    private int iMyResult;
```

```
    /**
     * Default constructor.<p>
     * pre: none<br>
     * post: getNumSides() = DEFAULT_SIDES, getResult() = 1
     */
    public Die()
    {   this(DEFAULT_SIDES);
    }


    /**
     * Create a Die with numSides sides<p>
     * pre: numSides > 1<br>
     * post: getNumSides() = numSides, getResult() = 1<br>
     * An exception will be generated if the preconditions are not met
     */
    public Die(int numSides)
    {   assert numSides > 1 : "Violation of precondition: numSides = " + numSides + "numSides must
be greater than 1";

        iMyNumSides = numSides;
        iMyResult = 1;
        assert getResult() == 1 && getNumSides() == numSides;
    }


    /**
     * Create a Die with numSides and top side and result set to result<p>
     * pre: numSides > 1, 1 <= result <= numSides<br>
     * post: getNumSides() = numSides, getResult() = 1<br>
     * An exception will be generated if the preconditions are not met
     */
    public Die(int numSides, int result)
    {   assert numSides > 1 && 1 <= result && result <= numSides : "Violation of precondition";

        iMyNumSides = numSides;
        iMyResult = result;
    }


    /**
     * roll this Die. Every side has an equal chance of being the new result<p>
     * pre: none<br>
     * post: 1 <= getResult() <= getNumSides()
     * @return the result of the Die after the roll
     */
```

```
    public int roll()
    {   iMyResult = ourRandNumGen.nextInt(iMyNumSides) + 1;

        assert ( 1 <= getResult() ) && ( getResult() <= getNumSides() );

        return iMyResult;
    }


    /**
     * return how many sides this Die has<p>
     * pre: none<br>
     * post: return how many sides this Die has
     * @return the number of sides on this Die
     */
    public int getNumSides()
    {   return iMyNumSides; }


    /**
     * get the current result or top number of this Die<p>
     * pre: none<br>
     * post: return the number on top of this Die
     * @return the current result of this Die
     */
    public int getResult()
    {   return iMyResult;   }


    /**
     * returns true if this Die and the parameter otherObj are equal<p>
     * pre: none<br>
     * post: return true if the parameter is a Die object with the same number of sides as this Die and
currently has the same result.
     * @return true if the the two Dice are equal, false otherwise
     */
    public boolean equals(Object otherObj)
    {   boolean result = true;
        if(otherObj == null)
            result = false;
        else if(this == otherObj)
            result = true;
        else if(this.getClass() != otherObj.getClass())
            result = false;
        else
        {   Die otherDie = (Die)otherObj;
```

```
            result = this.iMyResult == otherDie.iMyResult
                && this.iMyNumSides == otherDie.iMyNumSides;
        }
        return result;
    }


    /**
     * returns a String containing information about this Die<p>
     * pre: none<br>
     * post: return a String with information about the current state of this Die
     * @return: A String with the number of sides and current result of this Die
     */
    public String toString()
    {   return "Num sides " + getNumSides() + " result " + getResult();
    }


}// end of Die class
```

10. DemoClass

```
    /*****************************************************************/
/* Author: CS307 Course Staff                                    */
/* Date: February 14, 2005                                       */
/* Description: Demos constructors, static vs instance methods,   */
/*              and method overloading.                          */
/*****************************************************************/
public class DemoClass
{
    private int x;

    public DemoClass()
    {
        // assign default value
        x = 0;
    }

    public DemoClass(int x)
    {
        // use this.x to refer to the instance variable x
        // use x to refer to a local variable x (more specifically,
        // method parameter x)
```

```
        this.x = x;
    }

    public DemoClass(DemoClass otherDemo)
    {
        // copy the value from the otherDemo
        this.x = otherDemo.x;
    }

    // static method (aka class method)
    public static void s1() {
        return;
    }
    // instance method
    public void i1() {
        return;
    }

    // static calling static OK
    // static calling instance is a compile-time error
    public static void s2() {
//      i1();     // compile-time error
        s1();      // DemoClass.s1
        return;
    }

    // instance calling static OK
    // instance calling instance OK
    public void i2() {
        s1();      // DemoClass.s1();
        i1();      // this.i1();
        return;
    }

    // call various versions of overload() based on their
    // list of parameters (aka function signatures)
    public void overloadTester() {
        System.out.println("overloadTester:\n");

        overload((byte)1);
        overload((short)1);
        overload(1);
        overload(1L);
        overload(1.0f);
        overload(1.0);
        overload('1');
```

```
        overload(true);
    }

    public void overload(byte b) {
        System.out.println("byte");
    }
    public void overload(short s) {
        System.out.println("short");
    }
    public void overload(int i) {
        System.out.println("int");
    }
    public void overload(long l) {
        System.out.println("long");
    }
    public void overload(float f) {
        System.out.println("float");
    }
    public void overload(double d) {
        System.out.println("double");
    }
    public void overload(char c) {
        System.out.println("char");
    }
    public void overload(boolean b) {
        System.out.println("boolean");
    }

    public static void main(String[] args) {
        DemoClass dc = new DemoClass();
        dc.overloadTester();
    }
}

// end of DemoClass.java
```

11. Stopwatch class

```
    /*****************************************************************/
/* Author: CS307 Course Staff                                    */
/* Date: February 14, 2005                                       */
/* Description: Demos constructors, static vs instance methods,   */
/*              and method overloading.                          */
/*****************************************************************/
public class DemoClass
{
```

```
    private int x;

    public DemoClass()
    {
        // assign default value
        x = 0;
    }

    public DemoClass(int x)
    {
        // use this.x to refer to the instance variable x
        // use x to refer to a local variable x (more specifically,
        // method parameter x)
        this.x = x;
    }

    public DemoClass(DemoClass otherDemo)
    {
        // copy the value from the otherDemo
        this.x = otherDemo.x;
    }

    // static method (aka class method)
    public static void s1() {
        return;
    }
    // instance method
    public void i1() {
        return;
    }

    // static calling static OK
    // static calling instance is a compile-time error
    public static void s2() {
//      i1();     // compile-time error
        s1();      // DemoClass.s1
        return;
    }

    // instance calling static OK
    // instance calling instance OK
    public void i2() {
        s1();      // DemoClass.s1();
        i1();      // this.i1();
        return;
    }
```

```
    // call various versions of overload() based on their
    // list of parameters (aka function signatures)
    public void overloadTester() {
        System.out.println("overloadTester:\n");

        overload((byte)1);
        overload((short)1);
        overload(1);
        overload(1L);
        overload(1.0f);
        overload(1.0);
        overload('1');
        overload(true);
    }

    public void overload(byte b) {
        System.out.println("byte");
    }
    public void overload(short s) {
        System.out.println("short");
    }
    public void overload(int i) {
        System.out.println("int");
    }
    public void overload(long l) {
        System.out.println("long");
    }
    public void overload(float f) {
        System.out.println("float");
    }
    public void overload(double d) {
        System.out.println("double");
    }
    public void overload(char c) {
        System.out.println("char");
    }
    public void overload(boolean b) {
        System.out.println("boolean");
    }

    public static void main(String[] args) {
        DemoClass dc = new DemoClass();
        dc.overloadTester();
    }
}
```

```
// end of DemoClass.java
```

a. Documentation for Stopwatch class

```
    /**
 A class to measure time elapsed.
*/

public class Stopwatch
{
   private long startTime;
   private long stopTime;

   public static final double NANOS_PER_SEC = 1000000000.0;

          /**
           start the stop watch.
          */
          public void start(){
                    startTime = System.nanoTime();
          }

          /**
           stop the stop watch.
          */
          public void stop()
          {         stopTime = System.nanoTime();    }

          /**
          elapsed time in seconds.
          @return the time recorded on the stopwatch in seconds
          */
          public double time()
          {         return (stopTime - startTime) / NANOS_PER_SEC; }

          public String toString(){
             return "elapsed time: " + time() + " seconds.";
          }

          /**
          elapsed time in nanoseconds.
          @return the time recorded on the stopwatch in nanoseconds
```

```
            */
            public long timeInNanoseconds()
            {           return (stopTime - startTime);       }
}
```

12. Create a Set

```
  import java.util.Arrays;
import java.awt.Rectangle;

/**
 * A sample of a ploymorphic method.
 * @author scottm
 *
 */
public class CreateASet {

    public static void main(String[] args){
        String[] words = {"A", "B", "B", "D", "C", "A"};
        System.out.println( "original: " + Arrays.toString(words));
        System.out.println( "as a set: " + Arrays.toString(makeSet(words)));

        Rectangle[] rectList = {new Rectangle(), new Rectangle(),
                new Rectangle(0, 1, 2, 3), new Rectangle(0, 1, 2, 3)};
        System.out.println( "original: " + Arrays.toString(rectList));
        System.out.println( "as a set: " + Arrays.toString(makeSet(rectList)));


        Object[] mixed = {"A", "C", "A", "B", new Rectangle(),
              new Rectangle(), "A", new Rectangle(0, 1, 2, 3), "D"};
        System.out.println( "original: " + Arrays.toString(mixed));
        System.out.println( "as a set: " + Arrays.toString(makeSet(mixed)));
    }

    /**
     * An example of polymorphism in action. The method relies
     * on Java's inheritance requirement and polymorhphism to call
     * the correct equals method.
     * @param data != null, no elements of data are null
     * @return a Set (no duplicates) of the elements in data.
     */
    public static Object[] makeSet(Object[] data){
        assert data != null : "Failed precondition makeSet. parameter cannot be null";
```

```
        assert noNulls(data) : "Failed precondition makeSet. no elements of parameter can be null";
        Object[] result = new Object[data.length];
        int numUnique = 0;
        boolean found;
        int indexInResult;
        for(int i = 0; i < data.length; i++){
            // maybe should break this out into another method
            indexInResult = 0;
            found = false;
            while(!found && indexInResult < numUnique){
                found = data[i].equals(result[indexInResult]);
                indexInResult++;
            }
            if( ! found ){
                result[numUnique] = data[i];
                numUnique++;
            }
        }
        Object[] result2 = new Object[numUnique];
        System.arraycopy(result, 0, result2, 0, numUnique);
        return result2;
    }

    // pre: data != null
    // return true if all elements of data are non null,
    // false otherwise
    private static boolean noNulls(Object[] data){
        assert data != null : "Failed precondition makeSet. parameter cannot be null";
        boolean good = true;
        int i = 0;
        while( good && i < data.length ){
            good = data[i] != null;
            i++;
        }
        return good;
    }

}
```

Recursion examples

```
  public class RecursionExampleDirectory
{
            public int getSize(Directory dir)
            {           int total = 0;

                        //check files
```

```
		File[] files = dir.getFiles();
		for(int i = 0; i < files.length; i++)
			total += files[i].getSize();

		//get sub directories and check them
		Directory[] subs = dir.getSubs();
		for(int i = 0; i < subs.length; i++)
			total += getSize(subs[i]);

		return total;
}

public static void main(String[] args)
{	RecursionExampleDirectory r = new RecursionExampleDirectory();
		Directory d = new Directory();
		System.out.println( r.getSize(d) );
}

//pre: n >= 0
public static int fact(int n)
{	int result = 0;
		if(n == 0)
			result = 1;
		else
			result = n * fact(n-1);
		return result;
}

//pre: exp >= 0
public static int pow(int base, int exp)
{	int result = 0;
		if(exp == 0)
			result = 1;
		else
			result = base * pow(base, exp - 1);
		return result;
}

//slow fib
//pre: n >= 1
public static int fib(int n)
{	int result = 0;
		if(n == 1 || n == 2)
			result = 1;
		else
			result = fib(n-1) + fib(n-2);
```

```
		return result;
	}

	public static int minWasted(int[] items, int itemNum, int capLeft)
	{	int result = 0;
		if(itemNum >= items.length)
			result = capLeft;
		else if( capLeft == 0)
			result = 0;
		else
		{	int minWithout = minWasted(items, itemNum + 1, capLeft);
			if( capLeft <= items[itemNum])
			{	int minWith = minWasted(items, itemNum + 1, capLeft -
items[itemNum]);
				result = Math.min(minWith, minWithout);
			}
			else
				result = minWithout;
		}
		return result;
	}
}

class Directory
{	private Directory[] mySubs;
	private File[] myFiles;

	public Directory()
	{	int numSubs = (int)(Math.random() * 3);
		mySubs = new Directory[numSubs];
		int numFiles = (int)(Math.random() * 10);
		myFiles = new File[numFiles];

		for(int i = 0; i < myFiles.length; i++)
			myFiles[i] = new File( (int)(Math.random() * 1000 ) );
		for(int i = 0; i < mySubs.length; i++)
			mySubs[i] = new Directory();
	}

	public Directory[] getSubs()
	{	return mySubs;
	}

	public File[] getFiles()
	{	return myFiles;
	}
```

```
}

class File
{	private int iMySize;

	public File(int size)
	{	iMySize = size;
	}

	public int getSize()
	{	return iMySize;
	}
}
```

13. Eight Queens example

```
    import java.util.Arrays;
import java.util.ArrayList;

public class EightQueens {


	public static void main(String[] args) {
			solveNQueens(8);
			ArrayList<char[][]> solutions = getAllNQueens(8);
			System.out.println( solutions.size() );
			for( int i = 0; i < solutions.size(); i++){
				System.out.println("\n\nSolution " + (i+1));
				if( queensAreSafe(solutions.get(i)) )
					printBoard(solutions.get(i));
				else
					System.out.println("UH OH!!!!! BETTER FIX IT!!!!!");

	}

	/**
	 * determine if the chess board represented by board is a safe set up
	 * <p>pre: board != null, board.length > 0, board is a square matrix.
	 * (In other words all rows in board have board.length columns.),
	 * all elements of board == 'q' or '.'. 'q's represent queens, '.'s
	 * represent open spaces.<br>
	 * <p>post: return true if the configuration of board is safe,
	 * that is no queen can attack any other queen on the board.
	 * false otherwise.
	 * @param board the chessboard
	 * @return true if the configuration of board is safe,
```

```
 * that is no queen can attack any other queen on the board.
 * false otherwise.
*/
public static boolean queensAreSafe(char[][] board)
{	char[] validChars = {'q', '.'};
	assert (board != null) && (board.length > 0)
				&& isSquare(board) && onlyContains(board, validChars)
				: "Violation of precondition: queensAreSafe";


	return true;
}

public static ArrayList<char[][]> getAllNQueens(int size){
	ArrayList<char[][]> solutions = new ArrayList<char[][]>();
	char[][] board = blankBoard(size);
	solveAllNQueens(board, 0, solutions);
	return solutions;
}

public static void solveAllNQueens(char[][] board, int col, ArrayList<char[][]> solutions){
	// am I done? if so, add this solution to the ArrayList of solutions
	if( col == board.length){
		solutions.add( makeCopy(board));
		// all done
	} else {
		for(int row = 0; row < board.length; row++){
			// place queen
			board[row][col] = 'q';
			if( queensAreSafe(board) )
				// if safe go on to next column
				solveAllNQueens(board, col + 1, solutions);
			board[row][col] = '.';
		}
	}
}

// pre: mat != null, mat is rectangular
public static char[][] makeCopy(char[][] mat){
	assert mat != null;
	char[][] copy = new char[mat.length][mat[0].length];
	for(int r = 0; r < mat.length; r++)
		for(int c = 0; c < mat[0].length; c++)
			copy[r][c] = mat[r][c];
	return copy;
}
```

```
public static void printBoard(char[][] board){
	for(int r = 0; r < board.length; r++){
		for(int c = 0; c < board[r].length; c++)
			System.out.print(board[r][c]);
		System.out.println();
	}
}

public static void solveNQueens(int n){
	char[][] board = blankBoard(n);
	//start in column 0
	boolean solved = canSolve(board, 0);
	if( solved ){
		System.out.println("Solved the " + n + " queen problem.");
		printBoard(board);
	}
	else
		System.out.println("Can't solve the " + n + " queen problem.");
}

public static boolean
	canSolve(char[][] board, int col){


	//know when you are done!
	if( col == board.length)
		return true; // solved!!!!!

	// not done, try all the rows
	boolean solved = false;
	for(int row = 0; row < board.length && !solved; row++){
		//System.out.println(row + " " + col);
		// place queen
		board[row][col] = 'q';
		if( queensAreSafe(board) )
			solved = canSolve(board, col + 1);
		if( !solved )
			board[row][col] = '.';
	}
	return solved; //could be true(solved) or false(not solved)!!
}


private static char[][] blankBoard(int size){
	char[][] result = new char[size][size];
```

```
        for(int r = 0; r < size; r++)
            Arrays.fill(result[r], '.');
        return result;
}

private static boolean inbounds(int row, int col, char[][] mat){
        return row >= 0 && row < mat.length && col >= 0 && col < mat[0].length;
}

/* pre: mat != null
   post: return true if mat is a square matrix, false otherwise
*/
private static boolean isSquare(char[][] mat)
{       assert mat != null : "Violation of precondition: isSquare";

        final int numRows = mat.length;
        int row = 0;
        boolean square = true;
        while( square && row < numRows )
        {       square = ( mat[row] != null) && (mat[row].length == numRows);
                row++;
        }
        return square;
}

/* pre: mat != null, valid != null
   post: return true if all elements in mat are one of the characters in valid
*/
private static boolean onlyContains(char[][] mat, char[] valid)
{       assert mat != null && valid != null : "Violation of precondition: onlyContains";

        int row = 0;
        int col;
        boolean correct = true;
        while( correct && row < mat.length)
        {       col = 0;
                while(correct && col < mat[row].length)
                {       correct = contains(valid, mat[row][col]);
                        col++;
                }
                row++;
        }
        return correct;
}

/* pre: list != null
```

```
        post: return true if c is in list
    */
    private static boolean contains(char[] list, char c)
    {       assert ( list != null ) : "Violation of precondition: contains";

            boolean found = false;
            int index = 0;
            while( !found && index < list.length)
            {       found = list[index] == c;
                    index++;
            }
            return found;
    }

}
```

14. Airlines example

```
   import java.util.ArrayList;
import java.util.Scanner;
import java.io.File;
import java.io.IOException;
import java.util.Arrays;

public class AirlineProblem {

   public static void main(String[] args){
      Scanner scannerToReadAirlines = null;
      try{
         scannerToReadAirlines = new Scanner(new File("airlines.txt"));
      }
      catch(IOException e){
         System.out.println("Could not connect to file airlines.txt.");
         System.exit(0);
      }
      if(scannerToReadAirlines != null){
         ArrayList<Airline> airlinesPartnersNetwork = new ArrayList<Airline>();
         Airline newAirline;
         String lineFromFile;
         String[] airlineNames;

         while( scannerToReadAirlines.hasNext() ){
            lineFromFile = scannerToReadAirlines.nextLine();
            airlineNames = lineFromFile.split(",");
            newAirline = new Airline(airlineNames);
            airlinesPartnersNetwork.add( newAirline );
```

```
        }
        System.out.println(airlinesPartnersNetwork);
        Scanner keyboard = new Scanner(System.in);
        System.out.print("Enter airline miles are on: ");
        String start = keyboard.nextLine();
        System.out.print("Enter goal airline: ");
        String goal = keyboard.nextLine();
        ArrayList<String> pathForMiles = new ArrayList<String>();
        ArrayList<String> airlinesVisited = new ArrayList<String>();
        if( canRedeem(start, goal, pathForMiles, airlinesVisited, airlinesPartnersNetwork))
            System.out.println("Path to redeem miles: " + pathForMiles);
        else
            System.out.println("Cannot convert miles from " + start + " to " + goal + ".");
    }
}

private static boolean canRedeem(String current, String goal,
        ArrayList<String> pathForMiles, ArrayList<String> airlinesVisited,
        ArrayList<Airline> network){
    if(current.equals(goal)){
        //base case 1, I have found a path!
        pathForMiles.add(current);
        return true;
    }
    else if(airlinesVisited.contains(current))
        // base case 2, I have already been here
        // don't go into a cycle
        return false;
    else{
        // I have not been here and it isn't
        // the goal so check its partners
        // now I have been here
        airlinesVisited.add(current);

        // add this to the path
        pathForMiles.add(current);

        // find this airline in the network
        int pos = -1;
        int index = 0;
        while(pos == -1 && index < network.size()){
            if(network.get(index).getName().equals(current))
                pos = index;
            index++;
        }
        //if not in the network, no partners
```

```
        if( pos == - 1)
            return false;

        // loop through partners
        index = 0;
        String[] partners = network.get(pos).getPartners();
        boolean foundPath = false;
        while( !foundPath && index < partners.length){
            foundPath = canRedeem(partners[index], goal, pathForMiles, airlinesVisited, network);
            index++;
        }
        if( !foundPath )
            pathForMiles.remove( pathForMiles.size() - 1);
        return foundPath;
    }
}

private static class Airline{
    private String name;
    private ArrayList<String> partners;

    //pre: data != null, data.length > 0
    public Airline(String[] data){
        assert data != null && data.length > 0 : "Failed precondition";
        name = data[0];
        partners = new ArrayList<String>();
        for(int i = 1; i < data.length; i++)
            partners.add( data[i] );
    }

    public String[] getPartners(){
        return partners.toArray(new String[partners.size()]);
    }

    public boolean isPartner(String name){
        return partners.contains(name);
    }

    public String getName(){
        return name;
    }

    public String toString(){
        return name + ", partners: " + partners;
    }
}
```

```
}
```

15. Minesweeper

```
   public class MineSweeper
{          private int[][] myTruth;
           private boolean[][] myShow;

           public void cellPicked(int row, int col)
           {          if( inBounds(row, col) && !myShow[row][col] )
                      {          myShow[row][col] = true;

                                 if( myTruth[row][col] == 0)
                                 {          for(int r = -1; r <= 1; r++)
                                                      for(int c = -1; c <= 1; c++)
                                                                 cellPicked(row + r, col + c);
                                 }
                      }
           }

           public boolean inBounds(int row, int col)
           {          return 0 <= row && row < myTruth.length && 0 <= col && col < myTruth[0].length;
           }
}
```

16. GenericListVersion2

```
    import java.util.Collection;
import java.util.Iterator;

/**
 * A class to provide a simple list.
 * List resizes automatically. Used to illustrate
 * various design and implementation details of
 * a class in Java.
 *
 * @author scottm
 *
 */
public class GenericListVersion2 implements Iterable{
   // 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 Object[] iValues;
private int iSize;

private class GenericListIterator implements Iterator{
    private int position;
    private boolean removeOK;

    private GenericListIterator(){
        position = 0;
        removeOK = false;
    }

    public boolean hasNext(){
        return position < iSize;
    }

    public Object next(){
        Object result = iValues[position];
        position++;
        removeOK = true;
        return result;
    }

    public void remove(){
        if( !removeOK )
            throw new IllegalStateException();
        // which element should be removed??
        removeOK = false;
        GenericListVersion2.this.remove(position - 1);
        position--;
    }
}

public Iterator iterator(){
    return new GenericListIterator();
}

public void addAll(Collection c){
   // for each loop
    for(Object obj : c){
        this.add(obj);
    }
}

/**
```

```
 * 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(Object x){
    insert(iSize, x);
}

public Object get(int pos){
    return iValues[pos];
}

/**
 * Insert obj at position pos.
 * post: get(pos) = x, size() = old size() + 1
 * @param pos 0 <= pos <= size()
 * @param obj The element to add.
 */
public void insert(int pos, Object obj){
    ensureCapcity();
    for(int i = iSize; i > pos; i--){
        iValues[i] = iValues[i - 1];
    }
    iValues[pos] = obj;
    iSize++;
}

public Object remove(int pos){
    Object removedValue = iValues[pos];
    for(int i = pos; i < iSize - 1; i++)
        iValues[i] = iValues[i + 1];
    iValues[iSize - 1] = null;
    iSize--;
    return removedValue;
}

private void ensureCapcity(){
    // is there extra capacity available?
    // if not, resize
    if(iSize == iValues.length)
        resize();
}

public int size(){
    return iSize;
}
```

```
// resize internal storage container by a factor of 2
private void resize() {
    Object[] temp = new Object[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].toString() + ", ";
    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 GenericListVersion2(){
    //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 GenericListVersion2(int initialCap) {
    assert initialCap > 0 : "Violation of precondition. IntListVer1(int initialCap):"
        + "initialCap must be greater than 0. Value of initialCap: " + initialCap;
    iValues = new Object[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 is not null and refers to an IntList
      GenericListVersion2 otherList = (GenericListVersion2)other;
      result = this.size() == otherList.size();
      int i = 0;
      while(i < iSize && result){
        result = this.iValues[i].equals( otherList.iValues[i] );
        i++;
      }
    }
```

```
        return result;
    }

}
```

17. GenericListVersion3

```
   package Fall0811;
import java.util.Iterator;

public class GenericList<E> implements Iterable<E>{
 // class constant
    private static final int DEFAULT_CAP = 10;

    // instance variables
    protected E[] container; // the array is NOT the list
    private int listSize;

    public Iterator<E> iterator(){
        return new GenListIterator();
    }

    // inner class
    private class GenListIterator implements Iterator<E>{
        private int indexOfNextElement;
        private boolean okToRemove;

        private GenListIterator(){
            indexOfNextElement = 0;
            okToRemove = false;
        }

        public boolean hasNext(){
            return indexOfNextElement < size();
        }

        public E next(){
            assert hasNext();
            okToRemove = true;
            indexOfNextElement++;
            return container[indexOfNextElement - 1];
        }

        public void remove(){
            assert okToRemove;
            okToRemove = false;
```

```
        indexOfNextElement--;
        GenericList.this.remove(indexOfNextElement);
    }

}

public boolean equals(Object obj){
    assert this != null;
    if(obj == null)
        return false;
    else if (this == obj)
        return true;
    else if( this.getClass() != obj.getClass() )
        return false;
    else{
        // obj is a non null GenericList
        GenericList list = (GenericList)obj;
        if( list.size() != size() )
            return false;
        for(int i = 0; i < size(); i++)
            if( (get(i) == null && list.get(i) != null) || !get(i).equals(list.get(i)) )
                return false;
        return true;
    }


}


// creates an empty IntList
public GenericList(){
    this(DEFAULT_CAP);
}

// pre: initialCap >= 0
public GenericList(int initialCap){
    assert initialCap >= 0 : "failed precondition";
    container = (E[])(new Object[initialCap]);
    listSize = 0;
}

public void insertAll(int pos, GenericList<E> otherList){

    for(int i = 0; i < otherList.listSize; i++){
        this.insert(pos + i, otherList.container[i]);
    }
```

```
}

// pre: 0 <= pos < size()
public E remove(int pos){
    E result = container[pos];
    listSize--;
    for(int index = pos; index < size(); index++){
        container[index] = container[index + 1];
    }
    container[listSize] = null;
    return result;
}

// pre: 0 <= pos <= size()
public void insert(int pos, E element){
    assert 0 <= pos && pos <= size();
    if( size() == container.length )
        resize();
    for(int index = size(); index > pos; index--){
        assert index > 0;
        container[index] = container[index - 1];
    }
    container[pos] = element;
    listSize++;
}

// get size of list
public int size(){
    return listSize;
}

// access elements
// pre: 0 <= position < size()
public E get(int position){
    assert 0 <= position && position < size();
    return container[position];
}

// pre: none
public void add(E element){
    insert(size(), element);
}

private void resize() {
    E[] temp = (E[])(new Object[container.length * 2 + 1]);
    System.arraycopy(container, 0, temp, 0, size());
```

```
        container = temp;
    }

    public String toString(){
        StringBuffer result = new StringBuffer("[");
        final int LIMIT = size() - 1;
        for(int i = 0; i < LIMIT; i++){
            if( this == this.get(i) )
                result.append("this list");
            else{
                result.append(get(i));
            }
            result.append(", ");
        }
        if( size() != 0)
            result.append(get(size() - 1));
        result.append("]");
        return result.toString();
    }

}
```

18. ListNode

```
        /**
 * A class that represents a node to be used in a linked list.
 * These nodes are singly linked.
 *
 * @author Mike Scott
 * @version July 27, 2005
 */

 public class ListNode
 {
             // instance variables

            // the data to store in this node
            private Object myData;

            // the link to the next node (presumably in a list)
            private ListNode myNext;

            /**
             * default constructor
             * pre: none<br>
             * post: getData() = null, getNext() = null
```

```
 */
public ListNode()
{           this(null, null);
}

/**
 * create a ListNode that holds the specified data and refers to the specified next element
 * pre: none<br>
 * post: getData() = item, getNext() = next
 * @param item the  data this ListNode should hold
 * @param next the next node in the list
 */
public ListNode(Object data, ListNode next)
{           myData = data;
            myNext = next;
}


/**
 * return the data in this node
 * pre: none<br>
 * @return the data this ListNode holds
 */
public Object getData()
{           return myData;    }


/**
 * return the ListNode this ListNode refers to
 * pre: none<br>
 * @return the ListNode this ListNode refers to (normally the next one in a list)
 */
public ListNode getNext()
{           return myNext;    }


/**
 * set the data in this node
 * The old data is over written.<br>
 * pre: none<br>
 * @param data the new data for this ListNode to hold
 */
public void setData(Object data)
{           myData = data;   }

/**
```

```
         * set the next node this ListNode refers to
         * pre: none<br>
         * @param next the next node this ListNode should refer to
         */
        public void setNext(ListNode next)
        {           myNext = next;   }
}
```

19. IList

```
  import java.util.Iterator;

/**
 * Interface for a simple List. Random access to all items in the list is provided.
 * The numbering of elements in the list begins at 0.
 *
 */
public interface IList<E> extends Iterable<E>{

        /**
         * Add an item to the end of this list.
         * <br>pre: none
         * <br>post: size() = old size() + 1, get(size() - 1) = item
         * @param item the data to be added to the end of this list
         */
        void add(E item);

        /**
         * Insert an item at a specified position in the list.
         * <br>pre: 0 <= pos <= size()
         * <br>post: size() = old size() + 1, get(pos) = item, all elements in
         * the list with a positon >= pos have a position = old position + 1
         * @param pos the position to insert the data at in the list
         * @param item the data to add to the list
        */
        void insert(int pos, E item);

        /**
         * Change the data at the specified position in the list.
         * the old data at that position is returned.
         * <br>pre: 0 <= pos < size()
         * <br>post: get(pos) = item, return the
         * old get(pos)
```

```
    * @param pos the position in the list to overwrite
             * @param item the new item that will overwrite the old item
             * @return the old data at the specified position
             */
            E set(int pos, E item);

            /**
             * Get an element from the list.
             * <br>pre: 0 <= pos < size()
             * <br>post: return the item at pos
             * @param pos specifies which element to get
             * @return the element at the specified position in the list
             */
            E get(int pos);


            /**
             * Remove an element in the list based on position.
             * <br>pre: 0 <= pos < size()
             * <br>post: size() = old size() - 1, all elements of
             * list with a positon > pos have a position = old position - 1
             * @param pos the position of the element to remove from the list
             * @return the data at position pos
             */
            E remove(int pos);

            /**
             * Remove the first occurrence of obj in this list.
             * Return <tt>true</tt> if this list changed as a result of this call, <tt>false</tt> otherwise.
             * <br>pre: none
             * <br>post: if obj is in this list the first occurence has been removed and size() = old size() -
1.
             * If obj is not present the list is not altered in any way.
             * @param obj The item to remove from this list.
             * @return Return <tt>true</tt> if this list changed as a result of this call, <tt>false</tt>
otherwise.
             */
            boolean remove(E obj);

            /**
             * Return a sublist of elements in this list from <tt>start</tt> inclusive to <tt>stop</tt>
exclusive.
             * This list is not changed as a result of this call.
             * <br>pre: <tt>0 <= start < size(), start <= stop <= size()</tt>
             * <br>post: return a list whose size is stop - start and contains the elements at positions start
through stop - 1 in this list.
```

```
     * @param start index of the first element of the sublist.
     * @param stop stop - 1 is the index of the last element of the sublist.
     * @return a list with <tt>stop - start</tt> elements, The elements are from positions
<tt>start</tt> inclusive to
     * <tt>stop</tt> exclusive in this list.
     */
    IList<E> getSubList(int start, int stop);

    /**
     * Return the size of this list. In other words the number of elements in this list.
     * <br>pre: none
     * <br>post: return the number of items in this list
     * @return the number of items in this list
     */
    int size();

    /**
     * Find the position of an element in the list.
     * <br>pre: none
     * <br>post: return the index of the first element equal to item
     * or -1 if item is not present
     * @param item the element to search for in the list
     * @return return the index of the first element equal to item or a -1 if item is not present
     */
    int indexOf(E item);

    /**
     * find the position of an element in the list starting at a specified position.
     * <br>pre: 0 <= pos < size()
     * <br>post: return the index of the first element equal to item starting at pos
     * or -1 if item is not present from position pos onward
     * @param item the element to search for in the list
     * @param pos the position in the list to start searching from
     * @return starting from the specified position return the index of the first element equal to
item or a -1 if item is not present between pos and the end of the list
     */
    int indexOf(E item, int pos);

    /**
     * return the list to an empty state.
     * <br>pre: none
     * <br>post: size() = 0
     */
    void makeEmpty();

    /**
```

```
            * return an Iterator for this list.
            * <br>pre: none
            * <br>post: return an Iterator object for this List
            */
    Iterator<E> iterator();

    /**
     * Remove all elements in this list from <tt>start</tt> inclusive to <tt>stop</tt> exclusive.
     * <br>pre: <tt>0 <= start < size(), start <= stop <= size()</tt>
     * <br>post: <tt>size() = old size() - (stop - start)</tt>
     * @param start position at beginning of range of elements to be removed
     * @param stop stop - 1 is the position at the end of the range of elements to be removed
     */
    void removeRange(int start, int stop);

    /**
     * Return a String version of this list enclosed in
     * square brackets, []. Elements are in
     * are in order based on position in the
     * list with the first element
     * first. Adjacent elements are seperated by comma's
     * @return a String representation of this IList
     */
    public String toString();

    /**
     * Determine if this IList is equal to other. Two
     * ILists are equal if they contain the same elements
     * in the same order.
     * @return true if this IList is equal to other, false otherwise
     */
    public boolean equals(Object other);
}
```

20. LinkedList

```
   package Fall0811;

import java.util.Iterator;

import Summer08.Node;

public class LinkedList implements Iterable {
    private Node head;
    private Node tail;
    private int size;
```

```
public Iterator iterator(){
    return new LLIterator();
}

private class LLIterator implements Iterator{
    private Node nextNode;
    private boolean removeOK;
    private int posToRemove;

    private LLIterator(){
        nextNode = head;
        removeOK = false;
        posToRemove = -1;
    }

    public boolean hasNext(){
        return nextNode != null;
    }

    public Object next(){
        assert hasNext();

        Object result = nextNode.getData();
        nextNode = nextNode.getNext();

        removeOK = true;
        posToRemove++;

        return result;
    }

    public void remove(){
        assert removeOK;
        removeOK = false;
        LinkedList.this.remove(posToRemove);
        posToRemove--;
    }
}

public void makeEmpty(){
    // let GC do its job!!!!!!!
    head = tail = null;
    size = 0;
}
```

```
public Object remove(int pos){
    assert pos >= 0 && pos < size;
    Object result;
    if( pos == 0 ){
        result = head.getData();
        head = head.getNext();
        if( size == 1 )
            tail = null;
    }
    else{
        Node temp = head;
        for(int i = 1; i < pos; i++)
            temp = temp.getNext();
        result = temp.getNext().getData();
        temp.setNext( temp.getNext().getNext() );
        if( pos == size - 1)
            tail = temp;
    }
    size--;
    return result;
}

public Object get(int pos){
    assert pos >= 0 && pos < size;
    // array based list
    // return myCon[pos]
    Object result;
    if( pos == size - 1 )
        result = tail.getData(); //O(1)
    else{
        Node temp = head;
        for(int i = 0; i < pos; i++)
            temp = temp.getNext();
        result = temp.getData();
        // average case O(N) :((((
    }
    return result;
}

public void insert(int pos, Object obj){
    assert pos >= 0 && pos <= size;

    // addFirst?
    if(pos == 0)
        addFirst(obj); // O(1)
    // add last?
```

```
        else if( pos == size )
            add(obj); //at end O(1)
        else{
            // general case
            Node temp = head;
            for(int i = 1; i < pos; i++)
                temp = temp.getNext();
            // I know temp is pointing at the
            // node at position pos - 1
            Node newNode = new Node(obj, temp.getNext());
            temp.setNext( newNode );
            size++;
        }
    }

    public void add(Object obj){
        Node newNode = new Node(obj, null);
        if( size == 0 )
            head = newNode;
        else
            tail.setNext(newNode);
        tail = newNode;
        size++;
    }

    public void addFirst(Object obj){
        if(size == 0)
            add(obj);
        else{
            Node newNode = new Node(obj, head);
            head = newNode;
            size++;
        }
    }

    public String toString(){
        String result = "";
        Node temp = head;
        for(int i = 0; i < size; i++){
            result += temp.getData() + " ";
            temp = temp.getNext();
        }
        return result;
    }
}
```

21. UnsortedHashSet

```
import java.util.LinkedList;
import java.lang.reflect.Array;

public class UnsortedHashSet<E> {

    private static final double LOAD_FACTOR_LIMIT = 0.7;

    private int size;
    private LinkedList<E>[] con;

    public UnsortedHashSet() {
        con  = (LinkedList<E>[])(new LinkedList[10]);
    }

    public boolean add(E obj) {
        int oldSize = size;
        int index = Math.abs(obj.hashCode()) % con.length;
        if(con[index] == null)
            con[index] = new LinkedList<E>();
        if(!con[index].contains(obj)) {
            con[index].add(obj);
            size++;

        }
        if(1.0 * size / con.length > LOAD_FACTOR_LIMIT)
            resize();
        return oldSize != size;
    }

    private void resize() {
        UnsortedHashSet<E> temp = new UnsortedHashSet<E>();
        temp.con = (LinkedList<E>[])(new LinkedList[con.length * 2 + 1]);
        for(int i = 0; i < con.length; i++){
            if(con[i] != null)
                for(E e : con[i])
                    temp.add(e);
        }
        con = temp.con;
    }
```

```
    public int size() {
        return size;
    }
}
```

22. UnsortedSetTest

```
  package Solution;


import java.io.File;
import java.lang.reflect.Method;
import java.util.Arrays;
import java.util.Collection;
import java.util.HashSet;
import java.util.Scanner;
import java.util.TreeSet;

public class UnsortedSetTest {

    public static void main(String[] args) throws Exception {
        String[] allFileNames = {"hounds.txt", "huckfinn.txt", "oz.txt", "war.txt", "ciaFactBook2008.txt"};
        String[] noCIA = {"hounds.txt", "huckfinn.txt", "oz.txt", "war.txt"};
        countWords(new BinarySearchTree<String>(), allFileNames[0]);
        for(String s : allFileNames) {
            System.out.println(s);
            countWordsOurUnsortedSet(s);
            countWordsOurBinarySearchTree(s);
            countWordsOurHash(s);
            countWordsCollection(new TreeSet<String>(), s);
            int[] result = countWordsCollection(new HashSet<String>(), s);
            System.out.println(result[0] + " total words.");
            System.out.println(result[1] + " distinct words.");
            System.out.println();

        }
    }

    // return total num words, and num distinct words
    public static int[] countWordsCollection(Collection<String> c, String fileName) throws Exception{
        c.clear();
        Scanner fileScanner = new Scanner(new File(fileName));
        Stopwatch st = new Stopwatch();
        st.start();
        int total = 0;
        while(fileScanner.hasNext()){
```

```
            c.add(fileScanner.next());
            total++;
        }
        st.stop();
        System.out.println("Time for " + c.getClass() + " : \n" + st);
//        System.out.println(c.size() + " distinct words");
//        System.out.println(total + " total words including duplicates: ");
        assert total >= c.size();
        System.out.println();
        return new int[]{total, c.size()};
    }


    // GACKY GACKY GACKY repition. Look into removing repetition with reflection
    // we assume there will be add and size methods
    public static int[] countWordsOurHash(String fileName) throws Exception {
        Scanner fileScanner = new Scanner(new File(fileName));
        Stopwatch st = new Stopwatch();
        UnsortedHashSet<String> c = new UnsortedHashSet<String>();
        st.start();
        int total = 0;
        while(fileScanner.hasNext()) {
            c.add(fileScanner.next());
            total++;
        }
        st.stop();
        System.out.println("Time for our hashtable (closed address hashing): \n" + st);
//        System.out.println(c.size() + " distinct words");
//        System.out.println(total + " total words including duplicates: ");
        assert total >= c.size();
        System.out.println();
        return new int[]{total, c.size()};
    }

    public static int[] countWordsOurUnsortedSet(String fileName) throws Exception {
        Scanner fileScanner = new Scanner(new File(fileName));
        Stopwatch st = new Stopwatch();
        UnsortedSet<String> c = new UnsortedSet<String>();
        st.start();
        int total = 0;
        while(fileScanner.hasNext()){
            c.add(fileScanner.next());
            total++;
        }
        st.stop();
        System.out.println("Time for our unsorted set based on ArrayList: \n" + st);
```

```
//        System.out.println(c.size() + " distinct words");
//        System.out.println(total + " total words including duplicates: ");
        assert total >= c.size();
        System.out.println();
        return new int[]{total, c.size()};
    }

    public static int[] countWordsOurBinarySearchTree(String fileName) throws Exception {
        Scanner fileScanner = new Scanner(new File(fileName));
        Stopwatch st = new Stopwatch();
        BinarySearchTree<String> c = new BinarySearchTree<String>();
        st.start();
        int total = 0;
        while(fileScanner.hasNext()){
            c.add(fileScanner.next());
            total++;
        }
        st.stop();
        System.out.println("Time for our binary search tree: \n" + st);
//        System.out.println(c.size() + " distinct words");
//        System.out.println(total + " total words including duplicates: ");
        assert total >= c.size();
        System.out.println();
        return new int[]{total, c.size()};
    }


    // a try at reflection. Not working on Binary Search tree from class.
    // Hunch. Due to add method taking in Comparable, not Object!
    // Alterantives: search list of methods for name?
    public static int[] countWords(Object c, String fileName) throws Exception {
        Scanner fileScanner = new Scanner(new File(fileName));
        Stopwatch st = new Stopwatch();
        System.out.println(Arrays.toString(c.getClass().getMethods()));
        Method addMethod = c.getClass().getMethod("add", Object.class);
        st.start();
        int total = 0;
        while(fileScanner.hasNext()){
            addMethod.invoke(c, fileScanner.next());
            total++;
        }
        st.stop();
        System.out.println("Time for " + c.getClass() + ": "+ st);
        Method sizeMethod = c.getClass().getMethod("size");
        int distictWords = (Integer) sizeMethod.invoke(c);
//        System.out.println(distictWords + " distinct words");
```

```
//        System.out.println(total + " total words including duplicates: ");
        System.out.println();
        return new int[]{total, distictWords};
    }
}
```

23. SimpleWordCount

```
import java.io.File;
import java.io.IOException;
import java.util.Map;
import java.util.Scanner;
import java.util.TreeMap;

public class SimpleWordCounter {

    public static void main(String[] args) {
        try {
            File f = new File("ciaFactBook2008.txt");
            Scanner sc;
            sc = new Scanner(f);
            // sc.useDelimiter("[^a-zA-Z']+");
            Map<String, Integer> wordCount = new TreeMap<String, Integer>();
            while(sc.hasNext()) {
                String word = sc.next();
                if(!wordCount.containsKey(word))
                    wordCount.put(word, 1);
                else
                    wordCount.put(word, wordCount.get(word) + 1);
            }

            // show results
            for(String word : wordCount.keySet())
                System.out.println(word + " " + wordCount.get(word));
            System.out.println(wordCount.size());
        }
        catch(IOException e) {
            System.out.println("Unable to read from file.");
        }
    }
}
```

24. WordCount

```
  import java.io.File;
import java.io.FileNotFoundException;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.Iterator;
import java.util.Map;
import java.util.Scanner;

import javax.swing.JFileChooser;
import javax.swing.UIManager;


public class WordCount {

    public static void main(String[] args) {
        countWordsViaGUI();
    }


    // allow user to pick file to exam via GUI.
    // allow multiple picks
    public static void countWordsViaGUI() {
        setLookAndFeel();
        try {
            Scanner key = new Scanner(System.in);
            do {
                System.out.println("Opening GUI to choose file.");
                Scanner fileScanner = new Scanner(getFile());
                Stopwatch st = new Stopwatch();
                st.start();
                ArrayList<String> words = countWordsWithArrayList(fileScanner);
                st.stop();
                System.out.println("time to count: " + st);
                System.out.print("Enter number of words to display: ");
                int numWordsToShow = Integer.parseInt(key.nextLine());
                showWords(words, numWordsToShow);
                fileScanner.close();
                System.out.print("Perform another count? ");
            } while(key.nextLine().toLowerCase().charAt(0) == 'y');
            key.close();
        }
        catch(FileNotFoundException e) {
            System.out.println("Problem reading the data file. Exiting the program." + e);
        }
    }
```

```
// determine distinct words in a file using an array list
private static ArrayList<String> countWordsWithArrayList(Scanner fileScanner) {


    System.out.println("Total number of words: " + numWords);
    System.out.println("number of distincy words: " + result.size());
    return result;
}


// determine distinct words in a file and frequency of each word with a Map
private static Map<String, Integer> countWordsWithMap(Scanner fileScanner) {


    System.out.println("Total number of words: " + numWords);
    System.out.println("number of distincy words: " + result.size());
    return result;
}


private static void showWords(ArrayList<String> words, int numWordsToShow) {
    for(int i = 0; i < words.size() && i < numWordsToShow; i++)
        System.out.println(words.get(i));
}


private static void showWords(Map<String, Integer> words, int numWordsToShow) {



}


// perform a series of experiments on files. Determine average time to
// count words in files of various sizes
private static void performExp() {
    String[] smallerWorks = {"smallWords.txt", "2BR02B.txt", "Alice.txt", "SherlockHolmes.txt"};;
    String[] bigFile = {"ciaFactBook2008.txt"};
    timingExpWithArrayList(smallerWorks, 50);
    timingExpWithArrayList(bigFile, 3);
    timingExpWithMap(smallerWorks, 50);
    timingExpWithMap(bigFile, 3);
}
```

```
// pre: titles != null, elements of titles refer to files in the
// same path as this program, numExp >= 0
// read words from files and print average time to cound words.
private static void timingExpWithMap(String[] titles, int numExp) {
    try {
        double[] times = new double[titles.length];
        final int NUM_EXP = 50;
        for(int i = 0; i < NUM_EXP; i++) {
            for(int j = 0; j < titles.length; j++) {
                Scanner fileScanner = new Scanner(new File(titles[j]));
                Stopwatch st = new Stopwatch();
                st.start();
                Map<String, Integer> words = countWordsWithMap(fileScanner);
                st.stop();
                System.out.println(words.size());
                times[j] += st.time();
                fileScanner.close();
            }
        }
        for(double a : times)
            System.out.println(a / NUM_EXP);
    }
    catch(FileNotFoundException e) {
        System.out.println("Problem reading the data file. Exiting the program." + e);
    }
}


// pre: titles != null, elements of titles refer to files in the
// same path as this program, numExp >= 0
// read words from files and print average time to cound words.
private static void timingExpWithArrayList(String[] titles, int numExp) {
    try {
        double[] times = new double[titles.length];
        for(int i = 0; i < numExp; i++) {
            for(int j = 0; j < titles.length; j++) {
                Scanner fileScanner = new Scanner(new File(titles[j]));
                Stopwatch st = new Stopwatch();
                st.start();
                ArrayList<String> words = countWordsWithArrayList(fileScanner);
                st.stop();
                times[j] += st.time();
                fileScanner.close();
            }
        }
        for(int i = 0; i < titles.length; i++)
```

```
            System.out.println("Average time for " + titles[i] + ": " + (times[i] / numExp));
        }
        catch(FileNotFoundException e) {
            System.out.println("Problem reading the data file. Exiting the program." + e);
        }
    }


    // try to set look and feel to same as system
    private static void setLookAndFeel() {
        try {
            UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
        }
        catch(Exception e) {
            System.out.println("Unable to set look at feel to local settings. " +
            "Continuing with default Java look and feel.");
        }
    }


    /** Method to choose a file using a window.
     * @return the file chosen by the user. Returns null if no file picked.
     */
    private static File getFile() {
        // create a GUI window to pick the text to evaluate
        JFileChooser chooser = new JFileChooser(".");
        chooser.setDialogTitle("Select File To Count Words:");
        int retval = chooser.showOpenDialog(null);
        File f =null;
        chooser.grabFocus();
        if (retval == JFileChooser.APPROVE_OPTION)
            f = chooser.getSelectedFile();
        return f;
    }
}
```