Brian Bowden —— 10060818
February 15, 2019
CPSC 501 —— T03
Leonard Manzara
Assignment 1: Refactoring

CPSC 501 – Advanced Programming Techniques – Assignment 1

Name: Brian Bowden

Student ID: 10060818

1. **Introduction**

   The goal of this assignment was to refactor an old piece of code to improve the object-oriented design of the code. We were asked to implement five different methods of refactorings to change the structure of the code while leaving the functionality of the code intact. As a secondary and tertiary requirement of this assignment, we were asked to use version control to keep track of the refactors being implemented and use unit testing to ensure functionality has not been changed.

2. **The Code**

   I chose a piece of code I wrote in my second year of computer science. The code itself is small in scope: its purpose is to parse a file containing a "DNA strand" and the user would specify a string (or nucleotide) as input and the program would return the number and location of the instances found in the file.

3. **Pre-refactor**

   Initially looking at the code, I noticed it was prone to bugs and behavior that was improper, so I had to rework the code before I could begin to refactor. The code design was very procedural so I had to change it a bit to make it testable and workable. In the pre-refactor version, there are two classes: DNA.java and KMer.java, which will be referenced in this report. The DNA class has the main() method, handles most of the user input and invokes the KMer class. The KMer class handles the file parsing and the set up for the DNA class to query later.

4. **Refactoring**

   Any code snippets that were necessary are included in the attached appendix (Appendix A). Each refactor will have a code snippet, but this report will refer to each refactor by its version control number. It should be noted that the version control used is git. Format for reading:
   Version number - Refactor Number - Git Reference Number

(a) **Version 1.1 - bb78ce1**
intial commit with several tests
This is the first version of the code after some reworks mentioned above. Also included in this commit were the first of the tests, with minimal unit testing.
Testing was difficult here because of the very procedural design of the code.

(b) **Version 1.2 - Refactor 1 - 5e6a157**
Replace Error code with exception
Code Smell: Error code where an exception should be used
In Class: DNA.java
In Method: public static String inquire(int f)

The "while" loop was removed with the motivation that the caller could handle the problems with the call. The method now throws an exception instead of being in a loop within the method. The motivation behind this is to make the method more modular and more open to being tested.
The method could be further improved upon by throwing a more specific Exception so the caller can handle them in different ways. This required the calling code to be reworked as well to handle the Exceptions being thrown.
This was tested simply by recompiling and running the program again to ensure the functionality was unaffected.

(c) **Version 1.3 - Refactor 2 - c44edb1**
Extract Subclass
Code Smell: Feature Envy and Duplicated Code
In Class: DNA.java and KMer.java

There was some duplicate code in both classes and the motivation behind this refactor was to prepare both classes for a hierarchy.
KMer now extends DNA as a result of this refactor.

(d) **Version 1.4 - Refactor 3 - b5706ec**
Extract Method
Code Smell: Long Method

The main() method in DNA is trying to do a lot. It was easy to extract a simple method from the beginning of main(). The motivation here was to eventually be able to contain some of these extracted methods in a separate class.
New Method: public static String inquireFileName(Scanner scan)

throws Exception
This was again tested by recompiling and running the program.

(e) **Version 1.5 - Refactor 4 - 4f84721**
Pull-up Method
Code Smell: Duplicate code
In Class: KMer.java

Now that KMer extends DNA, it makes sense to deal with the duplicate code, namely the two similar methods in both of the classes. These methods are: "Sequencer()" and "decision()". They are deleted from the extension class.

(f) **Version 1.6 - Refactor 5 - 65fd515**
Extract Method
Code Smell: Long Method
In Class: KMer.java Essentially everything is being done in the constructor of KMer, so the motivation was to isolate the parts of the constructor and keep the bulk of the class outside of the scope of the constructor. Extract Method is used here to make a new method called solve() that handles more of the user involvement. In addition to the new constructor and the new method, many getters and setters were included to make the class more object oriented. It was also necessary to pull some of the local variables into a field as part of this proccess. Also, this required that the caller had to make an additional call to the class to continue the functionality of the program.

(g) **Version 1.7 - Refactor 6 - e8271aa**
Substitute Algorithm / Add Parameter
Code Smell: simply poorly written code
In Class: KMer.java

This method was just so poorly written. So without loss of functionality, this method was rewritten and a parameter was added so that the existing Scanner contained in the method could be removed, and it would be up to the caller to provide a Scanner as a parameter.

(h) **Version 1.8 - Refactor 7 - 9f80c84**
Extract Class
Code Smell: Large Class
In Class: KMer.java

KMer is trying to do too much. Now that the new method solve() exists, it is easier to group together parts of the code that

should be part of its own class. Many of the local variables from solve() and some of the fields from the KMer class are extracted to form a new class. This new class serves more as a data class, but the functionality does not change and the class is more modular. There is a motivation to add more functionality to the new class, but for now it is entirely getters and setters. There had to be some reworking to the existing code to ensure modularity of the new class and functionality, but it is mostly changing calls to variables to calls to methods within the new class.
New Class: KMerData.java

(i) **Version 1.9 - Refactor 8 - 732212f**
Extract Method
Code Smell: Long Method
In Class: KMer.java

Despite extracting a class from KMer and the method solve(), the method still feels too extraneous. A few simple cuts and pastes from the method solve() extracts three more methods, drastically shortening the method and again making this more modular. The three new methods help with set up and teardown of the method solve().
New Methods: initialize(), initialScan(), and tearDown().

5. **Post Refactor**

   Since this code was so poorly written and very procedural, I found it difficult to refactor this. Realistically, it probably should have been rewritten from the ground-up, but I think the end result is much better than what I started with. I'd like to keep refactoring this code, but for the sake of this assignment, it is in a state that I feel comfortable with as a "better product".

6. **Using this code**

   If inclined, a user could run this code in its pre-refactor format and post refactor format. This is how to run it:
   java preRefactor.DNA
   // for the pre-refactor format
   java refactor.DNA
   // for the post refactor format
   The user will be prompted for a file, use any of:
   yeast1.fasta
   yeast1Sample.fasta
   yeast1Test.fasta
   The user will then be prompted for a chain size (string size), I would usually enter 3 for this, but it should handle any integer
   The user is next prompted for a chain to search for, valid characters are [C,T,G,A] or 'q' to quit. The string must be the same length as the prompt before this.
   The program will return the number and location of these instances.

Appendix A
Version 1.2 - Refactor 1 - 5e6a157

```
while(noQuit){
  s = inquire(size);
  start = System.currentTimeMillis();
  query = s.toCharArray();
  if (s.contains("q")){
    System.out.println("Program will terminate...");
    noQuit = false;
  }
  else{
      System.out.println("you entered the sequence: " + s);
```

```
public static String inquire(int f){

  String ask = "Press q to exit, else, please enter nucleotide you are searching for, of length " + f + " : ";
  int i;
  boolean error = true;
  String inp = null;
  char[] c = new char[f];
  while (error){
```

```
        System.out.println(ask);
        inp = sc.nextLine();
        c = inp.toCharArray();
        error = false;
        if (!inp.contains("q")){
            if (inp.length() != f){
            System.out.println("Length of string incorrect, please try again");
              error = true;
          }
          else{
            for (i = 0; i < f; i++){
              if ((c[i] != 'A')&&(c[i] != 'C')&&(c[i] != 'G')&&(c[i] != 'T')){
                System.out.println("You entered an invalid character: " + c[i] + " please re-enter string");
                error = true;
              }
            }
          }
        }
      }
      return inp;
    }

    //replaced with

      while(noQuit){
        while(true) {
```

```
      try{
        s = inquire(size);
        break;
      } catch (Exception e) {
        continue;
      }
    }
    start = System.currentTimeMillis();
    if (s.contains("q")){
      System.out.println("Program will terminate...");
      noQuit = false;
    }
    else{
      System.out.println("you entered the sequence: " + s);


public static String inquire(char[] c, int f) throws Exception {

  String ask = "Press q to exit, else, please enter nucleotide you are searching for, of length " + f + " : ";
  int i;
  String inp = null;
  System.out.println(ask);
  inp = sc.nextLine();
  c = inp.toCharArray();
    if (!inp.contains("q")){
```

```
        if (inp.length() != f){
          System.out.println("Length of string incorrect, please try again");
          throw new Exception();
        }
        else{
          for (i = 0; i < f; i++){
            if ((c[i] != 'A')&&(c[i] != 'C')&&(c[i] != 'G')&&(c[i] != 'T')){
              System.out.println("You entered an invalid character: " + c[i] + " please re-enter string");
              throw new Exception();
            }
          }
        }
      }
      return inp;
  }
```

Version 1.3 - Refactor 2 - c44edb1

```
public class KMer{

//refactored

public class KMer extends DNA{
```

Version 1.4 - Refactor 3 - b5706ec

```
    System.out.println("Please enter a filename: ");
    try{
      fileName = sc.nextLine();
    }
    catch(Exception e){
      System.out.println("Usage: Main <filename>" );
      System.exit(1);
    }

//refactored to

    try {
      fileName = inquireFileName(sc);
    } catch (Exception e) {
      System.exit(1);
    }

// calling the method:

  public static String inquireFileName(Scanner scan) throws Exception{
    String filename;
    System.out.println("Please enter a filename: ");
```

```
    try{
      filename = scan.nextLine();
      return filename;
    }
    catch(Exception e){
      System.out.println("Usage: Main <filename>" );
      throw new Exception();
    }

  }
```

Version 1.5 - Refactor 4 - 4f84721

```
public class KMer extends DNA{

  //other fields and methods
  ...


  public static int Sequencer(int[] n, int s){

    int j = 0;
    int r = 0;
    for (int i = 0; i < s; i++){
      r = (int) Math.pow(10, (s-i) - 1);
      j += n[i] * r;
    }
    return j;
  }


  public static int decision(char c){
    int decider = 0;
    if (c == 'A')
      decider = 1;
    else if (c == 'G')
```

```
      decider = 2;
    else if (c == 'C')
      decider = 3;
    else if (c == 'T')
      decider = 4;
    return decider;
  }

}
```

Version 1.6 - Refactor 5 - 65fd515

```
//The KMer class before refactor

  private int size = 0;
  private int count;
  public int hashsize;
  public int[] nucleo;
  public int key;
  public ArrayDeque<Character> lump = new ArrayDeque<Character>();
  public HashMap<Integer, ArrayList<Integer>> DNA = new HashMap<Integer, ArrayList<Integer>>();
  public int position = 0;

  public KMer (BufferedReader fIn){
    //parses the fasta file

    int asksize = 0;
    int sequence = 0;
    char c;
    String ch;
    char[] stray; // KMer chain in char array
    int j = 0;
    ArrayList<Integer> buck;
    long start;
    long end;
```

```
inquireSize();

stray = new char[size];
nucleo = new int[size];
hashsize = (int) Math.pow(5, size);

Scanner two = new Scanner(fIn);  //delimiter may not be needed
start = System.currentTimeMillis();
String s = two.nextLine(); // gets the header of the fasta file

System.out.println(s);
ch = two.nextLine();

while (two.hasNextLine()){
  ch += two.nextLine();  //The bulk of the file
}

count = ch.length();
System.out.println("count is: " + count);
stray = ch.toCharArray();
buck = new ArrayList<Integer>(); // buckets of the hashmap

while (j < size){            //takes care of initial size queue
  lump.offer(stray[j]);
  nucleo[j] = decision(stray[j]); // the nucleotide being searched for
```

```
  j++;
}

buck.add(position);
sequence = Sequencer(nucleo, size);
key = sequence % hashsize;

DNA.put(key, buck);

lump.poll();
lump.offer(stray[position + size]);

position++;

while (position + size < count){
  asksize = 0;
  while (asksize < size){
    c = lump.poll();
    nucleo[asksize] = decision(c);
    lump.offer(c);
    asksize++;
  }
  sequence = Sequencer(nucleo, size);

  lump.offer(stray[position + size]);
  key = sequence % hashsize;
```

```
      if (DNA.containsKey(key)){      //checks the Map before colliding
        DNA.get(key).add(position);
      }
      else{
        buck = new ArrayList<Integer>();
        buck.add(position);
        DNA.put(key, buck);
      }
      position++;
    }

    end  = System.currentTimeMillis() - start;
    System.out.println("time to parse file = " + end + '\n');
    two.close();
  }

// Refactored to

  private int size;
  private int count;

  public int hashsize;
  public int[] nucleo;
  public int key;

  public ArrayDeque<Character> lump;
```

```
public HashMap<Integer, ArrayList<Integer>> DNA;
public int position = 0;

private Scanner scan;


public KMer (BufferedReader fIn){
  setSize(0);
  setHashSize(0);
  setDNA(new HashMap<Integer, ArrayList<Integer>>());
  setLump(new ArrayDeque<Character>());
  setScan(new Scanner(fIn));

}

public void solve(){

  //parses the fasta file

  int asksize = 0;
  int sequence = 0;
  char c;
  String ch;
  char[] stray; // KMer chain in char array
  int j = 0;
  ArrayList<Integer> buck;
```

```
long start;
long end;

inquireSize();

stray = new char[size];
nucleo = new int[size];
hashsize = (int) Math.pow(5, size);

start = System.currentTimeMillis();
String s = scan.nextLine(); // gets the header of the fasta file

System.out.println(s);
ch = scan.nextLine();

while (scan.hasNextLine()){
  ch += scan.nextLine();  //The bulk of the file
}

count = ch.length();
System.out.println("count is: " + count);
stray = ch.toCharArray();
buck = new ArrayList<Integer>(); // buckets of the hashmap

while (j < size){            //takes care of initial size queue
  lump.offer(stray[j]);
```

```
  nucleo[j] = decision(stray[j]); // the nucleotide being searched for
  j++;
}

buck.add(position);
sequence = Sequencer(nucleo, size);
key = sequence % hashsize;

DNA.put(key, buck);

lump.poll();
lump.offer(stray[position + size]);

position++;

while (position + size < count){
  asksize = 0;
  while (asksize < size){
    c = lump.poll();
    nucleo[asksize] = decision(c);
    lump.offer(c);
    asksize++;
  }
  sequence = Sequencer(nucleo, size);

  lump.offer(stray[position + size]);
```

```
        key = sequence % hashsize;
        if (DNA.containsKey(key)){      //checks the Map before colliding
          DNA.get(key).add(position);
        }
        else{
          buck = new ArrayList<Integer>();
          buck.add(position);
          DNA.put(key, buck);
        }
        position++;
      }

      end  = System.currentTimeMillis() - start;
      System.out.println("time to parse file = " + end + '\n');
      scan.close();
    }

// and added setters for the constructors
// minor refactor in here is a pulled up field (private Scanner scan instead of the local variable Scanner scan)
// This will require an extra line of code in DNA.java


// new setters in KMer.java:
  private void setHashSize(int i) {
    this.hashsize = i;
  }
```

```
  public void setScan(Scanner scan) {
    this.scan = scan.useDelimiter("\n");
  }

  public void setDNA(HashMap<Integer, ArrayList<Integer>> dNA) {
    DNA = dNA;
  }

  private void setLump(ArrayDeque<Character> arrayDeque) {
    this.lump = arrayDeque;

  }
```



```
// extra line in DNA.java

    chain.solve();
```

Version 1.7 - Refactor 6 - e8271aa

```
  public void inquireSize(){
    @SuppressWarnings("resource")
    Scanner help = new Scanner(System.in);
    String size = "Please enter the length of the nucleotide string: ";
    System.out.println(size);
    this.size = help.nextInt();
  }

// After Refactor

  public void inquireSize(Scanner input){
    System.out.println("Please enter the length of the nucleotide string: ");
    setSize(input.nextInt());
  }

// added this in DNA.java

    chain.inquireSize(sc);
    sc.nextLine(); // Scanner needs to eat a line
```

Version 1.8 - Refactor 7 - 9f80c84

```
/ before

  private int size;
  private int count;

  public int hashsize;
  public int[] nucleo;
  public int key;

  public ArrayDeque<Character> lump;
  public HashMap<Integer, ArrayList<Integer>> DNA;
  public int position = 0;

  private Scanner scan;


  public KMer (BufferedReader fIn){
    setSize(0);
    setHashSize(0);
    setDNA(new HashMap<Integer, ArrayList<Integer>>());
    setLump(new ArrayDeque<Character>());
    setScan(new Scanner(fIn));
```

```
}

public void solve(){

  //parses the fasta file

  int asksize = 0;
  int sequence = 0;
  char c;
  String ch;
  char[] stray; // KMer chain in char array
  int j = 0;
  ArrayList<Integer> buck;
  long start;
  long end;

  stray = new char[size];
  nucleo = new int[size];
  hashsize = (int) Math.pow(5, size);

  start = System.currentTimeMillis();
  String s = scan.nextLine(); // gets the header of the fasta file

  System.out.println(s);
  ch = scan.nextLine();
```

```
while (scan.hasNextLine()){
  ch += scan.nextLine();  //The bulk of the file
}

count = ch.length();
System.out.println("count is: " + count);
stray = ch.toCharArray();
buck = new ArrayList<Integer>(); // buckets of the hashmap

while (j < size){            //takes care of initial size queue
  lump.offer(stray[j]);
  nucleo[j] = decision(stray[j]); // the nucleotide being searched for
  j++;
}

buck.add(position);
sequence = Sequencer(nucleo, size);
key = sequence % hashsize;

DNA.put(key, buck);

lump.poll();
lump.offer(stray[position + size]);

position++;
```

```
while (position + size < count){
  asksize = 0;
  while (asksize < size){
    c = lump.poll();
    nucleo[asksize] = decision(c);
    lump.offer(c);
    asksize++;
  }
  sequence = Sequencer(nucleo, size);

  lump.offer(stray[position + size]);
  key = sequence % hashsize;
  if (DNA.containsKey(key)){      //checks the Map before colliding
    DNA.get(key).add(position);
  }
  else{
    buck = new ArrayList<Integer>();
    buck.add(position);
    DNA.put(key, buck);
  }
  position++;
}

end  = System.currentTimeMillis() - start;
System.out.println("time to parse file = " + end + '\n');
scan.close();
```

```
  }

// After Refactor

public class KMer extends DNA{

  private int size;
  private int count;

  public int hashsize;
  public int key;

  public ArrayDeque<Character> lump;
  public HashMap<Integer, ArrayList<Integer>> DNA;
  public int position = 0;

  private Scanner scan;
  private KMerData data;

  private long start;
  private long end;

  private String string;


  public KMer (BufferedReader fIn){
```

```
    setSize(0);
    setHashSize(0);
    setDNA(new HashMap<Integer, ArrayList<Integer>>());
    setLump(new ArrayDeque<Character>());
    setScan(new Scanner(fIn));

  }

  public void solve(){

    //parses the fasta file

    int asksize = 0;
    int sequence = 0;
    char c;
    int j = 0;

    data = new KMerData(size);

    hashsize = (int) Math.pow(5, size);

    System.out.println(hashsize);

    start = System.currentTimeMillis();
    String s = scan.nextLine(); // gets the header of the fasta file
```

```
System.out.println(s);
string = scan.nextLine();

while (scan.hasNextLine()){
  string += scan.nextLine();  //The bulk of the file
}

count = string.length();
System.out.println("count is: " + count);
data.setStray(string.toCharArray());


while (j < size){            //takes care of initial size queue
  lump.offer(data.getStrayElement(j));
  data.setNucleoElement(j, decision(data.getStrayElement(j)));
  j++;
}


data.getBucket().add(position);
sequence = Sequencer(data.getNucleo(), size);
key = sequence % hashsize;

getDNA().put(key, data.getBucket());

lump.poll();
```

```
lump.offer(data.getStrayElement(position + size));

position++;

while (position + size < count){
  asksize = 0;
  while (asksize < size){
    c = lump.poll();
    data.setNucleoElement(asksize, decision(c));
    lump.offer(c);
    asksize++;
  }
  sequence = Sequencer(data.getNucleo(), size);

  lump.offer(data.getStrayElement(position + size));
  key = sequence % hashsize;
  if (DNA.containsKey(key)){      //checks the Map before colliding
    DNA.get(key).add(position);
  }
  else{
    data.setBucket();
    data.getBucket().add(position);
    getDNA().put(key, data.getBucket());
  }
  position++;
}
```

```
    end  = System.currentTimeMillis() - start;
    System.out.println("time to parse file = " + end + '\n');
    scan.close();
  }
  . . . // more methods
}

// and the new class
import java.util.ArrayList;

public class KMerData {
  private ArrayList<Integer> bucket;
  private String string;
  private char c;
  private char[] stray;
  private int[] nucleo;


  public KMerData (int size){
    setStray(new char[size]);
    setNucleo(new int[size]);
    setBucket();
  }

  public ArrayList<Integer> getBucket() {
```

```
    return bucket;
  }

  public void setBucket(){
    this.bucket = new ArrayList<Integer>();
  }


  public String getString() {
    return string;
  }


  public void setString(String string) {
    this.string = string;
  }


  public char getC() {
    return c;
  }


  public void setC(char c) {
    this.c = c;
  }
```

```
public char[] getStray() {
  return stray;
}

public char getStrayElement(int index){
  return stray[index];
}


public void setStray(char[] stray) {
  this.stray = stray;
}


public int[] getNucleo() {
  return nucleo;
}


public void setNucleo(int[] nucleo) {
  this.nucleo = nucleo;
}

public void setNucleoElement(int index, int value){
```

```
    nucleo[index] = value;
  }


}
```

Version 1.9 - Refactor 8 - 732212f

```
// before

public void solve(){

    //parses the fasta file

    int asksize = 0;
    int sequence = 0;
    char c;
    int j = 0;

    data = new KMerData(size);

    hashsize = (int) Math.pow(5, size);

    System.out.println(hashsize);

    start = System.currentTimeMillis();

    // split into methods here

    String s = scan.nextLine(); // gets the header of the fasta file
```

```
    System.out.println(s);
    string = scan.nextLine();

    while (scan.hasNextLine()){
      string += scan.nextLine();  //The bulk of the file
    }

    count = string.length();
    System.out.println("count is: " + count);
    data.setStray(string.toCharArray());

    . . .

    end  = System.currentTimeMillis() - start;
    System.out.println("time to parse file = " + end + '\n');
    scan.close();

  }

// After

  {
  . . .
    initialize();
    initialScan();
```

```
  . . .

  tearDown();
}

  public void inquireSize(Scanner input){
  System.out.println("Please enter the length of the nucleotide string: ");
  setSize(input.nextInt());
}

private void initialize(){
  data = new KMerData(size);

  hashsize = (int) Math.pow(5, size);

  System.out.println(hashsize);
  start = System.currentTimeMillis();
}

private void initialScan(){
  string = scan.nextLine();

  System.out.println(string);
  string = scan.nextLine();

  while (scan.hasNextLine()){
```

```
      string += scan.nextLine();    //takes care of newline characters?
    }

    count = string.length();
    System.out.println("count is: " + count);
    data.setStray(string.toCharArray());
  }

  private void tearDown(){
    end  = System.currentTimeMillis() - start;
    System.out.println("time to parse file = " + end + '\n');
    scan.close();
  }
```

Appendix B: Final Test Cases DNATest.java

```
package refactor.test;

import static org.junit.Assert.*;

import java.io.BufferedReader;

import org.junit.After;
import org.junit.AfterClass;
import org.junit.Before;
import org.junit.BeforeClass;
import org.junit.Test;

import preRefactor.DNA;

public class DNATest {

  private char good;
  private char bad;
  private int[] goodInput;
  private int[] badInput;
  private int goodOut;
  private int badOut;
  private String validFileName;
```

```
private String invalidFileName;
private BufferedReader fIn;

@BeforeClass
public static void setUpBeforeClass() throws Exception {
}

@AfterClass
public static void tearDownAfterClass() throws Exception {
}

@Before
public void setUp() throws Exception {
  good = 'G';
  bad = 'B';
  goodInput = new int[] {3,4,5};
  badInput = new int[] {0,0,1};
  goodOut = 345;
  badOut = 001;
  validFileName = new String("yeast1.fasta");
  invalidFileName = new String("invalid.file");
  fIn = null;
}


@After
```

```
public void tearDown() throws Exception {
}

@Test
public void testValidFileName() {

  try {
    fIn = DNA.opening(validFileName);
  } catch (Exception e) {
    fail();
  }

  assertFalse(fIn == null);

}

@Test
public void testInvalidFileName() {

  try {
    fIn = DNA.opening(invalidFileName);
  } catch (Exception e) {
    assertTrue(true);
  }
}
```

```
@Test
public void testGoodInput() {
  int Result;
  Result = DNA.Sequencer(goodInput, goodInput.length);
  assertEquals(goodOut, Result);
}

@Test
public void testBadInput() {
  int Result;
  Result = DNA.Sequencer(badInput, badInput.length);
  assertEquals(badOut, Result);
}


@Test
public void testGoodCharacter() {
  int result;
  int expected = 2;

  result = DNA.decision(good);
  assertEquals(expected, result);
}

@Test
public void testBadCharacter() {
```

```
    int result;
    int expected = 0;

    result = DNA.decision(bad);
    assertEquals(expected, result);
  }

}
```