EECS 4313: Assignment 4

Monday, 4 April 16

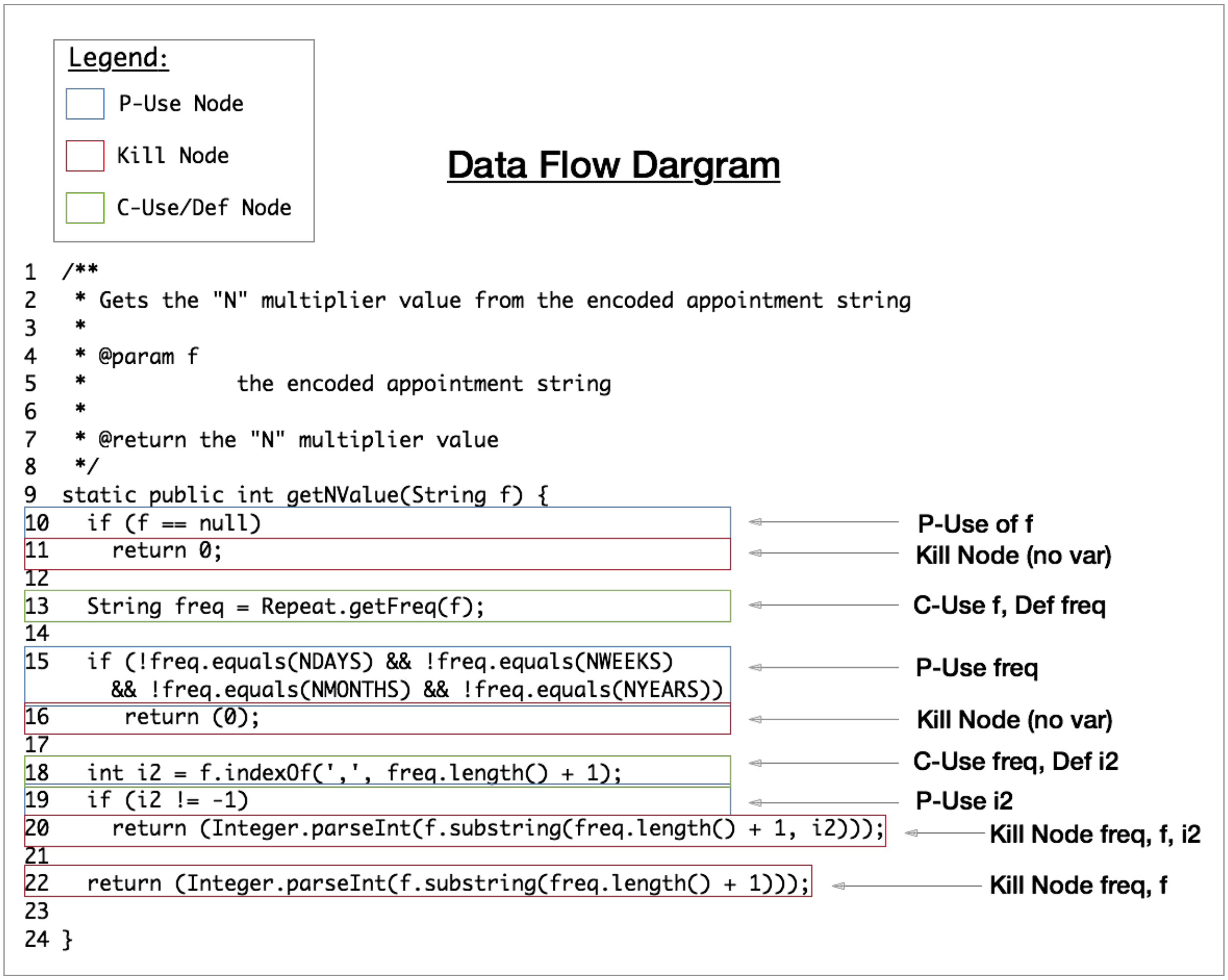
Team Members

```
- Drew Noel (212513784)
- Skyler Layne (212166906)
- Siraj Rauff (212592192)
```

Data Flow

Data Flow Diagram

The Data Flow Diagram below shows the segments labelled by the type of node.

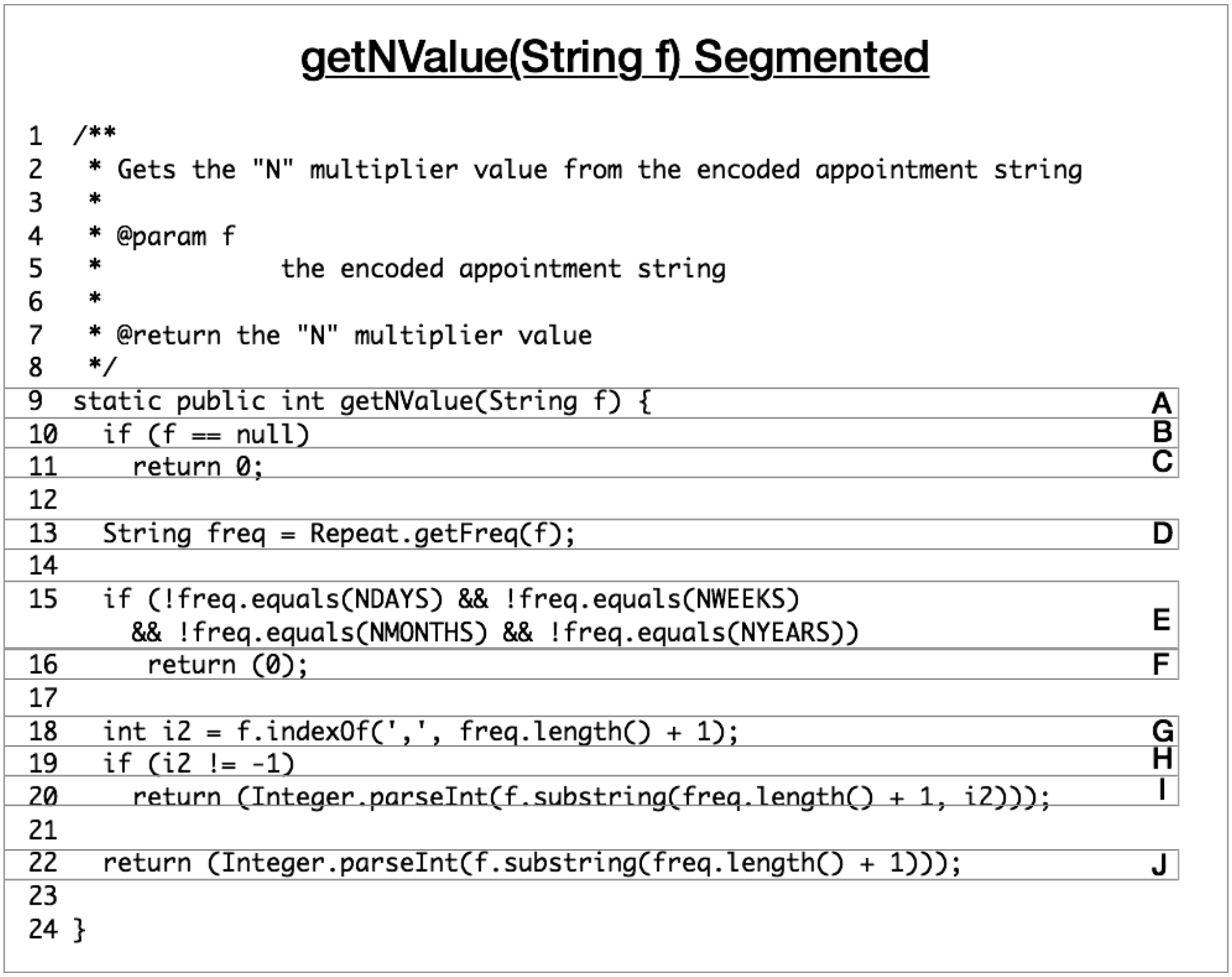


```
/**
 * Gets the "N" multiplier value from the encoded appointment string
 *
 * @param f
 *            the encoded appointment string
 *
 * @return the "N" multiplier value
 */
static public int getNValue(String f) {
  if (f == null)
    return 0;

  String freq = Repeat.getFreq(f);

  if (!freq.equals(NDAYS) && !freq.equals(NWEEKS)
    && !freq.equals(NMONTHS) && !freq.equals(NYEARS))
     return (0);

  int i2 = f.indexOf(',', freq.length() + 1);
  if (i2 != -1)
    return (Integer.parseInt(f.substring(freq.length() + 1, i2)));

  return (Integer.parseInt(f.substring(freq.length() + 1)));

}
```

Figure 1: getNValue(String f) Data Flow Diagram

Program Segmented

The same program has been broken up into corresponding segments basked on the data flow diagram above.



getNValue(String f) Segmented

```
/**
 * Gets the "N" multiplier value from the encoded appointment string
 *
 * @param f
 *            the encoded appointment string
 *
 * @return the "N" multiplier value
 */
static public int getNValue(String f) {
  if (f == null)
    return 0;

  String freq = Repeat.getFreq(f);

  if (!freq.equals(NDAYS) && !freq.equals(NWEEKS)
    && !freq.equals(NMONTHS) && !freq.equals(NYEARS))
     return (0);

  int i2 = f.indexOf(',', freq.length() + 1);
  if (i2 != -1)
    return (Integer.parseInt(f.substring(freq.length() + 1, i2)));

  return (Integer.parseInt(f.substring(freq.length() + 1)));

}
```

Figure 2: getNValue(String f) Segmented

Program Graph

The following figure shows the program graph for the segmented code segment above.

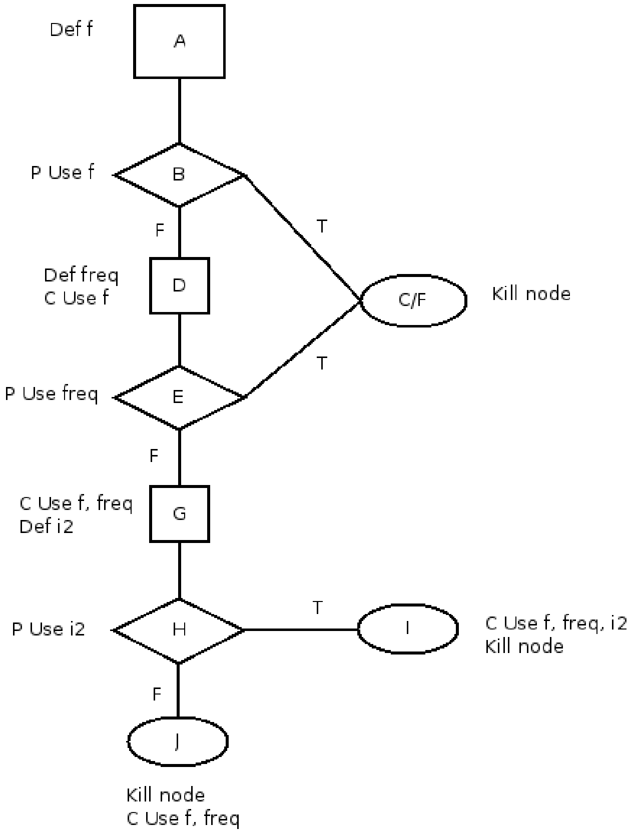


Figure 3: getNValue(String f) Graph

Data Flow Paths

Below we define all the data flow paths. The labelling scheme used here follows from the program graph defined above.

All-Defs

The following graphs satisfy the All definitions criteria:

`Each definition of each variable for at least one use of the definition:`

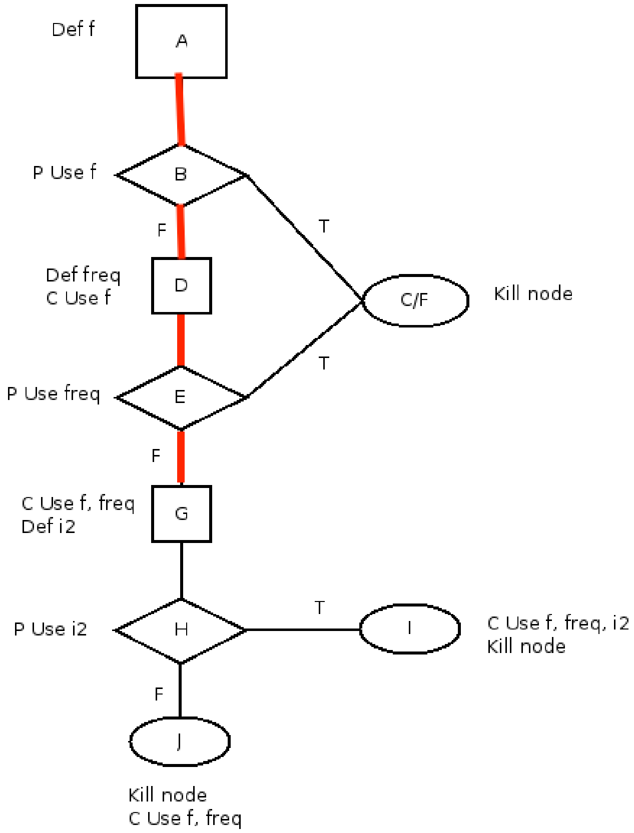


Figure 4: All Definitions (ABDEG)

All-Uses

The following graphs satisfy the All uses criteria:

`At least one path of each variable to each c-use of the definition:`

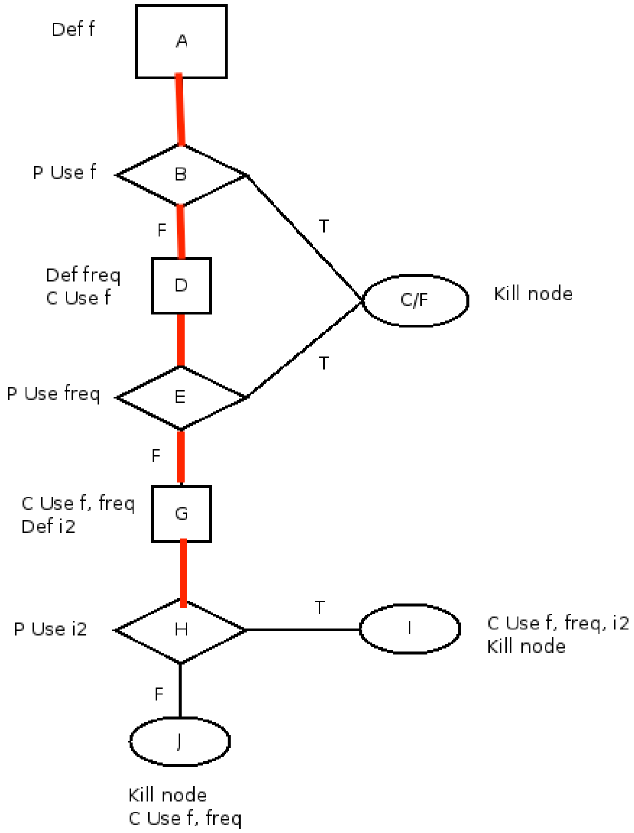


Figure 5: All Uses (ABDEGH)

All P Uses and Some C Uses

The following graphs show the path for All P uses and Some C uses for the given method. The criteria is:

```
At least one path of each variable definition to each p-use of the variable.
If any variable definitions are not covered by p-use, then use c-use:
```

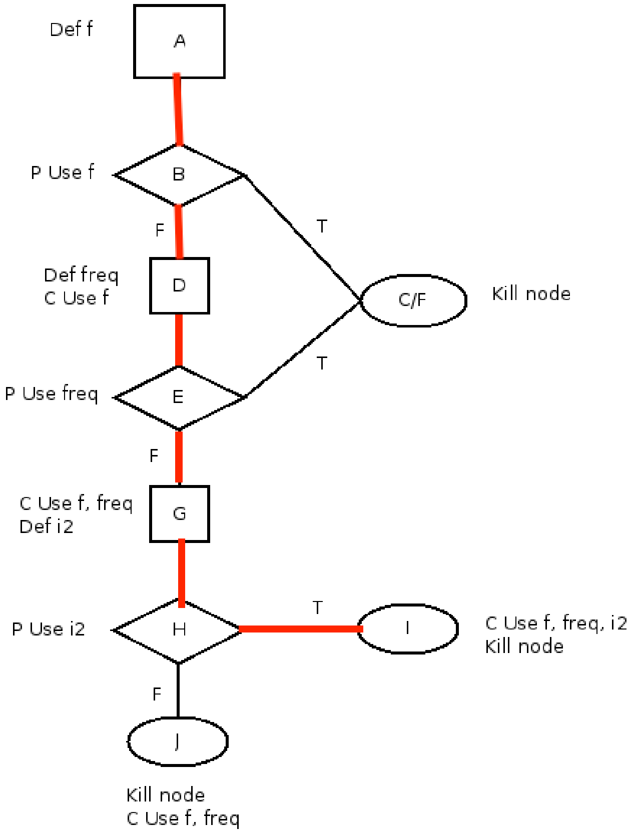


Figure 6: All P uses, Some C uses (ABDEGHI)

All C Uses and Some P Uses

The following diagrams show the paths which satisfy the criteria for All-C-Uses and Some-P-Uses:

```
At least one path of each variable definition to each c-use of the variable.
If any variable definitions are not covered, use p-use:
```

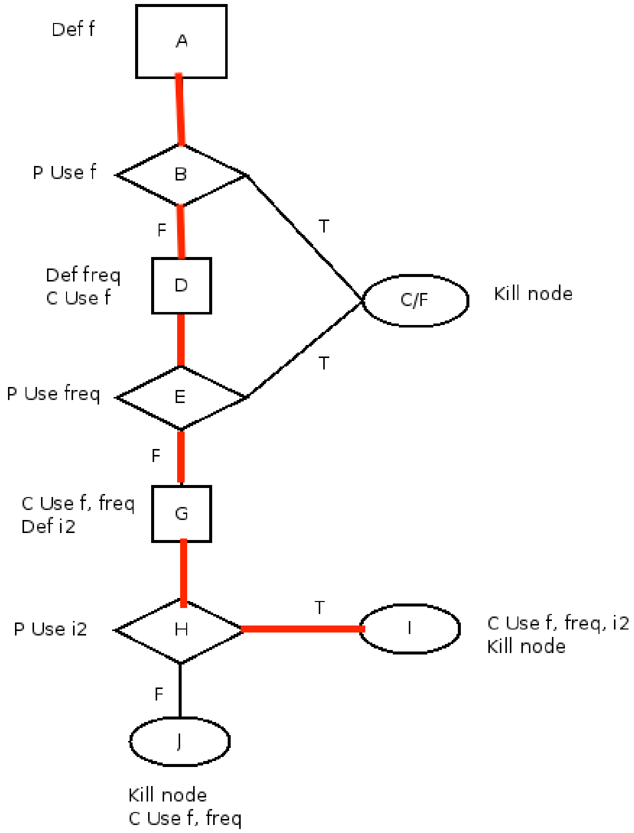


Figure 7: All C uses, Some P uses (ABDEGHI)

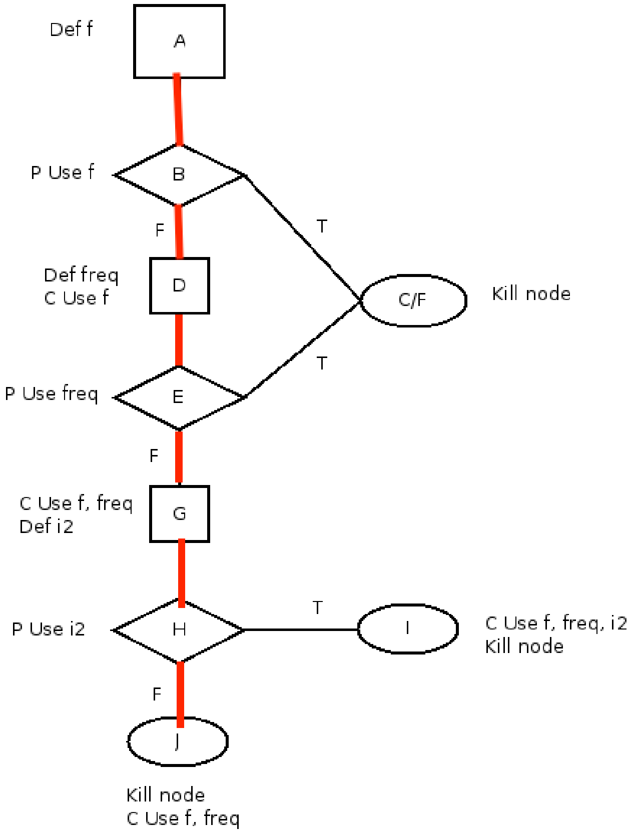


Figure 8: All C uses, Some P uses (ABDEGHJ)

Slices

Below we see the method under investigation, in this case `getNValue(String f)` from the `Repeat.java` class.

```
/**
 * Gets the "N" multiplier value from the encoded appointment string
 *
 * @param f
 *           the encoded appointment string
 *
 * @return the "N" multiplier value
 */
static public int getNValue(String f) {
  if (f == null)
    return 0;

  String freq = Repeat.getFreq(f);

  if (!freq.equals(NDAYS) && !freq.equals(NWEEKS) && !freq.equals(NMONTHS) && !freq.equals(NYEARS))
    return (0);

  int i2 = f.indexOf(',', freq.length() + 1);
  if (i2 != -1)
    return (Integer.parseInt(f.substring(freq.length() + 1, i2)));

  return (Integer.parseInt(f.substring(freq.length() + 1)));

}
```

NOTE: All line numbers are given relative to the method declaration, where the declaration itself is considered `0`. in this case the first statement is then `1`, as shown:

```
1  if (f == null)
2    return 0;
3
4  String freq = Repeat.getFreq(f);
5
6  if (!freq.equals(NDAYS) && !freq.equals(NWEEKS) && !freq.equals(NMONTHS) && !freq.equals(NYEARS))
7    return (0);
8
9  int i2 = f.indexOf(',', freq.length() + 1);
10 if (i2 != -1)
11   return (Integer.parseInt(f.substring(freq.length() + 1, i2)));
12
13 return (Integer.parseInt(f.substring(freq.length() + 1)));
```

Static Slicing 1: Forward Slicing

Forward slices of the form S(v,n) consist of all statements that are affected by the variable `v`, and statement n. In this form forward slices are only immediately useful in static slicing for A-defs, for every n at or after the definition (the slices are empty before the definition).

S(`freq`, 4)

```
String freq = Repeat.getFreq(f);

if (!freq.equals(NDAYS) && !freq.equals(NWEEKS) && !freq.equals(NMONTHS) && !freq.equals(NYEARS))
  return (0);

int i2 = f.indexOf(',', freq.length() + 1);
if (i2 != -1)
  return (Integer.parseInt(f.substring(freq.length() + 1, i2)));

return (Integer.parseInt(f.substring(freq.length() + 1)));
```

S(i2, 9)

```
int i2 = f.indexOf(',', freq.length() + 1);
if (i2 != -1)
  return (Integer.parseInt(f.substring(freq.length() + 1, i2)));

return (Integer.parseInt(f.substring(freq.length() + 1)));
```

Static Slicing 2: Backward Slicing

Backward slices of the form S(v,n) consist of all statements that affect the value of the variable `v` at statement n.

S(freq, 3)

This, along with any value of such that `n < 4`, results in an empty slice as `freq` is not defined.

S(freq, 4)

Any value of n such that `n >= 4`, including the P-use of `freq` at 6, results in the following slice:

```
if (f == null)
  return 0;

String freq = Repeat.getFreq(f);
```

Note that if `f` is null, `freq` is never defined, so it is included in this slice.

S(i2, 8)

This, along with any value of such that `n < 9`, results in an empty slice as `i2` is not defined.

S (i2, 9)

Any value of n such that `n >= 9`, including the P-use of `i2` at 10, results in the following slice:

```
if (f == null)
  return 0;

String freq = Repeat.getFreq(f);

if (!freq.equals(NDAYS) && !freq.equals(NWEEKS) && !freq.equals(NMONTHS) && !freq.equals(NYEARS))
  return (0);
```

```
int i2 = f.indexOf(',', freq.length() + 1);
```

As with S(`freq`, 4), we include the lines prior to i2 as if either of the preceding `if` statements evaluate to `true`, `i2` is never defined.

Dynamic Slicing

Dynamically slicing on f, we decide on the following inputs:

- `f` is `null`
- `f` is not `null` AND `freq` is not one of `NDAYS`, `NWEEKS`, `NMONTHS` or `NYEARS`
- `f` is not `null` AND `freq` is one of `NDAYS`, `NWEEKS`, `NMONTHS` or `NYEARS` and `f` contains a comma
- `f` is not `null` AND `freq` is one of `NDAYS`, `NWEEKS`, `NMONTHS` or `NYEARS` and `f` does not contain a comma

NOTE: We will not be considering here the cases when a variable is not defined prior to a specific line. These are the same regardless of dynamic or static slicing, and can be referenced from the previous section. We will also be slicing for each of these until the end for every variable, though the reader may determine the slice up to line n by eliminating any lines defined after n.

`f` == `null`

```
return 0;
```

We do not define this slice by any particular variable or code segment, as this input will cause the method to immediately return after the singular P-use of `f` that causes the return. This slice is then common to any of the variables `f`, `freq` or `i2` at any given n.

S(`freq`, 13), `f` is not `null` and is such that `Repeat.getFreq(f)` is not one of `NDAYS`, `NWEEKS`, `NMONTHS` or `NYEARS`

```
String freq = Repeat.getFreq(f);

return (0);
```

S(`i2`, 13), `f` is not `null` and is such that `Repeat.getFreq(f)` is not one of `NDAYS`, `NWEEKS`, `NMONTHS` or `NYEARS`

```
return (0);
```

S(`f`, 13), `f` is not `null` and is such that `Repeat.getFreq(f)` is not one of `NDAYS`, `NWEEKS`, `NMONTHS` or `NYEARS`

```
return (0);
```

S(`f`, 13), `f` is not `null` and is such that `Repeat.getFreq(f)` is one of `NDAYS`, `NWEEKS`, `NMONTHS` or `NYEARS` and `f` does contain a comma

```
String freq = Repeat.getFreq(f);

int i2 = f.indexOf(',', freq.length() + 1);

return (Integer.parseInt(f.substring(freq.length() + 1, i2)));
```

Note this is equivalent for S(`freq`, 13) and S(`i2`, 13).

S(f, 13), f is not null and is such that Repeat.getFreq(f) is one of NDAYS, NWEEKS, NMONTHS or NYEARS and f does not contain a comma

```
String freq = Repeat.getFreq(f);

int i2 = f.indexOf(',', freq.length() + 1);

return (Integer.parseInt(f.substring(freq.length() + 1)));
```

Note this is equivalent for S(freq, 13) and S(i2, 13).

Code Coverage

Our updated tests for the previous assignment (a3) cover 100% of all slices, and as such no further tests were added.

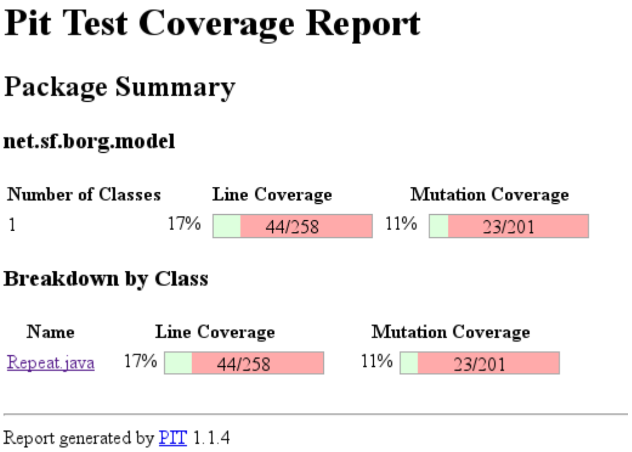
Mutation Testing

Assignment 2 Review

The mutation testing was limited to only tests written by this group, with all other tests forcibly removed. The testing was next limited to mutate only the class which contained the method being tested. As discussed in Assignment 2, some tests uncovered potential bugs in the BORGCalendar. PIT requires that all tests pass in order to run, so these tests were temporarily removed.

Repeat.java Test Analysis

Two of the three methods tested belong to net.sf.borg.model.Repeat. The following is the overview of the class using only the tests submitted for Assignment 2:



Pit Test Coverage Report

Package Summary

net.sf.borg.model

Number of Classes	Line Coverage		Mutation Coverage	
1	17%	44/258	11%	23/201

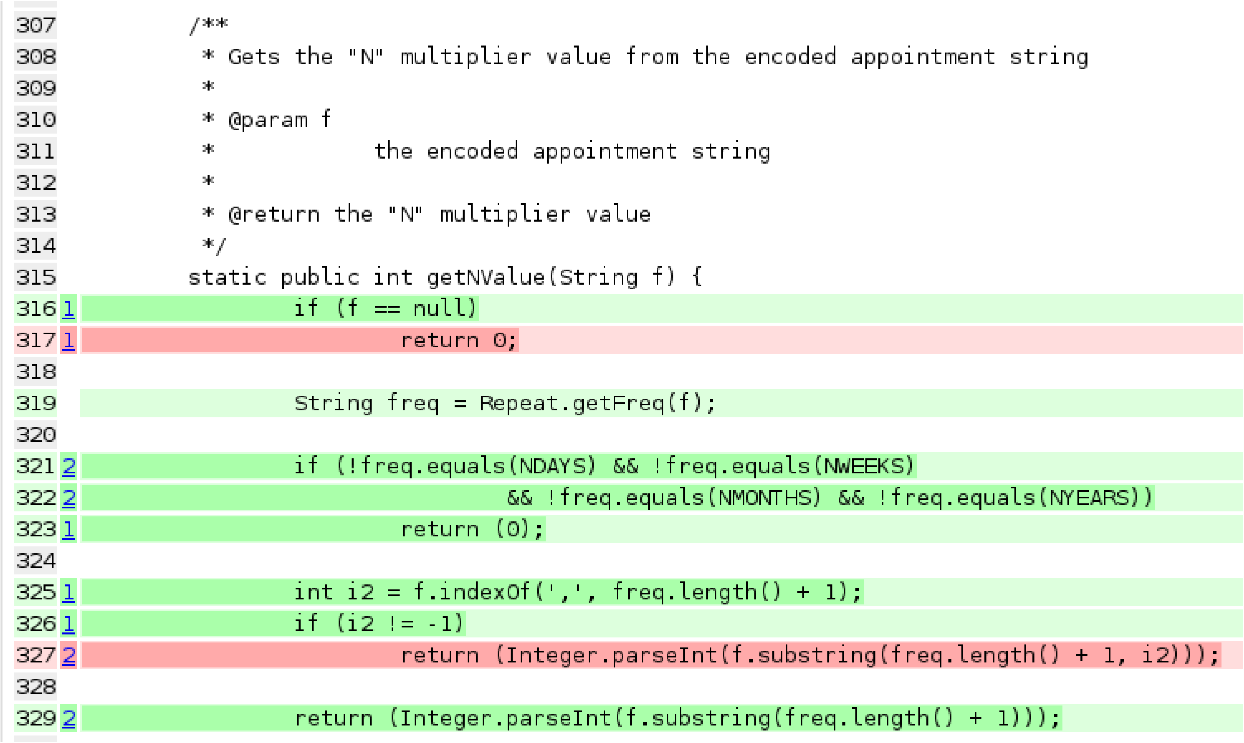
Breakdown by Class

Name	Line Coverage		Mutation Coverage	
Repeat.java	17%	44/258	11%	23/201

Report generated by PIT 1.1.4

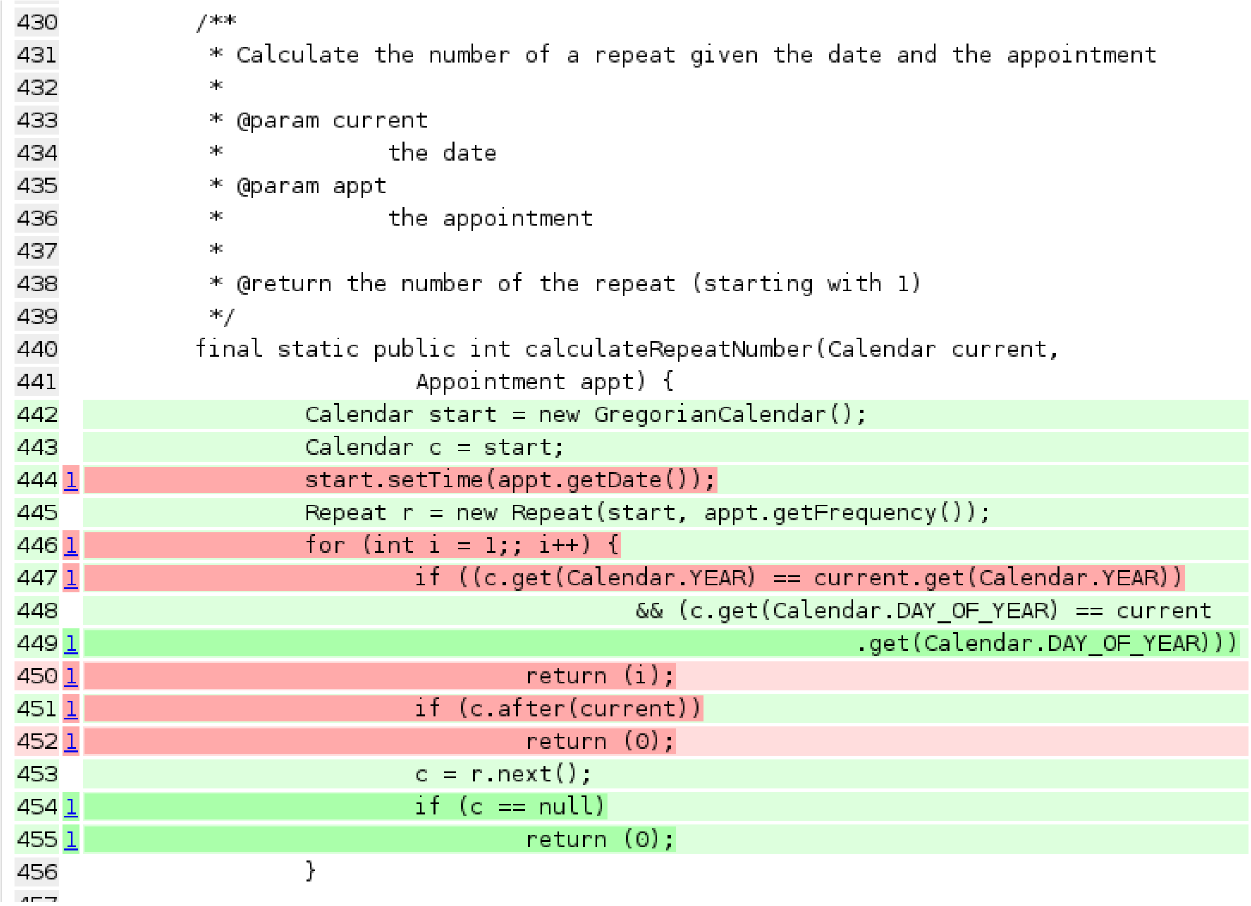
Figure 9: net.sf.borg.model.Repeat

This overview is not particularly useful since there are many unrelated methods in this class that were not the focus of the tests. The detailed mutation coverage of the two methods under test are as follows:



```
          /**
           * Gets the "N" multiplier value from the encoded appointment string
           *
           * @param f
           *             the encoded appointment string
           *
           * @return the "N" multiplier value
           */
          static public int getNValue(String f) {
1                 if (f == null)
1                         return 0;

                  String freq = Repeat.getFreq(f);

2                 if (!freq.equals(NDAYS) && !freq.equals(NWEEKS)
2                                 && !freq.equals(NMONTHS) && !freq.equals(NYEARS))
1                         return (0);

1                 int i2 = f.indexOf(',', freq.length() + 1);
1                 if (i2 != -1)
2                         return (Integer.parseInt(f.substring(freq.length() + 1, i2)));

2                 return (Integer.parseInt(f.substring(freq.length() + 1)));
```

Figure 10: net.sf.borg.model.Repeat getNValue(...)



```
430          /**
431           * Calculate the number of a repeat given the date and the appointment
432           *
433           * @param current
434           *            the date
435           * @param appt
436           *            the appointment
437           *
438           * @return the number of the repeat (starting with 1)
439           */
440          final static public int calculateRepeatNumber(Calendar current,
441                          Appointment appt) {
442                  Calendar start = new GregorianCalendar();
443                  Calendar c = start;
444 1                start.setTime(appt.getDate());
445                  Repeat r = new Repeat(start, appt.getFrequency());
446 1                for (int i = 1;; i++) {
447 1                        if ((c.get(Calendar.YEAR) == current.get(Calendar.YEAR))
448                                          && (c.get(Calendar.DAY_OF_YEAR) == current
449 1                                                        .get(Calendar.DAY_OF_YEAR)))
450 1                                return (i);
451 1                        if (c.after(current))
452 1                                return (0);
453                          c = r.next();
454 1                        if (c == null)
455 1                                return (0);
456                  }
```

Figure 11: net.sf.borg.model.Repeat calculateRepeatNumber(...)

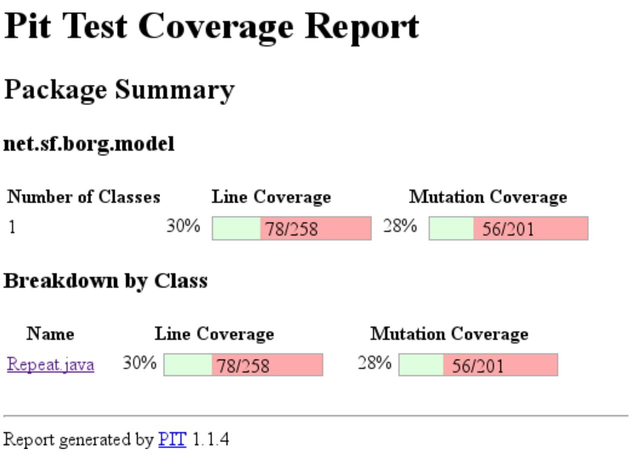
Since Assignment 2 was a black-box testing exercise, several of these are expected. For example, it was not clear from the documentation that `getNValue(...)` should behave differently if there were 2 commas in the input string (lines 325-337). Other mutation failures, such as ones in `calculateRepeatNumber(...)` indicate that the original testing was too minimal; some of those mutations are more trivial (lines 444, 451-452).

`Repeat.java` Test Repair

In order to repair the missing mutation handling, four new tests were written:

1. Test to ensure that `calculateRepeatNumber(...)` sets the dates correctly (kills lines 444) by testing that on two separate days, the repeat number is different.
2. Test to ensure that `calculateRepeatNumber(...)` loop calculates the repeat number as expected, by creating a more detailed typical use-case test.
3. Test to ensure that `getNValue(...)` returns 0 when passed in a `null`.
4. Test to ensure that `getNValue(...)` returns the number between two commas.

The updated source code has been submitted in the submit package. The updated coverage results are below:



Pit Test Coverage Report

Package Summary

net.sf.borg.model

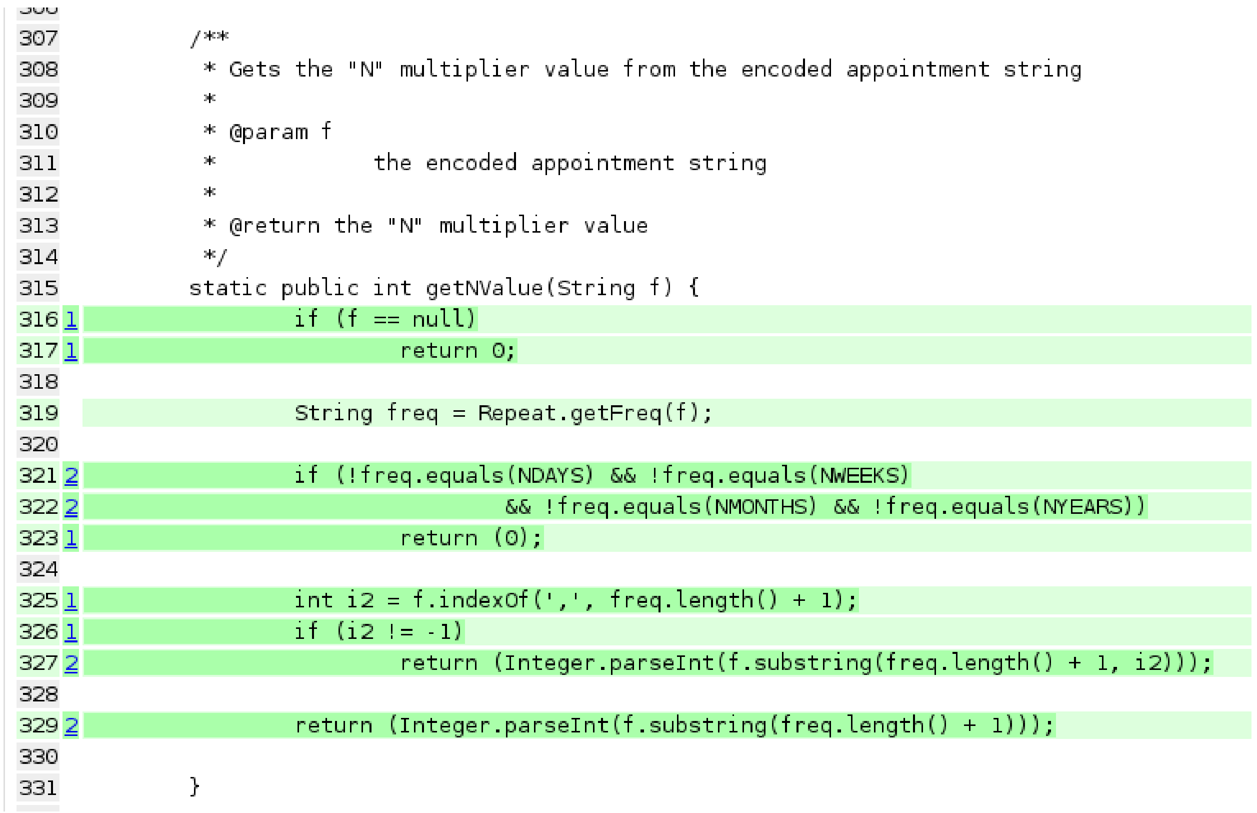
Number of Classes	Line Coverage		Mutation Coverage	
1	30%	78/258	28%	56/201

Breakdown by Class

Name	Line Coverage		Mutation Coverage	
Repeat.java	30%	78/258	28%	56/201

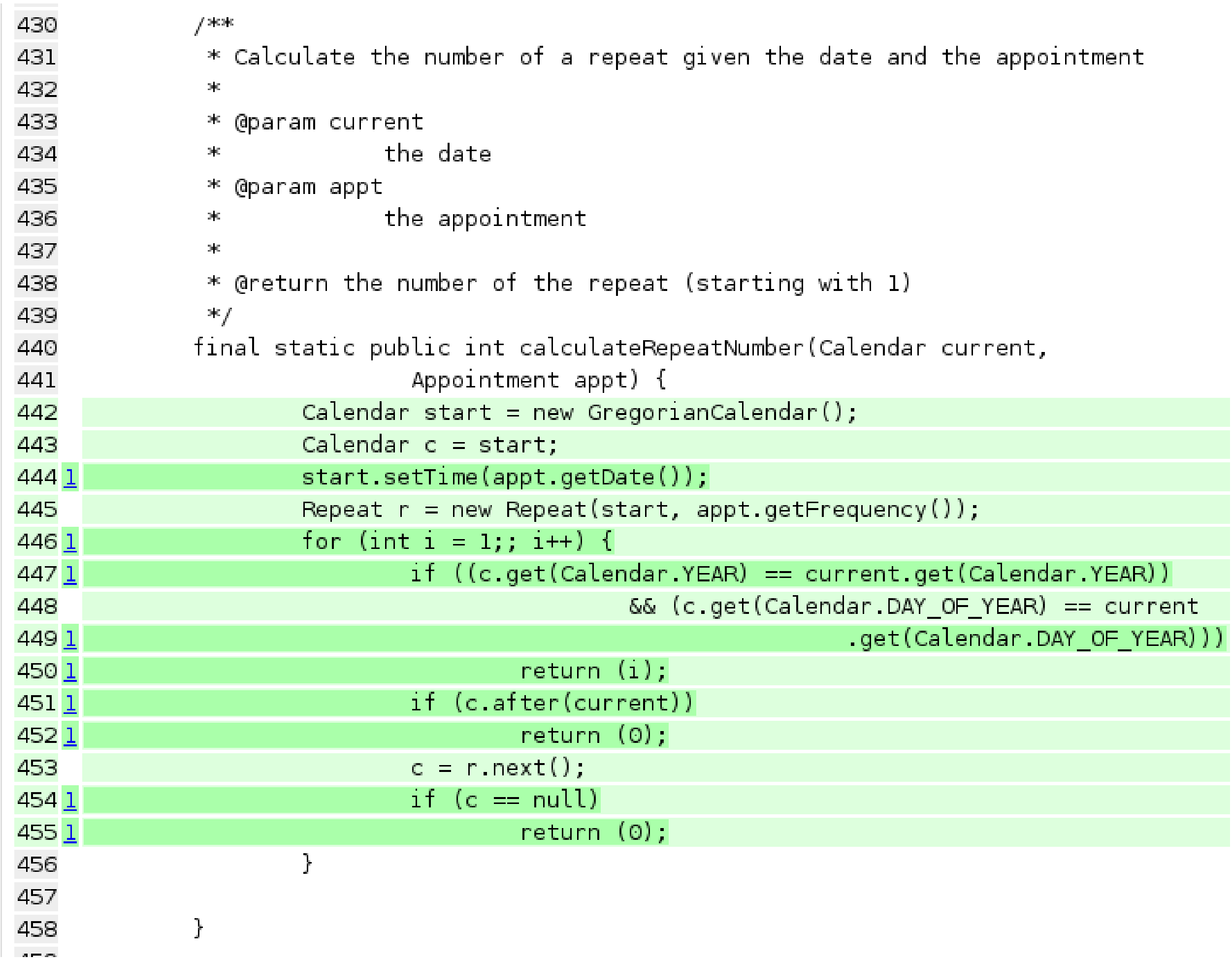
Report generated by PIT 1.1.4

Figure 12: net.sf.borg.model.Repeat Updated



```
/**
 * Gets the "N" multiplier value from the encoded appointment string
 *
 * @param f
 *            the encoded appointment string
 *
 * @return the "N" multiplier value
 */
static public int getNValue(String f) {
        if (f == null)
                return 0;

        String freq = Repeat.getFreq(f);

        if (!freq.equals(NDAYS) && !freq.equals(NWEEKS)
                        && !freq.equals(NMONTHS) && !freq.equals(NYEARS))
                return (0);

        int i2 = f.indexOf(',', freq.length() + 1);
        if (i2 != -1)
                return (Integer.parseInt(f.substring(freq.length() + 1, i2)));

        return (Integer.parseInt(f.substring(freq.length() + 1)));

}
```

Figure 13: net.sf.borg.model.Repeat getNValue(...) Updated

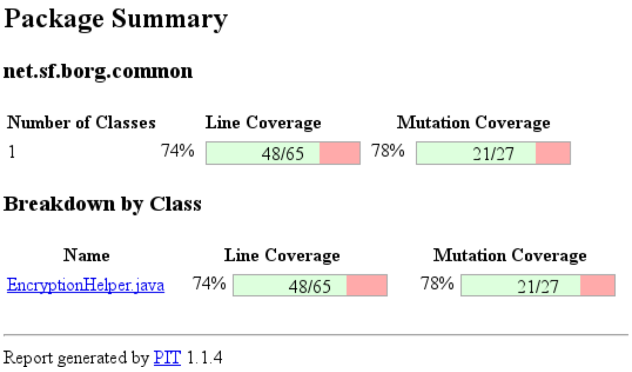


```
/**
 * Calculate the number of a repeat given the date and the appointment
 *
 * @param current
 *             the date
 * @param appt
 *             the appointment
 *
 * @return the number of the repeat (starting with 1)
 */
final static public int calculateRepeatNumber(Calendar current,
                Appointment appt) {
        Calendar start = new GregorianCalendar();
        Calendar c = start;
        start.setTime(appt.getDate());
        Repeat r = new Repeat(start, appt.getFrequency());
        for (int i = 1;; i++) {
                if ((c.get(Calendar.YEAR) == current.get(Calendar.YEAR))
                                && (c.get(Calendar.DAY_OF_YEAR) == current
                                                .get(Calendar.DAY_OF_YEAR)))
                        return (i);
                if (c.after(current))
                        return (0);
                c = r.next();
                if (c == null)
                        return (0);
        }

}
```

Figure 14: net.sf.borg.model.Repeat calculateRepeatNumber(...) Updated

EncryptionHelper Test Analysis

After restricting the tests to show the class and test results of only the written tests, the following overview was produced:



Package Summary

net.sf.borg.common

Number of Classes	Line Coverage		Mutation Coverage	
1	74%	48/65	78%	21/27

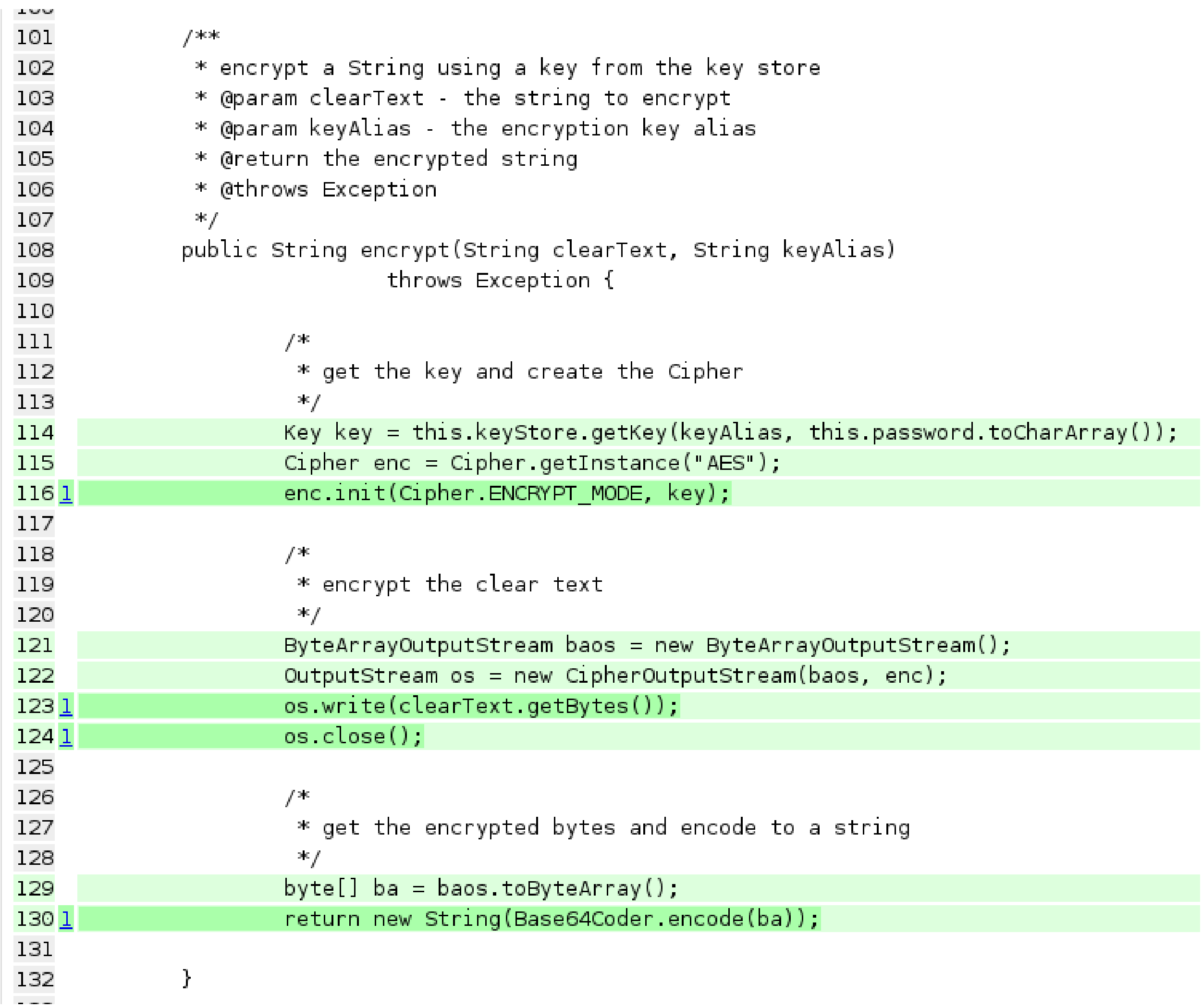
Breakdown by Class

Name	Line Coverage		Mutation Coverage	
EncryptionHelper.java	74%	48/65	78%	21/27

Report generated by PIT 1.1.4

Figure 15: net.sf.borg.common

There is only one method under test in this class, `encrypt(...)`. It passed all of the mutation tests with no modifications, as seen here:



```
        /**
         * encrypt a String using a key from the key store
         * @param clearText - the string to encrypt
         * @param keyAlias - the encryption key alias
         * @return the encrypted string
         * @throws Exception
         */
        public String encrypt(String clearText, String keyAlias)
                        throws Exception {

                /*
                 * get the key and create the Cipher
                 */
                Key key = this.keyStore.getKey(keyAlias, this.password.toCharArray());
                Cipher enc = Cipher.getInstance("AES");
                enc.init(Cipher.ENCRYPT_MODE, key);

                /*
                 * encrypt the clear text
                 */
                ByteArrayOutputStream baos = new ByteArrayOutputStream();
                OutputStream os = new CipherOutputStream(baos, enc);
                os.write(clearText.getBytes());
                os.close();

                /*
                 * get the encrypted bytes and encode to a string
                 */
                byte[] ba = baos.toByteArray();
                return new String(Base64Coder.encode(ba));

        }
```

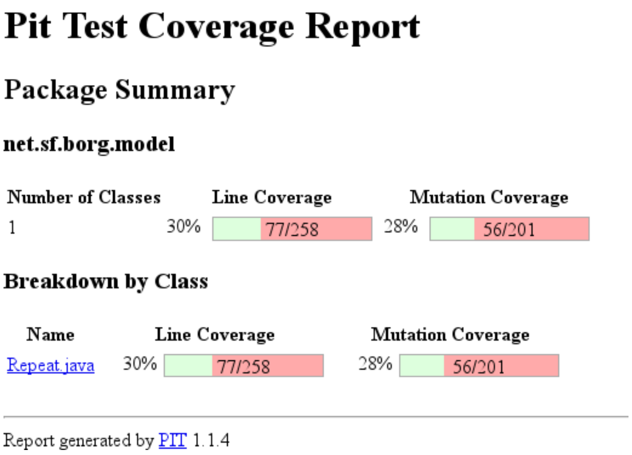
Figure 16: net.sf.borg.common encryption(...)

This is likely because there were many tests written for this method, yet this method just communicates with built-in Java methods. Since the original tests correctly killed all the mutants, the test source code was not modified.

Assignment 3 Review

`Repeat.java` Test Analysis

After being permitted to view the source code, the tests written for `net.sf.borg.model.Repeat` significantly improved in quality. The overview:



Pit Test Coverage Report

Package Summary

net.sf.borg.model

Number of Classes	Line Coverage		Mutation Coverage	
1	30%	77/258	28%	56/201

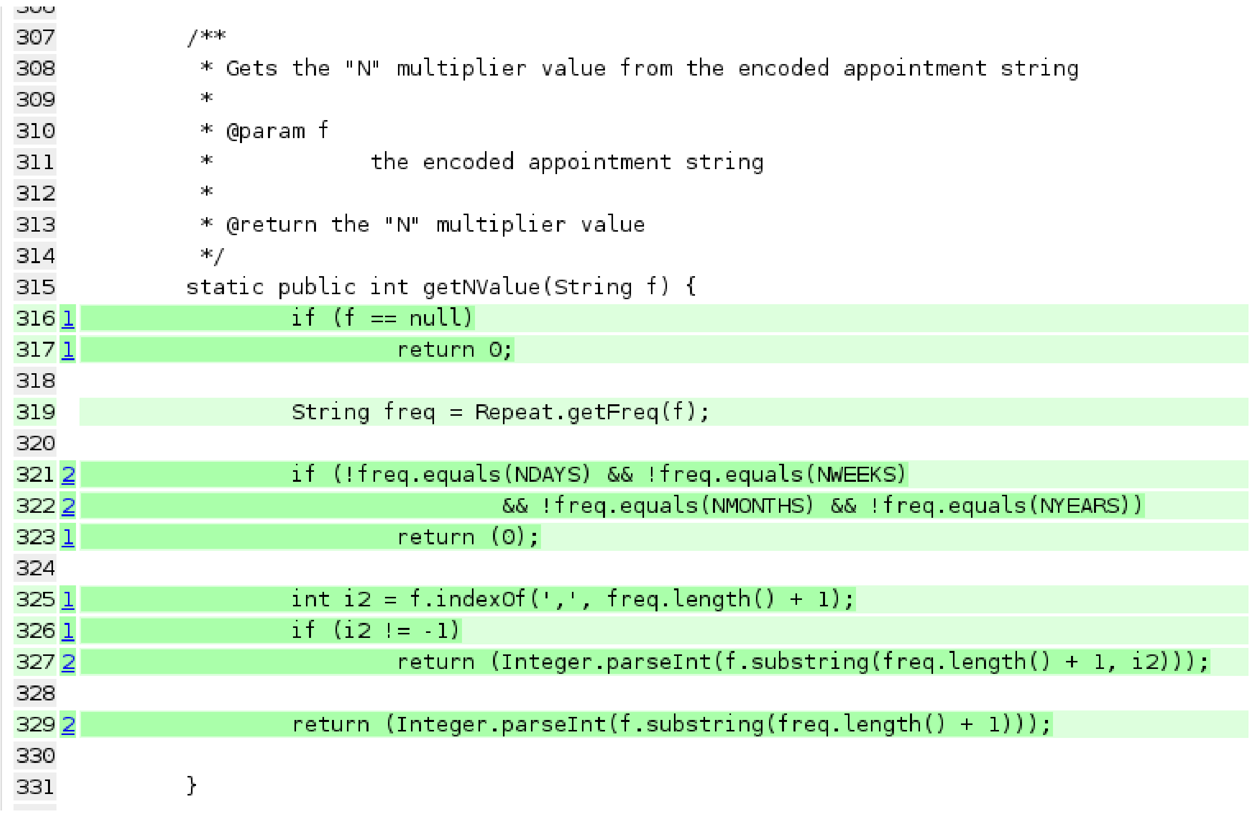
Breakdown by Class

Name	Line Coverage		Mutation Coverage	
Repeat.java	30%	77/258	28%	56/201

Report generated by PIT 1.1.4

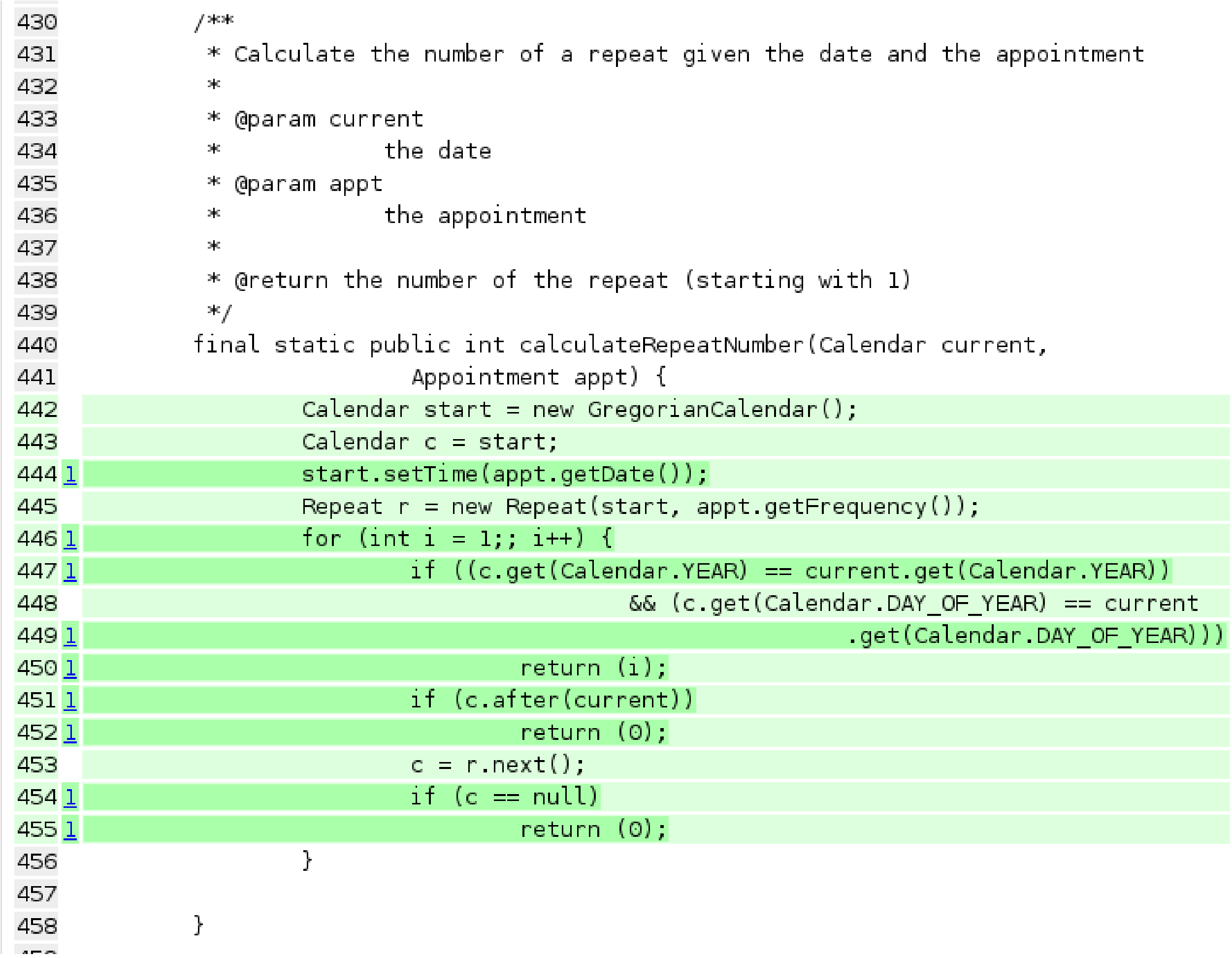
Figure 17: net.sf.borg.model.Repeat

Note that this is actually 1 fewer mutant kill than in the Assignment 2 analysis after adding tests. Viewing the detailed information confirmed this finding.



```
307          /**
308           * Gets the "N" multiplier value from the encoded appointment string
309           *
310           * @param f
311           *            the encoded appointment string
312           *
313           * @return the "N" multiplier value
314           */
315          static public int getNValue(String f) {
316 1                if (f == null)
317 1                        return 0;
318
319                  String freq = Repeat.getFreq(f);
320
321 2                if (!freq.equals(NDAYS) && !freq.equals(NWEEKS)
322 2                                && !freq.equals(NMONTHS) && !freq.equals(NYEARS))
323 1                        return (0);
324
325 1                int i2 = f.indexOf(',', freq.length() + 1);
326 1                if (i2 != -1)
327 2                        return (Integer.parseInt(f.substring(freq.length() + 1, i2)));
328
329 2                return (Integer.parseInt(f.substring(freq.length() + 1)));
330
331          }
```

Figure 18: net.sf.borg.model.Repeat getNValue(...)



```
/**
 * Calculate the number of a repeat given the date and the appointment
 *
 * @param current
 *             the date
 * @param appt
 *             the appointment
 *
 * @return the number of the repeat (starting with 1)
 */
final static public int calculateRepeatNumber(Calendar current,
                Appointment appt) {
        Calendar start = new GregorianCalendar();
        Calendar c = start;
        start.setTime(appt.getDate());
        Repeat r = new Repeat(start, appt.getFrequency());
        for (int i = 1;; i++) {
                if ((c.get(Calendar.YEAR) == current.get(Calendar.YEAR))
                                && (c.get(Calendar.DAY_OF_YEAR) == current
                                                .get(Calendar.DAY_OF_YEAR)))
                        return (i);
                if (c.after(current))
                        return (0);
                c = r.next();
                if (c == null)
                        return (0);
        }

}
```

Figure 19: net.sf.borg.model.Repeat calculateRepeatNumber(...)

The original tests from Assignment 2 killed all the mutants successfully, and so did not require any modifications. Additionally, the `EncryptionHelper` tests were not changed between Assignment 2 and Assignment 3, and so they still kill all their mutants as well. Therefore, all tests written and submitted as part of Assignment 3 killed all the mutants created by PIT. No code modifications were made for A4.

Appendix A

This appendix contains each of our method specifications from Assignment 2.

Method 1

```
/**
 * Calculate the number of a repeat given the date and the appointment
 *
 * @param current
 *            the date
 * @param appt
 *            the appointment
 *
 * @return the number of the repeat (starting with 1)
 */
final static public int calculateRepeatNumber(Calendar current,
    Appointment appt) {
  Calendar start = new GregorianCalendar();
  Calendar c = start;
  start.setTime(appt.getDate());
  Repeat r = new Repeat(start, appt.getFrequency());
  for (int i = 1;; i++) {
    if ((c.get(Calendar.YEAR) == current.get(Calendar.YEAR))
        && (c.get(Calendar.DAY_OF_YEAR) == current
            .get(Calendar.DAY_OF_YEAR)))
      return (i);
    if (c.after(current))
      return (0);
    c = r.next();
    if (c == null)
      return (0);
  }

}
```

Method 2

```
    /**
     * encrypt a String using a key from the key store
     * @param clearText - the string to encrypt
     * @param keyAlias - the encryption key alias
     * @return the encrypted string
     * @throws Exception
     */
    public String encrypt(String clearText, String keyAlias)
            throws Exception {

        /*
         * get the key and create the Cipher
         */
        Key key = this.keyStore.getKey(keyAlias, this.password.toCharArray());
        Cipher enc = Cipher.getInstance("AES");
        enc.init(Cipher.ENCRYPT_MODE, key);

        /*
         * encrypt the clear text
         */
        ByteArrayOutputStream baos = new ByteArrayOutputStream();
        OutputStream os = new CipherOutputStream(baos, enc);
        os.write(clearText.getBytes());
        os.close();

        /*
         * get the encrypted bytes and encode to a string
         */
        byte[] ba = baos.toByteArray();
        return new String(Base64Coder.encode(ba));

    }
```

Method 3

```
/**
 * Gets the "N" multiplier value from the encoded appointment string
 *
 * @param f
 *            the encoded appointment string
 *
 * @return the "N" multiplier value
 */
static public int getNValue(String f) {
  if (f == null)
    return 0;

  String freq = Repeat.getFreq(f);

  if (!freq.equals(NDAYS) && !freq.equals(NWEEKS)
      && !freq.equals(NMONTHS) && !freq.equals(NYEARS))
    return (0);

  int i2 = f.indexOf(',', freq.length() + 1);
  if (i2 != -1)
    return (Integer.parseInt(f.substring(freq.length() + 1, i2)));

  return (Integer.parseInt(f.substring(freq.length() + 1)));

}
```