EECS 4313 Assignment 3

Data Flow Testing, Slice-Based Testing and Mutation Testing

Anton Sitkovets (212118048)
Mina Zaki (212857975)
Jeremi Boston (212432399)
David Iliaguiev (210479830)

Task 1 - BORG Calendar

Program To Be Tested

```
1▼      /**
2        * generate a human readable string for a particular number of minutes
3        *
4        * @param mins - the number of minutes
5        *
6        * @return the string
7        */
8▼      public static String minuteString(int mins) {
9
10           int hours = mins / 60;
11           int minsPast = mins % 60;
12
13           String minutesString;
14           String hoursString;
15
16           if (hours > 1) {
17               hoursString = hours + " " + Resource.getResourceString("Hours");
18           } else if (hours > 0) {
19               hoursString = hours + " " + Resource.getResourceString("Hour");
20           } else {
21               hoursString = "";
22           }
23
24           if (minsPast > 1) {
25               minutesString = minsPast + " " + Resource.getResourceString("Minutes");
26           } else if (minsPast > 0) {
27               minutesString = minsPast + " " + Resource.getResourceString("Minute");
28           } else if (hours >= 1) {
29               minutesString = "";
30           } else {
31               minutesString = minsPast + " " + Resource.getResourceString("Minutes");
32           }
33
34           // space between hours and minutes
35           if (!hoursString.equals("") && !minutesString.equals(""))
36               minutesString = " " + minutesString;
37
38           return hoursString + minutesString;
39       }
40   }
```

Program Segmented

public static String minuteString(int mins) {	A
int hours = mins / 60;	B
int minsPast = mins % 60;	C
String minutesString;	D
String hoursString;	E
if (hours > 1) {	F
hoursString = hours + " " + Resource.getResourceString("Hours");	G
} else if (hours > 0) {	H
hoursString = hours + " " + Resource.getResourceString("Hour");	I

} else {
hoursString = "";
}
	J

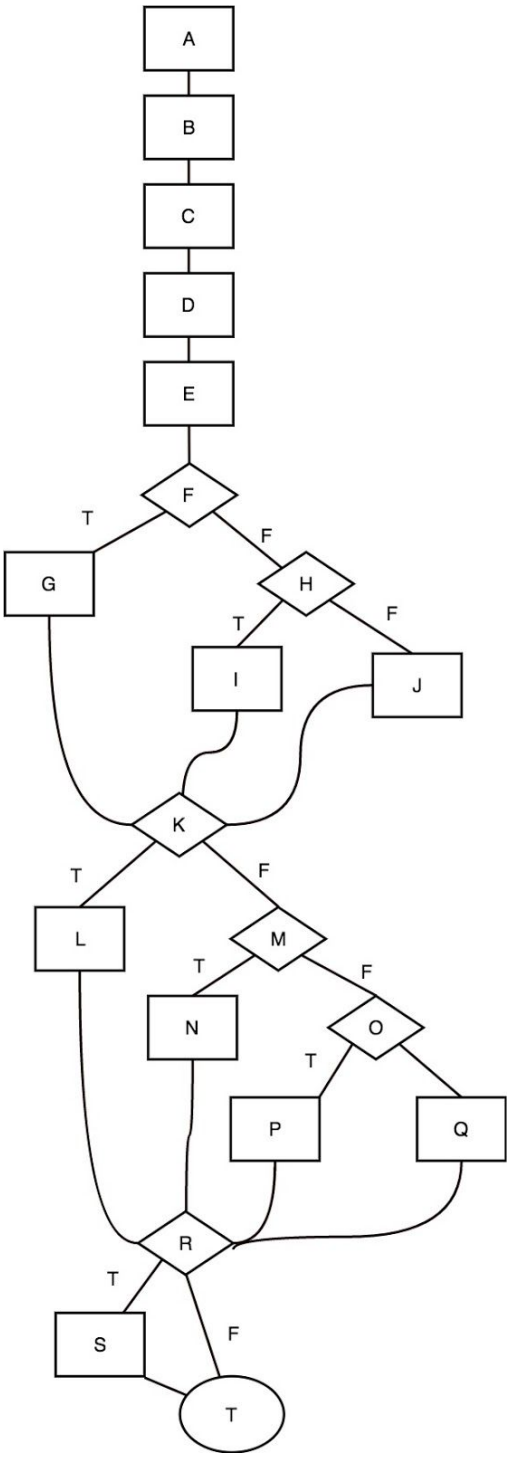
if (minsPast > 1) {	K
minutesString = minsPast + " " + Resource.getResourceString("Minutes");	L
} else if (minsPast > 0) {	M
minutesString = minsPast + " " + Resource.getResourceString("Minute");	N
} else if (hours >= 1) {	O
minutesString = "";	P

} else {
minutesString = minsPast + " " + Resource.getResourceString("Minutes");
}
	Q

if (!hoursString.equals("") && !minutesString.equals("")){	R
minutesString = " " + minutesString; }	S
return hoursString + minutesString;	T

}	

Program Graph



Data Flow Paths For All Variables

dc-Paths for mins

All Defs

- Each definition of each variable for at least one use of the definition
    - AB (or ABC)

All Uses

- At least one path of each variable definition to each p-use and each cuse of the definition
    - AB, ABC

All P-Uses / Some C-uses

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

All C-uses / 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
    - AB, ABC

Test Cases:

```
// All Defs: Path AB
// All Uses: Paths AB, ABC
// All P-Uses / Some C-uses: Path AB
// All C-uses / Some P-uses: Paths AB, ABC
assertEquals("2 Minutes", DateUtil.minuteString(2));
```

dc-Paths for hours

All Defs

- Each definition of each variable for at least one use of the definition
    - BCDEF (or BCDEFG or BCDEFH or BCDEFHI or BCDEFHKMO)

All Uses

- At least one path of each variable definition to each p-use and each cuse of the definition
    - BCDEF, BCDEFG, BCDEFH, BCDEFHI, BCDEFHKMO

All P-Uses / Some C-uses

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

All C-uses / 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
    - BCDEFG, BCDEFHI

Test Cases:

```
// All Defs: Path BCDEF
assertEquals("2 Hours 2 Minutes", DateUtil.minuteString(122));

// All Uses: Paths BCDEF, BCDEFG, BCDEFH, BCDEFHI, BCDEFHKMO
assertEquals("1 Hour 2 Minutes", DateUtil.minuteString(62));
assertEquals("2 Hours 2 Minutes", DateUtil.minuteString(122));
assertEquals("1 Hour", DateUtil.minuteString(60));

// All P-Uses / Some C-uses: Paths BCDEFHKMO
assertEquals("1 Hour", DateUtil.minuteString(60));

// All C-uses / Some P-uses: Paths BCDEFG, BCDEFHI
assertEquals("1 Hour 5 Minutes", DateUtil.minuteString(65));
assertEquals("2 Hours 12 Minutes", DateUtil.minuteString(132));
```

dc-Paths for minsPast

All Defs

- Each definition of each variable for at least one use of the definition
    - CDEFHK (or CDEFHKL or CDEFHKM or CDEFHKMN or CDEFHKMNOQ)

All Uses

- At least one path of each variable definition to each p-use and each cuse of the definition
    - CDEFHK, CDEFHKL, CDEFHKM, CDEFHKMN, CDEFHKMNOQ

All P-Uses / Some C-uses

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

All C-uses / 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
    - CDEFHKL, CDEFHKMN, CDEFHKMNOQ

Test Cases:

```
// All Defs: Path CDEFHK
assertEquals("2 Minutes", DateUtil.minuteString(2));

// All Uses: Paths CDEFHK, CDEFHKL, CDEFHKM, CDEFHKMN, CDEFHKMNOQ
assertEquals("1 Hour 1 Minute", DateUtil.minuteString(61));
assertEquals("0 Minutes", DateUtil.minuteString(0));

// All P-Uses / Some C-uses: Paths CDEFHK, CDEFHKM
assertEquals("1 Hour 3 Minutes", DateUtil.minuteString(63));
assertEquals("2 Hours 1 Minute", DateUtil.minuteString(121));

// All C-uses / Some P-uses: Paths CDEFHKL, CDEFHKMN, CDEFHKMNOQ
assertEquals("1 Hour 5 Minutes", DateUtil.minuteString(65));
assertEquals("2 Hours 12 Minutes", DateUtil.minuteString(132));
```

dc-Paths for minutesString

All Defs

- Each definition of each variable for at least one use of the definition
    - DEFHKL (or DEFHKMN or DEFHKMOP or DEFHKMOQ or DEFHKMOR or DEFHKMORS or ST)

All Uses

- At least one path of each variable definition to each p-use and each cuse of the definition
    - DEFHKL, DEFHKMN , DEFHKMOP, DEFHKMOQ, DEFHKMOR, DEFHKMORS, ST

All P-Uses / Some C-uses

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

All C-uses / 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
    - DEFHKL, DEFHKMN , DEFHKMOP, DEFHKMOQ, DEFHKMORS, ST

Test Cases:

```
// All Defs: Path DEFHKL
assertEquals("2 Minutes", DateUtil.minuteString(2));

// All Uses: Paths DEFHKL, DEFHKMN , DEFHKMOP, DEFHKMOQ, DEFHKMOR,
DEFHKMORS, ST
assertEquals("1 Hour 1 Minute", DateUtil.minuteString(61));
assertEquals("0 Minutes", DateUtil.minuteString(0));
assertEquals("1 Hour", DateUtil.minuteString(60));

// All P-Uses / Some C-uses: Paths DEFHKMOR
assertEquals("2 Hours 1 Minute", DateUtil.minuteString(121));
assertEquals("1 Hour", DateUtil.minuteString(60));
assertEquals("0 Minutes", DateUtil.minuteString(0));

// All C-uses / Some P-uses: Paths DEFHKL, DEFHKMN , DEFHKMOP,
DEFHKMOQ, DEFHKMORS, ST
assertEquals("1 Hour 5 Minutes", DateUtil.minuteString(65));
assertEquals("2 Hours 1 Minute", DateUtil.minuteString(121));
assertEquals("2 Minutes", DateUtil.minuteString(2));
assertEquals("1 Hour", DateUtil.minuteString(60));
```

dc-Paths for hoursString

All Defs

- Each definition of each variable for at least one use of the definition
    - EFG (or EFHI or EFHJ or EFHKMOR or EFHKMORT)

All Uses

- At least one path of each variable definition to each p-use and each cuse of the definition
    - EFG, EFHI, EFHJ, EFHKMOR, EFHKMORT

All P-Uses / Some C-uses

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

All C-uses / 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

- EFG, EFHI, EFHJ, EFHKMORT

Test Cases:

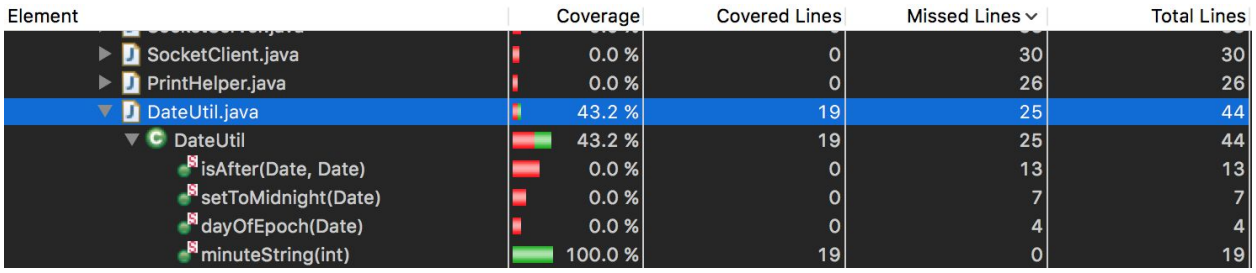
```
// All Defs: Path EFG
assertEquals("3 Hours", DateUtil.minuteString(180));

// All Uses: Paths EFG, EFHI, EFHJ, EFHKMOR, EFHKMORT
assertEquals("1 Hour 1 Minute", DateUtil.minuteString(61));
assertEquals("2 Hours 1 Minute", DateUtil.minuteString(121));
assertEquals("0 Minutes", DateUtil.minuteString(0));
assertEquals("1 Hour", DateUtil.minuteString(60));

// All P-Uses / Some C-uses: Paths  EFHKMOR
assertEquals("2 Hours 1 Minute", DateUtil.minuteString(121));
assertEquals("1 Hour", DateUtil.minuteString(60));
assertEquals("0 Minutes", DateUtil.minuteString(0));

// All C-uses / Some P-uses: Paths EFG, EFHI, EFHJ, EFHKMORT
assertEquals("1 Hour 5 Minutes", DateUtil.minuteString(65));
assertEquals("2 Hours 1 Minute", DateUtil.minuteString(121));
assertEquals("2 Minutes", DateUtil.minuteString(2));
```

Coverage Results

```
100         public static String minuteString(int mins) {
101
102 1               int hours = mins / 60;
103 1               int minsPast = mins % 60;
104
105               String minutesString;
106               String hoursString;
107
108 2               if (hours > 1) {
109                       hoursString = hours + " " + Resource.getResourceString("Hours");
110 2               } else if (hours > 0) {
111                       hoursString = hours + " " + Resource.getResourceString("Hour");
112               } else {
113                       hoursString = "";
114               }
115
116 2               if (minsPast > 1) {
117                       minutesString = minsPast + " " + Resource.getResourceString("Minutes");
118 2               } else if (minsPast > 0) {
119                       minutesString = minsPast + " " + Resource.getResourceString("Minute");
120 2               } else if (hours >= 1) {
121                       minutesString = "";
122               } else {
123                       minutesString = minsPast + " " + Resource.getResourceString("Minutes");
124               }
125
126               // space between hours and minutes
127 2               if (!hoursString.equals("") && !minutesString.equals(""))
128                       minutesString = " " + minutesString;
129
130 1               return hoursString + minutesString;
131         }
```

Element	Coverage	Covered Instructions	Missed Instructions	Total Instructions
SocketServer.java	0.0 %	0	151	151
IsAfterTest.java	0.0 %	0	106	106
DateUtil.java	53.2 %	115	101	216
DateUtil	53.2 %	115	101	216
isAfter(Date, Date)	0.0 %	0	48	48
setToMidnight(Date)	0.0 %	0	26	26
dayOfEpoch(Date)	0.0 %	0	24	24
minuteString(int)	100.0 %	115	0	115

Element	Coverage	Covered Branches	Missed Branches	Total Branches
SocketClient.java	0.0 %	0	12	12
EncryptionHelper.java	0.0 %	0	6	6
DateUtil.java	87.5 %	14	2	16
DateUtil	87.5 %	14	2	16
isAfter(Date, Date)	0.0 %	0	2	2
dayOfEpoch(Date)		0	0	0
minuteString(int)	100.0 %	14	0	14
setToMidnight(Date)		0	0	0



Element	Coverage	Covered Lines	Missed Lines	Total Lines
SocketClient.java	0.0 %	0	30	30
PrintHelper.java	0.0 %	0	26	26
DateUtil.java	43.2 %	19	25	44
DateUtil	43.2 %	19	25	44
isAfter(Date, Date)	0.0 %	0	13	13
setToMidnight(Date)	0.0 %	0	7	7
dayOfEpoch(Date)	0.0 %	0	4	4
minuteString(int)	100.0 %	19	0	19

As can be seen the the lines, branches, and instructions covered are all at 100%. Although prior to the data flow analysis, all these metrics were at 100%, adding the data flow analysis test cases were added to illustrate that all possible paths were considered and handled in the test class (seen in the appendix). Our original test cases from assignment 2 were able to achieve 100% coverage because we looked at the code and made sure to cover all the possible branches and results. Although we were unaware at the time, this was us directly doing data flow analysis of the variables and their respective usages.

Slicing

Program To Be Tested

```java
/**
 * generate a human readable string for a particular number of minutes
 *
 * @param mins - the number of minutes
 *
 * @return the string
 */
public static String minuteString(int mins) {

    int hours = mins / 60;
    int minsPast = mins % 60;

    String minutesString;
    String hoursString;

    if (hours > 1) {
        hoursString = hours + " " + Resource.getResourceString("Hours");
    } else if (hours > 0) {
        hoursString = hours + " " + Resource.getResourceString("Hour");
    } else {
        hoursString = "";
    }

    if (minsPast > 1) {
        minutesString = minsPast + " " + Resource.getResourceString("Minutes");
    } else if (minsPast > 0) {
        minutesString = minsPast + " " + Resource.getResourceString("Minute");
    } else if (hours >= 1) {
        minutesString = "";
    } else {
        minutesString = minsPast + " " + Resource.getResourceString("Minutes");
    }

    // space between hours and minutes
    if (!hoursString.equals("") && !minutesString.equals(""))
        minutesString = " " + minutesString;

    return hoursString + minutesString;
}
}
```

Forward Slicing

Forward slices are of the form S(v,n), which consists of all the statements that are affected by the variable v at the statement n. The slices before the definition n, are empty.
The testcase `assertEquals("2 Hours 1 Minute", DateUtil.minuteString(121));` will cover all the forward slices.

S(hours, 10)

```
int hours = mins / 60;
int minsPast = mins % 60;

String minutesString;
String hoursString;

if (hours > 1) {
    hoursString = hours + " " + Resource.getResourceString("Hours");
} else if (hours > 0) {
    hoursString = hours + " " + Resource.getResourceString("Hour");
} else {
    hoursString = "";
}

if (minsPast > 1) {
    minutesString = minsPast + " " + Resource.getResourceString("Minutes");
} else if (minsPast > 0) {
    minutesString = minsPast + " " + Resource.getResourceString("Minute");
} else if (hours >= 1) {
    minutesString = "";
} else {
    minutesString = minsPast + " " + Resource.getResourceString("Minutes");
}

// space between hours and minutes
if (!hoursString.equals("") && !minutesString.equals(""))
    minutesString = " " + minutesString;

return hoursString + minutesString;
```

S(minsPast, 11)

```
int minsPast = mins % 60;

String minutesString;

if (minsPast > 1) {
    minutesString = minsPast + " " + Resource.getResourceString("Minutes");
} else if (minsPast > 0) {
    minutesString = minsPast + " " + Resource.getResourceString("Minute");
} else if (hours >= 1) {
    minutesString = "";
} else {
    minutesString = minsPast + " " + Resource.getResourceString("Minutes");
}

// space between hours and minutes
if (!hoursString.equals("") && !minutesString.equals(""))
    minutesString = " " + minutesString;

return hoursString + minutesString;
```

S(minuteString, 13)

```
String minutesString;

if (minsPast > 1) {
    minutesString = minsPast + " " + Resource.getResourceString("Minutes");
} else if (minsPast > 0) {
    minutesString = minsPast + " " + Resource.getResourceString("Minute");
} else if (hours >= 1) {
    minutesString = "";
} else {
    minutesString = minsPast + " " + Resource.getResourceString("Minutes");
}

// space between hours and minutes
if (!hoursString.equals("") && !minutesString.equals(""))
    minutesString = " " + minutesString;

return hoursString + minutesString;
```

S(hoursString, 14)

```
String hoursString;

if (hours > 1) {
    hoursString = hours + " " + Resource.getResourceString("Hours");
} else if (hours > 0) {
    hoursString = hours + " " + Resource.getResourceString("Hour");
} else {
    hoursString = "";
}

// space between hours and minutes
if (!hoursString.equals("") && !minutesString.equals(""))
    minutesString = " " + minutesString;

return hoursString + minutesString;
```

Backward Slicing

Backward slices are of the form S(v,n), which consists of all the statements that affect the variable v at the statement n.

S(hours, 10)

```
public static String minuteString(int mins) {

        int hours = mins / 60;
```

For any value of n < 10, results would be an empty slice.

S(minsPast, 11)

```
public static String minuteString(int mins) {

        int minsPast = mins % 60;
```

S(minutesString, 25)

```
int minsPast = mins % 60;

String minutesString;

if (minsPast > 1) {
    minutesString = minsPast + " " + Resource.getResourceString("Minutes");
}
```

The testcase `assertEquals("2 Minutes", DateUtil.minuteString(2));` will cover all the above backward slices.

S(minutesString, 27)

```
int minsPast = mins % 60;

String minutesString;

if (minsPast > 1) {
    minutesString = minsPast + " " + Resource.getResourceString("Minutes");
}
else if (minsPast > 0) {
    minutesString = minsPast + " " + Resource.getResourceString("Minute");
```

The testcase `assertEquals("1 Minute", DateUtil.minuteString(1));` will cover all the above backward slice.

S(minutesString, 29)

```
int hours = mins / 60;
int minsPast = mins % 60;

String minutesString;


if (minsPast > 1) {
    minutesString = minsPast + " " + Resource.getResourceString("Minutes");
} else if (minsPast > 0) {
    minutesString = minsPast + " " + Resource.getResourceString("Minute");
} else if (hours >= 1) {
    minutesString = "";
}
```

The testcase `assertEquals("1 Hour", DateUtil.minuteString(60));` will cover all the above backward slice.

S(minutesString, 31)

```
int hours = mins / 60;
int minsPast = mins % 60;

String minutesString;


if (minsPast > 1) {
    minutesString = minsPast + " " + Resource.getResourceString("Minutes");
} else if (minsPast > 0) {
    minutesString = minsPast + " " + Resource.getResourceString("Minute");
} else if (hours >= 1) {
    minutesString = "";
}
else {
    minutesString = minsPast + " " + Resource.getResourceString("Minutes");
}
```

The testcase `assertEquals("0 Minutes", DateUtil.minuteString(0));` will cover all the above backward slice.

S(minutesString, 36) and S(hoursString, 36)

```
int hours = mins / 60;
int minsPast = mins % 60;

String minutesString;
String hoursString;

if (hours > 1) {
    hoursString = hours + " " + Resource.getResourceString("Hours");
} else if (hours > 0) {
    hoursString = hours + " " + Resource.getResourceString("Hour");
} else {
    hoursString = "";
}

if (minsPast > 1) {
    minutesString = minsPast + " " + Resource.getResourceString("Minutes");
} else if (minsPast > 0) {
    minutesString = minsPast + " " + Resource.getResourceString("Minute");
} else if (hours >= 1) {
    minutesString = "";
} else {
    minutesString = minsPast + " " + Resource.getResourceString("Minutes");
}

// space between hours and minutes
if (!hoursString.equals("") && !minutesString.equals(""))
    minutesString = " " + minutesString;
```

The testcase `assertEquals("1 Hour 2 Minutes", DateUtil.minuteString(62));` will cover all the above backward slice.

S(hoursString, 17)

```
int hours = mins / 60;

String hoursString;

if (hours > 1) {
    hoursString = hours + " " + Resource.getResourceString("Hours");
}
```

The testcase `assertEquals("2 Hours", DateUtil.minuteString(120));` will cover all the above backward slice.

S(hoursString, 19)

```
int hours = mins / 60;

String hoursString;

if (hours > 1) {
    hoursString = hours + " " + Resource.getResourceString("Hours");
}
else if (hours > 0) {
    hoursString = hours + " " + Resource.getResourceString("Hour");
}
```

The testcase `assertEquals("1 Hour", DateUtil.minuteString(60));` will cover all the above backward slice.

S(hoursString, 21)

```
int hours = mins / 60;

String hoursString;

if (hours > 1) {
    hoursString = hours + " " + Resource.getResourceString("Hours");
}
else if (hours > 0) {
    hoursString = hours + " " + Resource.getResourceString("Hour");
}
else {
    hoursString = "";
}
```

The testcase `assertEquals("2 Minutes", DateUtil.minuteString(2));` will cover all the above backward slice.

Dynamic Slicing

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

- Mins is 0. Tested by `assertEquals("0 Minutes", DateUtil.minuteString(0));`
- Mins is 1. Tested by `assertEquals("1 Minute", DateUtil.minuteString(1));`
- Mins is 2. Tested by `assertEquals("2 Minutes", DateUtil.minuteString(2));`
- Mins is 60. Tested by `assertEquals("1 Hour", DateUtil.minuteString(60));`
- Mins is 61. Tested by `assertEquals("1 Hour 1 Minute", DateUtil.minuteString(61));`

- Mins is 62. Tested by `assertEquals("1 Hour 2 Minutes", DateUtil.minuteString(62));`
- Mins is 122. Tested by `assertEquals("2 Hours 2 Minutes", DateUtil.minuteString(122));`

S(mins = 0, minutesString, 38)

```
int hours = mins / 60;
int minsPast = mins % 60;

String minutesString;

if (minsPast > 1) {

} else if (minsPast > 0) {

} else if (hours >= 1) {

} else {
    minutesString = minsPast + " " + Resource.getResourceString("Minutes");
}

// space between hours and minutes
if (!hoursString.equals("") && !minutesString.equals(""))
    minutesString = " " + minutesString;

return hoursString + minutesString;
```

S(mins = 0, hoursString, 38), S(mins = 1, hoursString, 38), S(mins = 2, hoursString, 38)

```
int hours = mins / 60;
int minsPast = mins % 60;

String minutesString;
String hoursString;

if (hours > 1) {

} else if (hours > 0) {

} else {
    hoursString = "";
}

// space between hours and minutes
if (!hoursString.equals("") && !minutesString.equals(""))


return hoursString + minutesString;
```

S(mins = 1, minutesString, 38)

```
int hours = mins / 60;
int minsPast = mins % 60;

String minutesString;

if (minsPast > 1) {

} else if (minsPast > 0) {
    minutesString = minsPast + " " + Resource.getResourceString("Minute");
} else if (hours >= 1) {

} else {

}

// space between hours and minutes
if (!hoursString.equals("") && !minutesString.equals(""))
    minutesString = " " + minutesString;

return hoursString + minutesString;
```

S(mins = 2, minutesString, 38)

```
int hours = mins / 60;
int minsPast = mins % 60;

String minutesString;


if (minsPast > 1) {
    minutesString = minsPast + " " + Resource.getResourceString("Minutes");
} else if (minsPast > 0) {

} else if (hours >= 1) {

} else {

}

// space between hours and minutes
if (!hoursString.equals("") && !minutesString.equals(""))
    minutesString = " " + minutesString;

return hoursString + minutesString;
```

S(mins = 60, minutesString, 38)

```
int hours = mins / 60;
int minsPast = mins % 60;

String minutesString;
String hoursString;

if (minsPast > 1) {

} else if (minsPast > 0) {

} else if (hours >= 1) {
    minutesString = "";
} else {

}

// space between hours and minutes
if (!hoursString.equals("") && !minutesString.equals(""))
    minutesString = " " + minutesString;

return hoursString + minutesString;
```

S(mins = 60, hoursString, 38), S(mins = 61, hoursString, 38), S(mins = 62, hoursString, 38)

```
int hours = mins / 60;
int minsPast = mins % 60;

String minutesString;
String hoursString;

if (hours > 1) {

} else if (hours > 0) {
    hoursString = hours + " " + Resource.getResourceString("Hour");
} else {

}

return hoursString + minutesString;
```

S(mins = 61, minutesString, 38)

```
int hours = mins / 60;
int minsPast = mins % 60;

String minutesString;
String hoursString;



if (minsPast > 1) {

} else if (minsPast > 0) {
    minutesString = minsPast + " " + Resource.getResourceString("Minute");
} else if (hours >= 1) {

} else {

}

// space between hours and minutes
if (!hoursString.equals("") && !minutesString.equals(""))
    minutesString = " " + minutesString;

return hoursString + minutesString;
```

S(mins = 62, minutesString, 38), S(mins = 122, minutesString, 38)

```
int hours = mins / 60;
int minsPast = mins % 60;

String minutesString;
String hoursString;

if (minsPast > 1) {
    minutesString = minsPast + " " + Resource.getResourceString("Minutes");
} else if (minsPast > 0) {

} else if (hours >= 1) {

} else {

}

// space between hours and minutes
if (!hoursString.equals("") && !minutesString.equals(""))
    minutesString = " " + minutesString;

return hoursString + minutesString;
```

S(mins = 122, hoursString, 38)

```
int hours = mins / 60;
int minsPast = mins % 60;

String minutesString;
String hoursString;

if (hours > 1) {
    hoursString = hours + " " + Resource.getResourceString("Hours");
} else if (hours > 0) {

} else {

}

return hoursString + minutesString;
```

No test cases were added to improve the test coverage of the slices as all these test cases already existed in the test suite. Our slice coverage was already high because we had done data flow analysis and considered all the possible paths of each variable.

Mutation Testing

The mutation tests were conducted on the tests we wrote. From assignment 2 we had 100% statement coverage over the methods we tested.

isAfterTest

```
40          public static boolean isAfter(Date d1, Date d2) {
41
42                  GregorianCalendar tcal = new GregorianCalendar();
43  1               tcal.setTime(d1);
44  1               tcal.set(Calendar.HOUR_OF_DAY, 0);
45  1               tcal.set(Calendar.MINUTE, 0);
46  1               tcal.set(Calendar.SECOND, 0);
47                  GregorianCalendar dcal = new GregorianCalendar();
48  1               dcal.setTime(d2);
49  1               dcal.set(Calendar.HOUR_OF_DAY, 0);
50  1               dcal.set(Calendar.MINUTE, 10);
51  1               dcal.set(Calendar.SECOND, 0);
52
53  1               if (tcal.getTime().after(dcal.getTime())) {
54  1                       return true;
55                  }
56
57  1               return false;
58          }
```

Mutations

43	1. removed call to java/util/GregorianCalendar::setTime → KILLED
44	1. removed call to java/util/GregorianCalendar::set → SURVIVED
45	1. removed call to java/util/GregorianCalendar::set → SURVIVED
46	1. removed call to java/util/GregorianCalendar::set → SURVIVED
48	1. removed call to java/util/GregorianCalendar::setTime → KILLED
49	1. removed call to java/util/GregorianCalendar::set → SURVIVED
50	1. removed call to java/util/GregorianCalendar::set → SURVIVED
51	1. removed call to java/util/GregorianCalendar::set → SURVIVED
53	1. negated conditional → KILLED
54	1. replaced return of integer sized value with (x == 0 ? 1 : 0) → KILLED
57	1. replaced return of integer sized value with (x == 0 ? 1 : 0) → KILLED

Looking at the result of the mutation test, we can see that the method isAfter will still pass tests despite removing the lines 44-46 and 49-51. This proves that these lines are not needed in the implementation of the method. After seeing the results of this test, we attempted to cover those cases by using the Date constructor with hours, minutes and seconds.

```
Date d3 = new Date(2017, 8, 15, 8, 30, 22);
Date d4 = new Date(2017, 8, 15, 9, 30, 22);
// Since both happen at the same day, but at different hours and
// the implementation ignores HMS, it will always be false
assertFalse(DateUtil.isAfter(d4, d3)); // this should be True
assertFalse(DateUtil.isAfter(d3, d4));
```

However, we found that the method isAfter does not pay attention to those values in the date object as it will automatically create a new Gregorian calendar with values of 0 for the hour, minute and second, no matter the values given as input. This is a new bug and the mutants cannot be killed by any tests as this is an issue in the code itself. We have attached a bug report for this in the appendix.

MinuteStringTest

```
100         public static String minuteString(int mins) {
101
102 1               int hours = mins / 60;
103 1               int minsPast = mins % 60;
104
105               String minutesString;
106               String hoursString;
107
108 2               if (hours > 1) {
109                       hoursString = hours + " " + Resource.getResourceString("Hours");
110 2               } else if (hours > 0) {
111                       hoursString = hours + " " + Resource.getResourceString("Hour");
112               } else {
113                       hoursString = "";
114               }
115
116 2               if (minsPast > 1) {
117                       minutesString = minsPast + " " + Resource.getResourceString("Minutes");
118 2               } else if (minsPast > 0) {
119                       minutesString = minsPast + " " + Resource.getResourceString("Minute");
120 2               } else if (hours >= 1) {
121                       minutesString = "";
122               } else {
123                       minutesString = minsPast + " " + Resource.getResourceString("Minutes");
124               }
125
126               // space between hours and minutes
127 2               if (!hoursString.equals("") && !minutesString.equals(""))
128                       minutesString = " " + minutesString;
129
130 1               return hoursString + minutesString;
131         }
```

102	1. Replaced integer division with multiplication → KILLED
103	1. Replaced integer modulus with multiplication → KILLED
108	1. changed conditional boundary → KILLED 2. negated conditional → KILLED
110	1. changed conditional boundary → KILLED 2. negated conditional → KILLED
116	1. changed conditional boundary → KILLED 2. negated conditional → KILLED
118	1. changed conditional boundary → KILLED 2. negated conditional → KILLED
120	1. changed conditional boundary → KILLED 2. negated conditional → KILLED
127	1. negated conditional → KILLED 2. negated conditional → KILLED
130	1. mutated return of Object value for net/sf/borg/common/DateUtil::minuteString to (if (x != null) null else throw new RuntimeException) → KILLED

When running the test from assignment 2, all mutants were killed and adding the data flow tests for assignment 3 did not introduce any new issues with the mutation test.

DayListTest Initially

```
276         static public Collection<Integer> getDaylist(String f) {
277                 ArrayList<Integer> daylist = new ArrayList<Integer>();
278 2               if (f == null || !f.startsWith(DAYLIST))
279 1                       return daylist;
280
281 1               int i2 = f.indexOf(',', DAYLIST.length() + 1);
282                 String list = null;
283 1               if (i2 != -1)
284 1                       list = f.substring(DAYLIST.length() + 1, i2);
285                 else
286 1                       list = f.substring(DAYLIST.length() + 1);
287
288 1               if (list.indexOf("1") != -1)
289                         daylist.add(new Integer(Calendar.SUNDAY));
290 1               if (list.indexOf("2") != -1)
291                         daylist.add(new Integer(Calendar.MONDAY));
292 1               if (list.indexOf("3") != -1)
293                         daylist.add(new Integer(Calendar.TUESDAY));
294 1               if (list.indexOf("4") != -1)
295                         daylist.add(new Integer(Calendar.WEDNESDAY));
296 1               if (list.indexOf("5") != -1)
297                         daylist.add(new Integer(Calendar.THURSDAY));
298 1               if (list.indexOf("6") != -1)
299                         daylist.add(new Integer(Calendar.FRIDAY));
300 1               if (list.indexOf("7") != -1)
301                         daylist.add(new Integer(Calendar.SATURDAY));
302
303 1               return (daylist);
```

278	1. negated conditional → KILLED 2. negated conditional → KILLED
279	1. mutated return of Object value for net/sf/borg/model/Repeat::getDaylist to (if (x != null) null else throw new RuntimeException) → KILLED
281	1. Replaced integer addition with subtraction → KILLED
283	1. negated conditional → KILLED
284	1. Replaced integer addition with subtraction → NO_COVERAGE
286	1. Replaced integer addition with subtraction → SURVIVED
288	1. negated conditional → KILLED
290	1. negated conditional → KILLED
292	1. negated conditional → KILLED
294	1. negated conditional → KILLED
296	1. negated conditional → KILLED
298	1. negated conditional → KILLED
300	1. negated conditional → KILLED
303	1. mutated return of Object value for net/sf/borg/model/Repeat::getDaylist to (if (x != null) null else throw new RuntimeException) → KILLED

Here we notice that the mutation tests have found two missing test cases not previously found by coverage calculations. The two mutants that survive, are situations where an addition is swapped for a subtraction when specifying indices of a substring. Additional unit tests were added to attempt to fix this issue.

After Additional Tests added

```
276          static public Collection<Integer> getDaylist(String f) {
277 1                    ArrayList<Integer> daylist = new ArrayList<Integer>();
278 2                    if (f == null || !f.startsWith(DAYLIST))
279 1                            return daylist;
280
281 1                    int i2 = f.indexOf(',', DAYLIST.length() + 1);
282                      String list = null;
283 1                    if (i2 != -1){
284 1                            list = f.substring(DAYLIST.length() + 1, i2);
285                      }else{
286 1                            list = f.substring(DAYLIST.length() + 1);
287                      }
288 1                    if (list.indexOf("1") != -1)
289                              daylist.add(new Integer(Calendar.SUNDAY));
290 1                    if (list.indexOf("2") != -1)
291                              daylist.add(new Integer(Calendar.MONDAY));
292 1                    if (list.indexOf("3") != -1)
293                              daylist.add(new Integer(Calendar.TUESDAY));
294 1                    if (list.indexOf("4") != -1)
295                              daylist.add(new Integer(Calendar.WEDNESDAY));
296 1                    if (list.indexOf("5") != -1)
297                              daylist.add(new Integer(Calendar.THURSDAY));
298 1                    if (list.indexOf("6") != -1)
299                              daylist.add(new Integer(Calendar.FRIDAY));
300 1                    if (list.indexOf("7") != -1)
301                              daylist.add(new Integer(Calendar.SATURDAY));
302
303 1                    return (daylist);
```

278	1. negated conditional → KILLED 2. negated conditional → KILLED
279	1. mutated return of Object value for net/sf/borg/model/Repeat::getDaylist to (if (x != null) null else throw new RuntimeException) → KILLED
281	1. Replaced integer addition with subtraction → KILLED
283	1. negated conditional → KILLED
284	1. Replaced integer addition with subtraction → KILLED
286	1. Replaced integer addition with subtraction → KILLED
288	1. negated conditional → KILLED
290	1. negated conditional → KILLED
292	1. negated conditional → KILLED
294	1. negated conditional → KILLED
296	1. negated conditional → KILLED
298	1. negated conditional → KILLED
300	1. negated conditional → KILLED
303	1. mutated return of Object value for net/sf/borg/model/Repeat::getDaylist to (if (x != null) null else throw new RuntimeException) → KILLED

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

1. Test to ensure that the method getDayList works as expected with poorly formatted inputs. This kills line 286 by asserting that numbers only occur after two indices away from the string “dlist”, i.e there should be a space between the end of “dlist” and the beginning of the sequence of integers.

```
assertEquals(l, Repeat.getDaylist("dlist2 1")); //where l is a list
containing the integer 1.
```

2. Test to ensure that the method getDayList works as expected with poorly formatted inputs that contain commas. This kills line 284 by asserting that numbers only occur after two indices away from the string “dlist” and before the comma. Meaning there should be a space between the end of “dlist” and the beginning of the sequence of integers, and the sequence of integers must be after the string “dlist” but before the comma.

```
assertEquals(l, Repeat.getDaylist("dlist2  1,")); //where l is a list
containing the integer 1.
```

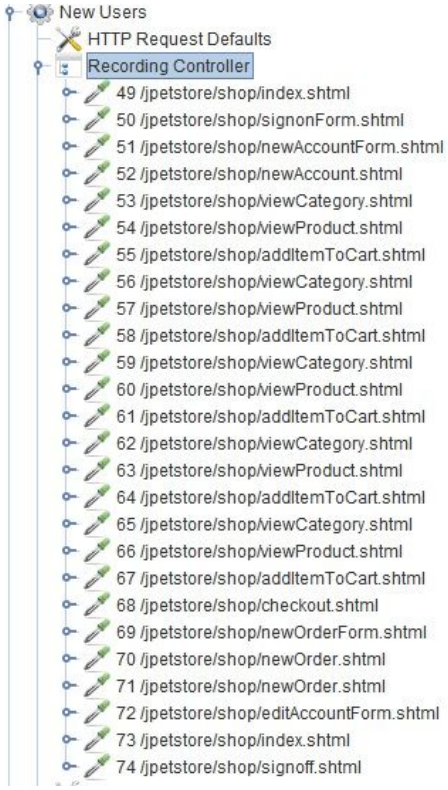
The updated tests have been submitted in the submit package and attached in the appendix.

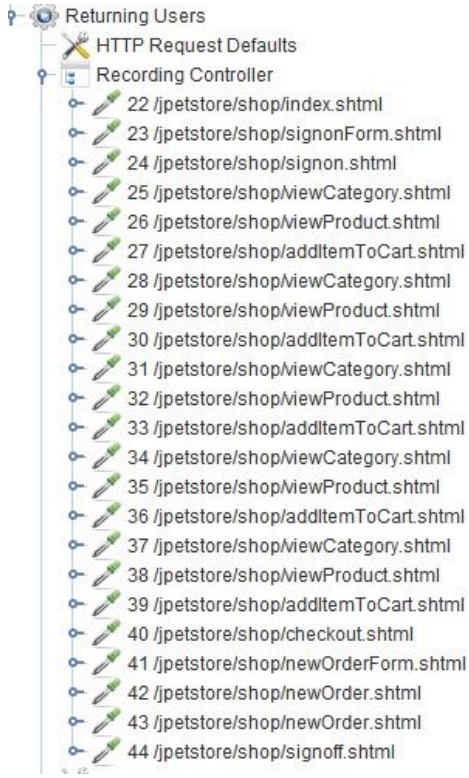
Task 2 -JPetStore

JPetStore Load Testing

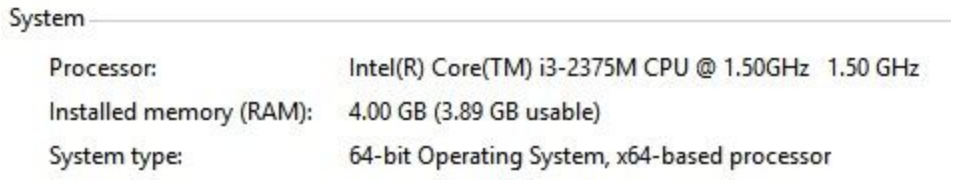
Looking through the website there was a couple test cases that we deemed necessary to check. The first is to create a new user, browse the website, add items to the cart, purchase, then logout. This was done through Jmeter with variables and 10 different usernames stored in a csv file. The second case is the same but for a returning customer there does not need to be a new user created but the returning users again are switched between 6 different users a1 to a6 these are also stored in a separate csv file. The use cases can be seen in the diagrams below.


Use Case 1
Use Case 2
New User
Login
Login
Browse
Browse
Purchase
Purchase
Logout
Logout


New Users
HTTP Request Defaults
Recording Controller
49 /jpetstore/shop/index.shtml
50 /jpetstore/shop/signonForm.shtml
51 /jpetstore/shop/newAccountForm.shtml
52 /jpetstore/shop/newAccount.shtml
53 /jpetstore/shop/viewCategory.shtml
54 /jpetstore/shop/viewProduct.shtml
55 /jpetstore/shop/addItemToCart.shtml
56 /jpetstore/shop/viewCategory.shtml
57 /jpetstore/shop/viewProduct.shtml
58 /jpetstore/shop/addItemToCart.shtml
59 /jpetstore/shop/viewCategory.shtml
60 /jpetstore/shop/viewProduct.shtml
61 /jpetstore/shop/addItemToCart.shtml
62 /jpetstore/shop/viewCategory.shtml
63 /jpetstore/shop/viewProduct.shtml
64 /jpetstore/shop/addItemToCart.shtml
65 /jpetstore/shop/viewCategory.shtml
66 /jpetstore/shop/viewProduct.shtml
67 /jpetstore/shop/addItemToCart.shtml
68 /jpetstore/shop/checkout.shtml
69 /jpetstore/shop/newOrderForm.shtml
70 /jpetstore/shop/newOrder.shtml
71 /jpetstore/shop/newOrder.shtml
72 /jpetstore/shop/editAccountForm.shtml
73 /jpetstore/shop/index.shtml
74 /jpetstore/shop/signoff.shtml


Returning Users
HTTP Request Defaults
Recording Controller
22 /jpetstore/shop/index.shtml
23 /jpetstore/shop/signonForm.shtml
24 /jpetstore/shop/signon.shtml
25 /jpetstore/shop/viewCategory.shtml
26 /jpetstore/shop/viewProduct.shtml
27 /jpetstore/shop/addItemToCart.shtml
28 /jpetstore/shop/viewCategory.shtml
29 /jpetstore/shop/viewProduct.shtml
30 /jpetstore/shop/addItemToCart.shtml
31 /jpetstore/shop/viewCategory.shtml
32 /jpetstore/shop/viewProduct.shtml
33 /jpetstore/shop/addItemToCart.shtml
34 /jpetstore/shop/viewCategory.shtml
35 /jpetstore/shop/viewProduct.shtml
36 /jpetstore/shop/addItemToCart.shtml
37 /jpetstore/shop/viewCategory.shtml
38 /jpetstore/shop/viewProduct.shtml
39 /jpetstore/shop/addItemToCart.shtml
40 /jpetstore/shop/checkout.shtml
41 /jpetstore/shop/newOrderForm.shtml
42 /jpetstore/shop/newOrder.shtml
43 /jpetstore/shop/newOrder.shtml
44 /jpetstore/shop/signoff.shtml

The Apache Tomcat server was run on had the following specifications:

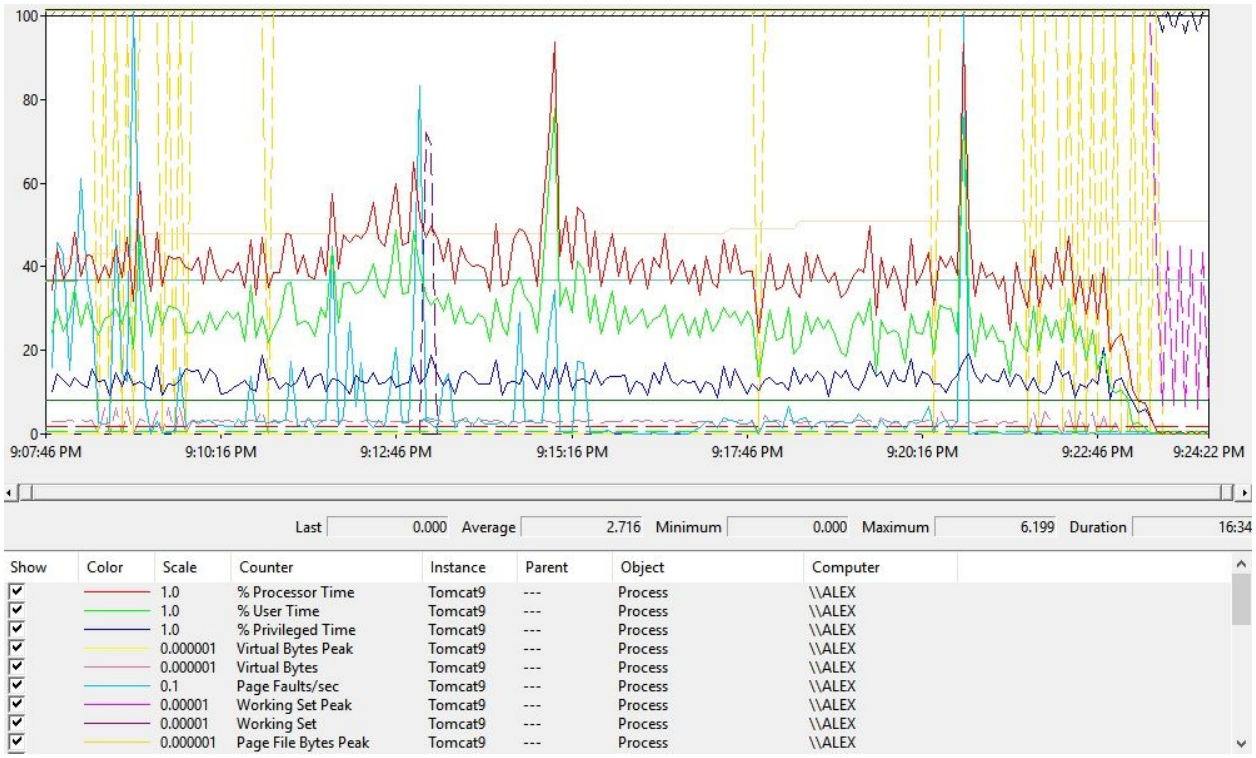


The load tests were run on the above 2 use cases on 3 separate occasions each for 15 minutes each. For both use cases the settings were as follows 5 threads (users) with 1 ramp up and recurring loops until stopped at 15 minutes. All requests were successful, no errors occurred, the server system performed well, some sample requests at the end of one of the runs can be seen below.

```
63036  127.0.0.1 - - [02/Apr/2017:21:24:46 -0400] "GET /jpetstore/shop/viewCategory.shtml?categoryId=BIRDS HTTP/1.1" 200 3657
63037  127.0.0.1 - - [02/Apr/2017:21:24:46 -0400] "GET /jpetstore/shop/addItemToCart.shtml?workingItemId=EST-17 HTTP/1.1" 200 6542
63038  127.0.0.1 - - [02/Apr/2017:21:24:46 -0400] "GET /jpetstore/shop/viewProduct.shtml?productId=FL-DLH-02 HTTP/1.1" 200 4094
63039  127.0.0.1 - - [02/Apr/2017:21:24:46 -0400] "GET /jpetstore/shop/viewProduct.shtml?productId=AV-CB-01 HTTP/1.1" 200 3855
63040  127.0.0.1 - - [02/Apr/2017:21:24:47 -0400] "GET /jpetstore/shop/viewCategory.shtml?categoryId=BIRDS HTTP/1.1" 200 3504
63041  127.0.0.1 - - [02/Apr/2017:21:24:47 -0400] "GET /jpetstore/shop/addItemToCart.shtml?workingItemId=EST-17 HTTP/1.1" 200 6542
63042  127.0.0.1 - - [02/Apr/2017:21:24:47 -0400] "GET /jpetstore/shop/addItemToCart.shtml?workingItemId=EST-18 HTTP/1.1" 200 6814
63043  127.0.0.1 - - [02/Apr/2017:21:24:58 -0400] "GET /jpetstore/shop/viewProduct.shtml?productId=AV-CB-01 HTTP/1.1" 200 3702
63044  127.0.0.1 - - [02/Apr/2017:21:24:58 -0400] "GET /jpetstore/shop/checkout.shtml HTTP/1.1" 200 4470
63045  127.0.0.1 - - [02/Apr/2017:21:25:01 -0400] "GET /jpetstore/shop/newOrderForm.shtml HTTP/1.1" 200 5113
63046  127.0.0.1 - - [02/Apr/2017:21:25:01 -0400] "GET /jpetstore/shop/addItemToCart.shtml?workingItemId=EST-18 HTTP/1.1" 200 6646
63047  127.0.0.1 - - [02/Apr/2017:21:25:06 -0400] "GET /jpetstore/shop/checkout.shtml HTTP/1.1" 200 3506
63048
```

In the graph below we can see the performance counter graph showing the end of one of the 15 minute runs. The graph shows the typical movement of the load requests tested with some peak moments in extra processing indicating I/O requests to the server, this is due to the moment when many POST requests are being made at the same time to the server. This is indicated in the Tomcat access logs, showing that at 9:21pm there were many POST requests sent and a spike in the processing and I/O occurred, and this can be seen at the same point on the PerfMon graph below as well. There were no stagnation points in reading or writing to disk, or steady movement and then incremental steps up in reading or writing to disk which would have indicated a memory leak and a possible crash on the server. Overall the system performed well under the load.

```
310590  127.0.0.1 - - [03/Apr/2017:21:21:32 -0400] "POST /jpetstore/shop/newAccount.shtml HTTP/1.1" 200 13189
310591  127.0.0.1 - - [03/Apr/2017:21:21:32 -0400] "POST /jpetstore/shop/newAccount.shtml HTTP/1.1" 200 13189
310592  127.0.0.1 - - [03/Apr/2017:21:21:32 -0400] "POST /jpetstore/shop/newAccount.shtml HTTP/1.1" 200 13189
310593  127.0.0.1 - - [03/Apr/2017:21:21:32 -0400] "POST /jpetstore/shop/newAccount.shtml HTTP/1.1" 200 13189
310594  127.0.0.1 - - [03/Apr/2017:21:21:32 -0400] "POST /jpetstore/shop/newOrder.shtml HTTP/1.1" 200 5297
```



Appendix

Specification of the selected Java methods

1.1 common > DateUtil > isAfter

public static boolean isAfter(Date d1, Date d2)
This method checks if one date happens on a later date than the other.
When d1 occurs after d2, the return value is true.
When d2 occurs after d1, the return value is false.

- The first argument is the first date in the comparison
- The second argument is the second date in the comparison

1.2 common > DateUtil > MinuteString

public static String minuteString(int mins)
This method takes a number of minutes and returns a human readable representation in hours and minutes.
The return values will look like this for 62 minutes: 1 Hour 2 Minutes.
The return value will look like this for 60 minutes: 1 Hour
The return value will look like this for 12 minutes: 12 Minutes

- The parameter mins is the number of minutes to convert to a minute string

1.3 model > Repeat > getDayList

static public Collection<Integer> getDaylist(String f)
This method takes a string formatted as “dlist xxxx” or “dlist xxxx,”, where xxxx refers to some set of numbers(1-7) and returns a list containing a set of the specified numbers. Here if the user wishes to specify that the appointment repeats on Sunday, Monday and Friday, they can replace xxxx with 126. The numbers coincide with the day of the week where Sunday is 1 and Saturday is 7.
If the f string is “dlist 1234516”, the return value will be a list of [1, 2, 3, 4, 5, 6]. Note that repeated values do not get added multiple times.

- The parameter is for the frequency string which specifies how often an appointment should be repeated per week.

isAfter BUG Report

Bug Report Title:
The isAfter method ignores the hour, minute, and seconds values of the given dates.
Reported by:
Jeremi Boston - thewermster@hotmail.com
Date reported:
Tuesday, March 28, 2017
Program (or component) name:
BORG Calendar
Configuration(s):
Mac OS X 10.12.2 Java 1.8.0_121 Runtime build 1.8.0_121-b13
Report type:
Design
Reproducibility:

Yes

Priority:

Minor Issue

Problem summary:

The isAfter method automatically sets the HOUR_OF_DAY, MINUTE, and SECOND values of the given dates d1 and d2 to zero.

Problem description:

The isAfter method in net.sf.borg.common.DateUtil does not take into account the hours, minutes, and seconds of the given dates when determining if one date is after the other. This is due to the fact that when the given dates, d1 and d2, are transformed to a GregorianCalendar the HOUR_OF_DAY, MINUTE, and SECOND values are set to zero for both dates. The expected functionality of an isAfter method is to determine whether one date is after the other. However, the Borg calendar method does not provide this functionality in the case that two dates are the same but one is after the other in either hours, minutes or seconds.

```
Date d3 = new Date(2017, 8, 15, 8, 30, 22);
Date d4 = new Date(2017, 8, 15, 9, 30, 22);
// Since both happen at the same day, but at different hours and
// the implementation ignores HMS, it will always be false
assertFalse(DateUtil.isAfter(d4, d3));
assertFalse(DateUtil.isAfter(d3, d4));
```

As can be seen from the both test case, the result of both isAfter calls on d3 and d4 will return false no matter the order. Hence the function does not take the HOUR_OF_DAY, MINUTE, and SECOND values into consideration.

This issue was discovered while running a mutation test using the PIT testing plugin for Eclipse.

Steps to Reproduce

1. Create a JUnit test suite for testing the method isAfter in net.sf.borg.common.DateUtil
2. Copy the above code into a test method.
3. Run the JUnit Test.
4. Notice that both calls to isAfter return false, as opposed to the first being true.

New or old bug:

New

Test Cases

MinuteStringTest

```
package eecs4313a3t1;

import net.sf.borg.common.DateUtil;
import static org.junit.Assert.*;
import org.junit.Test;



public class MinuteStringTest {

	@Test
	public void testMinuteStringBVT(){
		/*
		 * 1. Min-:    -1 minutes
		 * 2. Min:   0 minutes
		 * 3. Min+:  1 minute
		 * 4. Nominal:      1073741823 minutes
		 * 5. Max-:         2147483646
		 * 6. Max:          2147483647
		 * 7. Max+:         2147483648
		 */

		// 1. Min -
		assertEquals("-1 Minutes", DateUtil.minuteString(-1));

		// 2. Min
		assertEquals("0 Minutes", DateUtil.minuteString(0));

		// 3. Min +
		assertEquals("1 Minute", DateUtil.minuteString(1));

		// 4. Nominal
		assertEquals("17895697 Hours 3 Minutes",
DateUtil.minuteString(1073741823));

		// 5. Max-
		assertEquals("35791394 Hours 6 Minutes",
```

```
DateUtil.minuteString(2147483646));

            // 6. Max
            assertEquals("35791394 Hours 7 Minutes",
DateUtil.minuteString(2147483647));

            // 6. Max+
//          assertEquals("35791394 Hours 8 Minutes",
DateUtil.minuteString(2147483648));
            assertEquals("1 Hour", DateUtil.minuteString(60));

      }

      @Test
      public void testDataFlowMins(){

            // All Defs: Path AB
            // All Uses: Paths AB, ABC
            // All P-Uses / Some C-uses: Path AB
            // All C-uses / Some P-uses: Paths AB, ABC
            assertEquals("2 Minutes", DateUtil.minuteString(2));
      }

      @Test
      public void testDataFlowHours(){
            // All Defs: Path BCDEF
            assertEquals("2 Hours 2 Minutes", DateUtil.minuteString(122));

            // All Uses: Paths BCDEF, BCDEFG, BCDEFH, BCDEFHI, BCDEFHKMO
            assertEquals("1 Hour 2 Minutes", DateUtil.minuteString(62));
            assertEquals("2 Hours 2 Minutes", DateUtil.minuteString(122));
            assertEquals("1 Hour", DateUtil.minuteString(60));

            // All P-Uses / Some C-uses: Paths BCDEFHKMO
            assertEquals("1 Hour", DateUtil.minuteString(60));

            // All C-uses / Some P-uses: Paths BCDEFG, BCDEFHI
            assertEquals("1 Hour 5 Minutes", DateUtil.minuteString(65));
            assertEquals("2 Hours 12 Minutes", DateUtil.minuteString(132));
      }

      @Test
      public void testDataFlowMinsPast(){
            // All Defs: Path CDEFHK
            assertEquals("2 Minutes", DateUtil.minuteString(2));

            // All Uses: Paths CDEFHK, CDEFHKL, CDEFHKM, CDEFHKMN, CDEFHKMNOQ
```

```
            assertEquals("1 Hour 1 Minute", DateUtil.minuteString(61));
            assertEquals("0 Minutes", DateUtil.minuteString(0));

            // All P-Uses / Some C-uses: Paths CDEFHK, CDEFHKM
            assertEquals("1 Hour 3 Minutes", DateUtil.minuteString(63));
            assertEquals("2 Hours 1 Minute", DateUtil.minuteString(121));

            // All C-uses / Some P-uses: Paths CDEFHKL, CDEFHKMN, CDEFHKMNOQ
            assertEquals("1 Hour 5 Minutes", DateUtil.minuteString(65));
            assertEquals("2 Hours 12 Minutes", DateUtil.minuteString(132));
      }

      @Test
      public void testDataFlowMinutesString(){
            // All Defs: Path DEFHKL
            assertEquals("2 Minutes", DateUtil.minuteString(2));

            // All Uses: Paths DEFHKL, DEFHKMN , DEFHKMOP, DEFHKMOQ, DEFHKMOR,
DEFHKMORS, ST
            assertEquals("1 Hour 1 Minute", DateUtil.minuteString(61));
            assertEquals("0 Minutes", DateUtil.minuteString(0));
            assertEquals("1 Hour", DateUtil.minuteString(60));

            // All P-Uses / Some C-uses: Paths DEFHKMOR
            assertEquals("2 Hours 1 Minute", DateUtil.minuteString(121));
            assertEquals("1 Hour", DateUtil.minuteString(60));
            assertEquals("0 Minutes", DateUtil.minuteString(0));

            // All C-uses / Some P-uses: Paths DEFHKL, DEFHKMN , DEFHKMOP,
DEFHKMOQ, DEFHKMORS, ST
            assertEquals("1 Hour 5 Minutes", DateUtil.minuteString(65));
            assertEquals("2 Hours 1 Minute", DateUtil.minuteString(121));
            assertEquals("2 Minutes", DateUtil.minuteString(2));
            assertEquals("1 Hour", DateUtil.minuteString(60));

      }

      @Test
      public void testDataFlowHoursString(){

            // All Defs: Path EFG
            assertEquals("3 Hours", DateUtil.minuteString(180));

            // All Uses: Paths EFG, EFHI, EFHJ, EFHKMOR, EFHKMORT
            assertEquals("1 Hour 1 Minute", DateUtil.minuteString(61));
            assertEquals("2 Hours 1 Minute", DateUtil.minuteString(121));
            assertEquals("0 Minutes", DateUtil.minuteString(0));
```

```
            assertEquals("1 Hour", DateUtil.minuteString(60));

            // All P-Uses / Some C-uses: Paths  EFHKMOR
            assertEquals("2 Hours 1 Minute", DateUtil.minuteString(121));
            assertEquals("1 Hour", DateUtil.minuteString(60));
            assertEquals("0 Minutes", DateUtil.minuteString(0));

            // All C-uses / Some P-uses: Paths EFG, EFHI, EFHJ, EFHKMORT
            assertEquals("1 Hour 5 Minutes", DateUtil.minuteString(65));
            assertEquals("2 Hours 1 Minute", DateUtil.minuteString(121));
            assertEquals("2 Minutes", DateUtil.minuteString(2));

      }

}
```

IsAfterTest

```
package eecs4313a3t1;

import static org.junit.Assert.*;

import java.util.Date;

import org.junit.Test;

import net.sf.borg.common.DateUtil;

public class isAfterTest {

      @Test
      public void testIsAfter() {
            // test using equivalence class testing
            // variation: weak robust
            // which means we will test incorrect dates

            // ranges:
            // first date is not a real date | second date is after it - done
            // first date is not a real date | second date is not after it - done
            // first date is a real date and it is before second date - done
            // first date is a real date and it is after second date - done
            // first date is after the second date and second date is not a real
            //           date - done

            Date d1 = new Date(2017, 8, 15);
```

```
            Date d2 = new Date(2017, 8, 16);

            assertFalse(DateUtil.isAfter(d1, d2));
            assertTrue(DateUtil.isAfter(d2, d1));

            // february 31, 2017

            Date invalid_date = new Date(2017, 2, 31);
            Date pre_invalid_date = new Date(2017, 1, 1);

            assertFalse(DateUtil.isAfter(invalid_date, d2));
            assertTrue(DateUtil.isAfter(d2, invalid_date));

            assertFalse(DateUtil.isAfter(pre_invalid_date, invalid_date));
            assertTrue(DateUtil.isAfter(invalid_date, pre_invalid_date));

            Date feb = new Date(2017, 2, 31);
            Date march = new Date(2017, 3, 1);

            assertTrue(DateUtil.isAfter(march, feb));
            assertFalse(DateUtil.isAfter(feb, march));

            Date d3 = new Date(2017, 8, 15, 8, 30, 22);
            Date d4 = new Date(2017, 8, 15, 9, 30, 22);
            // Since both happen at the same day, but at different hours and the
implementatino ignores HMS, it will always be false
            assertFalse(DateUtil.isAfter(d4, d3));
            assertFalse(DateUtil.isAfter(d3, d4));

      }
}
```

DayListTest

```
package eecs4313a3t1;

import net.sf.borg.model.Repeat;

import static org.junit.Assert.*;
import eecs4313a2b.AllPermutations;

import java.util.ArrayList;
import java.util.List;

import org.junit.Test;
```

```java
public class DayListTest {

	@Test
	public void testDayList(){

		/*
		 * Conditions:
		 * C1: Input is empty
		 * C2: Input has a comma seperator between "dlist" and the day values
		 * C3: Input doesn't have a comma seperator between "dlist" and the
day values
		 * C4: Input has "dlist" but no numbers afterwards
		 * C5: Input contains a valid sequence of numbers
		 * C6: Non number after "dlist"
		 * C7: Non number in between numbers
		 *
		 * Actions:
		 * A1: Return empty list
		 * A2: Return a list of the extracted numbers
		 *
		 *
		 * C1 -> A1
		 * C2 -> A2
		 * C3 -> A2
		 * C4 -> A1
		 * C5 -> A2
		 * C6 -> A1
		 * C7 -> A2
		 */


		List<Integer> sq = new ArrayList<Integer>();

		// C1 -> A1
		assertEquals(sq, Repeat.getDaylist(""));

		//C4 -> A1
		assertEquals(sq, Repeat.getDaylist("dlist, "));

		//C6 -> A1
		assertEquals(sq, Repeat.getDaylist("dlist, agc"));

		//C2 -> A2
		sq.add(1);
		assertEquals(sq, Repeat.getDaylist("dlist, 1"));
```

```
            //C3 -> A2
            sq.add(2);
            assertEquals(sq, Repeat.getDaylist("dlist 12"));

            //C7 -> A2
            assertEquals(sq, Repeat.getDaylist("dlist, 1agc2"));


            // C5 => A2

            List<String> lists = AllPermutations.makeLists("", 7);

            for (int i = 0; i < lists.size(); i++){
                  String f = "dlist ";
                  List<Integer> list =
AllPermutations.createListFromString(lists.get(i));
                  assertEquals(list, Repeat.getDaylist(f + lists.get(i)));
            }

            // New Test Because of Mutation Testing
            List<Integer> l = new ArrayList<Integer>();
            l.add(1);
            assertEquals(l, Repeat.getDaylist("dlist2 1"));
            assertEquals(l, Repeat.getDaylist("dlist2  1,"));
      }

}
```