בדיקת תוכנה תרגיל 2:

מגישים: ג'יי טננבאום - 323639476, שלי נגר - 308367333

שאלה 1:

א.

```
-------------------------------
Results for class Count
-------------------------------
Live mutants: 32
Killed mutants: 37
Mutation Score: 53.0
Writing to file...
```

כפי שניתן לראות, ה- MUTATION SCORE עבור אוסף הבדיקות שניתן לנו הוא 53%.

ב.

אכן ניתן להוריד בדיקות מהאוסף הנתון ולשמור על אותו ציון MUTATION SCORE. ניתן להוריד את הבדיקות 2,4,5, ולשמור על אותו ציון MUTATION, כפי שניתן לראות:

```
import junit.framework.TestCase;

public class CountTest extends TestCase {
    public void test1() {
        Count c = new Count(new int[]{1,-1,-1});
        assertEquals("Counted value", c.count(), 1);
    }

//  public void test2() {
//      Count c = new Count(new int[]{1});
//      assertEquals("Counted value", c.count(), 1);
//  }

    public void test3() {
        Count c = new Count(new int[]{-1, 1});
        assertEquals("Counted value", c.count(), 1);
    }

//  public void test4() {
//      Count c = new Count(new int[]{1, 2});
//      assertEquals("Counted value", c.count(), 2);
//  }

//  public void test5() {
//      Count c = new Count(new int[]{});
//      assertEquals("Counted value", c.count(), 0);
//  }
}
```

```
Problems  @ Javadoc  Declaration  Console  Mutants and Results
TestConfig [MuClipse: Tests] C:\Program Files\Java\jre7\bin\javaw.exe (1 13:59:42 2014 במאי)
Active: 0
--------------------
-------------------------------
Results for class Count
-------------------------------
Live mutants: 32
Killed mutants: 37
Mutation Score: 53.0
Writing to file...
```

לא ניתן להסיק מכך דבר אודות יחס ההכלה בין MUTATION TESTING לבין PRIME PATHS COVERAGE. זאת כיוון שראינו שעבור אוסף הבדיקות test1-test5 יש כיסוי PRIME PATHS, אך אין כיסוי על MUTATION TESTING, כיוון שהניקוד היה 53, שהוא פחות מהמקסימלי שהשגנו בעקבות הוספת בדיקות בסעיף ג'. לכן נסיק כי כיסוי על MUTATION TESTING לא מוכל בכיסוי על PRIME PATHS. מצד שני, הבדיקות שלקחנו בסעיף ג', אשר כיסו MUTATION TESTING, לא היוו כיסוי PRIME PATHS (כדי להרכיב את הבדיקות של סעיף ג', הורדנו את המיותרים מהבדיקות הנתונות, והוספנו משלנו, כאשר דאגנו לא לעשות כיסוי PRIME PATHS), ולכן גם כיסוי על PRIME PATHS לא מוכל בכיסוי על MUTATION TESTING. לכן לא ניתן להסיק דבר אודות יחס ההכלה ביניהם.

ד.

הציון הגבוה ביותר שהגענו אליו הוא 91:

```
-------------------------------
Results for class Count
-------------------------------
Live mutants: 6
Killed mutants: 63
Mutation Score: 91.0
Writing to file...
```

לא ניתן להגיע ל-100%, כיוון שישנן 6 מוטציות שלא ניתן להרוג, כיוון שהן שקולות לתכנית המקורית:

.1

```
if (x++ == -1 && i > 0 && numbers[i-1] == -1) {
        break;
}
```

מוטנט זה שקול לתכנית המקורית, כיוון שאופרטור ++ לאחר המשתנה מחזיר את הערך הישן לפני ההעלאה ב-1. כיוון שבכל איטרציה אנו מגדירים את x מחדש, ולא משתמשים בו אף פעם לאחר ה-if הנ"ל, ההעלאה ב-1 לאחר החזרת x לא משנה את זרימת התכנית.

.2

```
if (x-- == -1 && i > 0 && numbers[i-1] == -1) {
        break;
}
```

מוטנט זה שקול לתכנית המקורית, כיוון שאופרטור -- לאחר המשתנה מחזיר את הערך הישן לפני ההורדה ב-1. כיוון שבכל איטרציה אנו מגדירים את x מחדש, ולא משתמשים בו אף פעם לאחר ה-if הנ"ל, ההורדה ב-1 לאחר החזרת x לא משנה את זרימת התכנית.

.3

```
return count++;
```

מוטנט זה שקול לתכנית המקורית, כיוון שאופרטור ++ לאחר המשתנה מחזיר את הערך הישן לפני ההעלאה ב-1. כיוון שהשורה ששינינו היא ב-return, המתודה לא ממשיכה לרוץ לאחר שינוי המשתנה count, ולכן המתודה Count לאחר מוטציה זו תחזיר את אותו הערך כמו המקורי.

.4

```
return count--;
```

מוטנט זה שקול לתכנית המקורית, כיוון שאופרטור -- לאחר המשתנה מחזיר את הערך הישן לפני ההורדה ב-1. כיוון שהשורה ששינינו היא ב-return, המתודה לא ממשיכה לרוץ לאחר שינוי המשתנה count, ולכן המתודה Count לאחר מוטציה זו תחזיר את אותו הערך כמו המקורי.

.5

```
if (x == -1 && i != 0 && numbers[i-1] == -1) {
        break;
}
```

מוטנט זה שקול לתכנית המקורית, כיוון ש-i תמיד אי- שלילי (מתחיל ב-0, ובכל איטרציה מועלה ב-1), ולכן לבדוק אם i>0 שקול ללבדוק אם i!=0.

.6

```
if (x >= -1 && i > 0 && numbers[i-1] == -1) {
        break;
}
```

מוטנט זה שקול לתכנית המקורית, כיוון שכדי להגיע לתנאי if זה, צריך לעבור דרך if(x<0), ולכן בהשוואה זו רק לוקחים x שהוא שלילי. x תמיד נלקח להיות ערך של איבר כלשהו במערך הנתון של int, ולכן x גם שלם. אבל עבור x שלילי שלם מתקיים x==-1 אם"ם x>=-1, ולכן המוטציה שקולה למקורי.

<u>שאלה 2:</u>

<u>א</u>. נרצה להראות שהנחת הבסיס מתקיימת. לשם כך נסתכל על המוטציה הבאה:

```
int f( int p, int q){

        Integer x = null;

        if(q<=0) x = p*q;

        if(p>=0 && q>0) x = p/q;

        return x+1;

}
```

כדי להוכיח שהנחת הבסיס מתקיימת, יש להראות שכל בדיקה שהורגת את המוטנט תחשוף את הבאג בתכנית המקורית (ש-x לא אותחל כראוי). ואכן, כדי להרוג את המוטנט, יש להריץ בדיקה עם קלט כך שהמוטציה תתנהג באופן שונה מהתכנית המקורית. זה מתקיים בדיוק כאשר q=0, ועבור p כלשהו (כי אם q=0, אז במוטציה נכנסים פנימה ב-if הראשון, ו-x מתאפס, ומחזירים 1, וזאת בניגוד לתכנית המקורית שתיכשל עקב נסיון העלאה של null ב-1, כיוון שלא תיכנס לאף if. אם q!=0 אז ההתנהגות של הפונקציה המקורית ושל המוטציה תהיה זהה, כיוון שהשינוי הוא בתנאי if, והרי אם q!=0 אין הבדל). לכן סה"כ אפשר להרוג מוטנט אם ורק אם מריצים בדיקה עם q=0, אבל ראינו כי בבדיקה עם קלט כזה הפונקציה המקורית נופלת, כיוון שלא נכנסים לאף IF, ולכן x נשאר null, ומנסים להעלות אותו, וזו שגיאה. לכן בהרצה על התכנית המקורית הבאג ש-x לא מאותחל כראוי נחשף, והרי לנו הנחת הבסיס של MUTATION TESTING.

ב. נרצה להראות שהנחת הבסיס לא מתקיימת. לשם כך נרצה להראות שלכל מוטנט שאפשר להרוג יש בדיקה שהורגת את המוטנט אבל לא תחשוף את הבאג בתכנית המקורית (שקוראים ל-sqr במקום ל- sqrt). נשים לב שבדיקה כזו עבור כל מוטנט כנ"ל היא הרצה עם קלט x=1, ופלט מצופה x=2. נראה שעבור המוטנטים המסורתיים זה מתקיים.

המוטנטים המסורתיים הרלוונטיים לתכנית זו הם אופרטורי ההזזה והאופרטורים האריתמטיים.

אופרטורי ההזזה:

המוטנט יכול להיות:

```
return (sqr(abs(x))>>2);

return (sqr(abs(x))>>>2);

return (sqr(abs(x))<<2);

return (sqr(abs(x))<<<2);
```

כאן הבאג לא ייתפס, אף על פי שהמוטנטים כן ייתפסו כיוון ש:

```
sqr(abs(1))= sqrt(abs(1))
```

ולכן כאשר מזיזים את התוצאה, תמיד מזיזים את אותו הערך, ולכן התוצאה נשמרת.

אופרטורים אריתמטיים:

המוטנטים האפשריים הם:

```
1. return (sqr(abs(x))+2);

2. return (sqr(abs(x))-2);
```

3. return (sqr(abs(x))/2);

4. return (sqr(abs(x))%2);

בדומה לקודם עבור אופרטור ההזזה, השינוי הוא תמיד על:

sqr(abs(1))= sqrt(abs(1))

ולכן הבאג לא נתפס.

5. return (sqr(abs(++x))*2);
6. return (sqr(abs(--x))*2);

כאן הבאג לא נתפס מאותה סיבה כמו קודם.