گزارش تکلیف دوم درس تست نرم افزار

نسترن عشوری – 810101225

لینک گیت : https://github.com/nastaran98/Software-testing---CA2

برای اجرا شدن ابتدا فایل pom.xml را تغییر میدهیم. سپس برای کلاس Gedcom_ServiceTest تست های زیر را مینویسم.

1) testBirthBeforeDeath : این تابع چک میکند که اگر در سیستم شخصی ثبت شود که تولدش بعد از مرگش باشد، سیستم حتما پیغام خطا داده و آن را در فایل چاپ کند.

```
@Test
void testBirthBeforeDeath() throws IOException, ParseException {
    Individual ind = new Individual("I1");
    ind.setName("Nastaran Ashoori");
    ind.setBirth("01/01/1990");
    ind.setDeath("01/01/1985");
    Gedcom_Service.individuals.put(ind.getId(), ind);

    Gedcom_Service.birthBeforeDeath(Gedcom_Service.individuals);

    String expectedOutput = "ERROR:INDIVIDUAL: User Story US03: Birth Before Death \nIndividual: I1 - Nastaran Ashoori was born after death\nDOB: 01/01/1990 DOD: 01/01
    assertTrue(outputStreamCaptor.toString().contains(expectedOutput));
}
```

2) testMarriageBeforeDivorce: این تابع چک میکند که اگر تاریخ طلاق 2 شخص قبل از تاریخ ازدواجشان باشد سیستم حتما پیغام خطا داده و آن را در فایل چاپ کند.

```
@Test
void testMarriageBeforeDivorce() throws IOException, ParseException {
    // Set up a family with marriage date after the divorce date
    Family fam = new Family( id: "F1");
    fam.setMarriage("01/01/2000");
    fam.setDivorce("01/01/1990");
    fam.setHusb("I1");
    fam.setWife("I2");
    Gedcom_Service.families.put(fam.getId(), fam);

    // Set up the individuals involved in the family
    Individual husb = new Individual("I1");
    husb.setName("Ali");
    Individual wife = new Individual("I2");
    wife.setName("Fatemeh");

    Gedcom_Service.individuals.put(husb.getId(), husb);
    Gedcom_Service.individuals.put(wife.getId(), wife);

    Gedcom_Service.Marriagebeforedivorce(Gedcom_Service.individuals, Gedcom_Service.families);

    String expectedOutput = "ERROR:FAMILY: User Story US04: Marriage Before Divorce \nFamily: F1\nIndividual: I1: AliI2: Fatemeh marriage date is before divorce dat
    assertTrue(outputStreamCaptor.toString().contains(expectedOutput));
```

3) testBirthBeforeMarriageOfParent: این تابع چک میکند اگر تاریخ تولد فرزندی قبل از ازدواج پدر مادرش باشد ، سیستم حتما پیغام خطا داده و آن را در فایل چاپ کند.

```
@Test
void testBirthBeforeMarriageOfParent() throws IOException, ParseException {

    Individual husb = new Individual("I1");
    husb.setName("Ali");
    Individual wife = new Individual("I2");
    wife.setName("Fatemeh");

    Family fam = new Family( id: "F1");
    fam.setMarriage("01/01/2000");
    Individual child = new Individual("I3");
    child.setBirth("01/01/1990");
    ArrayList<String> children = new ArrayList<>();
    children.add(child.getId());
    fam.setChild(children);
    Gedcom_Service.individuals.put(child.getId(), child);

    Gedcom_Service.individuals.put(husb.getId(), husb);
    Gedcom_Service.individuals.put(wife.getId(), wife);

    Gedcom_Service.families.put(fam.getId(), fam);

    Gedcom_Service.birthbeforemarriageofparent(Gedcom_Service.individuals, Gedcom_Service.families);

    String expectedOutput = "ERROR: User Story US08: Birth Before Marriage Date \nFamily ID: F1\nIndividual: I3: null Has been born before parents' marriage\nDOB: 01
    assertTrue(outputStreamCaptor.toString().contains(expectedOutput));
}
```

4) testMaleLastName: این تابع چک میکند نام خانوادگی تمامی مردان یک خانواده یکسان باشد و در غیر این صورت سیستم حتما پیغام خطا داده و آن را در فایل چاپ کند.

```
@Test
void testMaleLastName() throws IOException, ParseException {
    Family fam = new Family( id: "F1");
    Individual child1 = new Individual("I1");
    child1.setName("John Doe");
    child1.setSex("M");

    Individual child2 = new Individual("I2");
    child2.setName("Mike Smith");
    child2.setSex("M");

    ArrayList<String> children = new ArrayList<>();
    children.add(child1.getId());
    children.add(child2.getId());
    fam.setChild(children);

    Gedcom_Service.individuals.put(child1.getId(), child1);
    Gedcom_Service.individuals.put(child2.getId(), child2);

    Gedcom_Service.families.put(fam.getId(), fam);

    Gedcom_Service.Malelastname(Gedcom_Service.families);

    String expectedOutput = "ERROR: User Story US16: Male last name \nFamily ID: F1 family members don't have the same last name \n\n";
    assertTrue(outputStreamCaptor.toString().contains(expectedOutput));
}
```

5) testAuntsAndUnclesName: این تابع چک میکند ازدواجی بین فرزندان یک خانواده و عمه یا عموهایشان صورت نگرفته باشد و در غیر این صورت سیستم حتما پیغام خطا داده و آن را در فایل چاپ کند.

```
void testAuntsAndUnclesName() throws IOException, ParseException {
    childFam.setHusb(child.getId());
    childFam.setWife(aunt.getId());
    Gedcom_Service.families.put(childFam.getId(), childFam);

    // Populate individuals map
    Gedcom_Service.individuals.put(father.getId(), father);
    Gedcom_Service.individuals.put(mother.getId(), mother);
    Gedcom_Service.individuals.put(uncle.getId(), uncle);
    Gedcom_Service.individuals.put(aunt.getId(), aunt);
    Gedcom_Service.individuals.put(child.getId(), child);

    // Set child relationships
    father.setChildOf(grandParentFam.getId());
    uncle.setChildOf(grandParentFam.getId());
    aunt.setChildOf(grandParentFam.getId());
    child.setChildOf(parentFam.getId());

    Gedcom_Service.AuntsandUnclesname(Gedcom_Service.families);

    String expectedOutput = "ERROR: User Story US20: Aunts and Uncles\nIndividual: I5 - Child Doe is married to either their aunt or uncle I4 - Aunt Doe\n\n";
    assertTrue(outputStreamCaptor.toString().contains(expectedOutput));
}
```

6) testUniqueFamilyNameBySpouses : این تابع چک میکند که خانواده های تکراری ثبت نشده باشد و در غیر این صورت سیستم حتما پیغام خطا داده و آن را در فایل چاپ کند.(خانواده تکراری یعنی 2 شخص یکسان در یک روز یکسان ازدواج کرده باشند اما 2 رکورد برایشان ثبت شده باشد.)

```java
@Test
void testUniqueFamilyNameBySpouses() throws IOException, ParseException {
    Family fam1 = new Family( id: "F1");
    fam1.setHusb("I1");
    fam1.setWife("I2");
    fam1.setMarriage("01/01/2000");

    Family fam2 = new Family( id: "F2");
    fam2.setHusb("I1");
    fam2.setWife("I2");
    fam2.setMarriage("01/01/2000");

    Gedcom_Service.families.put(fam1.getId(), fam1);
    Gedcom_Service.families.put(fam2.getId(), fam2);

    Individual husb = new Individual("I1");
    husb.setName("Ali");
    Individual wife = new Individual("I2");
    wife.setName("Fatemeh");

    Gedcom_Service.individuals.put(husb.getId(), husb);
    Gedcom_Service.individuals.put(wife.getId(), wife);

    Gedcom_Service.uniqueFamilynameBySpouses(Gedcom_Service.individuals, Gedcom_Service.families);

    String expectedOutput = "ERROR: User Story US24: Unique Families By Spouse :\nF1: Husband Name: Ali, Wife Name: Fatemeh and F2: Husband Name: Ali, Wife Name:
    assertTrue(outputStreamCaptor.toString().contains(expectedOutput));
}
```

7) هم چنین دو تست نیز برای چک کردن ایجاد فایل خروجی نا معتبر و خواندن فایل نامعتبر ایجاد شده است.

در نهایت pitest را اجرا میکنیم. نتیجه کلی به شرح زیر است.

Pit Test Coverage Report

Package Summary

edu.stevens.ssw555

Number of Classes	Line Coverage		Mutation Coverage		Test Strength	
1	54%	160/298	40%	52/131	85%	52/61

Breakdown by Class

Name	Line Coverage		Mutation Coverage		Test Strength	
Gedcom_Service.java	54%	160/298	40%	52/131	85%	52/61

Report generated by PIT 1.16.1

فایل Gedcom_serviceTest.java.html ایجاد شده است که گزارش کاملی از mutation ها ارائه میکند. حال به تحلیل موارد خواسته شده میپردازیم.

1) جهش های نا معتبر: این جهش مثلا باعث ایجاد ارور syntax میشود یا یک فراخوانی تابع را انجام نمیدهد. از این نوع جهش نداشته ایم.

2) معادل کد اصلی : تست زیر نمونه ای از جهش های معادل کد اصلی است. اگر مثلا به جای assert equal از توابع مشابه برای سنجش تساوی استفاده کنیم.

```
@Test
public void testEquivalentMutationEqualityCheck() {
    Individual indi = new Individual("I1");
    indi.setBirth("01/01/2000");
    indi.setDeath("01/01/2020");

    assertEquals("01/01/2000", indi.getBirth());
    assertEquals("01/01/2020", indi.getDeath());
    // Equivalent mutation: if (indi.getBirth().equals(indi.getBirth()))
    // This does not change the logic but it's equivalent
}
```

از این نوع جهش در نتیجه pitest ایجاد نشده است. در واقع این نوع جهش ها در کد ما کاور نشده است.

3) جهش های معتبر ولی نا مفید که توسط اکثر تست ها قابل تشخیص است.

```
@Test
public void testValidButIneffectiveMutationArithmetic() {
    int result = 2 + 2;
    assertEquals(4, result);
    // Mutation: change to 2 + 3
    // This will be easily caught by the test
}
```

مثلا در جهش های ایجاد شده جهش زیر را داریم که معتبر ولی نامفید است. این جهش ها منطق کلی برنامه را زیر سوال نمیبرند. ولی به راحتی با اعمال این جهش ها (مثلا حذف print) تست fail میشود و توسط هر تستی میتوان آن را به دست آورد.

```
                    System.out.println("Please Enter Output File Path: ");
public static void createOutputFile(BufferedReader bufferRead) throws IOException {
1. createOutputFile : removed call to java/io/PrintStream::println → SURVIVED
                String fp = bufferRead.readLine();
```

تغییر در توابع پرینت اورهای نهایی نیز جزو همین دسته است. این جهش ها kill میشود ولی مفید نیستند.

4) جهش های مفید: این جهش ها killed شده هستند ولی تعداد کمی از تست ها آن را تشخیص میدهند. جهش ایجاد شده مفید در واقع نقاط کلیدی تابع را تست میکند و با تغییر آن منطق کلی برنامه زیر سوال میرود. نقیض کردن شروط از این نوع جهش هاست.

```
282 1               while (famEntries.hasNext()) {
283                     Map.Entry<String, Family> famEntry = famEntries.next();
284                     Family fam = famEntry.getValue();
285                     try {
286                             marriageDate = sdf.parse(fam.getMarriage());
287 1                           if (fam.getChild() != null) {
288 2                                   for (int i = 0; i < fam.getChild().size(); i++) {
289                                             Individual indi = indMap.get(fam.getChild().g
290                                             birthDate = sdf.parse(indi.getBirth());
291 1                                           if (birthDate.before(marriageDate)) {
```

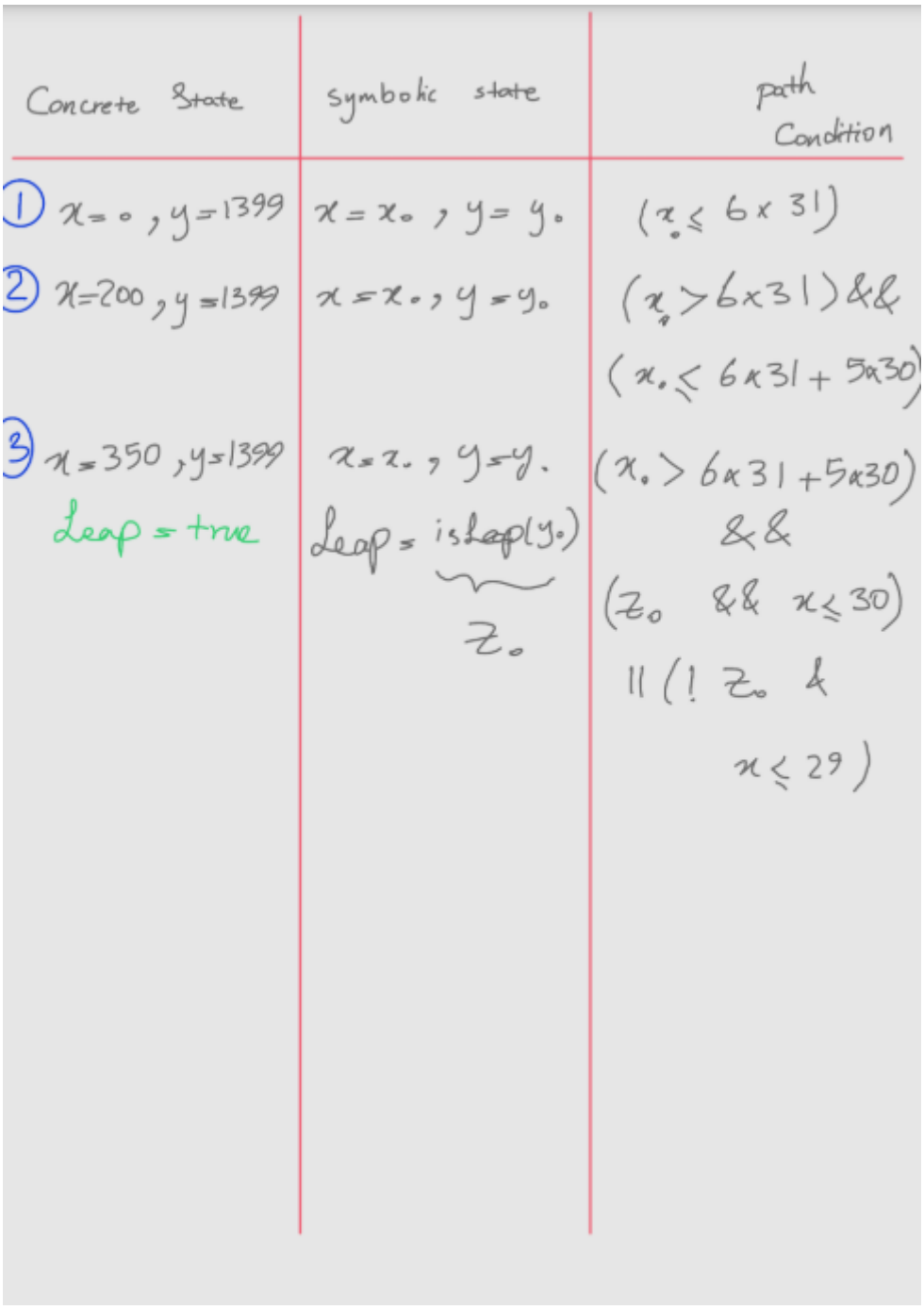
```
negated conditional → KILLED
negated conditional → KILLED
negated conditional → KILLED
changed conditional boundary → KILLED
negated conditional → KILLED
```

5) جهش های زنده

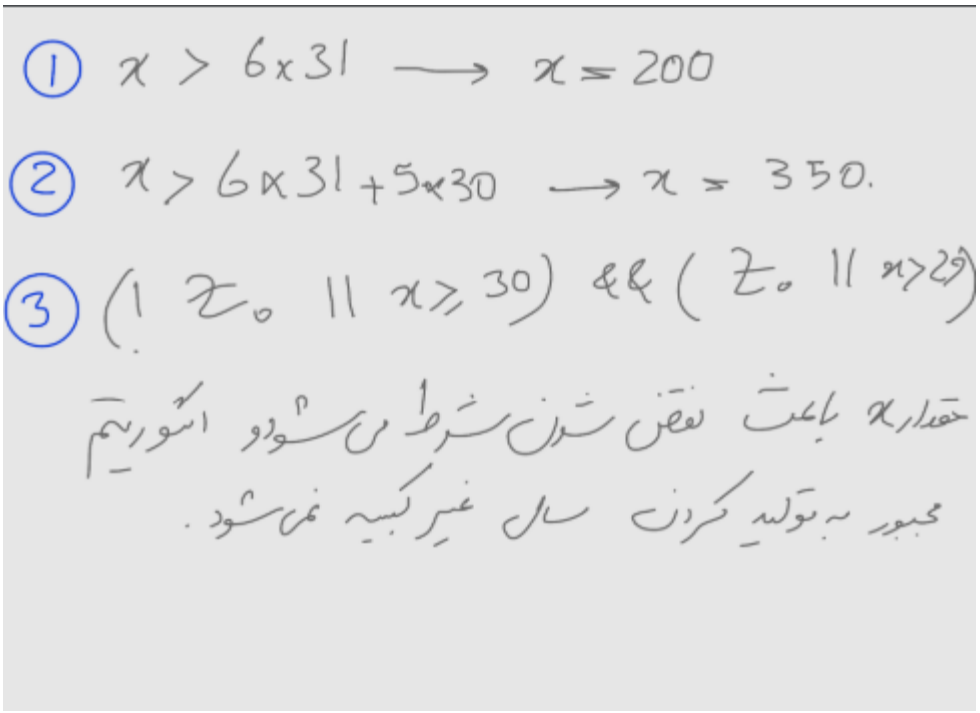
جهش هایی که در رده survive قرار دارند از این دسته هستند. در اینجا جهش صورت گرفته ولی تست ما متوجه تغییر نشده و fail نشده است.

6) جهش های تکراری: از این نوع جهش در جهش ها نداشتیم.

سوال 2) الف)

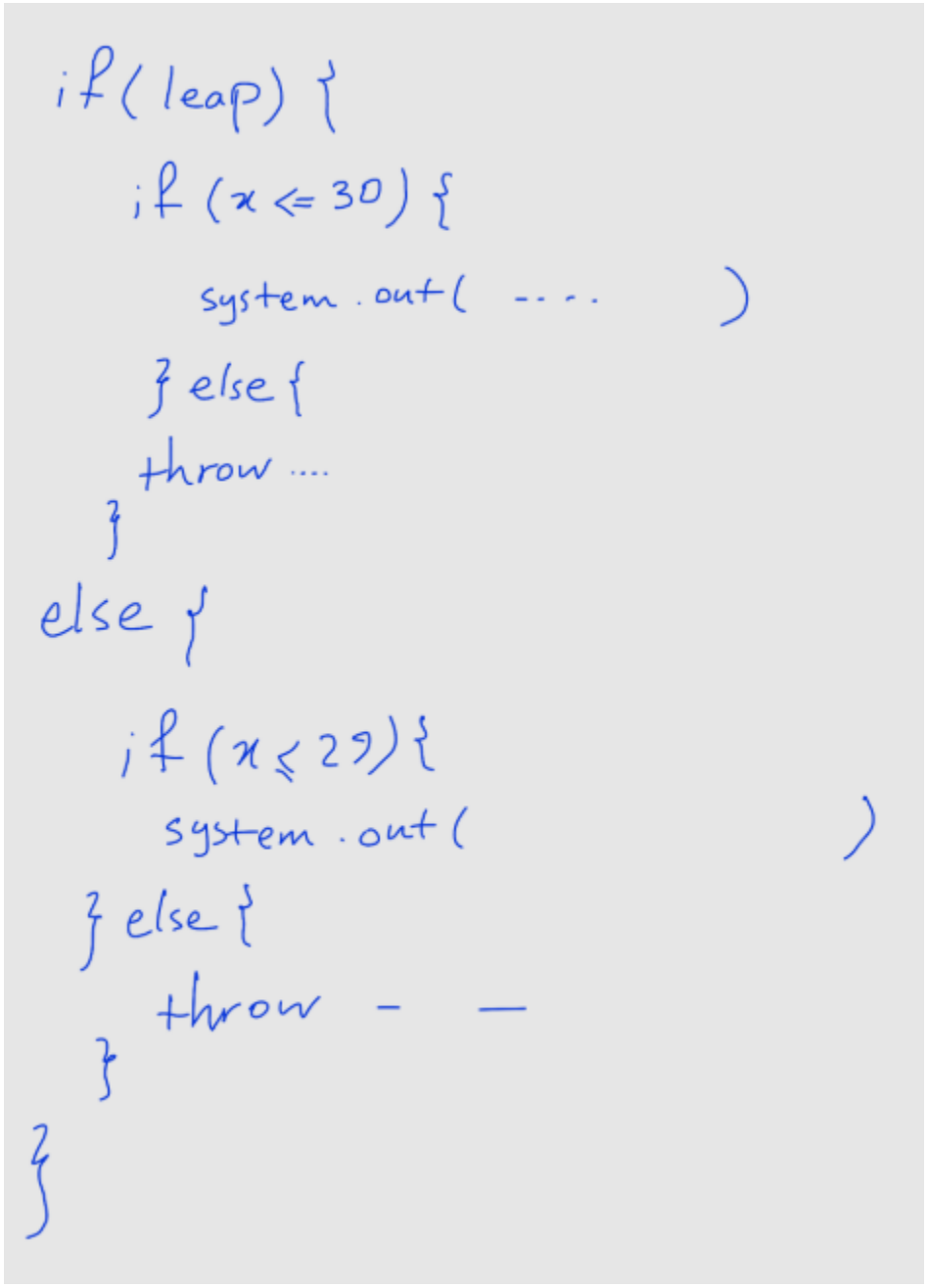


توضیحات نقیض کردن هر مرحله در تصویر بعد آمده است.



در نهایت برای catch کردن ارور باید مقدار 370 را به عنوان x بدهیم.

ب)


if (leap) {
if (x <= 30) {
system . out(....)
} else {
throw ....
}
else {
if (x <= 29) {
system . out( )
} else {
throw - —
}
}

Concrete State	symbolic state	path Condition
① $x=0, y=1399$	$x=x_0, y=y_0$	$(x_0 \leq 6\times 31)$
② $x=200, y=1399$	$x=x_0, y=y_0$	$(x_0 > 6\times 31)$ && $(x_0 \leq 6\times 31 + 5\times 30)$
③ $x=350, y=1399$ Leap = true	$x=x_0, y=y_0$ Leap = isLeap(y_0) z_0	leap

در مرحله ۳ برای نقض کردن شرط مجبوریم سال غیر کبیسه تولید کنیم.