Property based testing

by Urs Blum

Content

-  What?
-  Lazy
-  Stupid
-   Lazy and Stupid
-  Evil
- 

Real
- 

Good, bad, ugly

What?
Property based testing

- Normal ‘example’ based testing:
  -> Write enough tests to get sufficient trust

- Property based testing:
  A property shows that our code satisfies certain general properties (invariants) for a class of inputs (precondition)
  -> Look for properties instead of examples and test them

Lazy ApprovalsExample

ALCOR academy

```
public class ApprovalsExample {
   public static String mergeNameAndAge(String name, int age) {
      String title = "";
      if (age <= 18) {
         title = "Junior";
      }
      if (age > 18 && age <= 60) {
         title = "Mister";
      }
      if (age > 60) {
         title = "Senior";
      }
      return title + " " + name + ", " + age + " years";
   }
}
```

```
@Test
void forMergingNameAndAge() {
   String[] names = new String[]{"alex"};
   Integer[] ages = new Integer[]{15, 18, 26, 60, 63};

   CombinationApprovals.verifyAllCombinations(
         (name, age) -> {
            String result = ApprovalsExample.mergeNameAndAge(name, age);
            return result;
         }
         , names, ages);
}
```

Lazy
Look for properties

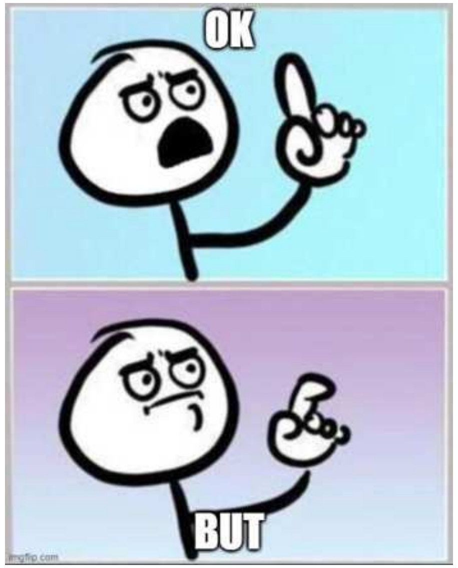
```
@Test
void printSomeExamples() {
    System.out.println(ApprovalsExample.mergeNameAndAge("Homer", 39));
    System.out.println(ApprovalsExample.mergeNameAndAge("Marge", 36));
    System.out.println(ApprovalsExample.mergeNameAndAge("Bart", 10));
    System.out.println(ApprovalsExample.mergeNameAndAge("Lisa", 8));
    System.out.println(ApprovalsExample.mergeNameAndAge("Maggie", 1));
}
```

Mister Homer, 39 years
Mister Marge, 36 years
Junior Bart, 10 years
Junior Lisa, 8 years
Junior Maggie, 1 years

Stupid
Test property

```
@Property
void theNameAndAgeMergeReturnsStringContainingInputParameters(
        @ForAll String arbitraryName,
        @ForAll int arbitraryAge) {
    String expectedPostfix = arbitraryName + ", " + arbitraryAge + " years";

    String result = ApprovalsExample.mergeNameAndAge(arbitraryName, arbitraryAge);
    System.out.println(result);

    assertTrue(result.endsWith(expectedPostfix));
}
```

Junior 躅→๑垸 ㎳㉚ 斝䬵 查, -22 years
Junior , -10828 years
Junior 焱 ⍰ , -207 years
Senior 昡尨ꝏ譔澒疛ђ갵 顺垈 뫔맖解瘀 鉶恩ძ 뜨 扏ц盛ɔᵓ
□惋爈 쎏쇲 飆濼宂 늵ᄙ:ː 鏖 韗剡䁣맷 ৼ 罳 透愦뗮⍰ 繽頤蝵뮡 谂덕芬 ⍰
眾弗삶腥 礵 喱跐샘뚴醇 網ψ쁴什斟 쬠뾟, 228279953 years
Senior 踁鶚 毟 撥敷ო, 2710 years
Junior ῃHg 쳀 況쒿⍰ᴿ 鮭遡뎩썽 羸랸鹽 啫븸, 5 years
Junior , -2 years
Junior 纾谧 あꝏ , -1339 years
Junior 豎ㅁ継업⁝ 曤, -2578 years
Senior , 104336 years
Senior ﬀ巚步□⍰挎 ҁ뵉彯쓲 쉤, 203 years
Junior 儒 睵 ঢ়, -388912 years
Junior , 0 years
Senior 趦¹幀헥, 2127 years
Junior , -6 years
…



Stupid Test property

```
@Property
void theNameAndAgeMergeReturnsStringContainingInputParameters(
      @ForAll @StringLength(min=2, max=25) @AlphaChars String arbitraryName,
      @ForAll @IntRange(min=0, max=123) int arbitraryAge) {
   String expectedPostfix = arbitraryName + ", " + arbitraryAge + " years";

   String result = ApprovalsExample.mergeNameAndAge(arbitraryName, arbitraryAge);
   System.out.println(result);

   assertTrue(result.endsWith(expectedPostfix));
}
```


```
Junior mgznrqqAtAa, 0 years
Mister AA, 51 years
Junior aa, 6 years
Junior HAYvzmJlyJuGKAEeXuZyYcPZ, 1 years
Mister QjQazaibAAzzataAZ, 21 years
Senior AuIhzTAAi, 82 years
Mister zaFaFzHr, 31 years
Senior ZKRbaZ, 99 years
Junior PataljTAkaaiaXzdQaLAp, 3 years
Senior zz, 102 years
Junior AcuBrzDyibgaMW, 8 years
Junior zWaaDzCBBzcQtzRw, 14 years
Mister lexTacbbwagIqdebrA, 29 years
Junior DaPzZvzxAFzFBjzZzRaTUHXC, 18 years
Junior zjWzYi, 9 years
Mister ZjzApZzHtjj, 33 years
Mister lzWxIazaAzURPZW, 49 years
Junior ydX, 4 years
Junior VInaKDaNVdaHAzZAAAauwz, 17 years
Senior ZCdZpMqzaXZEpZzLnzKzqipA, 81 years
Senior zAzosqxNzTHvQRawBGqazLkh, 71 years
Junior dOaDZaIaZaYHEh, 17 years
Senior zCAuabwAaX, 104 years
...
```

Lazy and stupid
Use a failing test to find another property

```
@Property
void theNameAndAgeMergeReturnsStringWithTitleAndInputParameters(
      @ForAll @StringLength(min=2, max=25) @AlphaChars String arbitraryName,
      @ForAll @IntRange(min=0, max=123) int arbitraryAge) {

   String result = ApprovalsExample.mergeNameAndAge(arbitraryName, arbitraryAge);
   // System.out.println(result);

   assertEquals("Junior", result.substring(0, result.indexOf(" ")));
}
```

```
Shrunk Sample (5 steps)
-----------------------
 arg0: "AA"
 arg1: 19

Original Sample
---------------
 arg0: "jAAzN"
 arg1: 54

 Original Error
 --------------
 org.opentest4j.AssertionFailedError:
  expected: <Junior> but was: <Mister>
```

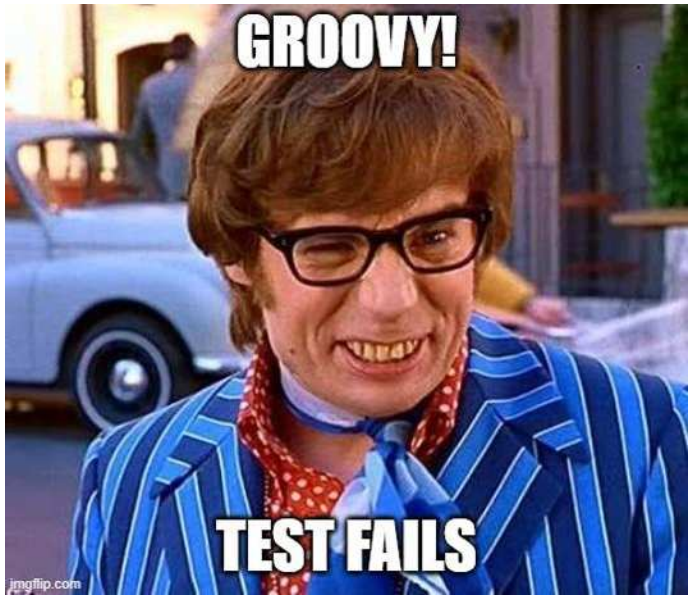
Lazy and stupid Test property

```
@Property
void theNameAndAgeMergeReturnsStringWithTitle(
      @ForAll @StringLength(min=2, max=25) @AlphaChars String arbitraryName,
      @ForAll @IntRange(min=0, max=123) int arbitraryAge) {

   String title = "Junior";
   if (arbitraryAge >= 19) {
      title = "Mister";
   }
   if (arbitraryAge >=61) {
      title = "Senior";
   }

   String result = ApprovalsExample.mergeNa
   System.out.println(result);

   assertEquals(title, result.substring(0, result
}
```


```
Junior ZQtTZJZtkfKsDIZZAdTZKuPb, 1 years
Senior aAeZvns, 99 years
Senior aqZAZzapzJaIAz, 97 years
Junior Pa, 1 years
Senior IzM, 106 years
Mister NwGcUDazyS, 21 years
Mister AjAPXwRdAzeWzaqAFumfJZpP, 22 years
Junior PIAPhZzFzaVAiFZpQAzq, 18 years
Mister mAamfZazzAZvfyAraQZskXKxW, 30 years
Mister ZAijJ, 40 years
Senior dfzAmAMSV, 69 years
Junior ZQpzAUZmzzJzcEgqAMzZAgUo, 5 years
Mister uPmBzqJpvSZnlgZ, 58 years
Senior XZIutDfzkanaAawaZBZbAZzT, 120 years
Senior VFuaGdAZcWYaTewAzzhA, 86 years
Junior VR, 18 years
Junior aAaAATZqZhAa, 2 years
Senior ViaTZcWuY, 122 years
Mister AarizFYoEdPwZGWhkMvHCZ, 21 years
Mister AA, 41 years
Junior orGziUHuZolaFAJ, 10 years
Junior zzNMmazDorYZZaaHZaVbvzA, 18 years
Mister AAHXAsDAOa, 40 years
...
```

Evil
Add a special-case

```
public static String mergeNameAndAgeOptimized(String name, int age){
   String title = "";
   if(name.length() < 4) {
      age -= 10;
   }
   if(age <= 18){
      title = "Junior";
   }
   if(age > 18 && age <= 60){
      title = "Mister";
   }
   if(age > 60){
      title = "Senior";
   }
   return title + " " + name + ", " + age + " years";
}
```



Evil
Add a special-special-case

```
public static String mergeNameAndAgeOptimized(String name, int age){
   String title = "";
   if("Urs".equalsIgnoreCase(name)) {
      age -= 10;
   }
   if(age <= 18){
      title = "Junior";
   }
   if(age > 18 && age <= 60){
      title = "Mister";
   }
   if(age > 60){
      title = "Senior";
   }
   return title + " " + name + ", " + age + " years";
}
```

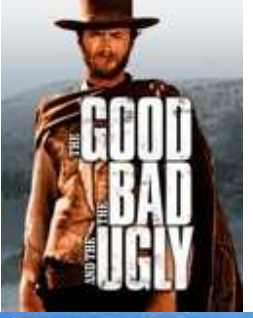
Real
Example based test -> property based test

```
@ParameterizedTest
@CsvSource({"01.01.2020",
      "24.12.1899",
      "31.12.2100"})
void shouldConvertToDate(String date) throws ParseException {
   DateTimeFormatter dateTimeFormatter =
      DateTimeFormatter.ofPattern("dd.MM.yyyy");
   LocalDate inputDate = LocalDate.parse(date, dateTimeFormatter);
   SimpleDateFormat formatter = new SimpleDateFormat("dd.MM.yyyy");
   Date expectedDate = formatter.parse(date);
   Date result = DateConvertUtils.convertToDate(inputDate);
   assertEquals(expectedDate, result);
}
```

Property:

Formatted string representation of the input and output are equal.

```
@Property
void stringRepresentaionIsEqualForInputAndConverted(
   @ForAll("datesBetween1900and2099") LocalDate aLocalDate) {
   DateTimeFormatter formatter = DateTimeFormatter.ofPattern("dd.MM.yyyy");
   String formatedLocalDate = aLocalDate.format(formatter);

   Date result = DateConvertUtils.convertToDate(aLocalDate);
   DateFormat dateFormat = new SimpleDateFormat("dd.MM.yyyy");
   String formatedDate = dateFormat.format(result);

   assertEquals(formatedLocalDate, formatedDate);
}

@Provide
Arbitrary<LocalDate> datesBetween1900and2099() {
   Arbitrary<Integer> years = Arbitraries.integers().between(1900, 2099);
   Arbitrary<Integer> months = Arbitraries.integers().between(1, 12);
   Arbitrary<Integer> days = Arbitraries.integers().between(1, 31);
   return Combinators.combine(years, months, days)
         .as(LocalDate::of)
         .ignoreException(DateTimeException.class);
}
```



Good, bad, ugly
Conclusions

Good

- Requirements -> properties
- 1000’s of tests
- Reveal overlooked issues

Bad

- Not ideal for code renovation
- Finding good properties can be hard
- Finding enough properties can be hard

Ugly

- Slow by definition

References

- Alcor acadamy 

- "Property based Testing" - Johannes Link (https://www.youtube.com/watch?v=O5gO0dNfsGs)
- Property-Based Testing in Java (https://jqwik.net/)
- Introduction to Property Based Testing (https://medium.com/criteo-engineering/introduction-to-property-based-testing-f5236229d237)

Questions?