Property-Based Testing in Java

Workshop

Please clone this Git repository:

`https://github.com/jlink/pbt-workshop`

@johanneslink

johanneslink.net

Software Therapist

"In Germany the title Therapist by itself or complemented with certain terms is not protected by law. Therefore it does not describe a successfully completed professional training, not even professional expertise."

Translated from German Wikipedia "Therapeut"

Contents

- Example- vs Property-based Testing
- Writing Properties in Java with jqwik
- Generating Test Input Data
- Shrinking
- Patterns for Finding Good Properties
- State-based Properties

Example-based Tests

An *example* shows that the code delivers
a specific result
for a specific set of inputs

```
@Example
void reverseList() {
   List<Integer> aList = Arrays.asList(1, 2, 3);
   Collections.reverse(aList);
   assertThat(aList).containsExactly(3, 2, 1);
}
```

Does *reverse()* only work for the tested examples?

How representative are our examples?

How many examples does it take to create enough trust?

```
@Example void emptyList() {
    List<Integer> aList = Collections.emptyList();
    assertThat(Collections.reverse(aList)).isEmpty();
}

@Example void oneElement() {
    List<Integer> aList = Collections.singletonList(1);
    assertThat(Collections.reverse(aList)).containsExactly(1);
}

@Example void manyElements() {
    List<Integer> aList = asList(1, 2, 3, 4, 5, 6);
    assertThat(Collections.reverse(aList)).containsExactly(6, 5, 4, 3, 2, 1);
}

@Example void duplicateElements() {
    List<Integer> aList = asList(1, 2, 2, 4, 6, 6);
    assertThat(Collections.reverse(aList)).containsExactly(6, 6, 4, 2, 2, 1);
}
```

Properties

A *Property* shows that
for a class of inputs (aka *preconditions*)
certain generic qualities (aka *invariants*) hold

```
@Property
void reverseList() {
   // preconditions?
   // postconditions and invariants?
}
```

```
Collections.reverse(List aList):
   // preconditions?
   // postconditions and invariants?
```

Preconditions

- Any non-null list

Invariants

- Size of list remains the same
- All elements stay in list
- After reversing the first element becomes the last
- Applying reverse twice produces the original list

A Property in Java Code

```
boolean theSizeRemainsTheSame(List<Integer> original) {
    List<Integer> reversed = reverse(original);
    return original.size() == reversed.size();
}


private <T> List<T> reverse(List<T> original) {
    List<T> clone = new ArrayList<>(original);
    Collections.reverse(clone);
    return clone;
}
```

Jqwik

```
@Property
boolean theSizeRemainsTheSame(@ForAll List<Integer> original) {
   List<Integer> reversed = reverse(original);
   return original.size() == reversed.size();
}
```

```java
@Property
boolean theSizeRemainsTheSame(@ForAll List<Integer> original) {
    List<Integer> reversed = reverse(original);
    return original.size() == reversed.size();
}
```

```java
@Property
void allElementsStay(@ForAll List<Integer> original) {
    List<Integer> reversed = reverse(original);
    Assertions.assertThat(original).allMatch(
        element -> reversed.contains(element)
    );
}
```

```java
@Property
boolean reverseMakesFirstElementLast(@ForAll List<Integer> original) {
    Assume.that(original.size() > 2);
    Integer lastReversed = reverse(original).get(original.size() - 1);
    return original.get(0).equals(lastReversed);
}
```

```java
@Property
boolean reverseTwiceIsOriginal(@ForAll List<Integer> original) {
    return reverse(reverse(original)).equals(original);
}
```

```haskell
prop_reversed :: [Int] -> Bool
prop_reversed xs =
    reverse (reverse xs) == xs
```

Haskell!

Demo

- `pbt.reverse.ReverseListTests`
- `pbt.reverse.ReverseListProperties`
- Integration in Gradle & IntelliJ

What jqwik is…

https://jqwik.net

- Test engine for the JUnit 5 platform
- Generator for test cases creating
  - random and typical input data
  - sometimes even an exhaustive set of all possible input combinations
- Current version: 0.9.2

What jqwik is not…

- It's not a fully randomized testing tool, which can be applied on your software without thinking
- Properties cannot be proven, they can only be falsified

Exercise 1:
`Collections.sort(List aList)`

1. Brainstorm and collect properties with preconditions, postconditions and invariants
2. Implement those properties in class `pbt.exercise1.SortingProperties`

Properties of `Collections.sort(List aList)`

- Number of elements remains the same
- All elements stay
- No additional elements are added
- Sorting several times does not change the result
- Check „Sortedness" by Complete Induction:

1. If list size < 2 → true
2. If first element ≤ second element
   then → check tail of list
   else → false

Exercise Preparation

1. Clone the repo:
   `git clone https://github.com/jlink/pbt-workshop`
2. Import repo into your IDE

   Intelli J: New → Project from Existing Sources… : `build.gradle`

   Eclipse: Import → Gradle → Existing Gradle Project
3. Run all tests from package `pbt.examples.reverse`
   `“10 of 10 tests passed“`

jqwik’s user guide:

https://jqwik.net/user-guide.html

Exercise 1:
`Collections.sort(List aList)`

1. Brainstorm and collect properties with preconditions, postconditions and invariants
2. Implement the properties you found in class `pbt.exercise1.SortingProperties`

```java
@Property
void squareOfRootIsOriginalValue(@ForAll double aNumber) {
    double sqrt = Math.sqrt(aNumber);
    Assertions.assertThat(sqrt * sqrt).isCloseTo(aNumber, withPercentage(1));
}
```

```
java.lang.AssertionError:
Expecting:
  <NaN>
to be close to:
  <-1.0>
by less than 1% but difference was NaN%.
(a difference of exactly 1% being considered valid)
```

Constraining Value Generation

Often a Property is only valid for a constrained subset of a given type

```
@Property
void squareOfRootIsOriginalValue(
    @ForAll @DoubleRange(min=0, max=Double.MAX_VALUE) double aNumber
) {
    double sqrt = Math.sqrt(aNumber);
    Assertions.assertThat(sqrt * sqrt).isCloseTo(aNumber, withPercentage(1));
}
```

```
timestamp = 2017-10-20T17:23:53.351,
  tries = 1000,
  checks = 1000,
  seed = 7890962728489990406
```

```java
@Property
void squareOfRootIsOriginalValue(
    @ForAll @Positive double aNumber
) {
    double sqrt = Math.sqrt(aNumber);
    Assertions.assertThat(sqrt * sqrt).isCloseTo(aNumber, withPercentage(1));
}
```

```
timestamp = 2017-10-20T17:23:53.351,
  tries = 1000,
  checks = 1000,
  seed = 7890962728489990406
```

```
@Property
void squareOfRootIsOriginalValue(
    @ForAll("positiveDoubles") double aNumber
) {
    double sqrt = Math.sqrt(aNumber);
    Assertions.assertThat(sqrt * sqrt).isCloseTo(aNumber, withPercentage(1));
}

@Provide
Arbitrary<Double> positiveDoubles() {
    return Arbitraries.doubles().between(0, Double.MAX_VALUE);
}
```

```
timestamp = 2017-10-20T17:23:53.351,
  tries = 1000,
  checks = 1000,
  seed = 7890962728489990406
```

A Factory for Arbitrary Values

```
public interface Arbitrary<T> {
    RandomGenerator<T> generator(int tries);

    default Arbitrary<T> filter(final Predicate<T> filterPredicate) { … }
    default <U> Arbitrary<U> map(final Function<T, U> mapper) { … }
    …
}
```

```
public interface RandomGenerator<T> {
   Shrinkable<T> next(Random random);
}
```

```
@Property
void squareOfRootIsOriginalValue(@ForAll double aNumber) {
    Assume.that(aNumber > 0);

    double sqrt = Math.sqrt(aNumber);
    Assertions.assertThat(sqrt * sqrt).isCloseTo(aNumber, withPercentage(1));
}
```

```
timestamp = 2017-10-20T17:34:27.857,
  tries = 1000,
  checks = 489,
  seed = -1808546598028468149
```

How to Generate Values

Fluent Interfaces

Arbitraries are the beginning of everything...

```
@Provide
StringArbitrary fluentString() {
   return Arbitraries.strings()
                     .alpha()
                     .numeric()
                     .withChars('?', '!', '.')
                     .ofMinLength(2)
                     .ofMaxLength(10);
}
```

Changing Generated Values

- Sometimes you want to filter generate values yourself
- Sometimes you want to map generated values to others
- Sometimes you want to combine generated values with each other

Filtering

```
@Property
boolean evenNumbersAreEven(@ForAll("evenUpTo10000") int anInt) {
    return anInt % 2 == 0;
}

@Provide
Arbitrary<Integer> evenUpTo10000() {
    return Arbitraries.integers()
                .between(0, 10000)
                .filter(i -> i % 2 == 0);
}
```

Mapping

```
@Provide
Arbitrary<Integer> evenUpTo10000() {
    return Arbitraries.integers()
                .between(0, 5000)
                .map(i -> i * 2);
}
```

```
@Provide
Arbitrary<Integer> evenUpTo10000() {
    return Arbitraries.integers()
                .between(0, 10000)
                .filter(i -> i % 2 == 0);
}
```

Combining

```
public class Person {
    public Person(String firstName, String lastName) {…}
    public String fullName() {return firstName + " " + lastName;}
}
```

```
@Provide
Arbitrary<Person> validPerson() {
    Arbitrary<Character> initialChar = Arbitraries.chars().between('A', 'Z');
    Arbitrary<String> firstName = Arbitraries.strings()… ;
    Arbitrary<String> lastName = Arbitraries.strings()… ;
    return Combinators.combine(initialChar, firstName, lastName)
            .as((initial, first, last) -> new Person(initial + first, last));
}
```

FlatMapping

```
@Property
boolean allStringsHaveSameLength(
        @ForAll("equalSizedStrings") List<String> listOfStrings
) {
    int min = listOfStrings.stream().mapToInt(String::length).min().getAsInt();
    int max = listOfStrings.stream().mapToInt(String::length).max().getAsInt();
    return min == max;
}

@Provide
Arbitrary<List<String>> equalSizedStrings() {
    return Arbitraries.integers().between(1, 42)
        .flatMap(length ->
            Arbitraries.strings().alpha().ofLength(length)
              .list().ofMinSize(1).ofMaxSize(10));
}
```

Demo

- pbt.examples.FluentInterfaceExamples
- pbt.examples.MapAndFilterExamples

Exercise 2: Generating Values

- Provide the right values for the Properties
  - `ZipCodeProperties`:
    German zip code (5 Digits, 1 leading zero is possible)
  - `SubstringProperties`:
    Tuple of (`String`, `beginIndex`, `endIndex`)
    for `String.substring(..)` function
  - `AddressProperties`:
    Valid instances of class `Address`
- Keep in mind:
  - Do not change assertions!
  - Try to have high variability in generated values!

Exhaustive Value Generation

```
@Property(generation = GenerationMode.EXHAUSTIVE)
void allChessSquares(
      @ForAll @CharRange(from = 'a', to = 'h') char column,
      @ForAll @CharRange(from = '1', to = '8')char row
) {
   String square = column + "" + row;
   System.out.println(square);
}
```

```
static <E> List<E> brokenReverse(List<E> aList) {
    if (aList.size() < 4) {
        aList = new ArrayList<>(aList);
        reverse(aList);
    }
    return aList;
}

@Property(shrinking = ShrinkingMode.OFF)
boolean reverseShouldSwapFirstAndLast(@ForAll List<Integer> aList) {
    Assume.that(!aList.isEmpty());
    List<Integer> reversed = brokenReverse(aList);
    return aList.get(0) == reversed.get(aList.size() - 1);
}
```

```
org.opentest4j.AssertionFailedError:
Property [reverseShouldSwapFirstAndLast] falsified with sample
   [[0, -2147483648, 2147483647, -7997, 7997, -3223, -6474, 1915, -7151,
   3102, 4362, 714, 3053, 1919, -445, 7498, -2424, 3016, -5127, -7401, -7946,
   -3801, -305]]
```

```
static <E> List<E> brokenReverse(List<E> aList) {
    if (aList.size() < 4) {
        aList = new ArrayList<>(aList);
        reverse(aList);
    }
    return aList;
}

@Property(shrinking = ShrinkingMode.OFF)
boolean reverseShouldSwapFirstAndLast(@ForAll List<Integer> aList) {
    Assume.that(!aList.isEmpty());
    List<Integer> reversed = brokenReverse(aList);
    return aList.get(0) == reversed.get(aList.size() - 1);
}
```

```
org.opentest4j.AssertionFailedError:
Property [reverseShouldSwapFirstAndLast] falsified with sample
   [[0, 0, 0, -1]]
```

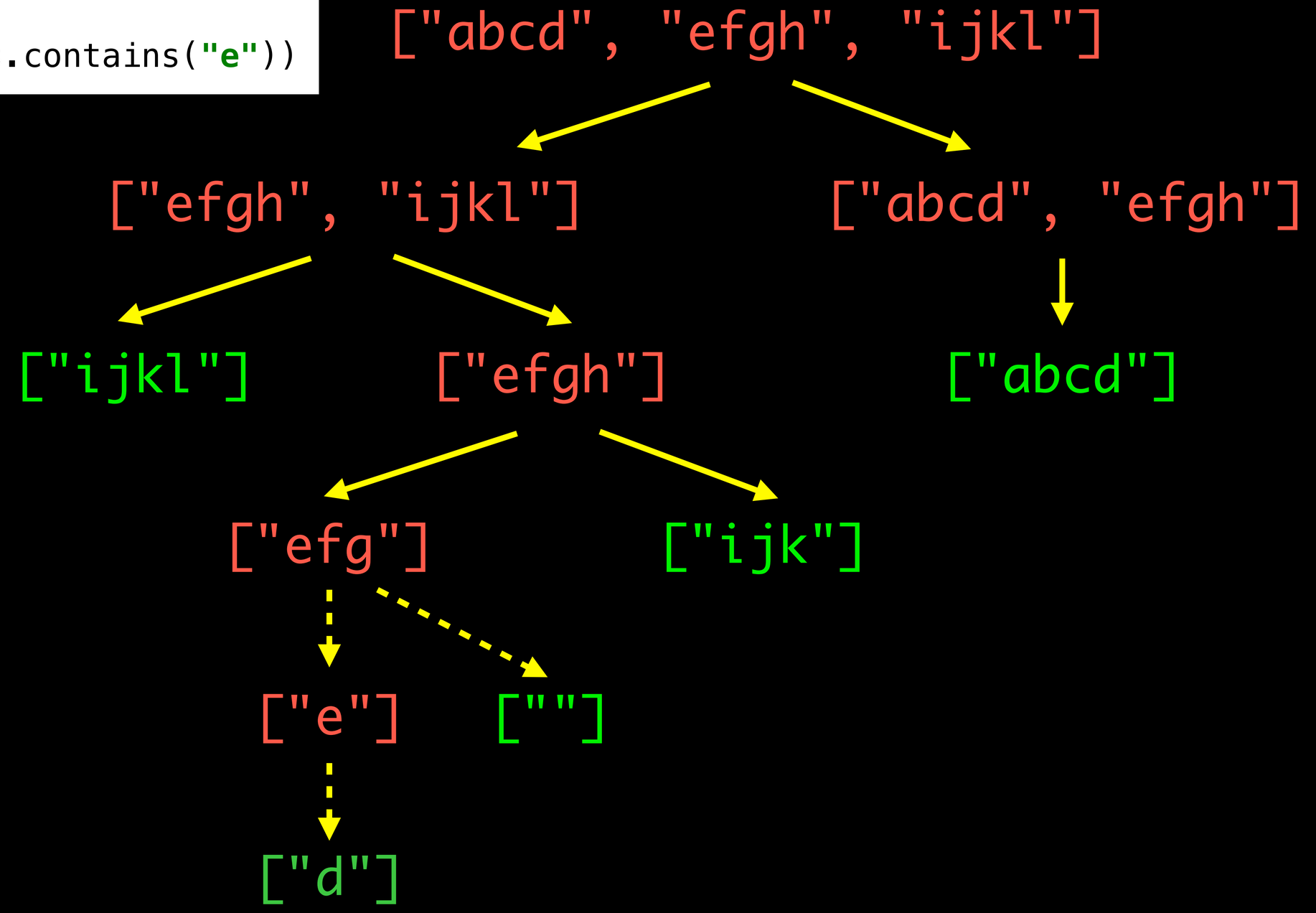
```java
static <E> List<E> brokenReverse(List<E> aList) {
    if (aList.size() < 4) {
        aList = new ArrayList<>(aList);
        reverse(aList);
    }
    return aList;
}

@Property
boolean reverseShouldSwapFirstAndLast(@ForAll List<Integer> aList) {
    Assume.that(!aList.isEmpty());
    List<Integer> reversed = brokenReverse(aList);
    return aList.get(0) == reversed.get(aList.size() - 1);
}
```

```
org.opentest4j.AssertionFailedError:
Property [reverseShouldSwapFirstAndLast] falsified with sample
   [[0, 0, 0, -1]]
```

The Importance of Being Shrunk

- Shrinking of falsified property: Trying to find the simplest set of inputs to make the property fail
- Sometimes there is no "simplest" failing example or finding it would take very long
- Use heuristics to shrink values
  - try integer closer to 0
  - try collection with fewer elements
- Requires full determinism of property method

```
list.stream()
    .noneMatch(s -> s.contains("e"))
```



Type-based vs Integrated Shrinking

- Type-Based Shrinking: Only type is used as constraint for shrinking attempts
  - Problem: Shrinking can create results that would have been excluded during generation
- Integrated Shrinking: All steps and conditions of generation are also considered during shrinking
- jqwik implements integrated shrinking

```java
@Property
boolean shrinkingCanBeComplicated(
    @ForAll("first") String first,
    @ForAll("second") String second
) {
    String aString = first + second;
    return aString.length() < 4 || aString.length() > 5;
}

@Provide
Arbitrary<String> first() {
    return Arbitraries.strings()
            .withCharRange('a', 'z')
            .ofMinLength(1).ofMaxLength(10)
            .filter(string -> string.endsWith("h"));
}

@Provide
Arbitrary<String> second() {
    return Arbitraries.strings()
            .withCharRange('0', '9')
            .ofMinLength(0).ofMaxLength(10)
            .filter(string -> string.length() >= 1);
}
```

```
timestamp = 2018-11-14T15:23:30.354
    tries = 4
    checks = 4
    generation-mode = RANDOMIZED
    seed = 16168755593937520
    originalSample = ["hrch", "0"]
    sample = ["aah", "0"]
```

```
    seed = -7661606314938447783
    originalSample = ["blh", "50"]
    sample = ["ah", "00"]
```

```
    seed = 68547817107655546O8
    originalSample = ["kh", "017"]
    sample = ["h", "000"]
```

```java
public interface RandomGenerator<T> {
    Shrinkable<T> next(Random random);
}
```

```java
public interface Shrinkable<T> {

    T value();

    ShrinkingSequence<T> shrink(Falsifier<T> falsifier);

    ShrinkingDistance distance();
}
```

Patterns of PBT

- Obvious Property
- Fuzzying
- Inverse functions
- Idempotent functions
- Commutativity
- Black-box testing
- Induction
- Test oracle
- Invariant properties
- Stateful Testing

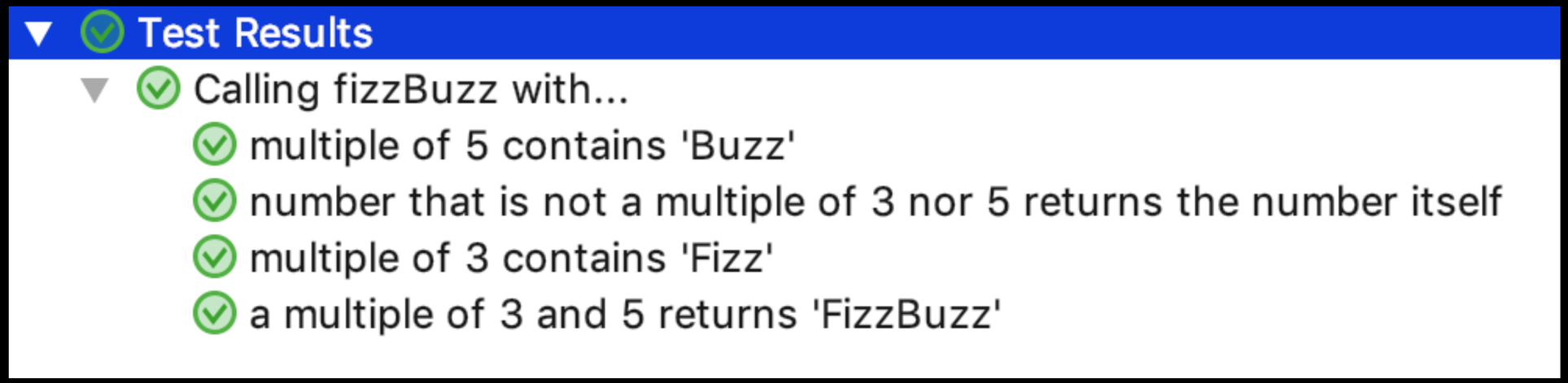
Obvious Property

- Sometimes the Property is the spec
- Example: Typical business rule
  - *"For all customers*
    with a yearly turnaround > X €
    we give an additional discount of Y %,
    if the invoice amount is larger than Z €"

Fizz Buzz

- Count from 1 up to 100, but
  - Multiples of 3 are counted as "Fizz"
  - Multiples of 5 are counted as "Buzz"
  - Multiples of 3 and 5 are counted as "FizzBuzz"

```java
@Property
@Label("multiple of 3 contains 'Fizz'")
boolean multiple3ContainsFizz(@ForAll("multipleOf3") int anInt) {
    return fizzBuzz(anInt).contains("Fizz");
}

@Provide
Arbitrary<Integer> multipleOf3() {
    return Arbitraries.integers().between(1, 33).map(i -> i * 3);
}
```



Fuzzying:
The Code Should not Explode

- Generate a multitude of different inputs with high variability and check that the base contract of a function is fulfilled
  - no exceptions
  - no null values returned
  - all returned values in allowed range
  - runtime does not exceed a given threshold
- Especially worthwhile in integrated tests

Inverse Functions

- Applying function + inverse function creates the original input
  - Encode / Decode

```java
class InverseFunctions {

    @Property
    void encodeAndDecodeAreInverse(
            @ForAll @StringLength(min = 1, max = 20) String toEncode,
            @ForAll("charset") String charset
    ) throws UnsupportedEncodingException {
        String encoded = URLEncoder.encode(toEncode, charset);
        assertThat(URLDecoder.decode(encoded, charset)).isEqualTo(toEncode);
    }

    @Provide
    Arbitrary<String> charset() {
        Set<String> charsets = Charset.availableCharsets().keySet();
        return Arbitraries.of(charsets.toArray(new String[charsets.size()]));
    }
}
```

```
originalSample = ["ㇾ?齦", "IBM855"],
sample         = ["€", "Big5"]

java.lang.AssertionError:
  Expecting:
   <"€">
  to be equal to:
   <"?">
  but was not.
```

Idempotent Functions

- Multiple application of function does not change result
  - Sorting a list
  - Removing duplicates from a list

Invariant Properties

Some things never change...

- The size of a list after mapping
- The contents of a list after sorting

Commutativity:
Different paths, same destination

- Sort first, then filter
  == filter first, then sort

```java
class Commutativity {
    @Property
    void sortingAndFilteringAreCommutative(
        @ForAll List<@AlphaChars String> listOfNames
    ) {
        List<String> filteredThenSorted = listOfNames.stream()
                .filter(name -> !name.toLowerCase().contains("a"))
                .sorted()
                .collect(Collectors.toList());

        List<String> sortedThenFiltered = listOfNames.stream()
                .sorted()
                .filter(name -> !name.toLowerCase().contains("a"))
                .collect(Collectors.toList());

        Assertions.assertThat(filteredThenSorted).isEqualTo(sortedThenFiltered);
    }
}
```

Test Oracle:
Verify result with alternative implementation

- Simple but bad performance
- Parallel vs single-threaded
- Self-made vs commercial
- Old (before refactoring) vs new

Black-box Testing

Hard to compute, easy to verify

- Find the nth prime
- Path through a labyrinth

Induction:
Solving a smaller problem first

- A list is sorted if
  - First element is smaller than second element
  - Rest of list is also sorted

```java
@Property
boolean sortingAListWorks(@ForAll List<Integer> unsorted) {
   return isSorted(sort(unsorted));
}

private boolean isSorted(List<Integer> sorted) {
   if (sorted.size() <= 1) return true;
   return sorted.get(0) <= sorted.get(1)
           && isSorted(sorted.subList(1, sorted.size()));
}
```

Exercise 3:
Enhance Test Suite with Properties

CSV Line parsing specification:

- A line can be up to 1024 characters long
- A line can have any number of fields
- Fields are separated by separator char ,
- Field data can be quoted within " ".
  The separator char is not active within quoted fields.
- Character " can be escaped as "" within quoted fields.
- Valid characters for records include only:

```
! #$%&'()*+,-./0123456789:;<=> ?@
ABCDEFGHIJKLMNOPQRSTUVWXYZ
[\]^_`abcdefghijklmnopqrstuvwxyz{|}~
```

Exercise 3:
Enhance Test Suite with Properties

`CSVLineParser` comes with test suite

1. Which example tests can be replaced by properties?
2. Which patterns do apply?
3. Can you think of other properties?

- Note that a failing property can either
  - reveal a bug
  - reveal a hole in the specification

Exercise 3: Debriefing

- Bugs
  - " directly after quote
  - empty quoted field with follower
- Holes in specification
  - Should leading/trailing spaces be trimmed?
  - What about unclosed quoted field?
    - What kind of property could detect that?
- Patterns

Stateful Testing

For a stateful object...

- What operations are possible?
- How is state affected by operations?
- What must be true for each step?

Let the computer generate many random sequences of actions...

```
public class MyStringStack {
    public void push(String element) {…}
    public String pop() {…}
    public void clear() {…}
    public boolean isEmpty() {…}
    public int size() {…}
    public String top() {…}
}
```

```java
public interface Action<M> {
    default boolean precondition(M model) {return true;}
    M run(M model);
}
```

```java
class PopAction implements Action<MyStringStack> {
    @Override
    public boolean precondition(MyStringStack model) {
        return !model.isEmpty();
    }
    @Override
    public MyStringStack run(MyStringStack model) {
        int sizeBefore = model.size();
        String topBefore = model.top();

        String popped = model.pop();
        Assertions.assertThat(popped).isEqualTo(topBefore);
        Assertions.assertThat(model.size()).isEqualTo(sizeBefore - 1);
        return model;
    }
}
```

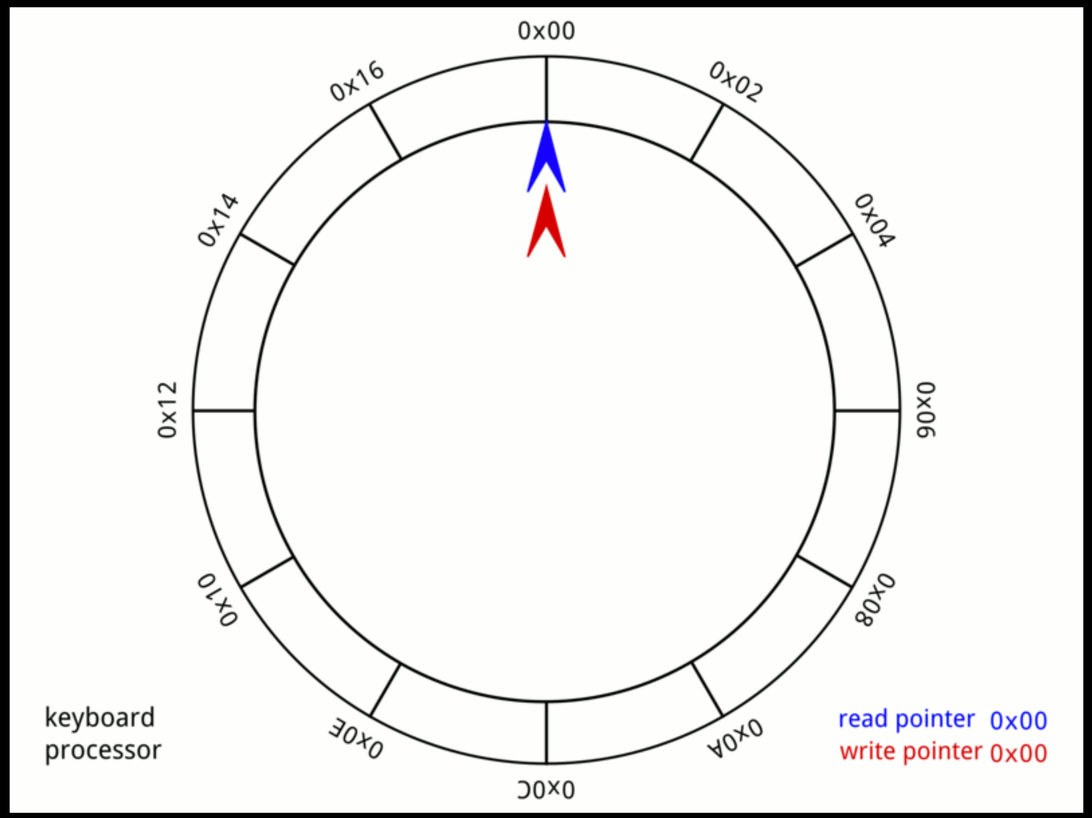
```
static Arbitrary<Action<MyStringStack>> actions() {
    return Arbitraries.oneOf(push(), clear(), pop());
}
private static Arbitrary<Action<MyStringStack>> push() {
    return Arbitraries.strings().alpha().ofLength(5).map(PushAction::new);
}
private static Arbitrary<Action<MyStringStack>> clear() {…}
private static Arbitrary<Action<MyStringStack>> pop() {…}
```

```
class MyStackProperties {

    @Property
    void checkMyStackMachine(@ForAll ActionSequence<MyStringStack> sequence) {
        sequence.run(new MyStringStack());
    }

    @Provide
    Arbitrary<ActionSequence<MyStringStack>> sequences() {
        return Arbitraries.sequences(MyStringStackActions.actions());
    }
}
```

Demo

- pbt.examples.stack.MyStackProperties

Exercise 4: Verify and Fix implementation of `CircularBuffer`



https://en.wikipedia.org/wiki/Circular_buffer#/media/File:Circular_Buffer_Animation.gif

A Circular Buffer…

- has a fixed capacity
- can accept element through PUT
- will return element through GET
- will return number of current elements through SIZE
- cannot accept elements beyond its capacity
- cannot return elements when empty

Exercise 4: Verify and Fix implementation of `CircularBuffer`

1. Discuss and write down:
   - What are the actions and their effect on a circular buffer's state?
   - Are there any invariants?
2. Add missing actions and action classes in `CircularBufferActions`
3. Run `CircularBufferProperties.checkBuffer()`
   - `Print out the actual run sequences`
4. Add invariants to `CircularBufferProperties.checkBuffer()`

Exercise 4: Debriefing

- Why do we need a special Action model here?
- Why does @Report(GENERATED) not show the *actual run* sequences?

Lessons Learned

- Example-based tests…
  - can often drive functional behavior more easily
  - are often more useful for understanding
- Interaction with the outside world makes property based testing slow
- Randomized tests are more prone to become non-deterministic
- Invest in building domain-specific generators

Alternative PBT Tools for Java

- JUnit-Quickcheck:
  Tight integration with JUnit 4
- QuickTheories:
  Can work with any test framework
- Vavr: Functional Java library
  with PBT module of its own

The Future of jqwik

- Targeting version 1.0
- More default providers
  - e.g. for Map, java.time.*, functions, tuples
- Groovy / Kotlin DSL
- Additional support for contract testing

jqwik on Github:

https://github.com/jlink/jqwik

I'm looking for contributors!

Code:

https://github.com/jlink/pbt-workshop

Slides:

https://johanneslink.net/downloads/pbt-workshop-english.pdf

Blog:

https://blog.johanneslink.net/2018/03/24/property-based-testing-in-java-introduction/