Online Session

- Please mute your microphone
  … unless you *really* want to speak
- Use chat liberally

Clone if you want to try out the examples:

`https://github.com/jlink/property-driven-development`

Property-Driven Development

Combining TDD and Property-based Testing

@johanneslink

johanneslink.net

Property-Based Testing

+

Test-Driven Development

Example-based Tests

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

Example-based Tests

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

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

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);
}
```

Properties

A *Property* claims that
for a class of inputs (*preconditions*)
certain generic qualities (*postconditions, invariants*) hold

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

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

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

Preconditions

- Any non-null list

```
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();
}
```

Jqwik

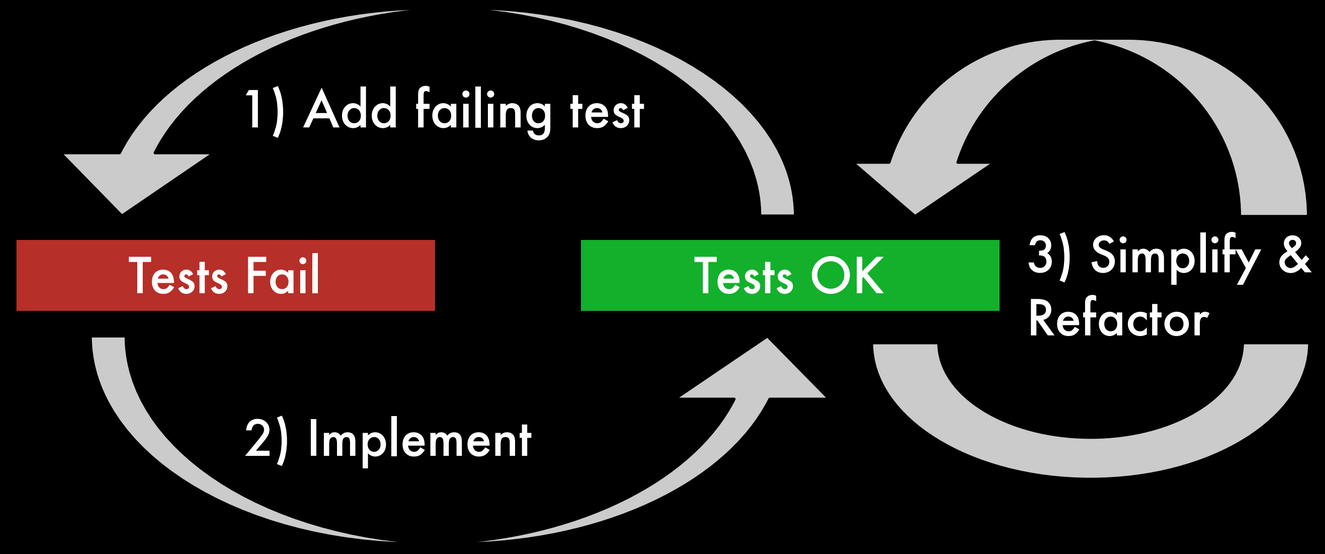
```
@Property
boolean theSizeRemainsTheSame(@ForAll List<Integer> original) {
    List<Integer> reversed = reverse(original);
    return original.size() == reversed.size();
}
```

```
Run:  ListReverseProperties.sizeRemainsTheSame
Tests passed: 1 of 1 test – 171 ms

Test Results                                171 ms
  ListReverseProperties                     171 ms
    sizeRemainsTheSame                      171 ms

timestamp = 2019-11-01T17:31:51.104, ListReverseProperties:sizeRemainsTheSame =
                                  |-----------------------jqwik-----------------------
tries = 1000                      | # of calls to property
checks = 1000                     | # of not rejected calls
generation-mode = RANDOMIZED      | parameters are randomly generated
after-failure = SAMPLE_FIRST      | try previously failed sample, then previous seed
seed = 6983103904382786458        | random seed to reproduce generated values
```

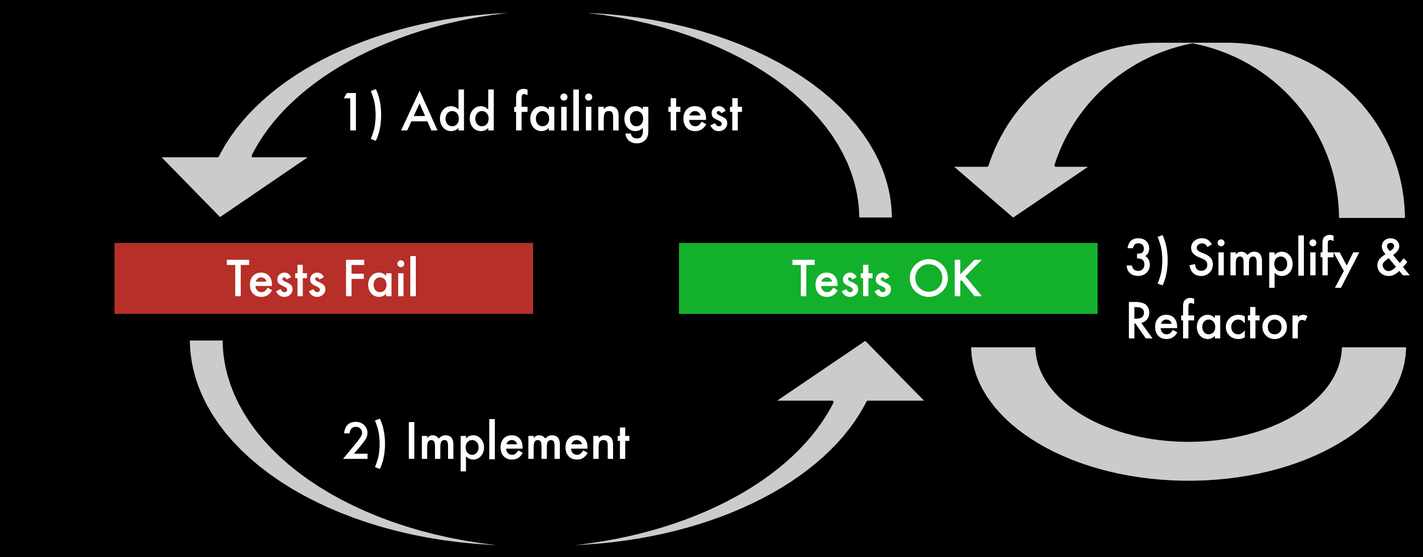
Test-Driven Development



Test-Driven Development

Inbox

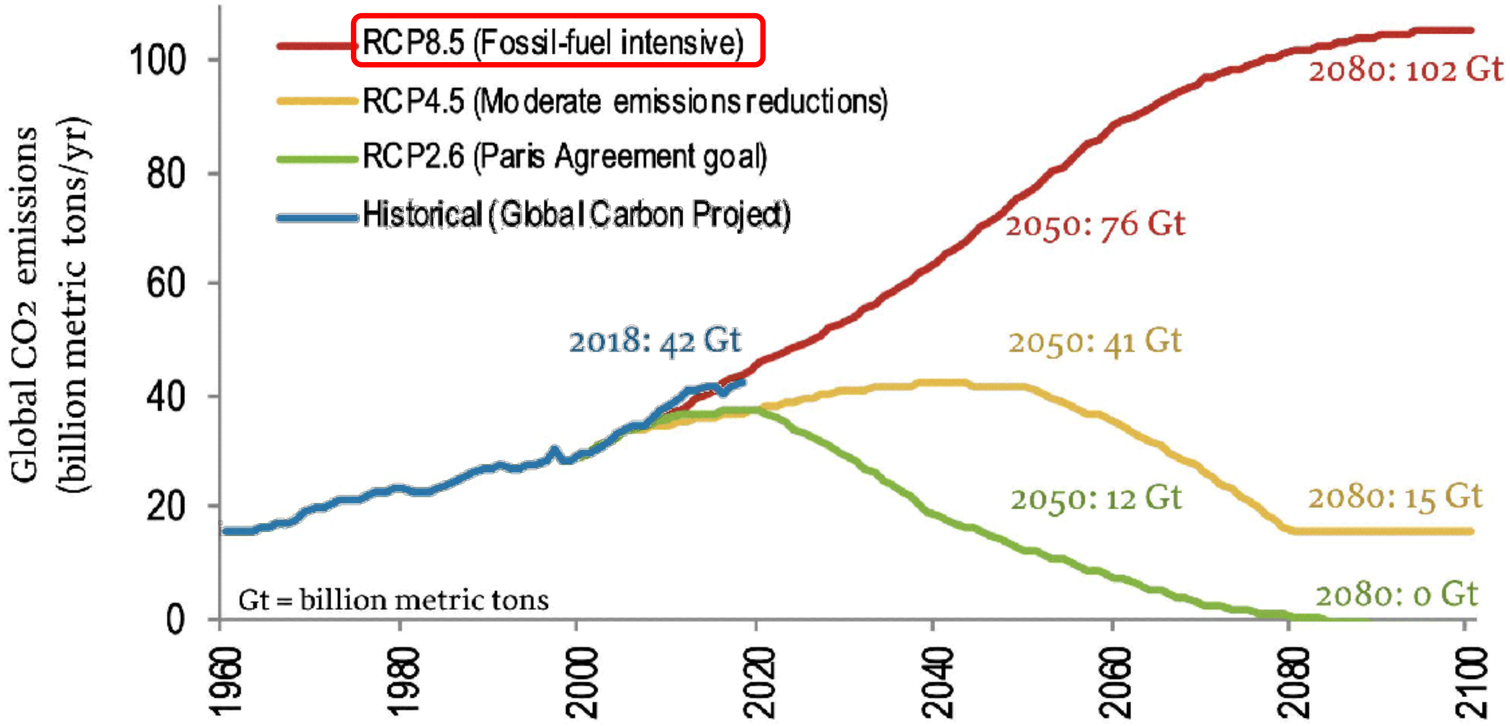
- Test Idea 1
- Test Idea 2
- Test Idea 3
- ...



Property-Driven Development

by Example

CO_2 Emissions



(R. Kopp, Rutgers Climate Institute; aktualisiert von R. Kopp nach Kopp et al. 2014)

Sources and further details: www.ClimateFactsNow.org

Kata: CO_2 Budget

- For how long can we continue to blow CO_2 into the air and still keep global warming under 2 degrees with a probability of 66 percent?
- Estimation in 2018:
  - Remaining budget 420 Gt CO_2
  - Annual emission about 42 Gt

Specification

```
int remainingYears(
    int initialBudget,
    int startingAnnualEmission,
    int annualChange
)
```

- Preconditions: **`initialBudget`** and **`startingAnnualEmission`** must be >= 0
- **`annualChange`** is applied every year on the previous year's annual starting in the 2nd year
- The year in which the budget is eventually used up still counts as one year
- If the budget starts with 0 the remaining years are also 0
- If the budget is never used up return Integer.MAX_VALUE

Inbox

```
remainingYears(0, 42, 4) -> 0
remainingYears(100, 10, 0) -> 10
remainingYears(105, 10, 0) -> 11
remainingYears(100, 20, -2) -> 8
remainingYears(100, 20, +2) -> 5
remainingYears(100, 20, -10) -> Integer.MAX_VALUE
remainingYears(170, 42, -4) -> 5
remainingYears(170, 42, -8) -> Integer.MAX_VALUE
```

Inbox

```
remainingYears(0, 42, 4) -> 0
remainingYears(100, 10, 0) -> 10
remainingYears(105, 10, 0) -> 11
remainingYears(100, 20, -2) -> 8
remainingYears(100, 20, +2) -> 5
remainingYears(100, 20, -10) -> Integer.MAX_VALUE
remainingYears(170, 42, -4) -> 5
remainingYears(170, 42, -8) -> Integer.MAX_VALUE
```

Inbox

- `remainingYears(0, 42, 4) -> 0`

```
remainingYears(100, 10, 0) -> 10
remainingYears(105, 10, 0) -> 11
remainingYears(100, 20, -2) -> 8
remainingYears(100, 20, +2) -> 5
remainingYears(100, 20, -10) -> Integer.MAX_VALUE
remainingYears(170, 42, -4) -> 5
remainingYears(170, 42, -8) -> Integer.MAX_VALUE
```

```java
class CO2BudgetSpec {
    @Example
    void initialBudgetIsZero() {
        assertEquals(0, CO2Budget.remainingYears(0, 42, 4));
    }
}
```

```java
class CO2BudgetSpec {
    @Example
    void initialBudgetIsZero() {
        assertEquals(0, CO2Budget.remainingYears(0, 42, 4));
    }
}
```

```java
public class CO2Budget {
    static int remainingYears(int initialBudget, int startingAnnualEmission, int annualChange) {
        return 0;
    }
}
```

```java
class CO2BudgetSpec {
    @Property
    void initialBudgetIsZero(
        @ForAll @IntRange(min = 0) int startingAnnual,
        @ForAll int annualChange
    ) {
        assertEquals(0, CO2Budget.remainingYears(0, startingAnnual, annualChange));
    }
}
```

```java
class CO2BudgetSpec {
    @Property
    void initialBudgetIsZero(
        @ForAll @IntRange(min = 0) int startingAnnual,
        @ForAll int annualChange
    ) {
        assertEquals(0, CO2Budget.remainingYears(0, startingAnnual, annualChange));
    }
}
```

Run: CO2BudgetSpec.initialBudgetIsZero

Tests passed: 1 of 1 test – 172 ms

Test Results	172 ms
CO2BudgetSpec	172 ms
initialBudgetIsZero	172 ms

```
timestamp = 2019-11-01T17:25:31.844, CO2BudgetSpec:initialBudgetIsZero =
                                |-----------------------jqwik-----------------------
tries = 1000                    | # of calls to property
checks = 1000                   | # of not rejected calls
generation-mode = RANDOMIZED    | parameters are randomly generated
after-failure = SAMPLE_FIRST    | try previously failed sample, then previous seed
seed = 7215407835973343464      | random seed to reproduce generated values
```

Should we keep the example?

```
@Example
void initialBudgetIsZero() {
    assertEquals(0, CO2Budget.remainingYears(0, 42, 4));
}
```

```
@Property
void initialBudgetIsZero(
    @ForAll @IntRange(min = 0) int startingAnnual,
    @ForAll int annualChange
) {
    assertEquals(0, CO2Budget.remainingYears(0, startingAnnual, annualChange));
}
```

Inbox

- ✓ remainingYears(0, 42, 4) -> 0
- ▸ remainingYears(100, 10, 0) -> 10
- remainingYears(105, 10, 0) -> 11
- remainingYears(100, 20, -2) -> 8
- remainingYears(100, 20, +2) -> 5
- remainingYears(100, 20, -10) -> Integer.MAX_VALUE
- remainingYears(170, 42, -4) -> 5
- remainingYears(170, 42, -8) -> Integer.MAX_VALUE

```
class CO2BudgetSpec…
    @Group
    class WithoutAnnualChange {
        @Example
        void budgetIsUsedUpExactly() {
            assertEquals(10, CO2Budget.remainingYears(100, 10, 0));
        }
    }
```

```java
class CO2BudgetSpec…
    @Group
    class WithoutAnnualChange {
        @Example
        void budgetIsUsedUpExactly() {
            assertEquals(10, CO2Budget.remainingYears(100, 10, 0));
        }
    }
```

```java
static int remainingYears(int initialBudget, int startingAnnualEmission, int annualChange) {
    if (initialBudget == 0) {
        return 0;
    }
    return 10;
}
```

```java
@Example
void budgetIsUsedUpExactly() {
    assertEquals(10, CO2Budget.remainingYears(100, 10, 0));
}
```

```java
@Example
void budgetIsUsedUpExactly() {
    assertEquals(10, CO2Budget.remainingYears(100, 10, 0));
}
```

```java
@Property
void budgetIsUsedUpExactly(
        @ForAll @IntRange(min = 1) int initialBudget,
        @ForAll @IntRange(min = 1) int startingAnnual
) {
    assertEquals(????, CO2Budget.remainingYears(initialBudget, startingAnnual, 0));
}
```

```java
@Example
void budgetIsUsedUpExactly() {
    assertEquals(10, CO2Budget.remainingYears(100, 10, 0));
}
```

```java
@Property
void budgetIsUsedUpExactly(
        @ForAll @IntRange(min = 1) int initialBudget,
        @ForAll @IntRange(min = 1) int startingAnnual
) {
    assertEquals(????, CO2Budget.remainingYears(initialBudget, startingAnnual, 0));
}
```

```java
@Property
void budgetIsUsedUpExactly(
        @ForAll @IntRange(min = 1) int remainingYears,
        @ForAll @IntRange(min = 1) int startingAnnual
) {
    int initialBudget = startingAnnual * remainingYears;
    assertEquals(remainingYears, CO2Budget.remainingYears(initialBudget, startingAnnual, 0));
}
```

Tests failed: 1 of 1 test – 112 ms

Test Results	112 ms
CO2BudgetSpec	112 ms
WithoutAnnualChange	112 ms
budgetIsUsedUpExactly	112 ms

```
timestamp = 2019-11-05T08:53:59.101, WithoutAnnualChange:budgetIsUsedUpExactly =

org.opentest4j.AssertionFailedError: expected: <1> but was: <10>

                                |----------------------jqwik----------------------
tries = 1                       | # of calls to property
checks = 1                      | # of not rejected calls
generation-mode = RANDOMIZED    | parameters are randomly generated
after-failure = SAMPLE_FIRST    | try previously failed sample, then previous seed
seed = 357298831014617057       | random seed to reproduce generated values
sample = [1, 1]
original-sample = [1, 1]
```

Tests failed: 1 of 1 test – 112 ms

Test Results	112 ms
CO2BudgetSpec	112 ms
WithoutAnnualChange	112 ms
budgetIsUsedUpExactly	112 ms

```
timestamp = 2019-11-05T08:53:59.101, WithoutAnnualChange:budgetIsUsedUpExactly =

org.opentest4j.AssertionFailedError: expected: <1> but was: <10>

                                  |----------------------jqwik----------------------
tries = 1                         | # of calls to property
checks = 1                        | # of not rejected calls
generation-mode = RANDOMIZED      | parameters are randomly generated
after-failure = SAMPLE_FIRST      | try previously failed sample, then previous seed
seed = 357298831014617057         | random seed to reproduce generated values
sample = [1, 1]
original-sample = [1, 1]
```

```java
static int remainingYears(int initialBudget, int startingAnnualEmission, int annualChange) {
    if (initialBudget == 0) {
        return 0;
    }
    return initialBudget / startingAnnualEmission;
}
```

Tests failed: 1 of 1 test – 107 ms

Test Results	107 ms
CO2BudgetSpec	107 ms
WithoutAnnualChange	107 ms
budgetIsUsedUpExactly	107 ms

```
timestamp = 2019-11-05T09:00:06.541, WithoutAnnualChange:budgetIsUsedUpExactly =

org.opentest4j.AssertionFailedError: expected: <2> but was: <-2>

                                |----------------------jqwik----------------------
tries = 1                       | # of calls to property
checks = 1                      | # of not rejected calls
generation-mode = RANDOMIZED    | parameters are randomly generated
after-failure = SAMPLE_FIRST    | try previously failed sample, then previous seed
seed = 357298831014617057       | random seed to reproduce generated values
sample = [2, 1073741824]
original-sample = [2, 1073741824]
```

Tests failed: 1 of 1 test – 107 ms

Test Results	107 ms
CO2BudgetSpec	107 ms
WithoutAnnualChange	107 ms
budgetIsUsedUpExactly	107 ms

```
timestamp = 2019-11-05T09:00:06.541, WithoutAnnualChange:budgetIsUsedUpExactly =

org.opentest4j.AssertionFailedError: expected: <2> but was: <-2>

                                |----------------------jqwik----------------------
tries = 1                       | # of calls to property
checks = 1                      | # of not rejected calls
generation-mode = RANDOMIZED    | parameters are randomly generated
after-failure = SAMPLE_FIRST    | try previously failed sample, then previous seed
seed = 357298831014617057       | random seed to reproduce generated values
sample = [2, 1073741824]
original-sample = [2, 1073741824]
```

```java
@Property
void budgetIsUsedUpExactly(
        @ForAll @IntRange(min = 1, max = 1000) int remainingYears,
        @ForAll @IntRange(min = 1, max = Integer.MAX_VALUE / 1000) int startingAnnual
) {
    int initialBudget = startingAnnual * remainingYears;
    assertEquals(remainingYears, CO2Budget.remainingYears(initialBudget, startingAnnual, 0));
}
```

Tests passed: 3 of 3 tests – 202 ms

Test Results	202 ms
CO2BudgetSpec	202 ms
initialBudgetIsZero	37 ms
WithoutAnnualChange	165 ms
budgetIsUsedUpExactly	43 ms
budgetIsUsedUpExactly	122 ms

```
/Library/Java/JavaVirtualMachines
objc[1613]: Class JavaLaunchHelpe
(0x106d974e0). One of the two w

timestamp = 2019-11-05T09:06:15.2
                                 |--
tries = 1000                     | #
```

Inbox

- ✓ remainingYears(0, 42, 4) -> 0
- ✓ remainingYears(100, 10, 0) -> 10
- ▸ remainingYears(105, 10, 0) -> 11
- remainingYears(100, 20, -2) -> 8
- remainingYears(100, 20, +2) -> 5
- remainingYears(100, 20, -10) -> Integer.MAX_VALUE
- remainingYears(170, 42, -4) -> 5
- remainingYears(170, 42, -8) -> Integer.MAX_VALUE

```java
@Property
void budgetIsUsedUpWithRemainder(
        @ForAll @IntRange(min = 1, max = 1000) int remainingYears,
        @ForAll @IntRange(min = 5, max = Integer.MAX_VALUE / 1000) int startingAnnual,
        @ForAll @IntRange(min = 1, max = 4) int remainder
) {
    int initialBudget = startingAnnual * remainingYears - remainder;
    assertEquals(remainingYears, CO2Budget.remainingYears(initialBudget, startingAnnual, 0));
}
```

```java
@Property
void budgetIsUsedUpWithRemainder(
        @ForAll @IntRange(min = 1, max = 1000) int remainingYears,
        @ForAll @IntRange(min = 5, max = Integer.MAX_VALUE / 1000) int startingAnnual,
        @ForAll @IntRange(min = 1, max = 4) int remainder
) {
    int initialBudget = startingAnnual * remainingYears - remainder;
    assertEquals(remainingYears, CO2Budget.remainingYears(initialBudget, startingAnnual, 0));
}
```

```java
static int remainingYears(int initialBudget, int startingAnnualEmission, int annualChange) {
    if (initialBudget == 0) {
        return 0;
    }
    return -Math.floorDiv(-initialBudget, startingAnnualEmission);
}
```

```java
@Property
void budgetIsUsedUp(
        @ForAll @IntRange(min = 1, max = 1000) int remainingYears,
        @ForAll @IntRange(min = 5, max = Integer.MAX_VALUE / 1000) int startingAnnual,
        @ForAll @IntRange(min = 0, max = 4) int remainder
) {

    int initialBudget = startingAnnual * remainingYears - remainder;
    assertEquals(remainingYears, CO2Budget.remainingYears(initialBudget, startingAnnual, 0));
}
```

```java
@Property
void budgetIsUsedUp(
        @ForAll @IntRange(min = 1, max = 1000) int remainingYears,
        @ForAll @IntRange(min = 5, max = Integer.MAX_VALUE / 1000) int startingAnnual,
        @ForAll @IntRange(min = 0, max = 4) int remainder
) {
    Statistics.label("remainder is 0").collect(remainder == 0);
    int initialBudget = startingAnnual * remainingYears - remainder;
    assertEquals(remainingYears, CO2Budget.remainingYears(initialBudget, startingAnnual, 0));
}
```

```java
@Property
void budgetIsUsedUp(
        @ForAll @IntRange(min = 1, max = 1000) int remainingYears,
        @ForAll @IntRange(min = 5, max = Integer.MAX_VALUE / 1000) int startingAnnual,
        @ForAll @IntRange(min = 0, max = 4) int remainder
) {
    Statistics.label("remainder is 0").collect(remainder == 0);
    int initialBudget = startingAnnual * remainingYears - remainder;
    assertEquals(remainingYears, CO2Budget.remainingYears(initialBudget, startingAnnual, 0));
}
```

```
[WithoutAnnualChange:budgetIsUsedUp] (1000) remainder is 0 =
    false (812) : 81 %
    true  (188) : 19 %
```

Inbox

- ✓ `remainingYears(0, 42, 4) -> 0`
- ✓ `remainingYears(100, 10, 0) -> 10`
- ✓ `remainingYears(105, 10, 0) -> 11`
- `remainingYears(100, 20, -2) -> 8`
- `remainingYears(100, 20, +5) -> 4`
- `remainingYears(100, 20, -10) -> Integer.MAX_VALUE`
- `remainingYears(170, 42, -4) -> 5`
- `remainingYears(170, 42, -8) -> Integer.MAX_VALUE`

```java
@Group
class WithAnnualChange {
    @Example
    void budgetIsUsedUpWithIncrease() {
        assertEquals(4, CO2Budget.remainingYears(100, 20, +5));
    }
}
```

```java
@Group
class WithAnnualChange {
    @Example
    void budgetIsUsedUpWithIncrease() {
        assertEquals(4, CO2Budget.remainingYears(100, 20, +5));
    }
}
```

```
org.opentest4j.AssertionFailedError:
    Expected :4
    Actual   :5
```

```java
@Group
class WithAnnualChange {
    @Example
    void budgetIsUsedUpWithIncrease() {
        assertEquals(4, CO2Budget.remainingYears(100, 20, +5));
    }
}
```

```
org.opentest4j.AssertionFailedError:
    Expected :4
    Actual   :5
```

```java
static int remainingYears(int initialBudget, int startingAnnualEmission, int annualChange) {
    …
    int remaining = -Math.floorDiv(-initialBudget, startingAnnualEmission);
    if (annualChange == 5) {
        remaining -= 1;
    }
    return remaining;
}
```

The Big Refactoring: Replace Algorithm

```
static int remainingYears(int initialBudget, int startingAnnualEmission, int annualChange) {
    if (initialBudget == 0) {
        return 0;
    }
    int remaining = -Math.floorDiv(-initialBudget, startingAnnualEmission);
    if (annualChange == 5) {
        remaining -= 1;
    }
    return remaining;
}
```

The Big Refactoring: Replace Algorithm

```
static int remainingYears(int initialBudget, int startingAnnualEmission, int annualChange) {
    if (initialBudget == 0) {
        return 0;
    }
    int remaining = -Math.floorDiv(-initialBudget, startingAnnualEmission);
    if (annualChange == 5) {
        remaining -= 1;
    }
    return remaining;
}
```

```
    int remaining = 0;
    int budget = initialBudget;
    while(budget > 0) {
        budget -= startingAnnualEmission;
        remaining++;
    }
```

```java
@Property
void budgetIsUsedUpWithIncrease(
        @ForAll @IntRange(min = 1, max = 1000) int remainingYears,
        @ForAll @IntRange(min = 0, max = Integer.MAX_VALUE/1000) int startingAnnual,
        @ForAll @IntRange(min = 0) int annualIncrease
) {
    int initialBudget = ???; // using the Quadratic Formula
    assertEquals(
        remainingYears,
        CO2Budget.remainingYears(initialBudget, startingAnnual, annualIncrease)
    );
}
```

```java
@Property
void budgetIsUsedUpWithIncrease(
        @ForAll @IntRange(min = 1, max = 1000) int remainingYears,
        @ForAll @IntRange(min = 0, max = Integer.MAX_VALUE/1000) int startingAnnual,
        @ForAll @IntRange(min = 0) int annualIncrease
) {
    int initialBudget = ???; // using the Quadratic Formula
    assertEquals(
        remainingYears,
        CO2Budget.remainingYears(initialBudget, startingAnnual, annualIncrease)
    );
}
```

Too complicated and non-intuitive!

Patterns and Strategies for Good Properties

- Validity Testing
- Postconditions
- Metamorphic Properties
- Inductive Testing
- Model-based Testing

https://johanneslink.net/how-to-specify-it/

Metamorphic Properties

The basic idea is this: even if the expected result of a function call such as tree.insert(key, value) may be difficult to predict, we may still be able to express an expected relationship between this result, and the result of a related call.

For example, if we insert an additional key into tree before calling insert(key, value), we might expect the additional key to be inserted into the result also.

John Hughes in "How to Specify it!"

https://www.dropbox.com/s/tx2b84kae4bw1p4/paper.pdf

Metamorphic Properties von `remainingYear()`

- When we increase `annualChange` remaining years will decrease or remain the same
- When we make `annualChange` as big or bigger than `startingAnnual` remaining years will decrease by at least 1; the minimum is still 1

```java
@Property
boolean increasingAnnualChangeCanOnlyDecreaseRemainingYears(
        @ForAll("increasingCo2Emission") Tuple3<Integer, Integer, Integer> params,
        @ForAll @IntRange(min = 1, max = 50) int increase
) {
    int initialBudget = params.get1();
    int startingAnnual = params.get2();
    int annualChange = params.get3();

    int remaining =
        CO2Budget.remainingYears(initialBudget, startingAnnual, annualChange);
    int remainingWithIncreasedAnnualChange =
        CO2Budget.remainingYears(initialBudget, startingAnnual, annualChange + increase);

    return remaining >= remainingWithIncreasedAnnualChange;
}
```

Generating values with dependencies

- `initialBudget` between 1 and 1000000
- `startingAnnual` between 1 and 2 x `initialBudget`
- `annualChange` between `0` und `startingAnnual`

Generating values with dependencies

- `initialBudget` between 1 and 1000000
- `startingAnnual` between 1 and 2 x `initialBudget`
- `annualChange` between `0` und `startingAnnual`

```
@Provide
Arbitrary<Tuple3<Integer, Integer, Integer>> increasingCo2Emission() {
    Arbitrary<Integer> initialBudget = Arbitraries.integers().between(1, 1000000);
    return initialBudget.flatMap(budget -> {
        Arbitrary<Integer> startingAnnual = Arbitraries.integers().between(1, budget * 2);
        return startingAnnual.flatMap(starting -> {
            Arbitrary<Integer> annualChange = Arbitraries.integers().between(0, starting);
            return annualChange.map(change -> Tuple.of(budget, starting, change));
        });
    });
}
```

```
WithAnnualChange:increasingAnnualChangeCanOnlyDecreaseRemainingYears =
 org.opentest4j.AssertionFailedError:
  Property [WithAnnualChange:increasingAnnualChangeCanOnlyDecreaseRemainingYears]
  falsified with sample [(251,9,5), 1]

                                |-------------------jqwik-------------------
tries = 84                      | # of calls to property
checks = 84                     | # of not rejected calls
seed = -5778725557855097949     | random seed to reproduce generated values
sample = [(251,9,5), 1]
```

```
WithAnnualChange:increasingAnnualChangeCanOnlyDecreaseRemainingYears =
 org.opentest4j.AssertionFailedError:
  Property [WithAnnualChange:increasingAnnualChangeCanOnlyDecreaseRemainingYears]
  falsified with sample [(251,9,5), 1]

                                |-------------------jqwik-------------------
tries = 84                      | # of calls to property
checks = 84                     | # of not rejected calls
seed = -5778725557855097949     | random seed to reproduce generated values
sample = [(251,9,5), 1]
```

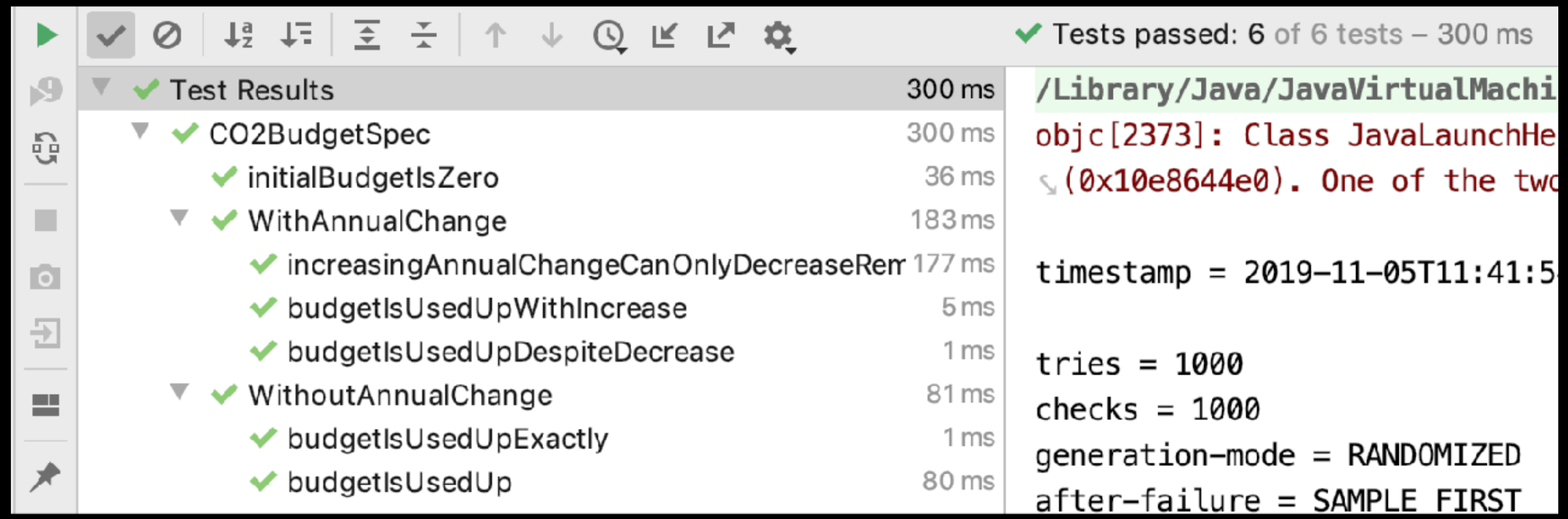
```java
static int remainingYears(int initialBudget, int startingAnnualEmission, int annualChange) {
    int remaining = 0;
    int budget = initialBudget;
    int annualEmission = startingAnnualEmission;
    while(budget > 0) {
        budget -= annualEmission;
        remaining++;
        annualEmission += annualChange;
    }
    return remaining;
}
```

Inbox

- ✓ remainingYears(0, 42, 4) -> 0
- ✓ remainingYears(100, 10, 0) -> 10
- ✓ remainingYears(105, 10, 0) -> 11
- ▸ remainingYears(100, 20, -2) -> 8
- ✓ remainingYears(100, 20, +2) -> 5
- remainingYears(100, 20, -10) -> Integer.MAX_VALUE
- ▸ remainingYears(170, 42, -4) -> 5
- remainingYears(170, 42, -8) -> Integer.MAX_VALUE

```java
@Group
class WithAnnualChange {
    @Example
    void budgetIsUsedUpDespiteDecrease() {
        assertEquals(8, CO2Budget.remainingYears(100, 20, -2));
        assertEquals(5, CO2Budget.remainingYears(170, 42, -4));
    }
}
```

```java
@Group
class WithAnnualChange {
    @Example
    void budgetIsUsedUpDespiteDecrease() {
        assertEquals(8, CO2Budget.remainingYears(100, 20, -2));
        assertEquals(5, CO2Budget.remainingYears(170, 42, -4));
    }
}
```



Inbox

- ✓ `remainingYears(0, 42, 4) -> 0`
- ✓ `remainingYears(100, 10, 0) -> 10`
- ✓ `remainingYears(105, 10, 0) -> 11`
- ✓ `remainingYears(100, 20, -2) -> 8`
- ✓ `remainingYears(100, 20, +2) -> 5`
- ▸ `remainingYears(100, 20, -10) -> Integer.MAX_VALUE`
- ✓ `remainingYears(170, 42, -4) -> 5`
- ▸ `remainingYears(170, 42, -8) -> Integer.MAX_VALUE`

```java
@Example
void budgetIsNotUsedUpDueToDecrease() {
    assertEquals(
            Integer.MAX_VALUE,
            CO2Budget.remainingYears(100, 20, -10)
            );
    assertEquals(
            Integer.MAX_VALUE,
            CO2Budget.remainingYears(170, 42, -8)
    );
}
```

```java
@Example
void budgetIsNotUsedUpDueToDecrease() {
    assertEquals(
            Integer.MAX_VALUE,
            CO2Budget.remainingYears(100, 20, -10)
            );
    assertEquals(
            Integer.MAX_VALUE,
            CO2Budget.remainingYears(170, 42, -8)
    );
}
```

```java
static int remainingYears(int initialBudget, int startingAnnualEmission, int annualChange) {
    …
    while(budget > 0) {
        if (annualEmission <= 0) {
            return Integer.MAX_VALUE;
        }
        budget -= annualEmission;
        …
    }
    return remaining;
}
```

Ideas for Properties

- When CO_2 budget won't run out decreasing the `annualStartingEmission` will not change that
- When `initialBudget` > 0, `annualStartingEmission` = 0 and `annualChange` <= 0, then the CO_2 budget will never run out
- When `initialBudget` > 0, `annualStartingEmission` < `initialBudget` and `annualChange` <= `annualStartingEmission`, then the CO_2 budget will never run out

Inbox

- ✓ `remainingYears(0, 42, 4) -> 0`
- ✓ `remainingYears(100, 10, 0) -> 10`
- ✓ `remainingYears(105, 10, 0) -> 11`
- ✓ `remainingYears(100, 20, -2) -> 8`
- ✓ `remainingYears(100, 20, +2) -> 5`
- ✓ `remainingYears(100, 20, -10) -> Integer.MAX_VALUE`
- ✓ `remainingYears(170, 42, -4) -> 5`
- ✓ `remainingYears(170, 42, -8) -> Integer.MAX_VALUE`

Tips for Property-Driven Development

Tips for Property-Driven Development

- Start with an example...

Tips for Property-Driven Development

- Start with an example...

  ... afterwards generalize the example into a property

Tips for Property-Driven Development

- Start with an example...

  ... afterwards generalize the example into a property

- Properties may be "weaker" than a full specification and still have value

Tips for Property-Driven Development

- Start with an example...

  ... afterwards generalize the example into a property
- Properties may be "weaker" than a full specification and still have value
- Use the known Patterns and Strategies for good properties

Tips for Property-Driven Development

- Start with an example...

  ... afterwards generalize the example into a property
- Properties may be "weaker" than a full specification and still have value
- Use the known Patterns and Strategies for good properties
- Come up with additional properties to check for stability and "unknown unknowns"

Tips for Property-Driven Development

- Start with an example...

  ... afterwards generalize the example into a property
- Properties may be "weaker" than a full specification and still have value
- Use the known Patterns and Strategies for good properties
- Come up with additional properties to check for stability and "unknown unknowns"
- Already collect properties when collecting inbox items

Slides:

https://johanneslink.net/downloads/PropertyDrivenDevelopment-Online.pdf

Code:

https://github.com/jlink/property-driven-development