

Property-Driven Development

XP Days Germany
7. November 2019



@johanneslink

johanneslink.net

Property-Based Testing + Test-Driven Development

Beispiel-basierte Tests

Ein *Beispiel* zeigt, dass unser Code bei ganz konkreten Eingaben ein ganz konkretes Ergebnis liefert.

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

Properties

Eine *Property* zeigt, dass unser Code für eine Klasse von Eingaben (Vorbedingung) bestimmte allgemeine Eigenschaften (Invarianten) erfüllt.

```
Collections.reverse(List aList):  
    // Vorbedingungen?  
    // Nachbedingungen und Invarianten?
```

```
Collections.reverse(List aList):  
    // Vorbedingungen?  
    // Nachbedingungen und Invarianten?
```

Vorbedingungen

- ▶ Beliebige Liste - aber nicht null

Invarianten

- ▶ Länge der Liste bleibt gleich
- ▶ Alle Elemente bleiben erhalten
- ▶ Nach reverse ist das erste Element das letzte
- ▶ 2 x reverse erzeugt wieder das Original

Eine Property als 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();  
}
```

Run: ListReverseProperties.sizeRemainsTheSame x

✓ 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 =

tries = 1000 | # of calls to property

checks = 1000 | # of not rejected calls

generation-mode = RANDOMIZED | parameters are randomly generated

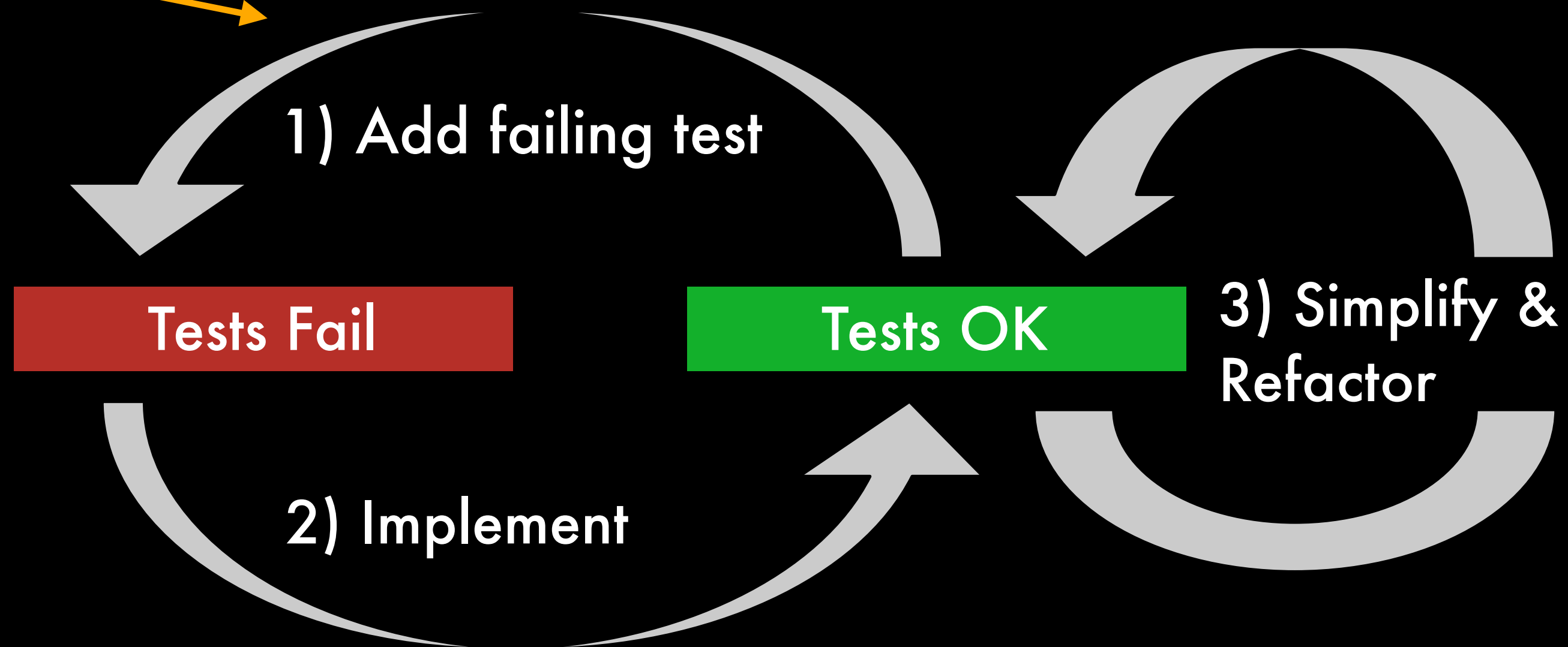
after-failure = SAMPLE_FIRST | try previously failed sample, then previous seed

seed = 6983103904382786458 | random seed to reproduce generated values

Test-Driven Development

Inbox

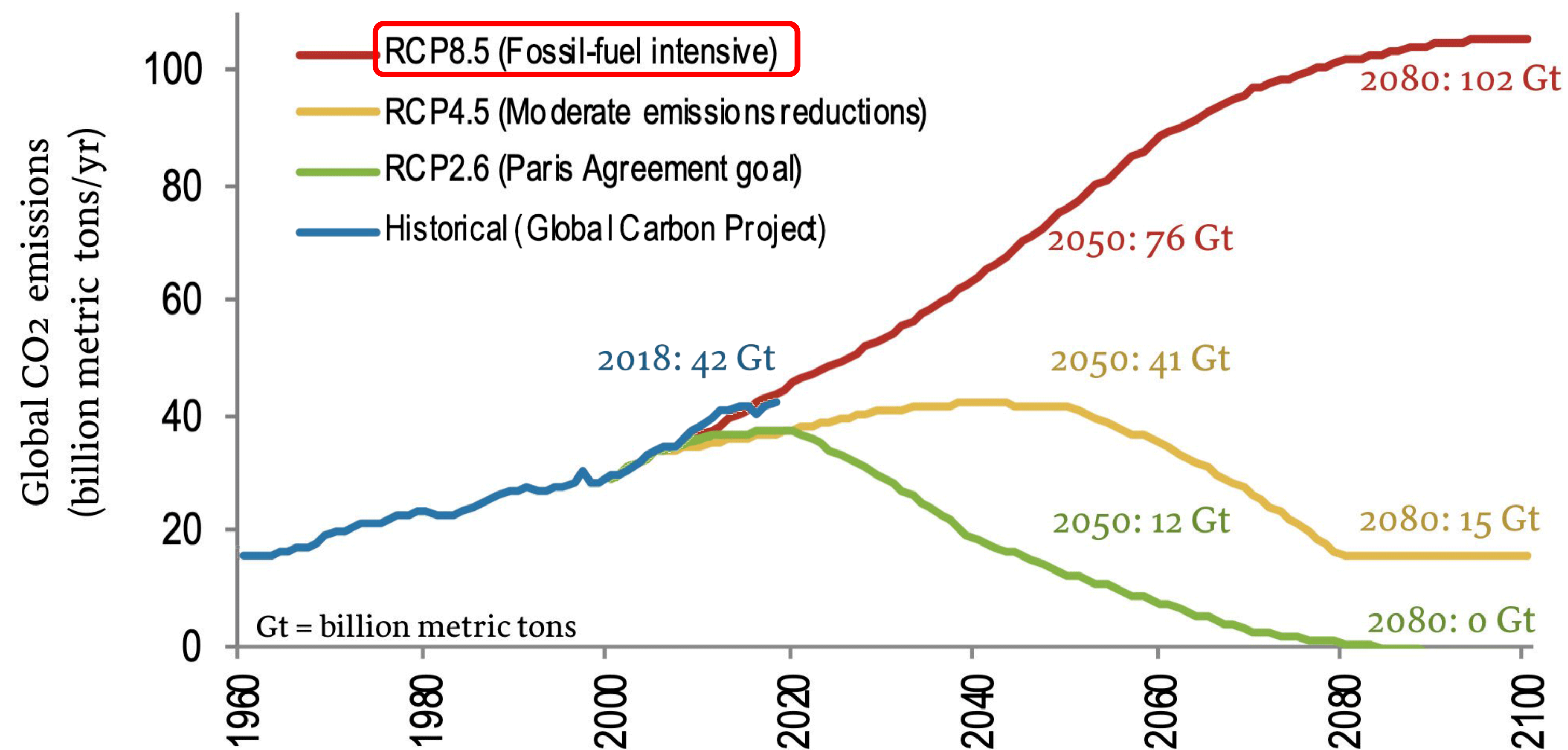
- Testidee 1
- Testidee 2
- Testidee 3
- ...



Property-Driven Development

by Example

CO₂ Emissionen



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

Quellen und weitere Details: www.ClimateFactsNow.org

Kata: CO₂-Budget

- Wie lange können wir noch CO₂ in die Luft blasen, um mit einer Wahrscheinlichkeit von 66% unter 2 Grad Erderwärmung zu bleiben?
- Stand 2018:
 - ▶ Restbudget 420 Gt CO₂
 - ▶ Jährlicher Ausstoß ca. 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`


```
class C02BudgetSpec {  
    @Example  
    void initialBudgetIsZero() {  
        assertEquals(0, C02Budget.remainingYears(0, 42, 4));  
    }  
}
```

```
public class C02Budget {  
    static int remainingYears(int initialBudget, int startingAnnualEmission, int annualChange) {  
        return 0;  
    }  
}
```

```

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

```

Run: CO2BudgetSpec.initialBudgetIsZero x

>> 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

Behalten wir das Beispiel?

@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 C02BudgetSpec...
    @Group
    class WithoutAnnualChange {
        @Example
        void budgetIsUsedUpExactly() {
            assertEquals(10, C02Budget.remainingYears(100, 10, 0));
        }
    }
}
```

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

@Example

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

@Property

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

@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));  
}
```

▶

✓

⊘

↓₂

↓_≡

≡

÷

↑

↓

🔍

↶

↷

⚙️

🚩

🔄

📄

📷

📤

📊

🚀

Tests failed: 1 of 1 test – 112 ms

Test Results112 ms

CO2BudgetSpec112 ms

WithoutAnnualChange112 ms

budgetIsUsedUpExactly112 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]

```
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	timestamp = 2019-11-05T09:00:06.541, WithoutAnnualChange:budgetIsUsedUpExactly =
WithoutAnnualChange	107 ms	org.opentest4j.AssertionFailedError: expected: <2> but was: <-2>
budgetIsUsedUpExactly	107 ms	<pre> -----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] </pre>

@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));
}

```


▶	<div> <div>✓</div> <div>⊘</div> <div>↓^a</div> <div>↓≡</div> <div>≡</div> <div>÷</div> <div>↑</div> <div>↓</div> <div>🕒</div> <div>↶</div> <div>↷</div> <div>⚙️</div> </div>	<div> <div>✓</div> <div>Tests passed: 3 of 3 tests – 202 ms</div> </div>
🔍	<div> <div>▼</div> <div>✓</div> <div>Test Results</div> </div>	<div> <div>202 ms</div> <div>/Library/Java/JavaVirtualMachines</div> </div>
🔄	<div> <div>▼</div> <div>✓</div> <div>CO2BudgetSpec</div> </div>	<div> <div>202 ms</div> <div>objc[1613]: Class JavaLaunchHelp</div> </div>
■	<div> <div>✓</div> <div>initialBudgetIsZero</div> </div>	<div> <div>37 ms</div> <div>↪ (0x106d974e0). One of the two w</div> </div>
📷	<div> <div>▼</div> <div>✓</div> <div>WithoutAnnualChange</div> </div>	<div> <div>165 ms</div> <div>timestamp = 2019-11-05T09:06:15.2</div> </div>
➡	<div> <div>✓</div> <div>budgetIsUsedUpExactly</div> </div>	<div> <div>43 ms</div> <div></div> </div>
—	<div> <div>✓</div> <div>budgetIsUsedUpExactly</div> </div>	<div> <div>122 ms</div> <div>tries = 1000</div> </div>

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`

```

@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));
}

```

```

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

```

```

@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));
}

```

```

[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`

```
@Group
class WithAnnualChange {
    @Example
    void budgetIsUsedUpWithIncrease() {
        assertEquals(4, C02Budget.remainingYears(100, 20, +5));
    }
}
```

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

```
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;  
}
```

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

```
@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 = ???; // irgendwas mit der Mitternachtsformel
    assertEquals(
        remainingYears,
        CO2Budget.remainingYears(initialBudget, startingAnnual, annualIncrease)
    );
}
```

Zu kompliziert und nicht intuitiv!

Muster- und Strategien für gute 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()`

- Wenn wir `annualChange` vergrößern, dann **nehmen** die verbleibenden Jahre **ab** oder **bleiben gleich**
- Wenn wir `annualChange` so groß wie `startingAnnual` machen, dann nehmen die verbleibenden Jahre um mindestens 1 ab; das Minimum bleibt aber 1

```
@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;
}
```

Generiere Werte mit Abhängigkeiten

- `initialBudget` zwischen 1 und 1000000
- `startingAnnual` zwischen 1 und 2 x `initialBudget`
- `annualChange` zwischen 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(1, 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]
```

```
tries = 84  
checks = 84  
seed = -5778725557855097949  
sample = [(251,9,5), 1]
```

	-----jqwik-----
	# of calls to property
	# of not rejected calls
	random seed to reproduce generated values

```
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`


```

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

```

✓ Tests passed: 6 of 6 tests – 300 ms

Test Results	300 ms
✓ CO2BudgetSpec	300 ms
✓ initialBudgetIsZero	36 ms
✓ WithAnnualChange	183 ms
✓ increasingAnnualChangeCanOnlyDecreaseRem	177 ms
✓ budgetIsUsedUpWithIncrease	5 ms
✓ budgetIsUsedUpDespiteDecrease	1 ms
✓ WithoutAnnualChange	81 ms
✓ budgetIsUsedUpExactly	1 ms
✓ budgetIsUsedUp	80 ms

/Library/Java/JavaVirtualMachi
objc[2373]: Class JavaLaunchHe
(0x10e8644e0). One of the two
timestamp = 2019-11-05T11:41:5
tries = 1000
checks = 1000
generation-mode = RANDOMIZED
after-failure = SAMPLE FIRST

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`

@Example

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

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

Ideen für Properties

- Wenn CO2-Budget nicht ausläuft, dann ändert auch eine Verringerung von `annualStartingEmission` nichts daran
- Wenn `initialBudget > 0`,
`annualStartingEmission = 0` und `annualChange <= 0`,
dann läuft das CO2-Budget nie aus
- Wenn `initialBudget > 0`,
`annualStartingEmission < initialBudget` und
`annualChange <= annualStartingEmission`,
dann läuft das CO2-Budget nie aus

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`

Tipps für Property-Driven Development

- Starte mit einem Beispiel...
... dann **verallgemeinere das Beispiel** zu einer Property
- Properties **dürfen "schwächer" sein** als eine vollständige fachliche Spezifikation
- Verwende die bekannten **Muster für gute Properties**
- Erstelle zusätzliche Properties, um die Implementierung auf **Stabilität** und **"unknown Unknowns"** zu testen
- Sammle Properties **schon beim Erstellen** der Inbox

Slides:

[http://johanneslink.net/downloads/
PropertyDrivenDevelopment.pdf](http://johanneslink.net/downloads/PropertyDrivenDevelopment.pdf)

Blog Series:

[http://blog.johanneslink.net/2018/03/24/
property-based-testing-in-java-introduction/](http://blog.johanneslink.net/2018/03/24/property-based-testing-in-java-introduction/)

Feedback

