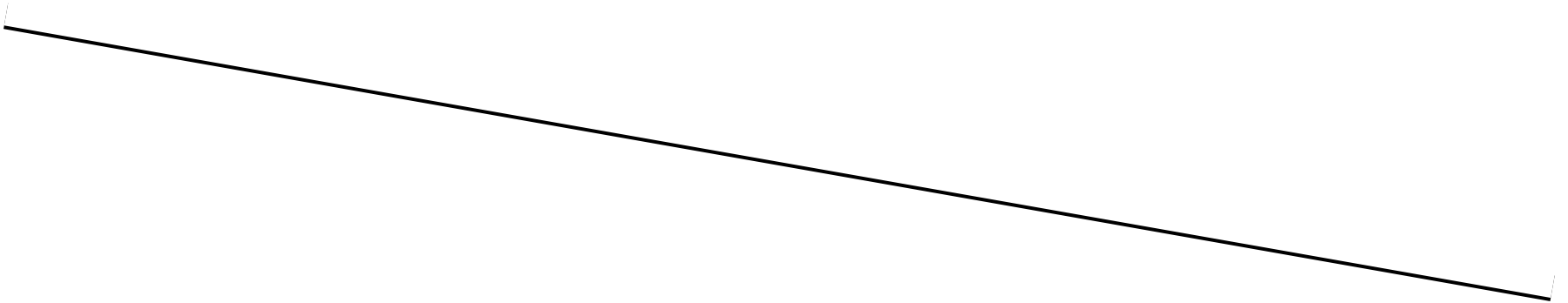




And Java becomes more functional with  
~~javaslang!~~







# Functional Programming

What does it mean?

# No more side effects!

Functions are first-class citizens

Immutability

Expressiveness

Safety

# Referential Transparency

```
int globalValue = 0;
// No Referential Transparency
int rq(int x) {
    globalValue++;
    return x + globalValue;
}
//Referential Transparency
int rt(int x) {
    return x + 1;
}
```

# Referential Transparency

## Other example

```
// No Referential Transparency
```

```
Math.random();
```

```
//Referential Transparency
```

```
Math.max(5, 42);
```

# Java 8 & Functional Programming

Lambda

Stream

Optional

Functional Interface

Function composition

Collections interfaces review





**There's still stuff missing**

# Who I am ?



Twitter : [@glours](#)

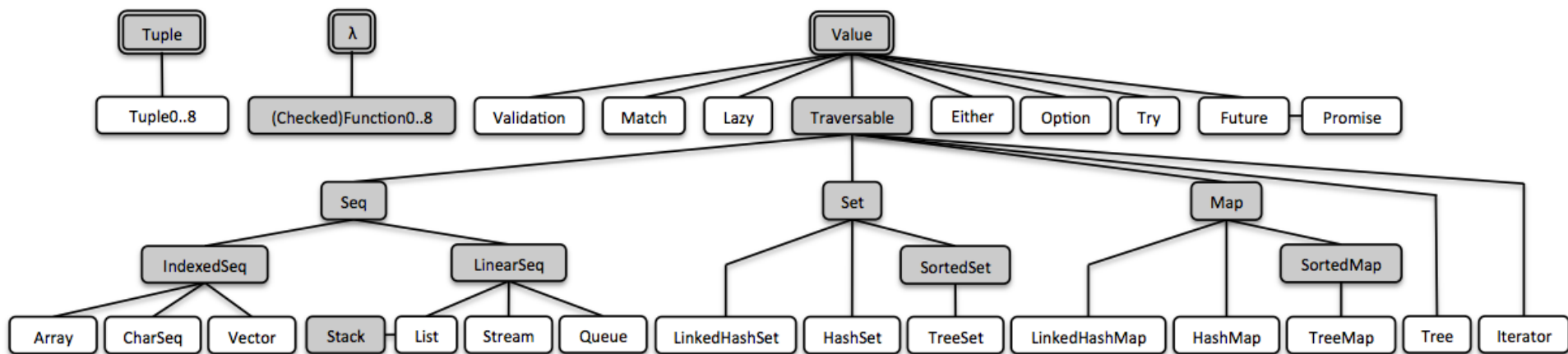
[Saagie](#) 'Frozen' Team Leader

Java and Javascript developer

Team member of Devovx France

So ...





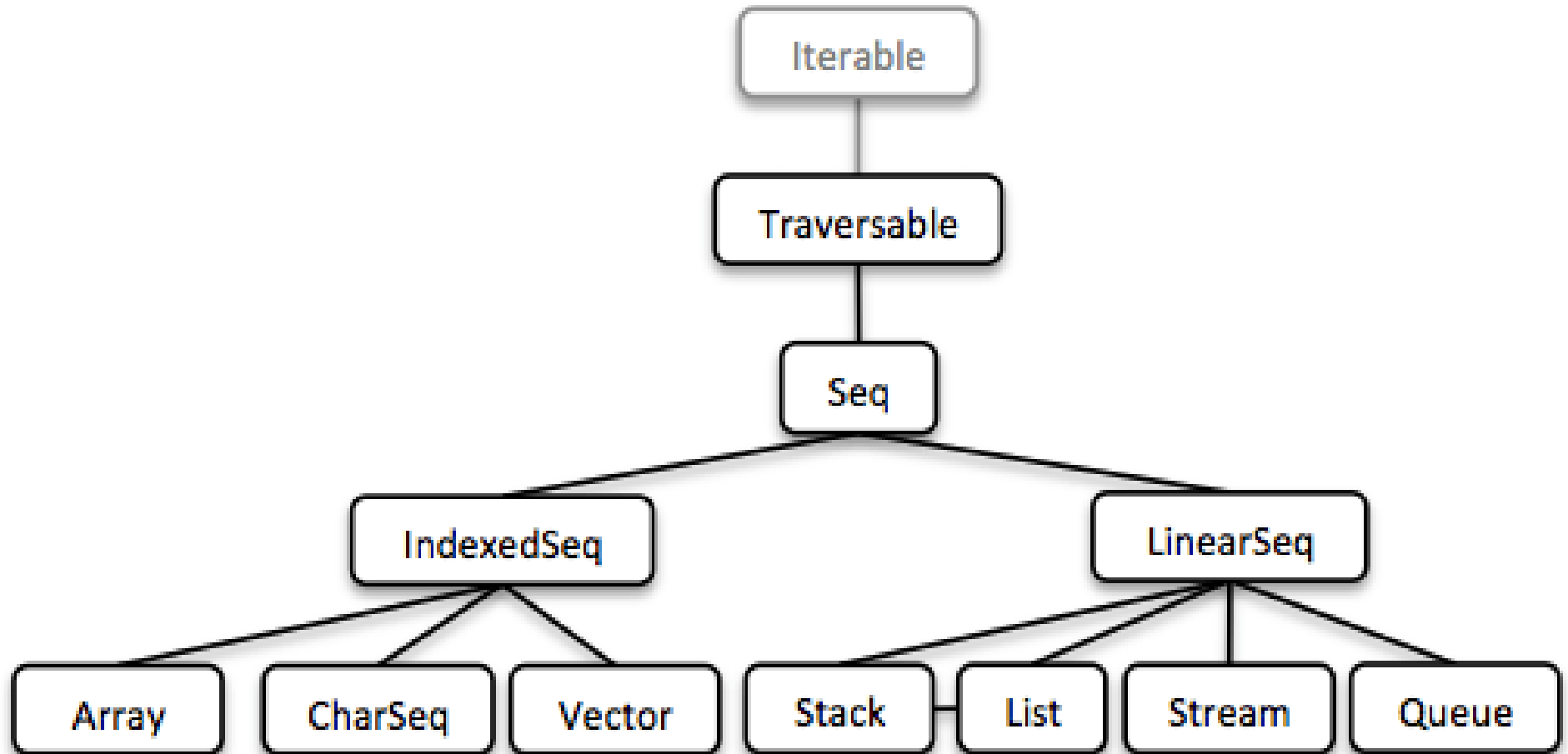


# It's already in JDK 8 !

Humm....

```
public List<Integer> unmodifiableListJdk8() {  
    List<Integer> jdkList = IntStream.range(0,20)  
        .boxed()  
        .collect(Collectors.toList());  
    return Collections.unmodifiableList(jdkList);  
}  
  
@Test  
public void should_verify_behaviour_of_jdk_8_unmodifiable_list()  
    List<Integer> unmodifiableList = this.examples.unmodifiab  
    unmodifiableList.add(21);  
    assertThat(unmodifiableList).hasSize(21);  
}
```

# List, Array, Stream ...



# Java 8 example

```
java.util.List<User> filterUserWithValidEmail() {  
    return this.usersJdk8  
        .stream()  
        .filter(user -> {  
            try {  
                return user.isEmailValid();  
            } catch (InvalidFormatException e) {  
                return false;  
            }  
        })  
        .collect(Collectors.toList());  
}
```



```
public List<User> filterUserWithValidAddressVavr() {
    return this.usersVavr
        .filter(user -> Try.of(() -> user
            .isAddressValid())
            .getOrElse(false));
}
```

# An other Java 8 example

```
public java.util.List<Address> filterInvalidAddressFromRouenJdk8(  
    return this.usersJdk8.stream()  
        .filter(User::isInvalidAddressFromRouen)  
        .map(User::getAddress)  
        .collect(Collectors.toList());  
}
```

# And With Vavr

```
public List<Address> filterInvalidAddressFromRouenVavr() {  
    return this.usersVavr  
        .filter(User::isInvalidAddressFromRouen)  
        .map(User::getAddress);  
}
```

# Stream consistency

## Java 8

```
public java.util.stream.Stream<String> mapUserToLowerCaseUserName(  
    java.util.stream.Stream<User> userNameStream = java.util.  
        userNameStream.map(user -> user.getUserName()  
                            .toUpperCase());  
  
    return userNameStream  
        .map(user -> user.getUserName().toLowerCase());  
}
```

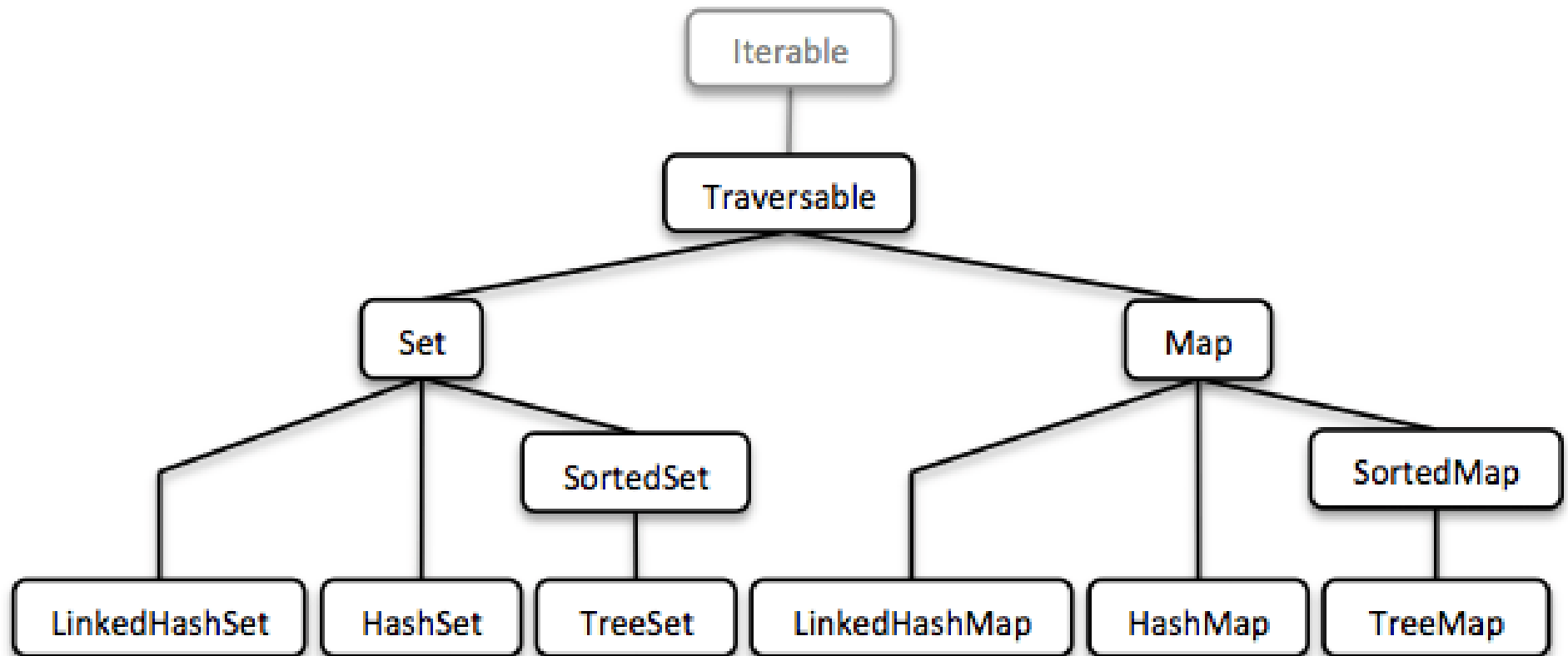
```
java.lang.IllegalStateException: stream has already been operated  
at java.util.stream.AbstractPipeline.<init>(AbstractPipeline.java  
at java.util.stream.ReferencePipeline.<init>(ReferencePipeline.ja  
at java.util.stream.ReferencePipeline$StatelessOp.<init>(Referenc  
at java.util.stream.ReferencePipeline$3.<init>(ReferencePipeline.  
at java.util.stream.ReferencePipeline.map(ReferencePipeline.java:
```

# Stream consistency

## Vavr

```
public Stream<String> mapUserToLowerCaseUserNameWithVavrStream()  
    Stream<User> userNameStream = Stream.of(generateUser(7),  
        userNameStream.map(user -> user.getUserName()  
                        .toUpperCase()));  
  
    return userNameStream  
        .map(user -> user.getUserName().toLowerCase());  
}
```

# Map, Set ...



# Tuple

javaslang

## Interface Tuple

All Known Implementing Classes:

`Tuple0`, `Tuple1`, `Tuple2`, `Tuple3`, `Tuple4`, `Tuple5`, `Tuple6`, `Tuple7`, `Tuple8`

# Java 8 example

```
public java.util.Map<String, User> filterMapOfUserWithValidAddress() {
    java.util.Map<String, User> usersMap = this.usersToMapJdk();
    return usersMap.entrySet().stream()
        .filter(entry -> {
            try {
                return entry.getValue().isAddressValid();
            } catch (InvalidFormatException e) {
                return false;
            }
        })
        .collect(Collectors.toMap((entry -> entry.getKey(),
```



# With Vavr

```
public Map<String, User> filterMapOfUserWithValidAddressVavr() {  
    Map<String, User> userMap = this.usersToMap();  
    return userMap  
        .filter(tuple -> Try.of(() ->  
            tuple._2.isValidAddress()).getOrElse(false))  
}
```

# Value Types

Option

Try

Either

Validation

(Lazy, Future, Promise, Match)

# Option

## 1 Interface

### Class Option.None<T>

`java.lang.Object`  
`javaslang.control.Option.None<T>`

#### Type Parameters:

T - The type of the optional value.

#### All Implemented Interfaces:

`Serializable`, `Iterable<T>`, `Option<T>`, `Value<T>`

### Class Option.Some<T>

`java.lang.Object`  
`javaslang.control.Option.Some<T>`

#### Type Parameters:

T - The type of the optional value.

#### All Implemented Interfaces:

`Serializable`, `Iterable<T>`, `Option<T>`, `Value<T>`

# Java 8 Optional

```
public Optional<Address> optionalOfUserAddress(String userName)
    if( !this.usersJdk8.containsKey(userName)) {
        return Optional.empty();
    }
    return Optional.of(this.usersJdk8.get(userName).getAddress()
}

public Optional<Address> optionalOfNullUsageForUserAddress(String
    return Optional.ofNullable(this.usersJdk8.get(userName))
        .map(user -> user.getAddress());
}
```

# Vavr Option

```
public Option<Address> optionOfUserAddress(String userName) {  
    return this.usersVavr.get(userName)  
        .map(user -> user.getAddress());  
}  
/* Because Map.get(key) return an Option in Vavr  
Option<V>get(K var1);  
*/
```

# Try

May return an exception or a successful result

```
public Try tryOfUserAddress(String userName) {  
    return Try.of(() -> this.usersVavr.get(userName)  
                .get().getAddressIfValid());  
}
```



# Either Example

```
public Either<String, User> eitherOfUser(String userName) {  
    return this.usersVavr.get(userName)  
        .toRight("Not Found");  
}
```





Example available on github

GitHub: <https://goo.gl/BHcYRJ>





*In fact Vavr functional interfaces are Java  
8 functional interfaces on steroids*

**Vavr documentation**



# Composition example

```
private Function1<Option<User>, Option<String>> lastName
    = user -> user.map(exist -> exist.getLastName());

private Function1<Option<String>, Option<String>> toUpperCase
    = value -> value.map(string -> string.toUpperCase());

private Function1<Option<User>, Option<String>> lastNameInUpperCa
    = lastName.andThen(toUpperCase);

public String userLastNameToUpperCase(String userName) {
    return lastNameInUpperCa
        .apply(usersJavlang.get(userName))
        .getOrElse("User Not Found");
}
```





# Lifting example

```
private Function1<Option<User>, Address> getAddressWithSideEffect
    = user -> user.get().getAddress();

private Function1<Option<User>, Option<Address>> safeGetAddress
    = Function1.lift(getAddressWithSideEffect);

public Address sideEffectGetAddress(String userName) {
    return getAddressWithSideEffect
        .apply(usersJavlang.get(userName));
}

public Option<Address> safeGetAddress(String userName) {
    return safeGetAddress
        .apply(this.usersVavr.get(userName));
}
```



# Partial Application example

```
private Function4 <Integer,Integer,Integer,Integer,Integer> sum
    = (a, b, c, d ) -> a + b + c + d;

private Function1<Integer, Integer> partialApplicationFunc
    = sum.apply(1,2,3);

public int partialApplication(int val) {
    return partialApplicationFunc.apply(val);
}
```



# Currying example

```
private Function1<Integer, Function1<Integer, Function1<Integer,  
    = sum.curried().apply(5);  
  
public int currying(int val1, int val2, int val3) {  
    return curriedFunc.apply(val1).apply(val2).apply(val3);  
}
```



# Memoization example

```
private Function0>Double< memoizedRandom = Function0.of(Math::ran
public double memoize() {
    return memoizedRandom.apply();
}

@Test
public void should_use_memoization_to_add() throws Exception {
    double firstCall = this.examples.memoize();

    assertThat(List.range(0, 20)
        .map(val -< this.examples.memoize()))
        .allMatch(val -< val == firstCall);
}
```





# Pattern Matching, the Vavr way

`$()` - wildcard pattern

`$(value)` - equals pattern

`$(predicate)` - conditional pattern

# Syntactic Sugar

## Predicate

```
Case(is(1), "one")
```

## Multiple conditions

```
Case(isIn("-h", "--help"), ...)
```

## User-Defined Patterns, Guards ...

# Pattern Matching example

```
Function0<Option<String>> usageDocumentation = () -> Option.of("
Function1<Option<String>, Option<String>> versionDocumentation =
Function0<Option<String>> helpDocumentation = usageDocumentation.
Function2<Option<String>, Option<String>, Option<String>> invalid

public static void main( String[] args )
{
    Option<String> arg = Array.of(args).headOption();
    String commandDescription = API.Match(arg.getOrElse("")).of(
        Case(isIn("-h", "--help"), helpDocumentation.apply())
        Case(isIn("-v", "--vesion"), versionDocumentation.app
        Case($(), invalidCommand.apply(helpDocumentation.appl
    ).getOrElse("Error when parsing argument");
    System.out.println(commandDescription);
}
```

# Another example

```
public List<Either<Address, Address>> patternMatchingList() {  
    List<User> users = this.usersVavr.map(tuple -> tuple._2).  
    return users.map(user ->  
        Match(validateUser(Option.of(user))).of(  
            Case(Valid($()), validUser -> Either.right(validUser)  
            Case(Invalid($()), errorList -> Either.left(user.get  
        ));  
}
```



# Property Checking example

```
@Test
public void should_133t_string() throws Exception {
    Arbitrary<String> leetCharEto3 = Arbitrary.string(Gen.fr
        Tuple.of(1, Gen.choose('A', 'Z')),
        Tuple.of(1, Gen.choose('a', 'z')))
    ))
    .filter(s -> s.length() > 10)
    .filter(s -> s.matches("\\w*[eE]+\\w*"));

    Function1<String, String> transformETo3 =
        s -> s.replaceAll("[eE]", "3");

    CheckedFunction1<String, Boolean> checkTransformETo3 =
        s -> transformETo3.apply(s)
            .matches("\\w*[^eE]+\\w*")
            && transformETo3.apply(s).contains("3").
```

# Vavr Modules

## Vavr Modules

---

There are several additional modules that group around Vavr's *core*. All of these are supported by the Vavr open source organization.



### vavr-gwt

GWT module for Vavr.



### vavr-match

Adds compile time support for Vavr's structural pattern matching feature.



### vavr-test

A property check framework for random testing of program properties.



### vavr-jackson

Jackson datatype module, the standard JSON library for Java.



### vavr-render

A rendering library, currently housing tree renderers (ascii and graphviz).

# Vavr Links

Documentation: <https://goo.gl/dMKKjN>

Javadoc: <https://goo.gl/PHQ81g>

Source code : <https://goo.gl/p3ivLM>



A photograph of a man in a white lab coat and glasses, smiling and looking down. He is in the foreground. In the background, a woman in a red shirt is partially visible, looking towards the camera. The setting appears to be a laboratory or medical office, with a window with blinds on the left and a sink area on the right.

**Thank You**

# Questions ?

source code: <https://goo.gl/BHcYRJ>