Fuzion — Java developer’s intro

Mapping Java's Features to Simpler Mechanisms

Fridtjof Siebert
Tokiwa Software GmbH

FOSDEM, 5. Feb 2023, Brussels
Who is this guy?

Fridtjof Siebert

Email: siebert@tokiwa.software
github: fridis
twitter: @fridi_s

'90-'94 AmigaOberon, AMOK PD
'97 FEC Eiffel Sparc / Solaris
'98-'99 OSF: TurboJ Java Compiler
'00-'01 PhD on real-time GC
'02-'19 JamaicaVM real-time JVM based on CLASSSPATH / OpenJDK,

VeriFlux static analysis tool

'20-... Fuzion
'21-... Tokiwa Software
Motivation

John Backus:

[My] work in functional programming languages failed, and would likely always fail, because it was easy to do hard things but incredibly difficult to do simple things.

Source: Grady Booch, Twitter
https://twitter.com/Grady_Booch/status/1016041695501139968
Motivation: Fuzion Language

Many languages overloaded with concepts like classes, methods, interfaces, constructors, traits, records, structs, packages, values, ...

➡️ Fuzion has one concept: a feature

Today’s compilers and tools are more powerful

➡️ Tools make better decisions

Systems are safety-critical

➡️ we need to ensure correctness
Fuzion Resources

Fuzion available

→ sources: github.com/tokiwa-software/fuzion
Fuzion Resources

Fuzion available

➡ sources: github.com/tokiwa-software/fuzion
➡ Website: flang.dev
  • tutorial
  • design
  • examples
  • ...
Backing Company

→ supports development of Fuzion
→ currently four employees
→ hiring
→ searching for funding
This Talk

Fuzion and Algebraic Effects

- Quick Fuzion Intro
- Dangers of side-effects
- Algebraic Effects
- Examples / defining your own Effect / more examples
Short Fuzion Language Intro

Everything is a feature     Java equivalent
Short Fuzion Language Intro

Everything is a feature

Java equivalent

```java
package demo;

demo is
```
Short Fuzion Language Intro

Everything is a feature

demo is
  hello is

Java equivalent

package demo;
class hello {

}

FOSDEM'23: Fuzion: Java developer's intro
Short Fuzion Language Intro

Everything is a feature

demo is
hello is
greet unit is

Java equivalent

```java
package demo;
class hello {
    void greet() {
    }
}
```
Short Fuzion Language Intro

Everything is a feature

demo is
hello is
  greet unit is

Java equivalent

package demo;
class hello {
  void greet() {
  }
}
Short Fuzion Language Intro

Everything is a feature

demo is
  hello is
    greet unit is

Java equivalent

```java
package demo;

class hello {
  void greet() {
  }
}
```
Short Fuzion Language Intro

Everything is a feature

demo is
  hello is
    greet unit is
      say "Hello World!"

Java equivalent

package demo;

class hello {
  void greet() {
    System.out.println("Hello World!");
  }
}
Everything is a feature

demo is
hello is
greet unit is
say "Hello World!"

Java equivalent

```java
package demo;
class hello {
    void greet() {
        System.out.println("Hello World!");
    }
}
```
Short Fuzion Language Intro

Everything is a feature

demo is
  hello is
    greet ⇒
      say "Hello World!"

Java equivalent

```java
package demo;

class hello {
  void greet() {
    System.out.println("Hello World!");
  }
}
```
Short Fuzion Language Intro

Everything is a feature

demo is
  hello is
    greet ⇒
    say "Hello World!"

Java equivalent

```java
package demo;

class hello {
  void greet() {
    System.out.println("Hello World!");
  }
}
```
Short Fuzion Language Intro

Everything is a feature

demo is
    hello is
        greet ⇒
            say "Hello World!"

demo.hello.greet

Java equivalent

```java
package demo;

class hello {
    void greet() {
        System.out.println("Hello World!");
    }
}

class universe {
    public static void main(String[] args) {
        new demo.hello().greet();
    }
}
```
Short Fuzion Language Intro

Everything is a feature

demo is
hello is
greet(a String) ⇒
say "Hello $a!"

demo.hello.greet "World"

Java equivalent

```java
package demo;
class hello {
    void greet(String a) {
        System.out.println("Hello "+a+"!");
    }
}

class universe {
    public static void main(String[] args) {
        new demo.hello.greet("World");
    }
}
```

FOSDEM'23: Fuzion: Java developer's intro
Short Fuzion Language Intro

Everything is a feature

demo is
   hello is
      greet(a String) ⇒
         say "Hello $a!"

demo.hello.greet "World"

Java equivalent

```java
package demo;

class hello {
   void greet(String a) {
      System.out.println("Hello "+a+"!");
   }
}

class universe {
   public static void main(String[] args) {
      new demo.hello.greet("World");
   }
}
```
Short Fuzion Language Intro

Everything is a feature

demo is
  hello is
    greet(a String) ⇒
      say "Hello $a!"
  hello2(def String) : hello is
    run ⇒ greet def

h := demo.hello2 "World"
h.run

Java equivalent

```java
package demo;

class hello {
    void greet(String a) {
        System.out.println("Hello "+a+");
    }
}

class hello2 extends hello {
    String def;
    hello2(String d) { def = d; }
    void run() { greet(def); }
}

class universe {
    public static void main(String[] args) {
        var h = new demo.hello2("World");
        h.run();
    }
}
```

FOSDEM’23: Fuzion: Java developer’s intro
Short Fuzion Language Intro

Everything is a feature

demo is

hello is

greet(a String) ⇒
say "Hello $a!"

hello2(def String) : hello is

run ⇒ greet def

h := demo.hello2 "World"

Java equivalent

```java
package demo;

class hello {
    void greet(String a) {
        System.out.println("Hello "+a+"!");
    }
}
class hello2 extends hello {
    String def;
    hello2(String d) { def = d; }
    void run() { greet(def); }
}
class universe {
    public static void main(String[] args) {
        var h = new demo.hello2("World");
        h.run();
    }
}
```
What does Fuzion not have?

Capabilities considered harmful:

- Dynamic Loading
- Macros
- Reflection
- Pointer Arithmetic
- (uncontrolled) Mutability
- Exceptions

Reasons:

- We must know what code does
- Static Analysis
- Safety
- Performance
(Side-) Effects and Safety / Security
(Side-) Effects and Safety / Security

Recent security alerts
(Side-) Effects and Safety / Security

Recent security alerts

➡ log4shell
(Side-) Effects and Safety / Security

Recent security alerts

- log4shell
- SpringShell
(Side-) Effects and Safety / Security

Recent security alerts

→ log4shell
→ SpringShell
→ rustdecimal crate
(Side-) Effects and Safety / Security

Recent security alerts

- log4shell
- SpringShell
- rustdecimal crate

Common problem?
(Side-) Effects and Safety / Security

Recent security alerts

- log4shell
- SpringShell
- rustdecimal crate

Common problem

- Code has unexpected (side-) effects

FOSDEM’23: Fuzion: Java developer’s intro
Algebraic Effects
Algebraic Effects

An Algebraic Effect is

→ a set of (non-functional) operations code may perform
Algebraic Effects

An Algebraic Effect is

→ a set of (non-functional) operations code may perform
  • Java has one effect: throws with one operation throw
An Algebraic Effect is

→ a set of (non-functional) operations code may perform
Algebraic Effects

An Algebraic Effect is

- a set of (non-functional) operations code may perform
- the operations can resume or abort
Algebraic Effects

An Algebraic Effect is

- a set of (non-functional) operations code may perform
- the operations can resume or abort
- the operations can be implemented by an effect handler
Algebraic Effects

An Algebraic Effect is

- a set of (non-functional) operations code may perform
- the operations can resume or abort
- the operations can be implemented by an effect handler
- Effects may be nested
Algebraic Effects

An Algebraic Effect is

- a set of (non-functional) **operations** code may perform
- the operations can **resume** or **abort**
- the operations can be implemented by an **effect handler**
- Effects may be **nested**
- Effects may be seen as required **capabilities**
  - code that throws exception requires capability to catch
Example: my_exc effect

Exception Effect
Example: my_exc effect

Exception Effect

my_exc : simpleEffect is
Example: my_exc effect

Exception Effect

my_exc : simpleEffect is
  throw ⇒ abort
Example: my_exc effect

Exception Effect

\[
\text{my_exc} : \text{simpleEffect is} \\
\quad \text{throw } \Rightarrow \text{abort}
\]

\[
f \Rightarrow \\
\text{my_exc.env.throw}
\]
Example: my_exc effect

Exception Effect

my_exc : simpleEffect is
  throw ⇒ abort

f! my_exc ⇒

  my_exc.env.throw
Exception Effect

my_exc : simpleEffect is
  throw ⇒ abort

f ! my_exc ⇒
  say "before throw"
  my_exc.env.throw
  say "after throw  *** not reachable ***"
Example: my_exc effect

Exception Effect

my_exc : simpleEffect is
  throw ⇒ abort

f ! my_exc ⇒
  say "before throw"
  my_exc.env.throw
  say "after throw  *** not reachable ***"

my_exc.use ()⇒f
Example: my_exc effect

Exception Effect

my_exc : simpleEffect is
  throw ⇒ abort

f ! my_exc ⇒
  say "before throw"
  my_exc.env.throw
  say "after throw  *** not reachable ***"

say "install my_exc"
my_exc.use ()→f
say "done with my_exc"
Example: my_exc effect

Exception Effect

my_exc : simpleEffect is
    throw ⇒ abort

f ! my_exc ⇒
    say "before throw"
    my_exc.env.throw
    say "after throw  *** not reachable ***"

say "install my_exc"
my_exc.use ()⇒f
say "done with my_exc"

> fz exception.fz
Example: my_exc effect

Exception Effect

```fz
my_exc : simpleEffect is	high => abort

f ! my_exc =>	say "before throw"
my_exc.env.throw	say "after throw  *** not reachable ***"
say "install my_exc"
my_exc.use ()→f	say "done with my_exc"
```

> fz exception.fz
install exc
before throw
done with exc
>
Fuzion and Mutation

Fields in Fuzion are immutable
Fuzion and Mutation

Fields in Fuzion are immutable

x := 123
say x
x := 2*x
say x
Fuzion and Mutation

Fields in Fuzion are immutable

\[ x := 123 \]
\[ \text{say } x \]
\[ x := 2 \times x \]
\[ \text{say } x \]

> fz mutate1.fz
Fields in Fuzion are immutable

```fz
x := 123
say x
x := 2*x
say x
```

```
> fz mutate1.fz
123
246
>
```
Fuzion and Mutation

Fields in Fuzion are immutable

```
show_x ⇒ say x
x := 123
show_x
x := 2*x
show_x
```
Fuzion and Mutation

Fields in Fuzion are immutable

```plaintext
  show_x ⇒ say x
  x := 123
  show_x
  x := 2*x
  show_x
```

> fz mutate2.fz
Fuzion and Mutation

Fields in Fuzion are immutable

```
show_x ⇒ say x
x := 123
show_x
x := 2*x
show_x
```

```
> fz mutate2.fz
mutable_fields2.fz:1:19:
  : error 1: Ambiguous call targets found for call to 'x' (no arguments)
    show_x ⇒ say x
------------------^
Found several possible targets that match this call:
```
Fuzion and Mutation

Fields in Fuzion are immutable.

```fz
> fz mutate2.fz
mutable_fields2.fz:1:19:: error 1: Ambiguous call targets found for call to 'x' (no arguments)
    show_x ⇒ say x
____________________^  

Found several possible targets that match this call:
'x' defined at mutable_fields2.fz:4:5:
    x := 123
____^  

and 'x' defined at mutable_fields2.fz:6:5:
    x := 2*x
_____
```

one error.
Fuzion and Mutation

Fields in Fuzion are immutable

```java
show_x ⇒ say x

x := 123
show_x
x := 2*x
show_x
```
Fuzion and Mutation

Fields in Fuzion are immutable

```
show_x => say x
x := mut 123
show_x
x ← 2 * x.get
show_x
```
Fuzion and Mutation

Fields in Fuzion are immutable

```java
show_x ⇒ say x
x := mut 123
show_x
x ← 2 * x.get
show_x
```
Fuzion and Mutation

Fields in Fuzion are immutable

```plaintext
show_x => say x
x := mut 123
show_x
x ← 2 * x.get
show_x
```
Fuzion and Mutation

Fields in Fuzion are immutable

```fz
show_x ⇒ say x
x := mut 123
show_x
x ← 2 * x.get
show_x
```

```bash
> fz mutate3.fz
123
246
>
```
Fuzion and Mutation

Fields in Fuzion are immutable

show_x ⇒ say x
x := mut 123
show_x
x ← 2 * x.get
show_x

> fz mutate3.fz
123
246
> fz -effects mutate3.fz
Fuzion and Mutation

Fields in Fuzion are immutable

```java
show_x => say x
x := mut 123
show_x
x ← 2 * x.get
show_x
```

```bash
> fz mutate3.fz
123
246
> fz -effects mutate3.fz
io.out
mutate
> 
```
Fuzion and Mutation

Loop index variables

for
  i := 0, i + 1
while i < 10 do
  say i
  say "done."
Fuzion and Mutation

Loop index variables

```fz
for
  i := 0, i + 1
while i < 10 do
  say i
say "done."
```

```bash
> loop.fz
```

FOSDEM'23: Fuzion: Java developer's intro
Fuzion and Mutation

Loop index variables

```java
for
  i := 0, i + 1
while i < 10 do
  say i
 say "done."
```
Fuzion and Mutation

Loop index variables

for
  i := 0, i + 1
while i < 10 do
  say i
  say "done."

> fz -effects loop.fz
Fuzion and Mutation

Loop index variables

```
for
    i := 0, i + 1
while i < 10 do
    say i
say "done."
```

> fz -effects loop.fz
> io.out
>
Fuzion and Mutation

Loop index variables

```java
for
    i := 0, i + 1
while i < 10 do
    say i
say "done."
```

No variable is mutated, a new instance is created per iteration.

```bash
> fz -effects loop.fz
io.out
>
done.
```
Error Handling
Error Handling

Division by zero

\[
\text{divide} (a, b \text{ i32}) \Rightarrow \\
a / b
\]
Error Handling

Division by zero

\[
\text{divide} \ (a, \ b \ i32) \Rightarrow \\
\quad a \ / \ b
\]

\[
\text{show\_div}(a, \ b \ i32) \Rightarrow \\
\quad v := \text{divide} \ a \ b \\
\quad \text{say} \ "\text{result is } v"
\]

show\_div 100 12
show\_div 100 0
show\_div 10 100
Error Handling

Division by zero

divide (a, b i32) ⇒
  a / b

show_div(a, b i32) ⇒
  v := divide a b
  say "result is $v"

show_div 100 12
show_div 100 0
show_div 10 100
FOSDEM’23: Fuzion: Java developer’s intro

Error Handling

Division by zero

\[
\text{divide} \ (a, \ b \ i32) \Rightarrow \\
\quad a / b
\]

\[
\text{show\_div}(a, \ b \ i32) \Rightarrow \\
\quad v := \text{divide} \ a \ b \\
\quad \text{say} \ "\text{result is} \ \$v"$
\]

\[
\begin{align*}
\text{show\_div} & 100 \ 12 \\
\text{show\_div} & 100 \ 0 \\
\text{show\_div} & 10 \ 100 
\end{align*}
\]
Error Handling

Division by zero

```fz
> fz div.fz
result is 8

$FUZION/lib/i32.fz:59:13: error 1: Precondition does not hold
    safety: other ≠ 0
    safety: other ≠ 0
    safety: other ≠ 0

For call to i32.infix /
Call stack:
divide: div.fz:2:9:
    a / b
    a / b

show_div: div.fz:8:12:
    v := divide a b
    v := divide a b

#universe: div.fz:13:5:
```

> show_div 100 0
result is 8

> show_div 100 0
Error 1: Precondition does not hold
```
Error Handling

Division by zero

divide (a, b i32) ⇒
a / b

show_div(a, b i32) ⇒
v := divide a b
say "result is $v"

show_div 100 12
show_div 100 0
show_div 10 100

> fz div.fz
result is 8

$FUZION/lib/
i32.fz:59:13: error 1:
Precondition does not hold
    safety: other ≠ 0
For call to i32.infix /
Using choice type `outcome`

Using outcome

```r
divide (a, b i32) ⇒
  a / b
```

```r
divide (a, b i32) ⇒
  v := divide a b
  say "result is $v"
```

```r
show_div 100 12
show_div 100 0
show_div 10 100
```
Using choice type outcome

Using outcome

\[
\text{divide } (a, b \ \text{i32}) \ \text{outcome i32 is}\n\]
\[
a / b
\]

\[
\text{show\_div}(a, b \ \text{i32}) \Rightarrow
\]
\[
v := \text{divide } a \ b
\]
\[
say \ "\text{result is } v"
\]

show\_div 100 12
show\_div 100 0
show\_div 10 100
Using choice type `outcome`

Using outcome

```rust
divide (a, b i32) outcome i32 is
  if b = 0 then
    error "div by 0!"
  else
    a / b

show_div(a, b i32) =>
  v := divide a b
  say "result is $v"
```

show_div 100 12
show_div 100 0
show_div 10 100
Using choice type outcome

Using outcome

divide (a, b i32) outcome i32 is
  if b = 0 then
    error "div by 0!"
  else
    a / b

show_div(a, b i32) ⇒
  v := divide a b
  say "result is $v"

show_div 100 12
show_div 100 0
show_div 10 100

> fz outcome_div0.fz
Using choice type outcome

Using outcome

```fz
divide (a, b i32) outcome i32 is
    if b = 0 then
        error "div by 0!"
    else
        a / b

show_div(a, b i32) =>
    v := divide a b
    say "result is $v"
```

```
> fz outcome_div0.fz
result is 8
result is --error: div by 0!--
result is 0
```

show_div 100 12
show_div 100 0
show_div 10 100
```
Using choice type outcome

Using outcome

\[
\text{divide (a, b i32) outcome i32 is}
\]
\[
\text{if } b = 0 \text{ then}
\]
\[
\text{error "div by 0!"}
\]
\[
\text{else}
\]
\[
a / b
\]

\[
\text{show_div(a, b i32) ⇒}
\]
\[
v := \text{divide a b}
\]
\[
\text{say "result is $v"}
\]

show_div 100 12
show_div 100 0
show_div 10 100
Using choice type outcome

Using outcome

```plaintext
divide (a, b i32) outcome i32 is
  if b = 0 then
    error "div by 0!"
  else
    a / b

show_div(a, b i32) ⇒
  match divide a b
    v i32 ⇒ say "ok, result is $v"
    e error ⇒ say "not ok: $e"
```

```
show_div 100 12
show_div 100 0
show_div 10 100
```
Using choice type `outcome`

Using `outcome`:

```fz
divide (a, b i32) outcome i32 is
  if b = 0 then
    error "div by 0!"
  else
    a / b

show_div(a, b i32) =>
  match divide a b
    v i32   => say "ok, result is $v"
    e error => say "not ok: $e"

show_div 100 12
show_div 100 0
show_div 10 100
```

> fz outcome_div.fz

FOSDEM'23: Fuzion: Java developer's intro
Using choice type outcome

Using outcome

\[
\text{divide} \ (a, \ b \ i32) \ \text{outcome} \ i32 \ \text{is} \\
\quad \text{if} \ b = 0 \ \text{then} \\
\quad \quad \text{error } "\text{div by 0!}" \\
\quad \text{else} \\
\quad \quad a \ / \ b \\
\]

\[
\text{show\_div}(a, \ b \ i32) \Rightarrow \\
\text{match} \ \text{divide} \ a \ b \\
\quad v \ i32 \Rightarrow \text{say} "\text{ok, result is } v" \\
\quad e \ \text{error} \Rightarrow \text{say} "\text{not ok: } e" \\
\]

\[
\text{show\_div} \ 100 \ 12 \\
\text{show\_div} \ 100 \ 0 \\
\text{show\_div} \ 10 \ 100
\]
### Throwing Errors using try-effect

**Using try-effect**

```racket
divide (a, b i32) outcome i32 is
  if b = 0 then
    error "div by 0!"
  else
    a / b

show_div(a, b i32) ⇒
  match divide a b
  v i32 ⇒ say "ok, result is $v"
  e error ⇒ say "not ok: $e"
```

show_div 100 12
show_div 100 0
show_div 10 100
Throwing Errors using try-effect

Using try-effect

```r
divide (a, b i32) outcome i32 is
  if b = 0 then
    error "div by 0!"
  else
    a / b

show_div(a, b i32) ⇒
  match divide a b
    v i32 ⇒ say "ok, result is $v"
    e error ⇒ say "not ok: $e"
```

show_div 100 12
show_div 100 0
show_div 10 100
```
Throwing Errors using try-effect

Using try-effect

```
divide (a, b i32) i32 ! try is
  if b = 0 then
    error "div by 0!"
  else
    a / b

show_div(a, b i32) =>
  match divide a b
    v i32 => say "ok, result is $v"
    e error => say "not ok: $e"
```

show_div 100 12
show_div 100 0
show_div 10 100
Thruing Errors usign try-effect

Using try-effect

\[
\text{divide} \ (a, \ b \ i32) \ i32 \ ! \ \text{try is}
\]
\[
\text{if} \ b = 0 \ \text{then}
\]
\[
\text{error} \ "\text{div by 0!}" \\
\text{else}
\]
\[
a / b
\]

\[
\text{show}_{\text{div}}(a, \ b \ i32) \Rightarrow
\]
\[
\text{match} \ \text{divide} \ a \ b
\]
\[
\text{v} \ i32 \ \Rightarrow \ \text{say} \ "\text{ok, result is $v"}
\]
\[
\text{e} \ \text{error} \ \Rightarrow \ \text{say} \ "\text{not ok: $e"}
\]

\[
\text{show}_{\text{div}} \ 100 \ 12
\]
\[
\text{show}_{\text{div}} \ 100 \ 0
\]
\[
\text{show}_{\text{div}} \ 10 \ 100
\]
Throwing Errors using try-effect

Using try-effect

```plaintext
divide (a, b i32) i32 ! try is
    if b = 0 then
        try.env.raise (error "div by 0!")
    else
        a / b

show_div(a, b i32) =>
    match divide a b
    v i32 => say "ok, result is $v"
    e error => say "not ok: $e"

show_div 100 12
show_div 100 0
show_div 10 100
```
Throwing Errors using try-effect

Using try-effect

```ml
divide (a, b i32) i32 ! try is
  if b = 0 then
    try.env.raise (error "div by 0!")
  else
    a / b

show_div(a, b i32) =>
  match divide a b
  v i32      => say "ok, result is $v"
  e error    => say "not ok: $e"
```

show_div 100 12
show_div 100 0
show_div 10 100
Throwing Errors using try-effect

Using try-effect

\[
\begin{align*}
\text{divide } (a, b \ i32) \ i32 & \ ! \ \text{try is} \\
& \quad \text{if } b = 0 \text{ then} \\
& \quad \quad \text{try.env.raise (error "div by 0!")} \\
& \quad a / b \\
\text{show_div}(a, b \ i32) \Rightarrow \\
& \quad \text{match divide a b} \\
& \quad \quad v \ i32 \Rightarrow \text{say "ok, result is $v"} \\
& \quad \quad e \ \text{error} \Rightarrow \text{say "not ok: $e"} \\
\text{show_div 100 12} \\
\text{show_div 100 0} \\
\text{show_div 10 100}
\end{align*}
\]
Throwing Errors using try-effect

Using try-effect

divide (a, b i32) i32 ! try is
    if b = 0 then
        try.env.raise (error "div by 0!")
    a / b

show_div(a, b i32) =>
    match divide a b
        v i32 => say "ok, result is $v"
        e error => say "not ok: $e"

show_div 100 12
show_div 100 0
show_div 10 100
Throwing Errors using try-effect

Using try-effect

```haskell
divide (a, b i32) i32 ! try is
    if b = 0 then
        try.env.raise (error "div by 0!")
    a / b

show_div(a, b i32) =>
    match try (() -> divide a b)
    v i32   => say "ok, result is $v"
    e error => say "not ok: $e"
```

show_div 100 12
show_div 100 0
show_div 10 100
Throwing Errors using try-effect

Using try-effect

\[
\text{divide } (a, b \text{ i32}) \text{ i32 } \text{ ! try is}
\]
\[
\text{if } b = 0 \text{ then}
\]
\[
\text{try.env.raise (error "div by 0!")}
\]
\[
a / b
\]

\[
\text{show_div}(a, b \text{ i32}) \Rightarrow
\]
\[
\text{match try } ((\rightarrow \text{ divide a b})
\]
\[
 v \text{ i32 } \Rightarrow \text{ say "ok, result is $v"}
\]
\[
e \text{ error } \Rightarrow \text{ say "not ok: $e"}
\]

\[
\text{show_div } 100 12
\]
\[
\text{show_div } 100 0
\]
\[
\text{show_div } 10 100
\]
Using try-effect

```fz
divide (a, b i32) i32 ! try is
    if b = 0 then
        try.env.raise (error "div by 0!")
    a / b

show_div(a, b i32) =>
    match try (() -> divide a b)
    v i32   => say "ok, result is $v"
    e error => say "not ok: $e"
```

> try_div.fz
ok, result is 8
not ok: error: div by 0!
ok, result is 0
>
Fuzion: Status

Fuzion still under development

- language definition slowly getting more stable
- base library work in progress
- current implementation providing JVM and C backends
- Basic analysis tools available
Conclusion

Fuzion is a new functional language

→ Java maps very well to Fuzion
→ effects encapsulate non-functional aspects
  • mutability
  • i/o
  • exceptions
→ have a look, get involved!

@fuzion@types.pl
@FuzionLang
https://flang.dev
github.com/tokiwa-software/fuzion