The Challenges of Running the

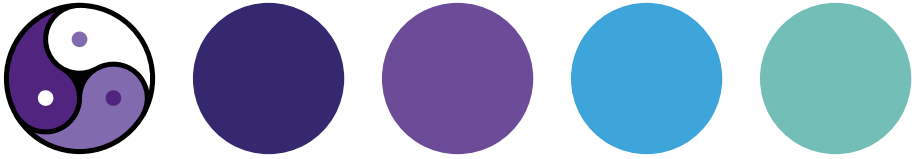
# Fuzion Language on OpenJDK

## Mapping a Functional Language to efficient Java Bytecode

Fridtjof Siebert
Tokiwa Software GmbH

FOSDEM 2024, 3. Feb 2024, Brussels

TOKIWA software



# Who is this guy?

Fridtjof Siebert


Email: siebert@tokiwa.software
github: fridis
Twitter etc. @fridi_s
@fridi@mastodon.social
@fridis.bsky.social
fridi_si@instagram


| | |
|---|---|
| ‘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 |

TOKIWA software



# Who is this guy?

Fridtjof Siebert


Email: siebert@tokiwa.software
github: fridis
Twitter etc. @fridi_s
@fridi@mastodon.social
@fridis.bsky.social
fridi_si@instagram


| | |
|---|---|
| ‘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 |

TOKIWA software



## The Challenges of Running the

# Fuzion Language on OpenJDK

overview

- ➡ Fuzion quick intro
- ➡ Tagged union types
- ➡ Product types with value semantics
- ➡ Type parameters
- ➡ Multiple Inheritance
- ➡ Classfile verifier

TOKIWA software



The Challenges of Running the

# Fuzion Language on OpenJDK

overview

- ➡ Fuzion quick intro
- ➡ Tagged union types
- ➡ Product types with value semantics
- ➡ Type parameters
- ➡ Multiple Inheritance
- ➡ Classfile verifier

TOKIWA software

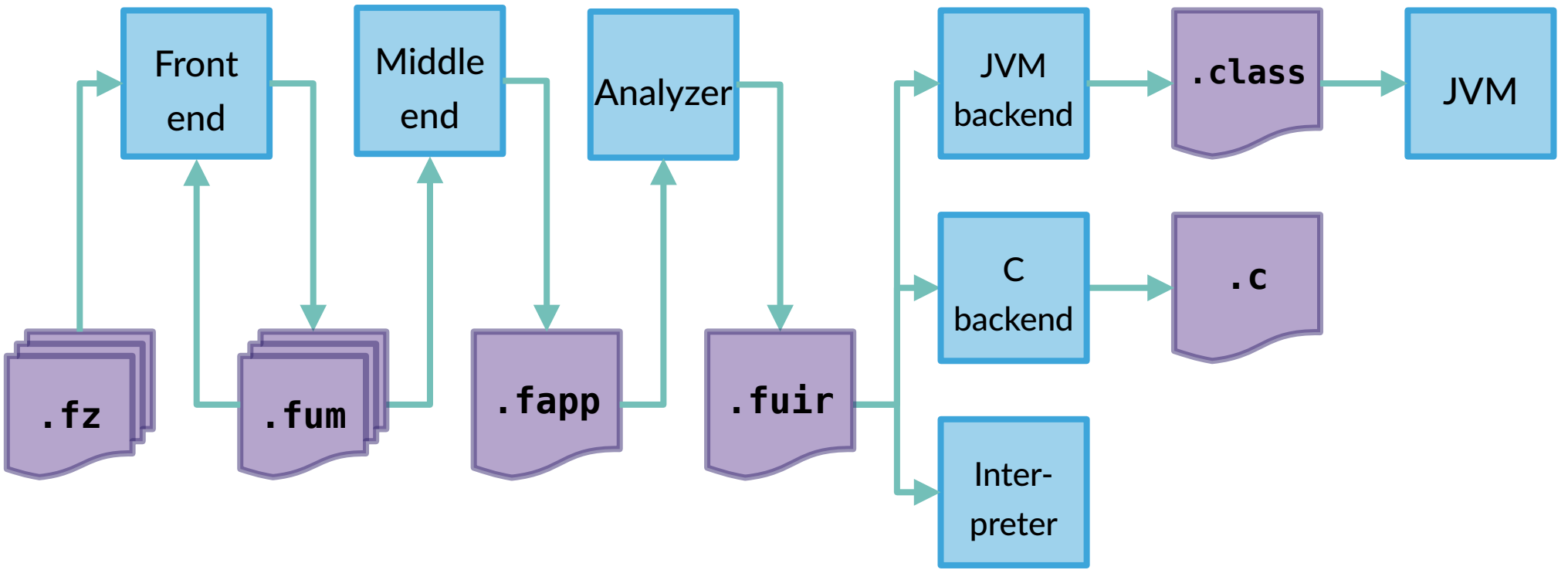


# Motivation: Fuzion Language

- ➡ One concept: a feature
- ➡ Systems are safety-critical
- ➡ Tools make developer's life easier
- ➡ Fuzion is
  - statically typed
  - polymorphic: union types, parametric types, inheritance
  - pure using effects



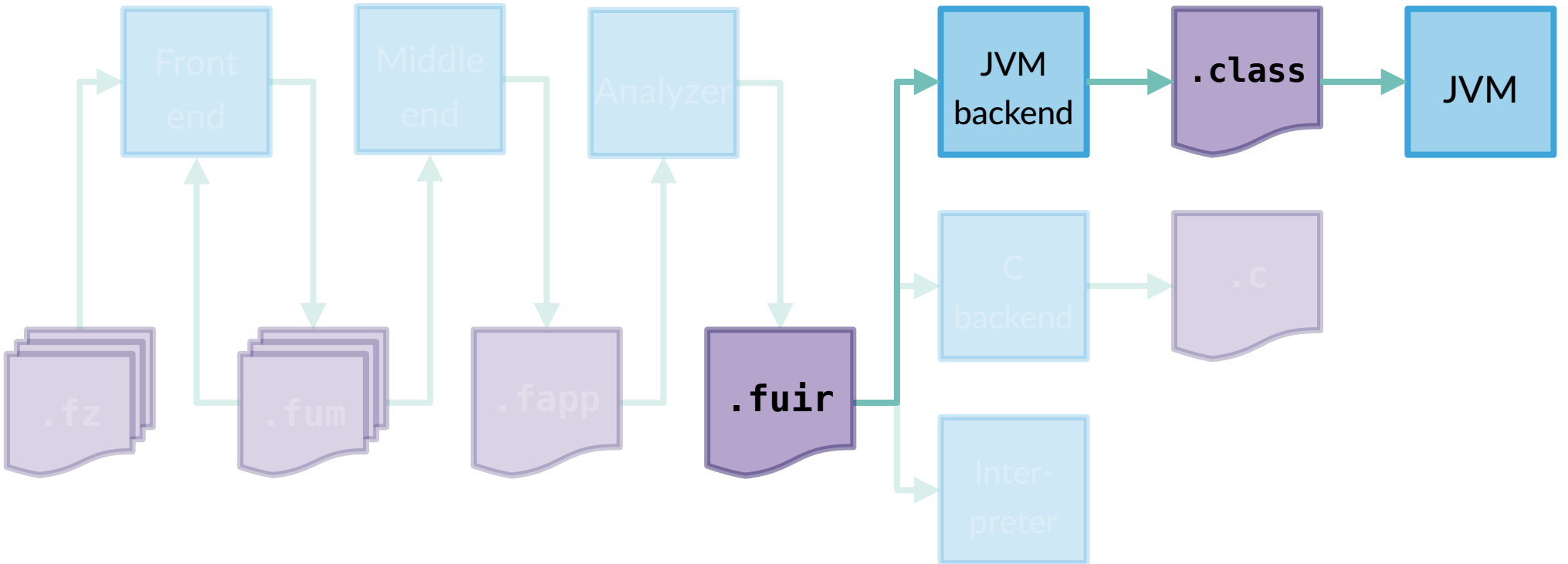
# Fuzion Toolchain






TOKIWA software

# Fuzion Toolchain






TOKIWA software

The Challenges of Running the

# Fuzion Language on OpenJDK

overview

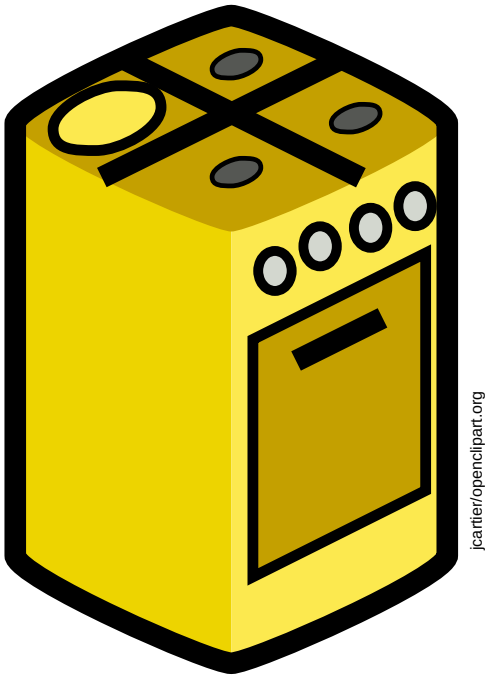
- ➡ Fuzion quick intro ✓
- ➡ Tagged union types
- ➡ Product types with value semantics
- ➡ Type parameters
- ➡ Multiple Inheritance
- ➡ Classfile verifier

TOKIWA software





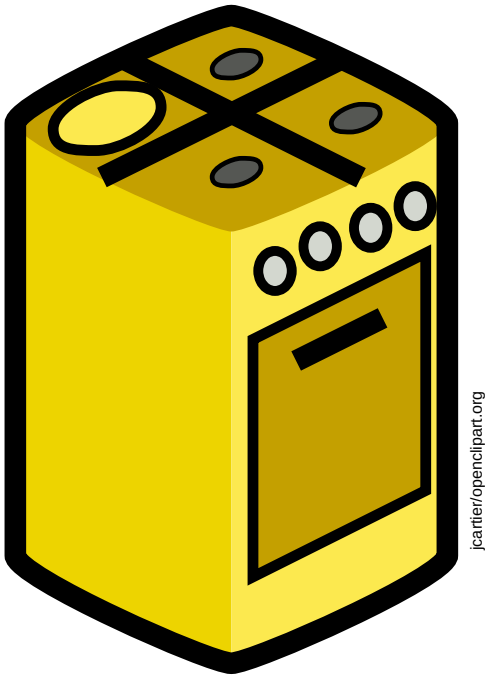
# Tagged Union Types: General



jcartier/openclipart.org

# Tagged Union Types: General

```
oven(setting off | degC | degF) is
```



jcartier/openclipart.org



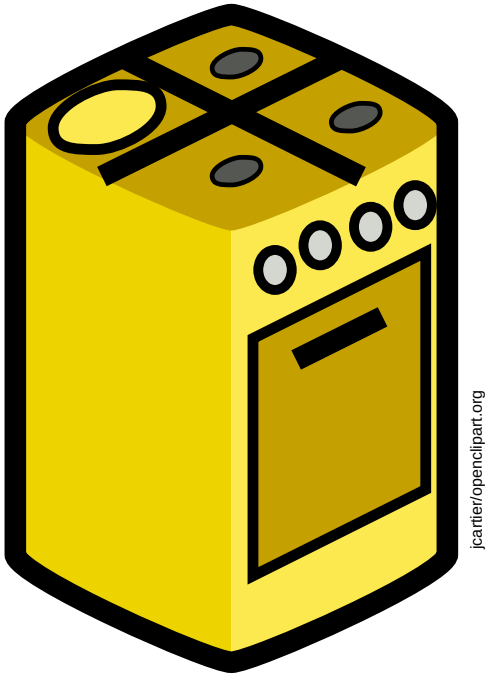
TOKIWA software



# Tagged Union Types: General

```
off.

oven(setting off | degC | degF) is
```



jcartier/openclipart.org



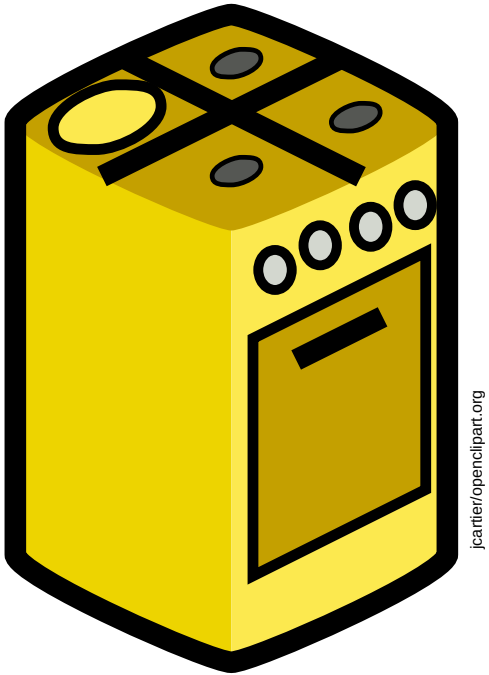
TOKIWA software



# Tagged Union Types: General

```
off.
degC(v i32).

oven(setting off | degC | degF) is
```



jcartier/openclipart.org

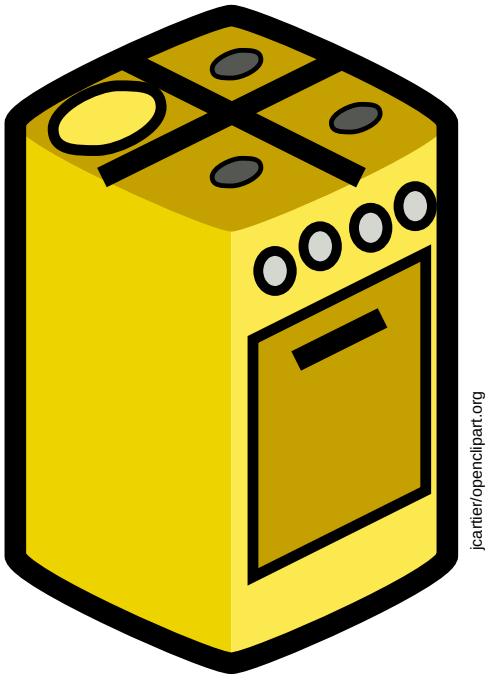


TOKIWA software



# Tagged Union Types: General

```
off.
degC(v i32).
degF(v f64).
oven(setting off | degC | degF) is
```



jcartier/openclipart.org

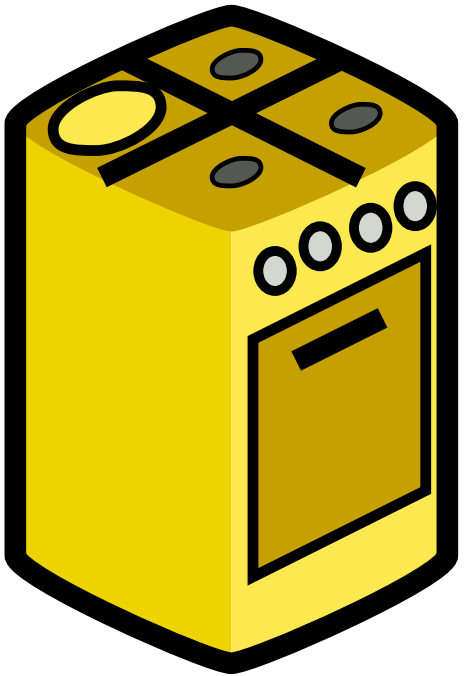


TOKIWA software



# Tagged Union Types: General

```
off.
degC(v i32).
degF(v f64).
oven(setting off | degC | degF) is
  match setting
```



jcartier/openclipart.org

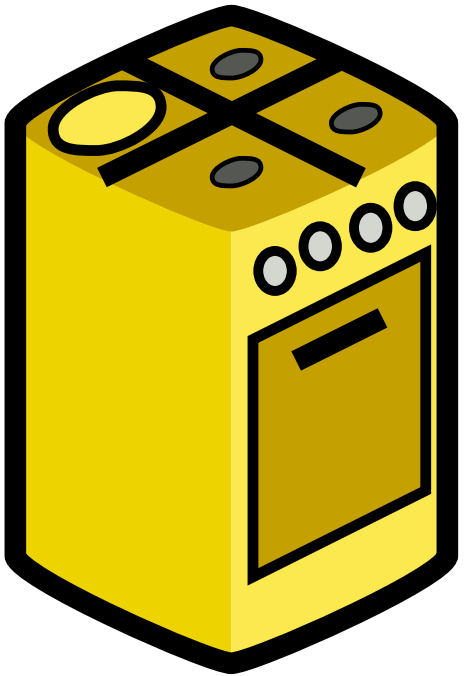


TOKIWA software



# Tagged Union Types: General

```
off.
degC(v i32).
degF(v f64).
oven(setting off | degC | degF) is
  match setting
    off      ⇒ switch_off
```



jcartier/openclipart.org

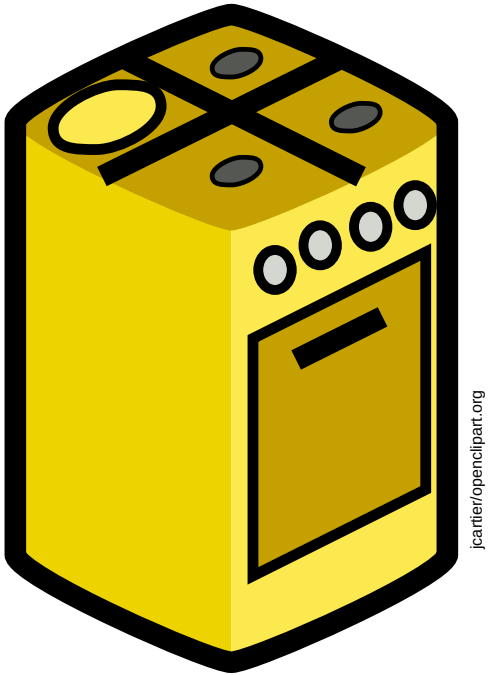


TOKIWA software



# Tagged Union Types: General

```
off.
degC(v i32).
degF(v f64).
oven(setting off | degC | degF) is
  match setting
    off     ⇒ switch_off
    tc degC ⇒ heat_to tc
```



jcartier/openclipart.org

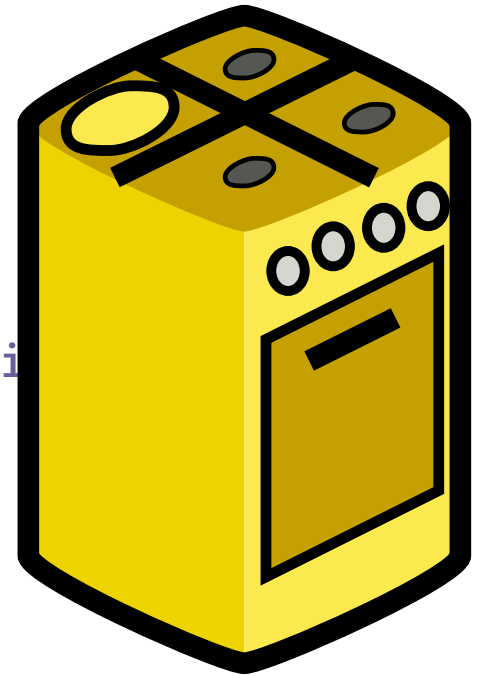


TOKIWA software



# Tagged Union Types: General

```
off.
degC(v i32).
degF(v f64).
oven(setting off | degC | degF) is
  match setting
    off     ⇒ switch_off
    tc degC ⇒ heat_to tc
    tf degF ⇒ heat_to (degC ((tf.v-32)*5/9).as_i
```



jcartier/openclipart.org

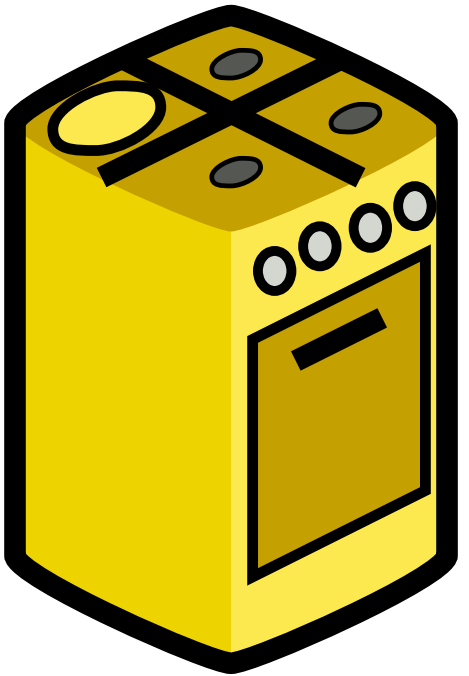


TOKIWA software



# Tagged Union Types: General

```
off.
degC(v i32).
degF(v f64).
oven(setting off | degC | degF) is
  match setting
    off     ⇒ switch_off
    tc degC ⇒ heat_to tc
    tf degF ⇒ heat_to (degC ((tf.v-32)*5/9).as_i32)
```



jcartier/openclipart.org



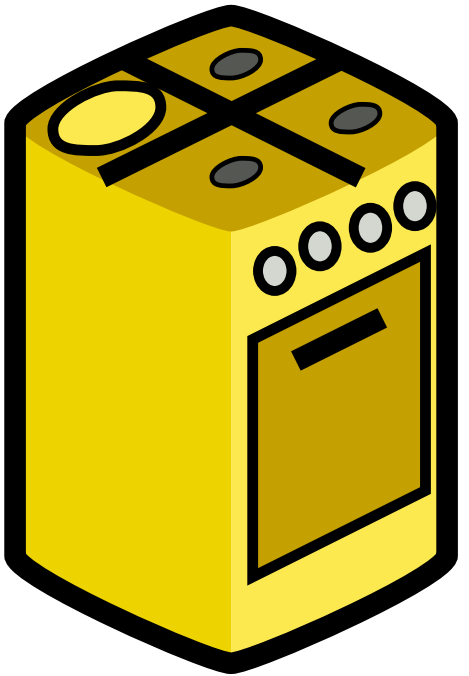
TOKIWA software



# Tagged Union Types: General

```
off.
degC(v i32).
degF(v f64).
oven(setting off | degC | degF) is
  match setting
    off     ⇒ switch_off
    tc degC ⇒ heat_to tc
    tf degF ⇒ heat_to (degC ((tf.v-32)*5/9).as_i32)
```

in Java, setting will be turned into three fields



jcartier/openclipart.org



TOKIWA software

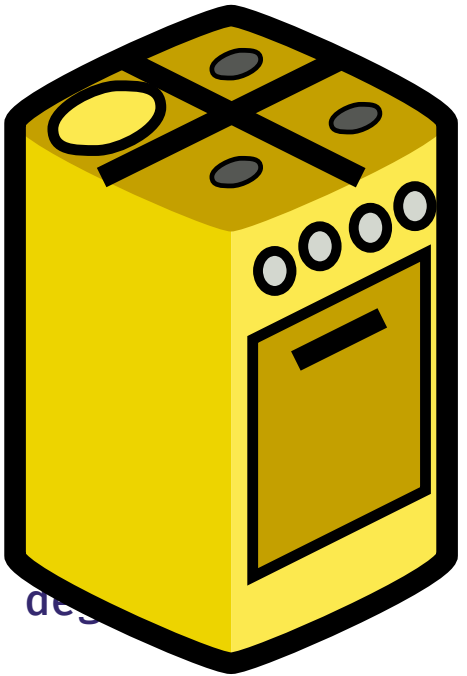


# Tagged Union Types: General

```
off.
degC(v i32).
degF(v f64).
oven(setting off | degC | degF) is
  match setting
    off     ⇒ switch_off
    tc degC ⇒ heat_to tc
    tf degF ⇒ heat_to (degC ((tf.v-32)*5/9).as_i32)
```

in Java, setting will be turned into three fields

```
int setting_tag;          // 0, 1 or 2 for off / degC / deg
```



jcartier/openclipart.org

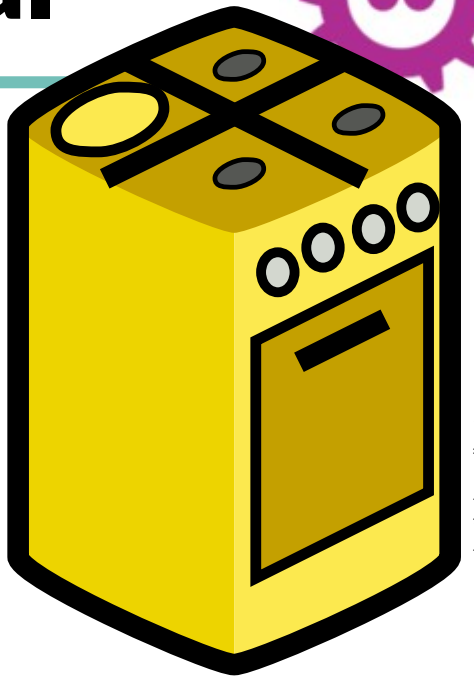


TOKIWA software



# Tagged Union Types: General

```
off.
degC(v i32).
degF(v f64).
oven(setting off | degC | degF) is
  match setting
    off     ⇒ switch_off
    tc degC ⇒ heat_to tc
    tf degF ⇒ heat_to (degC ((tf.v-32)*5/9).as_i32)
```



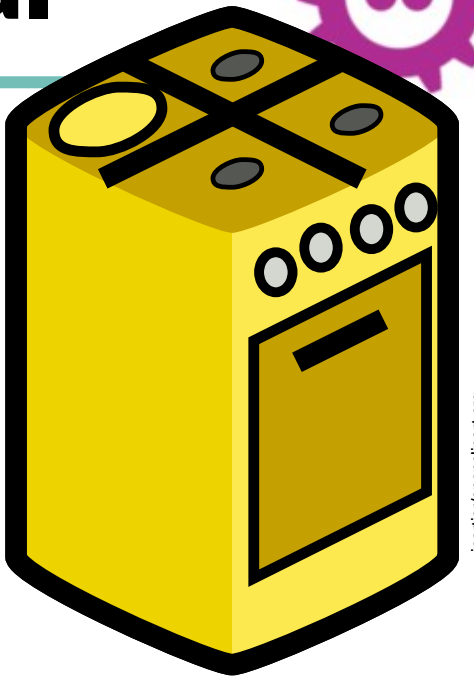
jcartier/openclipart.org

in Java, setting will be turned into three fields

```
int setting_tag;          // 0, 1 or 2 for off / degC / degF
```

# Tagged Union Types: General

```
off.
degC(v i32).
degF(v f64).
oven(setting off | degC | degF) is
  match setting
    off     ⇒ switch_off
    tc degC ⇒ heat_to tc
    tf degF ⇒ heat_to (degC ((tf.v-32)*5/9).as_i32)
```

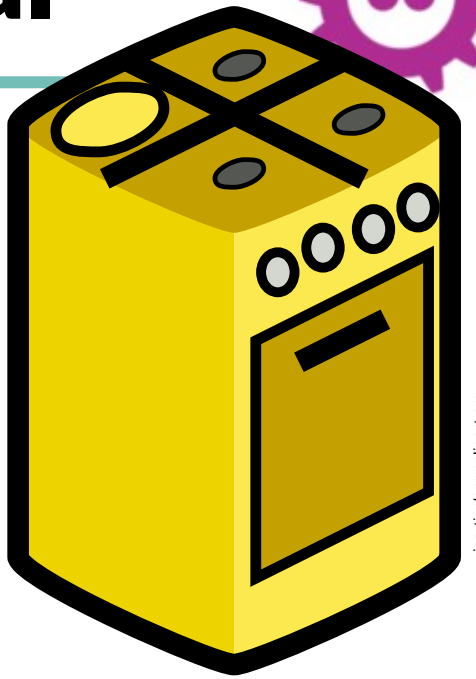



in Java, setting will be turned into three fields

```
int setting_tag;          // 0, 1 or 2 for off / degC / degF
int setting_degC_v;
```

# Tagged Union Types: General

```
off.
degC(v i32).
degF(v f64).
oven(setting off | degC | degF) is
  match setting
    off     ⇒ switch_off
    tc degC ⇒ heat_to tc
    tf degF ⇒ heat_to (degC ((tf.v-32)*5/9).as_i32)
```



jcartier/openclipart.org

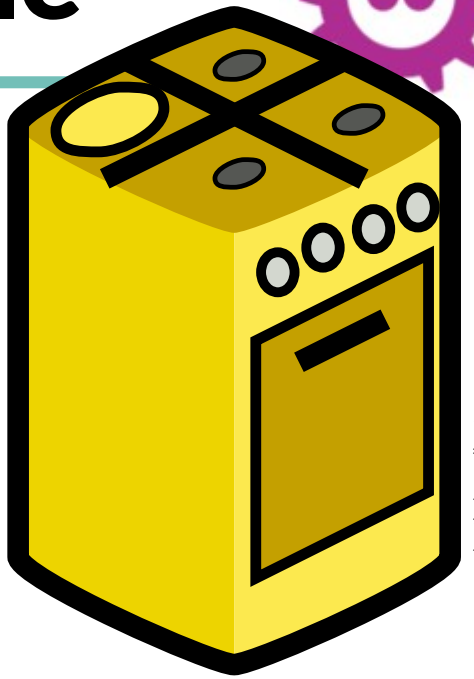
in Java, setting will be turned into three fields

```
int setting_tag;          // 0, 1 or 2 for off / degC / degF
int setting_degC_v;
double setting_degF_v;
```

TOKIWA software

# Tagged Union Types: Nullable

```
off.
Temperature ref is
  as_celsius i32 ⇒ abstract
oven(setting off | degC | degF) is
  match setting
    off     ⇒ switch_off
    tc degC ⇒ heat_to tc
    tf degF ⇒ heat_to (degC ((tf.v-32)*5/9).as_i32)
```



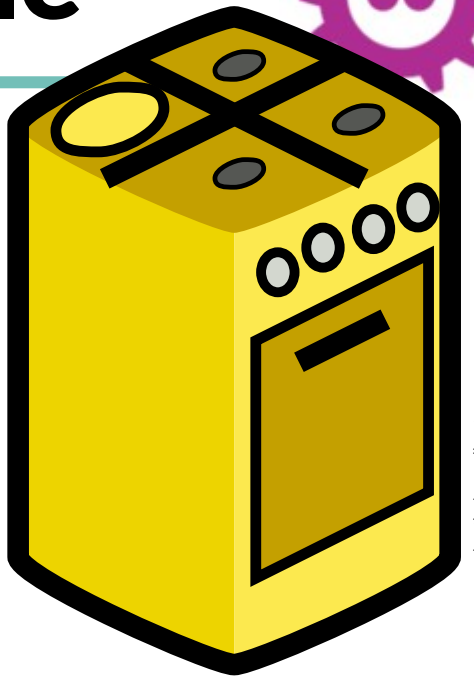
jcartier/openclipart.org



TOKIWA software

# Tagged Union Types: Nullable

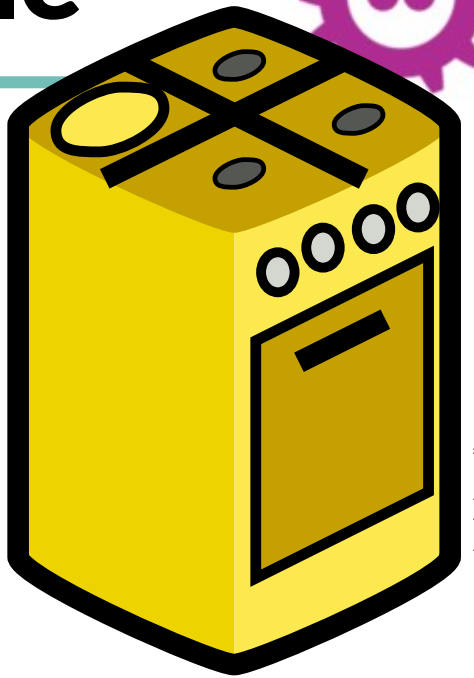
```
off.
Temperature ref is
  as_celsius i32 ⇒ abstract
oven(setting off | Temperature) is
  match setting
    off      ⇒ switch_off
    tc degC ⇒ heat_to tc
    tf degF ⇒ heat_to (degC ((tf.v-32)*5/9).as_i32)
```



jcartier/openclipart.org

# Tagged Union Types: Nullable

```
off.
Temperature ref is
  as_celsius i32 ⇒ abstract
oven(setting off | Temperature) is
  match setting
    off           ⇒ switch_off
    t Temperature ⇒ heat_to t.as_celsius
```



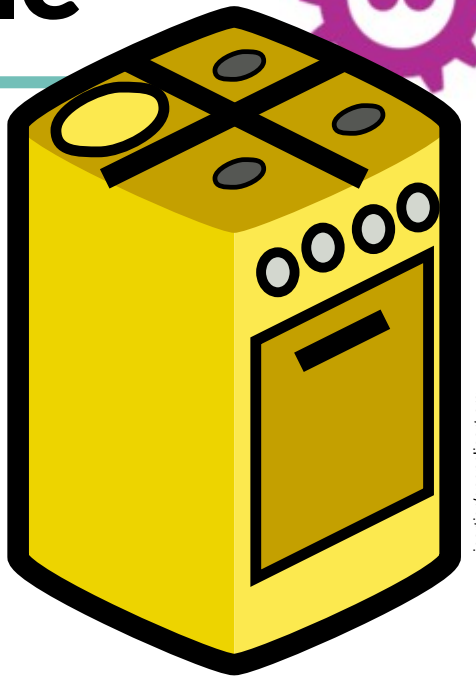
jcartier/openclipart.org



TOKIWA software

# Tagged Union Types: Nullable

```
off.
Temperature ref is
  as_celsius i32 ⇒ abstract
oven(setting off | Temperature) is
  match setting
    off           ⇒ switch_off
    t Temperature ⇒ heat_to t.as_celsius
```



jcartier/openclipart.org

in Java, this could be two fields...

```
int setting_tag;              // 0 for off, 1 for Temperature
Temperature setting_temperature;
```

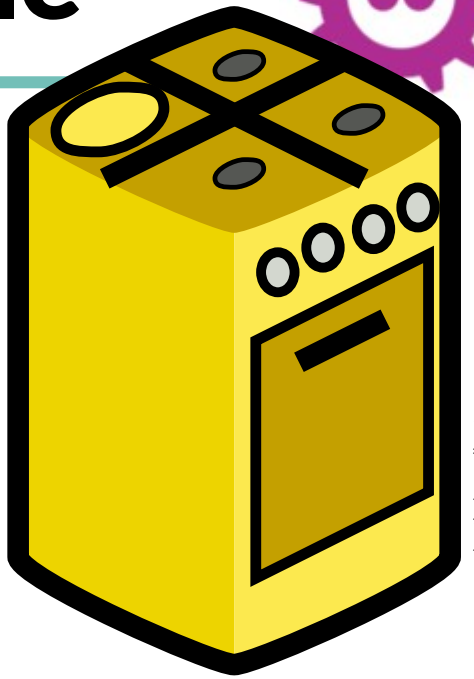


TOKIWA software



# Tagged Union Types: Nullable

```
off.
Temperature ref is
  as_celsius i32 ⇒ abstract
oven(setting off | Temperature) is
  match setting
    off           ⇒ switch_off
    t Temperature ⇒ heat_to t.as_celsius
```



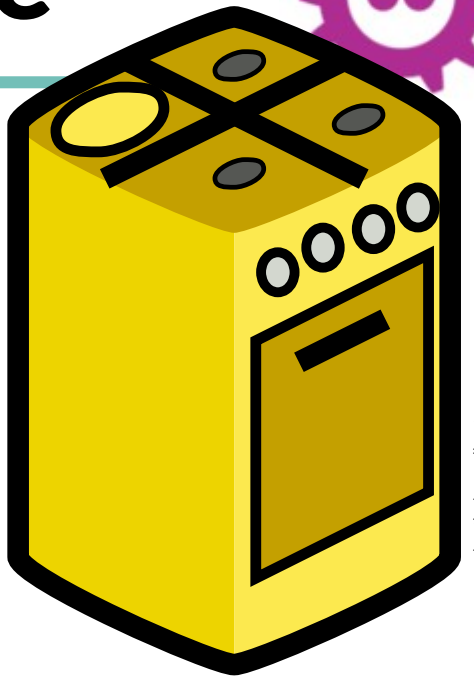
jcartier/openclipart.org

in Java, ...or a reference that might be null

```
Temperature setting_ref;    // null for off, Temperature otherwise
```

# Tagged Union Types: Ref-like

```
off.
clean.
Temperature ref is
  …
Error ref is
  …
oven(setting off | clean | Temperature | Error) is
  …
```



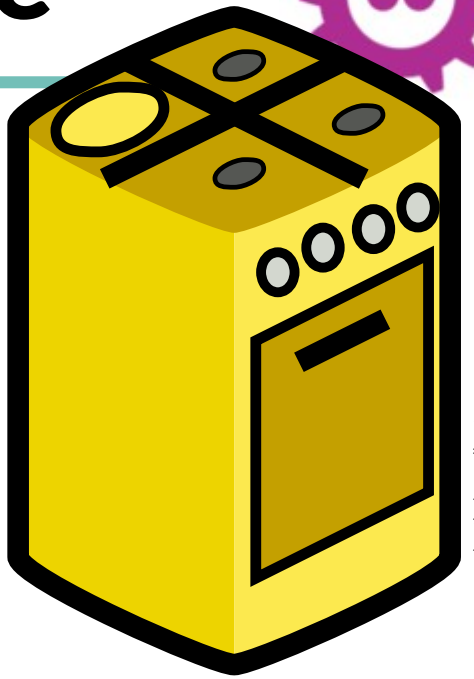
jcartier/openclipart.org



TOKIWA software

# Tagged Union Types: Ref-like

```
off.
clean.
Temperature ref is
  …
Error ref is
  …
oven(setting off | clean | Temperature | Error) is
  …
```



jcartier/openclipart.org

in Java, this could be three fields

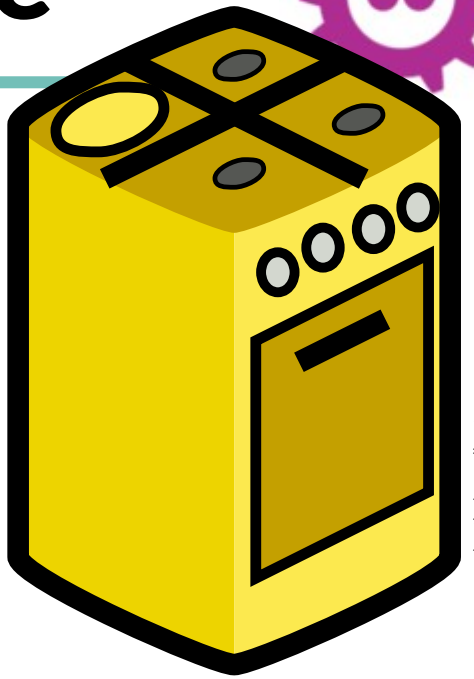
```
int setting_tag;              // 0/1/2/3 for off/clean/Temperature/Error
Temperature setting_temperature;
Error setting_error;
```



TOKIWA software

# Tagged Union Types: Ref-like

```
off.
clean.
Temperature ref is
  …
Error ref is
  …
oven(setting off | clean | Temperature | Error) is
  …
```



jcartier/openclipart.org

in Java, this could be three fields

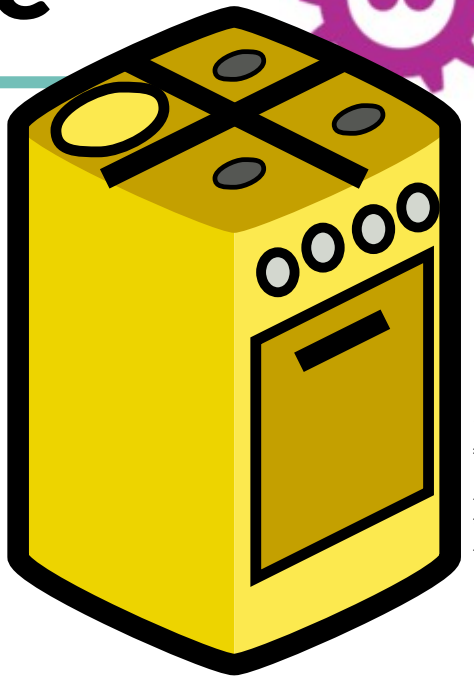
```
int setting_tag;              // 0/1/2/3 for off/clean/Temperature/Error
Temperature setting_temperature;
Error setting_error;
```



TOKIWA software

# Tagged Union Types: Ref-like

```
off.
clean.
Temperature ref is
  …
Error ref is
  …
oven(setting off | clean | Temperature | Error) is
  …
```



jcartier/openclipart.org

in Java, this could be two fields

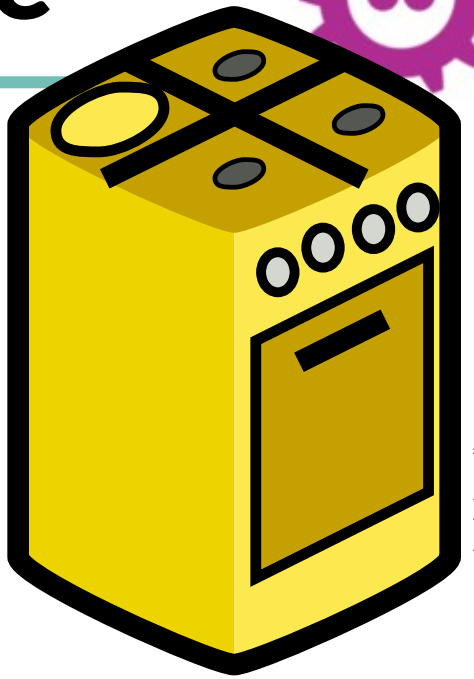
```
int setting_tag;          // 0/1/2/3 for off/clean/Temperature/Error
Object setting_ref;       // Temperature or Error
```



TOKIWA software

# Tagged Union Types: Ref-like

```
off.
clean.
Temperature ref is
  …
Error ref is
  …
oven(setting off | clean | Temperature | Error) is
  …
```



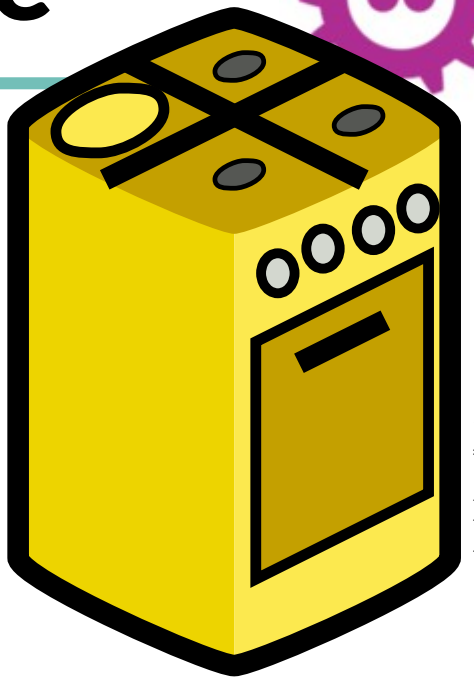
jcartier/openclipart.org

in Java, this could be two fields

```
int setting_tag;          // 0/1/2/3 for off/clean/Temperature/Error
Object setting_ref;       // Temperature or Error
```

# Tagged Union Types: Ref-like

```
off.
clean.
Temperature ref is
  …
Error ref is
  …
oven(setting off | clean | Temperature | Error) is
  …
```



jcartier/openclipart.org

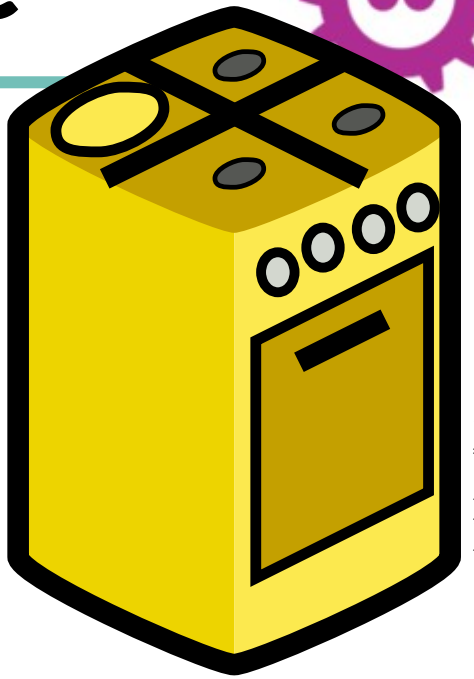
in Java, this could be one field

```
Object setting_ref;          // off_G, clean_G, Temperature or Error
static Object off_G   = new Integer(1);
static Object clean_G = new Integer(2);
```

TOKIWA software

# Tagged Union Types: Int-like

```
on.
off.
clean.
err.
oven(setting off | clean | on | err) is
  …
```



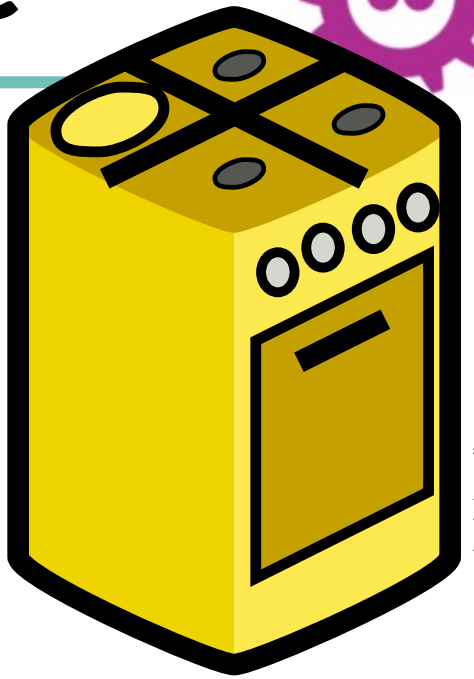
jcartier/openclipart.org



TOKIWA software

# Tagged Union Types: Int-like

```
on.
off.
clean.
err.
oven(setting off | clean | on | err) is
  …
```



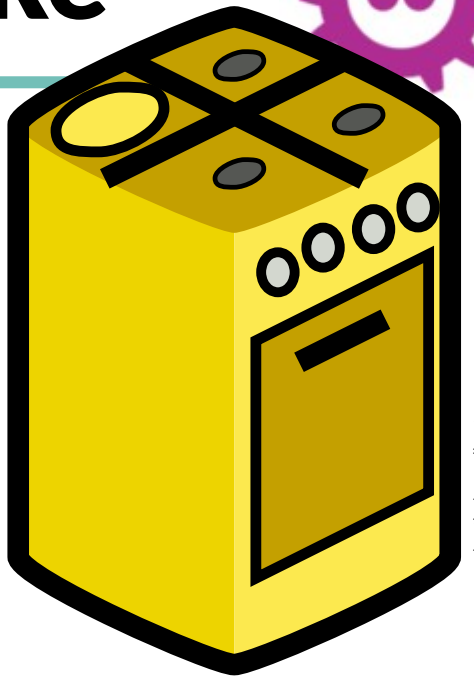
jcartier/openclipart.org

in Java, this could be one field

```
int setting_tag;          // 0/1/2/3 for on/off/clean/err
```

# Tagged Union Types: bool-like

```
on.
off.
oven(setting on | off) is
  …
```



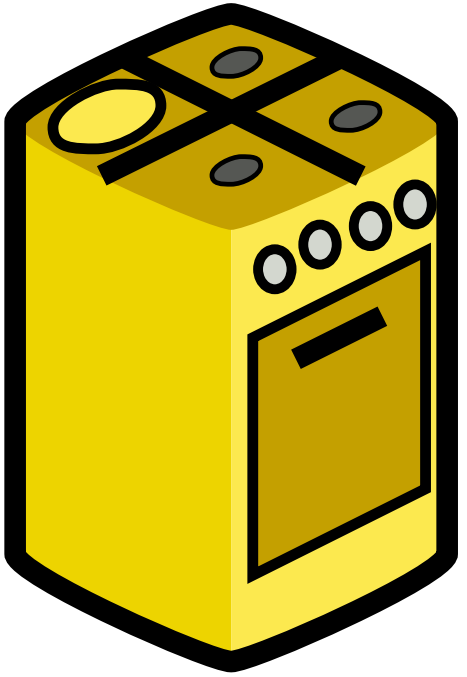
jcartier/openclipart.org

in Java, this could be one field

```
int setting;                    // 0/1 for on/off
```

# Tagged Union Types: bool-like

```
on.
off.
oven(setting on | off) is
  …
```



jcartier/openclipart.org

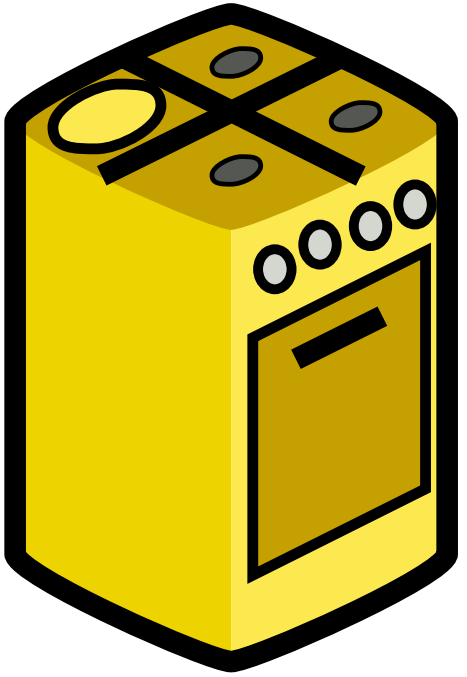
in Java, this could be one field

```
int setting;                    // 0/1 for on/off
```



# Tagged Union Types: bool-like

```
on.
off.
oven(setting on | off) is
  …
```



jcartier/openclipart.org

in Java, this could be one field

```
boolean setting;          // true/false for on/off
```

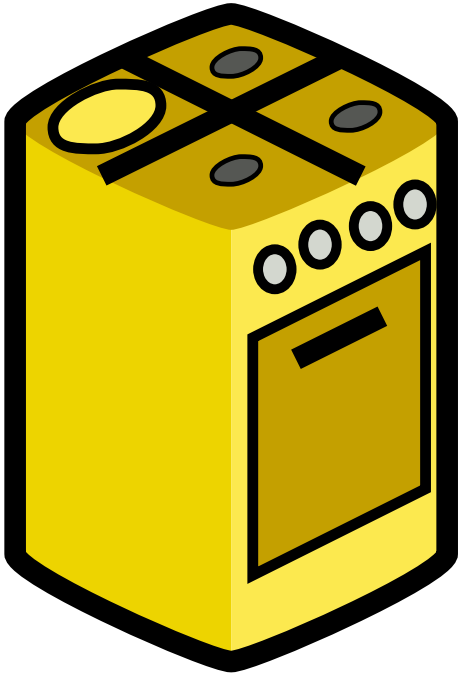


TOKIWA software



# Tagged Union Types: bool-like

```
on.
off.
oven(setting on | off) is
  …
```



jcartier/openclipart.org

in Java, this could be one field

```
boolean setting;          // true/false for on/off
```

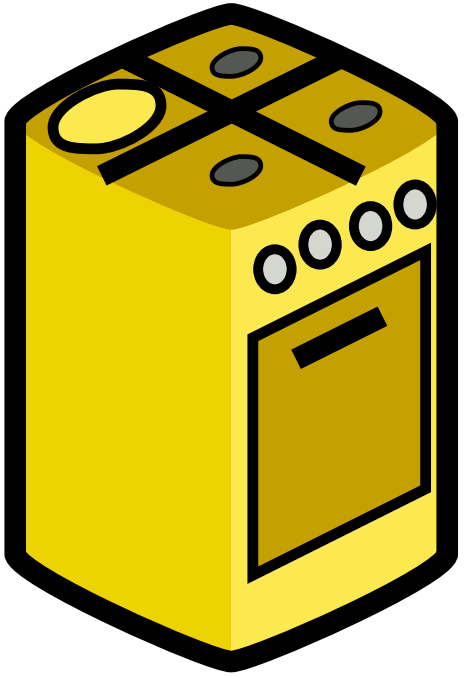


TOKIWA software



# Tagged Union Types: unit-like

```
on.                          # never used!
off.
oven(setting on | off) is
  …
```



jcartier/openclipart.org

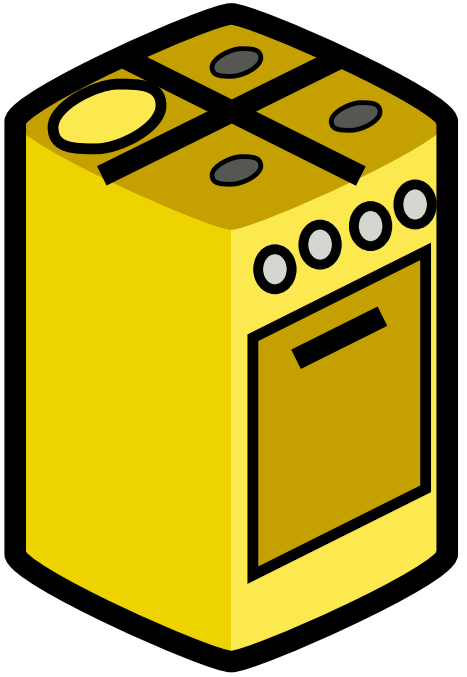
in Java, this could be one field

```
boolean setting;             // true/false for on/off
```



# Tagged Union Types: unit-like

```
void.                          # never used!
off.
oven(setting void | off) is
  …
```



jcartier/openclipart.org

in Java, this could be zero fields

```
// nothing, oven is always off
```

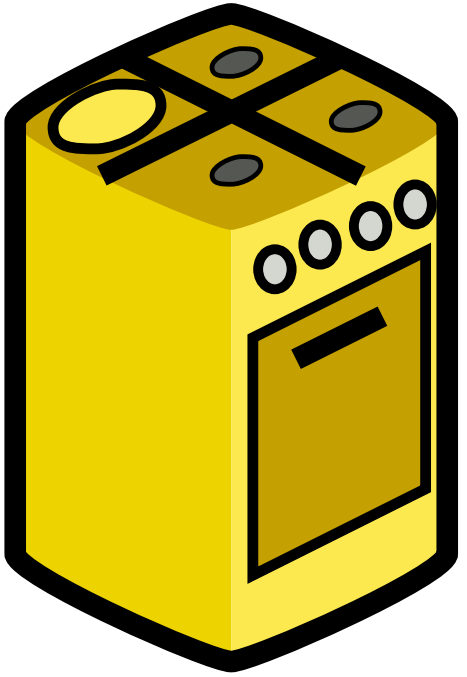


TOKIWA software



# Tagged Union Types: void-like

```
void.                          # never used!
off.                           # never used!
oven(setting void | off) is
  …
```



jcartier/openclipart.org

in Java, this could be zero fields

```
// nothing, oven is always off
```



# Tagged Union Types: void-like

```
void.                          # never used!
void.                          # never used!
oven(setting void | void) is
  …
```




in Java, this could be zero fields

```
// nothing, oven is always off

// no code for oven, it cannot be called!
```



The Challenges of Running the

# Fuzion Language on OpenJDK

overview

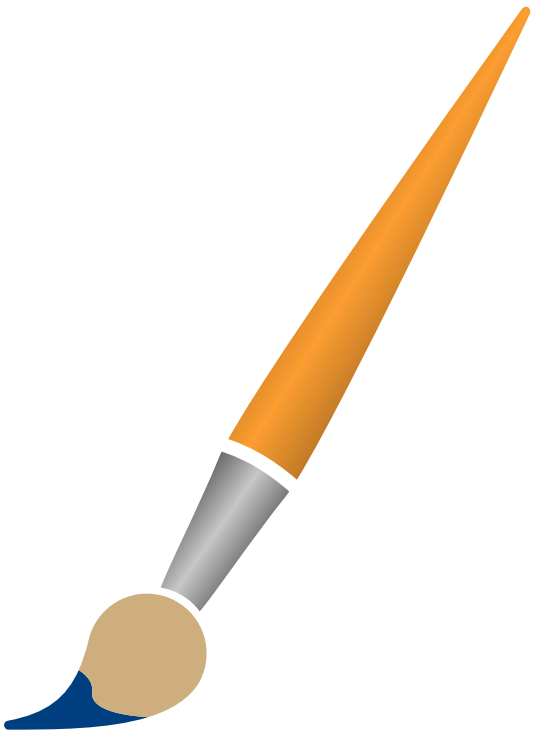
- ➡ Fuzion quick intro ✓
- ➡ Tagged union types ✓
- ➡ Product types with value semantics
- ➡ Type parameters
- ➡ Multiple Inheritance
- ➡ Classfile verifier

TOKIWA software



# Product Type defined as feature

```
point (x, y i32).
```



Astro/openclipart.org



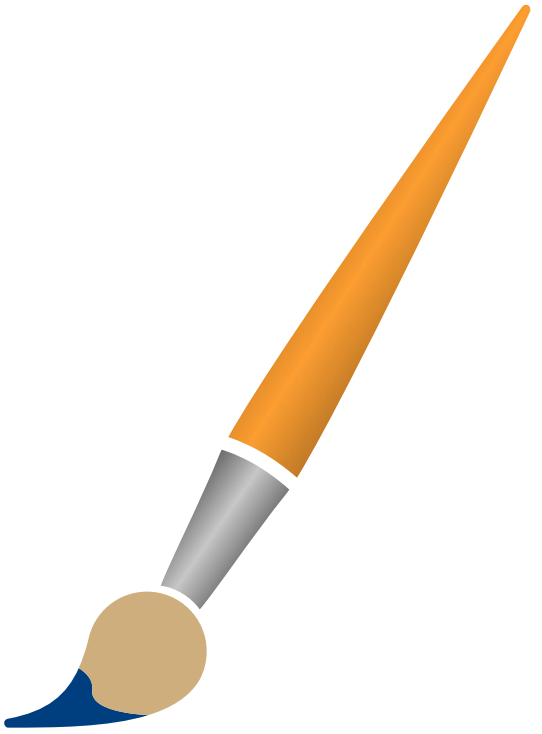
TOKIWA software



# Product Types: As several values

```
point (x, y i32).

draw (p point) ! graphics ⇒
  graphics.env.draw_point p.x p.y
```



Astro/openclipart.org

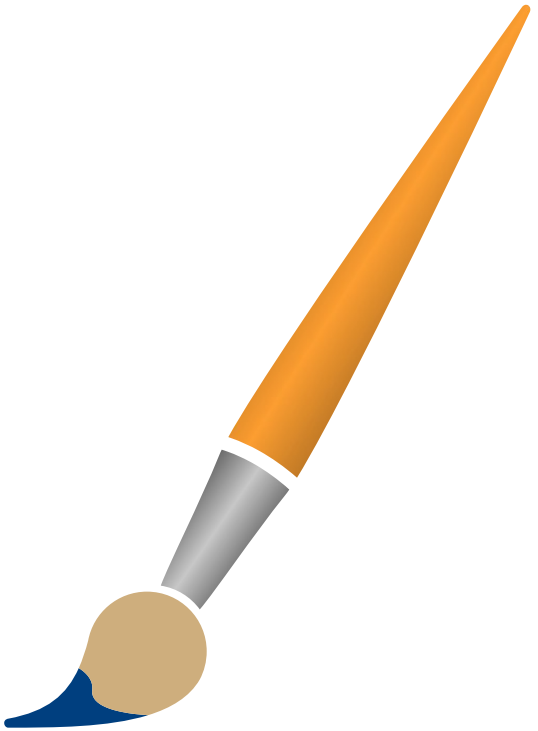


# Product Types: As several values

```
point (x, y i32).

draw (p point) ! graphics ⇒
  graphics.env.draw_point p.x p.y

p1 := point 3 4
draw p1
```



Astro/openclipart.org

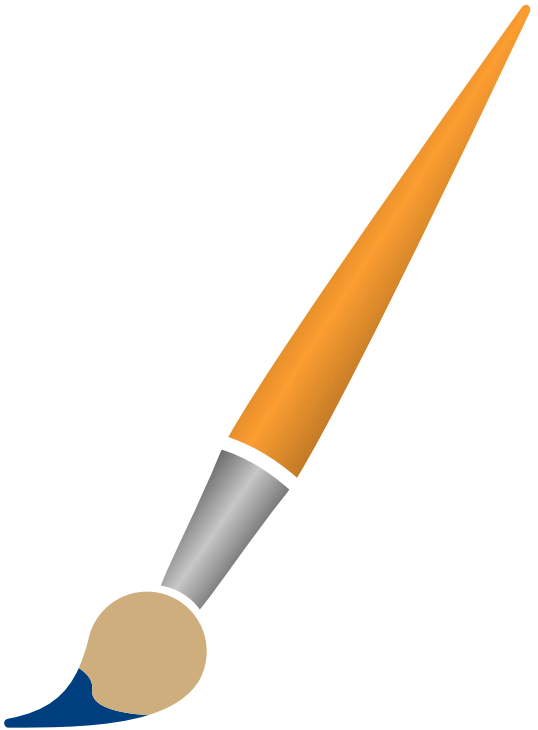


# Product Types: As several values

```
point (x, y i32).

draw (p point) ! graphics ⇒
  graphics.env.draw_point p.x p.y

p1 := point 3 4
draw p1
```



Astro/openclipart.org



TOKIWA software



# Product Types: As several values

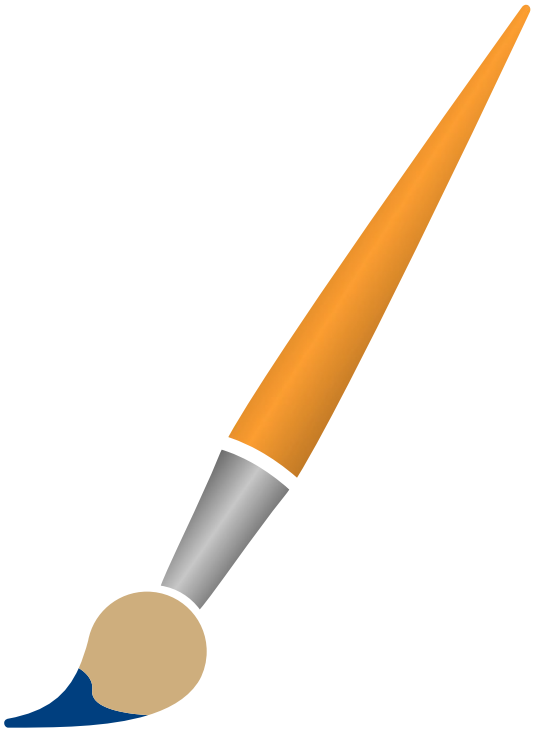
```
point (x, y i32).

draw (p point) ! graphics ⇒
  graphics.env.draw_point p.x p.y

p1 := point 3 4
draw p1
```

in Java, we need two variables or fields

```
void draw(int p_x, int p_y) { … }
```



Astro/openclipart.org



# Product Types: As several values

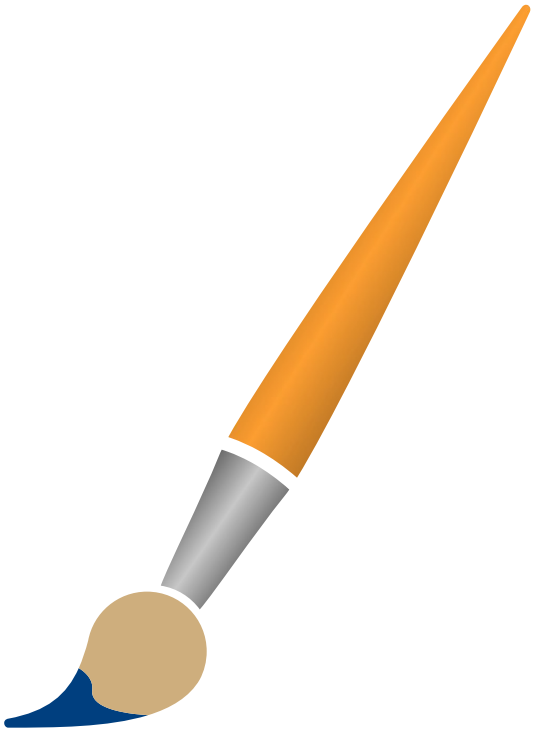
```
point (x, y i32).

draw (p point) ! graphics ⇒
  graphics.env.draw_point p.x p.y

p1 := point 3 4
draw p1
```

in Java, we need two variables or fields

```
void draw(int p_x, int p_y) { … }
```



Astro/openclipart.org

# Product Types: As several values

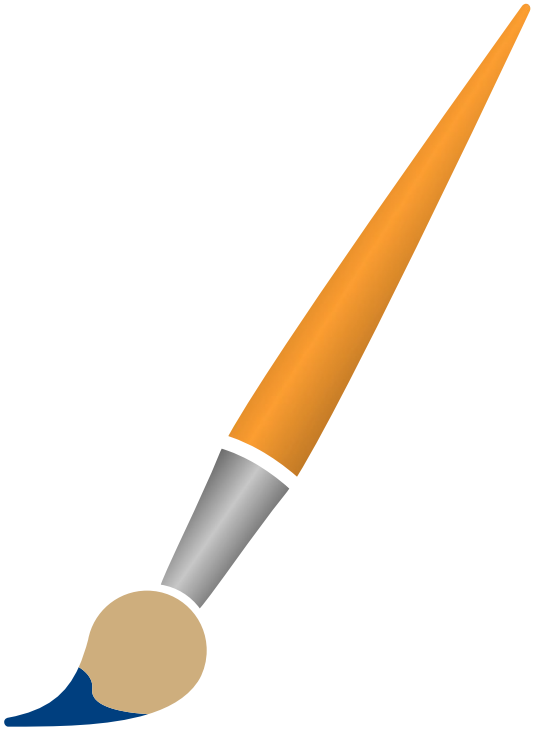
```
point (x, y i32).

draw (p point) ! graphics ⇒
  graphics.env.draw_point p.x p.y

p1 := point 3 4
draw p1
```

in Java, we need two variables or fields

```
void draw(int p_x, int p_y) { … }
int p1_x = 3;
int p1_y = 4;
```



Astro/openclipart.org

# Product Types: As several values

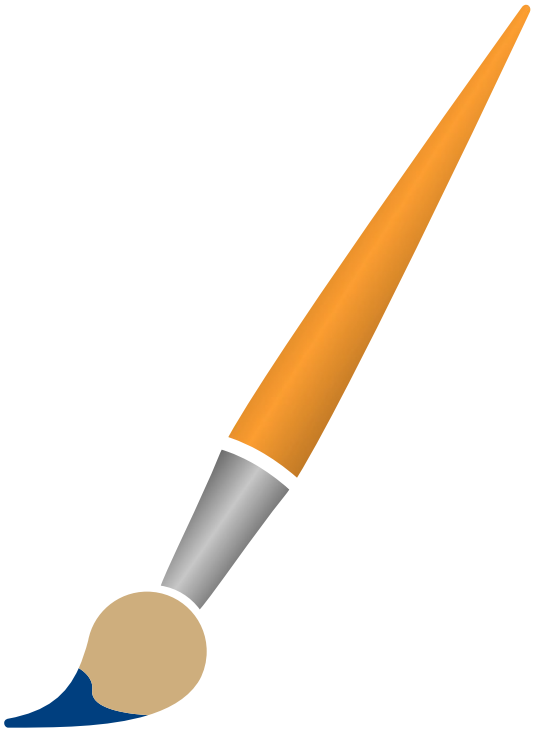
```
point (x, y i32).

draw (p point) ! graphics ⇒
  graphics.env.draw_point p.x p.y

p1 := point 3 4
draw p1
```

in Java, we need two variables or fields

```
void draw(int p_x, int p_y) { … }
int p1_x = 3;
int p1_y = 4;
```



Astro/openclipart.org

# Product Types: As several values

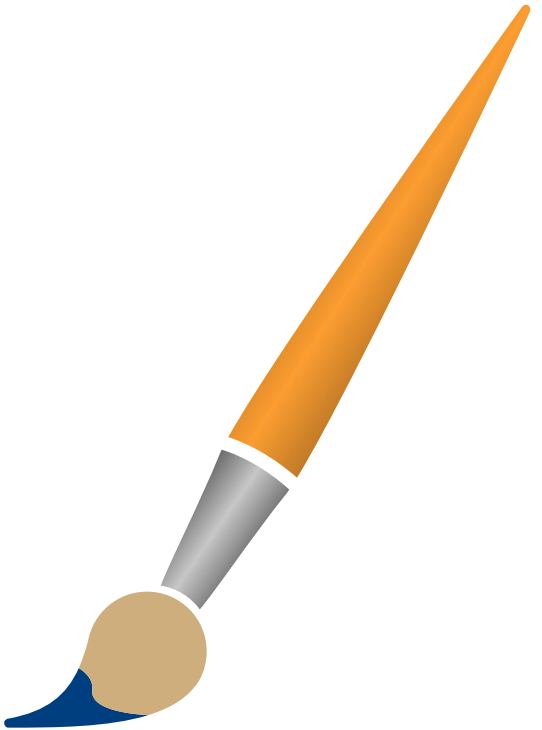
```
point (x, y i32).

draw (p point) ! graphics ⇒
  graphics.env.draw_point p.x p.y

p1 := point 3 4
draw p1
```

in Java, we need two variables or fields

```
void draw(int p_x, int p_y) { … }
int p1_x = 3;
int p1_y = 4;
draw(p1_x, p1_y);
```

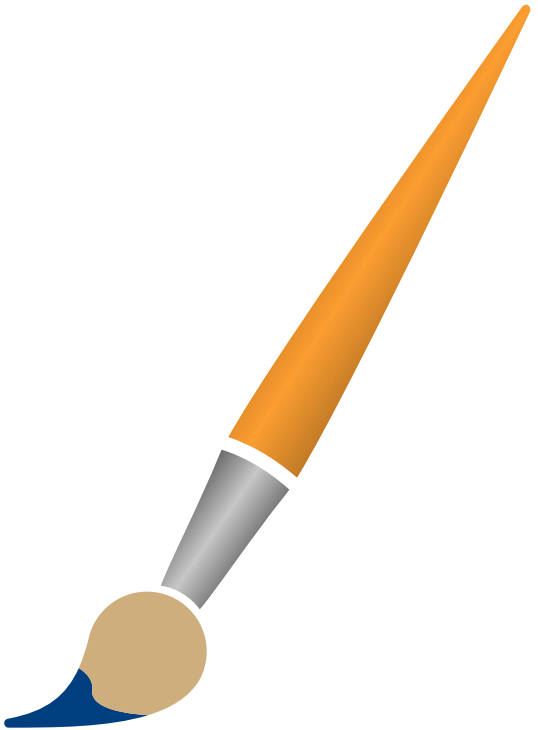


Astro/openclipart.org

# Returning Product Type Values

```
point (x, y i32) is
  shear (k i32) ⇒ point x+k*y y

p1 := point 3 4
p2 := p1.shear 2
```



Astro/openclipart.org

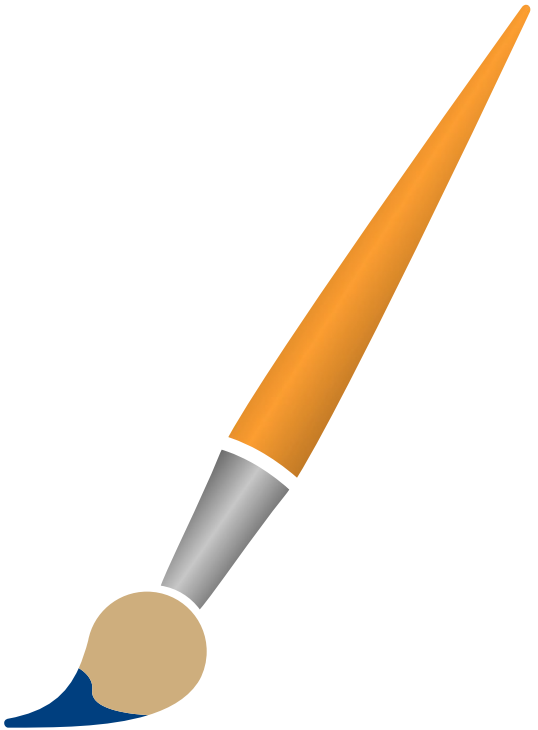


# Returning Product Type Values

```
point (x, y i32) is
  shear (k i32) ⇒ point x+k*y y

p1 := point 3 4
p2 := p1.shear 2
```

in Java, what can we do?



Astro/openclipart.org

# Returning Product Type Values

in Java, what can we do?



TOKIWA software



# Returning Product Type Values

in Java, what can we do?

➡ inline the call, so return is assignment to local variable



TOKIWA software



# Returning Product Type Values

in Java, what can we do?

➡ inline the call, so return is assignment to local variable

➡ alloc and return new container for two `int`s



TOKIWA software



# Returning Product Type Values

in Java, what can we do?

➡ inline the call, so return is assignment to local variable

➡ alloc and return new container for two `int`s

➡ using caller-allocated container and call-by-ref



# Returning Product Type Values

in Java, what can we do?

➡ inline the call, so return is assignment to local variable

➡ alloc and return new container for two `int`s

➡ using caller-allocated container and call-by-ref

➡ add static fields for results



# Returning Product Type Values

in Java, what can we do?

➡ inline the call, so return is assignment to local variable

➡ alloc and return new container for two `int`s

➡ using caller-allocated container and call-by-ref

➡ add static fields for results

➡ add result fields as ThreadLocal values



# Returning Product Type Values

in Java, what can we do?

➡ inline the call, so return is assignment to local variable

➡ alloc and return new container for two `int`s

➡ using caller-allocated container and call-by-ref

➡ add static fields for results

➡ add result fields as ThreadLocal values

➡ add result fields to current thread instance



TOKIWA software



# Returning Product Type Values

| in Java, what can we do? | JMH |
|---|---|
| ➡ inline the call, so return is assignment to local variable | 100% |
| ➡ alloc and return new container for two `int`s | 107% |
| ➡ using caller-allocated container and call-by-ref | 106% |
| ➡ add static fields for results | 80% |
| ➡ add result fields as ThreadLocal values | 6% |
| ➡ add result fields to current thread instance | n/a |



# Returning Product Type Values

| in Java, what can we do? | JMH | ad-hoc |
|---|---|---|
| ➡ inline the call, so return is assignment to local variable | 100% | 100% |
| ➡ alloc and return new container for two `int`s | 107% | 90% |
| ➡ using caller-allocated container and call-by-ref | 106% | 99% |
| ➡ add static fields for results | 80% | 95% |
| ➡ add result fields as ThreadLocal values | 6% | 5% |
| ➡ add result fields to current thread instance | n/a | 95% |



# Returning Product Type Values

| in Java, what can we do? | JMH | ad-hoc |
|---|---|---|
| ➡ inline the call, so return is assignment to local variable | 100% | 100% |
| ➡ alloc and return new container for two `int`s | 107% | 90% |
| ➡ using caller-allocated container and call-by-ref | 106% | 99% |
| ➡ add static fields for results | 80% | 95% |
| ➡ add result fields as ThreadLocal values | 6% | 5% |
| ➡ add result fields to current thread instance | n/a | 95% |



# Returning Product Type Values

| in Java, what can we do? | JMH | ad-hoc |
|---|---|---|
| ➡ inline the call, so return is assignment to local variable | 100% | 100% |
| ➡ alloc and return new container for two `int`s | 107% | 90% |
| ➡ using caller-allocated container and call-by-ref | 106% | 99% |
| ➡ add static fields for results | 80% | 95% |
| ➡ add result fields as ThreadLocal values | 6% | 5% |
| ➡ add result fields to current thread instance | n/a | 95% |



# Returning Product Type Values

| in Java, what can we do? | JMH | ad-hoc |
|---|---|---|
| ➡ inline the call, so return is assignment to local variable | 100% | 100% |
| ➡ alloc and return new container for two `int`s | 107% | 90% |
| ➡ using caller-allocated container and call-by-ref | 106% | 99% |
| ➡ add static fields for results | 80% | 95% |
| ➡ add result fields as ThreadLocal values | 6% | 5% |
| ➡ add result fields to current thread instance | n/a | 95% |



# Returning Product Type Values

## Project Valhalla

- Q-types is just what we need!
- best with VM guarantees for no heap allocation.



The Challenges of Running the

# Fuzion Language on OpenJDK

overview

- ➡ Fuzion quick intro ✓
- ➡ Tagged union types ✓
- ➡ Product types with value semantics ✓
- ➡ Type parameters
- ➡ Multiple Inheritance
- ➡ Classfile verifier

TOKIWA software



# Type Parameters

## Example

```
mean (a, b, c T : numeric) ⇒
  (a + b + c) / T.from_u32 3
```



TOKIWA software



# Type Parameters

Example

```
mean (a, b, c T : numeric) ⇒
  (a + b + c) / T.from_u32 3
```

can be called with `T` being `i32`, `f64`, etc.

```
say (mean 3 4 5)

say (mean 3.14 2.71 1.41)
```



# Type Parameters

Example

```
mean (a, b, c T : numeric) ⇒
  (a + b + c) / T.from_u32 3
```

can be called with `T` being `i32`, `f64`, etc.

```
say (mean 3 4 5)

say (mean 3.14 2.71 1.41)
```

Java uses type erasure for generics



# Type Parameters

Example

```
mean (a, b, c T : numeric) ⇒
  (a + b + c) / T.from_u32 3
```

can be called with `T` being `i32`, `f64`, etc.

```
say (mean 3 4 5)

say (mean 3.14 2.71 1.41)
```

Java uses type erasure for generics

Fuzion uses monomorphization!



# Type Parameters

Example

```
mean (a, b, c T : numeric) ⇒
  (a + b + c) / T.from_u32 3
```

JVM backend will create several Java versions for `i32`, `f64`, etc.:

```
mean_i32(int    a, int    b, int    c) { return (a+b+c)/3  ; }
mean_f32(double a, double b, double c) { return (a+b+c)/3.0; }
...
```



The Challenges of Running the

# Fuzion Language on OpenJDK

overview

- ➡ Fuzion quick intro ✓
- ➡ Tagged union types ✓
- ➡ Product types with value semantics ✓
- ➡ Type parameters ✓
- ➡ Multiple Inheritance
- ➡ Classfile verifier

TOKIWA software



# Multiple Inheritance

Options for dynamic binding

➡ table-/lookupswitch and invokestatic

➡ invokedynamic

➡ invokeinterface



# Multiple Inheritance

Options for dynamic binding

➡ table-/lookupswitch and invokestatic

➡ invokedynamic

➡ invokeinterface

Fuzion JVM backend

➡ invokestatic in case unique target type

➡ invokeinterface otherwise



The Challenges of Running the

# Fuzion Language on OpenJDK

overview

- ➡ Fuzion quick intro ✓
- ➡ Tagged union types ✓
- ➡ Product types with value semantics ✓
- ➡ Type parameters ✓
- ➡ Multiple Inheritance ✓
- ➡ Classfile verifier

TOKIWA software



# Classfile Verifier

what should I say?

➡ helped a lot

➡ made JVM backend much easier than C backend



The Challenges of Running the

# Fuzion Language on OpenJDK

overview

- ➡ Fuzion quick intro ✓
- ➡ Tagged union types ✓
- ➡ Product types with value semantics ✓
- ➡ Type parameters ✓
- ➡ Multiple Inheritance ✓
- ➡ Classfile verifier ✓

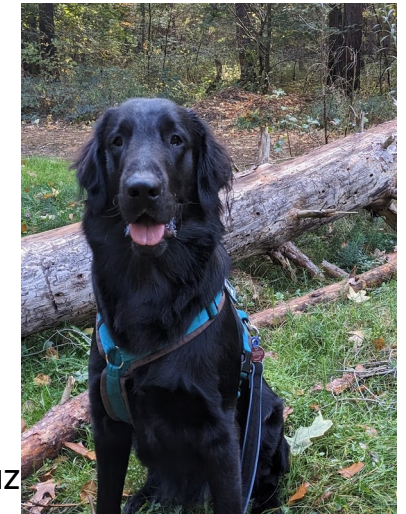
TOKIWA software



# 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
- ➡ Felix & Shadow

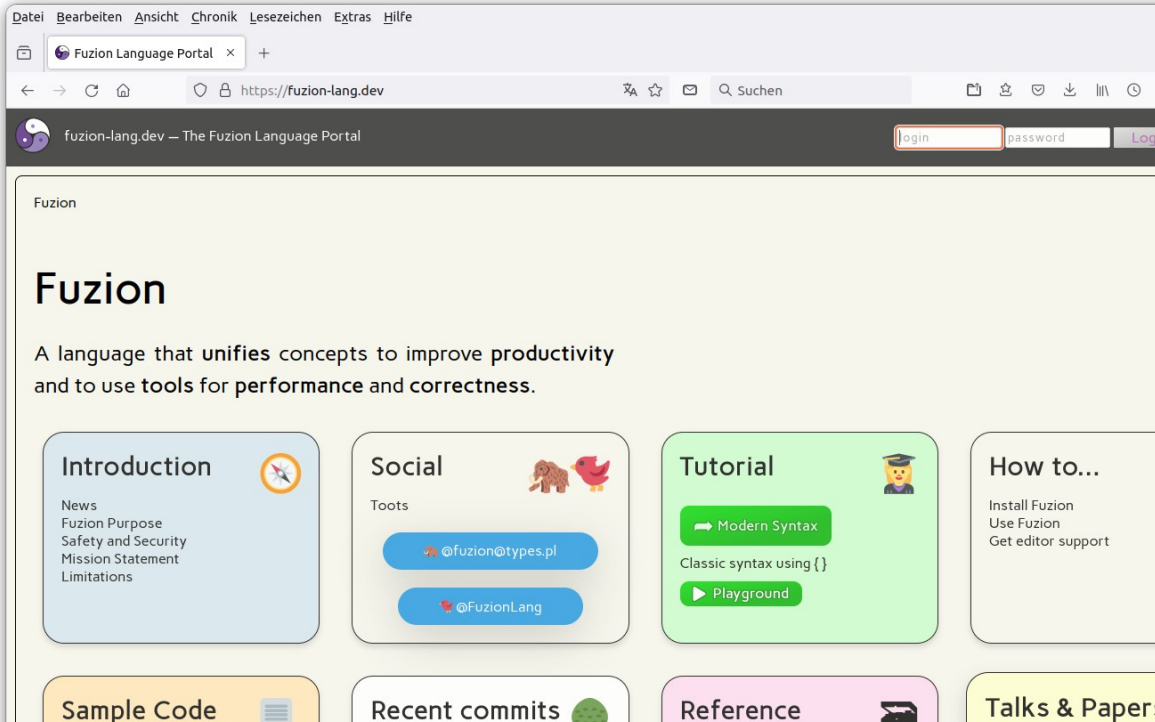




# Thank you. Any questions?

Please follow and stay informed

- ➡ https://github.com/tokiwa-software/fuzion
- ➡ https://fuzion-lang.dev
- ➡ @FuzionLang
- ➡ @Fuzion@types.pl






TOKIWA software