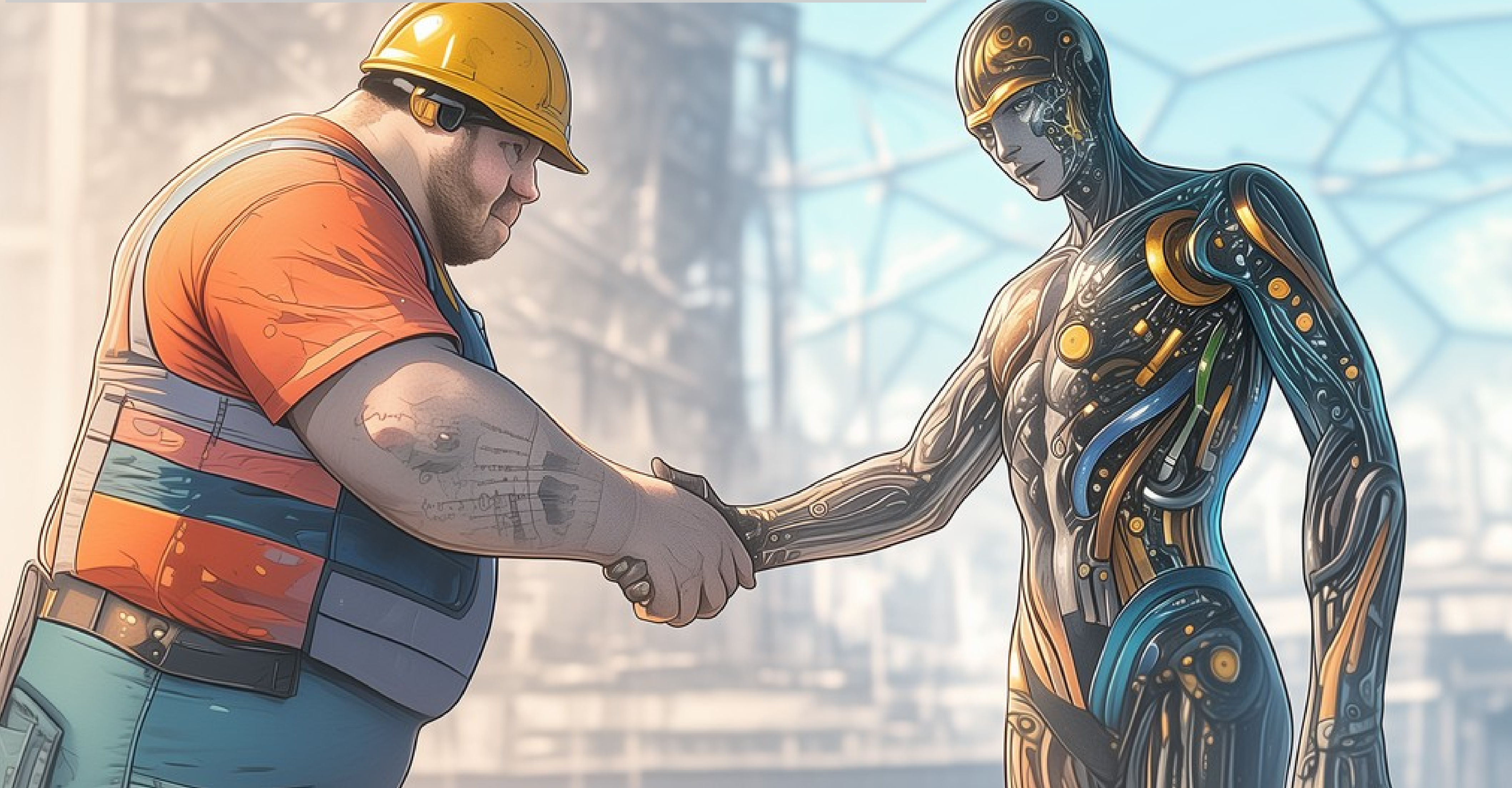
Fearless

Fearless

Fearless has two souls working together



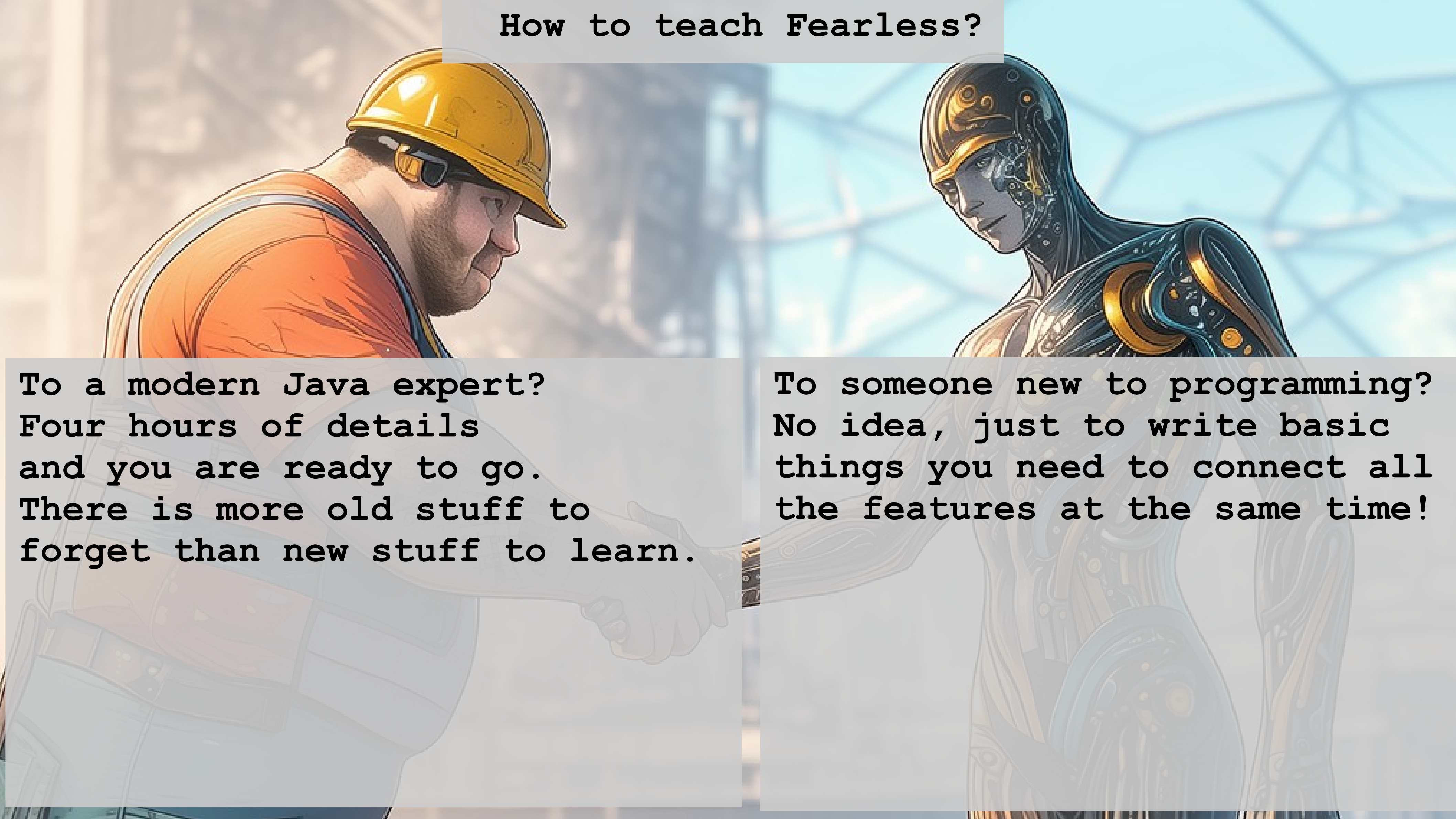
Fearless has two souls working together

Object oriented

Functional

You need to understand both mindsets to be effective in fearless.

How to teach Fearless?



To a modern Java expert?
Four hours of details
and you are ready to go.
There is more old stuff to
forget than new stuff to learn.

To someone new to programming?
No idea, just to write basic
things you need to connect all
the features at the same time!

booleans slide here

Embedded DSL

- A library define terms and how to combine them. It is a language nested inside the programming language.
- An **E**mbedded **D**omain **S**pecific **L**anguage is a library embracing this idea.
- Code using this library looks like is using a different language, with different rules and conventions.
- Example: Java streams
- Languages with minimal / flexible syntax are great for EDSLs

Find the tallest!

```
TallestPerson:Comparator[Person]{p1,p2->
  NumComparator.compare(p1.height, p2.height)
  }
```

```
//Declarative
  Find-The-TallestPerson-In-myList

//Declarative/Relational
  Query.findThe TallestPerson .in myList

//Functional/Declarative
  myList.flow.max TallestPerson .get

//Imperative
  Block#
    .var res = {myList.get(0)}
    .for i = {myList.range}
      .do{ res.set(TallestPerson.greater(res#,myList.get(i))) }
    .return {res}
```

Find the tallest!

```
TallestPerson:Comparator[Person]{p1,p2->
  NumComparator.compare(p1.height, p2.height)
  }
```

```
//Declarative
  Find-The-TallestPerson-In-myList

//Declarative/Relational
  Query.findThe TallestPerson .in myList

//Functional/Declarative
  myList.flow.max TallestPerson .get

//Imperative
  Block#
    .var res = {myList.get(0)}
    .for i = {myList.range}
      .do{ res.set(TallestPerson.greater(res#,myList.get(i))) }
    .return {res}
```

Find the tallest!

```
TallestPerson:Comparator[Person]{p1,p2->
  NumComparator.compare(p1.height, p2.height)
  }
```

```
//Declarative
  Find-The-TallestPerson-In-myList

//Declarative/Relational
  Query.findThe TallestPerson .in myList

//Functional/Declarative
  myList.flow.max TallestPerson .get

//Imperative
  Block#
    .var res = {myList.get(0)}
    .var i={1}
    .loop {Block#
      .if {i#>myList.size} .break
      .do {res.set(TallestPerson.greater(res#,myList.get(i#)))}
      .continue}
    .return {res}
```

Embedded DSL in Fearless

- flows
- block
- tests
- guis/forms/IQL
- regexes
- parsing
- webAPI <->

Example, some code to test

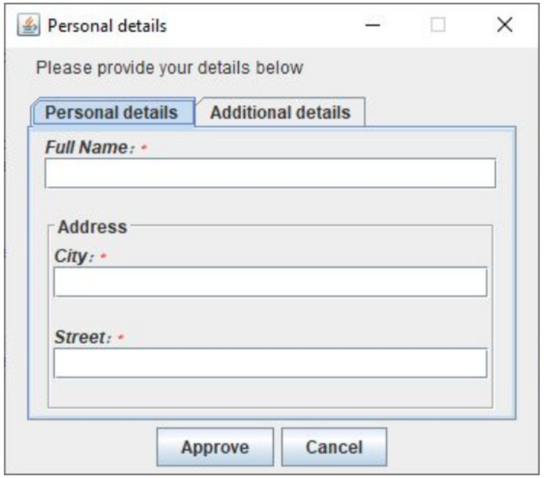
```
Fractions: F[Int,Int,Fraction]{num,den -> Fraction: {
  .numerator: Int -> num,
  .denominator: Int -> den,
  .divide: Float -> num.float / (den.float),
  .str:Str-> "Fraction["+num+", "+den+"]",
  }}
```

Fluent tests

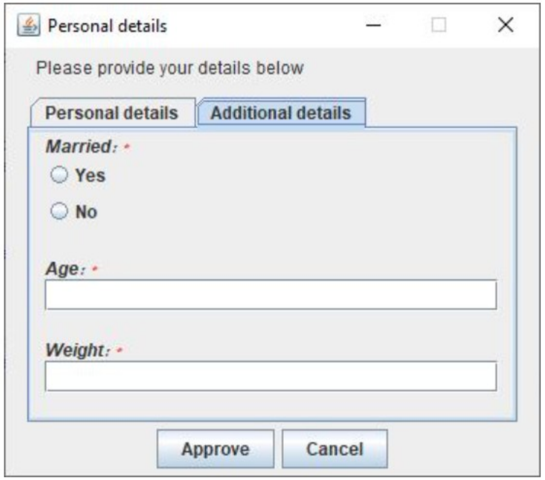
```
TestMyApp: TestMain{sys,runner->runner
  .withReporter(runner.stdOutPrinter(sys.io))

.test TestStaysPositive: Test{Block#
  .do {Assert!(Fractions#(+5, +4).divide > 0.0)}
  .do {Assert!(Fractions#(+1, +2).divide > 0.0)}
  .done
  }
.testLog TestGetsNegative: Test{log->Block#
  .let expected= {0.0}
  .do {log#{Assert!(Fractions#(+5, -4).divide < expected)}}
  .do {log#{Assert!(Fractions#(+1, -2).divide < expected)}}
  .done
  }
.test TestStrRepr: Test{Block#
  .do{"Fraction[+5,+4]".assertEq(Fractions#(+5, +4).str)}
  .done
  }
.testSuite MoreTestFractions
}
```

Fluent API Forms

```
Iql
.title "Personal details"
.single "Please provide your details below"
.tab("Personal details",
  Iql.entry("name", "Full Name:", "String"),
  Iql.group("Address",
     Iql.entry("city", "City:", "String"),
     Iql.entry("street", "Street:", "String")
     )
  )
.tab("Additional details",
  Iql.entry("married", "Married:", "Boolean"),
  Iql.entry("age", "Age:", "Integer"),
  Iql.entry("weight", "Weight:", "Decimal")
  )
.queryUser(system)//--> gives a flow of Map[String,Data]
```


Personal details

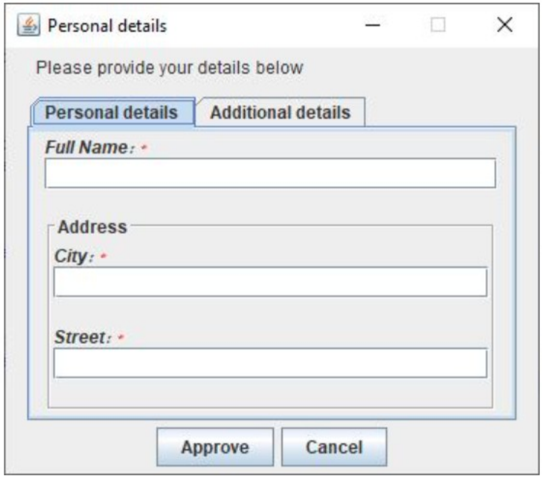
Please provide your details below


Personal details

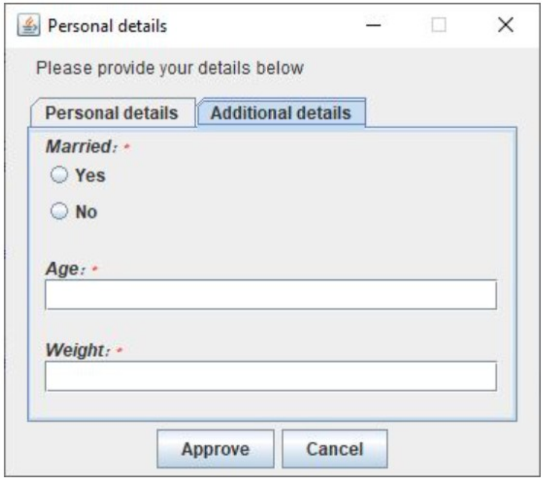
Please provide your details below

Fluent API Forms

```
Iql
.title "Personal details"
.single "Please provide your details below"
.tab "Personal details"
   .entry("name", "Full Name:", "String")
   .group "Address"
       .entry("city", "City:", "String"),
       .entry("street", "Street:", "String")
      !
   !
.tab "Additional details"
   .entry("married", "Married:", "Boolean")
   .entry("age", "Age:", "Integer")
   .entry("weight", "Weight:", "Decimal")
   !
.queryUser(system)//--> gives a flow of Map[String,Data]
```



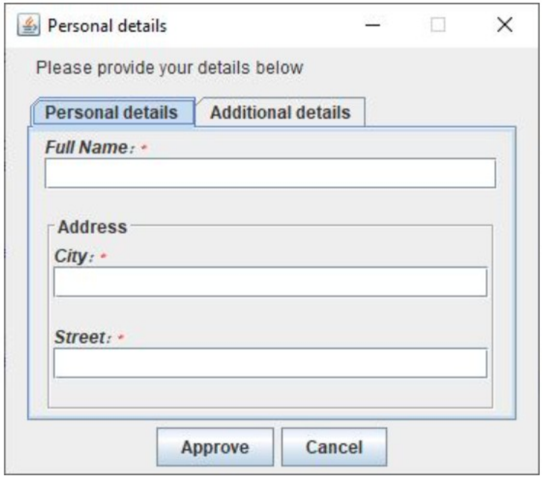
Please provide your details below

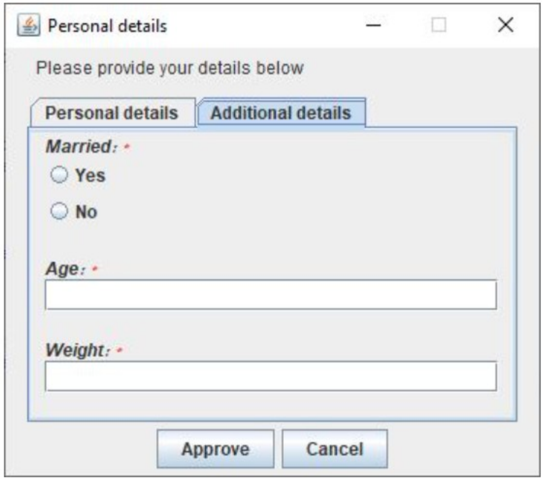


Please provide your details below

Fluent API Forms

```
Iql
.title "Personal details"
.single "Please provide your details below"
.tab{::#"Personal details"
  .entry("name", "Full Name:", "String")
  .group{::#"Address
      .entry("city", "City:", "String"),
      .entry("street", "Street:", "String")
      }
  }
.tab{::#"Additional details"
  .entry("married", "Married:", "Boolean")
  .entry("age", "Age:", "Integer")
  .entry("weight", "Weight:", "Decimal")
  }
.queryUser(system)//--> gives a flow of Map[String,Data]
```





Readable API for regexes

```
.let r = Regex
.let stringLit = r.sequence(
  r.chars"\"",
  r.any(   //.any is *, .many is +, .optional is ?
    r.alternative(
      r.anyCharExcept("\"", "\\"),
      r.sequence(
        r.chars"\\",
        r.alternativeStrings("\"", "\\", "n", "t")
        )
      )
    ),
  r.chars"\""
  )
```

equivalent to the unreadable `"(?:[^"\\]|\\["\\nt])*"`

A good API is a wall separating simplicity from thorny complexity

Parsing with Flows

```
App:Main{ sys -> Block#
  .let io = {sys.io}
  .let ns = {"123\n678\n"}
  .let parsed = {Tokenize#(ns)}
  .return {io.println(parsed.flow#(Flow.str "; "))}
  } //prints 123; 678

Tokenize:{#(str: Str): List[Str] -> str.flow
  .actor[mut Str, Str](iso"", {downstream, state, current -> Block#
    .if {current != "\n"}
      .return {Block#( state.add(current), ActorRes.continue)}
    .do {downstream#(state.str)}
    .do {state.clear}
    .return {ActorRes.continue}
    })
  .list
  }
```