Dependency Injection

Alexander Kuklev[1,2] ‹a@kuklev.com›

[1]Radboud University Nijmegen, Software Science
[2]JetBrains Research

External services and components can be framed global singletons like this:

```
object Database : DbConnection("jdbc:mysql://user:pass@localhost:3306/ourApp")
```

Global singletons are visible to one another and created on demand when first used, so no dependency injection is necessary. With this approach we have no boilerplate code whatsoever, but it has many drawbacks:

- it is not possible to read configuration from files and/or command line arguments,
- initialization happens in an uncontrolled manner,
- tight coupling prevents unit testing.

These issues are solved by introducing the application class which initialises necessary singletons in the right order according to their dependencies (which may include parallel initialisation for independent ones) and interconnects them. We believe that the language should provide a specialised syntax to make this approach a drop-in replacement for the naïve one with no syntactic penalty. We propose the following one:

```
// Declarations:
//   init Name(Dependencies) {initialiser}

init ConfigPath() = "./etc/config.yaml"

init Config(ConfigPath) = Yaml.fromFile(ConfigPath)

init Database(Config) = DbConnection(Config.dbConnString)

// Usage:
//   class Foo init(Dependencies) {
//     ..everywhere inside dependencies are available as
//       if they were introduced as objects inside Foo
//   }
// Transitive dependencies that are not listed are also
// initialized, but not accessible by name inside.

class OurApp(params) init(Config, Database) {
  ...
}

fun main() {
  OurApp(params).run()
}
// or
fun main(args : Array<String>) {
  OurApp(ConfigPath = args[0] if args.size > 0)
  .run()
}

class UnitTests {
  val app = App(params,
                ConfigPath = "tests/config.yaml",
                Database = MockDb)
  ... tests
}
```

// Inspired by Resource::Silo by https://github.com/dallaylaen