codecentric

Project Jigsaw_

Florian Troßbach

whoami

- codecentric Karlsruhe
- Plain Old Java Dev
- Currently trying to tame the SMACK stack

Services

Modules can provide services

- java.util.ServiceLoader

Method Summary

All Methods | Static Methods | Instance Methods | Concrete Methods

Modifier and Type	Method and Description
`Iterator<S>`	**`iterator`**`()` Lazily loads the available providers of this loader's service.
`static <S> ServiceLoader<S>`	**`load`**`(Class<S> service)` Creates a new service loader for the given service type, using the current thread's **context class loader**.
`static <S> ServiceLoader<S>`	**`load`**`(Class<S> service, ClassLoader loader)` Creates a new service loader for the given service type and class loader.
`static <S> ServiceLoader<S>`	**`loadInstalled`**`(Class<S> service)` Creates a new service loader for the given service type, using the extension class loader.
`void`	**`reload`**`()` Clear this loader's provider cache so that all providers will be reloaded.
`String`	**`toString`**`()` Returns a string describing this service.

Modules can provide services

- Modules provide service implementations
- Other modules can use services
- ServiceLoader finds all implementations on the module path
- No dependency on the providing modules needed => decoupling!

Example

```
module org.codefx.demo.advent {
        // list the required modules
        requires org.codefx.demo.advent.calendar;
        // list the used services
        uses org.codefx.demo.advent.surprise.SurpriseFactory;
}


module org.codefx.demo.advent.factory.quote {
        requires public org.codefx.demo.advent.surprise;
        // specify which class provides which service
        provides org.codefx.demo.advent.surprise.SurpriseFactory
                with org.codefx.demo.advent.factory.quote.QuoteFactory;
}
```

Example

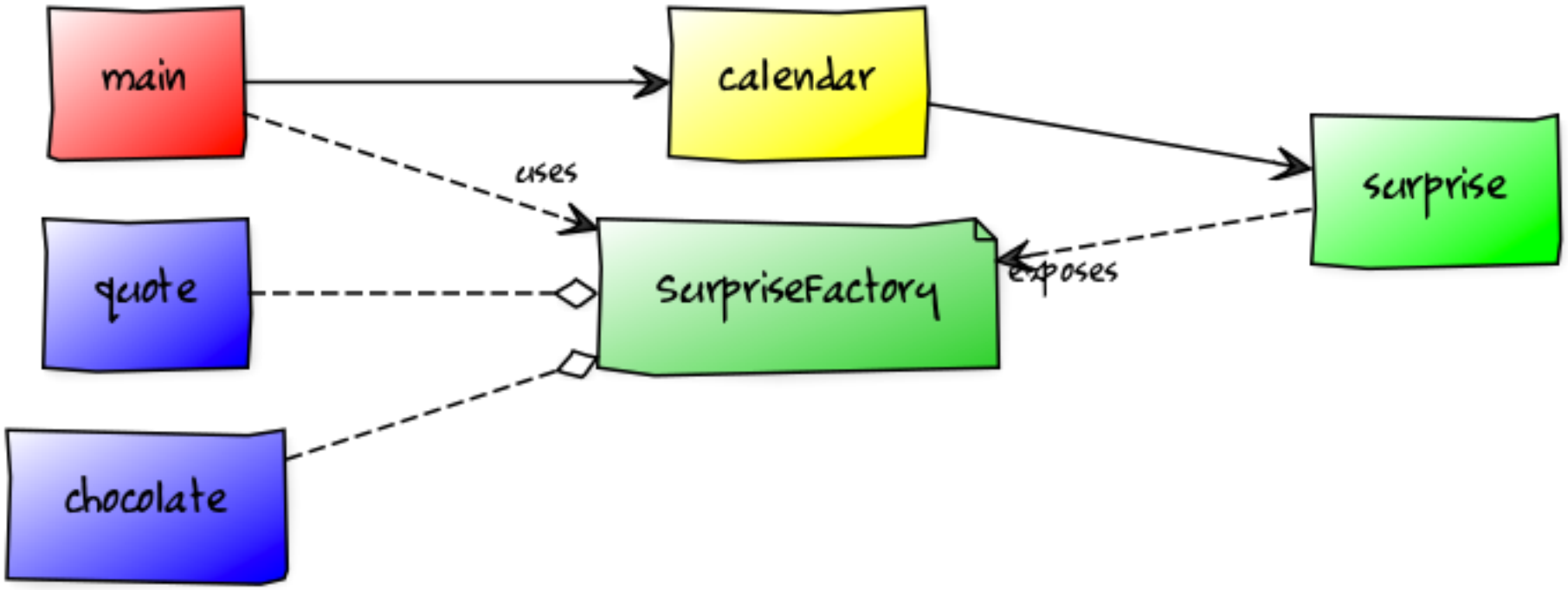


Diagram by Nicolai Parlog, http://blog.codefx.org/java/dev/jigsaw-hands-on-guide/

Working with non-modular code

Non-modular code

- Using libraries
  - Spring, Guava, ...
  - your internal library

- Using modular code from non-modular code

What is the name of an automatic module?

- The name of the jar
  - guava.jar => “guava”
  - ~~guava-19.0.jar~~

What does an automatic module export?

- All its packages
  - => All public types

What does an automatic module require?

- All exports of all modules on the module path
- All public types in the unnamed module

What is the name of the unnamed module?

- Any guesses?

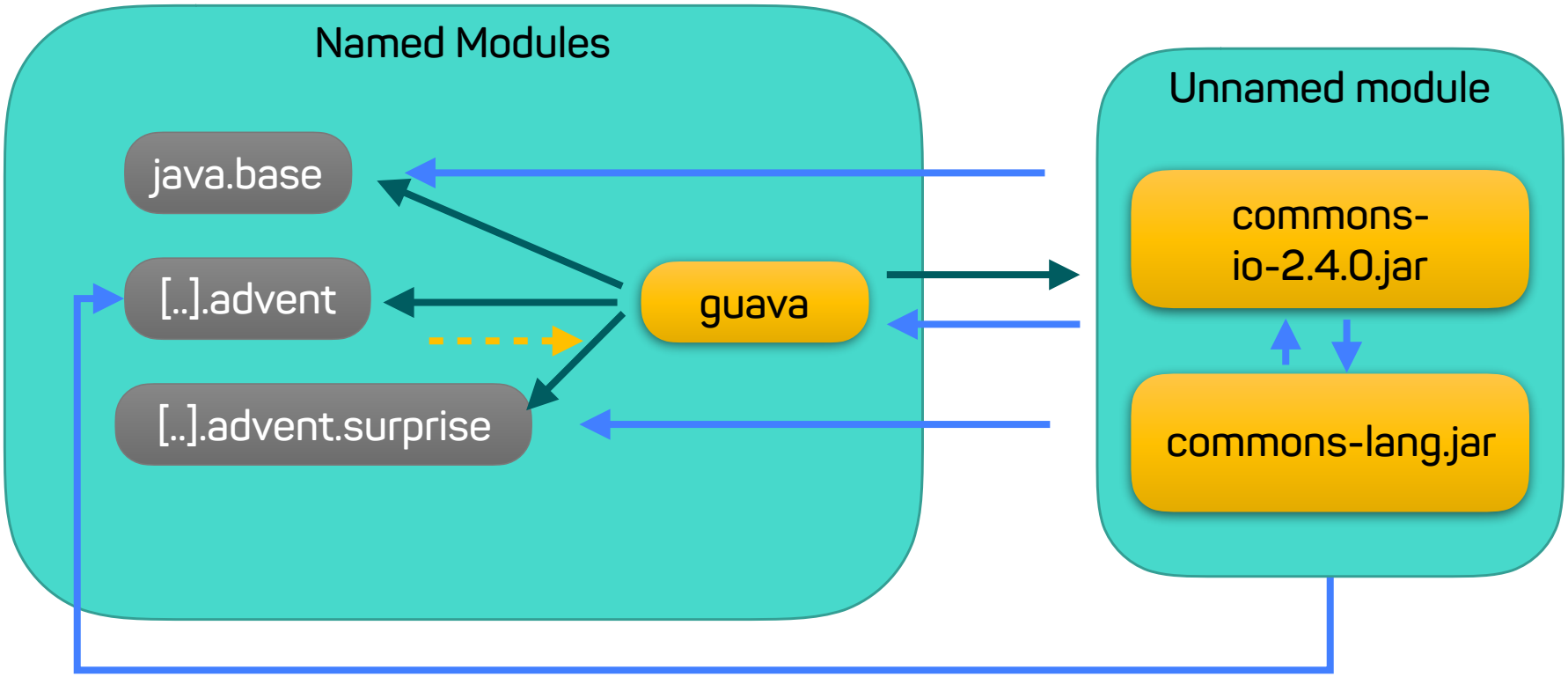
What does the unnamed module export?

- All its packages
- ~~`requires unnamed;`~~

What does an the unnamed module require?

- All exports of all modules on the module path

Overview


Named Modules
java.base
[..].advent
[..].advent.surprise
guava
Unnamed module
commons-
io-2.4.0.jar
commons-lang.jar

Example

Breaking Jigsaw - Live

jlink

Build your own JRE!

- Create your own Java runtime
- Pick which modules to include
- Can lead to really small distributions (< 15 MB)

Example

Resources

- https://github.com/ftrossbach/demo-jigsaw-advent-calendar

- Mark Reinhold, “The State of the Module System”
  - http://openjdk.java.net/projects/jigsaw/spec/sotms/

- Nicolai Parlog’s posts on Jigsaw
  - http://blog.codefx.org/tag/project-jigsaw/

- My blog posts on Jigsaw
  - https://blog.codecentric.de/en/2015/11/first-steps-with-java9-jigsaw-part-1/
  - https://blog.codecentric.de/en/2015/12/first-steps-with-java9-jigsaw-part-2/