Spring Framework 5.0 on JDK 8 & 9

Juergen Hoeller
Spring Framework Lead
Pivotal

Spring Framework 5.0 (Overview)

- **5.0 GA as of September 28th, 2017 – one week after JDK 9 GA!**
- **Embracing JDK 9 as well as Kotlin and Project Reactor**
- **Driven by functional API design and reactive architectures**

- **Major baseline upgrade: Java SE 8+, Java EE 7+**
  - JDK 8, Servlet 3.1, Bean Validation 1.1, JPA 2.1, JMS 2.0
  - support for JUnit 5 (next to JUnit 4.12)

- **Comprehensive integration with Java EE 8 API level**
  - Servlet 4.0, Bean Validation 2.0, JPA 2.2, JSON Binding API 1.0
  - e.g. Tomcat 9.0, Hibernate Validator 6.0, Apache Johnzon 1.1

spring by Pivotal

Many Community Contributions

spring-projects / spring-framework

Unwatch 3,084 | Star 20,052 | Fork 13,278

Code | Pull requests 170 | Wiki | Insights | Settings

Filters | is:pr author:igor-suhorukov | Labels | Milestones | New pull request

Clear current search query, filters, and sorts

0 Open | 50 Closed | Author | Labels | Projects | Milestones | Reviews | Assignee | Sort

- **Polish: all branches in a conditional structure should not have exactly the same implementation** ✓
  #1766 by igor-suhorukov was merged 2 days ago
- **Polish: remove redundant check** ✓
  #1765 by igor-suhorukov was merged 2 days ago
- **Polish: short-circuit logic should be used in boolean contexts** ✓
  #1764 by igor-suhorukov was merged 2 days ago
- **Polish: simplify boolean method return value** ✓ 2
  #1763 by igor-suhorukov was closed 2 days ago
- **Polish: remove unreachable statement** ✓ 1
  #1760 by igor-suhorukov was closed 2 days ago
- **Polish: nested "enum"s should not be declared static** ✓ 1
  #1759 by igor-suhorukov was closed 3 days ago
- **Polish: avoid unnecessary autoboxing** ✓
  #1757 by igor-suhorukov was merged 3 days ago

JDK 8

Rich Java 8 Development Support in Spring Framework 4.3

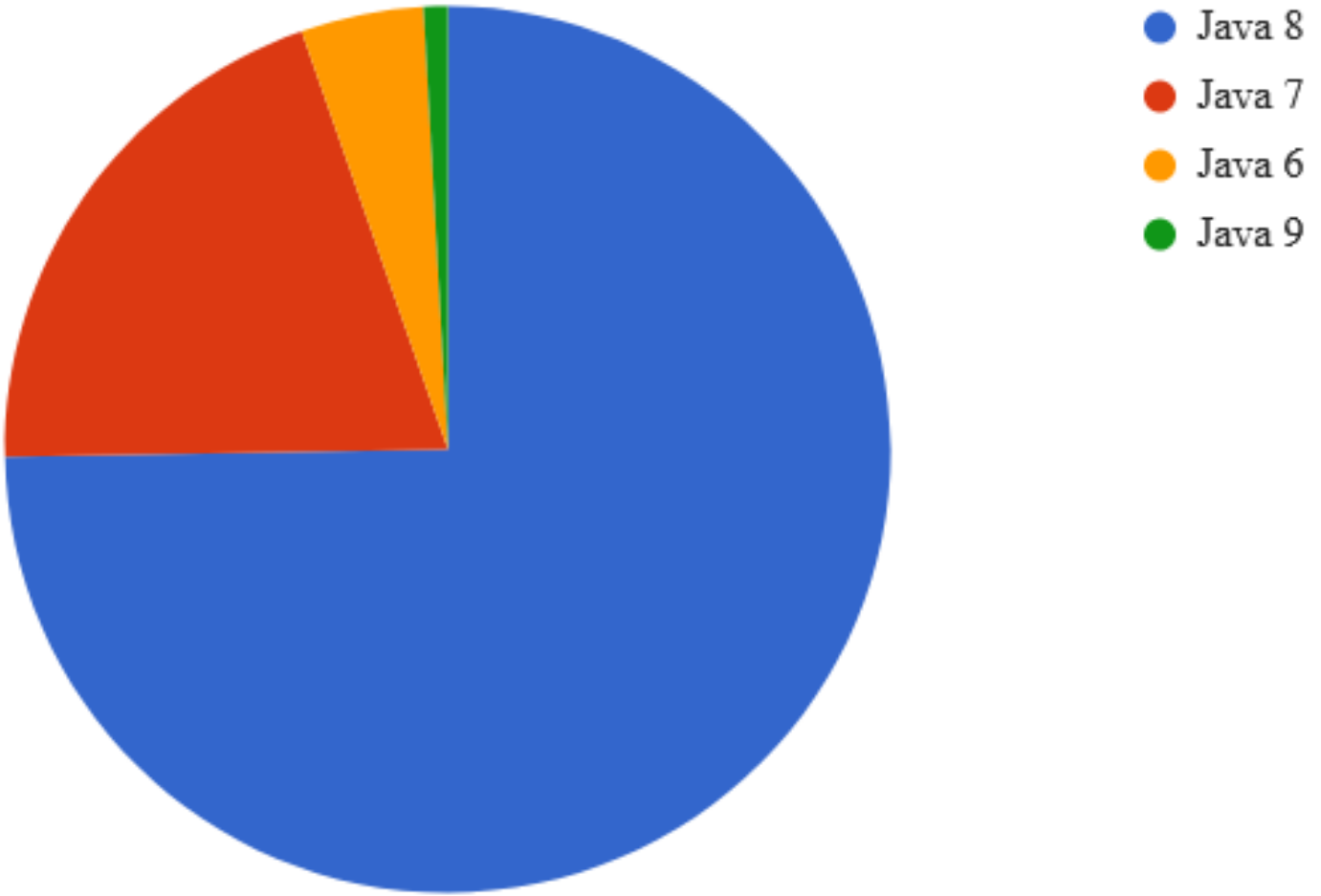
- **Spring Framework 4.3 delivers a rich Java 8 experience for applications**
  - despite being Java 6 based in the core framework itself
  - feels like a Java 8 based framework already (for many scenarios)

- **Reflectively adapting to the use of Java 8 constructs in user code**
  - injection points, handler method parameters
  - Optional, CompletableFuture, java.time, etc

- **Alignment with Java 8 conventions**
  - callback interfaces as “functional interfaces” for lambda expressions
  - repeatable annotation declarations (simply ignored on Java 6 & 7)

spring by Pivotal

Java 8 Adoption as of October 2017 (Survey by Baeldung)



Java 8+ Baseline in Spring Framework 5.0

- **Entire framework codebase is Java 8 based**
  - internal use of lambda expressions and collection streams
  - efficient introspection of constructor/method parameter signatures
- **Framework APIs can expose Java 8 API types**
  - Executable, CompletableFuture, Instant, Duration
  - java.util.function interfaces: Supplier, Consumer, Predicate
- **Framework interfaces make use of Java 8 default methods**
  - existing methods with default implementations *for convenience*
  - new methods with default implementations *for backwards compatibility*

ObjectProvider API Design with java.util.function

- `@Autowired ObjectProvider<MyBean> myBeanProvider`

- **Original ObjectProvider methods (dating back to 4.3)**
  - `T getIfAvailable()`
  - `T getIfUnique()`

- **Overloaded variants with java.util.function callbacks (new in 5.0)**
  - `T getIfAvailable(Supplier<T> defaultSupplier)`
  - `void ifAvailable(Consumer<T> dependencyConsumer)`
  - `T getIfUnique(Supplier<T> defaultSupplier)`
  - `void ifUnique(Consumer<T> dependencyConsumer)`

Programmatic Bean Registration with Java 8

```
// Starting point may also be AnnotationConfigApplicationContext
GenericApplicationContext ctx = new GenericApplicationContext();
ctx.registerBean(Foo.class);
ctx.registerBean(Bar.class,
        () -> new Bar(ctx.getBean(Foo.class)));

// Or alternatively with some bean definition customizing
GenericApplicationContext ctx = new GenericApplicationContext();
ctx.registerBean(Foo.class, Foo::new);
ctx.registerBean(Bar.class,
        () -> new Bar(ctx.getBean(Foo.class)),
        bd -> bd.setLazyInit(true));
```

Functional Web Endpoints with Method References

```
RouterFunction<?> router =
    route(GET("/users/{id}"), handlerDelegate::getUser)
    .andRoute(GET("/users"), handlerDelegate::getUsers);


public class MyReactiveHandlerDelegate {
    ...

    public Mono<ServerResponse> getUser(ServerRequest request) {
        Mono<User> user = Mono.justOrEmpty(request.pathVariable("id"))
            .map(Long::valueOf).then(this.repository::findById);
        return ServerResponse.ok().body(user, User.class);
    }

    public Mono<ServerResponse> getUsers(ServerRequest request) {
        Flux<User> users = this.repository.findAll();
        return ServerResponse.ok().body(users, User.class);
    }
}
```

Functional Web Endpoints in Lambda Style

```
UserRepository repository = ...;

RouterFunction<?> router =
    route(GET("/users/{id}"),
        request -> {
            Mono<User> user = Mono.justOrEmpty(request.pathVariable("id"))
                .map(Long::valueOf).then(repository::findById);
            return ServerResponse.ok().body(user, User.class);
        })
    .andRoute(GET("/users"),
        request -> {
            Flux<User> users = repository.findAll();
            return ServerResponse.ok().body(users, User.class);
        });
```

Java 8+ Open Source Frameworks in Q1 2018

- JUnit 5
- Reactor 3
- Tomcat 9
- Jetty 9.3 & 9.4
- Undertow 2.0
- Hibernate ORM 5.2 & 5.3
- Hibernate Validator 6.0

(certainly more to follow in the course of this year)

JDK 9+

JDK 9: Not Just Jigsaw

- **Many general JVM improvements**
  - Compact Strings, G1 by default, custom JVM distributions (jlink)
- **TLS stack ready for HTTP/2 out of the box**
  - e.g. for Tomcat 9 to enable HTTP/2 without JVM modifications
- **Jigsaw – module path as structured alternative to class path**
  - symbolic module names and requires/exports metadata for jar files
- **JDK 9 is GA as of late September 2017**
  - Spring 5 is fully aligned in terms of JDK 9's policies and constraints

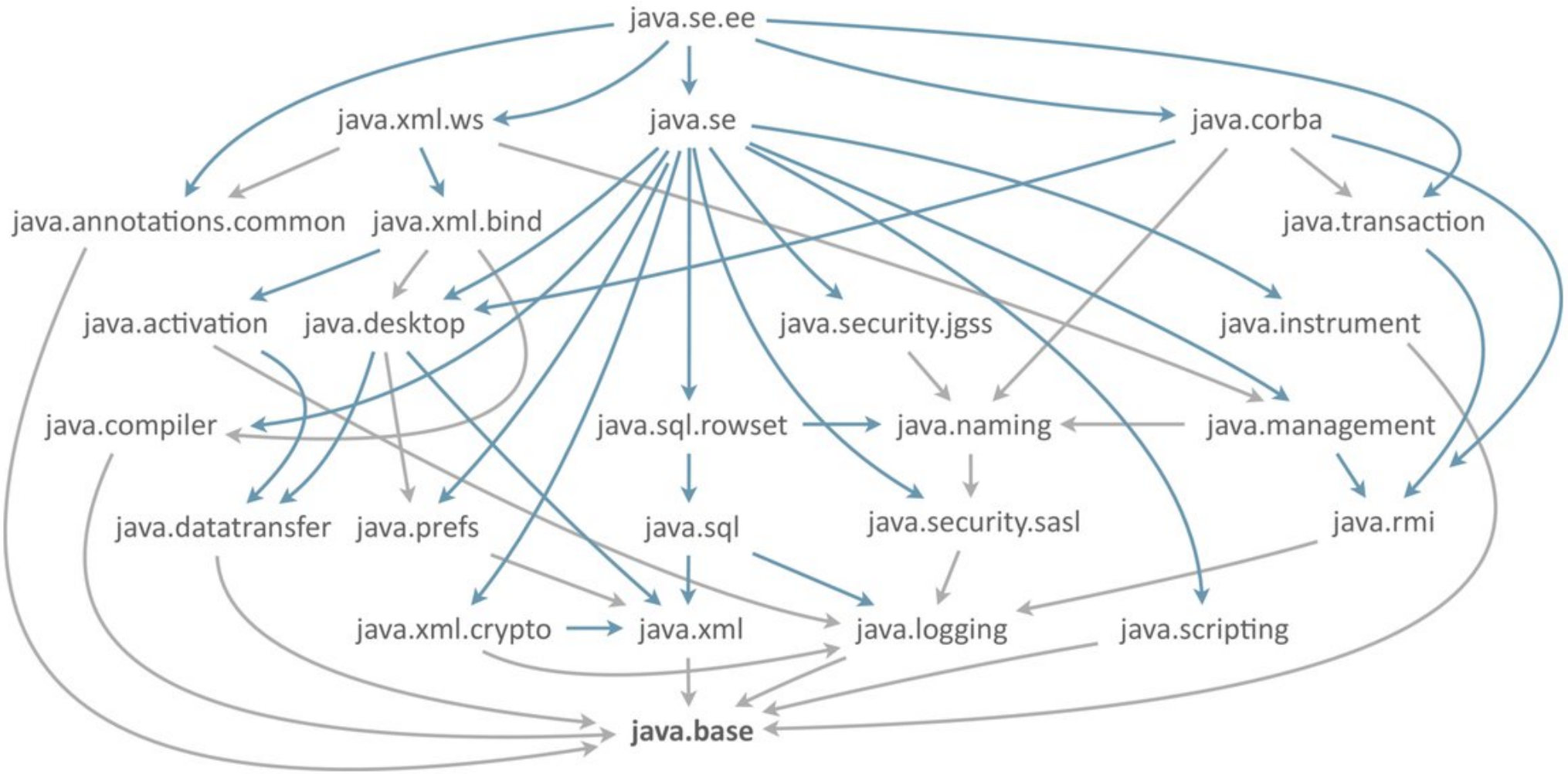
spring by Pivotal

JDK 9's Modular Nature

- **JDK 9 itself comes decomposed into several modules (as jmod files)**
  - java.base, java.sql, java.activation, java.logging, java.naming, java.xml.bind
  - Common Annotations deprecated (for JDK inclusion) in a separate module
- **Custom JVM distributions via jlink**
  - custom runtime image with selected modules plus dependencies
  - JAXB etc easily replaceable with standalone implementations
- **Frameworks have to revisit their dependency assumptions**
  - JUL, JNDI, JAXB, Common Annotations to be considered as optional
  - no unnecessary hard API references in framework APIs & SPIs

spring by Pivotal

JDK 9 Module Graph



Jigsaw for Application Modules

- **Effectively just a jar file with extra metadata**
  - coherent content with clean separation
  - no split packages, no cyclic dependencies
- **Explicit module with module-info descriptor**
  - required modules for dependencies (by module name)
  - exported and opened packages (by package name)
- **Automatic module**
  - “Automatic-Module-Name” manifest (or default name derived from jar file)
  - dependencies automatically resolved at runtime + all packages exported

Using Jigsaw with Spring

- **Framework jars as Jigsaw-compliant modules on the module path**
  - automatic modules with stable manifest module names (Spring 5, JUnit 5)
  - spring.context, spring.jdbc, spring.webmvc, spring.webflux

- **An application's module-info.java may refer to framework modules**

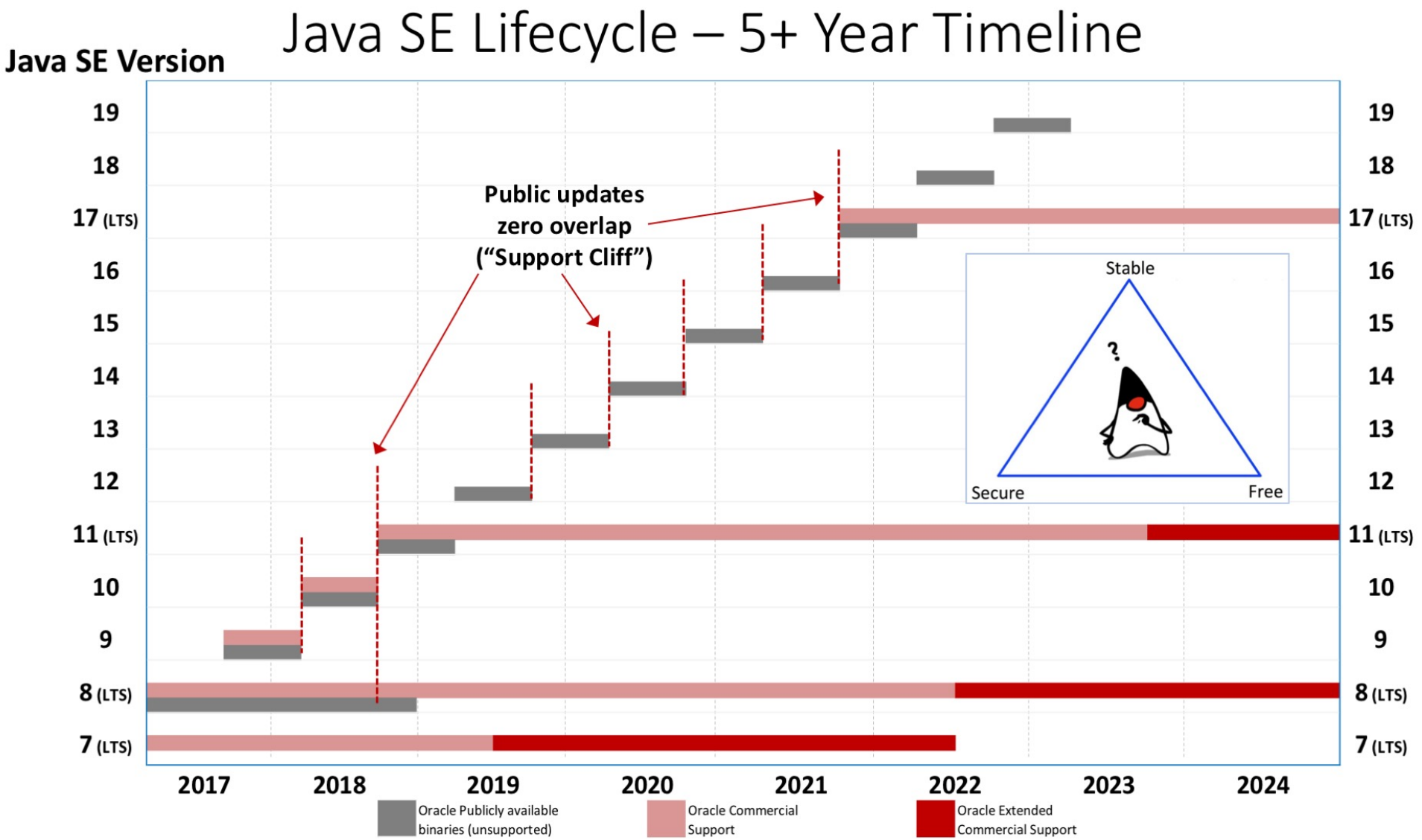
```
module my.app.db {
    requires java.sql;
    requires spring.jdbc;
    requires spring.context;

    exports my.app.db.service;
    exports my.app.db.util;
}
```

Reflection into Module-Contained Classes

- **Modules choose to 'export' all or selected packages**
  - 'export' means public types only
  - no access to internals, not even via reflection!
- **Application modules may choose to 'open' certain packages**
  - allowing for deep reflection through frameworks
  - accessing private field state, invoking non-public methods
- **For Spring purposes, application modules do NOT have to be 'open'**
  - as long as all relevant constructors and handler methods are public
  - 'open' declarations only needed for interaction with non-public artifacts

Long-Term Support: Oracle versus Azul



spring by Pivotal

Preparing for JDK 10 & 11

- **JDK 11 (GA in Sep 2018) will be the next long-term support release**
  - JDK 9 & 10 are effectively just intermediate releases on the way to 11
- **Limited technology support options towards 2019**
  - immediate upgrade to JDK 11, 12, 13, etc for free
  - commercial long-term support for JDK 8 (until 2025!!)
  - commercial long-term support for JDK 11 (until 2026)
- **The Spring Framework project (5.0.5+) is buildable on JDK 8, 9, 10**
  - Gradle build and default dependencies work on all three JDKs
  - same setup expected to work for JDK 11

Upgrade Considerations

- **Consider staying on the JVM classpath**
  - Spring 5 runs fine in classpath mode as well as on the module path
  - however, other libraries may not work in a module setup quite yet
  - Tomcat and co run in a custom class loader arrangement anyway

- **Consider staying at Java 8 bytecode level**
  - Spring 5's ASM 6.0 fork accepts Java 8 as well as Java 9+ bytecode level
  - ASM 5.x (very common) rejects unknown bytecode levels beyond Java 8
  - compiling your code with target 1.8 reduces the risk of such tools breaking

- **Build against JDK 8, run against JDK 9 / 10 / 11?**

Spring Framework 5.0

Q3 2017

JDK 8 baseline
support for JDK 9
prepared for JDK 10

integration with Java EE 8 APIs
functional style with Java & Kotlin
reactive web stack on Reactor

spring by Pivotal

Spring Framework 5.1

Q3 2018

full support for JDK 11
shipping with ASM 6.1
shipping with Reactor 3.2

new JDK 11 HttpClient API
Lookup.defineClass on JDK 9+
support for Hibernate ORM 5.3

spring by Pivotal