Ioannis Kostaras

MODERN JAVA
JAVA FEATURES 8 - 17

Agenda

- History
- Features:
  - Java 8
  - Java 9
  - Java 10
  - Java 11
  - Java 12
  - Java 13
  - Java 14
  - Java 15
  - Java 16
  - Java 17

Java overview

- Java is:
  - Object-oriented
  - Functional
  - Static (compiled) & Dynamic (interpreted)
  - Platform independent
  - Safe
  - Portable
  - Multi-threaded
  - Distributed

Java History

- Created in 1991 by James Gosling in Sun MicroSystems with the name *Oak*
- In 1994 it is renamed “Java” and targets Internet
- Netscape Navigator 2 and Internet Explorer support Java in 1995
- Java Development Kit 1.02 in 1996
- JavaBeans in JDK 1.1 in 1997

Java History (cont.)

- JDK 1.2 provides Swing for UI development
- JDK 1.4 provides JDBC
- JDK 5 supports Generics, Enumerated Types, autoboxing, varargs, annotations, static imports and new multi-threading library
- JDK 6 improves performance by 30%
- JDK 7 provides small improvements
- Sun Microsystems is acquired by Oracle

Editions

- Java Standard Edition (JSE)
- Java Enterprise Edition (JEE)
  - ➔ Jakarta EE
- Java Micro Edition (JME)

- JavaFX

IDEs

- BlueJ
- DrJava
- Eclipse
- IDEA
- JDeveloper
- Apache NetBeans or OpenBeans
- Visual Studio Code

Online compilers

- Browxy
- Codecademy
- CodeChef
- CodePad
- Codiva
- CompileJava
- Guru99
- Ideone
- JDoodle
- TryJShell - Online JShell
- Learn Java Online
- Paiza.io
- Repl.it
- Rextester
- Online Java Debugger
- Trinket
- Tutorialspoint
- Visualizer

Version overview

- JLS - Java Language Specification
- JSR - Java Specification Requests.
  - The formal documents that describe proposed specifications and technologies for additions to the Java platform.
- JEP - JDK Enhancement Proposal

* LTS (Long Term Support)

Features can be

- **Standard** ✔
- **Deprecated** ☒
- **Removed** ✗
- **Preview** 🔍 (`--enable-preview`)
  - fully specified and implemented, but not yet considered to be final. Almost complete, waiting for an additional round of real-world feedback.
  - have to be explicitly enabled.
- **Experimental** 💥
  - less stable, and more likely to change.
  - have to be explicitly enabled.
- **Incubator** 🥚 (`--add-modules`)
  - non-final tools and API's; distributed in separate modules.

Java 8* (18/03/2014)

- JVM
  - Remove Permanent Generation (JEP 122) ✗
- Language
  - Lambda Expressions (JSR 335) ✔
  - Default Methods in Interfaces (JSR 335) ✔
  - Effectively Final Variables (JSR 335) ✔
- API
  - Streams (java.util.stream) (JEP 107) ✔
  - Lambda APIs (java.util.function) (JEP 109) ✔
  - Date Time (java.time) (JSR 310, JEP 150) ✔

What is λ

- A λ (lambda) expression is an anonymous function that can be passed as argument or returned as the value of function calls.
- In the Java context they are similar to anonymous methods. Like a method it provides a list of formal parameters and a body—an expression or block—expressed in terms of those parameters
- They make it easier to distribute processing of collections over multiple threads
- Collection methods take a function and apply it to every element

λ-expressions

- `f(x) = y`, maps x -> y,

e.g. `f(x) = 2*x`, maps x -> 2*x

Parameter list

- `double dbl(double x) { return 2*x; }`

return type

param type

body

➔ `(double x) -> 2*x`

- Lambdas are lexically scoped, meaning that a lambda recognizes the immediate environment around its definition as the next outermost scope.

λ in Java

Syntax:

```
 (parameters) -> expression
or
 (parameters) -> { statements; }
```

E.g.

```
(int x, int y) -> x + y
(x, y) -> x % y
() -> Math.pi
(String s) -> s.toUpperCase()  or
String::toUpperCase
x -> 2 * x
c -> { int n = c.size(); c.clear(); return n ; }
```

No return

← *Method Reference*

Functional interfaces

```
@FunctionalInterface
public interface MyInterface<T>
{
    MyInterface<T> apply();
}
```

A *functional interface* is an interface that has just one abstract method (and zero or more default or implemented methods), and thus represents a functional contract.

java.util.function

- `Consumer<T>` `T -> void`
- `Supplier<T>` `() -> T`
- `Predicate<T>` `T -> boolean`
- `Function<T,R>` `T -> R`

What is a Stream

- Streams provide (optionally) ordered sequence of values without providing any storage for those values; they are just a means for expressing bulk data operations.
- In other words, they process data but not store them
- Operations of streams can be combined to create UNIX-like pipelines
- Streams from Collections, Files, Strings etc.

```
java.util.Stream
- IntStream
- LongStream
- DoubleStream
```

Creating streams

- How to create a stream:

```
java.util.Collection.stream();
java.util.Collection.parallelStream();
java.util.stream.Stream.of(T);
```

Transforming streams

- `Stream<T> filter(Predicate<T>);`
- `Stream<R> map(Function<T,R>);`
- `Stream<R> flatMap(Function<T,Stream<R>>);`
- `Stream<T> sorted(Comparator<T>);`
- `Stream<T> distinct();`
- `Stream<T> skip(long);`
- `Stream<T> limit(long);`

truncate

Ending streams

- `boolean anyMatch(Predicate<T>);`
- `boolean allMatch(Predicate<T>);`
- `boolean noneMatch(Predicate<T>);`

search

- `IntSummaryStatistics summaryStatistics();`

reduce (sum, min, count, …)

- `R collect(Collector<T,A,R>);`
  - `Collector<T, List<T>) toList();`
  - `Collector<T, Set<T>) toSet();`
  - `Collector<T, Map<K,U>) toMap();`

collect

- `void forEach(Consumer<T>);`

Stream examples

```
Predicate<Integer> even =
            i -> i % 2 == 0;
list.stream().filter(even)
    .forEach(System.out::println);
Function<Integer, Integer> dbl =
            i -> i * 2;
List<Integer> dblList =
list.stream().map(dbl)
   .collect(Collectors.toList());
```

Default methods

```
public interface AInterface {
  default String message() {
     return "Hallo A";
  }
}
```

Access modifiers

Access modifiers	*Supported?*	*Comments*
`public abstract`	Yes	≥ JDK 1
`public static`	Yes	≥ JDK 8
`public default`	Yes	≥ JDK 8
`private static`	Yes	≥ JDK 9, no override
`private`	Yes	≥ JDK 9, no override
`private abstract`	No	Must be overriden (`abstract`) but it cannot (`private`)
`private default`	No	Must be overriden (`abstract`) but it cannot (`private`)

java.util.Optional<T>

- Optional<T> // ifPresent()

E.g.

```
Optional<String> found =
prices.stream().
filter(name ->
    name.startsWith(letter)).
findFirst();
found.orElse("Not found!");
found.ifPresent(name ->
     name.append(”!"));
```

Java 9

(21/09/2017)

- Language
  - Module System (JEP 261) ✔
  - Private Methods in Interfaces (JEP 213) ✔
- API
  - Var Handles (JEP 193) ✔
  - Reactive Streams (JEP 266) ✔
  - HTTP 2 Client: An HTTP 2.0 Client for HTTP 2.0 and WebSockets (JEP 110) 🔍
- Tools
  - jshell (JEP 222) - Read-Eval-Print Loop ✔
  - Multi-Release JAR Files (JEP 238) ✔
  - jlink (JEP 282) ✔

Java 9 is Feature Complete!

- Modularity
  - 200: The Modular JDK (Jigsaw/JSR 376 and JEP 261)
  - 201: Modular Source Code
  - 220: Modular Run-Time Images
  - 238: Multi-Release JAR Files
  - 261: Module System
  - 275: Modular Java Application Packaging
  - 282: jlink: The Java Linker

Packages & Access modifiers

- Classes are arranged into packages
  - `com.company.app.MyClass` → `com/company/app/MyClass.java`
- Packages are globally visible and open for extension
- Unit of delivery is a Java archive (jar)
  - Access control is only managed in the level of classes/methods
- Classes and methods can restrict access by these access modifiers:
  - `public`
  - `protected`
  - `private`

Access modifier	Class	Package	Subclass	Unrestricted
public	✔	✔	✔	✔
protected	✔	✔	✔	
- (default)	✔	✔		
private	✔			

Packages & Access modifiers

- How do you access a class from another package, but preventing other classes from using it?
  - You can only make the class `public`, thus exposing it to all other classes ➔ **breaks encapsulation**
- No explicit dependencies
  - explicit import statements are only at compile time; there is no way to know which other JAR files your JAR needs at run-time; user has to provide correct jars in classpath during execution

➔ Maven or OSGi

- Maven solves compile-time dependency management by defining POM (Project Object Model) files. (Gradle works in a similar way)
- OSGi solves run-time dependencies by requiring imported packages to be listed as metadata in JARs, which are then called bundles

Classpath

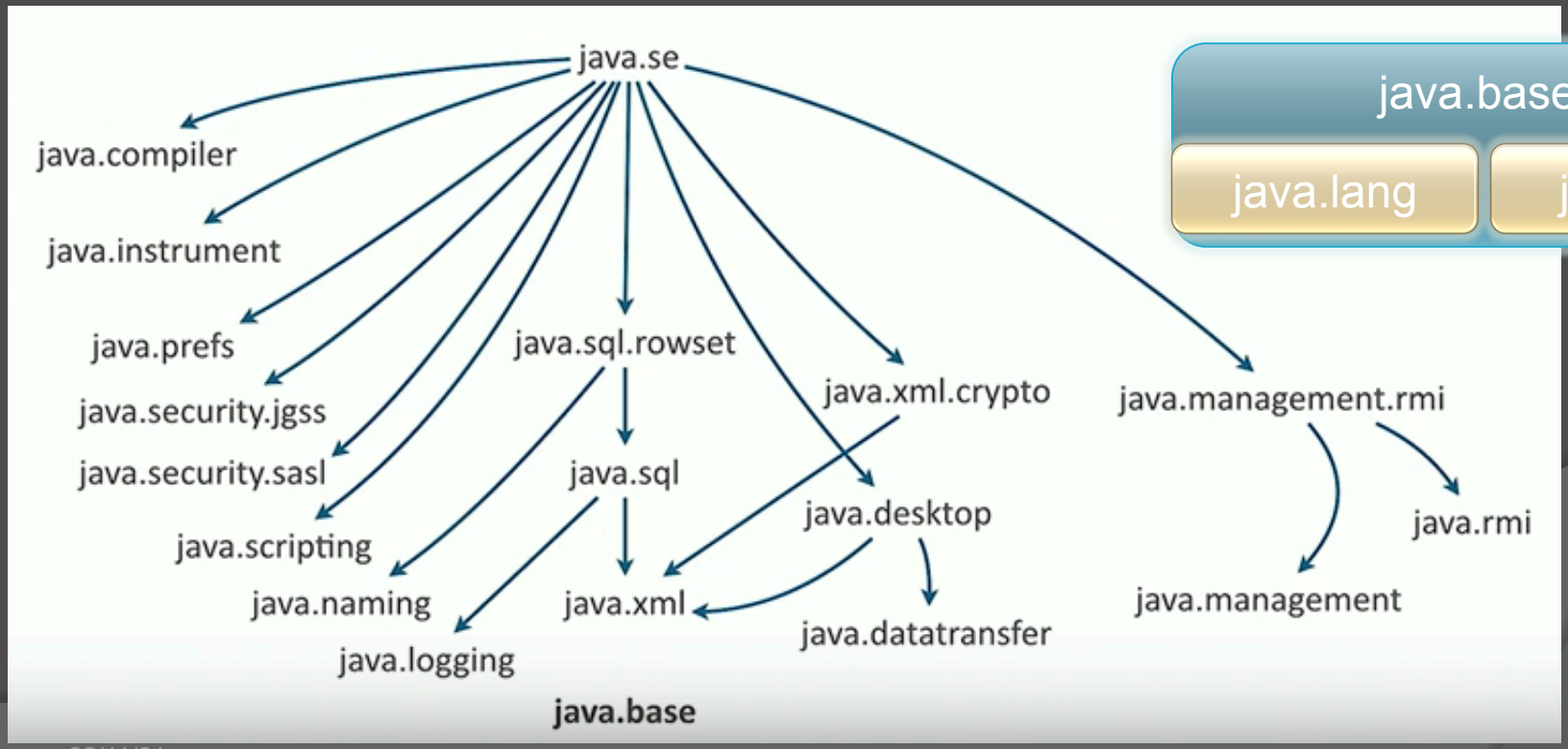
- Once a classpath is loaded by the JVM, all classes are sequenced into a flat list, in the order defined by the –`classpath` argument.
- When the JVM loads a class, it reads the classpath in fixed order to find the right one.
- As soon as the class is found, the search ends and the class is loaded. What happens when duplicate classes are in the classpath? ➔ Only one wins
- The JVM cannot efficiently verify the completeness of the classpath upon starting. If a class cannot be found in the classpath, then you get a run-time exception.
- The term “Classpath Hell” or “JAR Hell” should now be clearer to you

Modularisation & Modular Architecture

- *Characteristics of modular systems:*
  - **Strong encapsulation**: A module must be able to conceal part of its code from other modules. Consequently, encapsulated code may change freely without affecting users of the module.
  - **Well-defined interfaces**: modules should expose well-defined and stable interfaces to other modules.
  - **Explicit dependencies**: dependencies must be part of the module definition, in order for modules to be self-contained. A *module graph*: *nodes* represent *modules*, and *edges* represent *dependencies* between modules

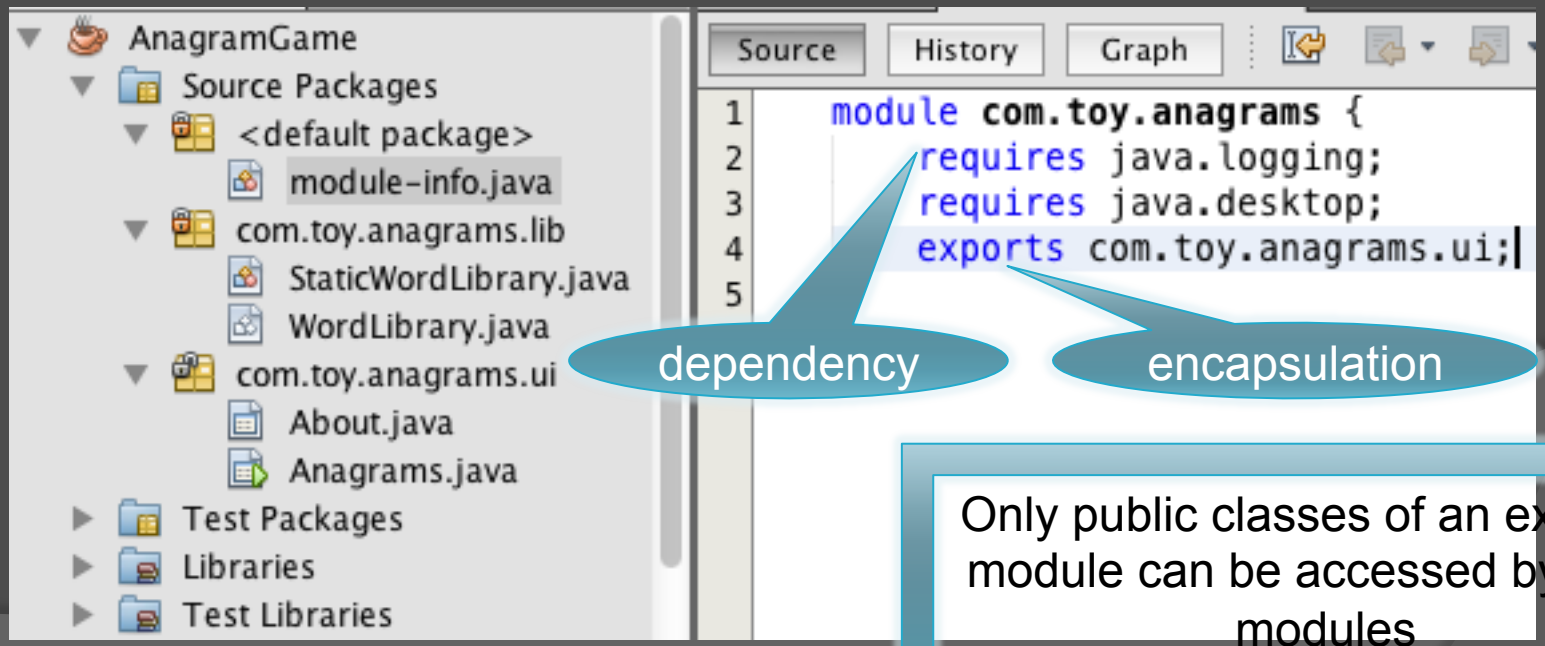
JDK 9 Platform Modules

- Module `java.base` exposes packages `java.lang`, `java.util` etc. It is the core Java module which is imported by default
- JDK now consists of 19 platform modules



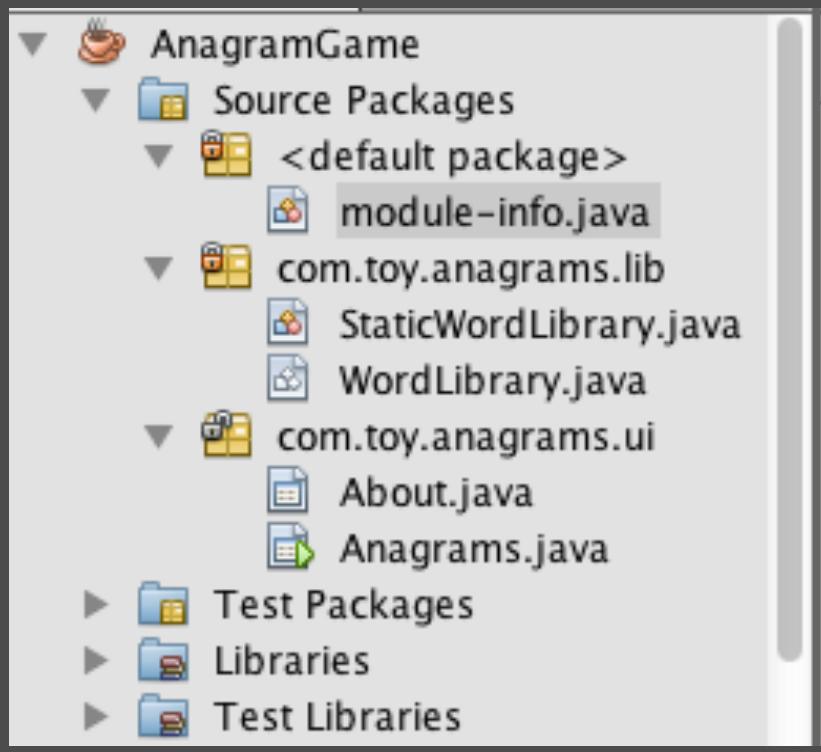
Modules in Java 9

- A module has a *name* (e.g. `java.base`), it groups related code and possibly other resources, and is described by a *module descriptor*.
- Like packages are defined in `package-info.java`, modules are defined in `module-info.java` (in root package)
- A *modular* jar is a jar with a `module-info.class` inside it



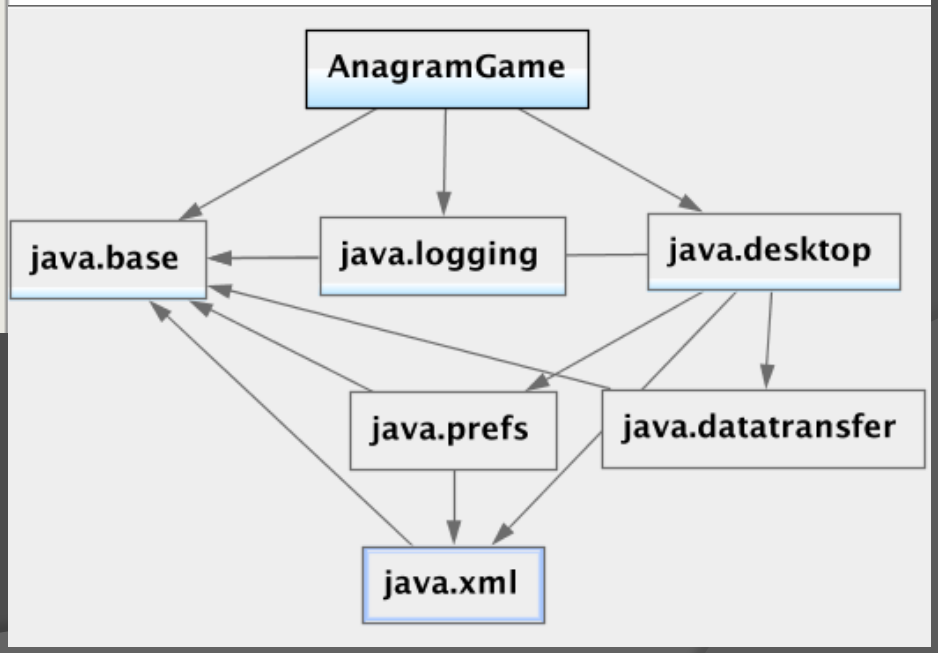
Only public classes of an exported module can be accessed by other modules

module-info.java

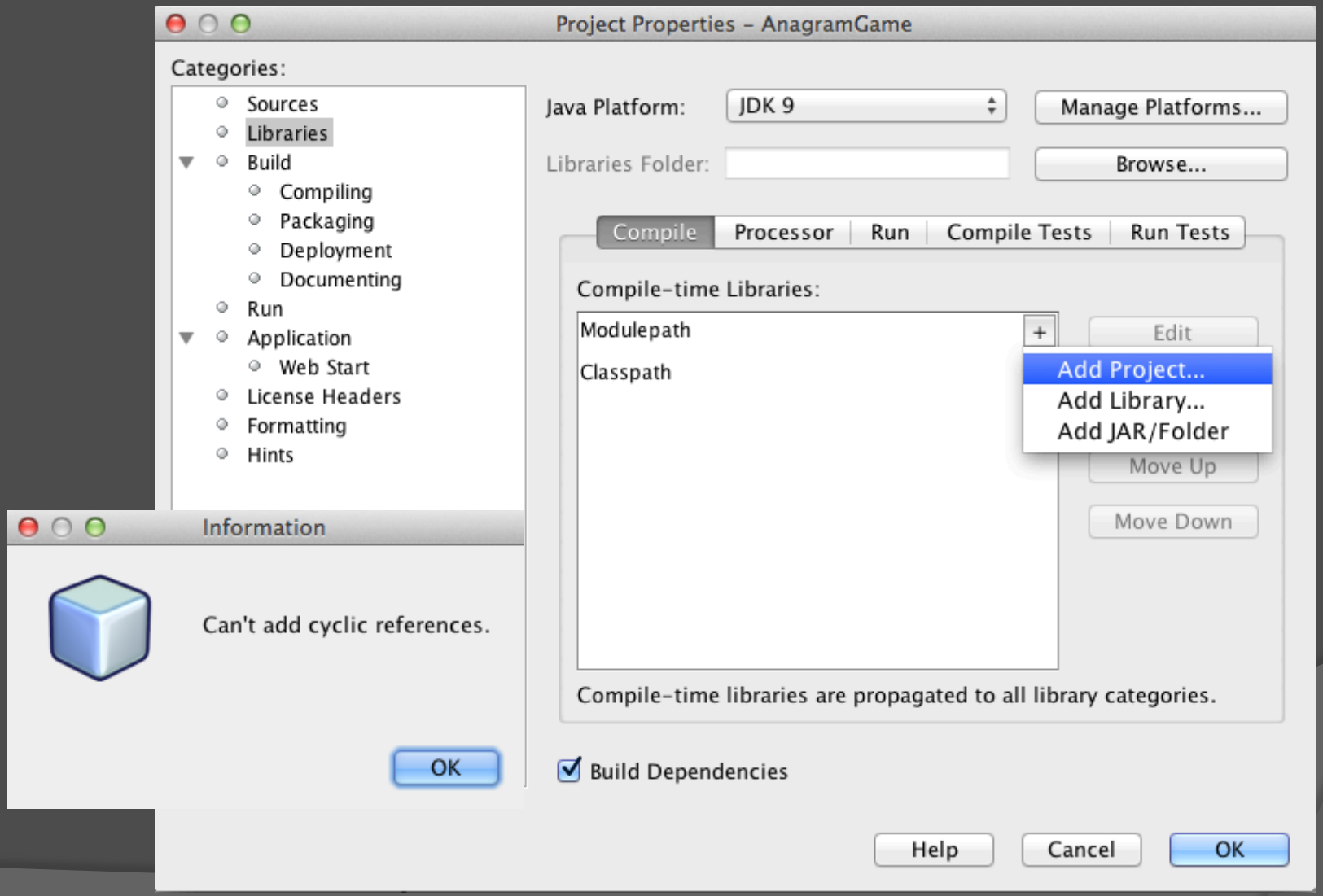


Source History Graph

```
1  module com.toy.anagrams {
2      requires java.logging;
3      requires java.desktop;
4      exports com.toy.anagrams.ui;
5  }
6
```



Module dependencies



Java 10 (12 JEPs) (20/03/2018)

- JVM
  - Garbage Collector Interface ✔
  - G1 improvements (parallel full GC) ✔
  - Graal VM (JEP 317) ✔
- Language
  - Local variable type inference (`var`) ✔
    - `var num = 42;`
- API additions ✔
  - `Collectors.toUnmodifiableList()`
  - `Collectors.toUnmodifiableSet()`
  - `Collectors.toUnmodifiableMap()`
  - Additional Unicode Language-Tag Extensions (JEP 314)

Local variable type inference

- `var userLists = new HashMap<User, List<String>>();`

Compilation errors:

- `var num; // uninitialized`
- `public class MyClass {`
  `var aField = calc(x, y);`
- `public MyMethod(var v) {}`

Java 11* (17 JEPs) (25/09/2018)

- JVM
  - ZGC 💥 concurrent, NUMA-aware, scalable low-latency low-pause (<10ms) garbage collector
  - Epsilon 💥 no-op garbage collector
  - Low-Overhead Heap Profiling via JMTI (**JEP 331**)
- Language
  - HTTPClient (**JEP 320**) ✔
  - Java EE and CORBA removed modules (**JEP 320**) ✗
  - Unicode 10 Support (**JEP 327**) ✔
  - Nashorn JavaScript Engine deprecated (**JEP 335**) ☒
  - New Cryptographic Algorithms (**JEP 324** , **JEP 329**) ✔
  - TLS 1.3 (**JEP 332**) ✔
- API
  - Local variable type inference (**JEP 327**) ✔
    - With support for lambdas

```
(var fName, var lName) -> fName + lName;
```

- Tools
  - Flight Recorder (**JEP 328**) ✔

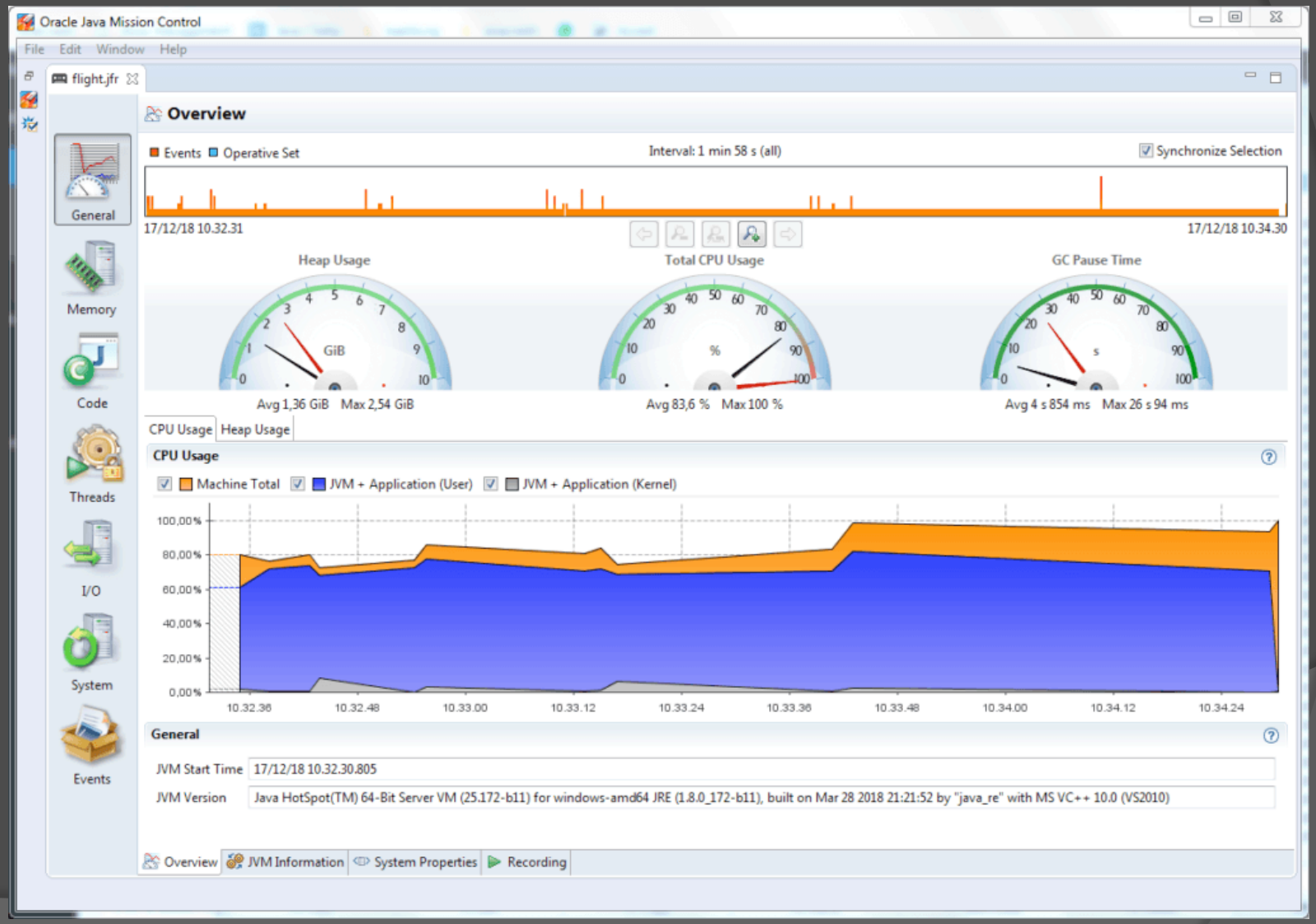
HTTPClient (9 → 11)

```
HttpClient httpClient =
     HttpClient.newBuilder().build();

HttpRequest request =
  HttpRequest.newBuilder()
    .uri(URI.create("https://amazon.com/"))
    .GET()
    .build();

HttpResponse<String> response =
  httpClient.send(request,
BodyHandlers.ofString());
```

Java Flight Recorder



Java 12 (8 JEPs)

(19/03/2019)

- **JVM**
  - G1 improvements ✔
    - 1
    - 2
  - Shenandoah GC (**JEP 189**) 💥
    - low-pause garbage collector
- **Language**
  - Switch Expressions (**JEP 325**) 🔍
- **API**
  - JVM Constants (**JEP 334**)
- **Tools**
  - Microbenchmark Suite based on JMH ✔

Switch Expressions (12 → 13 → 14)

```
int numDay = switch (day) {
    case MONDAY, FRIDAY, SUNDAY -> 6;
    case TUESDAY                -> 7;
    default       -> {
      String s = day.toString();
      int result = s.length();
      yield result;
    }
};
```

Java 13 (5 JEPs) (17/09/2019)

- **JVM**
  - Dynamic CDS Archives (**JEP 350**)
  - ZGC: Uncommit Unused Memory (**JEP 351**)
  - Linux/AArch64 support for ZGC ✔
- **Language**
  - Switch Expressions (**JEP 354**) 🔍
  - Text Blocks (**JEP 355**) 🔍
- **API**
  - Reimplement the Legacy Socket API (**JEP 353**) 🔍

Text blocks (13 → 14 → 15)

```
String html = """
     <html>
         <body>
             <p>Hello, world</p>
         </body>
     </html>
""";
```

- No entire text block on the same line
- `\` (suppresses `\n`) and `\s` (prevent the stripping of trailing white spaces)

Java 14 (16 JEPs) (17/03/2020)

- **JVM**
  - JFR Event Streaming (JEP 349)
  - Helpful `NullPointerExceptions` (JEP 358) with extra hints what caused it
  - CMS GC ✗
  - Solaris and SPARC Ports ☒
  - Parallel Scavenge + SerialOld GC (JEP 366) ☒
- **Language**
  - Switch Expressions (JEP 361) ✔
  - Records (JEP 359) 🔍
  - Text Blocks (JEP 368) 🔍
  - Pattern Matching for `instanceof` (JEP 305) 🔍

Java 14 (cont.) (17/03/2020)

- **API**
  - Non-Volatile Mapped Byte Buffers (**JEP 352**)
  - Foreign memory access API 🔍
    - to safely and efficiently use off-heap memory
    - Solves the `ByteBuffer` vs `Unsafe` dilemma
- Tools
  - Packaging Tool supporting native package formats 🥚
  - `msi`, `exe`, `pkg`, `dmg`, `deb` and `rpm`

Records (14 → 15)

```
public record Point(int x, int y) {
}

Point p = new Point (0, 0);
int x = p.x();
```

- Records are *named tuples*
- Records are immutable
- Compact constructors can be defined
- No inheritance (`extends`) but they can implement interfaces

Pattern matching of `instanceof`

```
if (obj instanceof String) {
    String s = (String) obj;
    System.out.println(s.length());
}
---> jdk 14
if (obj instanceof String s) {
    System.out.println(s.length());
}
```

`s` is final and it is scoped only in this block

Helpful `NullPointerExceptions` with extra hints what caused it

- Enabled with `-XX:+ShowCodeDetailsInExceptionMessages` in JDK 14

```
a.b.c.i = 99;
---
Exception in thread "main"
java.lang.NullPointerException:
 Cannot read field "c" because "a.b" is null
```

Java 15 (14 JEPs) (17/09/2020)

- **JVM**
  - Disable and Deprecate Biased Locking (JEP 374) ✗
  - ZGC (JEP 377) ✔
  - Shenandoah GC (JEP 379) ✔
  - Remove the Solaris and SPARC Ports (JEP 381) ✗
- **Language**
  - Sealed Classes (JEP 360) 🔍
  - Pattern Matching for `instanceof` (JEP 375) 🔍
  - Text Blocks (JEP 378) ✔
  - Records (JEP 384) 🔍
  - Local Interfaces and Enums (JEP 384) 🔍

Java 15 (14 JEPs) (17/09/2020)

- **API**
  - Edwards-Curve Digital Signature Algorithm (EdDSA) (**JEP 339**)
  - Hidden Classes (**JEP 371**) ✔
    - cannot be used directly by the bytecode of other classes
  - Remove the Nashorn JavaScript Engine (**JEP 372**)
  - Foreign-Memory Access API (Second Incubator) (**JEP 383**)
  - Deprecate RMI Activation for Removal (**JEP 385**)
- **Internal**
  - Reimplement the Legacy `DatagramSocket` API (**JEP 373**)

Sealed classes (15 🔍)

- can restrict which other classes may extend them

```
public abstract sealed class Shape
    permits Circle, Rectangle {...}

public class Circle extends Shape {...}
public class Rectangle extends Shape {...}
public class Triangle extends Shape {...}
// No need for default case
double area = switch (shape) {
  case Circle c -> Math.pow(c.radius(),
2)*Math.PI
  case Rectangle r -> r.a() * r.b()
};
```

final, sealed, non-sealed

Java 16 (17 JEPs) (15/03/2021)

- **JVM**
  - ZGC: Concurrent Thread Processing (**JEP 376**)
  - Alpine Linux Port (**JEP 386**)
  - Windows/AArch64 Port (**JEP 388**)
  - Strongly Encapsulate JDK Internals by Default (**JEP 396**)
  - Elastic metaspace (JEP 387)
    - to minimize metaspace waste
- **Language**
  - Warnings for Value-Based Classes (**JEP 390**) ✔
  - Pattern Matching for `instanceof` (**JEP 394**) ✔
  - Records (**JEP 395**) ✔
  - Static Members in Inner Classes (**JEP 395**) ✔
  - Sealed Classes (**JEP 397**) 🔍

Java 16 (17 JEPs) (cont.) (15/03/2021)

- **API**
  - Unix-Domain Socket Channels (**JEP 380**)
    - `java.net.UnixDomainSocketAddress`
  - Vector API (JEP 338) 🥚
    - provides vector computation (SIMD)
  - Foreign Linker API (JEP 389) 🥚
    - a pure-Java way to access native code, replacing JNI
- **Tools**
  - Packaging Tool (**JEP 392**) ✔
    - `jdk.jpackage`
- **Internal**
  - Enable C++14 Language Features (**JEP 347**)
  - Migrate to Git/GitHub (**JEP 357**,**JEP 369**)

Foreign Linker API (16)

```
#include <stdio.h>
void helloworld() {
    printf("hello world!\n");
}
$ gcc -c helloworld.c
$ gcc -shared -o helloworld.so
helloworld.o
```

Foreign Linker API (cont.)

```
var lib =
LibraryLookup.ofPath(Path.of("helloworld.so"));
var sym = lib.lookup("helloworld").get();
var fd = FunctionDescriptor.ofVoid();
var mt = MethodType.methodType(Void.TYPE);
var mh = CLinker.getInstance().downcallHandle(
                sym.address(),
                mt,
                fd);
mh.invokeExact();
```

```
javac --add-modules jdk.incubator.foreign
HelloWorldC.java
java --add-modules jdk.incubator.foreign
  -Dforeign.restricted=permit HelloWorldC
```

Java 17* (14 JEPs) (14/09/2021)

- **JVM**
  - New macOS Rendering Pipeline (**JEP 382**)
    - Uses Apple's Metal API for the Java2D API
  - macOS/AArch64 Port (**JEP 391**)
  - Strongly Encapsulate JDK Internals (**JEP 403**)
- **Language**
  - Pattern Matching for `switch` (**JEP 406**)
  - Sealed Classes (**JEP 409**)

Java 17* (14 JEPs) (cont.) (14/09/2021)

- **API**
  - Enhanced Pseudo-Random Number Generators (**JEP 356**)
  - Deprecate the Applet API for Removal (**JEP 398**) ☒
  - Remove RMI Activation (**JEP 407**) ✗
  - Deprecate the Security Manager for Removal (**JEP 411**) ☒
  - Vector API (JEP 414) ⬮
    - SIMD (single instruction multiple data) type of operation
  - Foreign Function & Memory API (JEP 412) ⬮
- **Tools**
  - Remove the Experimental AOT and JIT Compiler from GraalVM (**JEP 410**) ✗

Pattern matching for Switch Expressions

```
public String checkShape(Shape shape) {
    return switch (shape) {
        case Triangle t &&
(t.getNumberOfSides() == 3)-> "Triangle";
        case Circle c -> "Circle";
        case null -> null;
        default -> "";
    };
}
```

Future?

- Project Loom: lightweight or virtual threads (fibers); many lightweight threads to share a single OS thread
- Project Valhalla: value type is a new form of data type that is programmed like objects but accessed like primitives. => `List<int>`
- Project Amber
  - Pattern matching for records and arrays (Java 18)

GCs

- Java < 8
  - Serial
  - Parallel
  - ParallelOld
  - CMS
  - iCMS
- Java 8
  - G1
- Java 9
  - iCMS ✗
  - G1 (default)
- Java 11
  - Epsilon
  - Z 💥
- Java 12
  - Shenandoah 💥
- Java 14
  - CMS ✗
- Java 15
  - Shenandoah ✔
  - Z ✔
- Java 16
  - Z: Concurrent Thread Processing ✔

Further reading

- Java Magazine
- Java Almanac
- Foojay
- The Java Tutorial
- Kostaras I.N. (2018), Εισαγωγή στη γλώσσα προγραμματισμού Java, Mathesis.
- Bloch J. (2018), *Effective Java*, 3rd Edition, Addison-Wesley.
- Darwin I. F. (2014), *Java Cookbook*, 3rd Ed., O' Reilly.
- Evans B. J., Flanagan D. (2019), *Java in a Nutshell*, 7th Ed., O' Reilly.
- Juneau J. (2017), *Java 9 Recipes*, 3rd Ed., APress.
- Sharan K. (2017), *Java 9 Revealed: For Early Adoption and Migration*, Apress.
- Τζίτζικας Γ. (2016), Μια Γεύση από Java, LeanPub.
- Kabutz H. (2002), "Once Upon an Oak…", JavaSpecialists Newsletter 055.

Further reading (cont.)

- Inside Java 15: Fourteen JEPs in five buckets
- Modern Java toys that boost productivity, from type inference to text blocks
- Records Come to Java
- Inside the Language: Sealed Types
- Pattern Matching for instanceof in Java 14
- What is new in Java 16 ?
- New Features in Java 17
- Java 17 is here: 14 JEPs with exciting new language and JVM features
- Amber, Lambda, Loom, Panama, Valhalla: The major named Java projects
- OpenJDK wiki