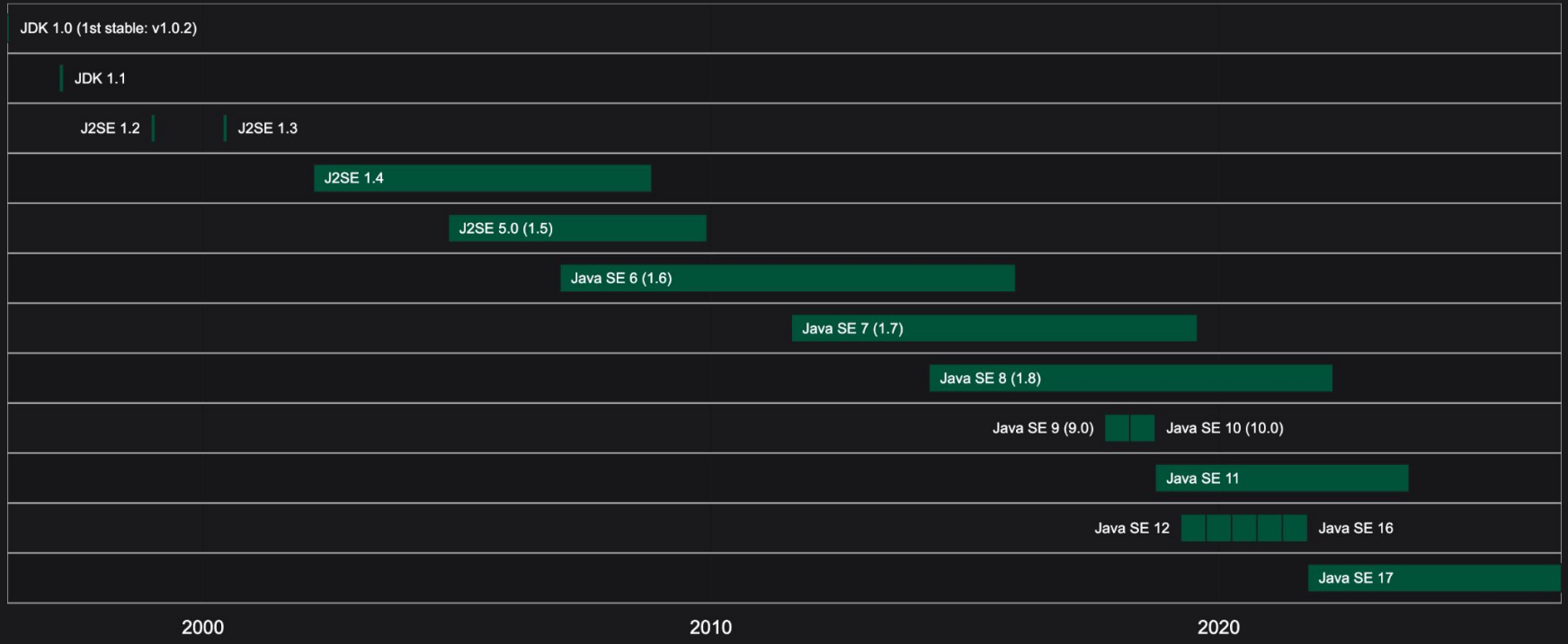
USF UNIVERSITY OF SAN FRANCISCO
CHANGE THE WORLD FROM HERE

Brief History of Java

JDK 1.0 to Java SE 17

Professor Sophie Engle
Department of Computer Science



https://jsfiddle.net/sjengle/6759bwtj/show • https://www.oracle.com/java/technologies/java-se-support-roadmap.html

https://jsfiddle.net/sjengle/6759bwtj/show • https://www.oracle.com/java/technologies/java-se-support-roadmap.html

JDK 1.1 to J2SE 1.4

- **JDK 1.1** (1997)**:** AWT event model, inner classes, Java Database Connectivity (JDBC)
- **J2SE 1.2** (1998)**:** Swing, Collections
- **J2SE 1.4** (2002)**:** assert keyword, regular expressions, exception chaining, non-blocking IO (NIO)

Module java.base
Package java.util

Class Collections

java.lang.Object
    java.util.Collections

`public class` **`Collections`**
`extends Object`

This class consists exclusively of static methods that operate on or return collections. It contains polymorphic algorithms that operate on collections, "wrappers", which return a new collection backed by a specified collection, and a few other odds and ends.

The methods of this class all throw a `NullPointerException` if the collections or class objects provided to them are null.

The documentation for the polymorphic algorithms contained in this class generally includes a brief description of the *implementation*. Such descriptions should be regarded as *implementation notes*, rather than parts of the *specification*. Implementors should feel free to substitute other algorithms, so long as the specification itself is adhered to. (For example, the algorithm used by `sort` does not have to be a mergesort, but it does have to be *stable*.)

The "destructive" algorithms contained in this class, that is, the algorithms that modify the collection on which they operate, are specified to throw `UnsupportedOperationException` if the collection does not support the appropriate mutation primitive(s), such as the `set` method. These algorithms may, but are not required to, throw this exception if an invocation would have no effect on the collection. For example, invoking the `sort` method on an unmodifiable list that is already sorted may or may not throw `UnsupportedOperationException`.

This class is a member of the Java Collections Framework.

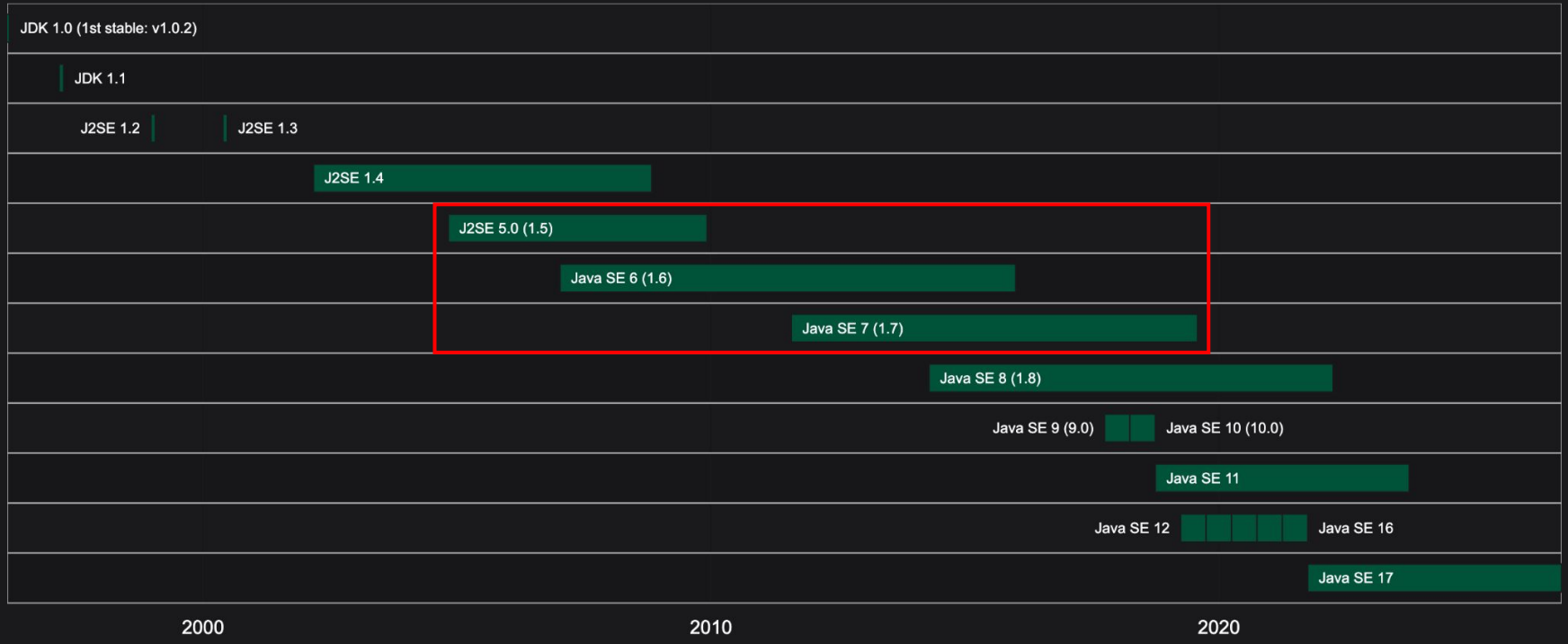
Since:

1.2

See Also:

`Collection`, `Set`, `List`, `Map`

https://www.cs.usfca.edu/~cs272/javadoc/api/java.base/java/util/Collections.html



https://jsfiddle.net/sjengle/6759bwtj/show • https://www.oracle.com/java/technologies/java-se-support-roadmap.html

J2SE 5.0 to Java SE 7 (1.7)

- **J2SE 5.0** (2004)**:** Generics, annotations, autoboxing/unboxing, enumerations, enhanced for each loops, concurrency enhancements
- **Java SE 6** (2006)**:** Performance improvements
- **Java SE 7** (2011)**:** Try-with-resources, catching multiple exception types, new file IO

Module java.base
Package java.lang

Interface Iterable<T>

Type Parameters:
T - the type of elements returned by the iterator

All Known Subinterfaces:
BeanContext, BeanContextServices, BlockingDeque<E>, BlockingQueue<E>, Collection<E>, Deque<E>, DirectoryStream<T>, EventSet, List<E>, NavigableSet<E>, NodeSetData<T>, Path, Queue<E>, SecureDirectoryStream<T>, Set<E>, SortedSet<E>, TransferQueue<E>, XPathNodes

All Known Implementing Classes:
AbstractCollection, AbstractList, AbstractQueue, AbstractSequentialList, AbstractSet, ArrayBlockingQueue, ArrayDeque, ArrayList, AttributeList, BatchUpdateException, BeanContextServicesSupport, BeanContextSupport, ConcurrentHashMap.KeySetView, ConcurrentLinkedDeque, ConcurrentLinkedQueue, ConcurrentSkipListSet, CopyOnWriteArrayList, CopyOnWriteArraySet, DataTruncation, DelayQueue, DocTreePath, EnumSet, HashSet, JobStateReasons, LinkedBlockingDeque, LinkedBlockingQueue, LinkedHashSet, LinkedList, LinkedTransferQueue, PriorityBlockingQueue, PriorityQueue, RoleList, RoleUnresolvedList, RowSetWarning, SerialException, ServiceLoader, SQLClientInfoException, SQLDataException, SQLException, SQLFeatureNotSupportedException, SQLIntegrityConstraintViolationException, SQLInvalidAuthorizationSpecException, SQLNonTransientConnectionException, SQLNonTransientException, SQLRecoverableException, SQLSyntaxErrorException, SQLTimeoutException, SQLTransactionRollbackException, SQLTransientConnectionException, SQLTransientException, SQLWarning, Stack, SyncFactoryException, SynchronousQueue, SyncProviderException, TreePath, TreeSet, Vector

---

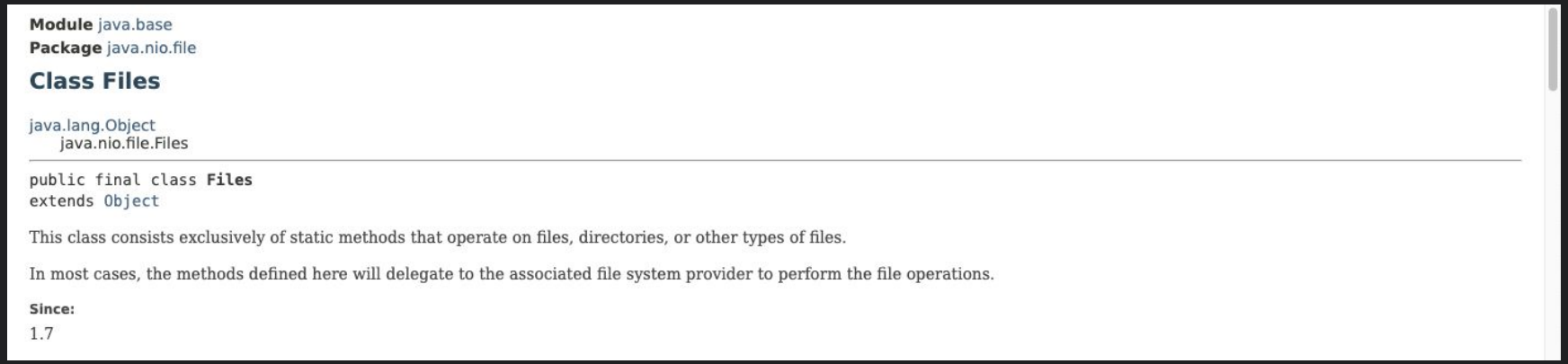
`public interface Iterable<T>`

Implementing this interface allows an object to be the target of the enhanced `for` statement (sometimes called the "for-each loop" statement).

See *Java Language Specification*:
14.14.2 The enhanced for statement

Since:
1.5

https://www.cs.usfca.edu/~cs272/javadoc/api/java.base/java/lang/Iterable.html



Module java.base
Package java.nio.file

Class Files

java.lang.Object
    java.nio.file.Files

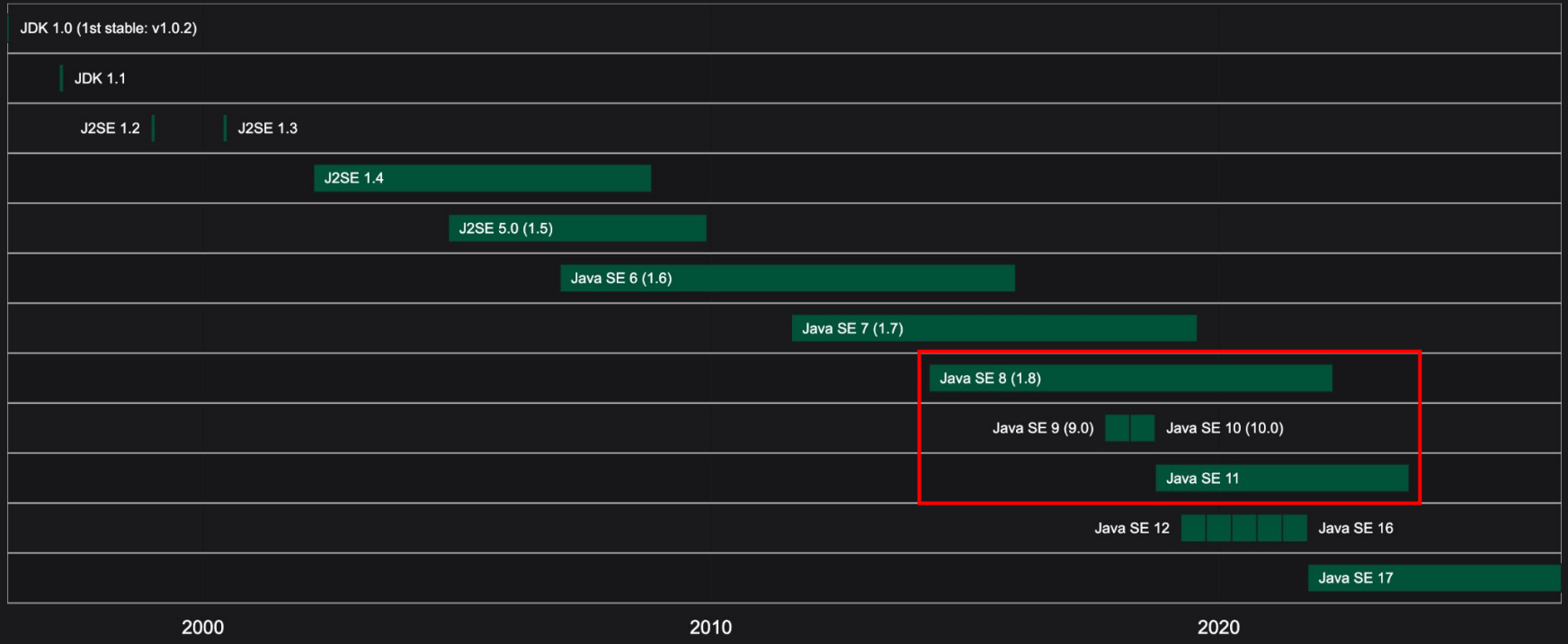
```
public final class Files
extends Object
```

This class consists exclusively of static methods that operate on files, directories, or other types of files.

In most cases, the methods defined here will delegate to the associated file system provider to perform the file operations.

Since:
1.7

https://www.cs.usfca.edu/~cs272/javadoc/api/java.base/java/nio/file/Files.html



https://jsfiddle.net/sjengle/6759bwtj/show • https://www.oracle.com/java/technologies/java-se-support-roadmap.html

Java SE 8 to 11

- **Java SE 8** (2014)**:** Lambda expressions, streams
- **Java SE 9** (2017)**:** Modularization, jshell
- **Java SE 10** (2018)**:** var keyword
- **Java SE 11** (2018)**:** Added java.net.http

Module java.base
Package java.lang

Annotation Type FunctionalInterface

```
@Documented
@Retention(RUNTIME)
@Target(TYPE)
public @interface FunctionalInterface
```

An informative annotation type used to indicate that an interface type declaration is intended to be a *functional interface* as defined by the Java Language Specification. Conceptually, a functional interface has exactly one abstract method. Since default methods have an implementation, they are not abstract. If an interface declares an abstract method overriding one of the public methods of `java.lang.Object`, that also does *not* count toward the interface's abstract method count since any implementation of the interface will have an implementation from `java.lang.Object` or elsewhere.

Note that instances of functional interfaces can be created with lambda expressions, method references, or constructor references.

If a type is annotated with this annotation type, compilers are required to generate an error message unless:

- The type is an interface type and not an annotation type, enum, or class.
- The annotated type satisfies the requirements of a functional interface.

However, the compiler will treat any interface meeting the definition of a functional interface as a functional interface regardless of whether or not a `FunctionalInterface` annotation is present on the interface declaration.

See *Java Language Specification*:

4.3.2 The Class Object
9.8 Functional Interfaces
9.4.3 Interface Method Body
9.6.4.9 @FunctionalInterface

Since:

1.8

https://www.cs.usfca.edu/~cs272/javadoc/api/java.base/java/lang/FunctionalInterface.html

14.4.1. Local Variable Declarators and Types

Each *declarator* in a local variable declaration declares one local variable, whose name is the *Identifier* that appears in the declarator.

If the optional keyword `final` appears at the start of the declaration, the variable being declared is a final variable (§4.12.4).

The declared type of a local variable is determined as follows:

- If *LocalVariableType* is an *UnannType*, and no bracket pairs appear in *UnannType* or *VariableDeclaratorId*, then *UnannType* denotes the type of the local variable.
- If *LocalVariableType* is an *UnannType*, and bracket pairs appear in *UnannType* or *VariableDeclaratorId*, then the type of the local variable is specified by §10.2.
- If *LocalVariableType* is `var`, then let T be the type of the initializer expression when treated as if it did not appear in an assignment context, and were thus a standalone expression (§15.2). The type of the local variable is the upward projection of T with respect to all synthetic type variables mentioned by T (§4.10.5).

  It is a compile-time error if T is the null type.

  Because the initializer is treated as if it did not appear in an assignment context, an error occurs if it is a lambda expression (§15.27) or a method reference expression (§15.13).

Example 14.4.1-1. Type of Local Variables Declared With `var`

The following code illustrates the typing of variables declared with `var`:

```
var a = 1;                // a has type 'int'
var b = java.util.List.of(1, 2);  // b has type 'List<Integer>'
var c = "x".getClass();   // c has type 'Class<? extends String>'
                          // (see JLS 15.12.2.6)
var d = new Object() {};  // d has the type of the anonymous class
var e = (CharSequence & Comparable<String>) "x";
                          // e has type CharSequence & Comparable<String>
var f = () -> "hello";    // Illegal: lambda not in an assignment context
var g = null;             // Illegal: null type
```

Note that some variables declared with `var` cannot be declared with an explicit type, because the type of the variable is not denotable.

https://docs.oracle.com/javase/specs/jls/se17/html/jls-14.html#jls-14.4

https://jsfiddle.net/sjengle/6759bwtj/show • https://www.oracle.com/java/technologies/java-se-support-roadmap.html

Java SE 12 to 14

- **Java SE 12** (2019)**:** Compact number formatting, previews
- **Java SE 13** (2019)**:** Reimplemented Socket API, new FileSystems methods
- **Java SE 14** (2020)**:** Added switch expressions, more helpful NullPointerException output

https://www.oracle.com/java/technologies/javase/14-relnote-issues.html#NewFeature

Java SE 15 to 17

- **Java SE 15** (2020)**:** Text blocks (finally!), hidden classes, CharSequence.isEmpty method, performance improvements to TreeMap compute/merge methods
- **Java SE 16** (2021)**:** Stream.toList method, record classes
- **Java SE 17** (2021)**:** Sealed classes, javadoc updates

https://docs.oracle.com/en/java/javase/17/language/java-language-changes.html

Switch Expressions

```
int numLetters = 0;
Day day = Day.WEDNESDAY;
switch (day) {
  case MONDAY, FRIDAY, SUNDAY → numLetters = 6;
  case TUESDAY                → numLetters = 7;
  case THURSDAY, SATURDAY     → numLetters = 8;
  case WEDNESDAY              → numLetters = 9;
  default → throw new Exception("Invalid day: " + day);
};
System.out.println(numLetters);
```

https://docs.oracle.com/en/java/javase/17/language/switch-expressions.html

Text Blocks

```
// ORIGINAL
String message = "'The time has come,' the Walrus said,\n" +
                 "'To talk of many things:\n" +
                 "Of shoes -- and ships -- and sealing-wax --\n" +
                 "Of cabbages -- and kings --\n" +
                 "And why the sea is boiling hot --\n" +
                 "And whether pigs have wings.'\n";
```

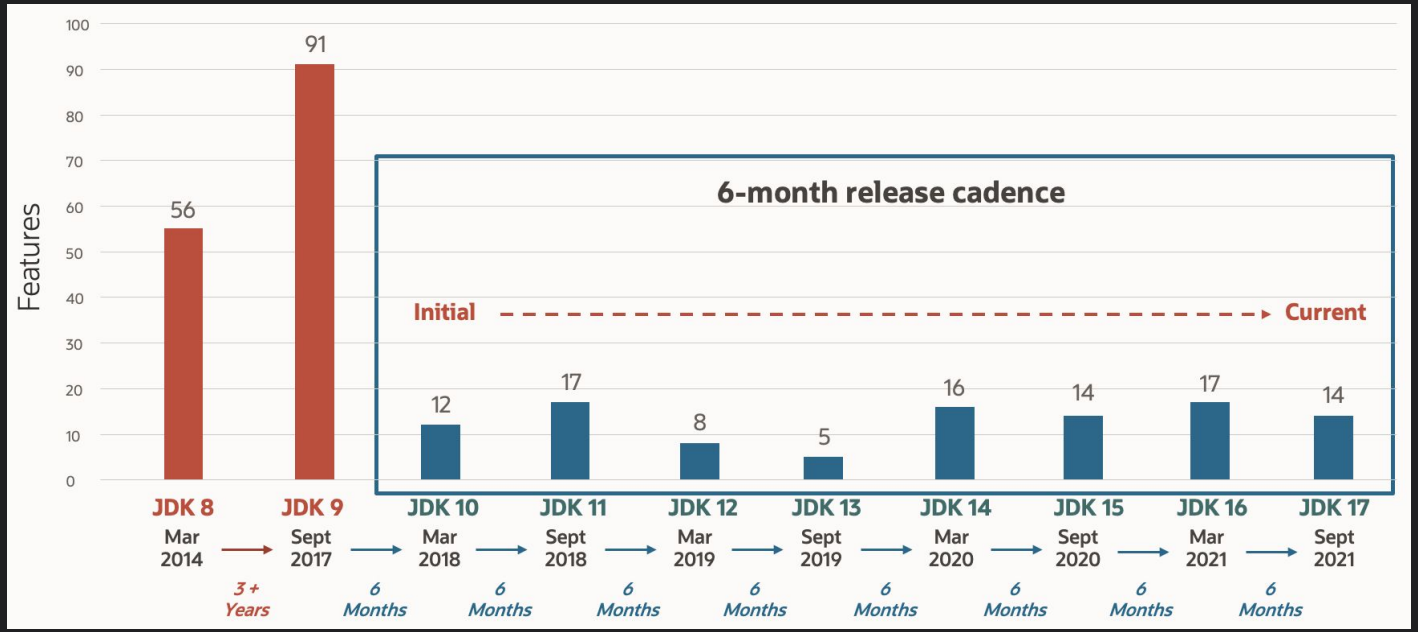
https://docs.oracle.com/en/java/javase/17/text-blocks/index.html

Text Blocks

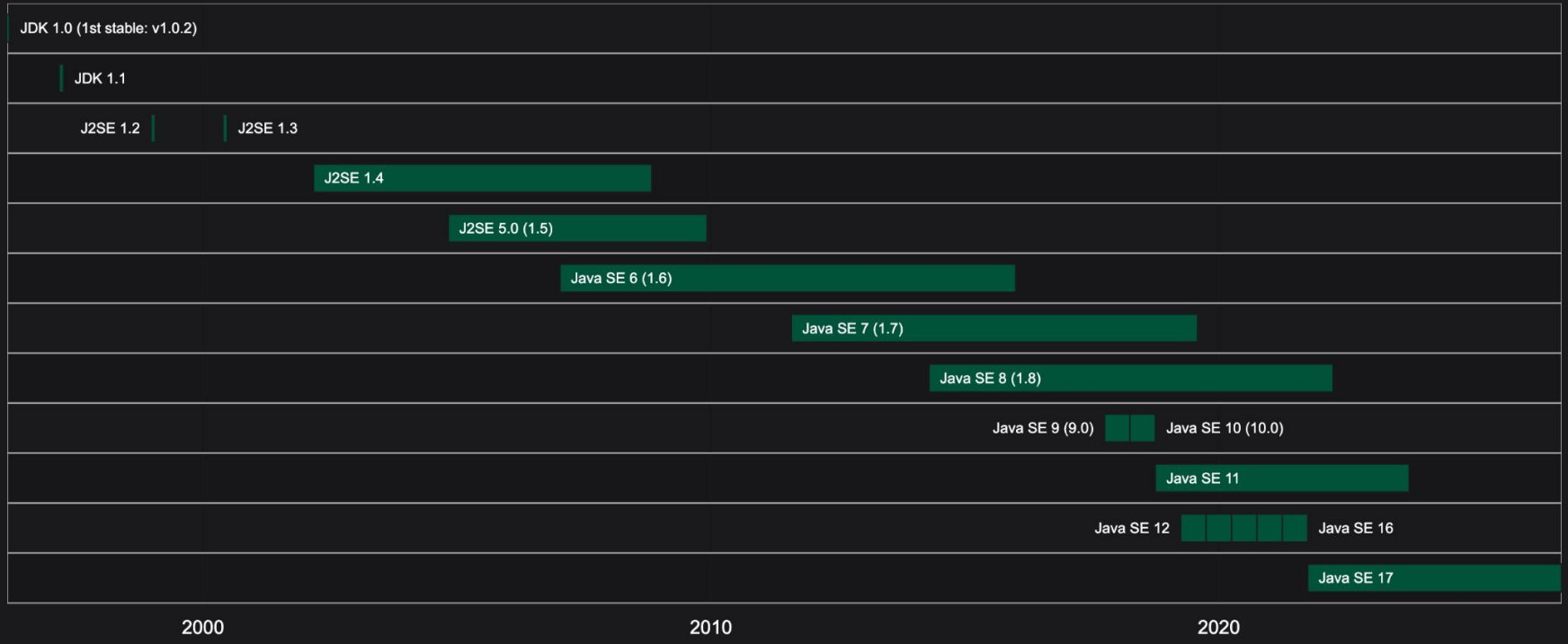
```
// BETTER
String message = """
    'The time has come,' the Walrus said,
    'To talk of many things:
    Of shoes -- and ships -- and sealing-wax --
    Of cabbages -- and kings --
    And why the sea is boiling hot --
    And whether pigs have wings.'
    """;
```

https://docs.oracle.com/en/java/javase/17/text-blocks/index.html

Features in JDK 8 to 17



https://blogs.oracle.com/java/post/announcing-java17



https://jsfiddle.net/sjengle/6759bwtj/show • https://www.oracle.com/java/technologies/java-se-support-roadmap.html

Release	General Availability	Premier Support Until	Extended Support Until	Sustaining Support
7 (LTS)	July 2011	July 2019	July 2022*****	Indefinite
8 (LTS)**	March 2014	March 2022	December 2030*****	Indefinite
9 (non-LTS)	September 2017	March 2018	Not Available	Indefinite
10 (non-LTS)	March 2018	September 2018	Not Available	Indefinite
11 (LTS)	September 2018	September 2023	September 2026	Indefinite
12 (non-LTS)	March 2019	September 2019	Not Available	Indefinite
13 (non-LTS)	September 2019	March 2020	Not Available	Indefinite
14 (non-LTS)	March 2020	September 2020	Not Available	Indefinite
15 (non-LTS)	September 2020	March 2021	Not Available	Indefinite
16 (non-LTS)	March 2021	September 2021	Not Available	Indefinite
17 (LTS)	September 2021	September 2026****	September 2029****	Indefinite
18 (non-LTS)	March 2022	September 2022	Not Available	Indefinite
19 (non-LTS)***	September 2022	March 2023	Not Available	Indefinite
20 (non-LTS)***	March 2023	September 2023	Not Available	Indefinite
21 (LTS)***	September 2023	September 2028	September 2031	Indefinite

https://www.oracle.com/java/technologies/java-se-support-roadmap.html

Java 18 New Features

- UTF-8 is now the default character set for Java
- Added a simple web server
- Added **`@snippet`** for code in Javadoc
- Preview of pattern matching for **`switch`** cases
- Incubator preview of vector computations

https://www.oracle.com/java/technologies/javase/18-relnote-issues.html#NewFeature

Java 19 New Features

- Previews of pattern matching in Record and switch
- Preview of virtual threads
- Incubator preview of structured concurrency
- Incubator preview of vector computations
- Support for Unicode 14 (more emoji!)

https://www.oracle.com/java/technologies/javase/19-relnote-issues.html#NewFeature

Resources

Wikipedia

https://en.wikipedia.org/wiki/Java_version_history

Oracle

https://www.oracle.com/technetwork/java/java-se-support-roadmap.html
https://docs.oracle.com/en/java/javase/17/language/java-language-changes.htm

See also "What's New in Java ..." articles for every major release.

Questions?