Unnamed Classes and Instance `main` Methods (Preview)

Changes to the Java® Virtual Machine Specification • Version 21-internal-adhoc.gbierman.20230515

This document describes changes to the Java Virtual Machine Specification ⇗ to support *Unnamed Classes and Instance `main` Methods*, which is a preview feature of Java SE 21. See JEP 445 ⇗ for an overview of the feature.

A companion document describes the changes needed to the Java Language Specification ⇗ to support Unnamed Classes and Instance `main` Methods.

Changes are described with respect to existing sections of the JLS. New text is indicated like this and deleted text is indicated ~~like this~~. Explanation and discussion, as needed, is set aside in grey boxes.

> *Changelog:*
>
> *2023-05-15: First draft released*

Chapter 5: Loading, Linking, and Initializing

5.2 Java Virtual Machine Startup

The Java Virtual Machine starts up by creating an initial class or interface using the bootstrap class loader (5.3.1 ⇗) or a user-defined class loader (5.3.2 ⇗). The Java Virtual Machine then links the initial class or interface, initializes it, and invokes ~~the `public static` method `void main(String[])`~~ a `main` method, as described in (JLS 12.1.4 ⇗). The invocation of this method drives all further execution. Execution of the Java Virtual Machine instructions constituting the `main` method may cause linking (and consequently creation) of additional classes and interfaces, as well as invocation of additional methods.

The initial class or interface is specified in an implementation-dependent manner. For example, the initial class or interface could be provided as a command line argument. Alternatively, the implementation of the Java Virtual Machine could itself provide an initial class that sets up a class loader which in turn loads an application. Other choices of the initial class or interface are possible so long as they are consistent with the specification given in the previous paragraph.

In contrast, the invocation of a `main` method of the initial class or interface is not specified in an implementation-dependent manner but, rather, proceeds according to the implementation-

independent rules given in (JLS 12.1.4 ⇗).

DRAFT 21-internal-adhoc.gbierman.20230515