JEP 306: Restore Always-Strict Floating-Point Semantics

Owner	Joe Darcy
Type	Feature
Scope	SE
Status	Closed / Delivered
Release	17
Component	specification / language
Discussion	hotspot dash dev at openjdk dot java dot net, core dash libs dash dev at openjdk dot java dot net
Effort	M
Duration	S
Reviewed by	Brian Goetz, John Rose, Mikael Vidstedt
Endorsed by	Mikael Vidstedt
Created	2017/02/27 15:51
Updated	2021/08/02 03:26
Issue	8175916

Summary

Make floating-point operations consistently strict, rather than have both strict floating-point semantics (`strictfp`) and subtly different default floating-point semantics. This will restore the original floating-point semantics to the language and VM, matching the semantics before the introduction of strict and default floating-point modes in Java SE 1.2.

Goals

- Ease development of numerically-sensitive libraries, including `java.lang.Math` and `java.lang.StrictMath`.
- Provide more regularity in a tricky aspect of the platform.

Non-Goals

It is not a goal to define any sort of "fast-fp" or "loose-fp" (c.f. JSR 84: Floating Point Extensions).

Motivation

The impetus for changing the default floating-point semantics of the platform in the late 1990's stemmed from a bad interaction between the original Java language and JVM semantics and some unfortunate peculiarities of the x87 floating-point co-processor instruction set of the popular x86 architecture. Matching the exact floating-point semantics in all cases, including subnormal operands and results, required large overheads of additional instructions. Matching the results in the absence of overflow or underflow could be achieved with much less overhead and that is roughly what is allowed by the revised default floating-point semantics introduced in Java SE 1.2.

However, the SSE2 (Streaming SIMD Extensions 2) extensions, shipped in Pentium 4 and later processors starting circa 2001, could support strict JVM floating-point operations in a straightforward manner without undue overhead.

Since Intel and AMD have both long supported SSE2 and later extensions which allow natural support for strict floating-point semantics, the technical motivation for having a default floating-point semantics different than strict is no longer present.

Description

The interfaces this JEP would modify include the Java Language Specification in its coverage of floating-point expressions (see JLS sections 4.2.3 *Floating-Point Types, Formats, and Values*, 5.1.13 *Value Set Conversion, 15.4 *FP-strict Expressions*, many small updates to other sections later in chapter 15) and similar sections of the Java Virtual Machine Specification (JVMS 2.3.2 *Floating-Point Types, Values Sets, and Values*, section 2.8.2 *Floating-Point Modes*, 2.8.3 *Value Set Conversion*, and many small updates to individual floating-point instructions). The concepts of value sets and value set conversion would be removed from the JLS and JVMS. Implementation changes in the JDK would include updating the HotSpot virtual machine to never run in a floating-point mode which allowed the extended exponent value set (such a mode is mandated to be present for `strictfp` operations) and updating `javac` to issue new lint warnings to unnecessary use of the `strictfp` modifier.

Testing

All existing numerical tests would remain valid with this change since this change always uses one of the currently allowable modes of operation.

Existing Java source tests using `strictfp` could be replicated without the modifier.

Risks and Assumptions

The proposed specification changes are low-risk, mostly deleting text and updating the floating-point overviews in JLS and JVMS. The platform's original semantics are

restored and strict-only was always an allowable implementation option. Therefore, there is negligible behavior compatibility risk from the changes in this JEP. Even on platforms where non-strict execution could occur, there was no idiom to force non-strict evaluation. To be robust, code always has to be able to work if executed strictly, which will again be the only option.