L-World Value Types

Welcome to the L-World Value Types early adopter's project !

What are Value Types?

- Value Types are small, immutable, identity-less types
- User model: "codes like a class, works like an int"
- Use cases: Numerics, algebraic data types, tuples, cursors, ...
- Removing identity commitment enables optimizations such as
    - flattening of value types in containers such as fields or arrays
        - reducing cost of indirection and locality of reference with attendant cache miss penalties
        - reducing memory footprint and load on garbage collectors
- Combining Immutability and no identity commitment allows value types to be stored in registers or stack or passed by value

What is the L-World Value Types project?

The L-World Value Types project is a series of early prototypes for Value Types.

- builds on work of the previous Minimal Values Types prototype (MVT)
    - provides a new type which is: immutable, identity-agnostic, non-nullable, non-synchronizable, final
    - Value Types contained in References, other Value Types or in Arrays are flatten-able
    - Value Types can contain primitives or references
    - JVMS class file model for MVT:
        - Separate descriptors to distinguish value types from object types using "Q" signatures (Q-Types) similar to how Object descriptors begin with "L" (L-Types).
        - Separate bytecodes, starting with "v", called "v-bytecodes" to distinguish from reference "a-bytecodes".
- MVT limitations:
    - No direct support for methods
    - No compatibility with existing Objects and Interfaces
- Enabled exploration of potential maximum optimizations for value types with minimal impact on existing objects (object identity types)

"L-World"

To maximize backward compatibility with existing Objects and interfaces, i.e. existing L-Types, the current prototype incorporates value types into the L-Type system or "L-World".

- Value Types may be referred to by the same "L-Type" descriptors the VM has always operated on:
    - May implement interfaces with value types
    - May pass a value type as a java.lang.Object, or an interface through existing APIs
- Value Type characteristics:
    - immutable: unmodifiable instance fields
        - may contain primitives, other value types, references to mutable objects
    - identity-less:
        - synchronization including use of wait(*), notify*() will fail with exception: IllegalMonitorStateException
        - reference equality with "==" (`if_acmp<eq|ne>`) always returns false, value equality requires using the "equals()" method
        - freely substitutable when equal, no visible change in behavior if equals()
    - final
- JVMS class file model for "LWorld"

        - Re-uses "L" descriptors
        - Re-uses "a-bytecodes"
        - There are only two new byte-codes, otherwise existing byte-codes have been engineered to accept and maintain value type characteristics (identity-less, flattenable, pass by value):
            - "default" - will create a new default value
            - "withfield"- allows updating value type fields via a copy-on-write semantic, i.e. new value based on the old value combined with new field value.

Recognize that the path to Valhalla is long, there are number of open issues facing value types. We wish to solve these incrementally. Fully generic specialization of value types with clear and sensible migration rules are going to take more than a single prototype.

"LW1" - Minimal L-World

LW1 initial prototype is intended to get this into your hands as early as possible so you can provide us with feedback.

Javac source support:

- Requires source level >= JDK11
- A class declaration may be tagged as being a value type by using the "__ByValue" modifier
    - Interfaces, annotation types, enums can not be value types
    - Top level, inner, nested, local classes may be value types. Value types may declare inner, nested, local types
    - Value Types are implicitly final, so cannot be abstract)
- Value Types may not declare an explicit super class. They implicitly extend java.lang.Object akin to enums, annotation types and interfaces
- Value Types may declare explicit interfaces
- Value Types constructors may not pass the "this" handle until all instance fields are definitely assigned.
- All instance fields of a value class are implicitly final
- Value types may not to declare fields of its own type directly or indirectly
- java.lang.Object methods:
    - javac automatically generates hashCode, equals, longHashCode and toString computed solely from the instance's state and not from its identity
    - javac does not clone(), finalize(), wait*), notify*() on value receivers
- javac does not allow comparison of values using ==, !=
- Value Types can not be assigned null, null can not be cast to or compared with value types
- Value Types cannot be type arguments in generic type parameterizations, type witnesses in generic method invocations, wildcard bounds
- Type migration, including partial recompilation is not supported

Java APIs:

- Support for java.lang.Object methods:
    - TODO
- Class.isValue() new API
- Class.newInstance() on a value type throws IllegalAccessException
- SetAccessible() on a value type throws InaccessibleObjectException

Runtime Exceptions:

- Generated class files require ValueTypes attribute, and will throw IncompatibleClassChangeError if value types are used which are not in the attribute
- IllegalMonitorException thrown on attempts to synchronize or call wait(*) or notify*() on a value type
- ClassCircularityError thrown if loading an instance field of a value type which declares its own type either directly or indirectly

Limitations for LW1

- platforms: x64 Linux, x64 Mac OS X, Windows
- no support for value class with no instance fields
- no support for atomic fields containing value types
- no support for @Contended value type fields
- static fields are not flattenable
- no AOT, CDS, ZGC, serviceability agent
- -Xint and C2 only, no C1, no tiered-compilation, no Graal
- unsafe field and array accessor APIs are not supported for value types
    - VarHandles support value types
        - however: they do not enforce immutability or non-nullability for value types
            - note that immutable types can be copied, so updates may not be seen
    - low-level unsafe APIs are UNSAFE and will not be changed to support value types
    - risks: If a value type has been flattened in a container, unsafe does not know the layout
        - getObject could return the first flattened element rather than the expected reference
- interpreter is not optimized, focus is on JIT optimization
- Please provide additional USE CASES for optimizations
- Working on additional test cases

Future Possibilities

- LW1 updates:
  - fix bugs, add optimizations and support for additional minor features
- LWX:
  - Addition of significant feature support
  - Addition of language level syntax and semantics
  - Eventually a preview with an openjdk release
  - Expect specialized generics and support for primitives as value types will have their own early access cycles

How to Try L-World Value Types

Target Audience

- Power users - Java/JVM Language, Framework, Library authors/experts
  - who are comfortable with early experimental software
  - who recognize that everything in the experiment - the model, the classfile extensions, the byte codes is likely to change
  - who want to contribute to early exploration of Value Types
  - who will not build any products based on these prototypes
- Who are willing to provide feedback to the developers on a subset of Value Type features
- Who will provide use cases for the development team to experiment with optimizations

Early Access Binaries

http://jdk.java.net/valhalla/

Repository and Build Instructions

To create a new local repository, based on "lworld" branch:

```
hg clone http://hg.openjdk.java.net/valhalla/valhalla valhalla-lworld

cd valhalla-lworld

hg defpath du <openjdkname>

hg update -r lw1 // name of branch
```

To update repository:

```
cd valhalla-lworld

hg pull

hg update -r lw1  // name of branch
```

To build repository

```
bash configure

make images
```

Instructions for working with branch repositories: http://cr.openjdk.java.net/~chegar/docs/sandbox.html

Note: Valhalla is a child of the jdk/jdk repository, to keep current with latest OpenJDK development.

Programming Model

CMH: Example code

Sample JMH benchmarks: http://mail.openjdk.java.net/pipermail/valhalla-dev/2018-June/004380.html

Run Experimental L-World

- **java -XX:+EnableValhalla <Test>**
- Further experimental flags can be found here

Helpful Feedback Please

This is intended as an early prototype to give you a chance to experiment and provide feedback on the currently supported features.

Please ensure you are on the latest EA binaries before reporting a problem.

Bugs are tracked in JIRA: Start summary with /[lworld/]. You can search for known problems or already reported bugs.

Send email to valhalla-dev@openjdk.java.net

- Use cases that worked for you, including any positive performance or footprint information
- Bugs or questions about surprise behavior
- Use cases that you would like to see optimized PLEASE

Too Early for Feedback

This is intended as an early prototype to give you a chance to experiment and provide feedback on the currently supported features.

At this time it would premature to provide feedback on

- Language syntax
- Problems already listed under Limitations

References

- Draft Java Virtual Machine Specification Value Type changes
- http://cr.openjdk.java.net/~chegar/docs/sandbox.html (instructions for working with branch repositories)