The sinuous path to Valhalla

Rémi Forax

Me, Myself and I

Assistant Prof at Paris East University

Open Source dev

- OpenJDK, ASM, Tatoo, Pro, etc

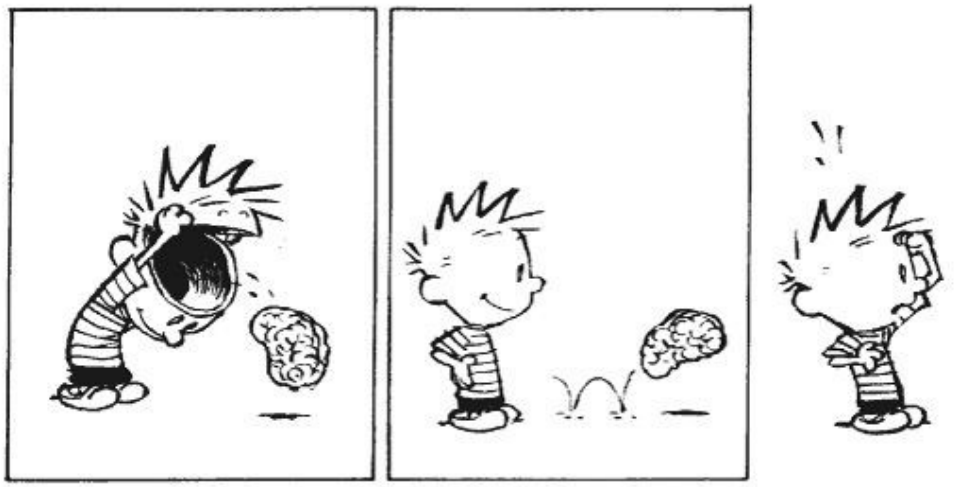
Java Spec expert

- One of the Father of invokedynamic (Java 7)
- lambda (Java 8), module (Java 9), constant dynamic (Java 11)
- record (Java 14) and valhalla (Java ??)

I’m not an Oracle Employee !

Java is open source

since a long time now



Don’t believe what I’m saying !

The sinuous path to Valhalla

Rémi Forax

Intro

At Java inception

Hardware in 1993

- Intel 486 SX 25 Mhz / 8 MB of RAM
- Pointer dereference ~~ one addition

Java Model

- Every objects should be hold by references

Nowadays

Hardware

- Intel Kaby Lake 2.4 Ghz / 16 GB of RAM
- Pointer dereference ~~ 300 additions

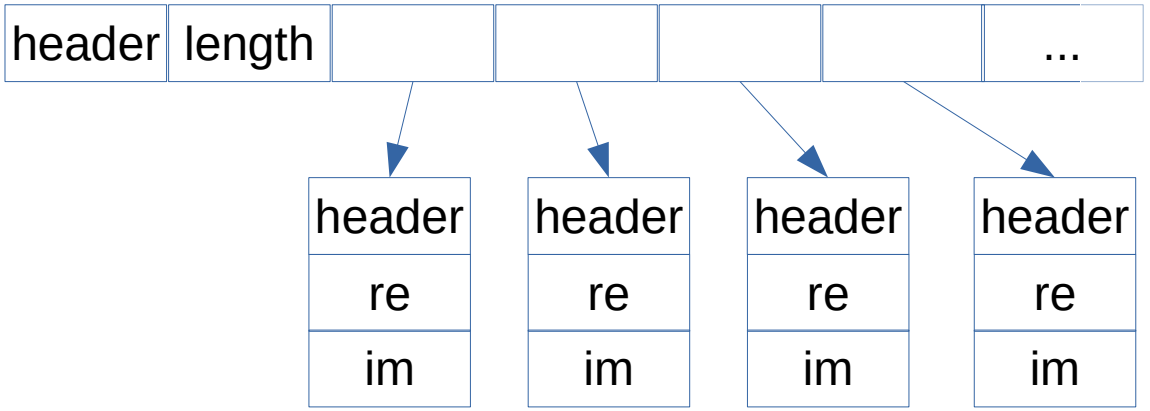
Java Model

- Every objects are still hold by references
  - Unless the JIT can prove that the identity is not used

    Escape Analysis (if …)

On heap representation

An array of class Complex



Inline object (first draft)

An inline object

- no identity (no address)
- not nullable (no pointer)

Runtime model

- inlined into another object/an array cells on heap
- scalarization into registers on stack

On heap representation

An array of an inline class Complex

no pointer
no header at runtime

C struct ?

```
typedef struct _complex {
  double re;
  double im;
} complex;
```

in C

- An array of struct is not an array of pointers
- Stack allocation can be scalarize
  - depends on inlining

Aliasing issue in C/C++

Same address in memory ??

```
void foo(complex& c1, complex& c2) {
  c1.re = 5;
  // is c2.re changed ?
}
```

Solution: make it immutable !

Inline object

An inline object

- no identity (no address)
- not nullable (no pointer)
- **not mutable**

Runtime model

- scalarization into registers on stack
- inlined into another object/an array cells on heap

Inlineable

What if ?

- Inline class or array too big ?
- Concurrent read/write of an inline object

Solution: Inline objects are ***inlineable***

- VM decides if it is inlined or not
  - volatile/static fields are never inlined

Examples

Inline class

Code like a class, works like an int

```
public inline class Complex {
  private final double re;
  private final double im;

  public Complex(double re, double im) {
    this.re = re;
    this.im = im;
  }

  public Complex add(Complex c) {
    return new Complex(re + c.re, im + c.im);
  }
}
```

Inline class

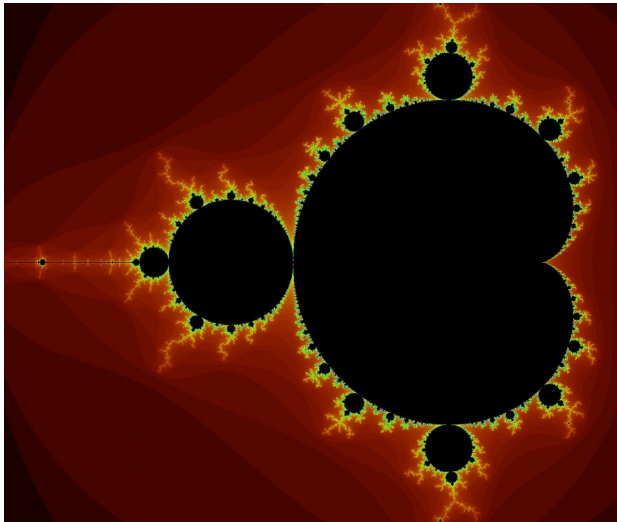
a class with

- keyword “inline”
- no superclass but interfaces
- all fields are final
  - Can not create a cycle !
- any methods

Mandelbrot

Complex provides a useful abstraction

```
for (var col = -32; col < +32; col++) {
  for (var row = -32; row < +32; row++) {
    var c = new Complex(row, col);
    var zn = new Complex(0.0, 0.0);
    var iteration = 0;
    while (zn.modulus() < 4 && iteration < max) {
       zn = zn.square().add(c);
       iteration++;
    }
    if (iteration < max) {
       image.setRGB(col, row, colors[iteration]);
    } else {
       image.setRGB(col, row, black);
    }
  }
}
```



Benchmark with JMH

Time to render the Mandelbrot 64x64 with 1000 iterations?

	score	**error**
primitive	1540 µs/op	22 µs/op
indirect	2816 µs/op	48 µs/op
inline	1541 µs/op	12 µs/op

With java `-XX:+EnableValhalla`

Example: IntBox

java.lang.Integer written as an inline class

```
public inline class IntBox {
  private final int value;

  public IntBox(int value) {
    this.value = value;
  }

  public int intValue() {
    return value;
  }
  public static IntBox zero() {
    return new IntBox(0);
  }
}
```

Array of inline objects

Sum of an array of IntBox

```
private static final IntBox[] ARRAY = new IntBox[100_000];
static {
  range(0, ARRAY.length).forEach(i -> ARRAY[i] = new IntBox(i));
}

@Benchmark
public int sum_IntBox() {
  var sum = 0;
  for(var value : ARRAY) {
    sum += value.intValue();
  }
  return sum;
}
```

Benchmark with JMH

Time to sum an array of 100 000 values

	score	**error**
primitive (int)	27.1 µs/op	0.2 µs/op
indirect (j.l.Integer)	60.7 µs/op	0.6 µs/op
inline (IntBox)	27.0 µs/op	0.1 µs/op

With java `-XX:+EnableValhalla`

Array of inline objects

What if we shuffle the array ?

```
private static final IntBox[] ARRAY = new IntBox[100_000];
static {
  range(0, ARRAY.length).forEach(i -> ARRAY[i] = new IntBox(i));
  Collections.shuffle(Arrays.asList(ARRAY));
}

@Benchmark
public int sum_IntBox() {
  var sum = 0;
  for(var value : ARRAY) {
    sum += value.intValue();
  }
  return sum;
}
```

Benchmark with JMH

Time to sum an array of 100 000 shuffled values

	score	**error**
primitive (int)	27.1 µs/op	0.2 µs/op
indirect (j.l.Integer)	**121.9 µs/op**	5.1 µs/op
inline (IntBox)	27.0 µs/op	0.1 µs/op

With java `-XX:+EnableValhalla`

OpenJDK projects dependency a.k.a more use cases

OpenJDK projects dependency

Panama

- Vector API (SIMD, AVX support)
  - Long128, Long256, etc
- Native Access API (safe/managed malloc)
  - Pointer, Slice, etc

Amber

- Pattern matching with de-construction
  - Carrier object

Roadmap

Current Roadmap

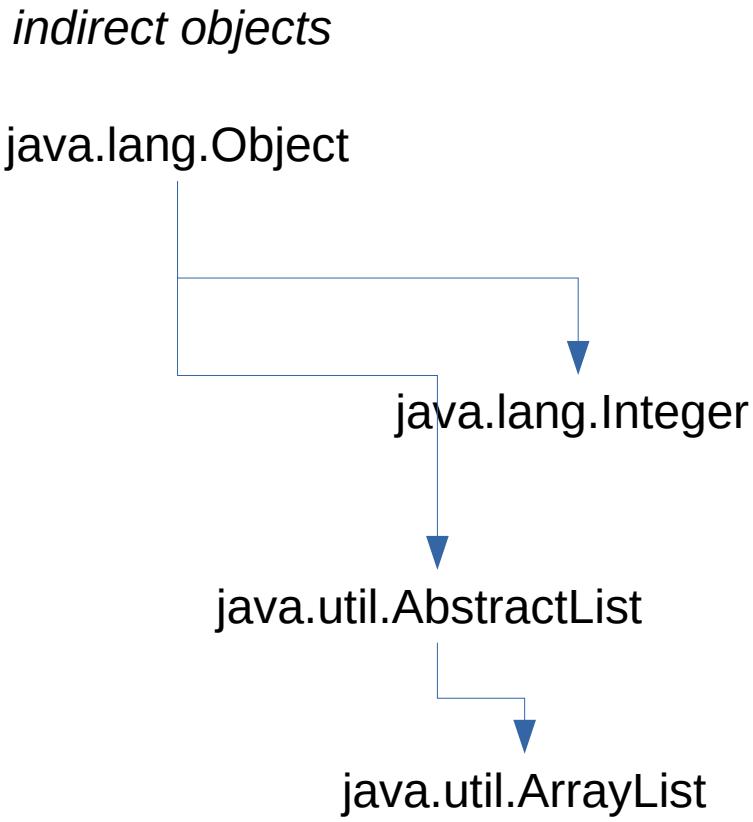
Early access prototypes (http://jdk.java.net/)

- MVT
- Lworld 1, 2

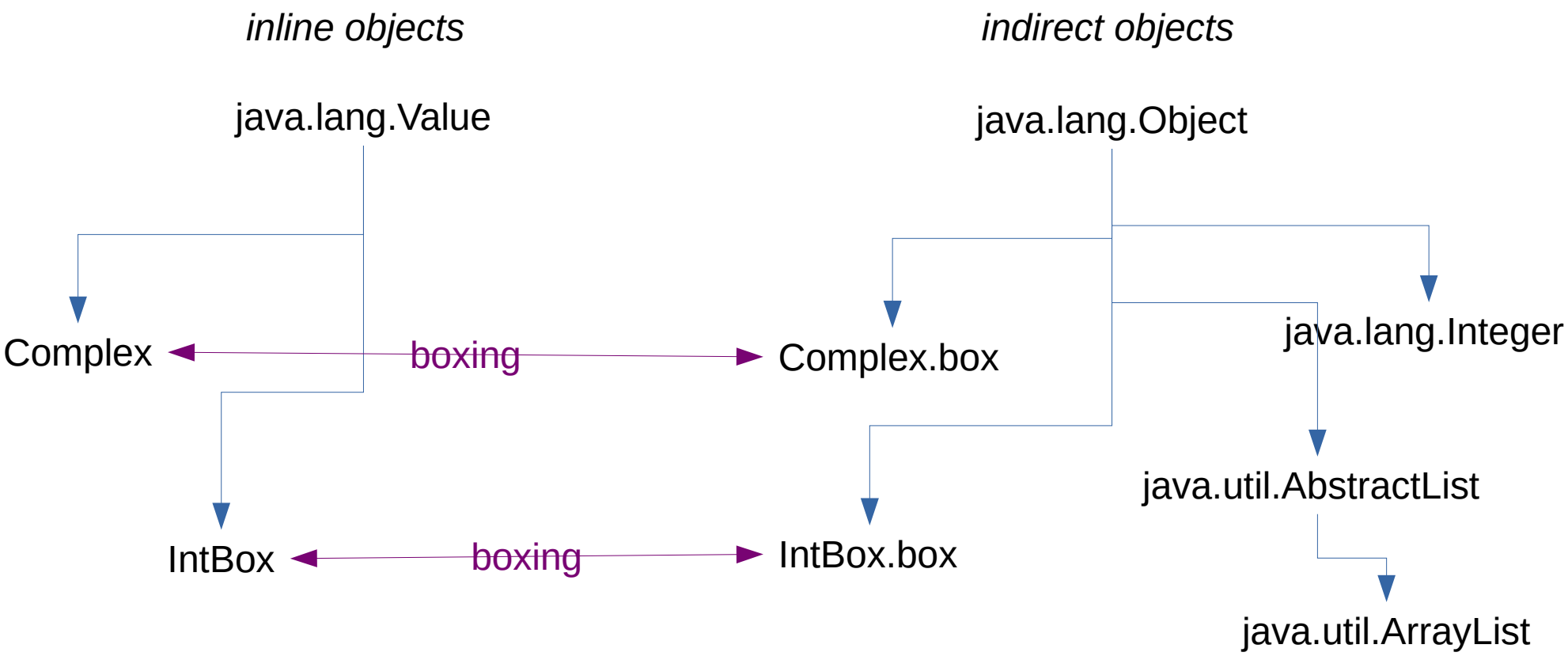
  => We are here now !
- Lword 10 - **preview release** in Java
  - Support erased generics
- Lworld 100 – finale release
  - specialized generics

Minimum Value Type
(first prototype)

j.l.Value vs j.l.Object



j.l.Value vs j.l.Object



MVT

Good perf with inline classes only code !

No perf regression in existing code !

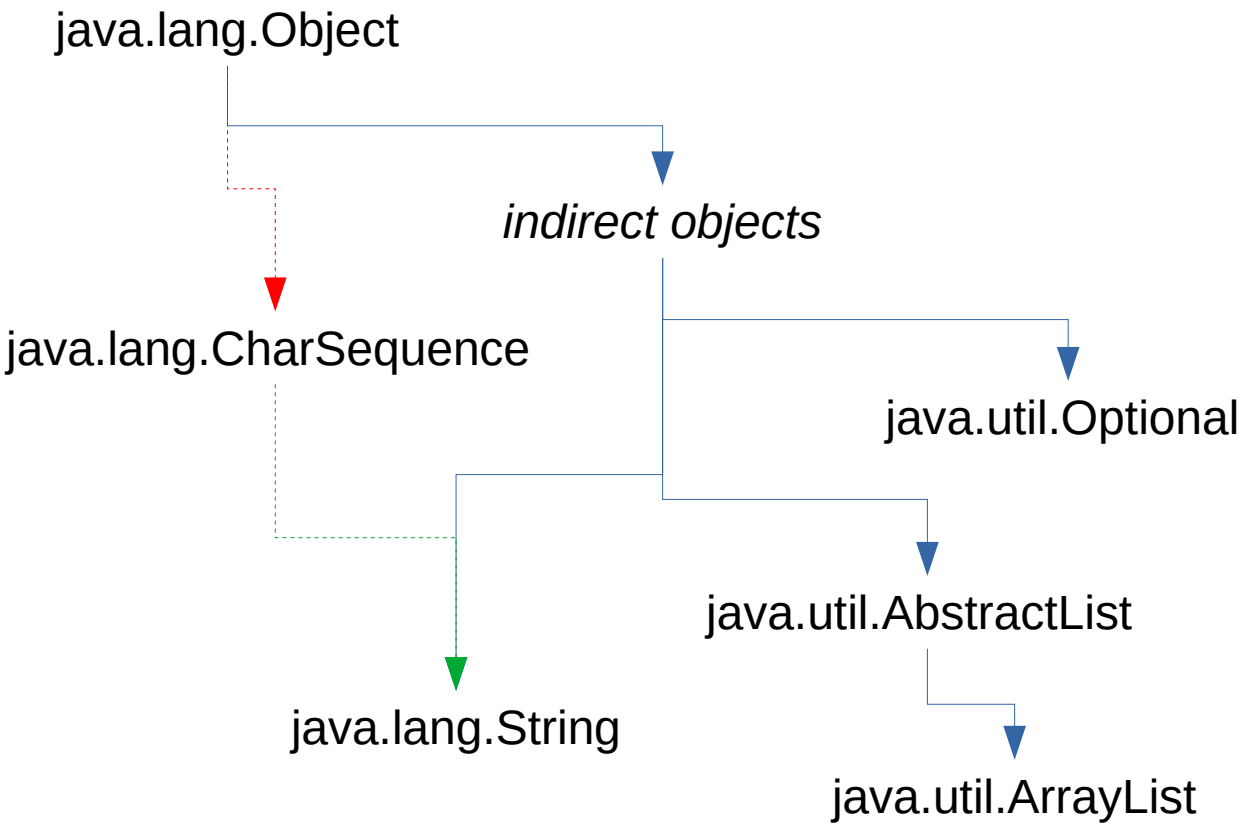
Doesn’t work well with real applications

Requires **too much boxing**

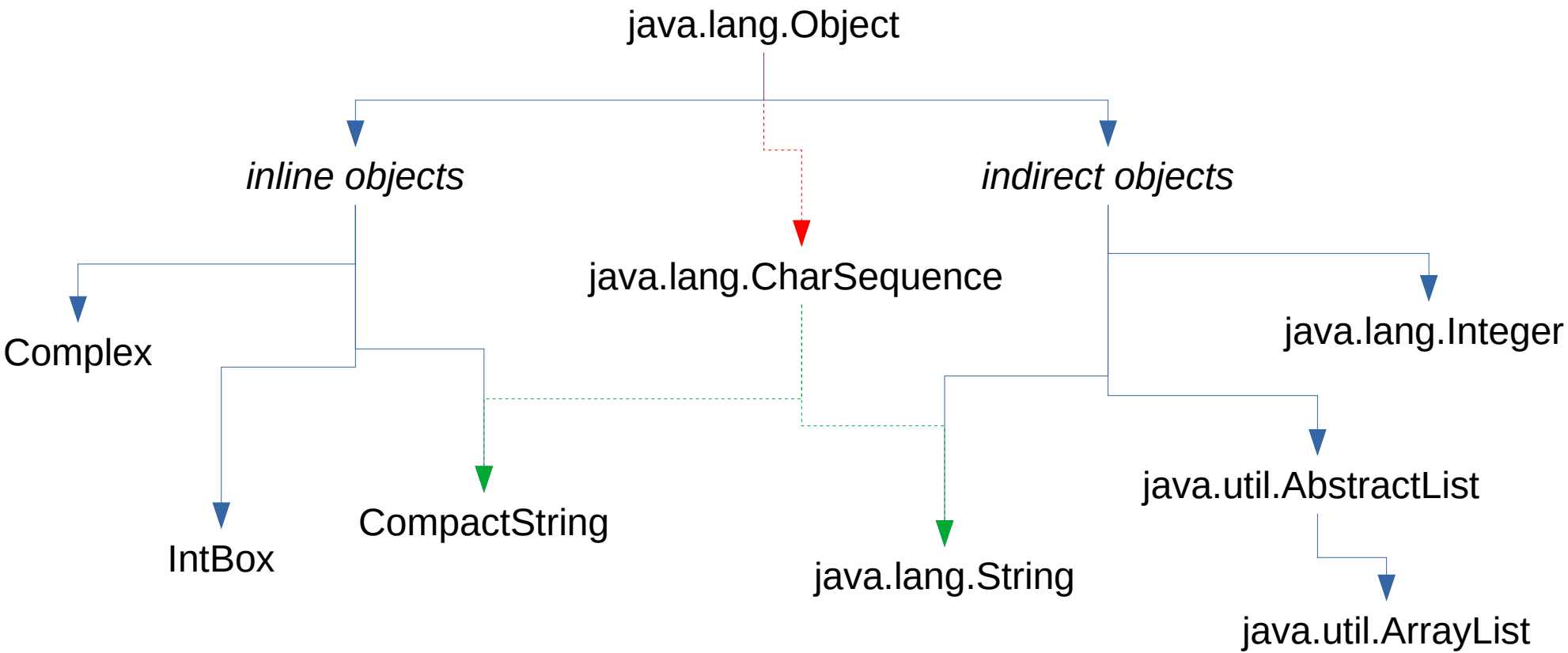
- println(Object)

L-world
(second prototype)

Object = root of all indirect objects



Object = root of every objects



L-world conversion

Subtype relationship

```
Complex c = ...
Object o = c;
```

Downcast (checkcast + nullcheck)

```
Object o2 = …
Complex c2 = (Complex) o2;
```

Bytecode representation

Problem: class late loading

In Java, a class is loaded as late as possible

```
class Holder { … }
class AnotherClass {
  public static Holder getMeAHolder() {
    return null;
  }
}
```

Even when getMeAHolder() is executed, Holder is not loaded

Inline class loading

but inline classes need to be loaded early

- memory layout
- method calling sequence
  - Scalarization even if method not inlined !
- verifier checks null pollution

solution: prefix class by ‘Q’ instead of ‘L’

‘Q’ => loaded early

IntBox (again)

java.lang.Integer written as an inline class

```
public inline class IntBox {
  private final int value;

  public IntBox(int value) {
    this.value = value;
  }

  public int intValue() {
    return value;
  }
- public static IntBox zero() {
    return new IntBox(0);
  }
}
```

IntBox as classfile

An inline class descriptor is prefixed by ‘Q’

```
public final inline class IntBox {
  private final int value;
    descriptor: I

  public static IntBox zero();
    descriptor: ()QintBox;

  ...
}
```

Problem: class initialization

In a constructor, you can see the fields being initialized

```
public inline class IntBox {
  private final int value;

  public IntBox(int value) {
    // here I can see the mutation !!
    this.value = value;
  }

  ...
}
```

True immutability ?

In a constructor, you can see the fields being initialized

=> not truly immutable

Solution: only initialize through a static factory

- Compiler transforms constructors to static methods

  “this” should not escape !

- Add 2 new opcodes: *defaultvalue* and *withfield*

Bytecode for IntBox constructor

```
public final inline class IntBox {
  private final int value;

  public static <init>(I)QIntBox;
       0: defaultvalue  #1                  // class IntBox
       3: astore_1
       4: iload_0
       5: aload_1
       6: swap
       7: withfield     #3                  // Field value:I
      10: astore_1
      11: aload_1
      12: areturn
```

IntBox

The opcode *defaultvalue* is accessible in Java

```
public inline class IntBox {
  private final int value;

  public IntBox(int value) {
    this.value = value;
  }
  public static IntBox zero() {
    return IntBox.default;
  }
}
```

Toward the preview release

Problem: array covariance

An array of Object

- an array of indirect type (array of *pointers*)
- an array of inline type (array of *structs*)

=> size of an array cell not constant !

Iterating over an array can be megamorphic !

Solution: Aggressive loop hoisting + loop duplication

Problem: subtyping relationship

All operations available on Object should be available on an inline class but ...

- No header
  - synchronized, wait/notify ?
  - System.identityHashCode ?
- No identity
  - Meaning of o == o2 if inline objects ?

No header

synchronized, wait/notify ?

Throw an IllegalMonitorStateException

- Avoid perf regression ?

System.identityHashCode ?

Redirected to hashCode()

No identity, == (aka acmp)

3 possible semantics

- always return false (no pointer)
- do a classcheck + components checks
  - perf regression in existing code ?
    - may stackoverflow / very slow if recursive
- redirect to call equals()
  - Perf regression ?

Still in flux !

Problem: erased generics

Inside a generics class, T is nullable

but inline classes are not nullable

Map<String,Complex> map = ...
map.get()

- API contract: Can return null
- Inline class not nullable: Can not return null

Interface == lightweight box

Complex.box is the nullable version of Complex
(an empty interface implemented by Complex)

```
// does not compile
Map<String,Complex> map = …

// ok !
Map<String,Complex.box> map = …
```

+ Auto boxing between Complex and Complex.box

Summary

Inline class - preview

- no identity, not mutable, not nullable, tearable
- runtime model
  - scalarization into registers on stack
  - **inlined** into another object/array cells on heap
- Object as **root** of everything
  - don’t use ==
- erased generic => lightweight boxing (interface)

Current Roadmap

Early access prototypes (http://jdk.java.net/)

- MVT
- Lworld 1, 2

  => We are here now !
- Lword 10 - **preview release** in Java
  - Support erased generics
- Lworld 100 – finale release
  - specialized generics

Questions ?

Supplementary slides

Eclair interface

Eclair interface

Write an inline class

- derives automatically an empty sealed interface
- Add unboxing between eclair and inline

Non nullable type → inline class

Nullable type → eclair interface

Example

```
public inline class Complex {
  private final double re;
  private final double im;

  public Complex add(Complex c) {
    return new Complex(re + c.re, im + c.im);
  }




}
```

Example

```
public inline class Complex implements Complex.box {
  private final double re;
  private final double im;

  public Complex add(Complex c) {
    return new Complex(re + c.re, im + c.im);
  }

  // generated automatically
  public sealed interface box permits Complex {
    // empty
  }
}
```

How to use it

Erased generics don’t allow inline class

```
List<Complex> l = …  // doesn’t compile
```

But using the eclair box is ok

```
List<Complex.box> l =
   List.of(new Complex(…)); // subtyping

Complex c = l.get(0);  // unboxing, may throw NPE
```

After the preview release

Problem: Value Based Class

Want to retrofit existing JDK classes

- java.util.Optional, java.time.LocalDate, etc.

Problem:

- value based class != an inline class
  - vbcs are nullable

Solution: hand retrofit as eclair

Change the VM spec

- So invokevirtual can call an interface

Change the VBC to be interface (eclair)

- add Optional.val as inline class
- JIT: Aggressive non null propagation
  - At worth, additional nullcheck

Inline class & concurrency

Value tearing

Accessing a value type concurrently

=> several reads/stores

Not a new problem

long/double on 32bits VMs has the same issue

No real solution => don’t do that !

Problem: Value tearing

```
inline class Value { final int x, y; public Value(int x, int y) { … } }

public static void main(String[] args) {
  var box = new Object() { Value shared; };
  var zero = new Value(0, 0);
  var one = new Value(1, 1);

  new Thread(() -> {
    for(;;) { box.shared = zero; }
  }).start();
  new Thread(() -> {
    for(;;) { box.shared = one; }
  }).start();

  for(;;) {
    var value = box.shared;
    if (value.x != value.y) {
     throw new AssertionError("oops");
    }
  }
```

Problem: Value tearing

```
inline class Value { final int x, y; public Value(int x, int y) { … } }

public static void main(String[] args) {
  var box = new Object() { Value shared; };
  var zero = new Value(0, 0);
  var one = new Value(1, 1);

  new Thread(() -> {
    for(;;) { box.shared = zero; }
  }).start();
  new Thread(() -> {
    for(;;) { box.shared = one; }
  }).start();

  for(;;) {
    var value = box.shared;
    if (value.x != value.y) {
     throw new AssertionError("oops");
    }
  }
```

Two stores

Two reads

Unprotected concurrent access