Проект Valhalla

Или как добавить в Java value типы,
не превращая ее в C++

Иван Углянский

Excelsior@Huawei

JUGNsk

sys.pro @ mmf.nsu

Алло, это отладочная?

@ugliansky

Иван Углянский

Excelsior@Huawei

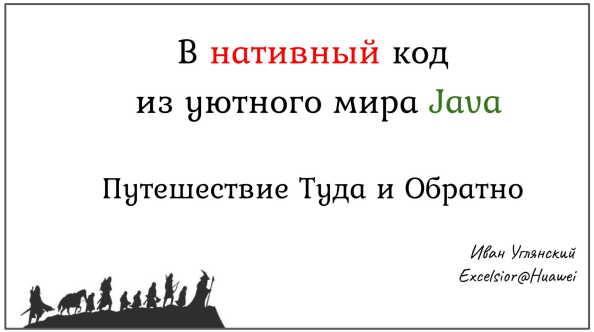
JUGNsk

sys.pro @ mmf.nsu

Алло, это отладочная?

@ugliansky

Люблю рассказывать
про мега-проекты:



<- Panama

<- Loom

Что За Вальхалла?

Project Valhalla

- Один из мега-проектов Java. Идет с 2014 года!

Project Valhalla

- Один из мега-проектов Java. Идет с 2014 года!
- JEP-401 (preview), JEP-513 (delivered), JEP null restricted values (draft), Parametric JVM (raw draft)

Project Valhalla

- Один из мега-проектов Java. Идет с 2014 года!
- JEP-401 (preview), JEP-513 (delivered), JEP null restricted values (draft), Parametric JVM (raw draft)
- a.k.a "Java's Epic Refactoring", a.k.a "L-world", a.k.a "добавление ~~value~~ ~~inline~~ ~~primitive~~ value классов"

Project Valhalla

- Один из мега-проектов Java. Идет с 2014 года!
- JEP-401 (preview), JEP-513 (delivered), JEP null restricted values (draft), Parametric JVM (raw draft)
- a.k.a "Java's Epic Refactoring", a.k.a "L-world", a.k.a "добавление ~~value~~ ~~inline~~ ~~primitive~~ value классов"
- Эпическая история борьбы с монструозным усложнением языка и JVM с (вроде бы) счастливым финалом

Project Valhalla

- Один из мега-проектов Java. Идет с 2014 года!
- JEP-401 (preview), JEP-513 (delivered), JEP null restricted values (draft), Parametric JVM (raw draft)
- a.k.a "Java's Epic Refactoring", a.k.a "L-world", a.k.a "добавление ~~value~~ ~~inline~~ ~~primitive~~ value классов"
- Эпическая история борьбы с монструозным усложнением языка и JVM с (вроде бы) счастливым финалом

(Почти) все примеры в этой презентации скомпилированы и запущены на early access build с поддержкой Valhalla.

```
# javac Test.java --enable-preview --release 27
# java --enable-preview Test
```

Наводящие вопросы

Наводящие вопросы #1

Q: что делает оператор new в Java?

```
record Point(int x, int y) {}
Point p = new Point(13, 42);
```

Наводящие вопросы #1

Q: что делает оператор new в Java?

```
record Point(int x, int y) {}
Point p = new Point(13, 42);
```

A: создает новый экземпляр класса

Наводящие вопросы #1

Q: что делает оператор new в Java?

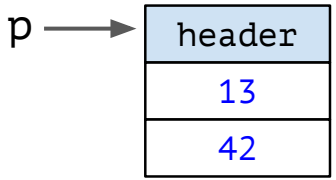
```
record Point(int x, int y) {}
Point p = new Point(13, 42);
```

A: создает новый экземпляр класса (под капотом по спецификации много чего: загрузка класса, выполнение блока статической инициализации, вызов конструктора, …)

Наводящие вопросы #1

Q: что делает оператор new в Java?

```
record Point(int x, int y) {}
Point p = new Point(13, 42);
```



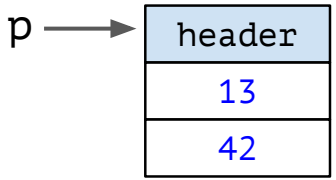
A: создает новый экземпляр класса (…);

Q: а **где** создает? В хипе?

Наводящие вопросы #1

Q: что делает оператор new в Java?

```
record Point(int x, int y) {}
Point p = new Point(13, 42);
```



A: создает новый экземпляр класса (…);

Q: а **где** создает? В хипе?

A: зачастую да, но вообще-то науке неизвестно, а спека не конкретизирует. Смотрим примеры:

Наводящие вопросы #1

```
record Point(int x, int y) {}

int getResult(int n) {
    int result = 0;
    for (int i = 0; i < n; i++) {
        var p = new Point(i, i);
        result = p.x + p.y;
    }
    return result;
}
```

Наводящие вопросы #1

```
record Point(int x, int y) {}

int getResult(int n) {
    int result = 0;
    for (int i = 0; i < n; i++) {
        var p = new Point(i, i);
        result = p.x + p.y;
    }
    return result;
}
```

```
0x0000026c5f72fbc4:   jne    0x0000026c5f72fc03

0x0000026c5f72fbca:   test   r8d,r8d
0x0000026c5f72fbcd:   jle    0x0000026c5f72fbe9

0x0000026c5f72fbcf:   add    r8d,r8d
0x0000026c5f72fbd2:   lea    eax,[r8-0x2]

0x0000026c5f72fbd6:   add    rsp,0x10
0x0000026c5f72fbda:   pop    rbp
0x0000026c5f72fbdb:   cmp    rsp,QWORD PTR [r15+0x460]
0x0000026c5f72fbe2:   ja     0x0000026c5f72fbed
0x0000026c5f72fbe8:   ret
0x0000026c5f72fbe9:   xor    eax,eax
0x0000026c5f72fbeb:   jmp    0x0000026c5f72fbd6
0x0000026c5f72fbed:   movabs r10,0x26c5f72fbdb
0x0000026c5f72fbf7:   mov    QWORD PTR [r15+0x478],r10
0x0000026c5f72fbfe:   jmp    0x0000026c5f021d00
```

Никаких следов Point и его аллокаций, компилятор C2 распылил его на атомы

Наводящие вопросы #1

```
record Point(int x, int y) {}

int getResult(int n) {
    int result = 0;
    for (int i = 0; i < n; i++) {
        var p = new Point(i, i);
        result = p.x + p.y;
    }
    return result;
}
```

```
0x0000026c5f72fbc4:   jne    0x0000026c5f72fc03

0x0000026c5f72fbca:   test   r8d,r8d
0x0000026c5f72fbcd:   jle    0x0000026c5f72fbe9

0x0000026c5f72fbcf:   add    r8d,r8d
0x0000026c5f72fbd2:   lea    eax,[r8-0x2]

0x0000026c5f72fbd6:   add    rsp,0x10
0x0000026c5f72fbda:   pop    rbp
0x0000026c5f72fbdb:   cmp    rsp,QWORD PTR [r15+0x460]
0x0000026c5f72fbe2:   ja     0x0000026c5f72fbed
0x0000026c5f72fbe8:   ret
0x0000026c5f72fbe9:   xor    eax,eax
0x0000026c5f72fbeb:   jmp    0x0000026c5f72fbd6
0x0000026c5f72fbed:   movabs r10,0x26c5f72fbdb
0x0000026c5f72fbf7:   mov    QWORD PTR [r15+0x478],r10
0x0000026c5f72fbfe:   jmp    0x0000026c5f021d00
```

Никаких следов Point и его аллокаций, компилятор C2 распылил его на атомы

Спровоцируем аллокацию, передав в no-inline метод!

Наводящие вопросы #1

```
record Point(int x, int y) {}

static void print(Point p) {
   // тяжесть, не инлайнится
}

int getResult(int n) {
    int result = 0;
    for (int i = 0; i < n; i++) {
        var p = new Point(i, i);
        print(p);
        result = p.x + p.y;
    }
    return result;
}
```

```
…
0x00000184a31fc695:   jae    0x00000184a31fc70f
0x00000184a31fc697:   mov    QWORD PTR [r15+0x1b8],r10
0x00000184a31fc69e:   prefetchw BYTE PTR [r10+0xc0]
0x00000184a31fc6a6:   movabs r10,0x230cb230               ;   {metadata('Test$Point')}
0x00000184a31fc6b0:   mov    r10,QWORD PTR [r10+0xb0]
0x00000184a31fc6b7:   mov    QWORD PTR [rdx],r10
0x00000184a31fc6ba:   mov    DWORD PTR [rdx+0x8],0x230cb230;   {metadata('Test$Point')}
0x00000184a31fc6c1:   mov    DWORD PTR [rdx+0x14],r12d    ;*new {reexecute=0 rethrow=0 return_oop=0 return_scalarized=0}
                                                          ; - Test::getResult@9 (line 14)
0x00000184a31fc6c5:   mov    DWORD PTR [rdx+0xc],r8d      ;*putfield x {reexecute=0 rethrow=0 return_oop=0 return_scalarized=0}
                                                          ; - Test$Point::<init>@6 (line 3)
                                                          ; - Test::getResult@15 (line 14)
0x00000184a31fc6c9:   mov    DWORD PTR [rdx+0x10],r8d     ;*invokespecial <init> {reexecute=0 rethrow=0 return_oop=0 return_scalarized=0}
                                                          ; - Test::getResult@15 (line 14)
0x00000184a31fc6cd:   mov    r10d,r8d
0x00000184a31fc6d0:   mov    DWORD PTR [rsp],r8d
0x00000184a31fc6d4:   add    r10d,r8d                     ;*iadd {reexecute=0 rethrow=0 return_oop=0 return_scalarized=0}
                                                          ; - Test::getResult@30 (line 15)
0x00000184a31fc6d7:   mov    DWORD PTR [rsp+0x4],r10d
0x00000184a31fc6dc:   data16 xchg ax,ax
0x00000184a31fc6df:   call   0x00000184a31d1260          ; ImmutableOopMap {}
                                                          ;*invokestatic print {reexecute=0 rethrow=0 return_oop=0 return_scalarized=0}
                                                          ; - Test::getResult@34 (line 16)
…                                                         ;   {static_call}
0x00000184a31fc70e:   ret
0x00000184a31fc70f:   mov    DWORD PTR [rsp],r8d
0x00000184a31fc713:   movabs rdx,0x230cb230               ;   {metadata('Test$Point')}
0x00000184a31fc71d:   xor    r8d,r8d
0x00000184a31fc720:   data16 xchg ax,ax
0x00000184a31fc723:   call   0x00000184a2ba2980          ; ImmutableOopMap {}
                                                          ;*new {reexecute=0 rethrow=0 return_oop=1 return_scalarized=0}
                                                          ; - Test::getResult@9 (line 14)
                                                          ;   {runtime_call _new_instance_Java}
0x00000184a31fc728:   nop    DWORD PTR [rax+rax*1+0x1000298];   {other}
0x00000184a31fc730:   mov    rdx,rax
0x00000184a31fc733:   mov    r8d,DWORD PTR [rsp]
0x00000184a31fc737:   jmp    0x00000184a31fc6c5
…
```

```
…
0x00000184a31fc695:   jae    0x00000184a31fc70f
0x00000184a31fc697:   mov    QWORD PTR [r15+0x1b8],r10
0x00000184a31fc69e:   prefetchw BYTE PTR [r10+0xc0]
0x00000184a31fc6a6:   movabs r10,0x230cb230               ;   {metadata('Test$Point')}
0x00000184a31fc6b0:   mov    r10,QWORD PTR [r10+0xb0]
0x00000184a31fc6b7:   mov    QWORD PTR [rdx],r10
0x00000184a31fc6ba:   mov    DWORD PTR [rdx+0x8],0x230cb230;   {metadata('Test$Point')}
0x00000184a31fc6c1:   mov    DWORD PTR [rdx+0x14],r12d    ;*new {reexecute=0 rethrow=0 return_oop=0 return_scalarized=0}
                                                          ; - Test::getResult@9 (line 14)
0x00000184a31fc6c5 :   mov    DWORD PTR [rdx+0xc],r8d      ;*putfield x {reexecute=0 rethrow=0 return_oop=0 return_scalarized=0}
                                                          ; - Test$Point::<init>@6 (line 3)
                                                          ; - Test::getResult@15 (line 14)
0x00000184a31fc6c9:   mov    DWORD PTR [rdx+0x10],r8d     ;*invokespecial <init> {reexecute=0 rethrow=0 return_oop=0 return_scalarized=0}
                                                          ; - Test::getResult@15 (line 14)
0x00000184a31fc6cd:   mov    r10d,r8d
0x00000184a31fc6d0:   mov    DWORD PTR [rsp],r8d
0x00000184a31fc6d4:   add    r10d,r8d                     ;*iadd {reexecute=0 rethrow=0 return_oop=0 return_scalarized=0}
                                                          ; - Test::getResult@30 (line 15)
0x00000184a31fc6d7:   mov    DWORD PTR [rsp+0x4],r10d
0x00000184a31fc6dc:   data16 xchg ax,ax
0x00000184a31fc6df:   call   0x00000184a31d1260          ; ImmutableOopMap {}
                                                          ;*invokestatic print  {reexecute=0 rethrow=0 return_oop=0 return_scalarized=0}
                                                          ; - Test::getResult@34 (line 16)
…                                                         ;   {static_call}
0x00000184a31fc70e:   ret
0x00000184a31fc70f :   mov    DWORD PTR [rsp],r8d
0x00000184a31fc713:   movabs rdx,0x230cb230               ;   {metadata('Test$Point')}
0x00000184a31fc71d:   xor    r8d,r8d
0x00000184a31fc720:   data16 xchg ax,ax
0x00000184a31fc723:   call   0x00000184a2ba2980          ; ImmutableOopMap {}
                                                          ;*new {reexecute=0 rethrow=0 return_oop=1 return_scalarized=0}
                                                          ; - Test::getResult@9 (line 14)
                                                          ;   {runtime_call _new_instance_Java }
0x00000184a31fc728:   nop    DWORD PTR [rax+rax*1+0x1000298];   {other}
0x00000184a31fc730:   mov    rdx,rax
0x00000184a31fc733:   mov    r8d,DWORD PTR [rsp]
0x00000184a31fc737:   jmp    0x00000184a31fc6c5
…
```

Вот и аллокация в хипе!

Наводящие вопросы #1

```
record Point(int x, int y) {}

static void print(Point p) {
    // тяжесть, не инлайнится
}

int getResult(int n) {
    int result = 0;
    for (int i = 0; i < n; i++) {
        var p = new Point(i, i);
        print(p);
        result = p.x + p.y;
    }
    return result;
}
```

В этот раз C2 не справился, поэтому аллокация осталась.

Наводящие вопросы #1

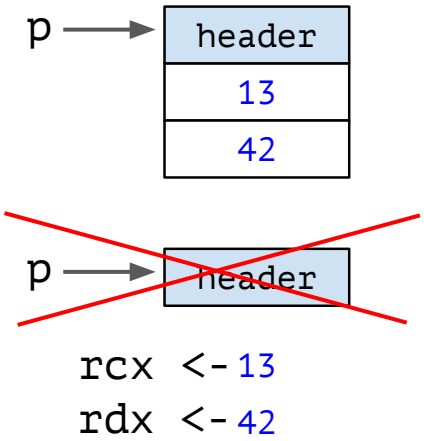
```
record Point(int x, int y) {}

static void print(Point p) {
    // тяжесть, не инлайнится
}

int getResult(int n) {
    int result = 0;
    for (int i = 0; i < n; i++) {
        var p = new Point(i, i);
        print(p);
        result = p.x + p.y;
    }
    return result;
}
```

В этот раз C2 не справился, поэтому аллокация осталась.

Все потому, что в прошлом примере сработала оптимизация, известная, как Scalar Replacement, а здесь уже сложновато.

Наводящие вопросы #1

Вывод: объект может существовать в хипе, а может и скаляризоваться (или аллоцироваться на стеке), зависит от использований.

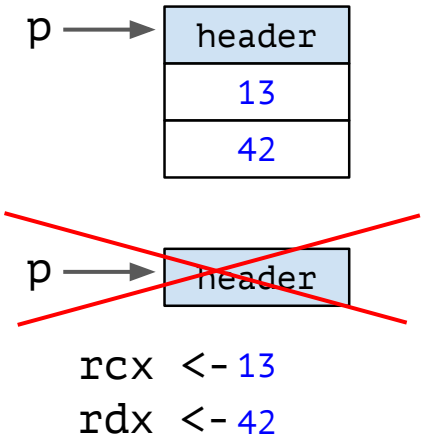


Наводящие вопросы #1

Вывод: объект может существовать в хипе, а может и скаляризоваться (или аллоцироваться на стеке), зависит от использований.

Скаляризация или стек-аллокация очень заманчивы в плане производительности:

- Нет трат на аллокацию,
- Нет нагрузки на GC,
- Нет дополнительных трат на заголовок



Наводящие вопросы #2

Q: какая раскладка в памяти будет у вот таких объектов?

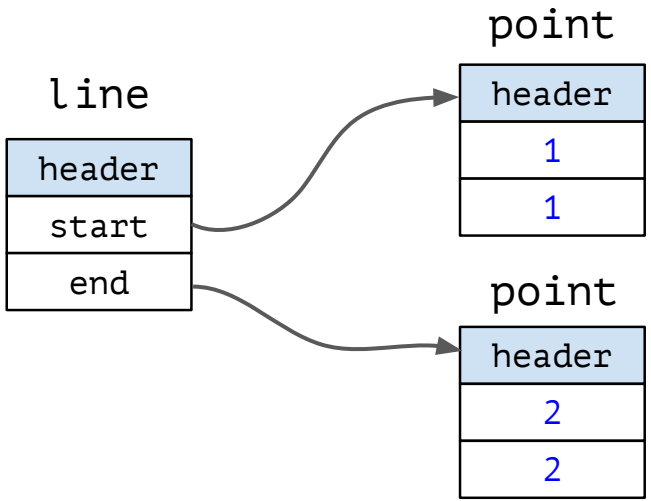
```
record Point(int x, int y) {}
record Line(Point start, Point end) {}

var line = new Line(new Point(1,1), new Point(2,2));
```

Наводящие вопросы #2

Q: какая раскладка в памяти будет у вот таких объектов?

```
record Point(int x, int y) {}
record Line(Point start, Point end) {}

var line = new Line(new Point(1,1), new Point(2,2));
```

A:



Наводящие вопросы #2

Q: какая раскладка в памяти будет у вот таких объектов?

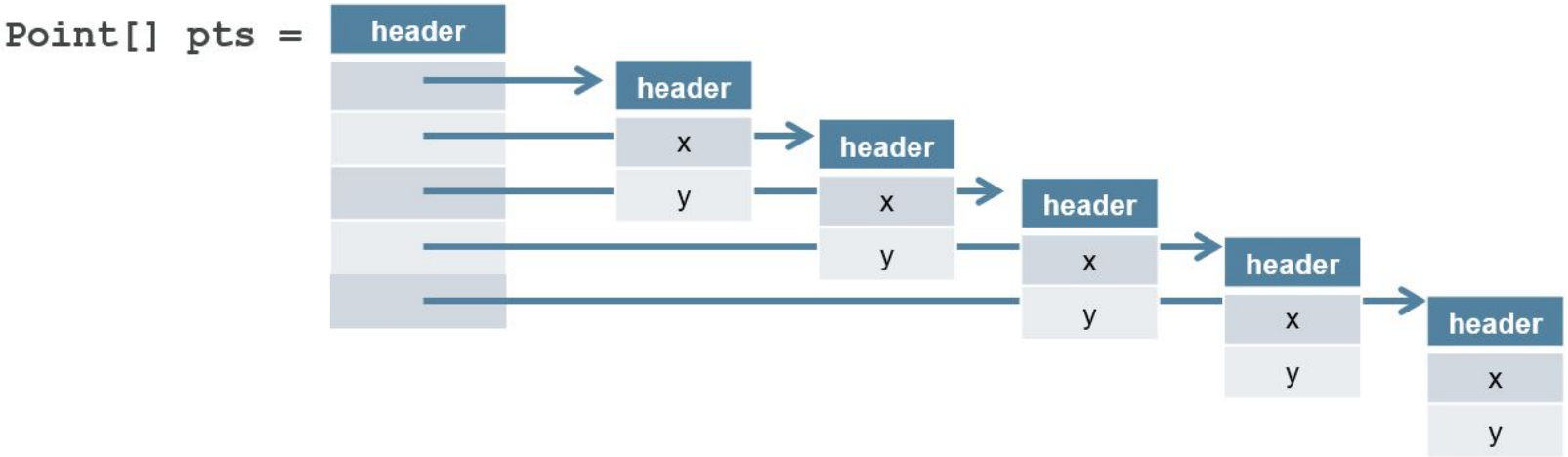
```
record Point(int x, int y) {}

var points = new Point[10];
```

Наводящие вопросы #2

Q: какая раскладка в памяти будет у вот таких объектов?

```
record Point(int x, int y) {}

var points = new Point[10];
```

A:

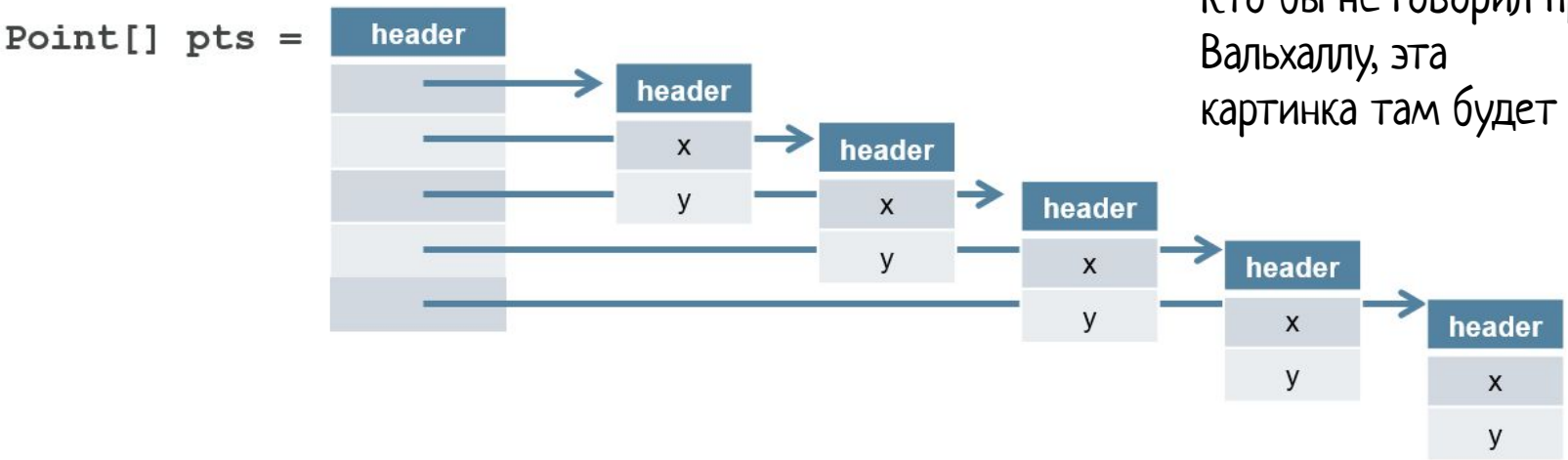


Наводящие вопросы #2

Q: какая раскладка в памяти будет у вот таких объектов?

```
record Point(int x, int y) {}

var points = new Point[10];
```

A:



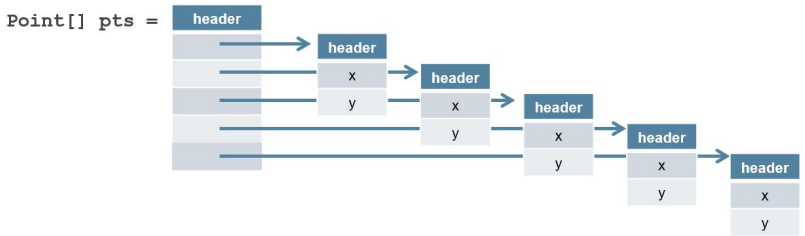
Кто бы не говорил про Вальхаллу, эта картинка там будет

Наводящие вопросы #2

Q: какая раскладка в памяти будет у вот таких объектов?

```
var points = new Point[10];
```

A:

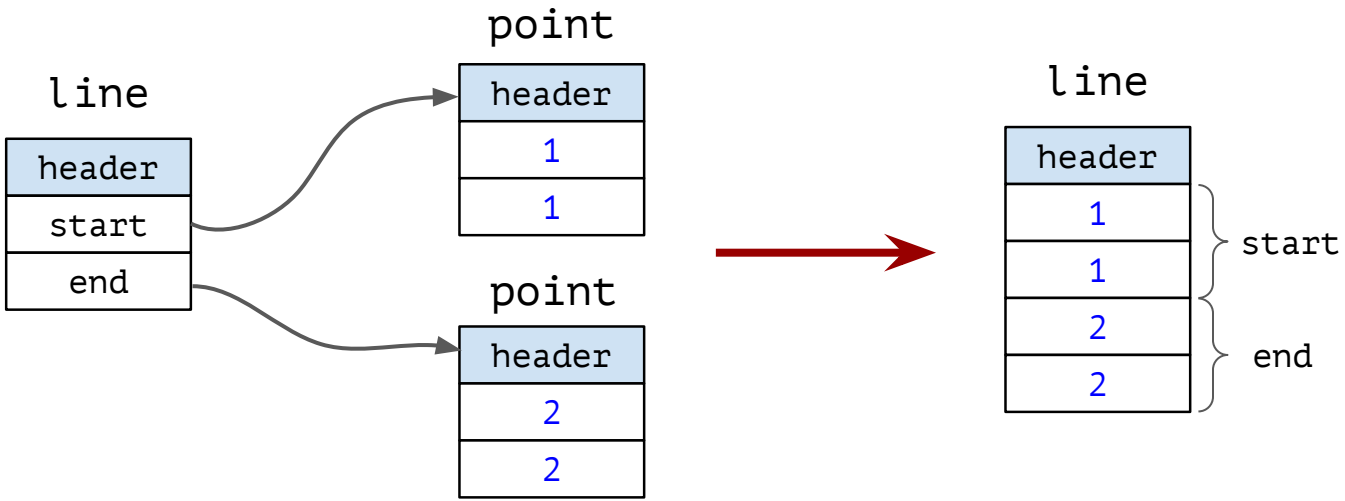


С такой раскладкой куча проблем:

- трата памяти на заголовки для объектов,
- плохая локальность данных,
- дополнительные разыменования 🤮

Наводящие вопросы #2

Q: почему бы не разрешить переходить вот к такой (c-struct like) плоской раскладке (аналогично с массивами)?



Наводящие вопросы #2

Q: почему бы не разрешить переходить вот к такой (c-struct like) плоской раскладке (аналогично с массивами)?

A: если начать делать это без ограничений, то:

- появятся объекты огромного размера (а это неочевидная нагрузка и помеха GC + неатомарный доступ, обсудим позже)

Наводящие вопросы #2

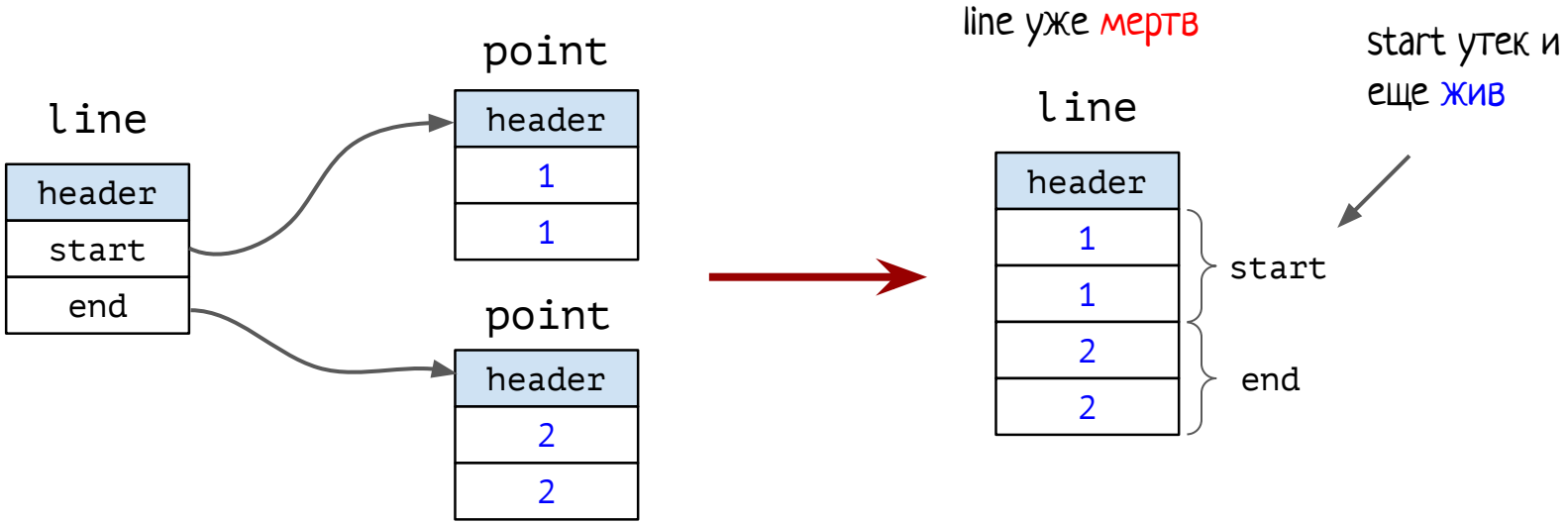
Q: почему бы не разрешить переходить вот к такой (c-struct like) плоской раскладке (аналогично с массивами)?

A: если начать делать это без ограничений, то:

- появятся объекты огромного размера (а это неочевидная нагрузка и помеха GC + неатомарный доступ, обсудим позже)
- а что, если ссылка на объект внутри другого объекта пережила ссылку на объемлющий объект?

Наводящие вопросы #2

Q: почему бы не разрешить переходить вот к такой (c-struct like) плоской раскладке (аналогично с массивами)?



Наводящие вопросы #2

Q: почему бы не разрешить переходить вот к такой (c-struct like) плоской раскладке (аналогично с массивами)?

A: если начать делать это без ограничений, то:

- появятся объекты огромного размера (а это неочевидная нагрузка и помеха GC + неатомарный доступ, обсудим позже)
- а что, если ссылка на объект внутри другого объекта пережила ссылку на объемлющий объект?

  Хранить ссылки на объемлющий объект?
  Искать объемлющие динамически?
  Огромное усложнение GC!

Наводящие вопросы #2

Q: почему бы не разрешить переходить вот к такой (c-struct like) плоской раскладке (аналогично с массивами)?

A: если начать делать это без ограничений, то:

- появятся объекты огромного размера (а это неочевидная нагрузка и помеха GC + неатомарный доступ, обсудим позже)
- а что, если ссылка на объект внутри другого объекта пережила ссылку на объемлющий объект?
- а еще это не в духе Java: оптимизации - дело JVM, а не Java программиста, он пусть думает о семантике.

Наводящие вопросы #2

Q: какая раскладка в памяти будет у вот таких объектов?
…

Q: почему бы не разрешить переходить вот к такой (c-struct like) плоской раскладке?

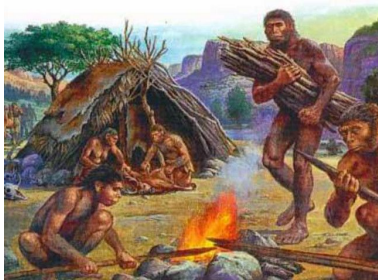
…

Вывод: позволять пользователю "инлайнить" объекты руками - слишком круто (сложные трейдофы), но вот позволять так делать JVM, когда это безопасно… это была бы очень мощная оптимизация!

Наводящие вопросы #3

Наводящие вопросы #3

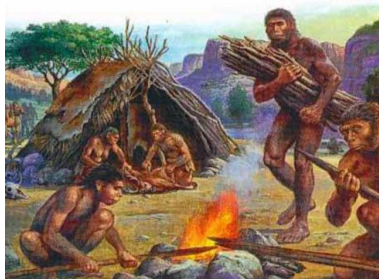
В Java есть примитивы (byte, short, int, double, …), а есть их боксы (j.l.Byte, j.l.Short, j.l.Double, …)



Наводящие вопросы #3

В Java есть примитивы (byte, short, int, double, …), а есть их боксы (j.l.Byte, j.l.Short, j.l.Double, …)

Q: зачем нужны боксы?



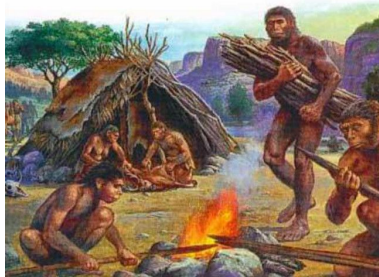
Наводящие вопросы #3

В Java есть примитивы (byte, short, int, double, …), а есть их боксы (j.l.Byte, j.l.Short, j.l.Double, …)

Q: зачем нужны боксы?
A: чтобы писать дженеричный код!

```
ArrayList<int> arr;
```
← не скомпилируется

```
Object answer = 42;
```
← создастся Box (сам int не лезет)

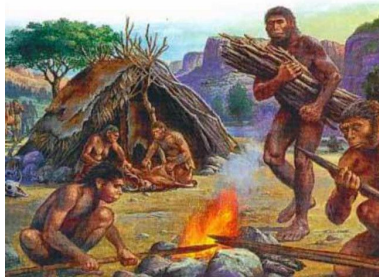


Наводящие вопросы #3

В Java есть примитивы (byte, short, int, double, …), а есть их боксы (j.l.Byte, j.l.Short, j.l.Double, …)

Q: зачем нужны боксы?
A: чтобы писать дженеричный код!

Q: а зачем тогда нужны примитивы? Ведь боксы явно круче!

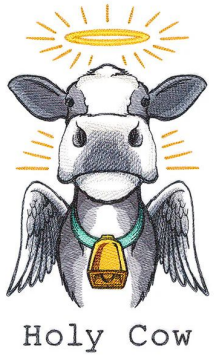


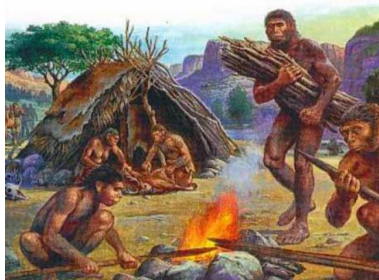
Наводящие вопросы #3

В Java есть примитивы (byte, short, int, double, …), а есть их боксы (j.l.Byte, j.l.Short, j.l.Double, …)

Q: зачем нужны боксы?
A: чтобы писать дженеричный код!

Q: а зачем тогда нужны примитивы? Ведь боксы явно круче!
(кроме обратной совместимости)





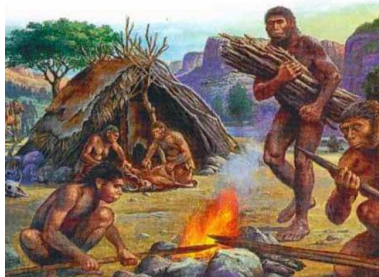
Наводящие вопросы #3

В Java есть примитивы (byte, short, int, double, …), а есть их боксы (j.l.Byte, j.l.Short, j.l.Double, …)

Q: зачем нужны боксы?
A: чтобы писать дженеричный код!

Q: а зачем тогда нужны примитивы? Ведь боксы явно круче! (кроме обратной совместимости)

A: так для производительности же!



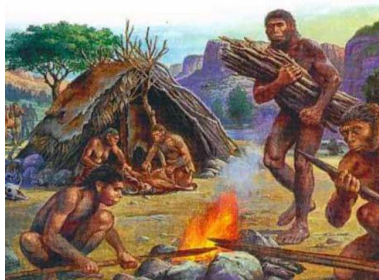
Наводящие вопросы #3

В Java есть примитивы (byte, short, int, double, …), а есть их боксы (j.l.Byte, j.l.Short, j.l.Double, …)

Q: зачем нужны боксы?
A: чтобы писать дженеричный код!

Q: а зачем тогда нужны примитивы? Ведь боксы явно круче! (кроме обратной совместимости)

A: так для производительности же! Никто не будет писать перемножение матриц на j.l.Double, медленно.



Наводящие вопросы #3

В Java есть примитивы (byte, short, int, double, …), а есть их боксы (j.l.Byte, j.l.Short, j.l.Double, …)

Q: зачем нужны боксы?
A: чтобы писать дженеричный код!

Q: а зачем тогда нужны примитивы? Ведь боксы явно круче! (кроме обратной совместимости)

A: так для производительности же! Никто не будет писать перемножение матриц на j.l.Double, медленно.

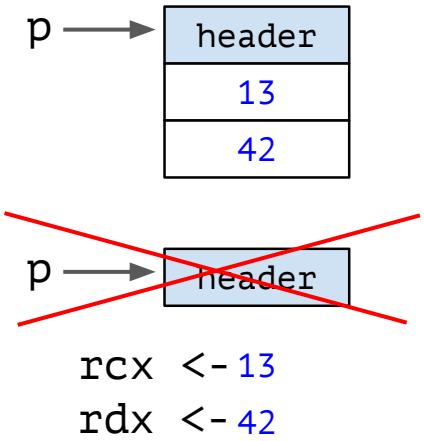
А примитивам: не нужны заголовки, не напрягают GC, сразу скаляризованы (ну, они же скаляры), живут внутри объектов… ничего не напоминает?

Наводящие вопросы #1

Вывод: объект может существовать в хипе, а может и скаляризоваться (или аллоцироваться на стеке), зависит от использований.

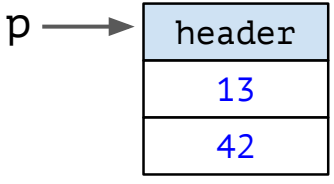
Скаляризация или стек-аллокация очень заманчивы в плане производительности:

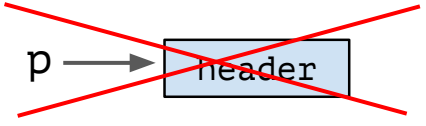
- Нет трат на аллокацию,
- Нет нагрузки на GC,
- Нет дополнительных трат на заголовок



Наводящие вопросы #1

Вывод: объект может существовать в хипе, а может и скаляризоваться (или аллоцироваться на стеке), зависит от использований.






```
rcx <- 13
rdx <- 42
```

Скаляризация или стек-аллокация очень заманчивы в плане производительности:

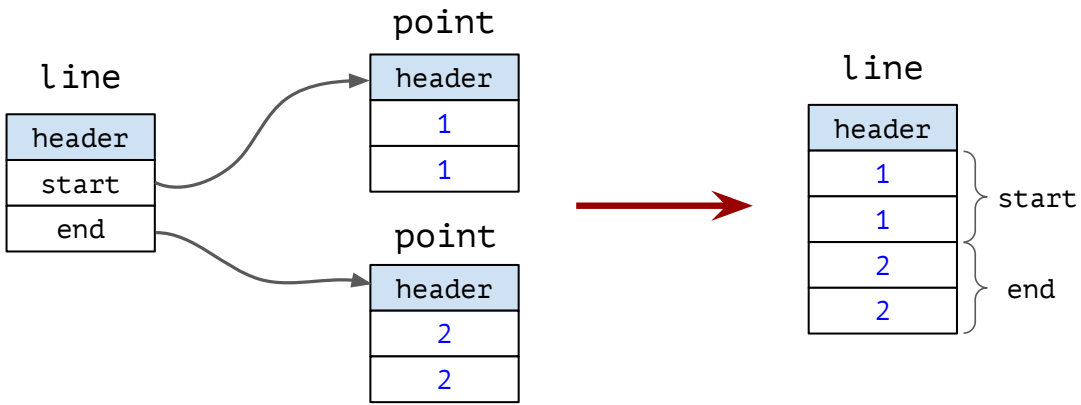
- Нет трат на аллокацию,
- Нет нагрузки на GC,
- Нет дополнительных трат на заголовок

… буквально как с примитивами

Наводящие вопросы #2

Q: какая раскладка в памяти будет у вот таких объектов?
…

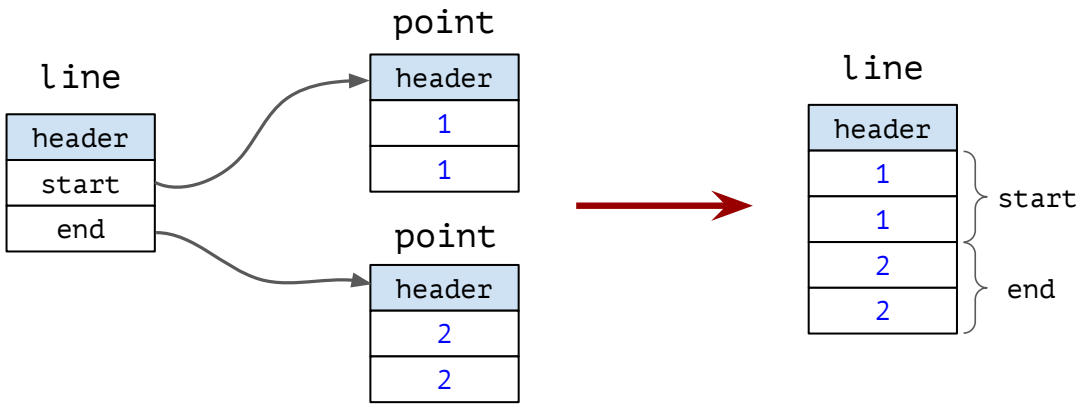
Q: почему бы не разрешить переходить вот к такой (c-struct like) плоской раскладке?



Наводящие вопросы #2

Q: какая раскладка в памяти будет у вот таких объектов?
…

Q: почему бы не разрешить переходить вот к такой (c-struct like) плоской раскладке?



… чтобы было буквально как с полями примитивами

Наводящие вопросы #3

В Java есть примитивы (byte, short, int, double,...), а есть их боксы (j.l.Byte, j.l.Short, j.l.Double, …)

Q: зачем нужны боксы?
A: обобщенный код

Q: а зачем тогда нужны примитивы? Ведь боксы явно круче!
A: обратная совместимость и производительность

Наводящие вопросы #3

В Java есть примитивы (byte, short, int, double,...), а есть их боксы (j.l.Byte, j.l.Short, j.l.Double, …)

Q: зачем нужны боксы?
A: обобщенный код

Q: а зачем тогда нужны примитивы? Ведь боксы явно круче!
A: обратная совместимость и производительность

Q: а что делать, если мне нужны примитивы посложнее, которых в Java нет? Комплексные числа? Точки на плоскости?

Наводящие вопросы #3

В Java есть примитивы (byte, short, int, double,...), а есть их боксы (j.l.Byte, j.l.Short, j.l.Double, …)

Q: зачем нужны боксы?
A: обобщенный код

Q: а зачем тогда нужны примитивы? Ведь боксы явно круче!
A: обратная совместимость и производительность

Q: а что делать, если мне нужны примитивы посложнее, которых в Java нет? Комплексные числа? Точки на плоскости?

A: или страдай с парой даблов, или теряй производительность с классами

Project Valhalla: цели

- Взять лучшее из двух миров: позволить пользователям писать удобные классы, которые дают производительность примитивов;

Project Valhalla: цели

- Взять лучшее из двух миров: позволить пользователям писать удобные классы, которые дают производительность примитивов;

- Объекты таких классов вероятнее будут скаляризовываться и смогут располагаться внутри других объектов. Но только по решению JVM;

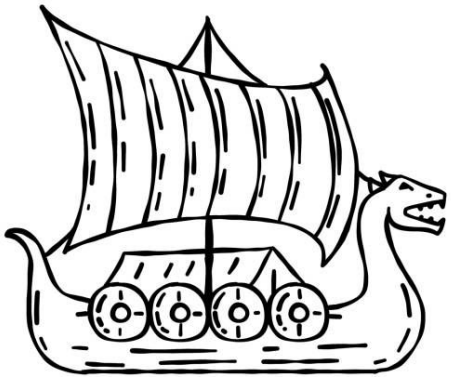
Project Valhalla: цели

- Взять лучшее из двух миров: позволить пользователям писать удобные классы, которые дают производительность примитивов;
- Объекты таких классов вероятнее будут скаляризовываться и смогут располагаться внутри других объектов. Но только по решению JVM;
- Со временем это позволит уйти от великого раскола между примитивами и классами в Java (в том числе в дженериках)

Project Valhalla: цели

- Взять лучшее из двух миров: позволить пользователям писать удобные классы (**value классы**), которые дают производительность примитивов;
- Объекты таких классов вероятнее будут скаляризовываться и смогут располагаться внутри других объектов. Но только по решению JVM;
- Со временем это позволит уйти от великого раскола между примитивами и классами в Java (в том числе в дженериках)

Как попасть в Вальхаллу?



Identity

Истинный путь в Вальхаллу - найти свою идентичность 💜

Identity

Истинный путь в Вальхаллу - ~~найти~~ потерять свою идентичность

Identity

Что такое identity в контексте языков программирования?

Identity

Что такое identity в контексте языков программирования?

Identity - это возможность отличить объекты друг от друга, даже если их содержимое побитово совпадает.

Identity

Что такое identity в контексте языков программирования?

Identity - это возможность отличить объекты друг от друга, даже если их содержимое побитово совпадает.

Примеры из Java, у чего есть identity, а у чего нет?

Identity

Что такое identity в контексте языков программирования?

Identity - это возможность отличить объекты друг от друга, даже если их содержимое побитово совпадает.

Примеры из Java, у чего есть identity, а у чего нет?

```
record Point(int x, int y) {}
Point p1 = new Point(13, 42);
Point p2 = new Point(13, 42);

System.out.println(p1 == p2); // false
```

Identity

Что такое identity в контексте языков программирования?

Identity - это возможность отличить объекты друг от друга, даже если их содержимое побитово совпадает.

Примеры из Java, у чего есть identity, а у чего нет?

```
int x = 42;
int y = 42;

System.out.println(x == y); // true
```

У классов есть identity, у примитивов - нет.

Identity

Что такое identity в контексте языков программирования?

Identity - это возможность отличить объекты друг от друга, даже если их содержимое побитово совпадает.

Примеры из Java, у чего есть identity, а у чего нет?

```
Integer x = 42;
Integer y = 42;

System.out.println(x == y); // ???
```

У классов есть identity, у примитивов - нет.

Identity

Что такое identity в контексте языков программирования?

Identity - это возможность отличить объекты друг от друга, даже если их содержимое побитово совпадает.

Примеры из Java, у чего есть identity, а у чего нет?

```
Integer x = 42;
Integer y = 42;

System.out.println(x == y); // true
```

Почему?? Это же классы!

У классов есть identity, у примитивов - нет.

Identity

Что такое identity в контексте языков программирования?

Identity - это возможность отличить объекты друг от друга, даже если их содержимое побитово совпадает.

Примеры из Java, у чего есть identity, а у чего нет?

```
Integer x = 10042;
Integer y = 10042;

System.out.println(x == y); // false
```

Почему?? Это же классы!
Да закэшировали прост))

У классов есть identity, у примитивов - нет.

Identity

Что такое identity в контексте языков программирования?

Identity - это возможность отличить объекты друг от друга, даже если их содержимое побитово совпадает.

У классов в Java есть identity, у примитивов - нет.

Как реализовать такое свойство?

Identity

Что такое identity в контексте языков программирования?

Identity - это возможность отличить объекты друг от друга, даже если их содержимое побитово совпадает.

У классов в Java есть identity, у примитивов - нет.

Как реализовать такое свойство? Например, брать адреса объектов в памяти. Так сделано в Java, но это именно деталь реализации, можно реализовать как угодно.

Identity

Что такое identity в контексте языков программирования?

Identity - это возможность отличить объекты друг от друга, даже если их содержимое побитово совпадает.

…

А зачем классам вообще identity? Что это дает?

Identity

```
public static class Person {
    String name;
    int age;

    public Person(String name, int age) {
        this.name = name;
        this.age = age;
    }

    public void setName(String newName) {
        this.name = newName;
    }
}
```

Identity

```
public static class Person {
    String name;
    int age;

    public Person(String name, int age) { … }
    public void setName(String newName) { … }
}

void main() {
    Person p1 = new Person("Ivanova", 18);
    Person p2 = new Person("Petrova", 18);
    System.out.println("Originally:" + (p1 == p2)); // false


}
```

Identity

```
public static class Person {
    String name;
    int age;

    public Person(String name, int age) { … }
    public void setName(String newName) { … }
}

void main() {
    Person p1 = new Person("Ivanova", 18);
    Person p2 = new Person("Petrova", 18);
    System.out.println("Originally:" + (p1 == p2)); // false
    p2.setName("Ivanova");
    System.out.println("But later:" + (p1 == p2));
}
```

Identity

```
public static class Person {
    String name;
    int age;

    public Person(String name, int age) { … }
    public void setName(String newName) { … }
}

void main() {
    Person p1 = new Person("Ivanova", 18);
    Person p2 = new Person("Petrova", 18);
    System.out.println("Originally:" + (p1 == p2)); // false
    p2.setName("Ivanova");
    System.out.println("But later:" + (p1 == p2));  // что же она,
                                                    // другим
                                                    // человеком
}                                                   // стала?
```

Identity

```
public static class Person {
    String name;
    int age;

    public Person(String name, int age) { … }
    public void setName(String newName) { … }
}

void main() {
    Person p1 = new Person("Ivanova", 18);
    Person p2 = new Person("Petrova", 18);
    System.out.println("Originally:" + (p1 == p2)); // false
    p2.setName("Ivanova");
    System.out.println("But later:" + (p1 == p2));  // false
}
```

Identity

Другим примером будет два разных хранилища, в которых в какой-то момент времени оказались одинаковые элементы. Это все еще разные хранилища!

```
public static class Person {
    String name;
    int age;

    public Person(String name, int age) { … }
    public void setName(String newName) { … }
}

void main() {
    Person p1 = new Person("Ivanova", 18);
    Person p2 = new Person("Petrova", 18);
    System.out.println("Originally:" + (p1 == p2)); // false
    p2.setName("Ivanova");
    System.out.println("But later:" + (p1 == p2));  // false
}
```

Identity

Что такое identity в контексте языков программирования?

Identity - это возможность отличить объекты друг от друга, даже если их содержимое побитово совпадает.

…

А зачем классам вообще identity? Что это дает?

Это дает возможность работать с **мутабельными** объектами (если в какой-то момент значения совпали, это еще не значит, что и объекты совпали)

Identity

Что такое identity в контексте языков программирования?

Identity - это возможность отличить объекты друг от друга, даже если их содержимое побитово совпадает.

…

А зачем классам вообще identity? Что это дает?

Это дает возможность работать с **мутабельными** объектами (если в какой-то момент значения совпали, это еще не значит, что и объекты совпали)

Еще один момент: синхронизация. Умеем различать => можем на конкретном объекте синхронизироваться.

Identity

Что такое identity в контексте языков программирования?

Identity - это возможность отличить объекты друг от друга, даже если их содержимое побитово совпадает.

…

А что позволило бы делать отсутствие identity?

Наводящие вопросы #1

```
record Point(int x, int y) {}

static void print(Point p) {
    // тяжесть, не инлайнится
}

int getResult(int n) {
    int result = 0;
    for (int i = 0; i < n; i++) {
        var p = new Point(i, i);
        print(p);
        result = p.x + p.y;
    }
    return result;
}
```

Почему здесь начались аллокации?

Почему не передать поля Point на регистрах в метод Print?

Наводящие вопросы #1

```
class Point { … }
Point answer = new Point(42, 42); // fields don't matter

static void print(Point p) {
    if (p == answer) { print("answer!"); }
    ...
}

int getResult(int n) {
    int result = 0;
    for (int i = 0; i < n; i++) {
        var p = new Point(i, i);
        print(p);
        result = p.x + p.y;
    }
    return result;
}
```

Почему здесь начались аллокации?

Почему не передать поля Point на регистрах в метод Print?

Наводящие вопросы #1

```
class Point { … }
Point answer = new Point(42, 42);

static void print(Point p) {
    if (p == answer) { print("answer!"); }
    ...
}

int getResult(int n) {
    int result = 0;
    for (int i = 0; i < n; i++) {
        var p = new Point(i, i);
        print(p);
        result = p.x + p.y;
    }
    return result;
}
```

Вдруг там начнут сравнивать по identity? А у вас объект на регистрах!

Почему здесь начались аллокации?

Почему не передать поля Point на регистрах в метод Print?

Наводящие вопросы #1

```
record Point(int x, int y) {}

static void print(Point p) {
   synchronized (p) { … }
   ...
}

int getResult(int n) {
    int result = 0;
    for (int i = 0; i < n; i++) {
        var p = new Point(i, i);
        print(p);
        result = p.x + p.y;
    }
    return result;
}
```

Вдруг решили синхронизироваться? А у вас объект на регистрах!

Почему здесь начались аллокации?

Почему не передать поля Point на регистрах в метод Print?

Наводящие вопросы #1

А вот если бы identity у таких классов не было, я бы спокойно скопировал значения на регистры, ведь меня бы никто никогда identity и не спросил

```
record Point(int x, int y) {}

static void print(Point p) {
    synchronized (p) { … }
    ...
}

int getResult(int n) {
    int result = 0;
    for (int i = 0; i < n; i++) {
        var p = new Point(i, i);
        print(p);
        result = p.x + p.y;
    }
    return result;
}
```

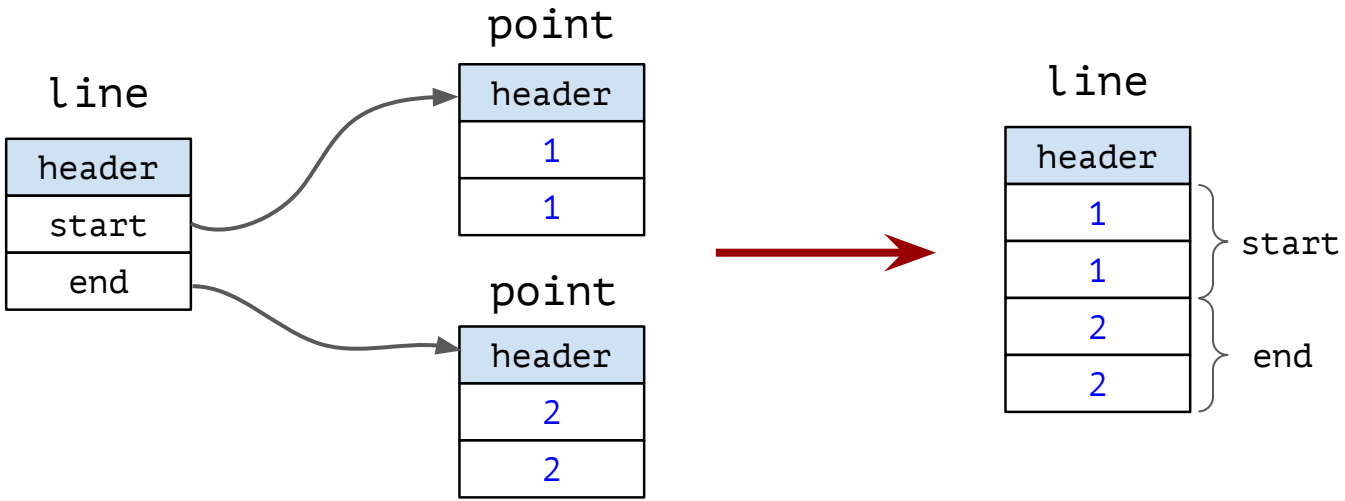
Вдруг решили синхронизироваться? А у вас объект на регистрах!

Почему здесь начались аллокации?

Почему не передать поля Point на регистрах в метод Print?

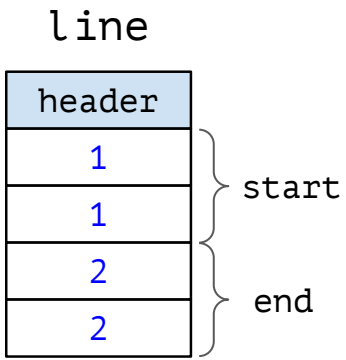
Наводящие вопросы #2

Q: почему бы не разрешить переходить вот к такой (c-struct like) плоской раскладке?



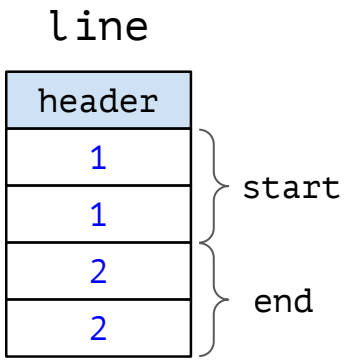
Наводящие вопросы #2

```
record Point(int x, int y) {}
record Line(Point start, Point end) {}

var line = new Line(new Point(1,1), new Point(2,2));
```



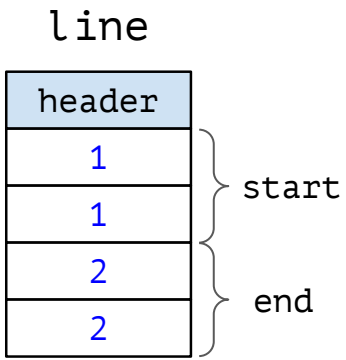
Наводящие вопросы #2

```
record Point(int x, int y) {}
record Line(Point start, Point end) {}

static Point cachedPoint;

var line = new Line(new Point(1,1), new Point(2,2));
```



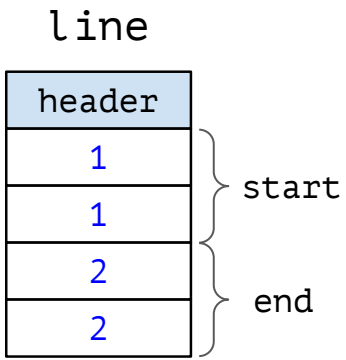
Наводящие вопросы #2

```
record Point(int x, int y) {}
record Line(Point start, Point end) {}

static Point cachedPoint;

var line = new Line(new Point(1,1), new Point(2,2));
cachedPoint = line.start;
```



Наводящие вопросы #2

```
record Point(int x, int y) {}
record Line(Point start, Point end) {}

static Point cachedPoint;

var line = new Line(new Point(1,1), new Point(2,2));
cachedPoint = line.start;
assert cachedPoint == line.start;
```



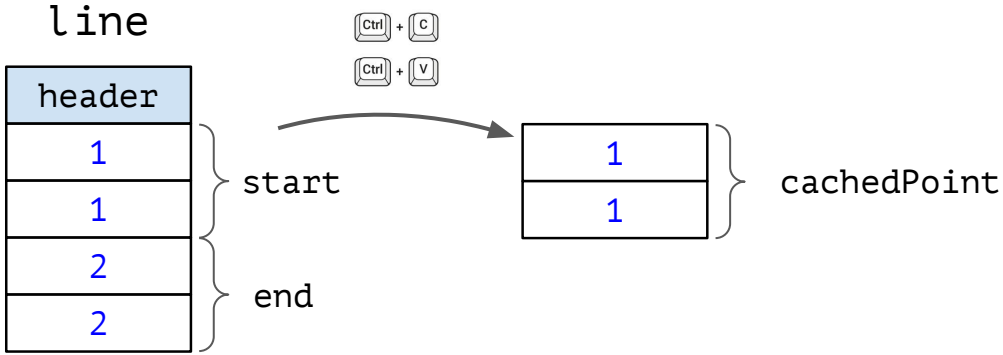
Наводящие вопросы #2

```
record Point(int x, int y) {}
record Line(Point start, Point end) {}

static Point cachedPoint;

var line = new Line(new Point(1,1), new Point(2,2));
cachedPoint = line.start;
assert cachedPoint == line.start;
```

Чтобы это сработало на объекте с identity, придется хранить в cachedPoint ссылку внутрь line (беды с GC)

Наводящие вопросы #2

```
record Point(int x, int y) {}
record Line(Point start, Point end) {}

static Point cachedPoint;

var line = new Line(new Point(1,1), new Point(2,2));
cachedPoint = line.start;
assert cachedPoint == line.start;
```



Чтобы это сработало на объекте с identity, придется хранить в cachedPoint ссылку внутрь line (беды с GC)

а если identity нет…

никто не заметит, что что-то не так, если мы просто скопируем 😈

Identity

Что такое identity в контексте языков программирования?

Identity - это возможность отличить объекты друг от друга, даже если их содержимое побитово совпадает.

…

А что позволило бы делать отсутствие identity?

Identity

Что такое identity в контексте языков программирования?

Identity - это возможность отличить объекты друг от друга, даже если их содержимое побитово совпадает.

…

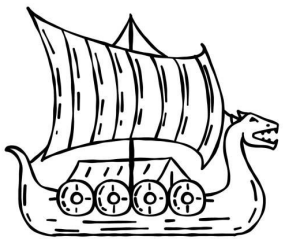
А что позволило бы делать отсутствие identity?

Не думать о том, где объект расположен!
JVM могла бы копировать его поля, как ей заблагорассудится!

Путь в Вальхаллу

1. Вводится новый тип классов - **value** классы

```
value record Point(int x, int y) {}

value class Date {
    ...
}
```

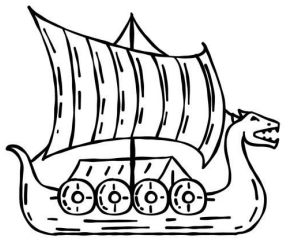


Путь в Вальхаллу

1. Вводится новый тип классов - **value** классы

```
value record Point(int x, int y) {}
value class Date { ... }
```

2. Ограничения value классов:
   a. Все поля таких классов становятся implicitly final,
   b. На объектах таких классов нельзя синхронизироваться,
   c. == сравнивает по содержимому, а не по ссылке!

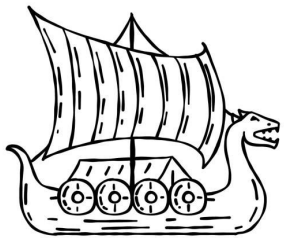


Путь в Вальхаллу

1. Вводится новый тип классов - **value** классы

```
value record Point(int x, int y) {}
value class Date { ... }
```

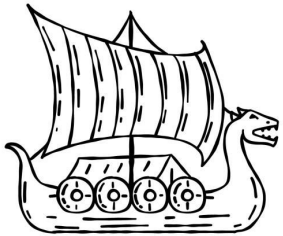
2. Ограничения value классов:
   a. Все поля таких классов становятся implicitly final,
   b. На объектах таких классов нельзя синхронизироваться,
   c. == сравнивает по содержимому, а не по ссылке!
   d. еще есть ограничения на наследование (не абстрактные value классы финальны) и инициализацию, про нее позже;



Путь в Вальхаллу

3. Перед JVM открывается огромный простор для оптимизаций:

    a. Calling conventions/scalarization: маленькие* валуи передаются в и возвращаются из функций на регистрах



Наводящие вопросы #1

```
class Point { … }
Point answer = new Point(42, 42);

static void print(Point p) {
    if (p == answer) { print("answer!"); }
    ...
}

int getResult(int n) {
    int result = 0;
    for (int i = 0; i < n; i++) {
        var p = new Point(i, i);
        print(p);
        result = p.x + p.y;
    }
    return result;
}
```

Вдруг там начнут сравнивать по identity? А у вас объект на регистрах!

Почему здесь начались аллокации?

Почему не передать поля Point на регистрах в метод Print?

Наводящие вопросы #1

```
value record Point(int x, int y) { }
Point answer = new Point(42, 42);

static void print(Point p) {
    if (p == answer) { print("answer!"); }
    ...
}

int getResult(int n) {
    int result = 0;
    for (int i = 0; i < n; i++) {
        var p = new Point(i, i);
        print(p);
        result = p.x + p.y;
    }
    return result;
}
```

теперь сравнение по содержимому, т.е. по полям x и y

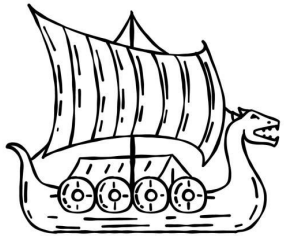
Point можно спокойно скаляризовать, скопировав на регистры при вызове

```
[Verified Inline Entry Point (RO)]
  # this:     rdx:rdx   = 'Test'
  # parm0:    r8        = int
  #           [sp+0x30]  (sp of caller)
  0x00000157912dbed0:   mov    DWORD PTR [rsp-0x8000],eax
  0x00000157912dbed7:   push   rbp
  0x00000157912dbed8:   sub    rsp,0x20
  0x00000157912dbedc:   cmp    DWORD PTR [r15+0x20],0x0
  0x00000157912dbee4:   jne    0x00000157912dbf6a           ;*synchronization entry
                                                            ; - Test::getResult@-1 (line 12)
  0x00000157912dbeea:   xor    eax,eax
  0x00000157912dbeec:   test   r8d,r8d
  0x00000157912dbeef:   jle    0x00000157912dbf34           ;*if_icmpge {reexecute=0 rethrow=0 return_oop=0 return_scalarized=0}
                                                            ; - Test::getResult@6 (line 13)
  0x00000157912dbef1:   xor    r10d,r10d
  0x00000157912dbef4:   mov    DWORD PTR [rsp+0x4],r8d
  0x00000157912dbef9:   nop    DWORD PTR [rax+0x0]          ;*new {reexecute=0 rethrow=0 return_oop=0 return_scalarized=0}
                                                            ; - Test::getResult@9 (line 14)
  0x00000157912dbf00:   mov    ebp,r10d
  0x00000157912dbf03:   mov    DWORD PTR [rsp],r10d
  0x00000157912dbf07:   add    ebp,r10d                     ;*iadd {reexecute=0 rethrow=0 return_oop=0 return_scalarized=0}
                                                            ; - Test::getResult@30 (line 15)
  0x00000157912dbf0a:   mov    edx,0x1
  0x00000157912dbf0f:   mov    r8d,r10d
  0x00000157912dbf12:   mov    r9d,r10d
  0x00000157912dbf15:   xchg   ax,ax
  0x00000157912dbf17:   call   0x0000015790b63000          ; ImmutableOopMap {}
                                                            ;*invokestatic print  {reexecute=0 rethrow=0 return_oop=0 return_scalarized=0}
                                                            ; - Test::getResult@34 (line 16)
                                                            ;   {static_call Test::print}
  0x00000157912dbf1c:   nop    DWORD PTR [rax+rax*1+0x20c]  ;*invokestatic print {reexecute=0 rethrow=0 return_oop=0 return_scalarized=0}
                                                            ; - Test::getResult@34 (line 16)
                                                            ;   {other}
  0x00000157912dbf24:   mov    r10d,DWORD PTR [rsp]
  0x00000157912dbf28:   inc    r10d                         ;*iinc {reexecute=0 rethrow=0 return_oop=0 return_scalarized=0}
                                                            ; - Test::getResult@37 (line 13)
  0x00000157912dbf2b:   cmp    r10d,DWORD PTR [rsp+0x4]
  0x00000157912dbf30:   jl     0x00000157912dbf00
  0x00000157912dbf32:   mov    eax,ebp                      ;*if_icmpge {reexecute=0 rethrow=0 return_oop=0 return_scalarized=0}
                                                            ; - Test::getResult@6 (line 13)
  0x00000157912dbf34:   add    rsp,0x20
  0x00000157912dbf38:   pop    rbp
  0x00000157912dbf39:   cmp    rsp,QWORD PTR [r15+0x460]    ;   {poll_return}
  0x00000157912dbf40:   ja     0x00000157912dbf54
  0x00000157912dbf46:   ret                                 ;*invokestatic print {reexecute=0 rethrow=0 return_oop=0 return_scalarized=0}
                                                            ; - Test::getResult@34 (line 16)
```

Код стал чуть менее оптимизированный, но аллокаций никаких нет (а копирование на регистры – есть)

Путь в Вальхаллу

3. Перед JVM открывается огромный простор для оптимизаций:
    a. Calling conventions/scalarization: маленькие* валуи передаются в и возвращаются из функций на регистрах

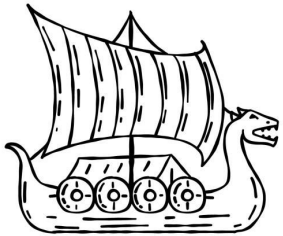


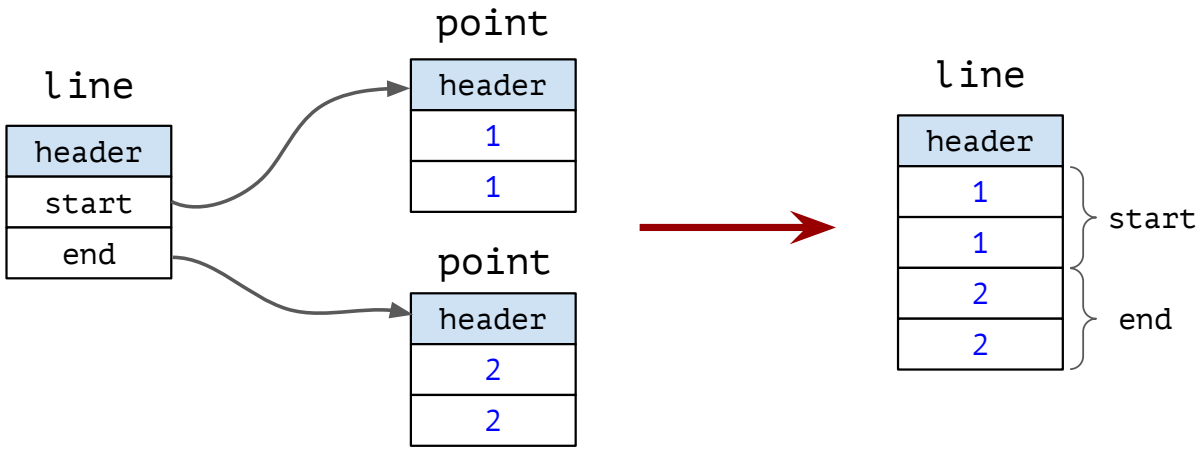
Путь в Вальхаллу

3. Перед JVM открывается огромный простор для оптимизаций:

   a. Calling conventions/scalarization: маленькие* валуи передаются в и возвращаются из функций на регистрах

   b. Heap-flattening: маленькие* валуи инлайнятся** в поля других объектов и массивов





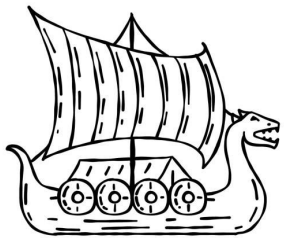
Путь в Вальхаллу

3. Перед JVM открывается огромный простор для оптимизаций:

    a. Calling conventions/scalarization: маленькие* валуи передаются в и возвращаются из функций на регистрах

    b. Heap-flattening: маленькие* валуи инлайнятся** в поля других объектов и массивов

    c. value классы - все еще наследники j.l.Object



Путь в Вальхаллу

3. Перед JVM открывается огромный простор для оптимизаций:

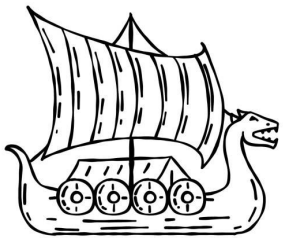
   a. Calling conventions/scalarization: маленькие* валуи передаются в и возвращаются из функций на регистрах

   b. Heap-flattening: маленькие* валуи инлайнятся** в поля других объектов и массивов

   c. value классы - все еще наследники j.l.Object

   d. Многие классы стандартной библиотеки становятся*** валуями (уменьшаем великий разлом):

      - java.lang.Integer, Float, …
      - java.time: Duration, Instant, LocalTime, …
      - java.util.Optional, …



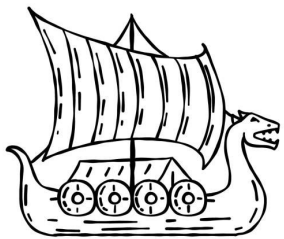
Путь в Вальхаллу

1. Вводится новый тип классов - **value** классы

2. На value классы ограничения:
   a. implicitly final поля,
   b. синхронизироваться нельзя,
   c. == сравнивает по содержимому, а не по ссылке!
   d. ограниченное наследование и инициализация

3. JVM начинает агрессивно скаляризовывать
   и инлайнить их в другие объекты 💪

4. Переводим на value, что можем



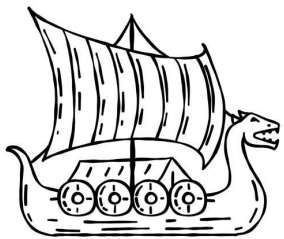
Путь в Вальхаллу

1. Вводится новый тип классов - **value** классы

2. На value классы ограничения:
   a. implicitly final поля,
   b. синхронизироваться нельзя,
   c. == сравнивает по содержимому, а не по ссылке!
   d. ограниченное наследование и инициализация

3. JVM начинает агрессивно скаляризовывать и инлайнить их в другие объекты 💪

4. Переводим на value, что можем



Вроде не очень сложно, почему этот проект делают с 2014 года?

Стражи Вальхаллы

Стражи Вальхаллы




Стражи Вальхаллы

Обратная совместимость

Сложные вопросы: обратная совместимость

1. Вводится новый тип классов - **value** классы

```
value record Point(int x, int y) {}
value class Date { ... }
```

2. Ограничения value классов:
   a. …
   b. На объектах таких классов нельзя синхронизироваться,
   c. == сравнивает по содержимому, а не по ссылке!
   d. ...

3. Не всегда очевидные моменты:
   a. это все еще наследники j.l.Object
   b. …

Сложные вопросы: обратная совместимость

1. Вводится новый тип классов - **value** классы

```
value record Point(int x, int y) {}
value class Date { ... }
```

2. Ограничения value классов:
   a. …
   b. На объектах таких классов нельзя синхронизироваться,
   c. == сравнивает по содержимому, а не по ссылке!
   d. ...

3. Не всегда очевидные моменты:
   a. это все еще наследники j.l.Object
   b. …

Это круто иметь великого праотца – j.l.Object, но…

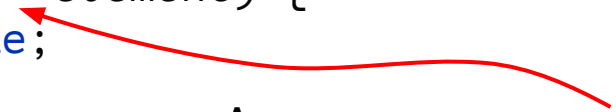
это ведь означает проверки в рантайме: валуй или не валуй!

Сложные вопросы: обратная совместимость

```
boolean contains(Object[] array, Object element) {
    for (int i = 0; i < array.length; i++) {
        if (array[i] == element) {
            return true;
        }
    }
    return false;
}
```

Сложные вопросы: обратная совместимость

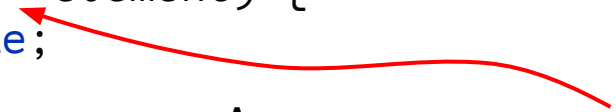
```
boolean contains(Object[] array, Object element) {
    for (int i = 0; i < array.length; i++) {
        if (array[i] == element) {
            return true;
        }
    }
    return false;
}
```



А что это за ==?
По ссылке или по значению?

Сложные вопросы: обратная совместимость

```
boolean contains(Object[] array, Object element) {
    for (int i = 0; i < array.length; i++) {
        if (array[i] == element) {
            return true;
        }
    }
    return false;
}
```

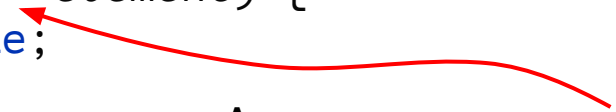


А что это за ==?
По ссылке или по значению?

Раньше это всегда было сравнение указателей, но теперь… зависит от array и от element (нельзя сравнивать value типы по указателю!)

Сложные вопросы: обратная совместимость

```
boolean contains(Object[] array, Object element) {
    for (int i = 0; i < array.length; i++) {
        if (array[i] == element) {
            return true;
        }
    }
    return false;
}
```



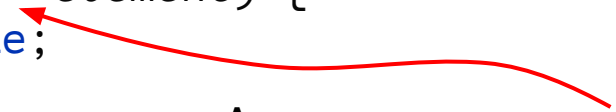
А что это за ==?
По ссылке или по значению?

…зависит от array и от element

получается, что поведение (и производительность!) изменились по сравнению с прошлыми Java

Сложные вопросы: обратная совместимость

```
boolean contains(Object[] array, Object element) {
    for (int i = 0; i < array.length; i++) {
        if (array[i] == element) {
            return true;
        }
    }
    return false;
}
```



А что это за ==?
По ссылке или по значению?

…зависит от array и от element

получается, что поведение (и производительность!) изменились по сравнению с прошлыми Java

и ладно производительность…

Сложные вопросы: обратная совместимость

```
boolean contains(Object[] array, Object element, Object lock) {
    synchronized(lock) {
        for (int i = 0; i < array.length; i++) {
            if (array[i] == element) {
                return true;
            }
        }
        return false;
    }
}
```

Сложные вопросы: обратная совместимость

```
boolean contains(Object[] array, Object element, Object lock) {
    synchronized(lock) {
        for (int i = 0; i < array.length; i++) {
            if (array[i] == element) {
                return true;
            }
        }
        return false;
    }
}

contains(array, el, Integer.valueOf(42));
```

Сложные вопросы: обратная совместимость

```
boolean contains(Object[] array, Object element, Object lock) {
    synchronized(lock) {
        for (int i = 0; i < array.length; i++) {
            if (array[i] == element) {
                return true;
            }
        }
        return false;
    }
}

contains(array, el, Integer.valueOf(42));

Java 26                    -> "works" fine
```

Сложные вопросы: обратная совместимость

```
boolean contains(Object[] array, Object element, Object lock) {
    synchronized(lock) {
        for (int i = 0; i < array.length; i++) {
            if (array[i] == element) {
                return true;
            }
        }
        return false;
    }
}

contains(array, el, Integer.valueOf(42));

Java 26                      -> "works" fine
Java 27 --enable preview -> Exception in thread "main" j.l.IdentityException:
                Cannot synchronize on an instance of value class java.lang.Integer
```

Путь в Вальхаллу

3. Перед JVM открывается огромный простор для оптимизаций:

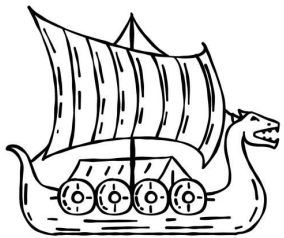
   a. Calling conventions/scalarization: маленькие* валуи передаются в и возвращаются из функций на регистрах

   b. Heap-flattening: маленькие* валуи инлайнятся** в поля других объектов и массивов

   c. value классы - все еще наследники j.l.Object

   d. Многие классы стандартной библиотеки становятся*** валуями (уменьшаем великий разлом):

      - java.lang.Integer, Float, …
      - java.time: Duration, Instant, LocalTime, …
      - java.util.Optional, …



Сложные вопросы: обратная совместимость

```
boolean contains(Object[] array, Object element, Object lock) {
    synchronized(lock) {
        for (int i = 0; i < array.length; i++) {
            if (array[i] == element) {
                return true;
            }
        }
        return false;
    }
}

contains(array, el, Integer.valueOf(42));

Java 26                     -> "works" fine
Java 27 --enable preview -> Exception in thread "main" j.l.IdentityException:
                Cannot synchronize on an instance of value class java.lang.Integer
```

Сложные вопросы: обратная совместимость

```
boolean contains(Object[] array, Object element, Object lock) {
    synchronized(lock) {
        for (int i = 0; i < array.length; i++) {
            if (array[i] == element) {
                return true;
            }
        }
        return false;
    }
}

contains(array, el, Integer.valueOf(42));
```


```
Java 26                      -> "works" fine
Java 27 --enable preview -> Exception in thread "main" j.l.IdentityException:
                Cannot synchronize on an instance of value class java.lang.Integer
```

Сложные вопросы: обратная совместимость

```
boolean contains(Object[] array, Object element, Object lock) {
    synchronized(lock) {
        for (int i = 0; i < array.length; i++) {
            if (array[i] == element) {
                return true;
            }
        }
        return false;
    }
}

contains(array, el, Integer.valueOf(42));
```

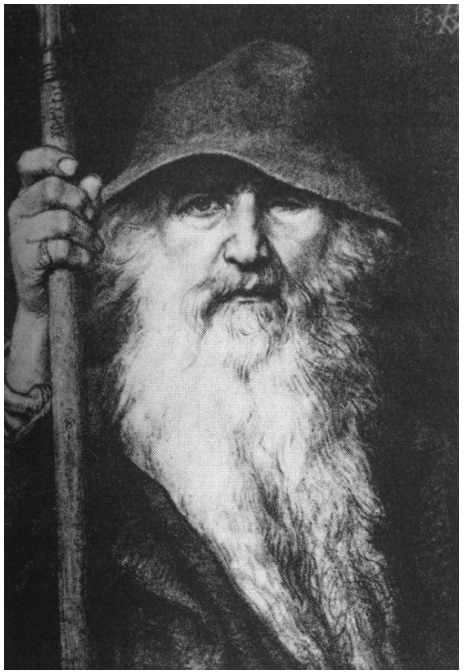
Конечно же, так и раньше не стоило делать (кэширование Integer-ов), но тем не менее!

```
Java 26                    -> "works" fine
Java 27 --enable preview -> Exception in thread "main" j.l.IdentityException:
                Cannot synchronize on an instance of value class java.lang.Integer
```

Сложные вопросы: обратная совместимость

Встраивание в общую систему типов:

- ➕ Позволяет писать обобщенный код для `reference` и `value` типов
- ➕ Много старого кода заработает лучше (быстрее) из коробки, если просто добавить классу `value`



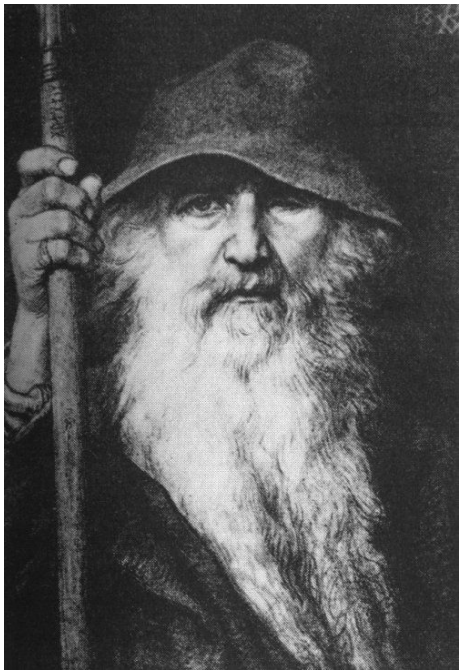
Сложные вопросы: обратная совместимость

Встраивание в общую систему типов:

Позволяет писать обобщенный код для reference и value типов

Много старого кода заработает лучше (быстрее) из коробки, если просто добавить классу value

Но цена этому: проверки в рантайме (значит старый код может замедлиться) и даже слом обратной совместимости (пусть и для не самого хорошего кода)



Стражи Вальхаллы

Обратная совместимость

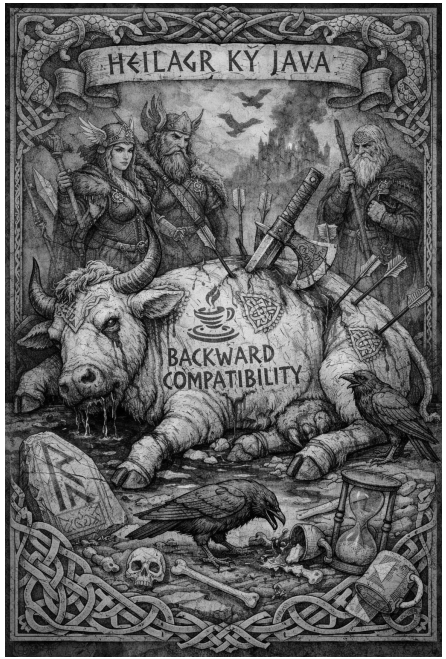
Стражи Вальхаллы

~~Обратная~~

~~совместимость~~

Runtime checks

Стражи Вальхаллы



~~Обратная совместимость~~

Runtime checks

Инициализация

Сложные вопросы: инициализация

Как известно, в Java есть проблемы с инициализацией объектов.

```
class Person {
   String name;
   int age;

   void print() {
       System.out.println("Person: name = " +
                          name + "; age = " +
                          age);
   }

   Person(String name, int age) {
       this.name = name;
       this.age = age;
       print();
   }
}
```

```java
class Person {
    String name;
    int age;

    void print() {
        System.out.println("Person: name = " +
                           name + "; age = " +
                           age);
    }

    Person(String name, int age) {
        this.name = name;
        this.age = age;
        print();
    }
}
```

```java
class Student extends Person {
    final int id;

    void print() {
        System.out.println("Student: name = " +
                           name + "; id = " +
                           id);
    }

    Student(String name, int age, int id) {
        super(name, age);
        this.id = id;
    }

}
```

```
class Person {
    String name;
    int age;

    void print() {
        System.out.println("Person: name = " +
                           name + "; age = " +
                           age);
    }

    Person(String name, int age) {
        this.name = name;
        this.age = age;
        print();
    }
}
```

```
class Student extends Person {
    final int id;

    void print() {
        System.out.println("Student: name = " +
                           name + "; id = " +
                           id);
    }

    Student(String name, int age, int id) {
        super(name, age);
        this.id = id;
    }

}
```

```
void main() {
    var s = new Student("Ivanov", 18, 421414);
    s.print();
}
```

```
class Person {
    String name;
    int age;

    void print() {
        System.out.println("Person: name = " +
                           name + "; age = " +
                           age);
    }

    Person(String name, int age) {
        this.name = name;
        this.age = age;
        print();
    }
}
```

```
class Student extends Person {
    final int id;

    void print() {
        System.out.println("Student: name = " +
                           name + "; id = " +
                           id);
    }

    Student(String name, int age, int id) {
        super(name, age);
        this.id = id;
    }

}
```

```
void main() {
    var s = new Student("Ivanov", 18, 421414);
    s.print();
}
```

```
Student: name = Ivanov; id = 0
```

```
class Person {
    String name;
    int age;

    void print() {
        System.out.println("Person: name = " +
                           name + "; age = " +
                           age);
    }

    Person(String name, int age) {
        this.name = name;
        this.age = age;
        print();
    }
}
```

```
class Student extends Person {
    final int id;

    void print() {
        System.out.println("Student: name = " +
                           name + "; id = " +
                           id);
    }

    Student(String name, int age, int id) {
        super(name, age);
        this.id = id;
    }

}
```

```
void main() {
    var s = new Student("Ivanov", 18, 421414);
    s.print();
}
```

```
Student: name = Ivanov; id = 0
Student: name = Ivanov; id = 421414
```

```java
class Person {
    String name;
    int age;

    void print() {
        System.out.println("Person: name = " +
                           name + "; age = " +
                           age);
    }

    Person(String name, int age) {
        this.name = name;
        this.age = age;
        print();
    }
}
```

```java
class Student extends Person {
    final int id;

    void print() {
        System.out.println("Student: name = " +
                           name + "; id = " +
                           id);
    }

    Student(String name, int age, int id) {
        super(name, age);
        this.id = id;
    }

}
```

```java
void main() {
    var s = new Student("Ivanov", 18, 421414);
    s.print();
}
```

```
Student: name = Ivanov; id = 0
Student: name = Ivanov; id = 421414
```

final поле поменялось, удобно!

Сложные вопросы: инициализация

Как известно, в Java есть проблемы с инициализацией объектов:

1. Т.к. super() выполняется первее тела наследника, то он может пронаблюдать непроинициализированные еще поля

Сложные вопросы: инициализация

Как известно, в Java есть проблемы с инициализацией объектов:

1. Т.к. super() выполняется первее тела наследника, то он может пронаблюдать непроинициализированные еще поля (кое-как спасает тот факт, что в Java [пока что] все имеет право быть проинициализированно нулем, но от этого не сильно легче)

Сложные вопросы: инициализация

Как известно, в Java есть проблемы с инициализацией объектов:

1. Т.к. super() выполняется первее тела наследника, то он может пронаблюдать непроинициализированные еще поля (кое-как спасает тот факт, что в Java [пока что] все имеет право быть проинициализированно нулем, но от этого не сильно легче)

2. Да и не только super. А что, если ваш this утек из конструктора до его окончания?

Сложные вопросы: инициализация

Как известно, в Java есть проблемы с инициализацией объектов:

1. Т.к. super() выполняется первее тела наследника, то он может пронаблюдать непроинициализированные еще поля (кое-как спасает тот факт, что в Java [пока что] все имеет право быть проинициализированно нулем, но от этого не сильно легче)

2. Да и не только super. А что, если ваш this утек из конструктора до его окончания? Кто угодно может пронаблюдать экземпляр класса в невалидном состоянии

Сложные вопросы: инициализация

Как известно, в Java есть проблемы с инициализацией объектов:

1. Т.к. super() выполняется первее тела наследника, то он может пронаблюдать непроинициализированные еще поля (кое-как спасает тот факт, что в Java [пока что] все имеет право быть проинициализированно нулем, но от этого не сильно легче)

2. Да и не только super. А что, если ваш this утек из конструктора до его окончания? Кто угодно может пронаблюдать экземпляр класса в невалидном состоянии

А как же быть с value классами? Что, если их кто-то увидит в невалидном состоянии?

Сложные вопросы: инициализация

Как известно, в Java есть проблемы с инициализацией объектов:

1. Т.к. super() выполняется первее тела наследника, то он может пронаблюдать непроинициализированные еще поля (кое-как спасает тот факт, что в Java [пока что] все имеет право быть проинициализированно нулем, но от этого не сильно легче)

2. Да и не только super. А что, если ваш this утек из конструктора до его окончания? Кто угодно может пронаблюдать экземпляр класса в невалидном состоянии

А как же быть с value классами? Что, если их кто-то увидит в невалидном состоянии? (они ведь иммутабельные! И == сравнивает их по содержимому! все инварианты сломаются, все гораздо хуже, чем с final)

Сложные вопросы: инициализация

Проблема #1: никто не должен видеть не до конца проинициализированные экземпляры `value` классов

Как это гарантировать?

Сложные вопросы: инициализация

Проблема #1: никто не должен видеть не до конца проинициализированные экземпляры value классов

Как это гарантировать?

Старое решение (до 2024 года):

1. Ограничить наследование value классов: разрешить наследовать только тех, у кого тривиальный конструктор

Сложные вопросы: инициализация

Проблема #1: никто не должен видеть не до конца проинициализированные экземпляры value классов

Как это гарантировать?

Старое решение (до 2024 года):

1. Ограничить наследование value классов: разрешить наследовать только тех, у кого тривиальный конструктор

2. Для создания валуев использовать новый набор байткодных инструкций и специальные сгенерированные статические fabric методы

Сложные вопросы: инициализация

Проблема #1: никто не должен видеть не до конца проинициализированные экземпляры `value` классов

Как это гарантировать?

Старое решение (до 2024 года):

```
value record Point(int x, int y) { }
```

```
static <vnew>(II):LPoint; {
    aconst_init class Point;
    iload_0;
    withfield Point.x:I;
    iload_1;
    withfield Point.y:I;
    areturn;
}
```

статический метод вместо <init>, возвращает сконструированный и корректный объект

Сложные вопросы: инициализация

Проблема #1: никто не должен видеть не до конца проинициализированные экземпляры value классов

Как это гарантировать?

Старое решение (до 2024 года):

```
value record Point(int x, int y) { }
```

```
static <vnew>(II):LPoint; {
    aconst_init class Point;
    iload_0;
    withfield Point.x:I;
    iload_1;
    withfield Point.y:I;
    areturn;
}
```

новая инструкция: создает (корректный) экземпляр value класса, проинициализированный "нулями"

Сложные вопросы: инициализация

Проблема #1: никто не должен видеть не до конца проинициализированные экземпляры `value` классов

Как это гарантировать?

Старое решение (до 2024 года):

```
value record Point(int x, int y) { }
```

```
static <vnew>(II):LPoint; {
    aconst_init class Point;
    iload_0;
    withfield Point.x:I;
    iload_1;
    withfield Point.y:I;
    areturn;
}
```

новые инструкции: создает новый (корректный) экземпляр value класса, копируя предыдущий, но добавляя к нему одно поле

Сложные вопросы: инициализация

Проблема #1: никто не должен видеть не до конца проинициализированные экземпляры `value` классов

Как это гарантировать?

Старое решение (до 2024 года):

```
value record Point(int x, int y) { }
```

```
static <vnew>(II):LPoint; {
    aconst_init class Point;
    iload_0;
    withfield Point.x:I;
    iload_1;
    withfield Point.y:I;
    areturn;
}
```

И никаких вызовов super! (т.к. тривиальные)

новые инструкции: создает новый (корректный) экземпляр value класса, копируя предыдущий, но добавляя к нему одно поле

Сложные вопросы: инициализация

Проблема #1: никто не должен видеть не до конца проинициализированные экземпляры value классов

Как это гарантировать?

Старое решение (до 2024 года):

Плюсы:

✓ ну, в принципе работает, никто не увидит утекший и не проинициализированный instance (может его копию)

Сложные вопросы: инициализация

Проблема #1: никто не должен видеть не до конца проинициализированные экземпляры value классов

Как это гарантировать?

Старое решение (до 2024 года):

Минусы:

✓ Жесткие ограничения на наследование 😖

Сложные вопросы: инициализация

Проблема #1: никто не должен видеть не до конца проинициализированные экземпляры `value` классов

Как это гарантировать?

Старое решение (до 2024 года):

Минусы:

- ✓ Жесткие ограничения на наследование 😖
- ✓ Параллельный механизм инициализации для `value` 🤯

Сложные вопросы: инициализация

Проблема #1: никто не должен видеть не до конца проинициализированные экземпляры value классов

Как это гарантировать?

Старое решение (до 2024 года):

Минусы:

- ✓ Жесткие ограничения на наследование 😖
- ✓ Параллельный механизм инициализации для value 🤯
- ✓ Все равно же могут утечь пусть и корректные, но еще по семантике не проинициализированные до конца валуи!

Сложные вопросы: инициализация

Проблема #1: никто не должен видеть не до конца проинициализированные экземпляры `value` классов

Как это гарантировать?

Новое решение (с 2024 года):

Сложные вопросы: инициализация

Проблема #1: никто не должен видеть не до конца
проинициализированные экземпляры value классов

Как это гарантировать?

Новое решение (с 2024 года):

Вообще, ситуация с изменяемыми final-ами тоже никому не нравится.
По этому поводу есть интересный JEP-513: Flexible Constructor Bodies

Сложные вопросы: инициализация

Вообще, ситуация с изменяемыми final-ами тоже никому не нравится.
По этому поводу есть интересный JEP-513: Flexible Constructor Bodies

Идея: давайте писать код в конструкторе ДО вызова super()...

Сложные вопросы: инициализация

Вообще, ситуация с изменяемыми final-ами тоже никому не нравится.
По этому поводу есть интересный JEP-513: Flexible Constructor Bodies

Идея: давайте писать код в конструкторе ДО вызова super()... но при этом запретим использовать до этого вызова this (кроме как для выставления полей)

```
class Person {
    String name;
    int age;

    void print() {
        System.out.println("Person: name = " +
                           name + "; age = " +
                           age);
    }

    Person(String name, int age) {
        this.name = name;
        this.age = age;
        print();
    }
}
```

```
class Student extends Person {
    final int id;

    void print() {
        System.out.println("Student: name = " +
                           name + "; id = " +
                           id);
    }

    Student(String name, int age, int id) {
        super(name, age);
        this.id = id;
    }

}
```

```
void main() {
    var s = new Student("Ivanov", 18, 421414);
    s.print();
}
```

```
Student: name = Ivanov; id = 0
Student: name = Ivanov; id = 421414
```

final поле поменялось, удобно!

```
class Person {
    String name;
    int age;

    void print() {
        System.out.println("Person: name = " +
                           name + "; age = " +
                           age);
    }

    Person(String name, int age) {
        this.name = name;
        this.age = age;
        print();
    }
}
```

```
class Student extends Person {
    final int id;

    void print() {
        System.out.println("Student: name = " +
                           name + "; id = " +
                           id);
    }

    Student(String name, int age, int id) {
        this.id = id;
        super(name, age);
    }

}
```

```
void main() {
    var s = new Student("Ivanov", 18, 421414);
    s.print();
}
```

```
Student: name = Ivanov; id = 421414
Student: name = Ivanov; id = 421414
```

теперь все хорошо 😀

```
class Person {
    String name;
    int age;

    void print() {
        System.out.println("Person: name = " +
                           name + "; age = " +
                           age);
    }

    Person(String name, int age) {
        this.name = name;
        this.age = age;
        print();
    }
}
```

```
class Student extends Person {
    final int id;

    void print() {
        System.out.println("Student: name = " +
                           name + "; id = " +
                           id);
    }

    Student(String name, int age, int id) {
        System.out.print(this);
        this.id = id;
        super(name, age);
    }
}
```

error: cannot reference this before supertype constructor has been called

```
void main() {
    var s = new Student("Ivanov", 18, 421414);
    s.print();
}
```

```
Student: name = Ivanov; id = 421414
Student: name = Ivanov; id = 421414
```

теперь все хорошо 😀

(ну, может чуть-чуть неудобно)

```
class Person {
    String name;
    int age;

    void print() {
        System.out.println("Person: name = " +
                           name + "; age = " +
                           age);
    }

    Person(String name, int age) {
        this.name = name;
        this.age = age;
        print();
    }
}
```

```
class Student extends Person {
    final int id;

    void print() {
        System.out.println("Student: name = " +
                           name + "; id = " +
                           id);
    }

    Student(String name, int id) {
        System.out.print(age);
        this.id = id;
        super(name, 18);
    }
}
```

error: cannot reference age before supertype constructor has been called

```
void main() {
    var s = new Student("Ivanov", 421414);
    s.print();
}
```

```
Student: name = Ivanov; id = 421414
Student: name = Ivanov; id = 421414
```

теперь все хорошо 😀

(ну, может чуть-чуть неудобно)

```
class Person {
    String name;
    int age;

    void print() {
        System.out.println("Person: name = " +
                           name + "; age = " +
                           age);
    }

    Person(String name, int age) {
        this.name = name;
        this.age = age;
        print();
    }
}
```

```
class Student extends Person {
    final int id;

    void print() {
        System.out.println("Student: name = " +
                           name + "; id = " +
                           id);
    }

    Student(String name, int id) {
        print();
        this.id = id;
        super(name, 18);
    }
}
```

error: cannot reference print() before supertype constructor has been called

```
void main() {
    var s = new Student("Ivanov", 421414);
    s.print();
}
```

```
Student: name = Ivanov; id = 421414
Student: name = Ivanov; id = 421414
```

теперь все хорошо 😀

(ну, может чуть-чуть неудобно)

Сложные вопросы: инициализация

Вообще, ситуация с изменяемыми final-ами тоже никому не нравится.
По этому поводу есть интересный JEP-513: Flexible Constructor Bodies

Идея: давайте писать код в конструкторе ДО вызова super()... но при этом запретим использовать до этого вызова this (кроме как для выставления полей)

Сложные вопросы: инициализация

Вообще, ситуация с изменяемыми final-ами тоже никому не нравится. По этому поводу есть интересный JEP-513: Flexible Constructor Bodies

Идея: давайте писать код в конструкторе ДО вызова super()... но при этом запретим использовать до этого вызова this (кроме как для выставления полей)

С одной стороны, это позволяет проинициализировать поля до super(), избавившись от проблемы. С другой - это относительно безопасно, до вызова super() this никуда не утечет.

Сложные вопросы: инициализация

Вообще, ситуация с изменяемыми final-ами тоже никому не нравится. По этому поводу есть интересный JEP-513: Flexible Constructor Bodies

Идея: давайте писать код в конструкторе ДО вызова super()... но при этом запретим использовать до этого вызова this (кроме как для выставления полей)

С одной стороны, это позволяет проинициализировать поля до super(), избавившись от проблемы. С другой - это относительно безопасно, до вызова super() this никуда не утечет.

И реализуется легко - чисто на уровне javac!

Сложные вопросы: инициализация

Вообще, ситуация с изменяемыми final-ами тоже никому не нравится.
По этому поводу есть интересный JEP-513: Flexible Constructor Bodies

Идея: давайте писать код в конструкторе ДО вызова super()... но при этом запретим использовать до этого вызова this (кроме как для выставления полей)

С одной стороны, это позволяет проинициализировать поля до super(), избавившись от проблемы. С другой - это относительно безопасно, до вызова super() this никуда не утечет.

И реализуется легко - чисто на уровне javac!

Status: Delivered. Но как это связано с value классами?

Сложные вопросы: инициализация

Вообще, ситуация с изменяемыми final-ами тоже никому не нравится. По этому поводу есть интересный JEP-513: Flexible Constructor Bodies

Идея: давайте писать код в конструкторе ДО вызова super()... но при этом запретим использовать до этого вызова this (кроме как для выставления полей)

С одной стороны, это позволяет проинициализировать поля до super(), избавившись от проблемы. С другой - это относительно безопасно, до вызова super() this никуда не утечет.

И реализуется легко - чисто на уровне javac!

Status: Delivered. Но как это связано с value классами?

Всё просто: **все поля** value классов обязаны инициализироваться до вызова super() по общим правилам. Называется strictly initialized.

```
abstract value class Point {
    int x, y;

    Point(int x, int y) {
        this.x = x;
        this.y = y;
    }
}
```

```
abstract value class Point {
    int x, y;

    Point(int x, int y) {
        this.x = x;
        this.y = y;
    }
}
```

```
value class ColoredPoint extends Point {
    byte color;

    ColoredPoint(int x, int y, byte color) {
        this.color = color;
        super(x, y);
        System.out.println(this);
    }
}
```

```
abstract value class Point {
    int x, y;

    Point(int x, int y) {
        this.x = x;
        this.y = y;
    }
}
```

```
value class ColoredPoint extends Point {
    byte color;

    ColoredPoint(int x, int y, byte color) {
        this.color = color;
        super(x, y);
        System.out.println(this);
    }
}
```

Все поля **обязаны** быть проинициализированы до super()!

```
abstract value class Point {
    int x, y;

    Point(int x, int y) {
        this.x = x;
        this.y = y;
    }
}
```

```
value class ColoredPoint extends Point {
    byte color;

    ColoredPoint(int x, int y, byte color) {
        this.color = color;
        super(x, y);
        System.out.println(this);
    }
}
```

Все поля **обязаны** быть проинициализированы до super()!

При этом до super() нельзя пользоваться this

```
abstract value class Point {
    int x, y;

    Point(int x, int y) {
        this.x = x;
        this.y = y;
    }
}
```

```
value class ColoredPoint extends Point {
    byte color;

    ColoredPoint(int x, int y, byte color) {
        this.color = color;
        System.out.println(this);
        super(x, y);
    }
}
```

cannot reference this before supertype constructor has been called

Все поля **обязаны** быть проинициализированы до super()!

При этом до super() нельзя пользоваться this

```
abstract value class Point {
    int x, y;

    Point(int x, int y) {
        this.x = x;
        this.y = y;
    }
}
```

```
value class ColoredPoint extends Point {
    byte color;

    ColoredPoint(int x, int y, byte color) {
        this.color = color;
        super(x, y);
        System.out.println(this);
    }
}
```

Все поля **обязаны** быть проинициализированы до super()!

При этом до super() нельзя пользоваться this (после - ок, уже все сконструировано)

```
abstract value class Point {
    int x, y;

    Point(int x, int y) {
        this.x = x;
        this.y = y;
        System.out.println(this);
    }
}
```

cannot reference this before supertype constructor has been called

```
value class ColoredPoint extends Point {
    byte color;

    ColoredPoint(int x, int y, byte color) {
        this.color = color;
        super(x, y);
        System.out.println(this);
    }
}
```

Все поля **обязаны** быть проинициализированы до super()!

При этом до super() нельзя пользоваться this (после - ок, уже все сконструировано)

```
abstract value class Point {
    int x, y;

    Point(int x, int y) {
        this.x = x;
        this.y = y;
        System.out.println(this);
        // implicit super();
    }
}

value class ColoredPoint extends Point {
    byte color;

    ColoredPoint(int x, int y, byte color) {
        this.color = color;
        super(x, y);
        System.out.println(this);
    }
}
```

cannot reference this before supertype constructor has been called

Все поля **обязаны** быть проинициализированы до super()!

При этом до super() нельзя пользоваться this (после - ок, уже все сконструировано)

Сложные вопросы: инициализация

Проблема #1: никто не должен видеть не до конца проинициализированные экземпляры value классов

Как это гарантировать?

Новое решение (с 2024 года):

Вообще, ситуация с изменяемыми final-ами тоже никому не нравится.
По этому поводу есть интересный JEP-513: Flexible Constructor Bodies

Все поля value классов - strictly initialized => никто никогда больше не сможет увидеть недоинициализированный инстанс value 🎉

Сложные вопросы: инициализация

Проблема #1: никто не должен видеть не до конца проинициализированные экземпляры value классов

Как это гарантировать?

Старое решение (до 2024 года):

```
value record Point(int x, int y) { }
```

```
static <vnew>(II):LPoint; {
    aconst_init class Point;
    iload_0;
    withfield Point.x:I;
    iload_1;
    withfield Point.y:I;
    areturn;
}
```

статический метод вместо <init>, возвращает сконструированный и корркетный объект

Сложные вопросы: инициализация

Проблема #1: никто не должен видеть не до конца проинициализированные экземпляры value классов

Как это гарантировать?

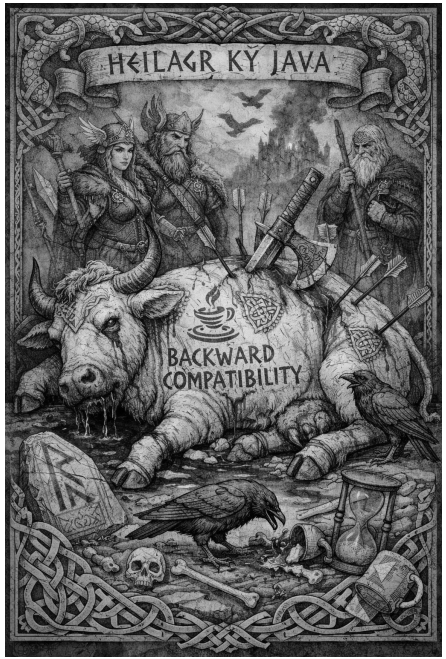
Новое решение (с 2024 года):

Вообще, ситуация с изменяемыми final-ами тоже никому не нравится. По этому поводу есть интересный JEP-513: Flexible Constructor Bodies

Все поля value классов - strictly initialized => никто никогда больше не сможет увидеть недоинициализированный инстанс value 🎉

Намного проще с точки зрение реализации, и аккуратно вписывается в язык, переплетаясь с другими новыми фичами!

Стражи Вальхаллы

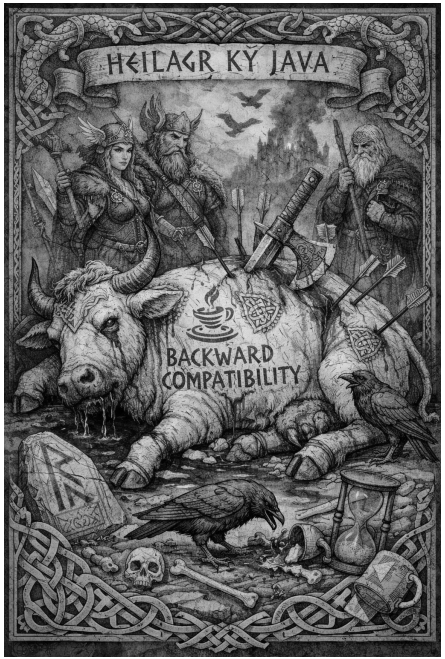


~~Обратная совместимость~~

Runtime checks

Инициализация

Стражи Вальхаллы



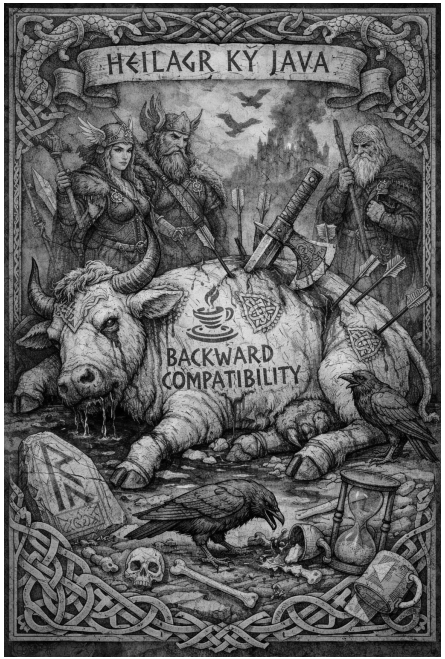
~~Обратная совместимость~~

Runtime checks

~~Инициализация~~

Strictly initialized fields

Стражи Вальхаллы



~~Обратная совместимость~~

Runtime checks

~~Инициализация~~

Strictly initialized fields

Нуллабильность

Сложные вопросы: nullability

Проблема #2: должны ли поля типа value класса (в другом объекте) принимать null, в качестве значения?

Сложные вопросы: nullability

Проблема #2: должны ли поля типа value класса (в другом объекте) принимать null, в качестве значения?

```
value record Point(int x, int y) { }
```

Сложные вопросы: nullability

Проблема #2: должны ли поля типа value класса (в другом объекте) принимать null, в качестве значения?

```
value record Point(int x, int y) { }

class Line {
    Point from;
    Point to;

    Line(int x1, int y1, int x2, int y2) {
        from = new Point(x1, y1);
        to = new Point(x2, y2);
    }
}
```

Сложные вопросы: nullability

Проблема #2: должны ли поля типа value класса (в другом объекте) принимать null, в качестве значения?

```
value record Point(int x, int y) { }

class Line {
    Point from;
    Point to;

    Line(int x2, int y2) {
        from = null;
        to = new Point(x2, y2);
    }

    Line(int x1, int y1, int x2, int y2) {
        from = new Point(x1, y1);
        to = new Point(x2, y2);
    }
}
```

Сложные вопросы: nullability

Проблема #2: должны ли поля типа value класса (в другом объекте) принимать null, в качестве значения?

```
value record Point(int x, int y) { }

class Line {
    Point from;
    Point to;

    Line(int x2, int y2) {
        from = null;
        to = new Point(x2, y2);
    }

    Line(int x1, int y1, int x2, int y2) {
        from = new Point(x1, y1);
        to = new Point(x2, y2);
    }
}
```

норм или стрём?

Сложные вопросы: nullability

Проблема #2: должны ли поля типа value класса (в другом объекте) принимать null, в качестве значения?

```
value record Point(int x, int y) { }

class Line {
    Point from;
    Point to;

    Line(int x2, int y2) {
        from = null;
        to = new Point(x2, y2);
    }

    Line(int x1, int y1, int x2, int y2) {
        from = new Point(x1, y1);
        to = new Point(x2, y2);
    }
}
```

В предыдущих версиях Valhalla это бы просто не скомпилировалось

норм или стрём?

Сложные вопросы: nullability

Проблема #2: должны ли поля типа value класса (в другом объекте, да и вообще) принимать null, в качестве значения?

Старое решение: это стрём. Никаких null-ов в переменных типа value классов. Примитивам же не нужны null-ы? А мы делаем кастомные примитивы.

Сложные вопросы: nullability

Проблема #2: должны ли поля типа value класса (в другом объекте, да и вообще) принимать null, в качестве значения?

Старое решение: это стрём. Никаких null-ов в переменных типа value классов. Примитивам же не нужны null-ы? А мы делаем кастомные примитивы.

Как реализовывали: вводили новый тип дескрипторов для value-классов

Для обычных референс типов: **Ljava/lang/Object;**
Для обычных value типов: **Qmy/value/Test;**

JVM на всех уровнях (javac, верификатор, рантайм) бдит, чтобы в переменную с Q-дескриптором не записали null.

Сложные вопросы: nullability

Проблема #2: должны ли поля типа value класса (в другом объекте, да и вообще) принимать null, в качестве значения?

Старое решение: это стрём. Никаких null-ов в переменных типа value классов. Примитивам же не нужны null-ы? А мы делаем кастомные примитивы.

Как реализовывали: вводили новый тип дескрипторов для value-классов

Для обычных референс типов: **Ljava/lang/Object;**
Для обычных value типов: **Qmy/value/Test;**

JVM на всех уровнях (javac, верификатор, рантайм) бдит, чтобы в переменную с Q-дескриптором не записали null.

Правда у value типов были и L-дескрипторы: **Lmy/value/Test;** 🙄
Все потому, что value типы могли бокситься автоматически (и начинать принимать null)

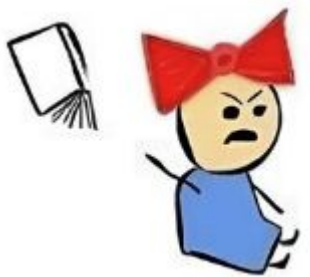
Сложные вопросы: nullability

Проблема #2: должны ли поля типа value класса (в другом объекте, да и вообще) принимать null, в качестве значения?

Старое решение: это стрём. Никаких null-ов в переменных типа value классов. Примитивам же не нужны null-ы? А мы делаем кастомные примитивы.

Как реализовывали: вводили новый тип дескрипторов для value-классов

Как это было: ОЧЕНЬ СЛОЖНО, мало кому нравилось



Сложные вопросы: nullability

Проблема #2: должны ли поля типа value класса (в другом объекте, да и вообще) принимать null, в качестве значения?

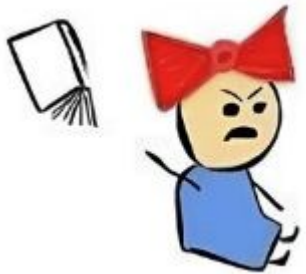
Старое решение: это стрём. Никаких null-ов в переменных типа value классов. Примитивам же не нужны null-ы? А мы делаем кастомные примитивы.

Как реализовывали: вводили новый тип дескрипторов для value-классов

Как это было: ОЧЕНЬ СЛОЖНО, мало кому нравилось, включая архитекторов JVM:

We don't need no stinkin' Q descriptors

Brian Goetz brian.goetz at oracle.com
Fri Jun 30 20:51:43 UTC 2023



Сложные вопросы: nullability

Проблема #2: должны ли поля типа value класса (в другом объекте, да и вообще) принимать null, в качестве значения?

Новое решение: это норм!

Сложные вопросы: nullability

Проблема #2: должны ли поля типа value класса (в другом объекте, да и вообще) принимать null, в качестве значения?

Новое решение: это норм! (ну а что не так? пользователи привыкли работать с классами, и что в переменные можно пихать null, примем этот факт)

Сложные вопросы: nullability

Проблема #2: должны ли поля типа value класса (в другом объекте) принимать null, в качестве значения?

```
value record Point(int x, int y) { }

class Line {
    Point from;
    Point to;

    Line(int x2, int y2) {
        from = null;
        to = new Point(x2, y2);
    }

    Line(int x1, int y1, int x2, int y2) {
        from = new Point(x1, y1);
        to = new Point(x2, y2);
    }
}
```

В предыдущих версиях Valhalla это бы просто не скомпилировалось

норм или стрём?

Сложные вопросы: nullability

Проблема #2: должны ли поля типа value класса (в другом объекте) принимать null, в качестве значения?

```
value record Point(int x, int y) { }

class Line {
    Point from;
    Point to;

    Line(int x2, int y2) {
        from = null;
        to = new Point(x2, y2);
    }

    Line(int x1, int y1, int x2, int y2) {
        from = new Point(x1, y1);
        to = new Point(x2, y2);
    }
}
```

норм или стрём? Норм!

Сложные вопросы: nullability

Проблема #2: должны ли поля типа value класса (в другом объекте) принимать null, в качестве значения?

```
value record Point(int x, int y) { }

class Line {
    Point from;
    Point to;

    Line(int x2, int y2) {
        from = null;
        to = new Point(x2, y2);
    }
}

Line l = new Line(10, 10);
println("l.from == Point(0,0)? " + (l.from == new Point(0,0)));
println("l.from == null)? " + (l.from == null));
```

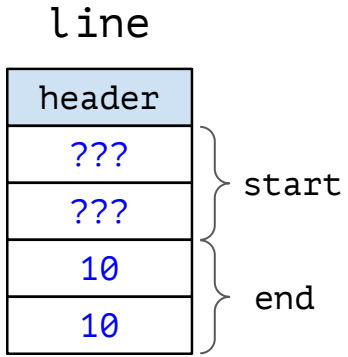
Но тогда придется думать о последствиях!

Сложные вопросы: nullability

Проблема #2: должны ли поля типа value класса (в другом объекте) принимать null, в качестве значения?

```
value record Point(int x, int y) { }

class Line {
    Point from;
    Point to;

    Line(int x2, int y2) {
        from = null;
        to = new Point(x2, y2);
    }
}
```

Но тогда придется думать о последствиях!




```
Line l = new Line(10, 10);
println("l.from == Point(0,0)? " + (l.from == new Point(0,0)));  <- false
println("l.from == null)? " + (l.from == null));                 <- true
```

Сложные вопросы: nullability

Проблема #2: должны ли поля типа value класса (в другом объекте, да и вообще) принимать null, в качестве значения?

Новое решение: это норм! (ну а что не так? пользователи привыкли работать с классами, и что в переменные можно пихать null, примем этот факт)

Последствия: нужно как-то по (потенциально скаляризованному или проинлайненному в другой объект) инстансу value класса понимать, null это или нет.

Сложные вопросы: nullability

Проблема #2: должны ли поля типа value класса (в другом объекте, да и вообще) принимать null, в качестве значения?

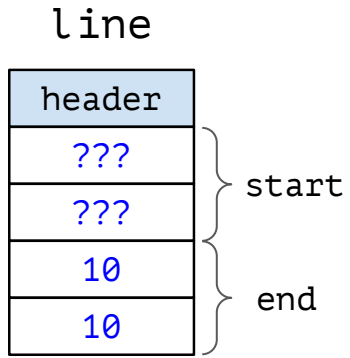
Новое решение: это норм! (ну а что не так? пользователи привыкли работать с классами, и что в переменные можно пихать null, примем этот факт)

Последствия: нужно как-то по (потенциально скаляризованному или проинлайненному в другой объект) инстансу value класса понимать, null это или нет. Придется где-то хранить эту информацию!

Сложные вопросы: nullability

Проблема #2: должны ли поля типа value класса (в другом объекте) принимать null, в качестве значения?

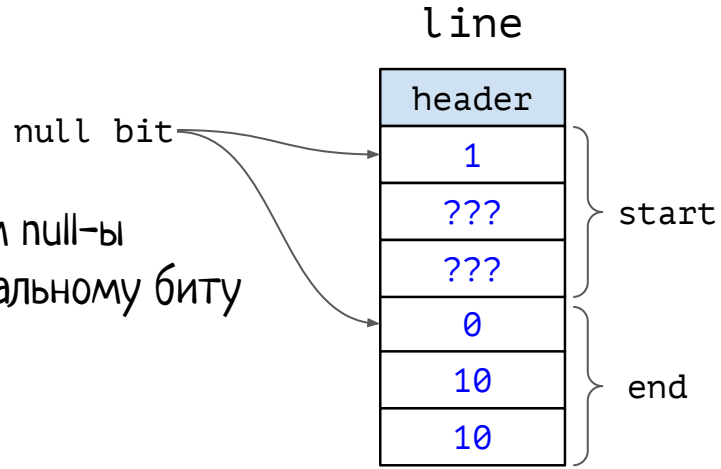
```
value record Point(int x, int y) { }

class Line {
    Point from;
    Point to;

    Line(int x2, int y2) {
        from = null;
        to = new Point(x2, y2);
    }
}

Line l = new Line(10, 10);
println("l.from == Point(0,0)? " + (l.from == new Point(0,0)));  <- false
println("l.from == null)? " + (l.from == null));                 <- true
```



Сложные вопросы: nullability

Проблема #2: должны ли поля типа value класса (в другом объекте) принимать null, в качестве значения?

```
value record Point(int x, int y) { }

class Line {
    Point from;
    Point to;

    Line(int x2, int y2) {
        from = null;
        to = new Point(x2, y2);
    }
}
```



Отличаем null-ы
по специальному биту

```
Line l = new Line(10, 10);
println("l.from == Point(0,0)? " + (l.from == new Point(0,0)));  <- false
println("l.from == null)? " + (l.from == null));                 <- true
```

Сложные вопросы: nullability

Проблема #2: должны ли поля типа value класса (в другом объекте, да и вообще) принимать null, в качестве значения?

Новое решение: это норм! (ну а что не так? пользователи привыкли работать с классами, и что в переменные можно пихать null, примем этот факт)

Последствия: нужно как-то по (потенциально скаляризованному или проинлайненному в другой объект) инстансу value класса понимать, null это или нет. Придется где-то хранить эту информацию!

Очень просто по семантике, но как-то больно: далеко не всем нужны эти null-ы, а размер у всех вырастет. Стрём!

Сложные вопросы: nullability

Проблема #2: должны ли поля типа value класса (в другом объекте, да и вообще) принимать null, в качестве значения?

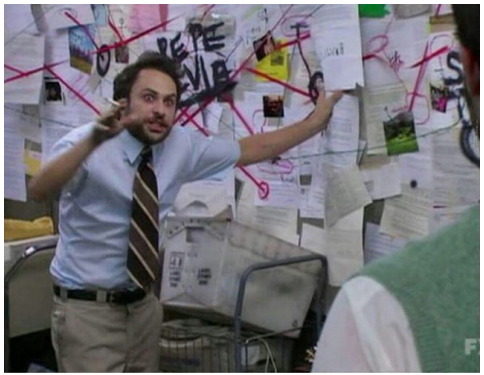
Новое решение (улучшенное): дать пользователю выбор, когда переменная или поле типа value может принимать null (тогда он за это заплатит), а когда нет.

Сложные вопросы: nullability

Проблема #2: должны ли поля типа value класса (в другом объекте, да и вообще) принимать null, в качестве значения?

Новое решение (улучшенное): дать пользователю выбор, когда переменная или поле типа value может принимать null (тогда он за это заплатит), а когда нет.

И это удивительным образом пересекается с идеей добавления в Java Null-Restricted и Nullable types: https://openjdk.org/jeps/8303099



Null-Restricted и Nullable types

Лирическое отступление: популярная во многих языках программирования (Kotlin, Swift, C#, etc) идея - помечать nullable переменные и поля.

Null-Restricted и Nullable types

Лирическое отступление: популярная во многих языках программирования (Kotlin, Swift, C#, etc) идея - помечать nullable переменные и поля. Теперь появляется в Java:

```
Object? nullableObj; // = null;
Object! nullRestrictedObj = new Object();
```

Null-Restricted и Nullable types

Лирическое отступление: популярная во многих языках программирования (Kotlin, Swift, C#, etc) идея - помечать nullable переменные и поля. Теперь появляется в Java:

```
Object? nullableObj; // = null;
Object! nullRestrictedObj = new Object();

nullableObj = nullRestrictedObj; // каст без проблем
```

Null-Restricted и Nullable types

Лирическое отступление: популярная во многих языках программирования (Kotlin, Swift, C#, etc) идея - помечать nullable переменные и поля. Теперь появляется в Java:

```
Object? nullableObj; // = null;
Object! nullRestrictedObj = new Object();

nullableObj = nullRestrictedObj; // каст без проблем

nullableObj = null;
nullRestrictedObj = nullableObj; // compiled, but NPE
```

Null-Restricted и Nullable types

Лирическое отступление: популярная во многих языках программирования (Kotlin, Swift, C#, etc) идея - помечать nullable переменные и поля. Теперь появляется в Java:

```
String? getSecondName(Person! person) {
    ... // прекрасно работает с параметрами и
        // возвращаемыми значениями
}
```

Null-Restricted и Nullable types

Лирическое отступление: популярная во многих языках программирования (Kotlin, Swift, C#, etc) идея - помечать nullable переменные и поля. Теперь появляется в Java:

```
class Person {
   String! name;  // и в целом с полями
   int age;




}
```

Null-Restricted и Nullable types

Лирическое отступление: популярная во многих языках программирования (Kotlin, Swift, C#, etc) идея - помечать nullable переменные и поля. Теперь появляется в Java:

```
class Person {
    String! name;  // и в целом с полями
    int age;

    Person(String name, int age) {
        System.out.println(this.name); // что напечатает?
        this.name = name;
        this.age = age;
    }
}
```

Null-Restricted и Nullable types

Лирическое отступление: популярная во многих языках программирования (Kotlin, Swift, C#, etc) идея - помечать nullable переменные и поля. Теперь появляется в Java:

```
class Person {
    String! name;  // и в целом с полями
    int age;

    Person(String name, int age) {
        System.out.println(this.name); // что напечатает?
        this.name = name;
        this.age = age;
    }
}
```

Ну нет, это плохо. Мы больше не можем инициализировать такие поля null-ами!

Знакомая проблема? Непроинициализированное поле в конструкторе

Сложные вопросы: инициализация

Проблема #1: никто не должен видеть не до конца проинициализированные экземпляры value классов

Как это гарантировать?

Новое решение (с 2024 года):

Вообще, ситуация с изменяемыми final-ами тоже никому не нравится.
По этому поводу есть интересный JEP-513: Flexible Constructor Bodies

Все поля value классов - strictly initialized => никто никогда больше не сможет увидеть недоинициализированный инстанс value 🎉

Намного проще с точки зрение реализации, и аккуратно вписывается в язык, переплетаясь с другими новыми фичами!

Сложные вопросы: инициализация

Проблема #1: никто не должен видеть не до конца проинициализированные экземпляры value классов

Как это гарантировать?

Новое решение (с 2024 года):

Вообще, ситуация с изменяемыми final-ами тоже никому не нравится. По этому поводу есть интересный JEP-513: Flexible Constructor Bodies

Все поля value классов - strictly initialized => никто никогда больше не сможет увидеть недоинициализированный инстанс value 🎉

Намного проще с точки зрение реализации, и аккуратно вписывается в язык, переплетаясь с другими новыми фичами!

Null-Restricted и Nullable types

Лирическое отступление: популярная во многих языках программирования (Kotlin, Swift, C#, etc) идея - помечать nullable переменные и поля. Теперь появляется в Java:

```
class Person {
    String! name;  // и в целом с полями
    int age;

    Person(String name, int age) {
        this.name = name;
        this.age = age;
        // super();
    }
}
```

Все null-restricted поля тоже становятся strictly initialized, т.е. должны быть проинициализированы до вызова super() ✨

Null-Restricted и Nullable types

Лирическое отступление: популярная во многих языках программирования (Kotlin, Swift, C#, etc) идея - помечать nullable переменные и поля. Теперь появляется в Java:

```
class Person {
    String! name;  // и в целом с полями
    int age;

    Person(String name, int age) {
        this.name = name;
        this.age = age;
        // super();
    }
}
```

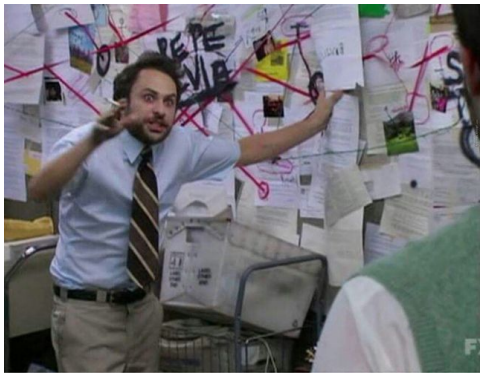
Все null-restricted поля тоже становятся strictly initialized, т.е. должны быть проинициализированы до вызова super(), ну и в безопасном контексте, без утекания this

Сложные вопросы: nullability

Проблема #2: должны ли поля типа value класса (в другом объекте, да и вообще) принимать null, в качестве значения?

Новое решение (улучшенное): дать пользователю выбор, когда переменная или поле типа value может принимать null (тогда он за это заплатит), а когда нет.

И это удивительным образом пересекается с идеей добавления в Java Null-Restricted и Nullable types: https://openjdk.org/jeps/8303099



Сложные вопросы: nullability

Проблема #2: должны ли поля типа value класса (в другом объекте, да и вообще) принимать null, в качестве значения?

Новое решение (улучшенное): дать пользователю выбор, когда переменная или поле типа value может принимать null (тогда он за это заплатит), а когда нет.

Почти в точности таким же синтаксисом, как с обычными типами:

```
value record Point(int x, int y) { }
class Line {
    Point from;
    Point to;

    Line(int x2, int y2) {
        from = null;        <-- норм
        to = new Point(x2, y2);
    }
}
```

Сложные вопросы: nullability

Проблема #2: должны ли поля типа value класса (в другом объекте, да и вообще) принимать null, в качестве значения?

Новое решение (улучшенное): дать пользователю выбор, когда переменная или поле типа value может принимать null (тогда он за это заплатит), а когда нет.

Почти в точности таким же синтаксисом, как с обычными типами:

```
value record Point(int x, int y) { }
class Line {
    Point! from;
    Point! to;

    Line(int x2, int y2) {
        from = null;      <-- compilation error
        to = new Point(x2, y2);
    }
}
```

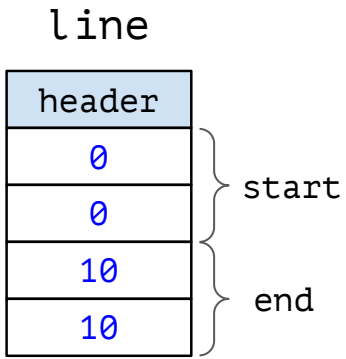
Сложные вопросы: nullability

Проблема #2: должны ли поля типа value класса (в другом объекте, да и вообще) принимать null, в качестве значения?

Новое решение (улучшенное): дать пользователю выбор, когда переменная или поле типа value может принимать null (тогда он за это заплатит), а когда нет.

Почти в точности таким же синтаксисом, как с обычными типами:

```
value record Point(int x, int y) { }
class Line {
    Point! from;
    Point! to;

    Line(int x2, int y2) {
        from = new Point(0, 0);
        to = new Point(x2, y2);
    }
}
```



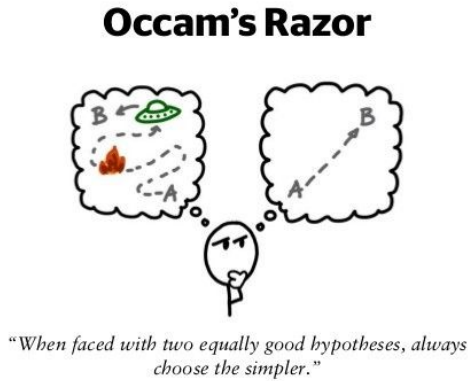
Теперь уверены про раскладку, она компактная делаем так 👍

Сложные вопросы: nullability

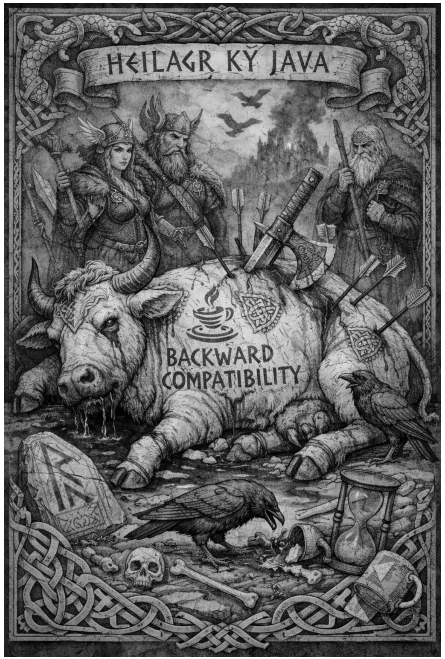
Проблема #2: должны ли поля типа value класса (в другом объекте, да и вообще) принимать null, в качестве значения?

Новое решение (улучшенное): дать пользователю выбор, когда переменная или поле типа value может принимать null (тогда он за это заплатит), а когда нет.

Реализуется исключительно на уровне javac, верификатора и пары проверок в рантайме. Очень просто!



Стражи Вальхаллы



~~Обратная совместимость~~

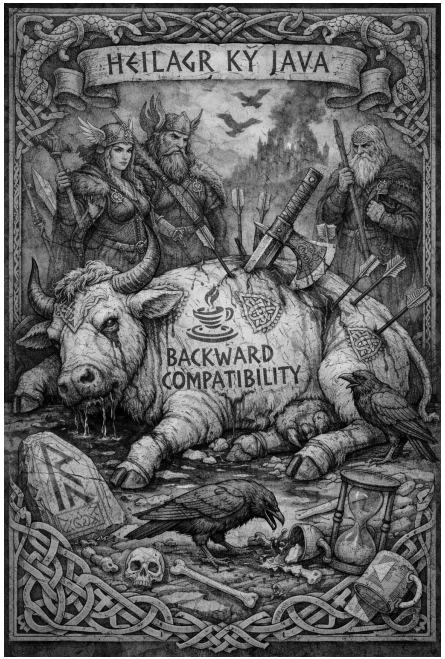
Runtime checks

~~Инициализация~~

Strictly initialized fields

Нуллабильность

Стражи Вальхаллы



~~Обратная совместимость~~

Runtime checks

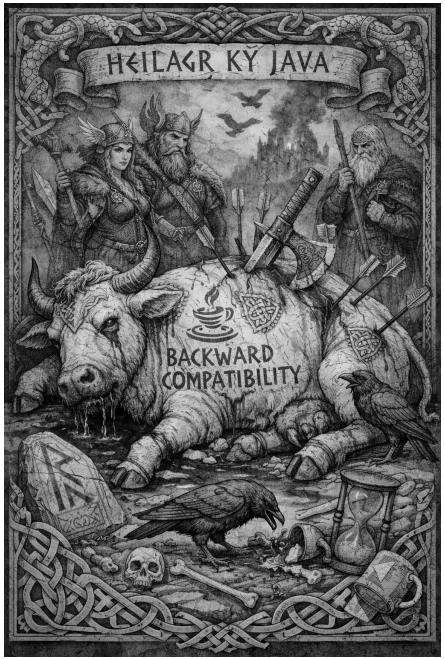
~~Инициализация~~

Strictly initialized fields

~~Нуллабильность~~

Null-restricted fields

Стражи Вальхаллы



~~Обратная совместимость~~

Runtime checks

~~Инициализация~~

Strictly initialized fields

~~Нуллабильность~~

Null-restricted fields

Неатомарный доступ к полям

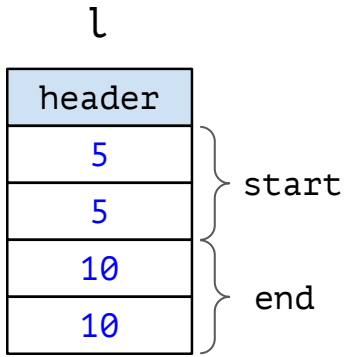
Сложные вопросы: атомарность доступа

Проблема #3: допустим, в классе есть поля типа value класса. Атомарен ли доступ к этому полю?

Сложные вопросы: атомарность доступа

Проблема #3: допустим, в классе есть поля типа value класса. Атомарен ли доступ к этому полю?

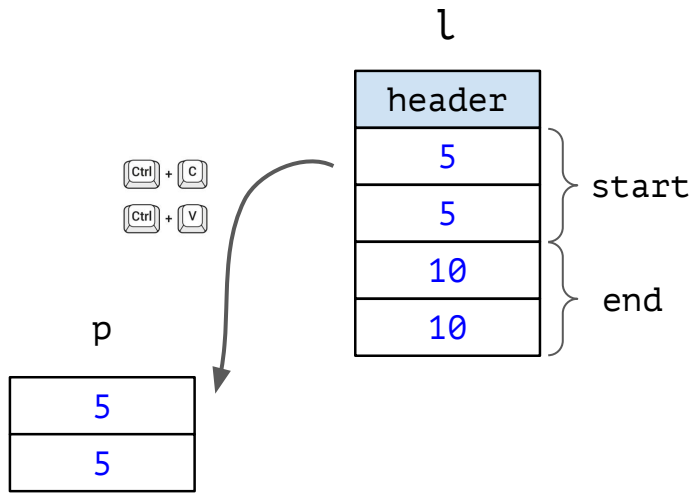
```
value record Point(int x, int y) { }

class Line {
    Point! from;
    Point! to;

    Line(int x1, int y1, int x2, int y2) {
        from = new Point(x1, y1);
        to = new Point(x2, y2);
    }
}

Line l = new Line(5, 5, 10, 10);
```



Сложные вопросы: атомарность доступа

Проблема #3: допустим, в классе есть поля типа value класса. Атомарен ли доступ к этому полю?

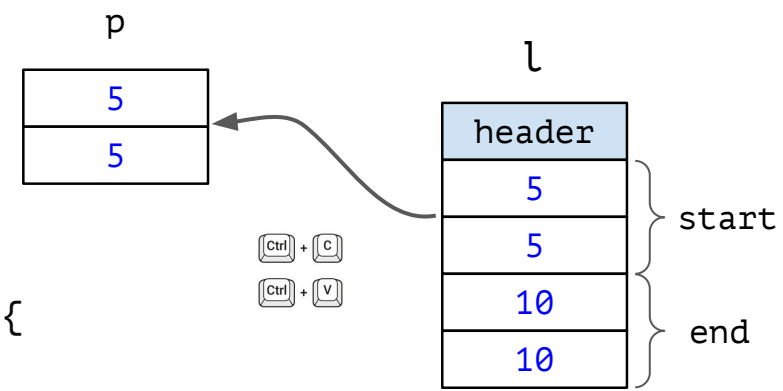
```
value record Point(int x, int y) { }
static Point! p = new Point(0, 0);

class Line {
    Point! from;
    Point! to;

    Line(int x1, int y1, int x2, int y2) {
        ...
    }
}

Line l = new Line(5, 5, 10, 10);
p = l.from;
```



Сложные вопросы: атомарность доступа

Проблема #3: допустим, в классе есть поля типа value класса. Атомарен ли доступ к этому полю?

```
value record Point(int x, int y) { }
static Point! p = new Point(0, 0);

class Line {
    Point! from;
    Point! to;

    Line(int x1, int y1, int x2, int y2) {
        ...
    }
}

Line l = new Line(5, 5, 10, 10);
p = l.from;
```

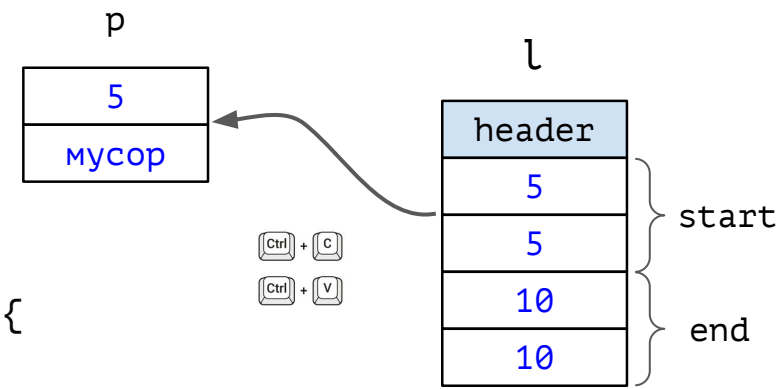


А что, если в середине копирования другой тред посмотрит на p?

Сложные вопросы: атомарность доступа

Проблема #3: допустим, в классе есть поля типа value класса. Атомарен ли доступ к этому полю?

```
value record Point(int x, int y) { }
static Point! p = new Point(0, 0);

class Line {
    Point! from;
    Point! to;

    Line(int x1, int y1, int x2, int y2) {
        ...
    }
}

Line l = new Line(5, 5, 10, 10);
p = l.from;
```

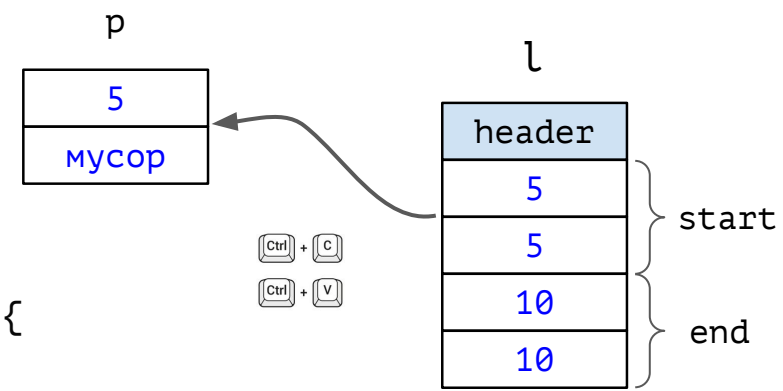


А что, если в середине копирования другой тред посмотрит на p? Он увидит p в невалидном состоянии.

Сложные вопросы: атомарность доступа

Проблема #3: допустим, в классе есть поля типа value класса. Атомарен ли доступ к этому полю?

```
value record Point(int x, int y) { }
static Point! p = new Point(0, 0);

class Line {
    Point! from;
    Point! to;

    Line(int x1, int y1, int x2, int y2) {
        ...
    }
}

Line l = new Line(5, 5, 10, 10);
p = l.from;
```

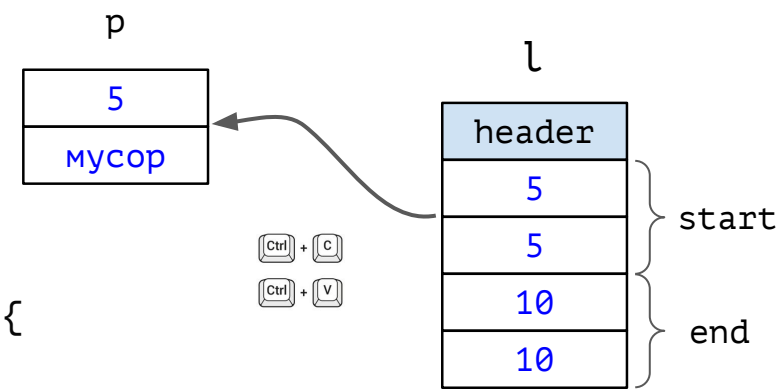


А что, если в середине копирования другой тред посмотрит на p? Он увидит p в невалидном состоянии. Прерывать нельзя!

Сложные вопросы: атомарность доступа

Проблема #3.5: доступ должен быть атомарным. А как этого достичь то?

```
value record Point(int x, int y) { }
static Point! p = new Point(0, 0);

class Line {
    Point! from;
    Point! to;

    Line(int x1, int y1, int x2, int y2) {
        ...
    }
}

Line l = new Line(5, 5, 10, 10);
p = l.from;
```

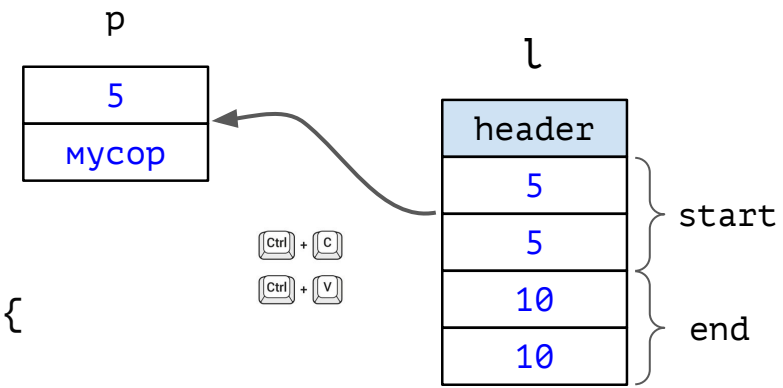


А что, если в середине копирования другой тред посмотрит на p? Он увидит p в невалидном состоянии. Прерывать нельзя!

Сложные вопросы: атомарность доступа

Проблема #3.5: доступ должен быть атомарным. А как этого достичь то? При том достичь максимально эффективно!

```
value record Point(int x, int y) { }
static Point! p = new Point(0, 0);

class Line {
    Point! from;
    Point! to;

    Line(int x1, int y1, int x2, int y2) {
        ...
    }
}

Line l = new Line(5, 5, 10, 10);
p = l.from;
```

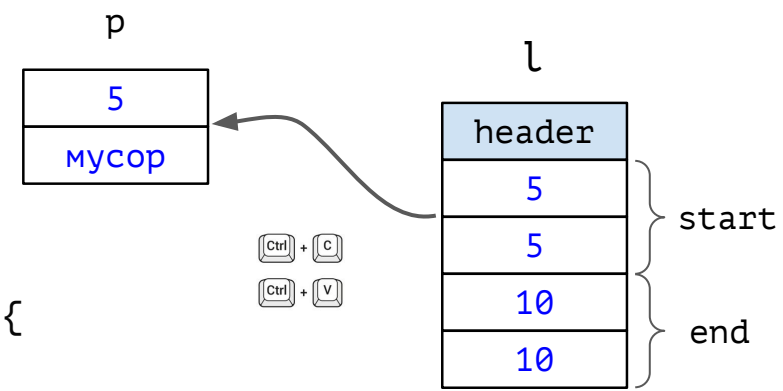


А что, если в середине копирования другой тред посмотрит на p? Он увидит p в невалидном состоянии. Прерывать нельзя!

Сложные вопросы: атомарность доступа

Иногда можем атомарно скопировать: если суммарный размер влезает в 64-бита (тогда скопируем все одним широким мувом)

```
value record Point(int x, int y) { }
static Point! p = new Point(0, 0);

class Line {
    Point! from;
    Point! to;

    Line(int x1, int y1, int x2, int y2) {
        ...
    }
}

Line l = new Line(5, 5, 10, 10);
p = l.from;
```



А что, если в середине копирования другой тред посмотрит на p? Он увидит p в невалидном состоянии. Прерывать нельзя!

Сложные вопросы: атомарность доступа

Иногда можем атомарно скопировать: если суммарный размер влезает в 64-бита (тогда скопируем все одним широким мувом)

А что делать, если value класс больше?

Сложные вопросы: атомарность доступа

Иногда можем атомарно скопировать: если суммарный размер влезает в 64-бита (тогда скопируем все одним широким мувом)

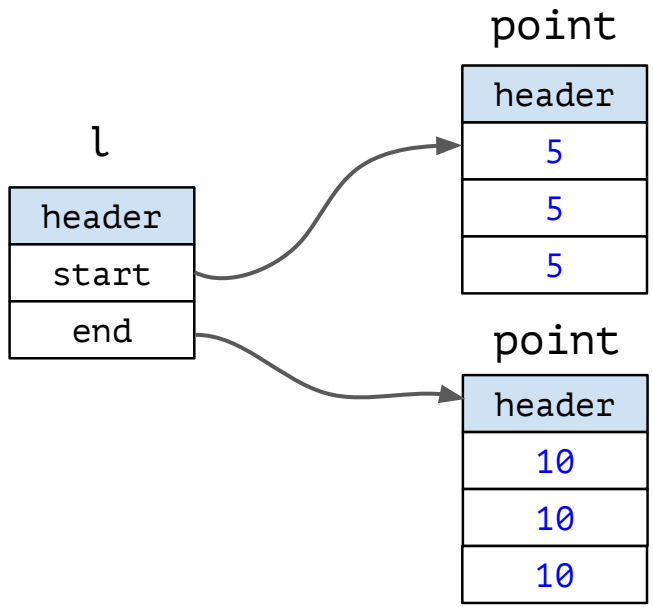
А что делать, если value класс больше? Не инлайнить в объект!

Сложные вопросы: атомарность доступа

Иногда можем атомарно скопировать: если суммарный размер влезает в 64-бита (тогда скопируем все одним широким мувом)

А что делать, если value класс больше? Не инлайнить в объект!

```
value record Point(int x, int y, int z) { }

class Line {
    Point! from;
    Point! to;

    Line(...) {
        ...
    }
}

Line l = new Line(5, 5, 5, 10, 10, 10);
```



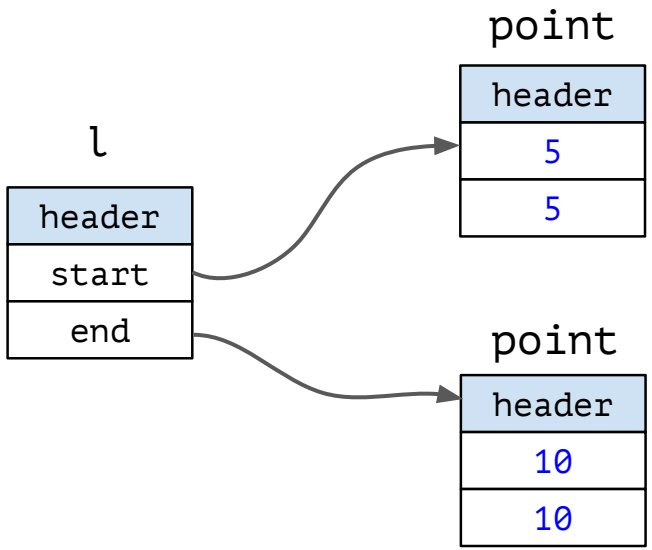
Сложные вопросы: атомарность доступа

Иногда можем атомарно скопировать: если суммарный размер влезает в 64-бита (тогда скопируем все одним широким мувом)

А что делать, если value класс больше? Не инлайнить в объект!

```
value record Point(int x, int y) { }

class Line {
    Point from;
    Point to;

    Line(...) {
        ...
    }
}

Line l = new Line(5, 5, 10, 10);
```

Почему?



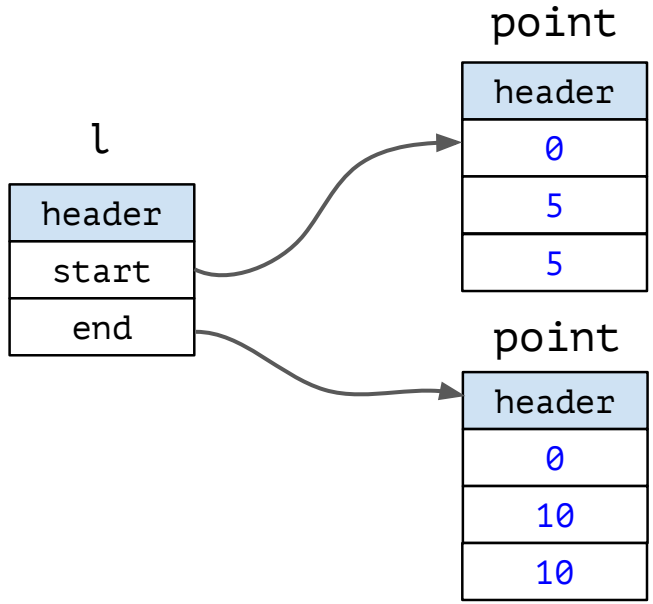
Сложные вопросы: атомарность доступа

Иногда можем атомарно скопировать: если суммарный размер влезает в 64-бита (тогда скопируем все одним широким мувом)

А что делать, если value класс больше? Не инлайнить в объект!

```
value record Point(int x, int y) { }

class Line {
    Point from;
    Point to;

    Line(...) {
        ...
    }
}

Line l = new Line(5, 5, 10, 10);
```

Почему?
Из-за null-bit-a!

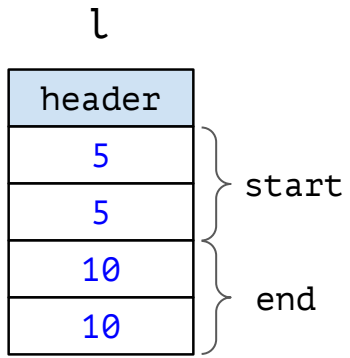


Сложные вопросы: атомарность доступа

Иногда можем атомарно скопировать: если суммарный размер влезает в 64-бита (тогда скопируем все одним широким мувом)

А что делать, если value класс больше? Не инлайнить в объект!

```
value record Point(int x, int y) { }

class Line {
    Point! from;
    Point! to;

    Line(...) {
        ...
    }
}

Line l = new Line(5, 5, 10, 10);
```



Сложные вопросы: атомарность доступа

Проблема #3: пусть в классе есть поля типа value класса. Атомарен ли доступ к этому полю?

Решение: должен быть атомарен, иначе можно пронаблюдать невалидный валуй из другого потока.

Сложные вопросы: атомарность доступа

Проблема #3: пусть в классе есть поля типа value класса. Атомарен ли доступ к этому полю?

Решение: должен быть атомарен, иначе можно пронаблюдать невалидный валуй из другого потока.

Как реализовать: инлайнить только достаточно маленькие значения (< 64 бит для nullable value, <= 64 бит для null-restricted value).

Сложные вопросы: атомарность доступа

Проблема #3: пусть в классе есть поля типа value класса. Атомарен ли доступ к этому полю?

Решение: должен быть атомарен, иначе можно пронаблюдать невалидный валуй из другого потока.

Как реализовать: инлайнить только достаточно маленькие значения (< 64 бит для nullable value, <= 64 бит для null-restricted value) и ждать новых эффективных инструкций для 128-бит.

Сложные вопросы: атомарность доступа

Проблема #3: пусть в классе есть поля типа value класса. Атомарен ли доступ к этому полю?

Решение: должен быть атомарен, иначе можно пронаблюдать невалидный валуй из другого потока.

Как реализовать: инлайнить только достаточно маленькие значения (< 64 бит для nullable value, <= 64 бит для null-restricted value) и ждать новых эффективных инструкций для 128-бит.

Но иногда так хочется инлайнить и объекты побольше...

Сложные вопросы: атомарность доступа

```
value class Point {
    double x;
    double y;

    public implicit Point();

    public Point(double x, double y) {
        this.x = x;
        this.y = y;
    }
}
```

Не влезает в 64-бита, каждое поле здесь 64-битное

Сложные вопросы: атомарность доступа

```
value class Point implements LooselyConsistentValue {
    double x;
    double y;

    public implicit Point();

    public Point(double x, double y) {
        this.x = x;
        this.y = y;
    }
}
```

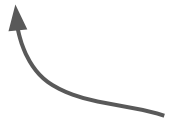
Сложные вопросы: атомарность доступа

```
value class Point implements LooselyConsistentValue {
    double x;
    double y;

    public implicit Point();

    public Point(double x, double y) {
        this.x = x;
        this.y = y;
    }
}
```

"я понимаю, что доступ может быть не атомарным и беру на себя ответственность"

Сложные вопросы: атомарность доступа

```
value class Point implements LooselyConsistentValue {
    double x;
    double y;

    public implicit Point();

    public Point(double x, double y) {
        this.x = x;
        this.y = y;
    }
}
```



"я понимаю, что доступ может быть не атомарным и беру на себя ответственность"

```
Point![] ps = { new Point(0.0, 1.0) };
new Thread(() -> ps[0] = new Point(2.0, 3.0)).start();
   Point p = ps[0]; // may be (2.0, 1.0), among other possibilities
```

Сложные вопросы: атомарность доступа

Проблема #3: пусть в классе есть поля типа value класса. Атомарен ли доступ к этому полю?

Решение: должен быть атомарен, иначе можно пронаблюдать невалидный валуй из другого потока.

Как реализовать: инлайнить только достаточно маленькие значения (< 64 бит для nullable value, <= 64 бит для null-restricted value) и ждать новых эффективных инструкций для 128-бит.

Если очень нужно инлайнить большие валуи, имплементируйте: LooselyConsistentValue, но тогда сами разбирайтесь с последствиями/гарантируйте атомарность.

Сложные вопросы: атомарность доступа

Проблема #3: пусть в классе есть поля типа value класса. Атомарен ли доступ к этому полю?

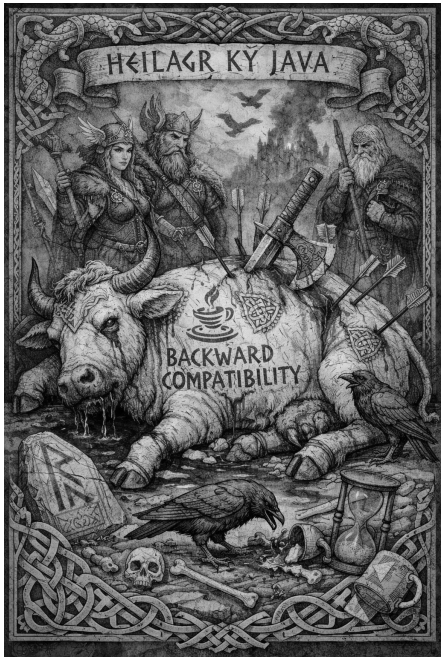
Решение: должен быть атомарен, иначе можно пронаблюдать невалидный валуй из другого потока.

Как реализовать: инлайнить только достаточно маленькие значения (< 64 бит для nullable value, <= 64 бит для null-restricted value) и ждать новых эффективных инструкций для 128-бит.

Если очень нужно инлайнить большие валуи, имплементируйте: LooselyConsistentValue, но тогда сами разбирайтесь с последствиями/гарантируйте атомарность.

P.S. похожая история была на 32-битных системах с лонгами и даблами, доступ к ним - неатомарен! 🤯

Стражи Вальхаллы



~~Обратная совместимость~~

Runtime checks

~~Инициализация~~

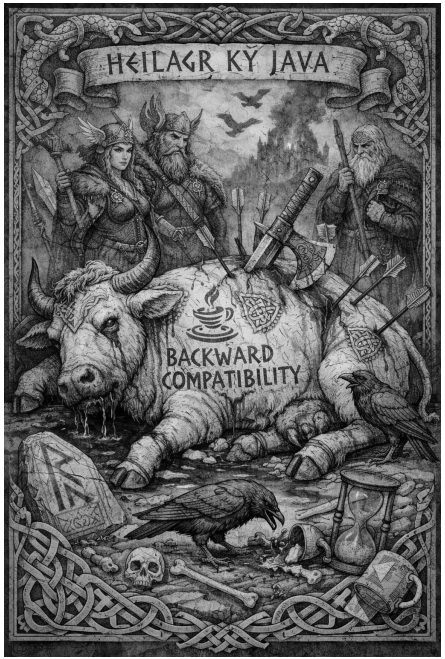
Strictly initialized fields

~~Нуллабильность~~

Null-restricted fields

Неатомарный доступ к полям

Стражи Вальхаллы



~~Обратная совместимость~~

Runtime checks

~~Инициализация~~

Strictly initialized fields

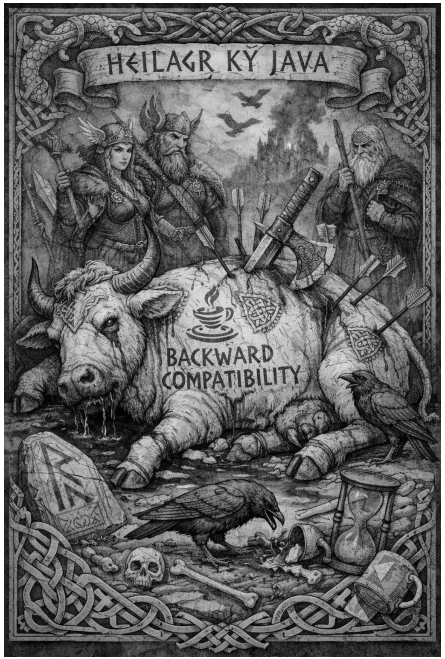
~~Нуллабильность~~

Null-restricted fields

~~Неатомарный доступ к полям~~

Heap flattening limitations

~~Стражи Вальхаллы~~



~~Обратная совместимость~~

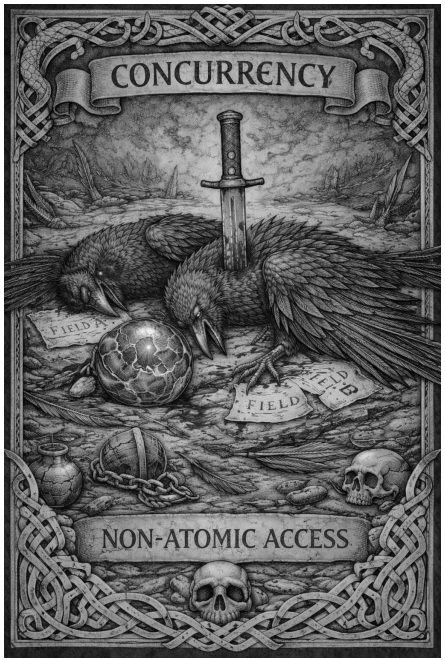
Runtime checks

~~Инициализация~~

Strictly initialized fields

~~Нуллабильность~~

Null-restricted fields



~~Неатомарный доступ к полям~~

Heap flattening limitations

Когда будем в Вальхалле?

Когда будет готово?

- JEP 513: Flexible Constructor Bodies (релизнулся в Java 25 в конце сентября)

Когда будет готово?

- JEP 513: Flexible Constructor Bodies (релизнулся в Java 25 в конце сентября)

- JEP-401: Value Classes and Objects. Скоро выйдет в preview (идет обсуждение, вероятно в Java ~~26~~ ~~27~~ 28)

Когда будет готово?

- JEP 513: Flexible Constructor Bodies (релизнулся в Java 25 в конце сентября)

- JEP-401: Value Classes and Objects. Скоро выйдет в preview (идет обсуждение, вероятно в Java ~~26~~ ~~27~~ 28)

- JEP: Null-Restricted and Nullable Types (draft) and JEP: null restricted values (draft)

Когда будет готово?

- JEP 513: Flexible Constructor Bodies (релизнулся в Java 25 в конце сентября)

предусловие

- JEP-401: Value Classes and Objects. Скоро выйдет в preview (идет обсуждение, вероятно в Java ~~26~~ ~~27~~ 28)

- JEP: Null-Restricted and Nullable Types (draft) and JEP: null restricted values (draft)

Позволит делать значительно больше оптимизаций и выправит семантику

Когда будет готово?

- JEP 513: Flexible Constructor Bodies (релизнулся в Java 25 в конце сентября)

предусловие

- JEP-401: Value Classes and Objects. Скоро выйдет в preview (идет обсуждение, вероятно в Java ~~26~~ ~~27~~ 28)

- JEP: Null-Restricted and Nullable Types (draft) and JEP: null restricted values (draft)

Позволит делать значительно больше оптимизаций и выправит семантику

- Parametric JVM (draft of a draft, 41 pages)

Когда будет готово?

- JEP 513: Flexible Constructor Bodies (релизнулся в Java 25 в конце сентября)

предусловие

- JEP-401: Value Classes and Objects. Скоро выйдет в preview (идет обсуждение, вероятно в Java ~~26~~ ~~27~~ 28)

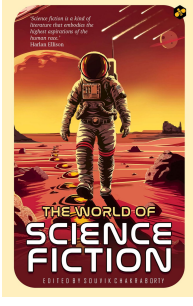
предусловие

- JEP: Null-Restricted and Nullable Types (draft) and JEP: null restricted values (draft)

Позволит делать значительно больше оптимизаций и выправит семантику

- Parametric JVM (draft of a draft, 41 pages)

Рассуждения о том, как реализовать специализированные дженерики (aka шаблоны) и ArrayList<int>



Чему нас может научить проект Вальхалла?

Чему нас может научить проект Вальхалла?

- Не все вопросы решаются на стороне реализации VM. Зачастую дизайн языка важнее;

Чему нас может научить проект Вальхалла?

- Не все вопросы решаются на стороне реализации VM. Зачастую дизайн языка важнее;

- Выкинуть переусложненный прототип - это нормально, и случается с лучшими из нас;

Чему нас может научить проект Вальхалла?

- Не все вопросы решаются на стороне реализации VM. Зачастую дизайн языка важнее;
- Выкинуть переусложненный прототип - это нормально, и случается с лучшими из нас;
- identity, nullability и атомарность доступа к полю - это "дополнительные услуги", которые можно брать (и платит за них), а можно отказаться от них и сэкономить.

Q & A

Алло, это отладочная?

@GDB_DBG