Lecture 17

Static Types

type safety, static vs dynamic checks, subtyping

Ras Bodik
Ali and Mangpo

Hack Your Language!
CS164: Introduction to Programming Languages and Compilers, Spring 2013
UC Berkeley

Announcements

Final project next steps:

peer feedback:

- proposals to be assigned by tomorrow
- Due after spring break

feedback from Ras:

- after spring break
- Watch for sign up email

Ras' office hours cancelled today

- Schedule a meeting via email

Outline

Motivation:

- Performance: better compilation via types
- Correctness: catching errors at compile time

Semantic analysis

symbol analysis + type analysis

Static vs. dynamic typing

also vs. no typing

Static type checks

For objects with subclassing; casts; and arrays

Dynamic checks

Why realistic static languages rely on static checks

Definitions

Static vs. dynamic; value vs. variable

Dynamic = known at run time

- specific to a given program input

Static = known at compile time

- not specific to given input => must be true for all inputs

Type = set of values and operations on them

- example: ints are -2^{32}, …, 0, … $2^{32}-1$, with operations +, -, …

Dynamic type of a variable

- is the type of the value stored in the variable at runtime

Static type of a variable

- Annotated by programmers or inferred by the compiler
- type of *all* values that the variable *might* hold at runtime

Why do we use static types?

Motivation: Why are some languages faster?

implementations of

Performance as measured on "language shootout"

An incomplete list of languages. See web site for more.

http://shootout.alioth.debian.org/

Norm. performance language + implementation (2007)

1.0	C / gcc
1.2	Pascal Free Pascal
1.5	Java 6 -server
1.6	Lisp SBCL
1.8	BASIC FreeBASIC
3.1	Fortran G95
11	Lua
14	Scheme MzScheme
17	Python
21	Perl
23	PHP
44	JavaScript SpiderMonkey
48	Ruby

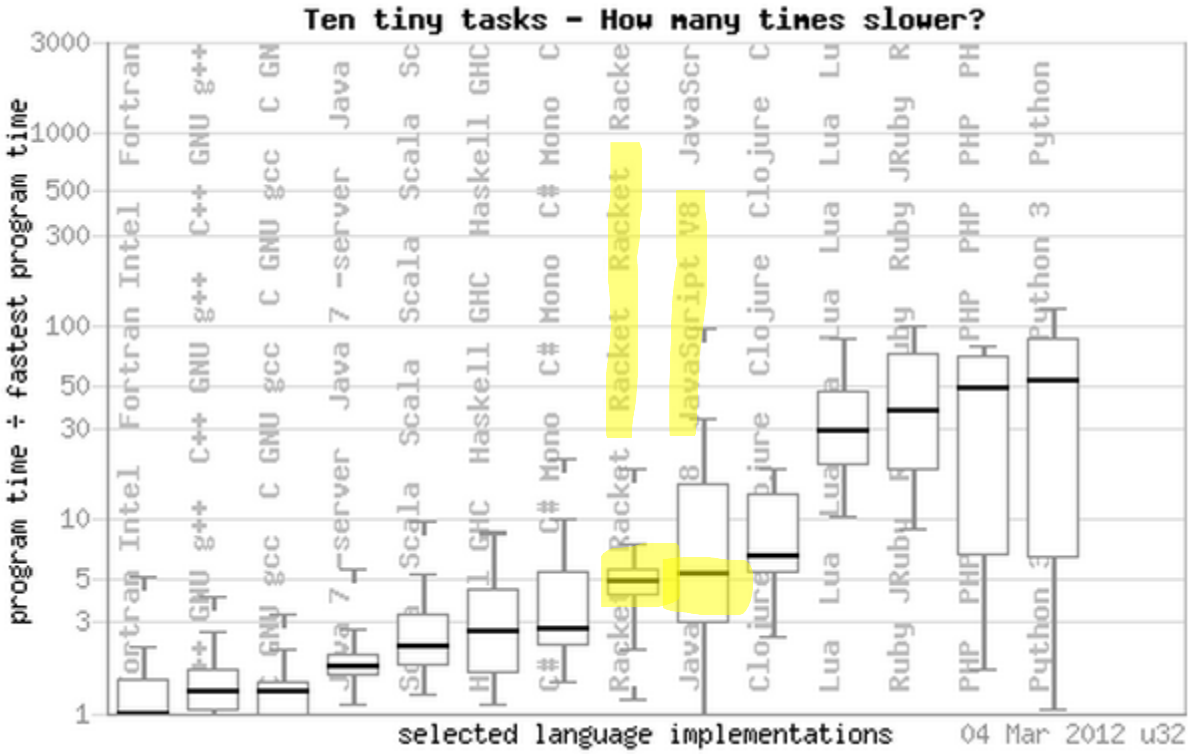
Performance differences form equivalence classes

Notice the huge step between 3x and 10x

What might be the source of the difference?

All languages under 3x are statically typed

For comparison, more recent results (2012)



compilers for dynamically typed languages improved since 2007

partly motivated by browser performance wars (JavaScript JITs)

Both Racket and JS are dynamically typed but they rely on a good JIT compiler to use information known at runtime

2012 results

chart

compare 2	|-	|---	25%	median	75%	---|	-|
Fortran Intel	1.00	1.00	1.00	**1.00**	1.49	2.24	5.15
C++ GNU g++	1.00	1.00	1.03	**1.27**	1.68	2.65	4.06
C GNU gcc	1.00	1.00	1.00	**1.30**	1.47	2.17	3.24
ATS	1.01	1.01	1.17	**1.37**	1.60	2.24	7.95
Ada 2005 GNAT	1.01	1.01	1.35	**1.62**	2.44	4.08	7.55
Java 7 -server	1.11	1.11	1.58	**1.76**	2.02	2.68	5.62
Scala	1.24	1.24	1.81	**2.18**	3.22	5.33	9.79
Pascal Free Pascal	1.38	1.38	2.01	**2.39**	2.84	4.10	5.36
Haskell GHC	1.10	1.10	1.64	**2.64**	4.34	8.38	8.73
C# Mono	1.44	1.44	2.26	**2.72**	5.46	10.26	20.52
Clean	1.76	1.76	2.11	**3.01**	4.15	7.21	11.17
OCaml	1.55	1.55	2.02	**3.40**	4.90	6.26	6.26
Lisp SBCL	1.02	1.02	1.87	**3.81**	4.99	9.67	10.87
F# Mono	1.43	1.43	2.53	**3.97**	5.62	10.24	18.28
Racket	1.17	2.16	4.19	**4.79**	5.55	7.58	18.58
Go	1.99	1.99	2.84	**5.15**	7.57	9.48	9.48
JavaScript V8	1.00	1.00	2.97	**5.27**	15.30	33.80	98.78
Clojure	2.47	2.47	5.47	**6.40**	13.79	18.59	18.59
Erlang HiPE	3.28	3.28	7.20	**12.79**	23.16	30.80	30.80
Smalltalk VisualWorks	5.35	5.35	12.10	**14.47**	25.93	46.68	78.54
Lua	10.45	10.45	19.68	**28.51**	46.85	86.87	86.87
Ruby JRuby	9.02	9.02	18.65	**37.10**	72.73	100.02	100.02
Ruby 1.9	6.24	6.24	9.28	**43.79**	89.90	210.84	262.44
PHP	1.68	1.68	6.58	**48.33**	71.26	79.70	79.70
Python 3	1.06	1.06	6.50	**52.17**	86.22	126.24	126.24
Mozart/Oz	6.73	6.73	34.71	**57.66**	76.90	140.17	147.58
Perl	2.22	2.22	7.31	**57.75**	103.44	220.49	220.49
C CINT	71.42	71.42	179.97	**251.18**	379.29	678.27	4790.34

Why static types allow compilation to efficient code

How to compile Python/Lua/JS to efficient C code?

try compiling, by hand, the statement a = b + c

Ideally, we need to know if b, c will hold ints or strings

But in JS we don't know types of values in b, c until runtime

So the compiler must emit C code that

i) **Obtains** (at run time, of course) the (dynamic) type of values in variables b and c.

ii) Based on these types, the code **dispatches** to either integer addition or to string concatenation or other

Why static types allow compilation to efficient code

Compare to code produced by a Java compiler

Lesson

Programs in languages with static typing run faster.

Reason: the compiler has more information about the program and can thus generate more efficient code.

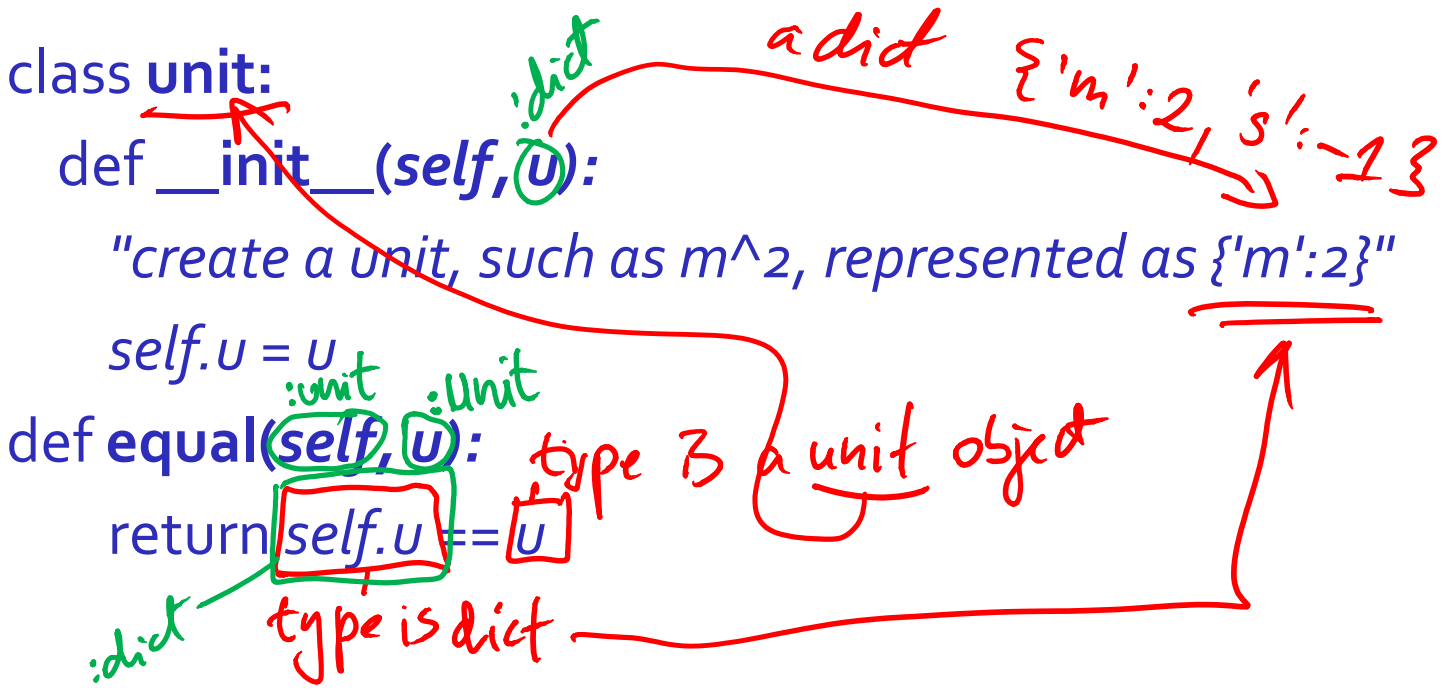
- Better layout of objects: structs rather than dictionaries
- No dynamic examination of types of values of base types such as ints, floats

Performance is one reason why we use static types.

- But JIT (runtime) compilers for dynamically typed languages can obtain some of this information at runtime, and produce better code.

Motivation 2: catching bugs w/out running the code

I ran into this bug (my own) in GCalc. Can you see it?



Would a compiler with static type checking catch the error?

Type safety

A type-safe language

What does "type safe language" mean?

It means that "things won't go wrong" during execution

What sorts of errors are eliminated by type safety?

It can't possibly mean that logical errors are eliminated!?!

So what notion of correctness do type checks guarantee?

A well-typed program will not

- add integer and list data types, etc
- access fields that do not exist in an object
- access array elements past the end of the array
- write an int into a location and later read it as a pointer

We'll show the dangers of not catching these errors

When are these errors caught?

These errors could be prevented with

- static checks or dynamic checks or left unnoticed (unsafe)

Example: accessing a field in an object

- In Lua, expression p.f is not guaranteed to find f in object p.
  - p.f will throw a (runtime) exception when f is not in the object
- In Java, p.f will always succeed at runtime
  - if the object has no field f, the compiler would reject the program; think of this as "compile-time exception"
- In assembly, p.f will be always executed
  - If, due to programmer error, p points to an object that has no field f, then the code for p.f=123 will write 123 into some unpredictable location

Lesson from the previous slide

Exceptions (runtime or compile time) are better than silent failures which manifest the error much later in the execution

Interestingly, Lua and Python are type safe just like Java.

They just check for errors at different times (runtime vs in the compiler)

Java is statically typed. Python is dynamically typed.

Clarification on silent type errors in C

Memory safety:

- Program will never access memory outside an object (incl. an array)

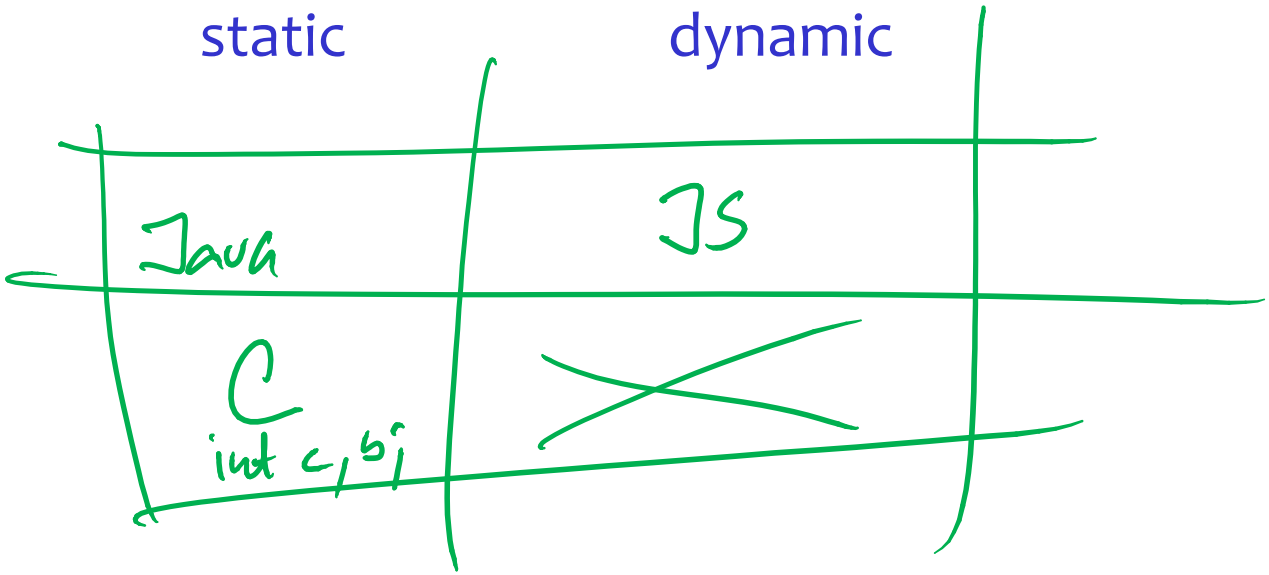
Why is memory safety important

- we compile "p.f" into "LD r1, offset(r2)"
- what could go wrong here?

Safe vs unsafe vs Static vs Dynamic

Two axes:

- statically checked types vs dynamically checked types
- safe vs. unsafe



The mechanics of type checking

How is static typing performed? Two phases

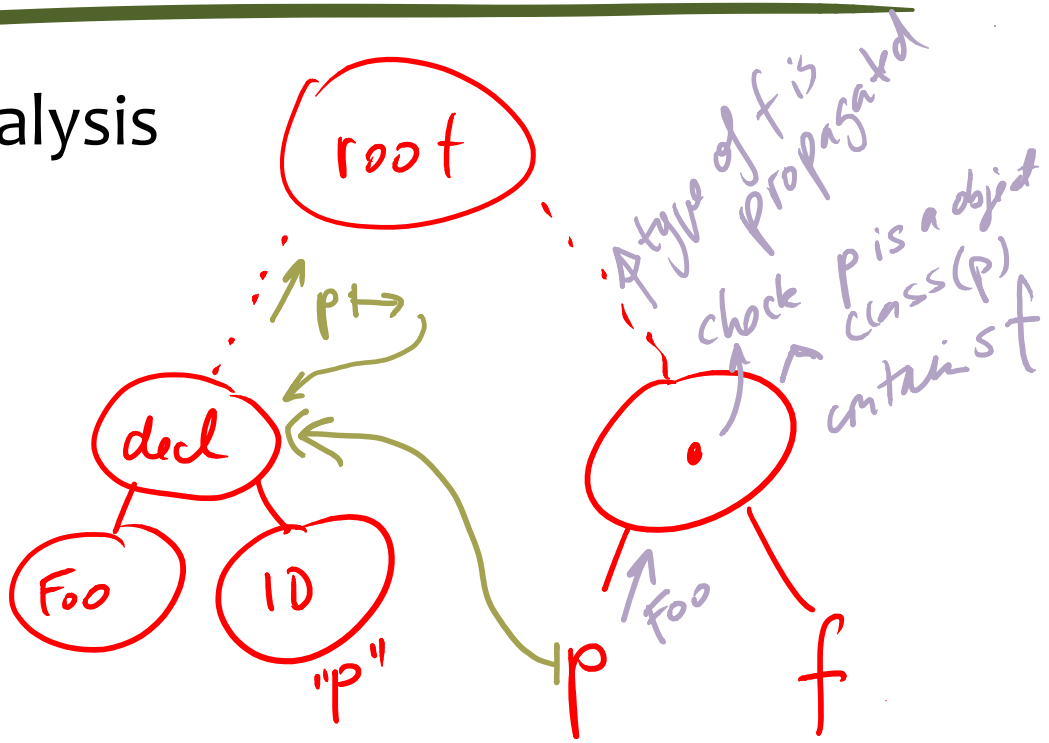
Also called semantic analysis

1) symbol analysis

bind symbols to their declarations

Foo p;
⋮
p.f

2) type analysis



Examples

Design simple type checking rules and visualize type checking on the AST of a program

Language 1: + operation ranging over int, float, string

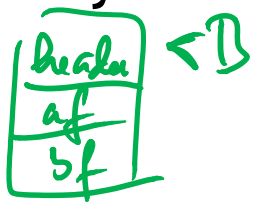
Language 2: as L1 but coercion is inserted by type checker, by calling toString(), to add ints and floats to strings

Types for OO Languages

Static and dynamic types for OO languages

Static type: declared type of a variable

Dynamic type: class of object referenced by the variable



Example: A a = new B()

- static type of a is A; dynamic type of a is B (B subclass of A)

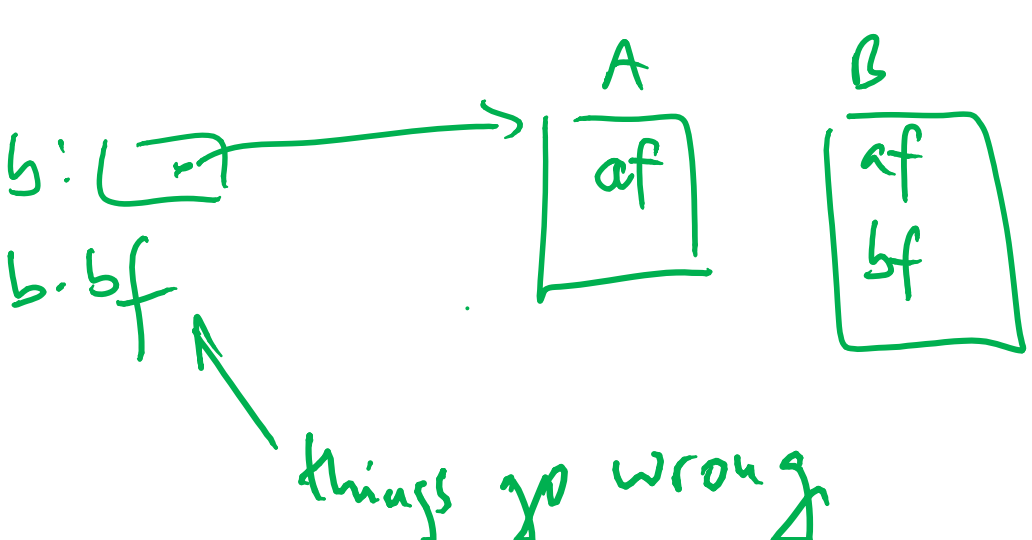
What assignments are legal?

a = a ✓

a = b ✓

b = a ✗

b = b ✓



Subtyping. Compatible types.

Compatible static types: tell us when x := y is legal

- it's legal when static type of y is compatible with that of x
- type(y) is compatible with type(x) iff type(y) $\leq$ type(x)
- where $t_1 \leq t_2$ means t_1 is a subclass of t_2 or $t_1 = t_2$

Example: given class A extends B {}; A a; B b;

b = a is legal

a = b is illegal (rejected by type checker)

Why notion of compatible types guarantees safety?

Safety:

- when p.f is executed, object pointed to by p will contain field f
- and that field will be in the expected location
- same for methods

The static check based on compatible types work because

static types **conservatively overestimate dynamic types** of values that will be stored in variables at run time

static type assumes a may point to an A or a B object

Pros and Cons of Static Type Checking

Pros: Type checking proves a theorem

When a program type checks, the compiler has proven a theorem stating that the program won't go wrong, e.g. we'll definitely find the field in the object

Cons: Static Type Checker Rejects Good Programs

This would run OK in Python, but rejected by Java type checker

```
class B extends A { int b; }
A a = new B()
a.b    // error: class A does not contain field b
```

Reason for error: the checker does not know that dynamic type of a is B

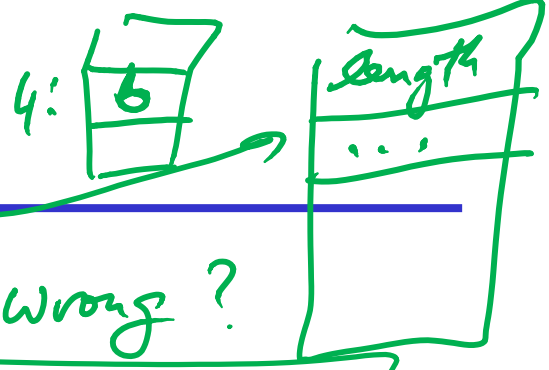
Static checker uses limited reasoning:

- considers only the declared types of variables (static types)
  - not the dynamic types
- checks operations independently from others
  - does not track flow of pointers through the program

Java type rules for arrays

Exercise for you

from an AP study guide




```
class Bar { int b; }
class Foo extends Bar { int f; }
Foo[] A = new Foo[2];
A[0] = new Foo();  A[1] = new Foo();
Object X = A;
Bar B = A;
Foo[] C = (Foo[]) B;
Foo[] Y = (Foo[]) X;
```

what goes wrong ?

int z = 3.5

expected got type: B or subclass of B; Foo[]

Rejected because something could go wrong at run-time.

- At which line does the static type checker rejects the program?
- What static type checking rule is violated? (What is the reported error?)
- What could go wrong at run time if the checker did not reject the program?

Extend the program with a few statements to show the run-time failure or silent error.

See the assigned reading notes for detailed discussion

See assigned reading

The rest of the slides contains exercises and questions that you should be able to answer.

Please see assigned reading and the linked AP Study guide for explanation of answers.

The assigned reading, linked from the course web page, also explains how omitting a safety check can lead to an exploit.

why static types

review

What does the compiler know with static types?

dynamically typed languages (Python/Lua/JS):

```
function foo(arr) {
    return arr[1]+2
}
```

statically typed languages (Java/C#/Scala):

```
function foo(arr:int[]) : int {
    return arr[1]+2
}
```

How do compilers exploit compile-time knowledge?

dynamically typed languages (Python/Lua/JS):

```
function foo(arr) {
    return arr[1]+2
}
```

statically typed languages (Java/C#/Scala):

```
function foo(arr:int[]) : int {
    return arr[1]+2
}
```

why we want dynamic checks in a statically typed language

downcasts

Downcasts. Why are downcasts needed?

Foo f = (Foo) p;

Why do we need them in a Java program?

Disallowing downcasts would make some programs impossible to write

Downcasts need to be checked at runtime. Why?

Why are downcasts not needed in dynamically typed languages?

Run-time checks in Java

If we want to ensure memory safety, we need some checks

Which of these checks are performed before p.f =r **at runtime**?

```
if (p==null)
    throw NullPointerException

if (class(p) does not contain a field “f”)
    throw MissingFieldException

if (type(r) not compatible with type(f))
    throw IncompatibleAssignmentException
```

Discussion

The last two checks are not needed

- these properties are ensured by checking types at compile time

An exercise for you:

- the null-pointer check may be expensive
- Q1: how would you implement it cheaply (or for free)?
- Q2: could you extend Java types to ensure that a pointer is not null?
  - What language constructs (types) would you add?
  - What compile time checks would you add?
  - What run-time time checks would you add?

Discussion

Q1:

Q2:

why we want dynamic checks in a statically typed language

assignment to arrays

Checks in Java, for arrays

Like downcasts, arrays also have static and dynamic checks:

- What static checks do we perform?
- What runtime checks do we perform??

To answer, let's answer these questions first:

- what invariants do static checks enforce?
- what invariants do dynamic checks enforce?
- what type information do arrays carry at run-time, if any?

Array checks in Java

Some type properties are checked statically, some dynamically.

Let's ask what rationale was used to guide this division of labor

Consider a fragment of our previous example:

```
class Foo extends Bar { int f; }
Bar[] b = new Foo[2];   // (2)
Foo[] c = (Foo[]) b;
```

If language designers allowed (2) to type check:

- what changes to compiled code would have to be made?
  - More dynamic check would be needed? What checks specifically?
- To determine what checks are needed, ask yourself what could go wrong at run time

Exercise: mark static and dynamic checks that fail

For each check, write a stmt that could go wrong if check was ignored:

```
class A  { int a; }    class B extends A { int b; }

A[] a = new A[2];      B[] b = new B[2];
a[1] = b[1];           b[1] = a[1];

b = (B[]) a;           a = (B[]) a;
b = a;                 a = b;
```

Static checks for arrays

What invariants do static checks enforce?

- a variable of static type A[] will point to null or an Array object that contains objects compatible with A.

What do we check statically? Which ops needs static checks?

Expression **Statically valid if** (assume a is of type A[])

- a = b
- $a[e_1] = e_2$
- p = a[e]

Dynamic checks for arrays

What invariants do dynamic checks enforce?

- same as the static check

What do we check dynamically? Which ops need checks?

(B[]) e (cast)

if (type(e) □ Array or type(r) = T[] where T is incompatible with B)
throw CastException

p[i]=r

if (p==null) throw NullPointerException
if (type(r) □ Array or type(r) = T[] where T is incompatible with A)
throw IncompatibleAssignmentException

Exercise: mark static and dynamic checks that fail

For each check, write a stmt that could go wrong if check was ignored:

```
class A  { int a; }    class B extends A { int b; }

A[] a = new A[2];              B[] b = new B[2];
a[1] = new A();
a = b;
a[1] = new A();
```

Dynamic checks for arrays (revisited)

What invariants do dynamic checks enforce?

- same as the static check

What do we check dynamically? Which ops need checks?

p[i]=r CORRECT SOLUTION assume p is type A[]

```
if (p==null)  throw NullPointerException
if (type(r) is incompatible with A)
   throw IncompatibleAssignmentException
```

How about array bounds checks?

- We also need to insert checks for A[e]
  - at both reads and writes
  - requires more meta data to carry with arrays

Fully static typing of array accesses: possible?

One way to eliminate dynamic array checks is by providing the range of index variables in the type

```
int(0,10) i      // type of i guarantees that i ranges over 0..10
int[20] A        //  array of ints; index range is 0..19
for i in i.range do:  A[i] += 1  // value of i is in bounds of array A
```

Bounds check removed!

- But programming became clumsy.

Can types remove all checks?

- how about "int(0,10) i; while (…) { i := i+1 }" ?
- we can't statically determine the value of i from its type alone
- hence we need a run-time check

Implementic downcasts in multiple inheritance

- Example

  C extends A, B { … }

  C c = new C();

  B b = c;

  C c2 = (C) b;

- How to implement the cast expression?

Back slides for section discussion

Implementation of interfaces in Java

List of semantic checks for an OO language

- Get this from an old project

Dynamic checks for arrays

What invariants do dynamic checks enforce?

- same as the static checks, actually

What do we check dynamically? Which ops need checks?

(B[]) e (cast)

if (type(e) □ Array or type(r) = T[] where T is incompatible with B)

throw CastException

p[i]=r

if (p==null) throw NullPointerException

if (type(r) □ Array or type(r) = T[] where T is incompatible with A)

throw IncompatibleAssignmentException

From dynamic to static typing

JS and Lua:

- objects are dictionaries

Let's fix the class structure (fields and methods)

- construct "class Foo { int field; int method() { return 1;} }
- now we know that instances of Foo always have field, method
- and hence can layout objects as structs, not as dictionaries
- so at new Foo, we create an object that is a struct

Are we done?

- we want to compile p.field efficiently, with a single load instruction
- here, p is declared to be of type "Foo p"