

# CSE 582 – Compilers

Code Shape II – Objects & Classes  
Hal Perkins  
Autumn 2002

11/12/2002 © 2002 Hal Perkins & UW CSE L-1

---

---

---


---

---

---

---

---



# Agenda for Today

- n Object representation and layout
- n Field access
- n What is this?
- n Object creation - new
- n Method calls
  - n Dynamic dispatch
  - n Method tables
  - n Super
- n Runtime type information

11/12/2002 © 2002 Hal Perkins & UW CSE L-2

---

---

---


---

---

---

---

---



# OverLoading and Hiding

```
class One {
  int tag;
  int it;
  void setTag() { tag = 1; }
  int getTag() { return tag; }
  void setIt(int it) {this.it = it;}
  int getIt() { return it; }
}

class Two extends One {
  int it;
  void setTag() {
    tag = 2; it = 3;
  }
  int getThat() { return it; }
  void resetIt() {
    super.setIt(42);
  }
}
```

11/12/2002 © 2002 Hal Perkins & UW CSE L-3

---

---

---

---

---

---

---

---



## What does this program print?

```

class One {
    int tag;
    int it;
    void setTag() { tag = 1; }
    int getTag() { return tag; }
    void setIt(int it) { this.it = it; }
    int getIt() { return it; }
}
class Two extends One {
    int it;
    void setTag() {
        tag = 2; it = 3;
    }
    int getThat() { return it; }
    void resetIt() { super.setIt(42); }
}

public static void main(String[] args) {
    Two two = new Two();
    One one = two;
    one.setTag();
    System.out.println(one.getTag());
    one.setIt(17);
    two.setTag();
    System.out.println(two.getIt());
    System.out.println(two.getThat());
    two.resetIt();
    System.out.println(two.getIt());
    System.out.println(two.getThat());
}

```

11/12/2002

© 2002 Hal Perkins & UW CSE

L-4

---

---

---

---

---

---

---

---



## Your Answer Here

11/12/2002

© 2002 Hal Perkins & UW CSE

L-5

---

---

---

---

---

---

---

---



## Object Representation

- n The naïve explanation is that an object contains
  - n Fields declared in its class and in all superclasses
    - n Redeclaration of a field hides superclass instance
  - n Methods declared in its class and in all superclasses
    - n Redeclaration of a method overrides (replaces)
      - n But overridden methods can still be accessed by super....
- n When a method is called, the method inside that particular object is called
  - n But we don't want to really implement it this way
    - n – we only want one copy of each method's code

11/12/2002

© 2002 Hal Perkins & UW CSE

L-6

---

---

---


---

---

---

---

---



## Actual representation

- n Each object contains
  - n An entry for each field (variable)
  - n A pointer to a runtime data structure describing the class
    - n Key component: method dispatch table
- n Basically a C/C++ struct
- n Fields hidden by declarations in extended classes are *still* allocated in the object and are accessible from superclass methods

11/12/2002 © 2002 Hal Perkins & UW CSE L-7

---

---

---


---

---

---

---

---



## Method Dispatch Tables

- n Often known as "vtables"
- n One pointer per method
- n Offsets fixed at compile time
- n One instance of this per class, not per object

11/12/2002 © 2002 Hal Perkins & UW CSE L-8

---

---

---


---

---

---

---

---



## Method Tables and Inheritance

- n Simple implementation
  - n Method table for extended class has pointers to methods declared in it
  - n Method table also contains a pointer to parent class method table
  - n Method dispatch
    - n Look in current table and use it if local
    - n Look in parent class table if not local
    - n Repeat
  - n Actually used in some dynamic systems

11/12/2002 © 2002 Hal Perkins & UW CSE L-9

---

---

---


---

---

---

---

---



## O(1) Method Dispatch

- n Idea: First part of method table for extended class has pointers in same order as parent class
  - n *BUT* pointers actually refer to overriding methods
  - n ∴ Method dispatch is indirect using fixed offsets known at compile time – O(1)
    - n In C: `*(object->vtbl[offset])(parameters)`
- n Pointers to additional methods in extended class are included following inherited/overridden ones

11/12/2002 © 2002 Hal Perkins & UW CSE L-10

---

---

---


---

---

---

---

---



## Method Dispatch Footnotes

- n Still want pointer to parent class method table for other purposes
  - n Casts and instanceof
- n Multiple inheritance requires more complex mechanisms
  - n Also multiple interfaces in perverse situations(!)

11/12/2002 © 2002 Hal Perkins & UW CSE L-11

---

---

---


---

---

---

---

---



This slide intentionally left blank

11/12/2002 © 2002 Hal Perkins & UW CSE L-12

---

---

---


---

---

---

---

---



## Perverse Example Revisited

```

class One {
    int tag;
    int it;
    void setTag() { tag = 1; }
    int getTag() { return tag; }
    void setIt(int it) { this.it = it; }
    int getIt() { return it; }
}
class Two extends One {
    int it;
    void setTag() {
        tag = 2; it = 3;
    }
    int getThat() { return it; }
    void resetIt() { super.setIt(42); }
}

public static void main(String[] args) {
    Two two = new Two();
    One one = two;
    one.setTag();
    System.out.println(one.getTag());
    one.setIt(17);
    two.setTag();
    System.out.println(two.getIt());
    System.out.println(two.getThat());
    two.resetIt();
    System.out.println(two.getIt());
    System.out.println(two.getThat());
}

```

11/12/2002 © 2002 Hal Perkins & UW CSE L-13

---

---

---


---

---

---

---

---



## Implementation & Trace

11/12/2002 © 2002 Hal Perkins & UW CSE L-14

---

---

---


---

---

---

---

---



## Now What?

- Need to explore
  - Object layout in memory
  - Compiling field references
    - Implicit and explicit use of "this"
  - Representation of vtables
  - Object creation – new
  - Code for dynamic dispatch
    - Including implementing "super.f"
  - Runtime type information – instanceof and casts

11/12/2002 © 2002 Hal Perkins & UW CSE L-15

---

---

---


---

---

---

---

---



## Object Layout

- n Typically, allocate fields sequentially
- n Follow processor/OS alignment conventions if appropriate
- n Use first 32 bits of object for pointer to method table
- n Objects are allocated on the heap
  - n No actual representation in the generated code

11/12/2002 © 2002 Hal Perkins & UW CSE L-16

---

---

---


---

---

---

---

---



## Local Variable Field Access

- n Source
 

```
int n = obj.fld;
```
- n X86
  - n Assuming that obj is a local variable in the current method
 

```
mov  eax,[ebp+offset_obj] ; load obj
mov  eax,[eax+offset_fld] ; load fld
mov  [ebp+offset_n],eax   ; store n
```

11/12/2002 © 2002 Hal Perkins & UW CSE L-17

---

---

---


---

---

---

---

---



## Local Fields

- n A method can refer to fields in the receiving object either explicitly as "this.f" or implicitly as "f"
  - n Both compile to the same code – an implicit "this." is inserted if not present
- n Mechanism: a reference to the current object is an implicit parameter to every method
  - n Can be in a register or on the stack

11/12/2002 © 2002 Hal Perkins & UW CSE L-18

---

---

---

---

---

---

---

---

## Source Level View

- When you write
 

```
void setIt(int it) {
    this.it = it;
}
...
obj.setIt(42);
```
- You really get
 

```
void setIt(ObjType this,
           int it) {
    this.it = it;
}
...
obj.setIt(obj,42)
```

11/12/2002 © 2002 Hal Perkins & UW CSE L-19

---

---

---

---

---

---

---

---

## x86 Conventions

- ecx is traditionally used as "this"
- Add to method call
  - `mov ecx,receivingObject; ptr to object`
  - Do this after arguments are evaluated and pushed, right before dynamic dispatch code (more about that to come)
  - Need to save ecx in a temporary in methods that call other non-static methods
    - One possibility: add to prologue
    - Following examples aren't careful about this

11/12/2002 © 2002 Hal Perkins & UW CSE L-20

---

---

---

---

---

---

---

---

## x86 Local Field Access

- Source
 

```
int n = fld; or int n = this.fld;
```
- X86
 

```
mov  eax,[ecx+offsetfld] ; load fld
mov  [ebp+offsetn],eax   ; store n
```

11/12/2002 © 2002 Hal Perkins & UW CSE L-21

---

---

---

---

---

---

---

---

## x86 Method Tables (vtables)

- These are generated in the assembly language source program
- Need to pick a naming convention for method labels; suggestion:
  - For methods, classname\$methodname
    - Need something more sophisticated to implement overloading
  - For the vtables themselves, classname\$\$
- First method table entry points to superclass table
- Also useful: second entry points to constructor
  - Makes implementation of super() particularly simple

11/12/2002

© 2002 Hal Perkins & UW CSE

L-22

---

---

---

---

---

---

---

---

## Method Tables For Perverse Example

```
class One {
  void setTag() { ... }
  int getTag() { ... }
  void setIt(int it) { ... }
  int getIt() { ... }
}

class Two extends One {
  void setTag() { ... }
  int getThat() { ... }
  void resetIt() { ... }
}

.data
One$$ dd 0 ; no superclass
      dd One$One
      dd One$setTag
      dd One$getTag
      dd One$setIt
      dd One$getIt
Two$$ dd One$$ ; parent
      dd Two$Two
      dd Two$setTag
      dd One$getTag
      dd One$setIt
      dd One$getIt
      dd Two$getThat
      dd Two$resetIt
```

11/12/2002

© 2002 Hal Perkins & UW CSE

L-23

---

---

---

---

---

---

---

---

## Method Table Footnotes

- Key point: First four non-constructor method entries in Two's method table are pointers to methods declared in One in *exactly the same order*
  - ∴ Compiler knows correct offset for a particular method *regardless of whether that method is overridden*

11/12/2002

© 2002 Hal Perkins & UW CSE

L-24

---

---

---


---

---

---

---

---



## Object Creation – new

- n Steps needed
  - n Call storage manager (malloc or similar) to get the raw bits
  - n Store pointer to method table in the first 4 bytes of the object
  - n Call the constructor
  - n Result of new is pointer to the constructed object

11/12/2002 © 2002 Hal Perkins & UW CSE L-25

---

---

---


---

---

---

---

---



## Object Creation

- n Source
 

```
One one = new One(...);
```
- n X86
 

```

push  nBytesNeeded      ; obj size + 4
call  mallocEquiv       ; addr of bits returned in eax
add   esp,4             ; pop nBytesNeeded
lea   edx,One$$         ; get method table address
mov   [eax],edx         ; store at beginning of object
mov   ecx,eax           ; set up "this" for constructor
push  ecx               ; save ecx (ctr might clobber it)
<push constructor arguments> ; arguments (if needed)
call  One$One           ; call constructor
<pop constructor arguments>  ; (if needed)
pop   eax               ; recover ptr to object
mov   [ebp+offset_one],eax ; store n
      
```

11/12/2002 © 2002 Hal Perkins & UW CSE L-26

---

---

---


---

---

---

---

---



## Constructor

- n Only special issue here is generating call to superclass constructor
  - n Same issues as super.method(...) calls – defer for now

11/12/2002 © 2002 Hal Perkins & UW CSE L-27

---

---

---


---

---

---

---

---



## Method Calls

- n Steps needed
  - n Push arguments as usual
  - n Put pointer to object in ecx (this)
  - n Get pointer to method table from first 4 bytes of object
  - n Jump indirectly through method table
  - n Restore ecx to point to current object (if needed)
    - n Useful hack: push it in the function prologue so it is always on the stack in a known location

11/12/2002 © 2002 Hal Perkins & UW CSE L-28

---

---

---


---

---

---

---

---



## Method Call

- n Source
 

```
obj.meth(...);
```
- n X86
 

```
<push arguments from right to left> ; (if needed)
mov ecx,[ebp+offset_obj] ; get pointer to object
mov eax,[ecx] ; get pointer to method table
call dword ptr [eax+offset_meth] ; call indirect via method tbl
<pop arguments> ; (if needed)
mov ecx,[ebp+offset_ecxtemp] ; (if needed)
```

11/12/2002 © 2002 Hal Perkins & UW CSE L-29

---

---

---


---

---

---

---

---



## Handling super

- n Almost the same as a regular method call with one extra level of indirection
- n Source
 

```
super.meth(...);
```
- n X86
 

```
<push arguments from right to left> ; (if needed)
mov ecx,[ebp+offset_obj] ; get pointer to
```

11/12/2002 © 2002 Hal Perkins & UW CSE L-30

---

---

---


---

---

---

---

---



## Runtime Type Checking

- n Use the method table for the class as a “runtime representation” of the class
- n The test for “o instanceof C” is
  - n Is o’s method table pointer == &C\$?
  - n Recursively, get the superclass’s method table pointer from the method table and check that
  - n Stop when you reach Object (or a null pointer, depending on how you represent things)

11/12/2002 © 2002 Hal Perkins & UW CSE L-31

---

---

---


---

---

---

---

---



## Coming Attractions

- n Code Generation
- n Two models
  - n Simple tree walk, which is adequate to complete the project
  - n More standard instruction selection/scheduling/register allocation regime
- n Rest of the quarter – survey of optimization plus special topics

11/12/2002 © 2002 Hal Perkins & UW CSE L-32

---

---

---

---

---

---

---

---