Announcements

• HW5: Oat v.2
  • Due Tuesday Nov 19
  • Files will be released on Canvas Saturday 12am
  • If you have submitted HW4 and want HW5 files now, email cs153-staff@seas.harvard.edu
    • We will email you a link to the files

• Guest lecturer Tuesday Nov 5: Eliza Kozyri
  • Steve away
Today

- Object Oriented programming ctd.
  - Dynamic dispatch
  - Code generation for methods and method calls
  - Fields
  - Creating objects
  - Extensions
  - Type system
Need for Dynamic Dispatch

• Methods look like functions. Can they be treated the same?
• Consider the following Java code: Same interface implemented by multiple classes

```
interface IntSet {
    public IntSet insert(int i);
    public boolean has(int i);
    public int size();
}

class IntSet1 implements IntSet {
    private List<Integer> rep;
    public IntSet1() {
        rep = new LinkedList<Integer>();
    }
    public IntSet1 insert(int i) {
        rep.add(new Integer(i));
        return this;
    }
    public boolean has(int i) {
        return rep.contains(new Integer(i));
    }
    public int size() {return rep.size();}
}

class IntSet2 implements IntSet {
    private Tree rep;
    private int size;
    public IntSet2() {
        rep = new Leaf(); size = 0;
    }
    public IntSet2 insert(int i) {
        Tree nrep = rep.insert(i);
        if (nrep != rep) {
            rep = nrep; size += 1;
        }
        return this;
    }
    public boolean has(int i) {
        return rep.find(i);
    }
    public int size() {return size;}
}
Need for Dynamic Dispatch

interface IntSet {
    public IntSet insert(int i);
    public boolean has(int i);
    public int size();
}

• Suppose a client uses the IntSet interface

```java
IntSet set = foo();
int x = set.size();
```

• Which code to call?
  • IntSet1.size? IntSet2.size?

• Client code doesn’t know which code! Could be either at runtime.
  • Objects must “know” which code to call
  • Invocation of method must indirect through object
Dynamic Dispatch Solution

• So we need some way at run time to figure out which code to invoke

• Solution: dispatch table (aka virtual method table, vtable)
  • Each class has table (array) of function pointers
  • Each method of class is at a known index of table

```cpp
IntSet set = foo();
int x = set.size();
```
What Offset Into the VTable?

• Want to make sure that every object of class B has same layout of dispatch table
  • Even if object is actually a subclass of B!

```java
class A {
  void foo() { ... }
}
class B extends A {
  void bar() { ... }
  void baz() { ... }
}
class C extends B {
  void foo() { ... }
  void baz() { ... }
  void quux() { ... }
}
```

• List methods in order
• Ensures that a dispatch table for class C also looks like a dispatch table for class B, and for class A
**Dispatch Tables**

<table>
<thead>
<tr>
<th>Dispatch table for class A</th>
<th>Dispatch table for class B</th>
<th>Dispatch table for class C</th>
</tr>
</thead>
<tbody>
<tr>
<td>&amp;A.foo</td>
<td>&amp;A.foo</td>
<td>&amp;C.foo</td>
</tr>
<tr>
<td></td>
<td>&amp;B.bar</td>
<td>&amp;B.bar</td>
</tr>
<tr>
<td></td>
<td>&amp;B.baz</td>
<td>&amp;C.baz</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&amp;C.quux</td>
</tr>
</tbody>
</table>

- **Dispatch table for class C looks like a dispatch table for class B**
  - i.e., address for method `foo` is at index 0 (offset 0 bytes)
  - address for method `bar` is at index 1 (offset 4 bytes)
  - address for method `baz` is at index 2 (offset 8 bytes)

- **And it looks like a dispatch table for class A**
  - i.e., address for method `foo` is at index 0
Generating Code for Methods

• For method declarations
  • Compiled just like top-level procedures, but...
  • Methods have implicit argument, the **receiver object** (i.e., `this`, `self`)
  • In essence, method `bar` declared in class `B`
    ```
    class B {
      void bar(int x) { ... }
    }
    ```
    is translated like a function
    ```
    void bar(B this, int x)
    ```

• For method call `o.bar(54)`
  • Essentially: `void (*f)(obj *,int);`
    ```
    f = o->class_ptr->vtable[offset for bar]
    f(o, 54);
    ```
  • i.e., use vtable to get pointer to appropriate function, invoke it with receiver and arguments
Sharing Dispatch Tables

- All instances of a class may share the same dispatch vector
  - Assuming that methods are immutable
- When an object is constructed, the object needs to point to the appropriate dispatch table

```
&A.foo
&B.bar
&B.baz
```

```
Dispatch table for class B
```
Inheritance: Sharing Code

- Inheritance: Method code “copied down” from the superclass
  - If not overridden in the subclass

```
&A.foo
&B.bar
&B.baz
```

```
&C.foo
&B.bar
&C.baz
&C.quux
```
Fields

• Same basic idea for fields as for methods!

```java
interface Point { int getx(); float norm(); }

class 2DPoint implements Point {
    int x;
    int y;
    ...
}

class 3DPoint implements Point {
    int z;
    ...
}
```

• Representation of object of class 3DPoint has space to store fields of 3DPoint and superclasses
Generating Code for Field Accesses

- To access field \texttt{x.f}:
  - \texttt{x} will be represented as pointer to object.
  - Need to know (static) type of \texttt{x}.
  - \texttt{x.f} refers to memory location at appropriate offset from base of object \texttt{x}.
- E.g., reading \texttt{o.y} would translate to dereferencing address \texttt{o+ (offset for y)}.
Creating Objects

- `new C` creates a new object of class `C`
  - Creates record big enough to hold a `C` object
  - Initializes pointer to dispatch table
  - Initializes instance variables
  - Evaluates to pointer to newly created object
Representation in LLVM

- During typechecking, create a class hierarchy
  - (We will discuss typechecking more later)
  - Map each class to its interface
    - Superclass
    - Constructor type
    - Fields
      - Method types (plus whether they inherit and from where)

- Compile the class hierarchy to produce
  - An LLVM IR struct type for each object instance
  - An LLVM IR struct type for each dispatch table
  - Global definitions that implement the class tables
Extensions...

- **Multiple inheritance**
  - Typically use multiple vtables (one for each base class) and switch between them based on the static type
  - Other approaches possible

- **Separate compilation**
  - Don’t know how many fields/method in superclass! (Superclass could be recompiled after subclass)
  - Resolve offsets at link or load time
Extensions...

• Prototype based OO languages
  • Similar approach, but vtable belongs with object (no classes!)
  • Objects are created by cloning other objects
  • Many objects will have the same vtable: can share them, with copy-on-write

• Runtime type check: o instanceof C
  • Each object contains pointer to its class, so can figure out at runtime if a o’s class is a subclass of C
  • But how to efficiently store inheritance information in runtime representation of classes?
• Visibility

  • To support encapsulation, some OO languages provide visibility restrictions on fields and methods
  • Java has private, protected, public (and some more)
    • private members accessible only to implementation of class
    • public members accessible by any code
    • protected members accessible only to implementation of class and subclasses

• Subclassing vs inheritance

  • Somewhat conflated in Java
  • Inheritance: reuse code from another class; Subclassing: every object of subclass is a superclass object
  • C++ has visibility restrictions on inheritance
OO Type Systems

- Subclassing vs subtyping
  - Not the same!
  - No contravariance in argument type in Java methods

- Overriding vs overloading
  - Given \( C.m(T_1, T_2, \ldots, T_n) \) and \( D.m(S_1, S_2, \ldots, S_m) \) where \( C \) is subclass of \( D \),
    \( C.m \) overrides \( D.m \) only if \( T_1, T_2, \ldots, T_n = S_1, S_2, \ldots, S_m \)
    - Otherwise, \( D.m \) just overloads the method name \( m \)...

- Null values
  - In Java type \( C \) for class \( C \) is analogous to \( C \) option in ML
    - Since any object value can be \texttt{null}
  - ...

Stephen Chong, Harvard University