Lecture 10: Part 1: OO Issues

CS 540
George Mason University

Object-Oriented Languages?

What is an OOL?
• A language that supports “object-oriented programming”
• Term is almost meaningless today — Smalltalk to C++ to Java

How does an OOL differ from an imperative language?
• Complex type system including inheritance and polymorphism.
• Object representation issues
• Resolution of names to their implementations

Implementing Object-Oriented Languages

Critical issues in OOL implementation:
• Typechecking
• Object representation – (inheritance)
• Dispatching – Mapping a method invocation name to a method implementation

Last two related to language’s namespace

An object is an abstract data type that encapsulates data, operations and internal state behind a simple, consistent interface.

The Concept:

Elaborating the concepts:
• Each object needs local storage for its attributes
  • Attributes are static (lifetime of object)
  • Access is through methods
• Some methods are public, others are private
• Object’s internal state leads to complex behavior

Data
Code

Data
Code

Data
Code
Typechecking OO Languages

- Subtyping – if B is a subclass of A, an object of type B can be used when an object of type A is expected.
  - parameters – actual subtype of formal
  - assignment – RHS subtype of LHS
  - expressions
- Visibility (private/public/…) – checked at compile time (not runtime).

- Can we do everything at compile time?
- Multiple inheritance?
  - If class C inherits from A and B, should it be usable anywhere either A or B are usable?
  - How to deal with ‘duplicate’ attributes/methods?
  - Typically, there is a linguistic mechanism for specifying what is inherited from each superclass.

Object Representation in OO Languages

- Static, private storage for attributes & instance variables
  - Heap allocate object records or “instances”
- Need consistent, fast access
  - Known, constant offsets
- Provision for initialization in NEW
- Inheritance issues

Simplistic approach: Object Representation

For object x of type A:

Each object gets copies of all attributes and methods
Better approach

For object x of type A:

Objects share methods (and static attributes)

More typically:

For object x of type A:

Objects share methods (and static attributes) via shared class object (can keep counter of objects N)

---

**OOL Object Representation**

**Class variables**
- Static class storage accessible by global name (class C)
  - Method code put at fixed offset from start of class area
  - Static variables and class related bookkeeping

**Object Variables**
- Object storage is heap allocated at object creation
  - Fields at fixed offsets from start of object storage
- Methods
  - Code for methods is stored with the class
  - Methods accessed by offsets from code vector
  - Allows method references inline
  - Method local storage in object (no calls) or on stack

---

**Object Representation: Single Inheritance**

- **Use prefixing of storage for objects**

  Class Point {
  int x, y;
  }

  Class ColorPoint extends Point {
  Color c;
  }

  A method defined for class Point will also work correctly on class ColorPoint
Object Representation: Multiple Inheritance

Suppose we have a class C that inherits from classes A and B. Will prefixing work?

<table>
<thead>
<tr>
<th>class pointer</th>
<th>A data</th>
<th>B data</th>
<th>C data</th>
</tr>
</thead>
</table>

Problem with B’s methods

• Want a data layout that will work correctly with methods of all three classes.

Various techniques have been proposed (and used).

– Modify B’s representation to include a buffer same size as A’s data
– Record offsets to be used at runtime (offsets for A and C are 0)
– …

Object-Oriented Languages - Dispatching

Mapping message names to methods

• Static mapping, known at compile-time (Java, C++)
  – Fixed offsets & indirect calls
• Dynamic mapping, unknown until run-time (Smalltalk, C++ with pointers)
  – Build a table of function pointers (added level of indirection). Look up name in class’ table of methods (can use hashing)
• Use a standard invocation sequence except that need to:
  – pass receiver’s object record (self, this, …)
  – method needs access to its class
  – block-style scoping – no static (access) link used