Support for Object-oriented Languages

SLIDES ARE A SELECTION AND SIZEABLE MODIFICATION FROM THE ONES MADE AVAILABLE BY THE AUTHORS OF THE BOOK.

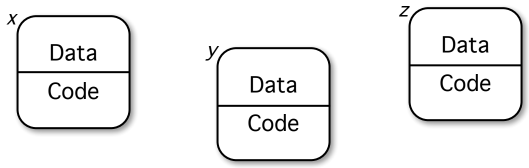
OO: Objects

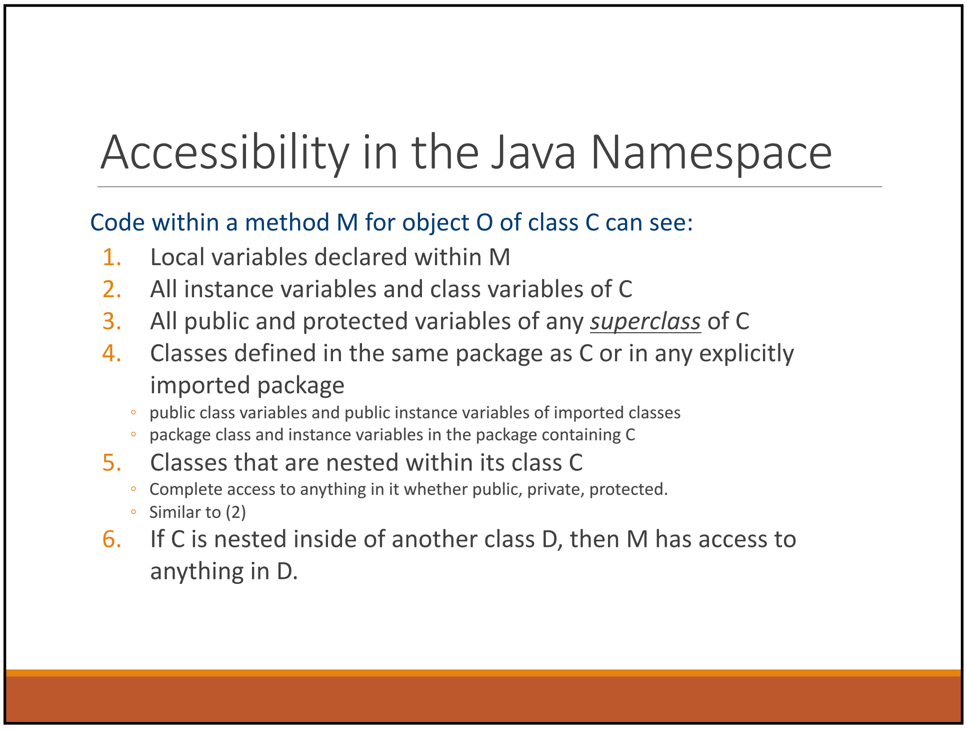
Each object has an internal state

- Data members
- External access is typically through code members

Each object has a set of associated procedures, or methods

Access to classes, methods and fields can be restricted through **`private`** and **`protected`**.

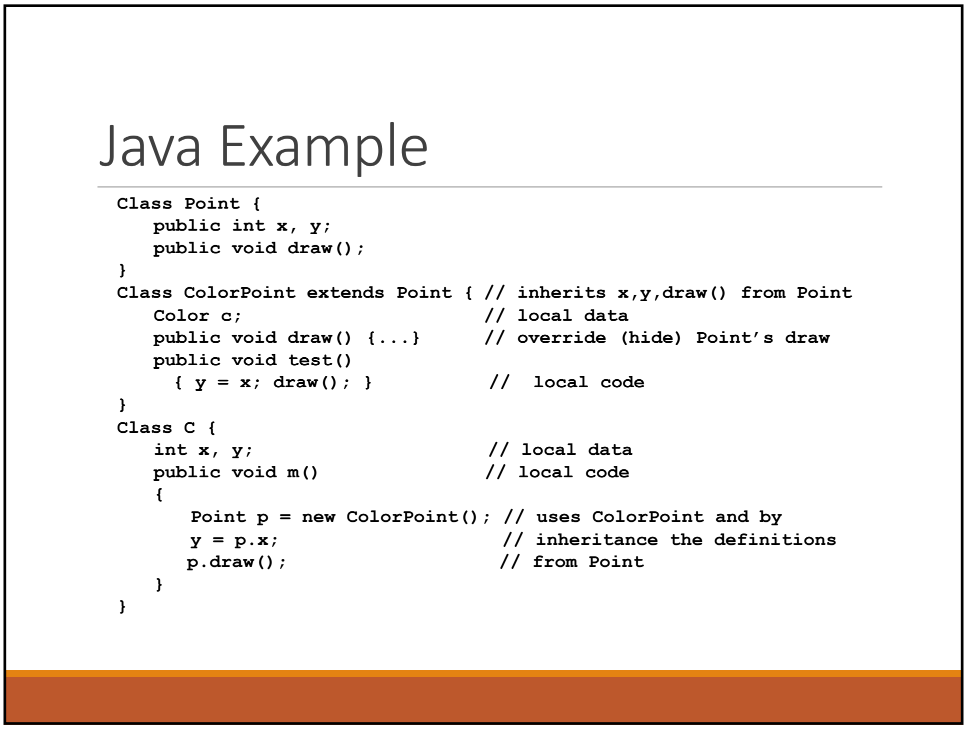




Accessibility in the Java Namespace

Code within a method M for object O of class C can see:

1. Local variables declared within M
2. All instance variables and class variables of C
3. All public and protected variables of any *superclass* of C
4. Classes defined in the same package as C or in any explicitly imported package
   - public class variables and public instance variables of imported classes
   - package class and instance variables in the package containing C
5. Classes that are nested within its class C
   - Complete access to anything in it whether public, private, protected.
   - Similar to (2)
6. If C is nested inside of another class D, then M has access to anything in D.



Java Example

```
Class Point {
   public int x, y;
   public void draw();
}
Class ColorPoint extends Point { // inherits x,y,draw() from Point
   Color c;                     // local data
   public void draw() {...}     // override (hide) Point's draw
   public void test()
     { y = x; draw(); }          //  local code
}
Class C {
   int x, y;                     // local data
   public void m()              // local code
   {
      Point p = new ColorPoint(); // uses ColorPoint and by
      y = p.x;                    // inheritance the definitions
      p.draw();                   // from Point
   }
}
```

OO Symbol Tables

To compile method M of object O in class C, the compiler needs:

Lexically scoped symbol table for the current block and its surrounding scopes

- Just like non-OO languages, inner declarations hide outer declarations

Chain of symbol tables for inheritance

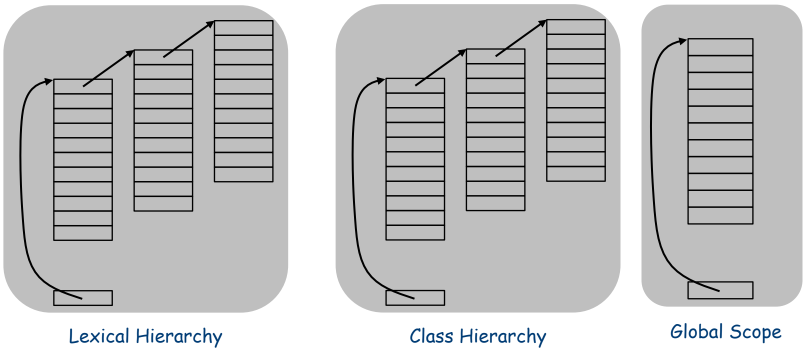
- Class C and all of its superclasses
- Need to find methods and instance variables in any superclass

Symbol tables for all global classes (package scope)

- Entries for all members with visibility
- Need to construct symbol tables for imported packages and link them into the structure in appropriate places

OO Symbol Tables

Conceptually



Search Order: lexical, class, global

Java Symbol Tables

To find the address for a reference to *x* in method M for an object O of class C, the compiler must:

For an unqualified use (i.e., x):

- Search the symbol table for the method’s lexical hierarchy
- Search the symbol tables for the receiver’s class hierarchy
- Search global symbol table (current package and imported)
- In each case check visibility attribute of x

For a qualified use (i.e.: Q.x):

- Find Q by the method above
- Search from Q for x
  - Must be a class or instance variable of Q or some class it extends
- Check visibility attribute of x

Runtime Structures for OOLs

Object lifetimes are independent

Each object needs an object record (OR) to hold its state

- Independent allocation and deallocation

Classes are treated as objects too

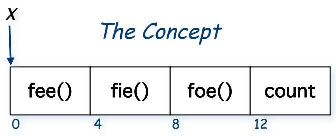
- ORs of classes instantiate the class hierarchy

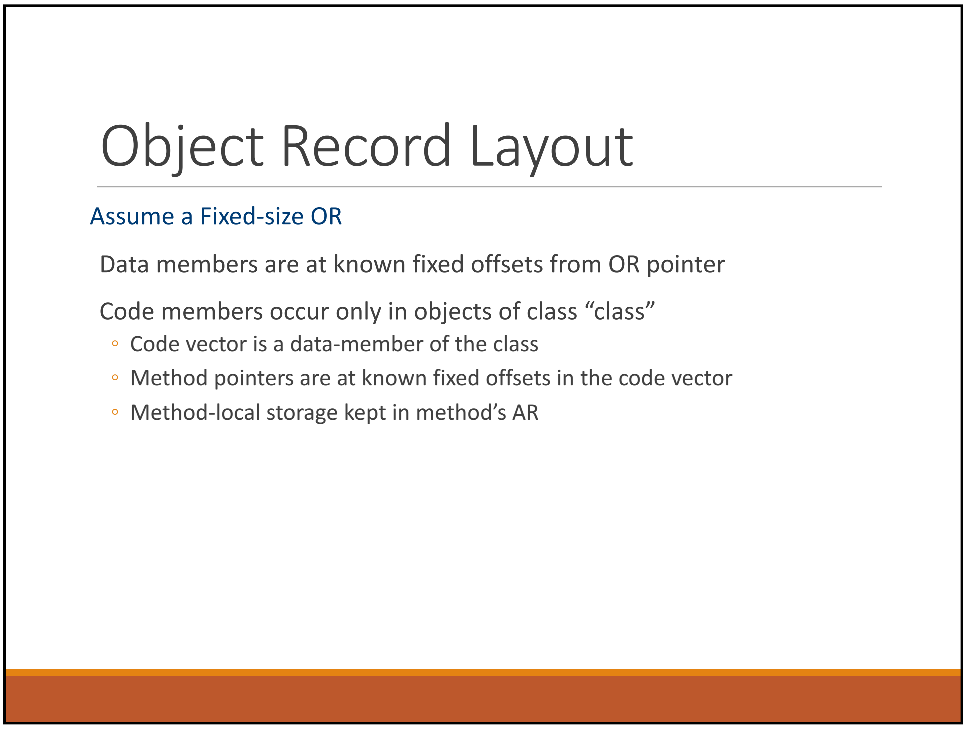
Object Records

Static private storage for members

Need fast, consistent access

- Known constant offsets from OR pointer





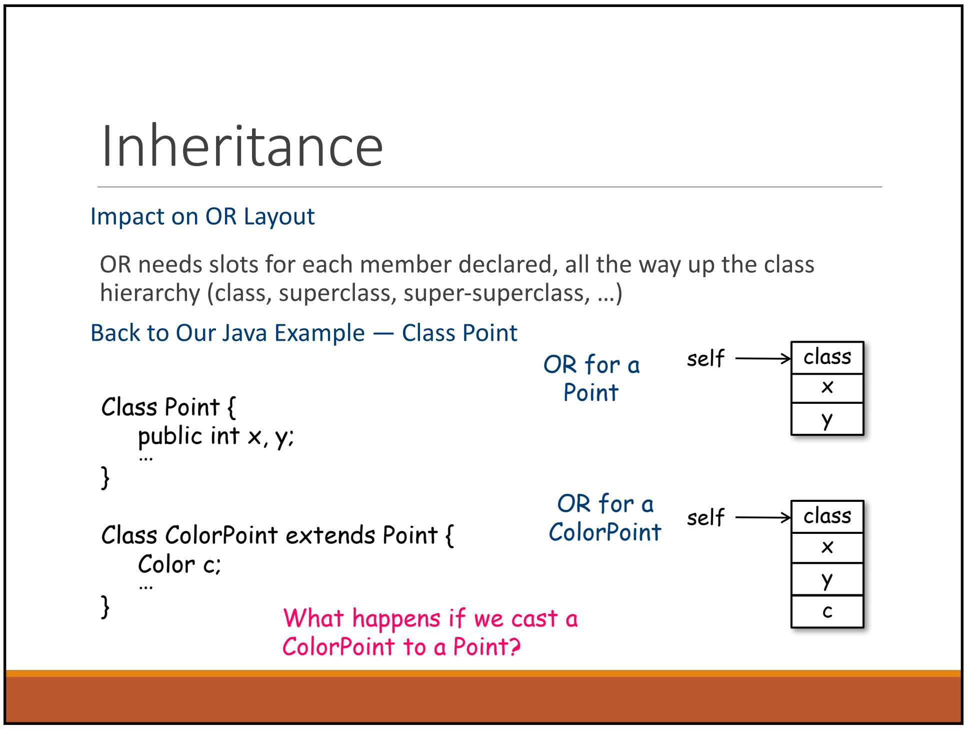
Object Record Layout

Assume a Fixed-size OR

Data members are at known fixed offsets from OR pointer

Code members occur only in objects of class “class”

- Code vector is a data-member of the class
- Method pointers are at known fixed offsets in the code vector
- Method-local storage kept in method’s AR



Inheritance

Impact on OR Layout

OR needs slots for each member declared, all the way up the class hierarchy (class, superclass, super-superclass, …)

Back to Our Java Example — Class Point

```
Class Point {
     public int x, y;
     …
}

Class ColorPoint extends Point {
     Color c;
     …
}
```

OR for a Point

self →

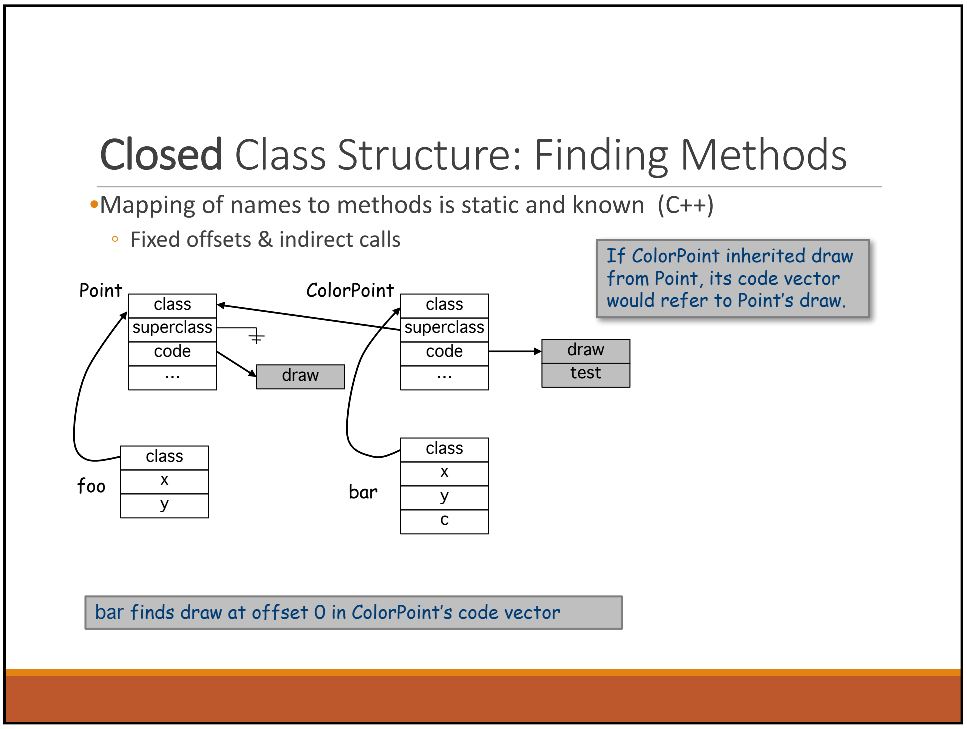
class
x
y

OR for a ColorPoint

self →

class
x
y
c

What happens if we cast a ColorPoint to a Point?

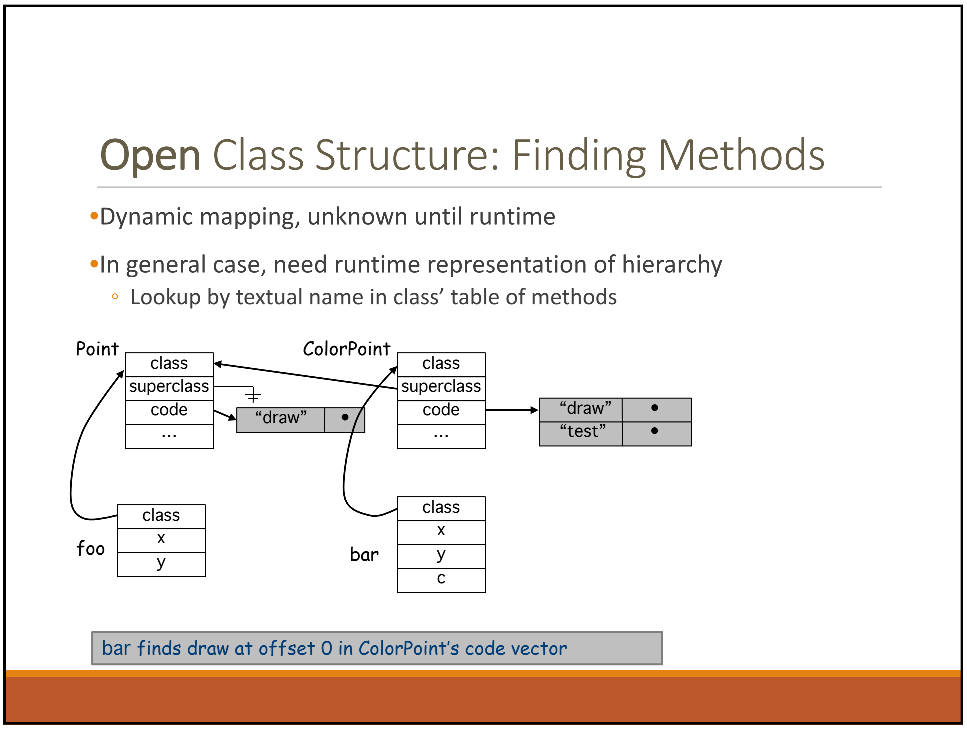


Closed Class Structure: Finding Methods

- Mapping of names to methods is static and known (C++)
  - Fixed offsets & indirect calls

If ColorPoint inherited draw from Point, its code vector would refer to Point's draw.

bar finds draw at offset 0 in ColorPoint's code vector



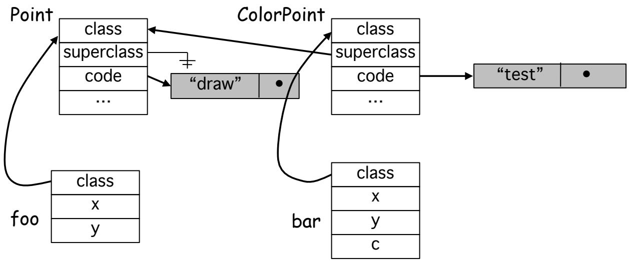
Open Class Structure: Finding Methods

- Dynamic mapping, unknown until runtime
- In general case, need runtime representation of hierarchy
  - Lookup by textual name in class' table of methods

bar finds draw at offset 0 in ColorPoint's code vector

Open Class Structure: Finding Methods

Locating an inherited method.



If ColorPoint inherited draw from Point, its code vector would lack a pointer to draw.

- Perform runtime search through hierarchy
  - This process is expensive
- Use a "method cache" to speed the search
  - Cache holds ‹*search key, class, method pointer*›