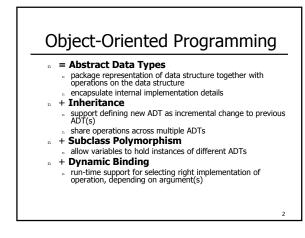
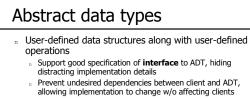
#### Object-Oriented Programming



#### Some OO languages

- <sup>n</sup> Simula 67: the original
- n Smalltalk-80: popularized OO
- <sup>n</sup> C++: OO for the hacking masses
- <sup>n</sup> Java, C#: cleaned up, more portable variants of C++
- <sup>n</sup> CLOS: powerful OO part of Common Lisp
- **.** Self: very pure OO language
- <sup>n</sup> **Cecil**, MultiJava, **EML**: OO languages from my research group
- Emerald, Kaleidoscope: other OO languages from UW



- Allow language to be extended with new types, raising & customizing the level of the language
- ... Called a **class** in OO languages
  - <sup>n</sup> data structures called **objects**, or **instances** of the class
  - ... operations called methods; data called instance variables
- n Modules have similar benefits

#### Inheritance

- Most recognizable aspect of OO languages & programs
- <sup>n</sup> Define new class as *incremental modification* of existing class
  - " new class is **subclass** of the original class (the **superclass**)
  - " by default, **inherit** superclass's methods & instance vars
  - $_{\scriptscriptstyle\rm In}$   $\,$  can add more methods & instance vars in subclass
  - " can **override** (replace) methods in subclass
    - but not instance variables, usually

class Rectangle {
Pointcenter;
intheight, wilth;
intareal ( return height \* wilth; }
voil draw (D utputD evice out) { ... }
voil move Pointnew\_c) {center= new\_c; }
...

Example

}

```
class CobredRectangle extends Rectangle {
 // center, height, & width inherited
 Cobrcobr;
 // area, move, etc. inherited
```

void draw (0 utputD evice out) { ... } //override!

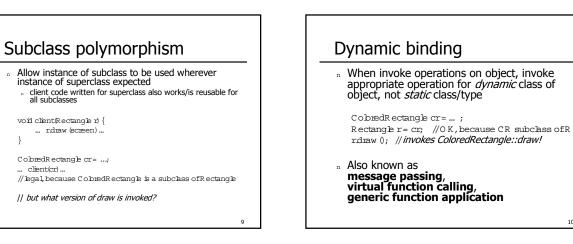
.

#### Benefits of inheritance

- <sup>n</sup> Achieve more code sharing by **factoring** code into common superclass
  - " superclass can be **abstract**
  - no direct instances, just reusable unit of implementation
  - encourages development of rich libraries of related data structures
- n May model real world scenarios well
  - " use classes to model different things
  - use inheritance for classification of things: subclass is a special case of superclass

#### Pitfalls of inheritance

- <sup>n</sup> Inheritance often overused by novices
- <sup>n</sup> Code gets fragmented into small factored pieces
- <sup>n</sup> Simple extension & overriding may be too limited
  - n e.g. exceptions in real-world classification hierarchies



## Method lookup

- n Given a message obj.msg(args) ...
- <sup>n</sup> Start with run-time class C of obj (the receiver)
  - n if msg is defined in C, then invoke it
  - $_{\rm n}\,$  otherwise, recursively search in superclass of  $_{\rm C}\,$
  - " if never find match, report run-time error
  - $\Rightarrow$  type checker guarantees this won't happen

11

#### Dynamic dispatching vs. static overloading

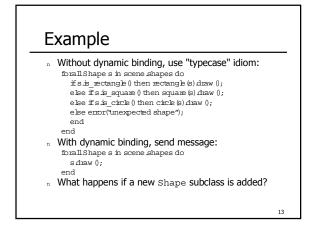
- Like overloading:
  - multiple methods with same name, in different classes use class/type of argument to resolve to desired method

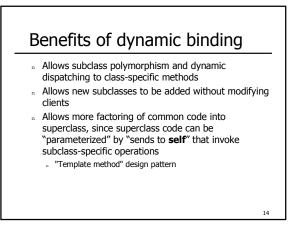
#### Unlike overloading:

- ... resolve using *run-time* class of argument,
- not static class/type ... consider only receiver argument, in most OO languages
- . C++ & Java: regular static overloading on arguments, too
- . CLOS, Cecil, MultiJava: resolve using all arguments (multiple dispatching)

10

12





## Pitfalls of dynamic binding

 Tracing flow of control of code is harder
 control can pop up and down the class hierarchy

#### n Adds run-time overhead

- <sup>n</sup> space for run-time class info
- n time to do method lookup
- but only an array lookup (or equivalent), not a search

15

# Issues in object-oriented language design

- D Object model:
  - hybrid vs. pure OO languages
  - n class-based vs. classless (prototype-based) languages
  - " single inheritance vs. multiple inheritance
- <sup>n</sup> Dispatching model:
  - " single dispatching vs. multiple dispatching
- n Static type checking:
  - n types vs. classes
  - n subtyping
  - " subtype-bounded polymorphism

16