Software Composition Paradigms

Sommersemester 2015

Radu Muschevici

Software Engineering Group, Department of Computer Science

TECHNISCHE
UNIVERSITÄT
DARMSTADT

2015-04-28

Exam Date

The exam date has changed to:

- Wednesday, 15 July 2015, 19.00–21.00
- Rooms S101/A1 and S311/08

Note: The lecture on Tuesday, 7 July 2015 will be used for review and questions (no new content).

Prototype-based Programming

Self: The Movie

Watch the *Self*[1] programming language video:
`https://www.youtube.com/watch?v=0x5P7QyL774`

[1] `http://www.selflanguage.org/`

Prototype-based Programming Languages

Prototype: an original or first model of something from which other forms are copied or developed

[Merriam-Webster Dictionary]

- No classes; only objects
- Any object can be the *prototype* of another object
- Object creation by *copying* the prototype
- Objects define their own properties and methods
- Objects *delegate* behaviour to other objects

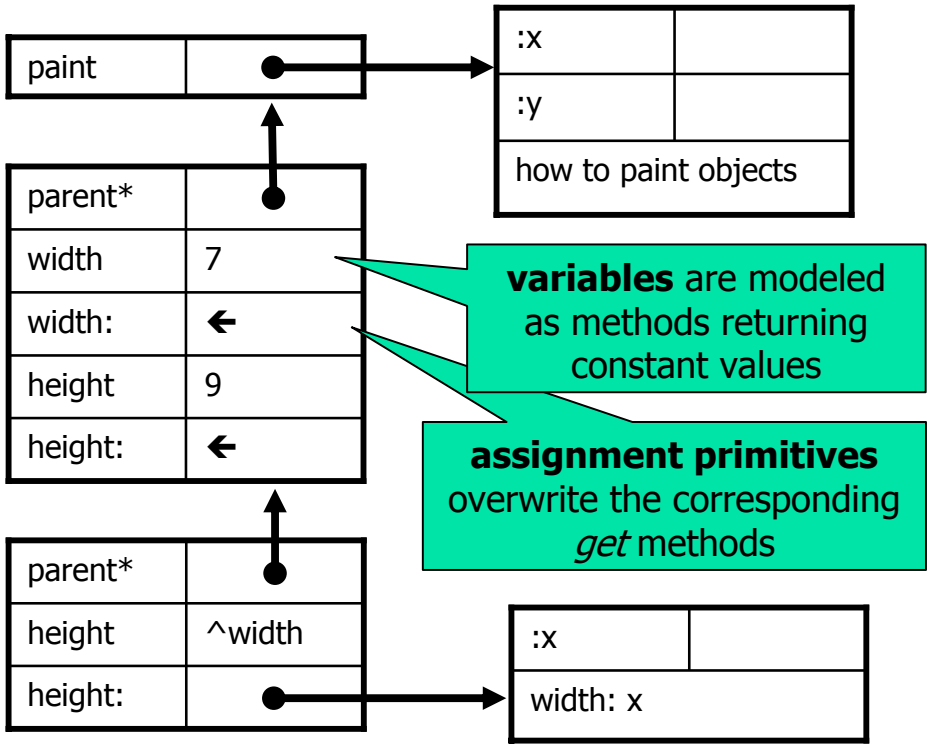
Many languages (but only few used outside research):
JavaScript, Self, Io, NewtonScript, Omega, Cecil, Lua, Object-Lisp, Exemplar, Agora, ACT-I, Kevo, Moostrap, Obliq, Garnet

Design Goals

Simplicity

- Leave out everything that dilutes the paradigm
- *Everything is an object*
- Fewer concepts $\Rightarrow$ simpler programming model, easier to understand, explain and use
- Simpler relationship between objects: object inherits from other object (Compare to class-based, where 1: object is an instance of a class, and 2: object's class is a subclass of another class)

Example: The Object Model of Self



- Every object contains a collection of slots
- Each slot has a name and an object
- Every slot can be marked as parent slot
- A non-method object simply returns itself when invoked as a method
- A method object contains a piece of code that's executed on invocation
- Slots of method objects can be marked as argument slots

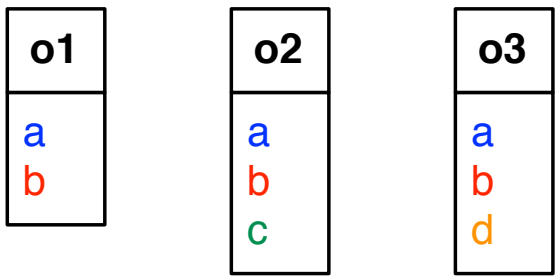
Some Issues with Classes

> "In OO programs, objects send messages to other objects. OO source code shows us…classes inheriting from classes. Oops. There is a complete disconnect in OOP between the source code and the runtime entities."
>
> [O. Nierstrasz, ECOOP 2010]

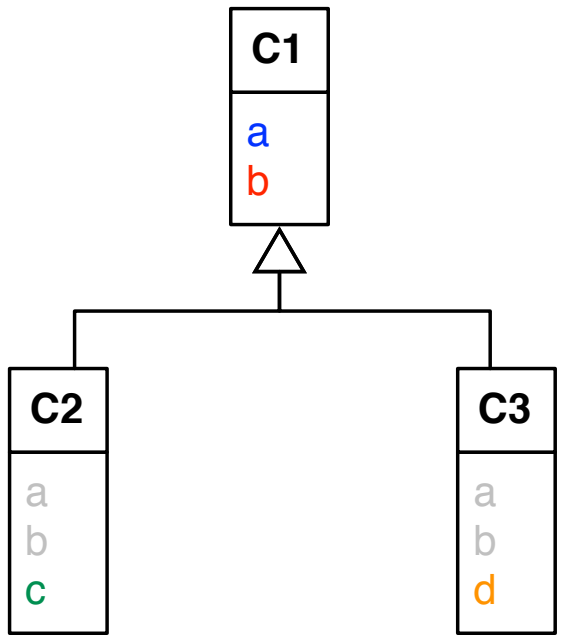
- Classes are abstract ("In the real world, there are only objects")
- Class-based languages force software engineers to design from abstract to concrete
- More natural: from concrete examples to abstract concepts
- Class-based architectures are sometimes seen as too rigid
- Designing a class inheritance hierarchy not always easy

Classification: An easy example

Three objects o1, o2, o3 sharing properties a, b, c, d:



A corresponding class hierarchy:



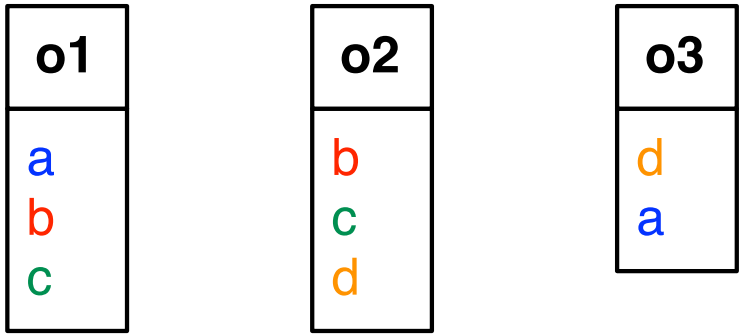

```
o1 = new C1();
o2 = new C2();
o3 = new C3();
```

Family Resemblance: When classification is not easy

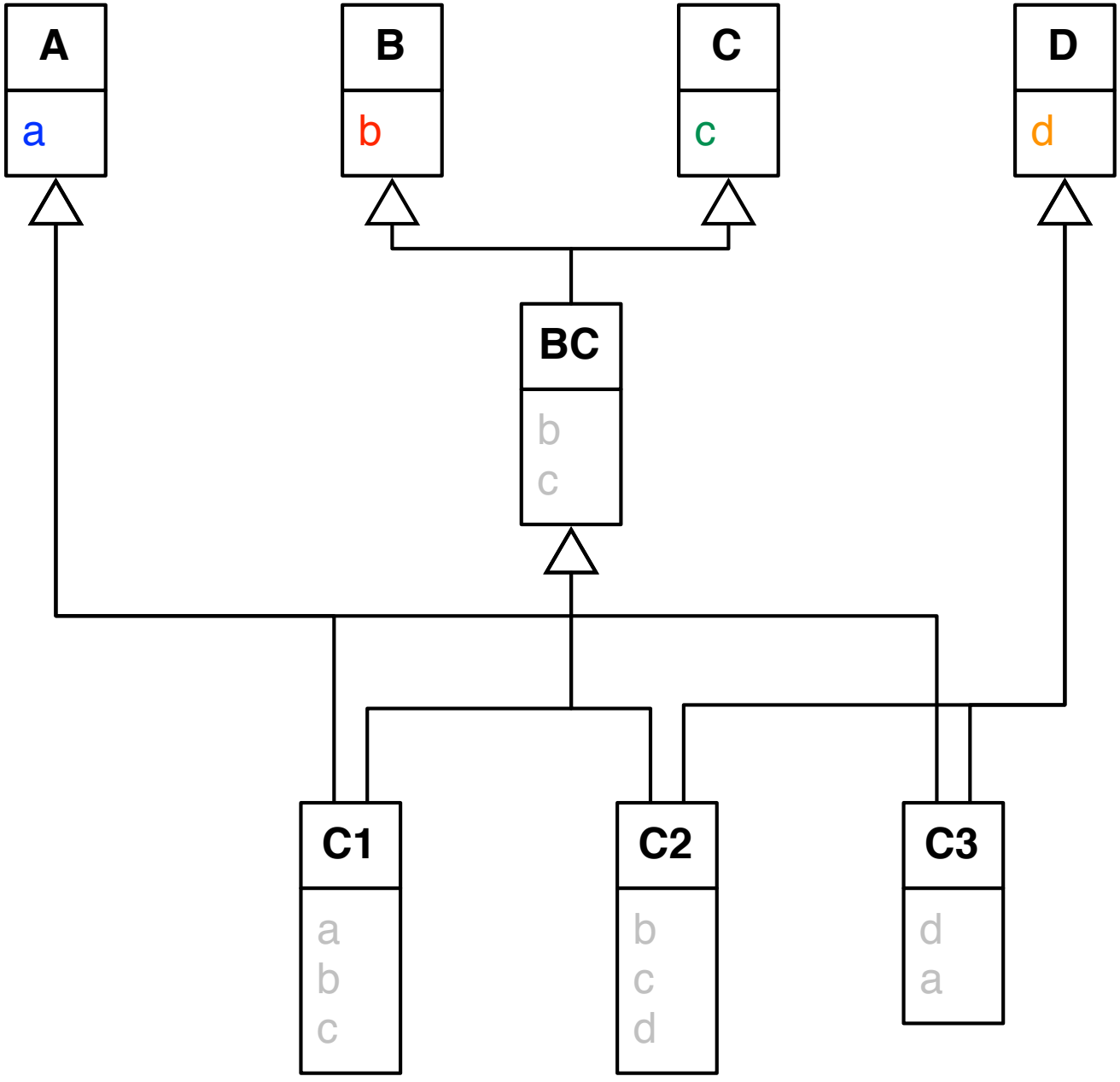
- Philosophical idea made popular by Wittgenstein (1889–1951)
- Observation: some things have no essential features in common, even though they may belong to same category.
- Instead they have a series of overlapping similarities.
- Members of a family are difficult to classify as a hierarchy.
- Examples: works of art, games, …

Family Resemblance: Example

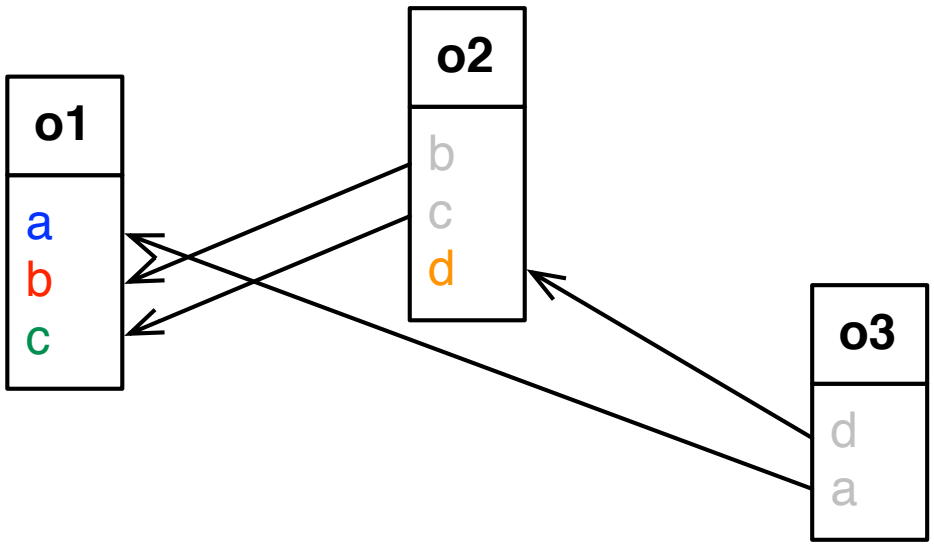
Three objects: o1, o2, o3
sharing some properties: a, b, c, d



Family Resemblance: Classification Attempt



Family Resemblance: Delegation



Prototypes & Object Creation

Prototypes

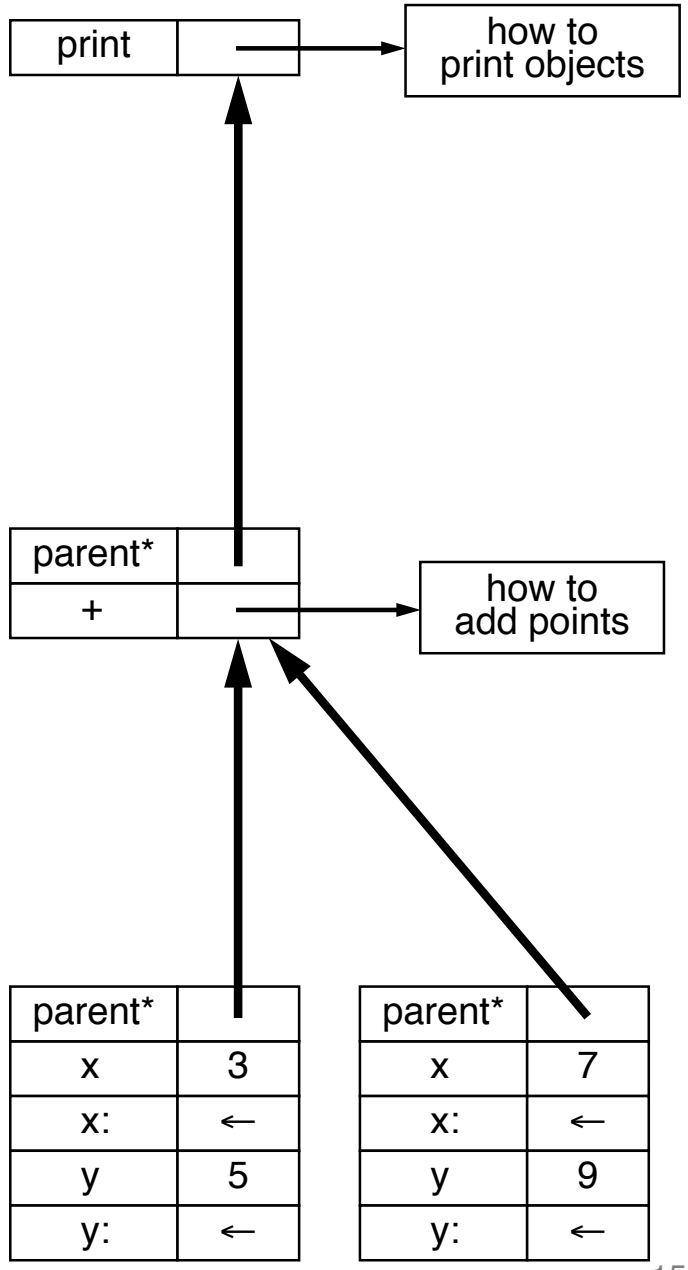
- Any object can be the prototype for another object

Object Creation

- By copying (*cloning*) an existing object (the *prototype*)
- Some languages support object creation *ex nihilo*
- Object is related to other objects through delegation

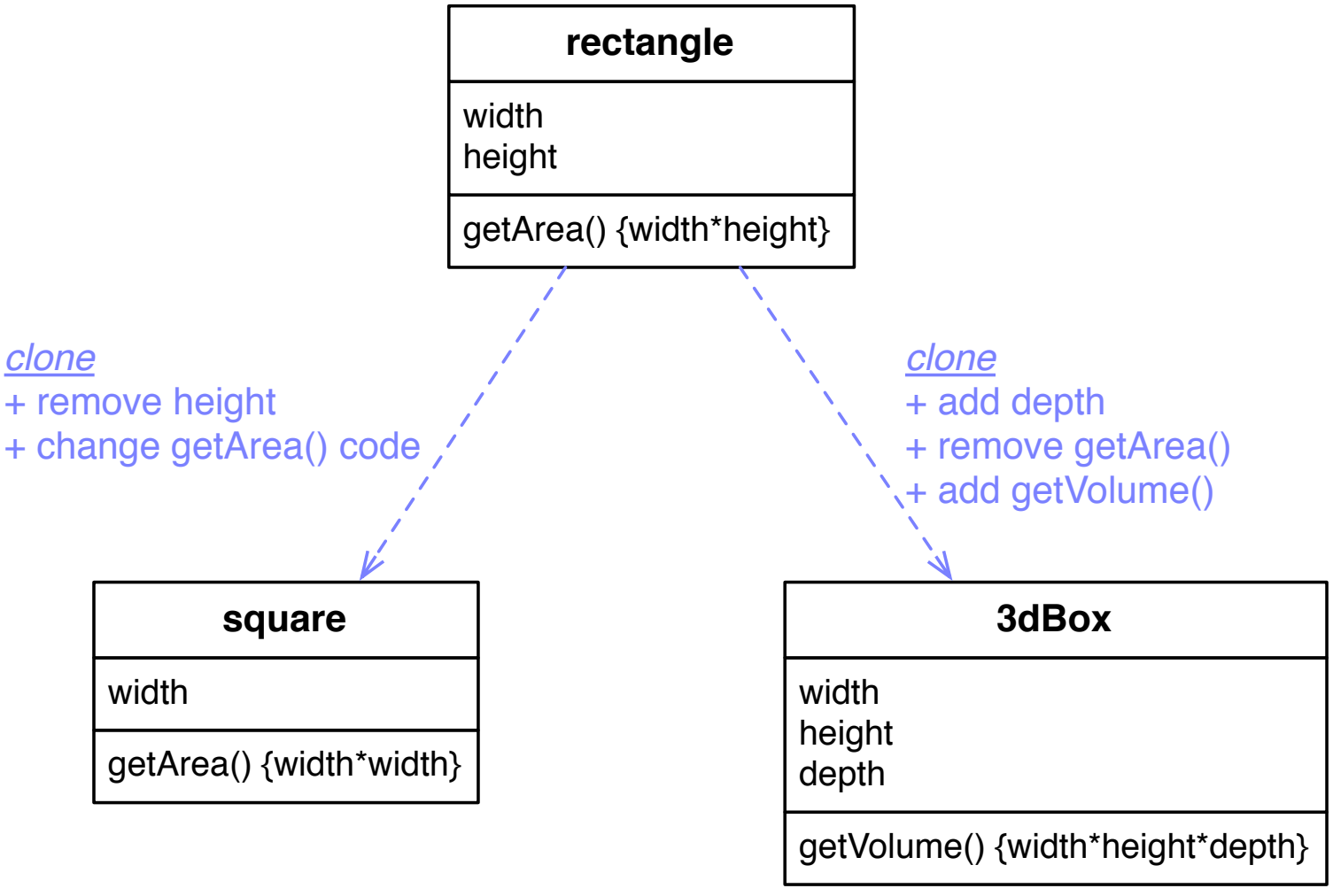
Delegation

- Mechanism to share data and behaviour among objects
- An object has *parent(s)* (sometime the prototype)
- If object receives a message it cannot handle, it delegates it to its parent(s)
- Multiple parents $\Rightarrow$ multiple inheritance (diamond problem etc.)
- More flexible than inheritance



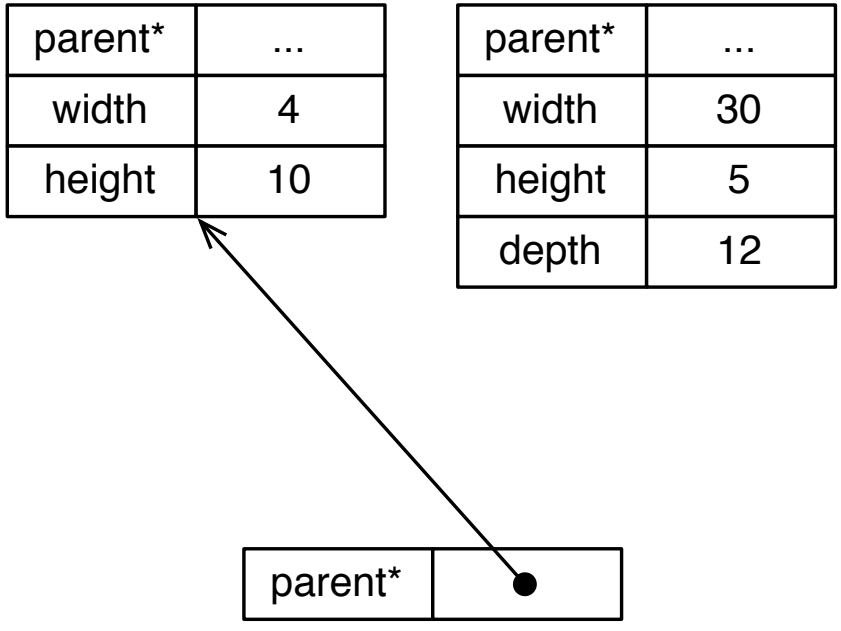
Object Creation: Flexibility

- Objects built from same prototype can differ in their behaviour
- Objects can dynamically change state variables and behaviour



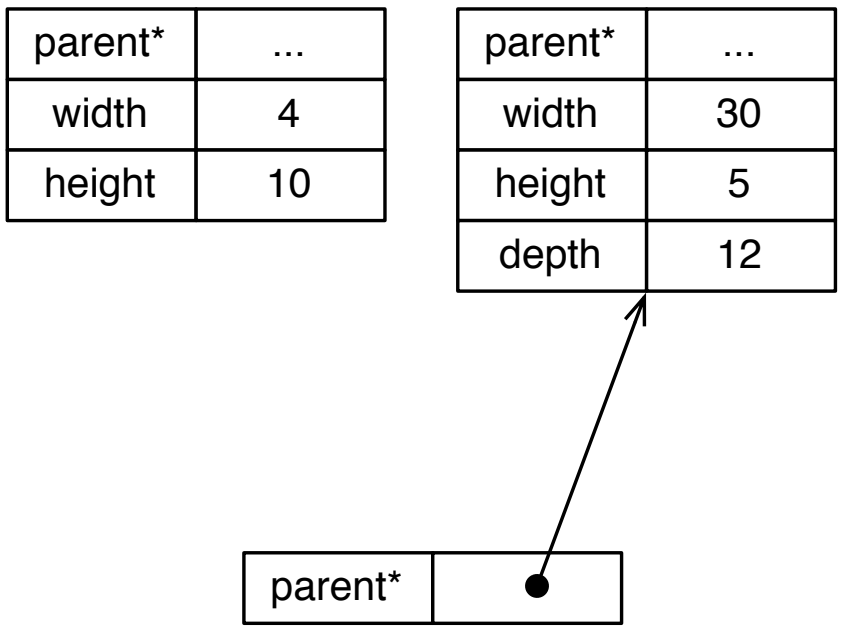
Delegation: Flexibility (2)

- An object's parent(s) can change dynamically



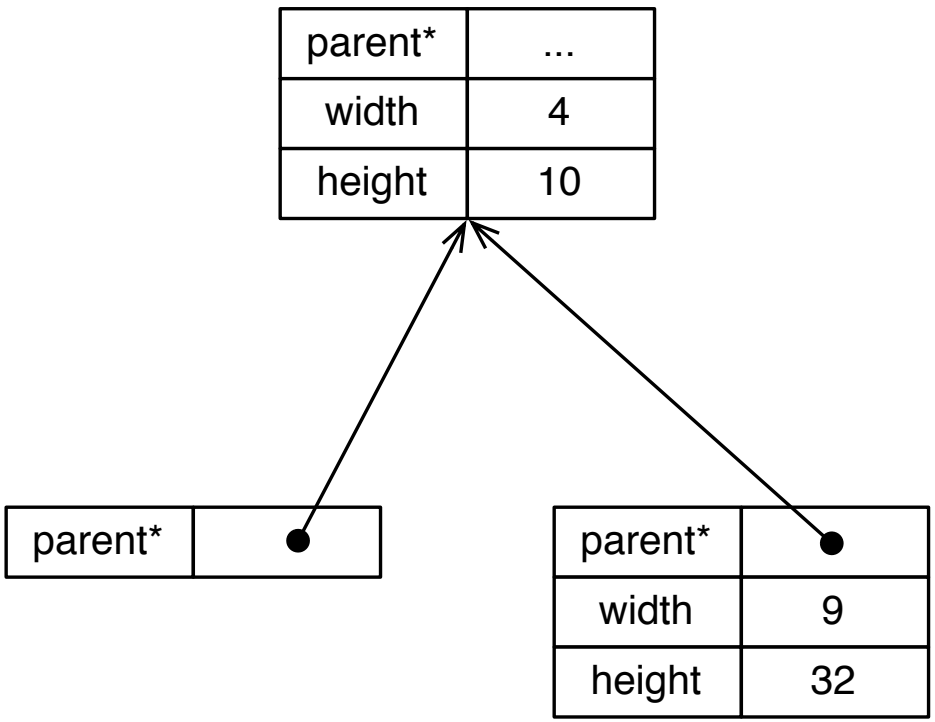
Delegation: Flexibility (2)

- An object's parent(s) can change dynamically



Delegation: Flexibility (3)

- Objects can have unique or shared values for state variables



Inheritance vs. Delegation

Inheritance

- Objects of the same class have the same behaviour
- Objects have unique values for state variables
- Object structure and behaviour is fixed at compile-time
- The class of an object is fixed

Delegation

- Objects built from same prototype can differ in their behaviour
- Objects can have unique or shared values for state variables
- Objects can dynamically change (add, remove) operations and state variables
- Object's parent(s) change can change dynamically

$\Rightarrow$ Delegation is more flexible than inheritance. Is this added flexibility helpful or harmful?

Class-based vs. Prototype-based Languages

Class-based

- Classes describe how objects behave
- Good for situations where many objects have same behaviour (one class – many instances)
- Extra effort to create singleton objects

Prototype-based

- Objects embody description of their own behaviour
- Good for situations where many objects have unique behaviour
- “Singleton” objects are easy

This Week's Reading Assignment

- David Ungar, Craig Chambers, Bay-Wei Chang, and Urs Hölzle, *Organizing Programs Without Classes*, Lisp and Symbolic Computation 4(3), Kluwer Academic Publishers, June, 1991.

- Download link:
  `http://bibliography.selflanguage.org/organizing-programs.html`