Introduction to Object Oriented Programming 2E

Timothy A. Budd

Chapter 7

Inheritance

© Timothy A. Budd 1996 (Modifications © H. C. Cunningham 1997)

Motivation for Inheritance

Use inheritance to create new software structures from existing software units to:

- improve productivity
- enhance quality

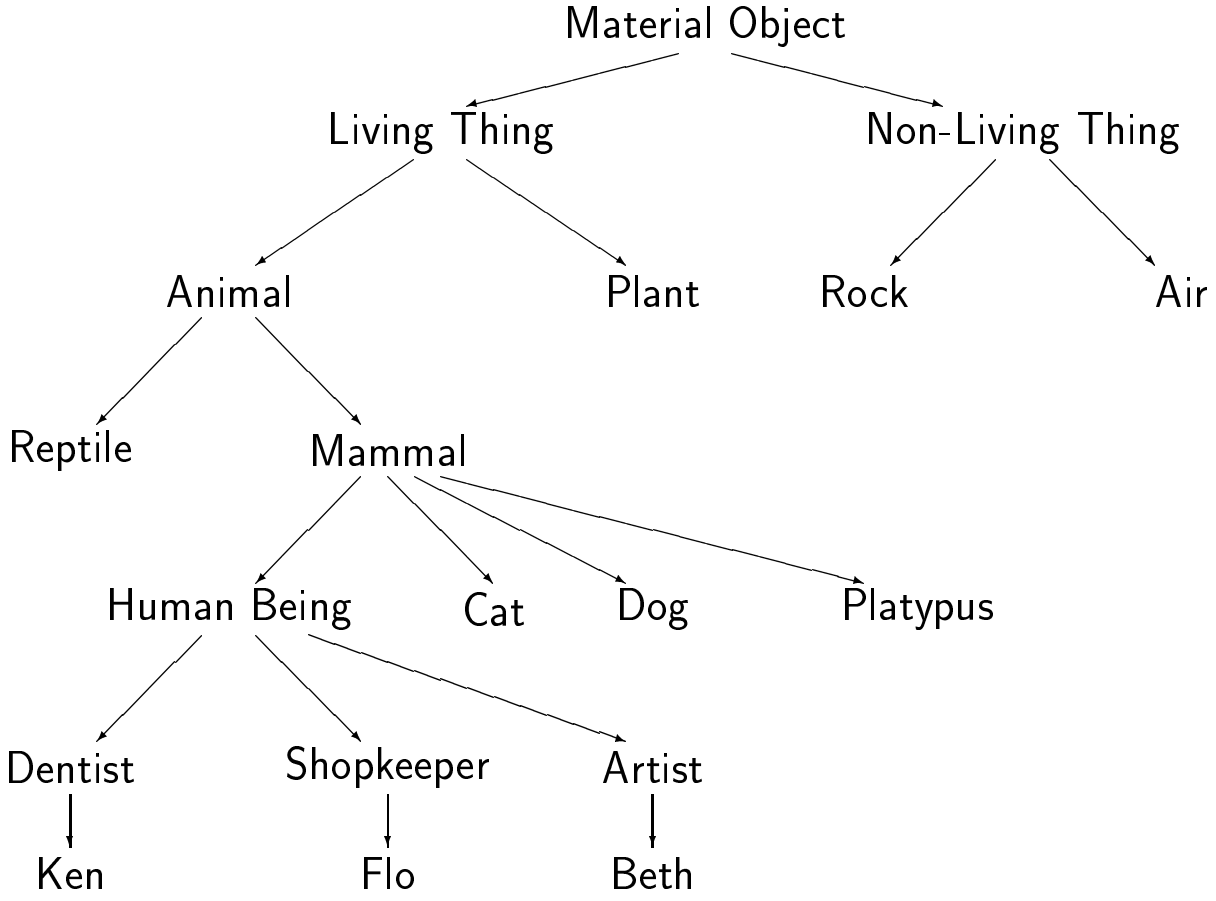
© Timothy A. Budd 1996 (Modifications © H. C. Cunningham 1997)

Tension between Generality and Specialization

- Specific projects usually require very specialized software
- Reusability usually requires very general software
- Inheritance allows general software to be specialized for a project

© Timothy A. Budd 1996 (Modifications © H. C. Cunningham 1997)

Abstract Idea of Inheritance



© Timothy A. Budd 1996 (Modifications © H. C. Cunningham 1997)

Practical Meaning of Inheritance

- Data and behavior defined in parent class are accessible within child
- Data and behavior of child class are extensions to data and behavior of parent

With respect to the parent class, a child class is, in some sense:

- an extension – larger set of properties
- a restriction – more specialized objects

© Timothy A. Budd 1996 (Modifications © H. C. Cunningham 1997)

Idealized Image of Inheritance

Consider the following argument:

- Subclass instances must possess all data areas of the parent
- Subclass instances must implement all functionality of the parent
- Thus, a subclass instance should be *indistinguishable* from a parent class instance – child can be substituted for parent

Principle of Substitutability: If B is a subclass of A, instances of B can be substituted for instances of A in *any situation* with *no observable effect*.

© Timothy A. Budd 1996 (Modifications © H. C. Cunningham 1997)

Subclass, Subtype, and Substitutability

subtype: class that satisfies principle of substitutability

subclass: something constructed using inheritance, whether or not it satisfies the principle of substitutability.

The two concepts are independent:

- Not all subclasses are subtypes
- Sometimes subtypes are constructed without being subclasses

© Timothy A. Budd 1996 (Modifications © H. C. Cunningham 1997)

Forms of Inheritance

Specialization: Child class is a special case (subtype) of parent

Most common form of inheritance

Example: Professor is a specialized form of Employee

Preserves substitutability

Specification: Parent class defines behavior implemented in the child, but not parent

Second next most common form of inheritance

Example: abstract class Stack defines methods for class StackInArray

Preserves substitutability

Subclasses are realizations of incomplete abstract specification

Defines common interface for group of related classes

© Timothy A. Budd 1996 (Modifications © H. C. Cunningham 1997)

Forms of Inheritance (Continued)

Construction: Parent class used only for its behavior – child class is not subtype – no *is-a* relationship to parent

Example: extending List class to develop Set, without "hiding" unneeded methods

Example: extending a byte-based I/O stream to a stream for handling other objects

Often violates substitutability

More common in dynamically typed languages than in statically typed

Can sometimes use aggregation instead

© Timothy A. Budd 1996 (Modifications © H. C. Cunningham 1997)

Forms of Inheritance (Continued)

Generalization: Child class modifies or overrides some methods of parent, extends the behavior to more general kind of object

Example: graphics Window generalized to ColorWindow (with background color)

Opposite of specialization – violates substitutability

Useful when must build from fixed, difficult-to-modify set of classes

Where possible, better to invert class hierarchy or use aggregation

Extension: Child class adds new functionality to parent, but does not change any inherited behavior

Example: StringSet extends Set, adding string-related methods (e.g, prefix search)

Preserves substitutability

© Timothy A. Budd 1996 (Modifications © H. C. Cunningham 1997)

Forms of Inheritance (Continued)

Limitation: Child class limits some of the behavior of parent

Example: Stack extends DoubleEndedQueue, replacing unneeded methods to give error messages

Violates substitutability

Useful when must build from fixed, difficult-to-modify set of classes

Avoid when possible, perhaps use aggregation

Variance: Child and parent class are variants of each other – inheritance to allow code sharing – arbitrary relationship

Example: graphics Tablet class extending Mouse to share similar control code

Violates substitutability

Better to define more general parent class like PointingDevice

© Timothy A. Budd 1996 (Modifications © H. C. Cunningham 1997)

Forms of Inheritance (Continued)

Combination: Child class inherits features from more than one parent – multiple inheritance

Example: GraduateInstructor inherits from both GraduateStudent and Faculty

Often difficult to understand and to implement language

Often use to "mix-in" specification of another role or protocol

© Timothy A. Budd 1996 (Modifications © H. C. Cunningham 1997)

Inheritance and Assertions

Suppose C is a *subtype* of P:

- P and C have interface invariants I and IC, respectively
- meth() is a public method of P with precondition Q and postcondition R
- meth() in C has precondition QC and postcondition RC

C should not violate I, Q, and R:

- IC *implies* I
- Q *implies* QC
- RC *implies* R

© Timothy A. Budd 1996 (Modifications © H. C. Cunningham 1997)

Trees versus Forests

There are two common views of class hierarchies.

Tree: all classes part of single class hierarchy

Advantage: root's functionality inherited by all objects – all have basic functionality

Disadvantage: tight coupling of classes, large libraries for an application

Languages: Smalltalk, Objective C, Delphi Object Pascal, Java

Forest: classes only placed in hierarchies if they have a relationship – many small hierarchies.

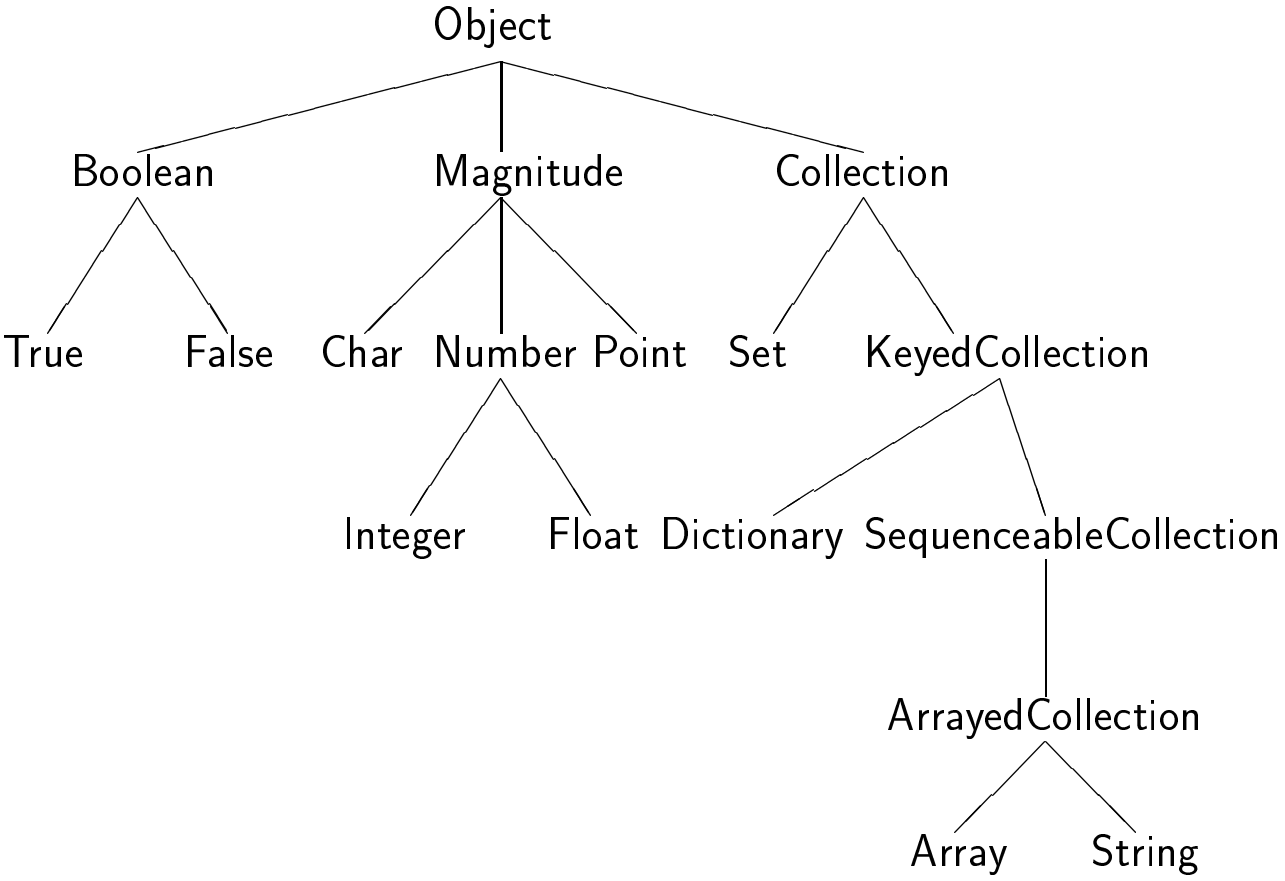
Advantage: smaller libraries of classes for application, less coupling possible

Disadvantage: no shared functionality among all objects

Language: C++, Apple Object Pascal

© Timothy A. Budd 1996 (Modifications © H. C. Cunningham 1997)

A Portion of the Little Smalltalk Hierarchy



© Timothy A. Budd 1996 (Modifications © H. C. Cunningham 1997)

Inheritance in Java

- Tree hierarchy with root class Object
- Single inheritance – using keyword extends after class name
- public features accessible from outside class
- private features accessible from inside class only
- protected features accessible inside class or inside any child class
- abstract classes cannot be instantiated
- final classes cannot be extended
- Limited multiple inheritance for specification (of a protocol or role)
  - interface construct to define
  - implements after class name to promise implementation of interface

© Timothy A. Budd 1996 (Modifications © H. C. Cunningham 1997)

Inheritance in Java

```
public abstract class Stack {  // extends Object by default
    // data definitions plus signatures and
    //     possibly implementations of methods
    }

public class StackInArray extends Stack {
    // extended features plus overridden implementations
    }

public interface Queue {
    // signatures of public methods Queues must provide
    }

public class QueueAsLinkedList implements Queue {
    // includes implementations of the Queue methods
    }
```

© Timothy A. Budd 1996 (Modifications © H. C. Cunningham 1997)

Benefits of Inheritance

- Software reuse
- Code sharing
- Improved reliability
- Consistency of interface
- Rapid prototyping
- Polymorphism
- Information hiding

© Timothy A. Budd 1996 (Modifications © H. C. Cunningham 1997)

Costs of Inheritance

- Execution speed
- Program size
- Message passing overhead
- Program complexity

© Timothy A. Budd 1996 (Modifications © H. C. Cunningham 1997)