Chapter 2: Part-C
The Object Model

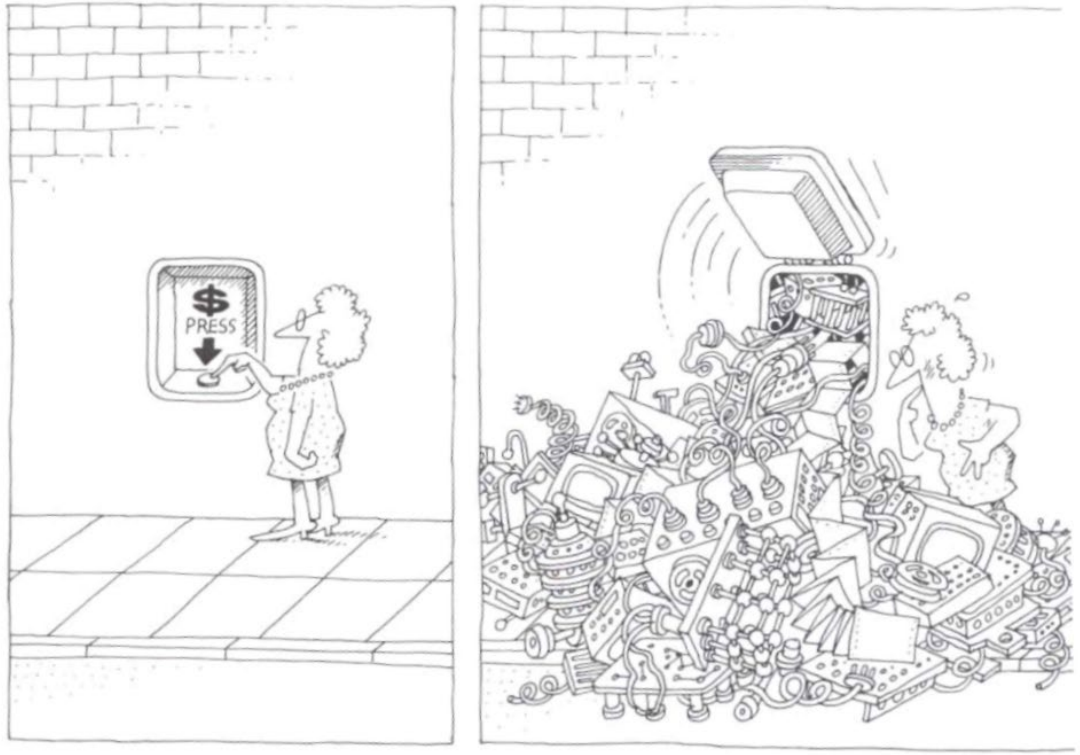
H.S. Sarjoughian

CSE 460: Software Analysis and Design

School of Computing, Informatics and Decision Systems Engineering
Fulton Schools of Engineering

Arizona State University, Tempe, AZ, USA

Creating Illusion of Simplicity



The task of the software development team is to engineer the illusion of simplicity.

ATM Machine

Source: *OOAD*

software engineers must deal with arbitrary complexity for building industrial-strength software

What Is The Object Model?

The Object model is the collection of principles that form the foundation of object-oriented analysis and design.

Object model provides a paradigm for software engineering emphasizing the principles of:

- Abstraction
- Encapsulation
- Modularity
- Hierarchy

basic principles

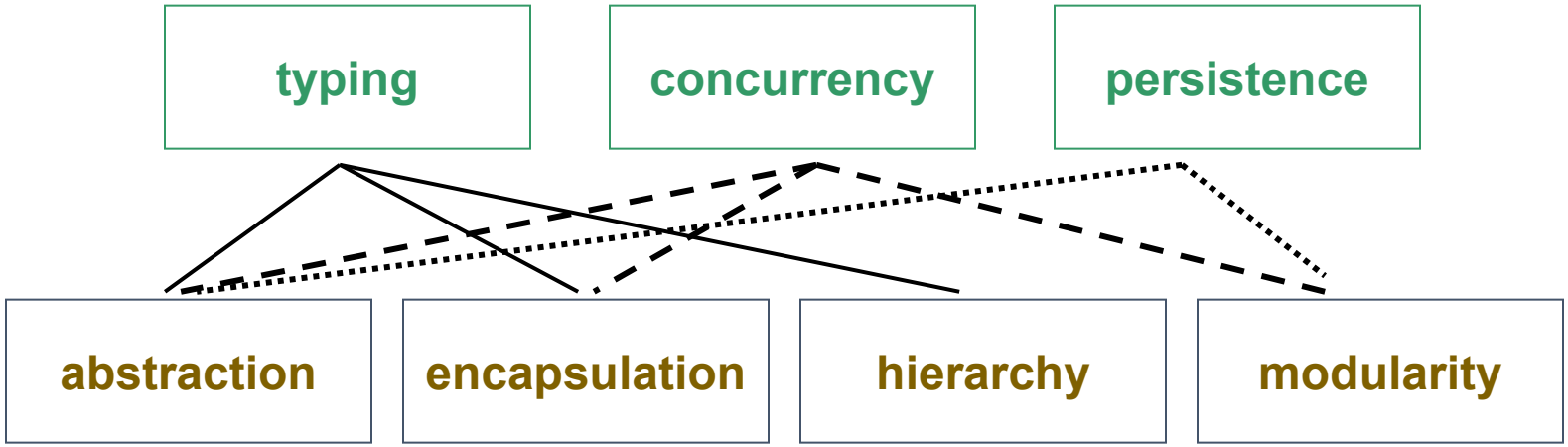
- Typing
- Concurrency
- Persistence

advanced principles

The Object Model

The Object Model is based on seven principles which collectively provide the foundation for object-oriented software development

higher-level elements



lower-level elements

——— directly related to the Object Model

– – – – somewhat directly related to the Object Model

..... indirectly related to the Object Model

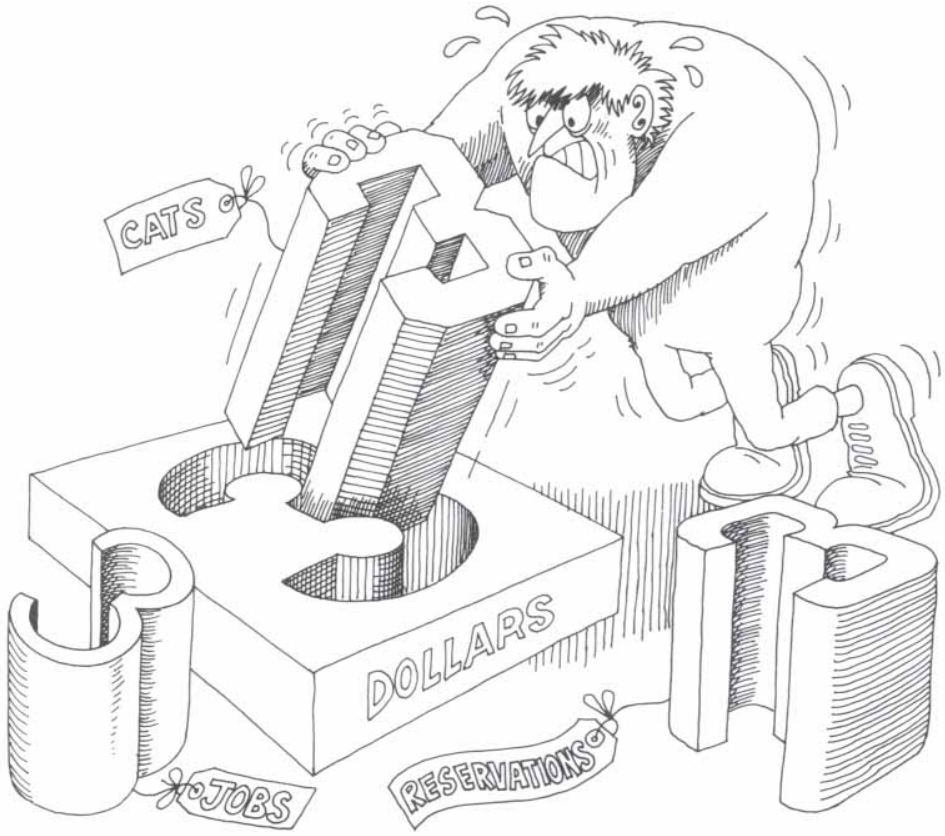
Typing

- A type is a precise characterization of structural or behavioral properties shared by a collection of entities
- Type and class are generally used interchangeably
- Typing: untyped, weakly typed, strongly typed

> typing is the enforcement of the class of an object, such that objects of *different types may not be interchanged*, or at the most, they may be *interchanged only in very restricted ways*

Typing Caricature

G. Booch, OOAD



Strong typing prevents mixing abstractions.

strong typing prevents mixing abstractions

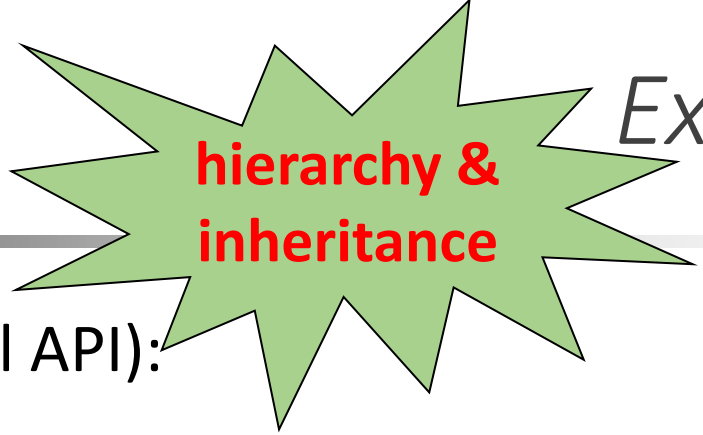
Programming Languages and Typing

- A programming language may be
  - untyped:
    - Type conformance is not enforced
    - Untyped languages provide highest level of flexibility (e.g., any object can be passed as an argument to another object)
    - To ensure correct, intended behavior, the burden is placed on the software designer and implementator
    - Prog. Lang.: Scheme and Smalltalk
  - strongly typed

Strong Typing

- Strongly typed programming languages
  - Type conformance is strictly enforced – i.e.,
    - every variable and expression has a known type at compile time (e.g., Java has primitive and reference types)
    - data values can be stored, passed as arguments, returned by methods, and operated on – exact signature matching of a class or its superclass is necessary
  - Strongly typed languages are important for complex systems (or programming in large), but requires care for handling dependencies
  - Prog. Lang.: Eiffel, Java, Swift, etc.

Examples



LinkedList class (Java Util API):

- LinkedList is a flexible concrete class suitable for flexible insert, access, and delete operations
  - Things can be inserted – addFirst(Object elem), add(int index, Object elem), …
  - Things can be retrieved – getLast( ), …
- However, using an element from a list instance requires knowing about its type!

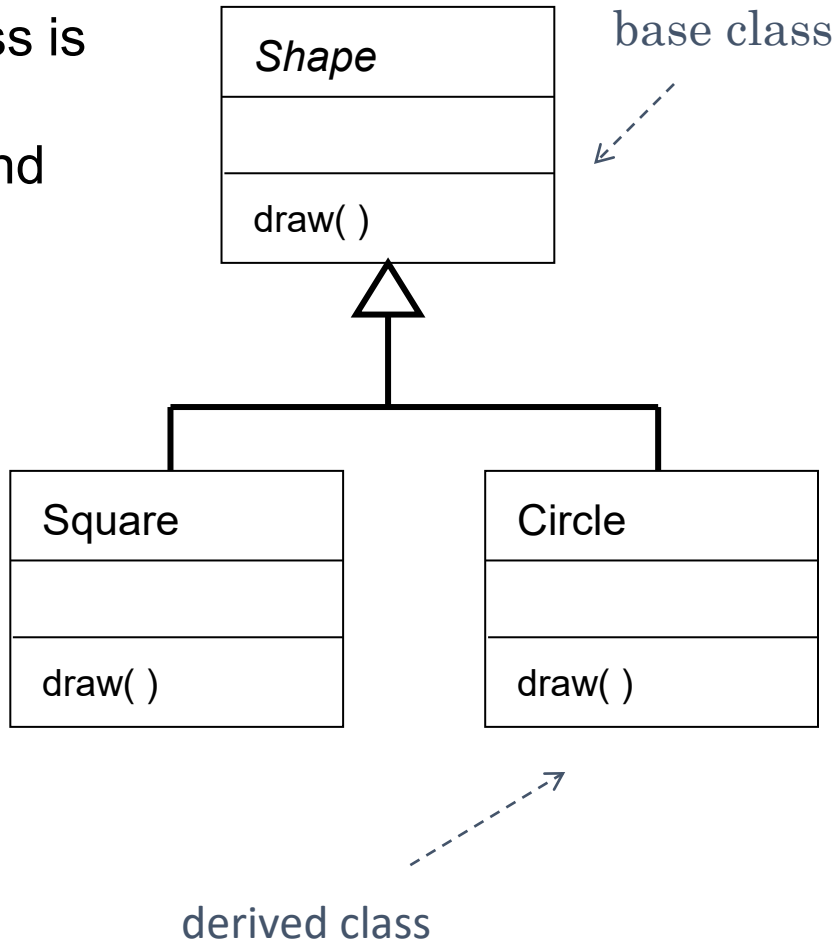
Iterator class (Java Util API)

- Iterator provides a uniform way to iterate through different concrete classes
  - ListIterator is an iterator interface which operates on objects of a LinkedList without knowing about their typing (hasNext ( ), hasPrvious( ), remove( ), …)

Examples (cont.)

manipulating handle of the base class is key for adding new behavior with minimal change to existing design and implementation

```
…
Vector v = new Vector( );
v.addElement(new Circle);
…
Enumeration e = v.elements( );
while(e.hasMoreElements( ))
   ((shape)e.nextElement( )).draw( );
…
```

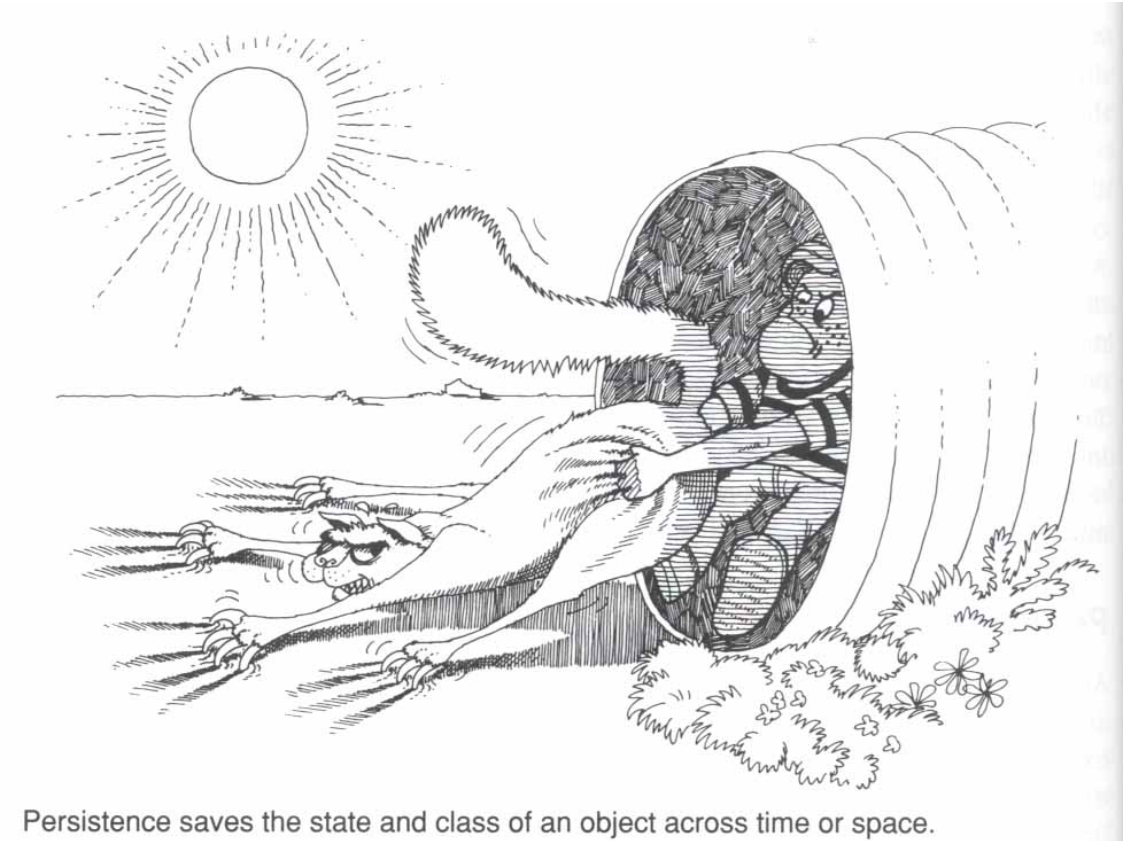


Static and Dynamic Bindings

Static and dynamic binding concepts are distinct from typing

- static (early) binding – compile time error detection
  - bindings of variables and expressions are fixed at the time of compilation
- dynamic (late) binding – run-time error detection
  - bindings of variables and expressions are decided at run time
  - polymorphism is based on inheritance and dynamic binding working together

Persistence Caricature

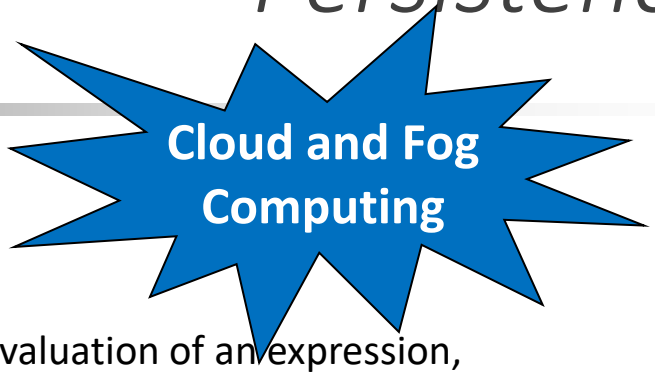


Persistence saves the state and class of an object across time or space.

OOAD, 2007

persistence saves the **state** and **class** of an object across time or space

Persistence



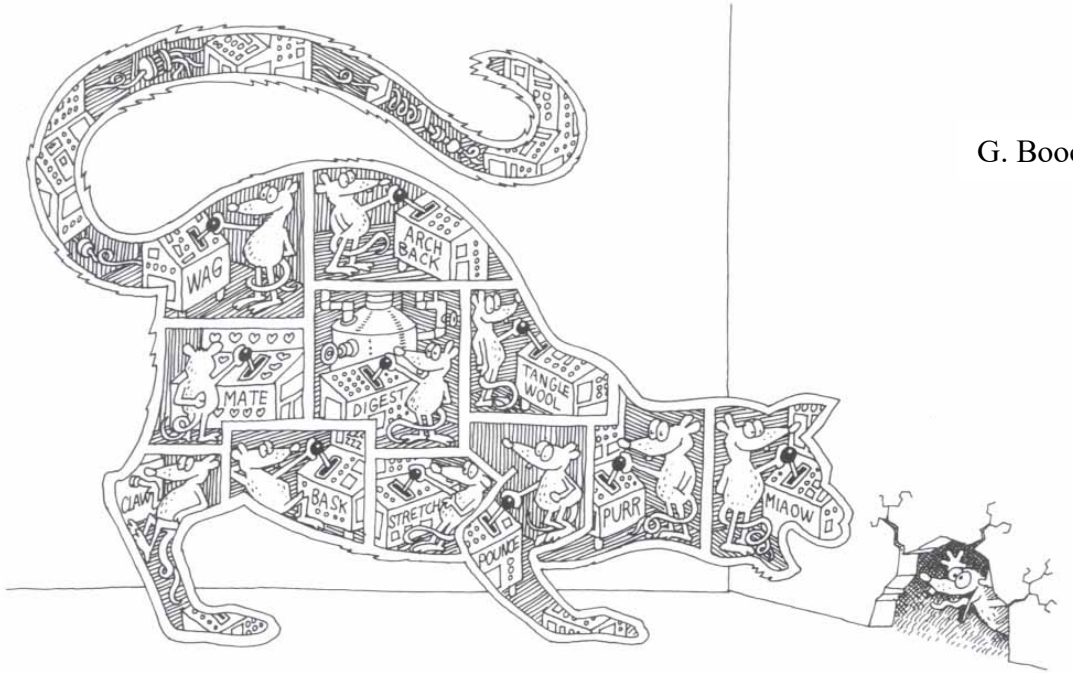
- Each **object** has time and space dimensions
- An object can
  - have transitory existence; e.g., temporary evaluation of an expression,
  - have dynamic (run-time) existence; e.g., instantiated object in a running program), or
  - have permanent (external) existence; e.g., a file residing on a physical media such as hard disk, …)
- Permanent objects can live in between multiple program invocations in time and space
- Software tools (e.g., GitHub) support permanent existence of data objects
- Databases (e.g., RDBMS and ORDBMS) support permanent storage

persistency is the property of an object through which its existence transcends time (i.e., the object continues to exists after its creator ceases to exist) and/or space (i.e., the objects location moves from the address space in which it was created to another).

Persistence in Java Prog. Lang.

- Java programming language provides persistence through “object serialization”
  - lightweight persistence – objects must be explicitly serialized and de-serialized (no keyword such as “persistence” exist)
  - objects are serialized into a sequence of bytes that may be restored later fully in their original form – a class implements the Serializable interface
  - serialized objects can be stored locally or transmitted across network while automatically accounting for different operating systems
    - Class Data implements serializable { …}
    - I/O provides automatic object serialization (e.g., OutputStream, ObjectOutputStream objects)
  - customized serialization – allow some things to be sent and others not

Concurrency Caricature



G. Booch, OOAD

Concurrency allows different objects to act at the same time.

concurrency allows different objects to **act** at the same time

- concurrency is the property that distinguishes a “live object” from one that is not alive

Concurrency

Concurrency refers to multiple processes executing simultaneously – can be either absolute (multiple CPUs) or partial (single CPU)

- a process is a self-contained running program with its own address space
- a multitasking OS supports running multiple processes at a time
- a thread (an independently executing subtask) is a single sequential flow of control within a process
- concurrency, using some synchronization algorithms (time- and resource-slicing), can support
  - multiple threads of control for one dedicated CPU and address space
  - multiple processes where each process has its own dedicated CPU and address space
- concurrency types
  - lightweight: shared data – inexpensive message passing
  - heavyweight: dedicated address space for each process – expensive message passing
- concurrency can improve program design and resource sharing both of which are important for building large distributed software systems

Concurrency in Java Prog. Lang.

- Java is a multi-threading programming language
- Java supports lightweight concurrency by allowing an object to run as a thread
- Java hides details and complexities associated with concurrent (lightweight multi-threading) programming
  - an object can have its own thread of control – e.g., methods such as run( ) and start( ) provide basic means to control and manage an object's execution in the presence of other objects

```
class Mythread extends Thread {
        …
        void run( ) { …}
        void init( ) {…}

        …

        synchronized void m1( )  {…} //dedicate data for m1 use only
        …
}
```

Summary

- The Object Model is fundamentally different from *classical structured* analysis and design models
- The Object Model incorporates structured analysis and design proven principles, methods, and techniques
- The Object Model provides the foundation for large- and very large-scale (complex) software systems
- Programming (object-oriented or not) is distinct from object-oriented analysis and design
- Challenge of object-oriented analysis and design is in discovering and inventing relationships that effectively bring together first and foremost the principles of abstraction, encapsulation, modularity, and hierarchy
- Typing, concurrency, and persistence play key roles in design and implementation of large-scale distributed software systems

References

- *Object-Oriented Analysis and Design with Applications, 2nd Edition, G. Booch, Benjamin Cummings, 1994*
- *Thinking in Java, B. Eckel*
- *Eclipse Modeling Framework, http://eclipse.org/*