

**Software Engineering Basics** 1. S/E Basics 1

- Progression of Roles
- Design Strategies in OO Programming
- Abstraction
- Practical Abstraction
- Abstraction
- Better Abstractions
- Mapping Abstraction to Software
- Separation of Interface from Implementation
- Interchangeability of Implementations
- Specificity of Interface
- Mapping Abstraction to Software in OO
- General Structure of a Class
- General Structure of an Object
- Multiple Instances of a Class
- Software Engineering Goals

Computer Science Dept Va Tech January 2000    OO Software Design and Construction    ©2000 McQuain WD

**Progression of Roles** 1. S/E Basics 2

Computer Science Dept Va Tech January 2000    OO Software Design and Construction    ©2000 McQuain WD

**Design Strategies in OO Programming** 1. S/E Basics 3

Abstraction	modeling essential properties
Separation	treat what and how independently
Composition	building complex structures from simpler ones
Generalization	identifying common elements

Computer Science Dept Va Tech January 2000    OO Software Design and Construction    ©2000 McQuain WD

**Abstraction** 1. S/E Basics 4

Modeling entities in software

Only essential aspects should be captured

- attributes
- behavior

**Wassily Kandinski**  
Cossacks, 1910-11

Computer Science Dept Va Tech January 2000    OO Software Design and Construction    ©2000 McQuain WD

Practical Abstraction 1. S/E Basics 5



Critique??

Computer Science Dept Va Tech January 2000 OO Software Design and Construction ©2000 McQuain WD

Abstraction 1. S/E Basics 6

A named collection of attributes and behavior relevant to modeling a given entity for some particular purpose.

Desirable Properties:

well named	name conveys aspects of the abstraction
coherent	makes sense
accurate	contains only attributes modeled entity contains
minimal	contains only attributes needed for the purpose
complete	contains all attributes and behavior needed for the purpose

Computer Science Dept Va Tech January 2000 OO Software Design and Construction ©2000 McQuain WD

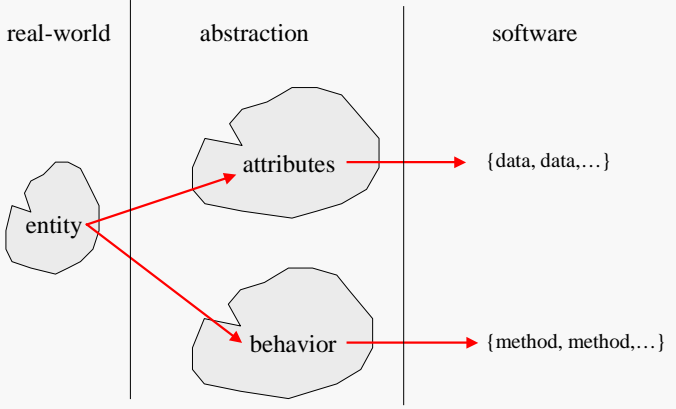
Better Abstractions 1. S/E Basics 7



**MEN** **WOMEN**

Computer Science Dept Va Tech January 2000 OO Software Design and Construction ©2000 McQuain WD

Mapping Abstraction to Software 1. S/E Basics 8



real-world	abstraction	software
entity	attributes	→ {data, data,...}
	behavior	→ {method, method,...}

Computer Science Dept Va Tech January 2000 OO Software Design and Construction ©2000 McQuain WD

### Separation of Interface from Implementation

1. S/E Basics 9

In programming, the independent specification of an interface and one or more implementations of that interface.

What is to be done  
vs  
How it is to be done

Computer Science Dept Va Tech January 2000 OO Software Design and Construction ©2000 McQuain WD

### Interchangeability of Implementations

1. S/E Basics 10

Allows the creation of multiple implementations with a common interface.

For example: a List ADT could use a dynamic linked list or a dynamic array for the underlying physical data structure. In either case, the same interface would be appropriate (and the user need not be concerned with the underlying structure in many cases).

Implementations that share a common interface are said to be “plug compatible”.

They may differ in algorithmic complexity, reliability, platform dependencies, etc.

Computer Science Dept Va Tech January 2000 OO Software Design and Construction ©2000 McQuain WD

### Specificity of Interface

1. S/E Basics 11

Also allows a single implementation to support multiple interfaces.

This allows the isolation of restricted set used in one situation versus another

For example, we could have a very general List ADT that supported both standard List operations, and also Stack operations. By “subsetting” the functionality of the ADT into separate interfaces, we could provide both categories of operation, in a natural way, without duplication of shared code.

In essence, we view the implementation as a library of related widgets.

Computer Science Dept Va Tech January 2000 OO Software Design and Construction ©2000 McQuain WD

### Mapping Abstraction to Software in OO

1. S/E Basics 12

real-world      abstraction      OO software

entity → attributes → {data, data,...}

entity → behavior → {method, method,...}

Computer Science Dept Va Tech January 2000 OO Software Design and Construction ©2000 McQuain WD

### General Structure of a Class

1. S/E Basics 13

class: a named software representation for an abstraction that separates the implementation of the representation from the interface of the representation

A class models an abstraction, which models an entity (possibly “real”).

A class represents all members of a group of objects (“instances” of the class).

A class provides a public interface and a private implementation.

The hiding of the data and “algorithm” from the user is important. Access restrictions prevent idle or malicious alterations.

The diagram shows a large rectangle labeled 'className'. Inside it, a dashed rectangle is labeled 'private' and contains '{data, data, ...}'. Outside the 'className' rectangle, a box labeled 'typical organization' contains '{method, method, ...}'. Arrows point from the 'typical organization' box to the 'private' area and from the 'private' area to the 'public' area (the space between the 'private' area and the 'className' boundary).

Computer Science Dept Va Tech January 2000    OO Software Design and Construction    ©2000 McQuain WD

### General Structure of an Object

1. S/E Basics 14

object: a distinct instance of a given class that encapsulates its implementation details and is structurally identical to all other instances of that class

An object “encapsulates” its data and the operations that may be performed on that data.

An object’s private data may ONLY be accessed via the member functions defined within the object’s class.

An object hides details of representation and implementation from the user.

**C++ note:**  
**Privacy restrictions are enforced at the class level, NOT the object level.**  
**That is, if A and B are of the same type, and A knows B’s name, then A can access the private members of B directly.**

The diagram shows a large circle representing an object. Inside, there are two boxes labeled 'code' and a box labeled 'data'. Arrows point from the 'code' boxes to the 'data' box. An arrow labeled 'interface' points from outside the circle to the 'code' boxes. An arrow labeled 'implementation' points from the 'code' and 'data' boxes to the circle boundary.

Computer Science Dept Va Tech January 2000    OO Software Design and Construction    ©2000 McQuain WD

### Multiple Instances of a Class

1. S/E Basics 15

The diagram shows a class 'SalesPerson' with private attributes 'Name', 'commissionRate', and 'totalSales', and public methods 'sellCar' and 'reportSales'. Below it are two instances: 'Joe Hokie' with '16%' commission and '\$250,000' total sales, and 'Jill Hokie' with '16%' commission and '\$275,000' total sales. Both instances have 'sellCar' and 'reportSales' methods.

Each instance, or object, usually has different values for the class-defined properties.

Class = Factory    Objects = Products

When developing abstractions, or classes, it may help to think of them as people-like entities with **responsibilities** and **collaborators**.

Responsibilities of knowing (respond with information to a query)

Responsibilities of doing (act on something, transform, move, sort, etc.)

Collaborators: associated objects in the system with their own responsibilities

Computer Science Dept Va Tech January 2000    OO Software Design and Construction    ©2000 McQuain WD

### Software Engineering Goals

1. S/E Basics 16

Objects and classes help programmers achieve a primary software-engineering goal: reusability

A single class is used repeatedly to create multiple object instances.

More importantly, encapsulation prevents other developers from inadvertently modifying an object’s data.

Separation allows different implementations to be used for an interface.

The diagram shows a box on the left labeled 'Software Structures' containing 'objects', 'classes', 'inheritance', 'templates', and 'design patterns'. A box on the right labeled 'Software Engineering Goals' contains 'reusability', 'extensibility', and 'flexibility'. Arrows point from 'objects', 'classes', 'inheritance', and 'templates' to 'reusability'. Arrows point from 'inheritance', 'templates', and 'design patterns' to 'extensibility'. An arrow points from 'design patterns' to 'flexibility'.

Computer Science Dept Va Tech January 2000    OO Software Design and Construction    ©2000 McQuain WD