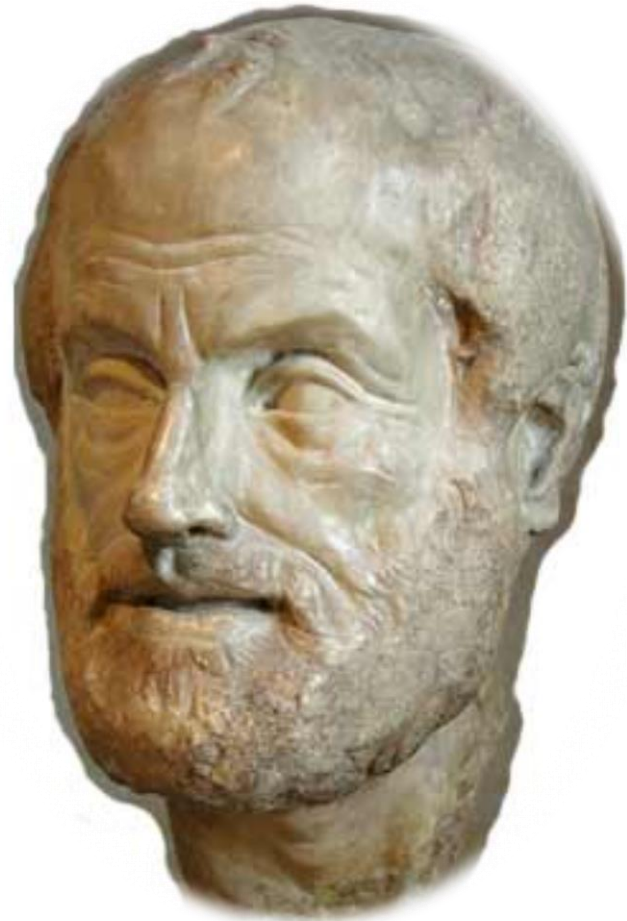


I Object

The Object Concept



Global vs. Local

- ▶ The most difficult problem in object-oriented programming is to give up the global knowledge of procedural programs, and use local knowledge of objects.



Paradigm Shift

- ▶ Procedural designs have processes, data flows, and data storage, independent of implementation language.
- ▶ A similar set of fundamental principles for object designs is:
 - Class name
 - Responsibilities
 - Collaborators

Objects and Complexity

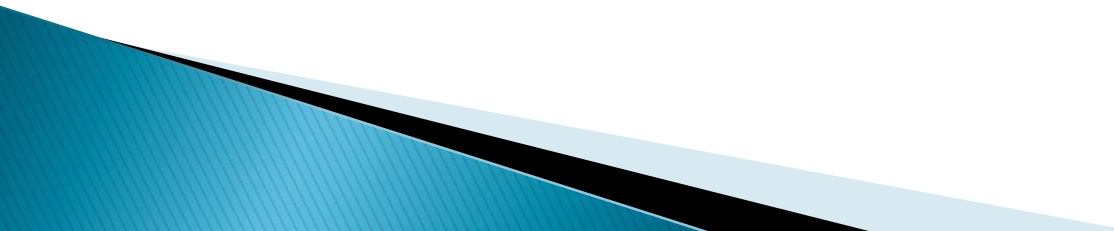
- ▶ Managing complexity is the most important technical topic in software development.
- ▶ Software's Primary Technical goal is to manage complexity.



Abstraction

- ▶ Humans deal with complexity by abstracting details away.
- ▶ To be useful, an abstraction (model) must be smaller than what it represents.

Problem Domain

- ▶ By designing in the problem domain, an extra layer of abstraction is not needed to model the problem.
 - ▶ Having to program in the computer's domain adds an extra level of abstraction.
- 

Exercise 1

- ▶ Memorize as many numbers from the following sequence as you can in 30 seconds.


1759376099873462875933451089427849701120765934

Exercise 1 Solution

- ▶ How many did you remember?
- ▶ How many could you remember with unlimited amounts of time?

Exercise 2

- ▶ Write down as many of the following telephone numbers as you can...

- Home:
 - Office:
 - Boss:
 - Co-worker:
 - Parents:
 - Spouse's office:
 - Fax:
 - Friend1:
 - Friend2:
 - Local Pizza:
- 

Exercise 2 Solution

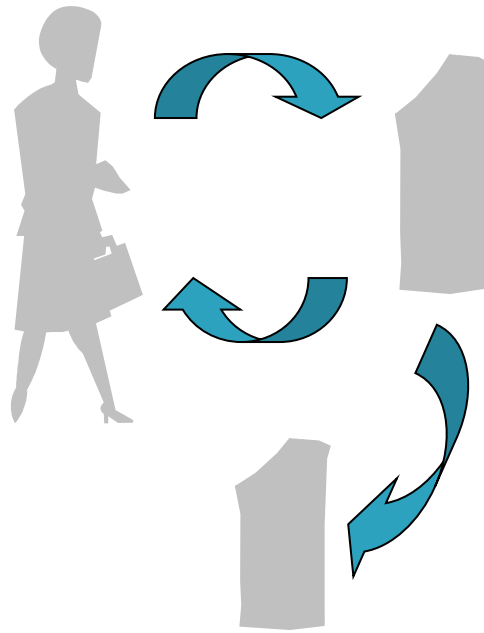
- ▶ How many did you remember?
- ▶ How many could you remember with more time?

Key Concept

- ▶ By abstracting the details of the numbers away and grouping them into a new concept (telephone number) we have increased our information handling capacity by nearly an order of magnitude.
- ▶ Working with abstractions lets us handle more information.

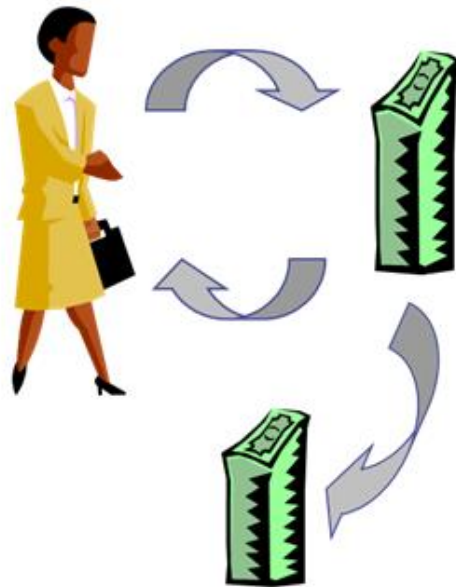


Procedure Oriented



Withdraw, deposit, transfer

Object Oriented



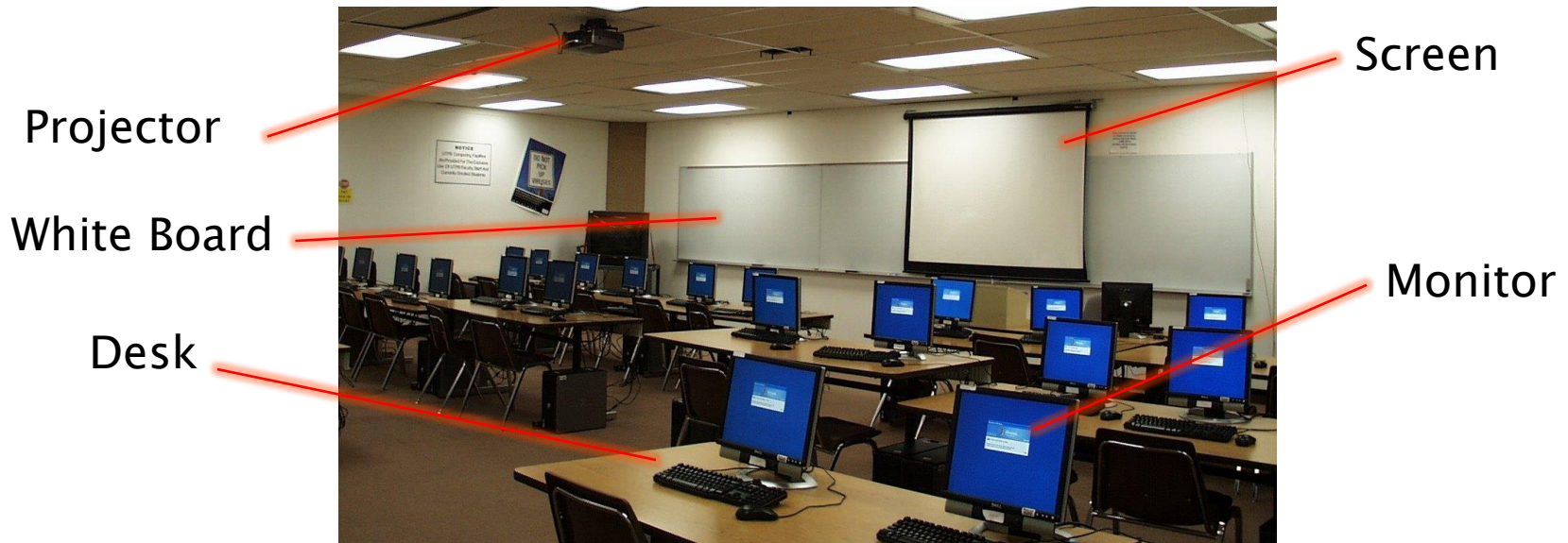
Customer, money, account

Different Viewpoints of an Object



What is an Object?

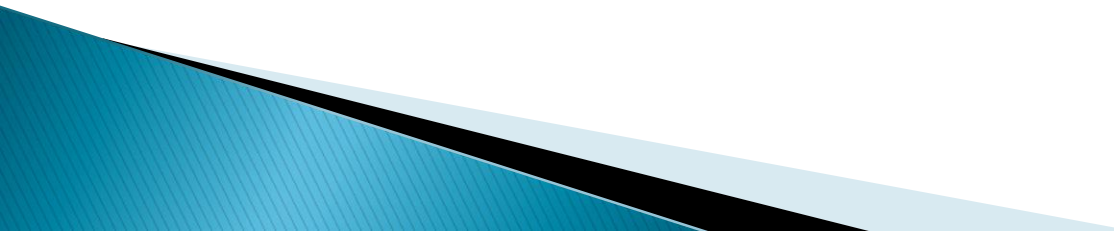
- ▶ An object represents an entity, either physical, conceptual, or software. An object is defined by it's attributes and behaviors.



Computer Science Object

- ▶ An object is a concept, abstraction, or thing with sharp boundaries and meaning for an application
- ▶ An object is something that has:
 - State
 - Behavior
 - Identity

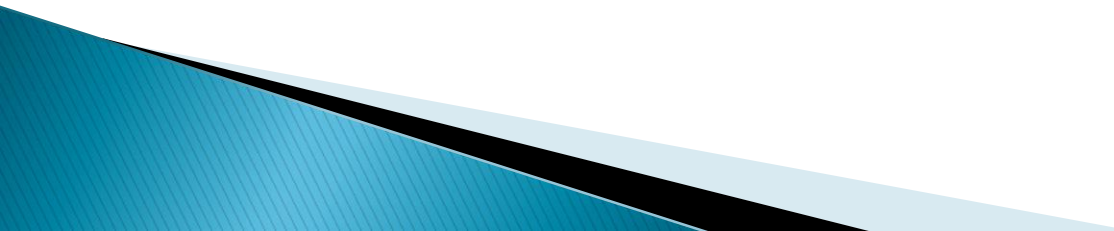
Technical Object Oriented Design

- ▶ In your previous course work with C++, Java and perhaps C#, you learned language specific programming techniques like:
 - **Encapsulation:** construction of classes
 - **Inheritance:** reuse of existing classes in a hierarchy
 - **Polymorphism:** class interface reuse
- 

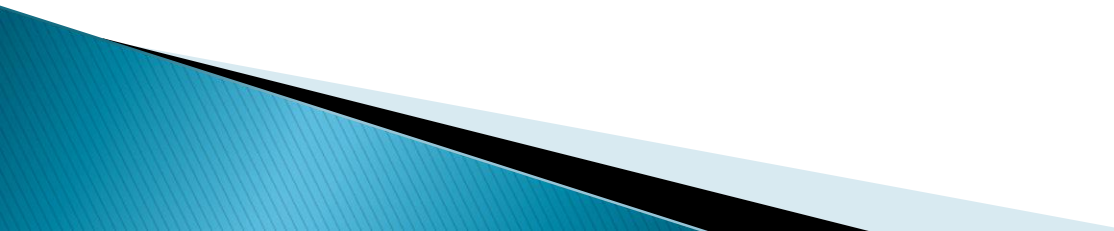
Example Object(s)



Platypus Features

- ▶ Duck Bill– To find food under water.
 - ▶ Webbed Feet– Swim in/under water.
 - ▶ Flat Tail– Helps it swim.
 - ▶ Lays Eggs– A mammal that lays eggs.
 - ▶ Mammary Glands– for feeding young.
 - ▶ Poisonous Claw – Defense.
- 

Properties of an Object

- ▶ Identity: the property of an object that distinguishes it from other objects
 - ▶ State: describes the data stored in the object
 - ▶ Behavior: describes the methods in the object's interface by which the object can be used
- 

SOLID

▶ **Single Responsibility Principle**

- A class should only have one reason to exist or change.

▶ **Open / Closed Principle**

- Extending a class shouldn't require modification of that class.

▶ **Liskov Substitution Principle**

- Derived classes must be substitutable for their base classes.

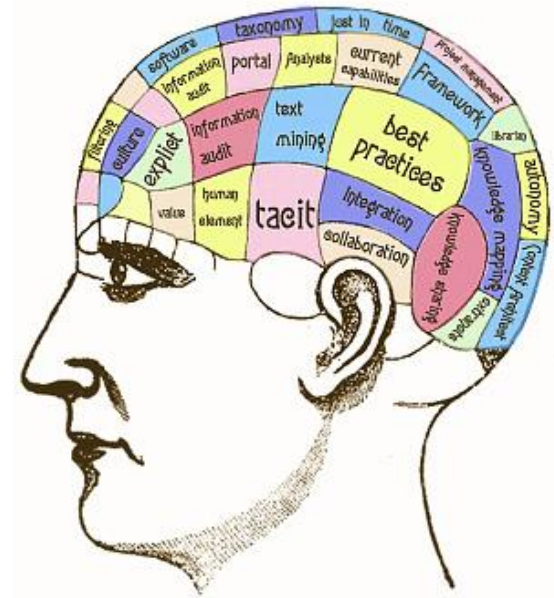
▶ **Interface Segregation Principle**

- Make interfaces client specific.

▶ **Dependency Inversion Principle**

- Program to the interface, not the implementation.

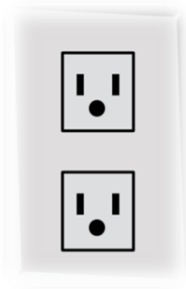
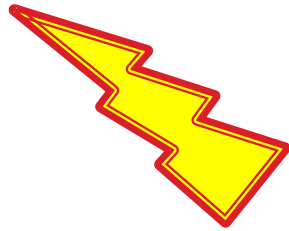
Need to Know



When modeling an object:

- ▶ What does the user need to know?
- ▶ What does the user NOT need to know?

Interface vs Implementation



Example: Car Interface



Change the Interface



Change the Implementation



Minimize Interface

- ▶ Always provide the user with as little knowledge of the inner workings of an object as possible.





Preparata al momento

Let's
Pizza

Cotta all'istante

Pronta in 2.5 minuti

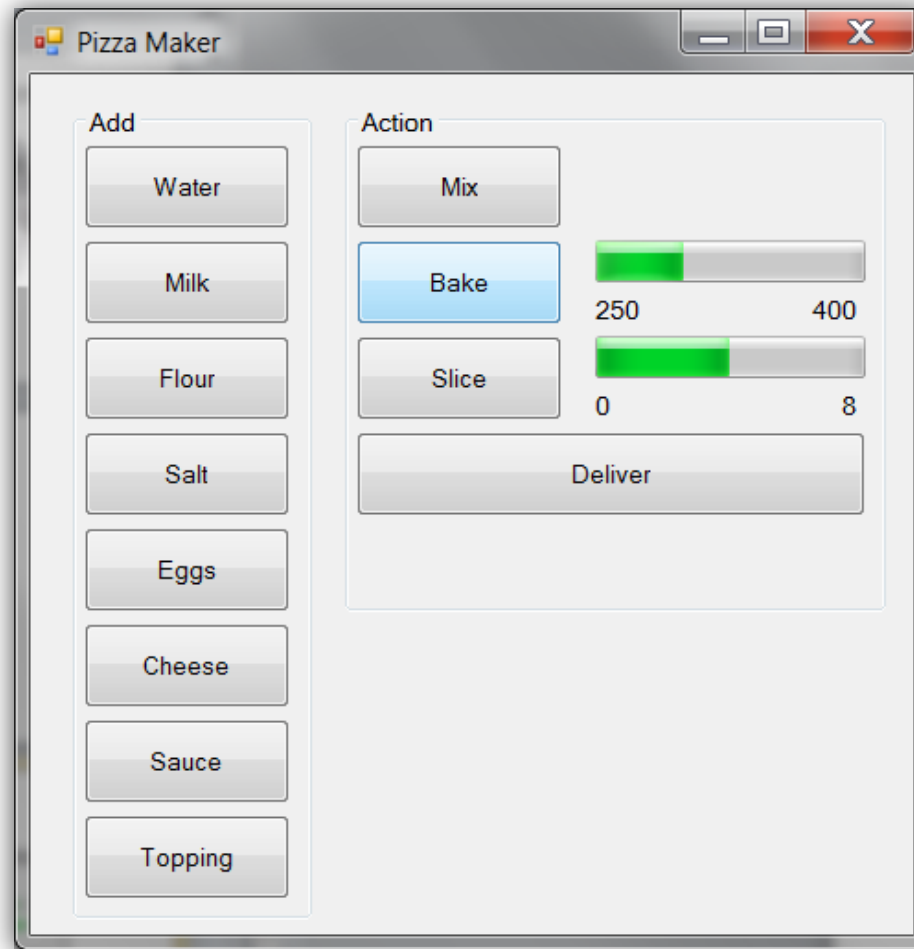
Let's Pizza Maker

Add	Action
Water	Min
Milk	Bake
Flour	Min
Salt	Done
Eggs	
Cheese	
Sauce	
Topping	

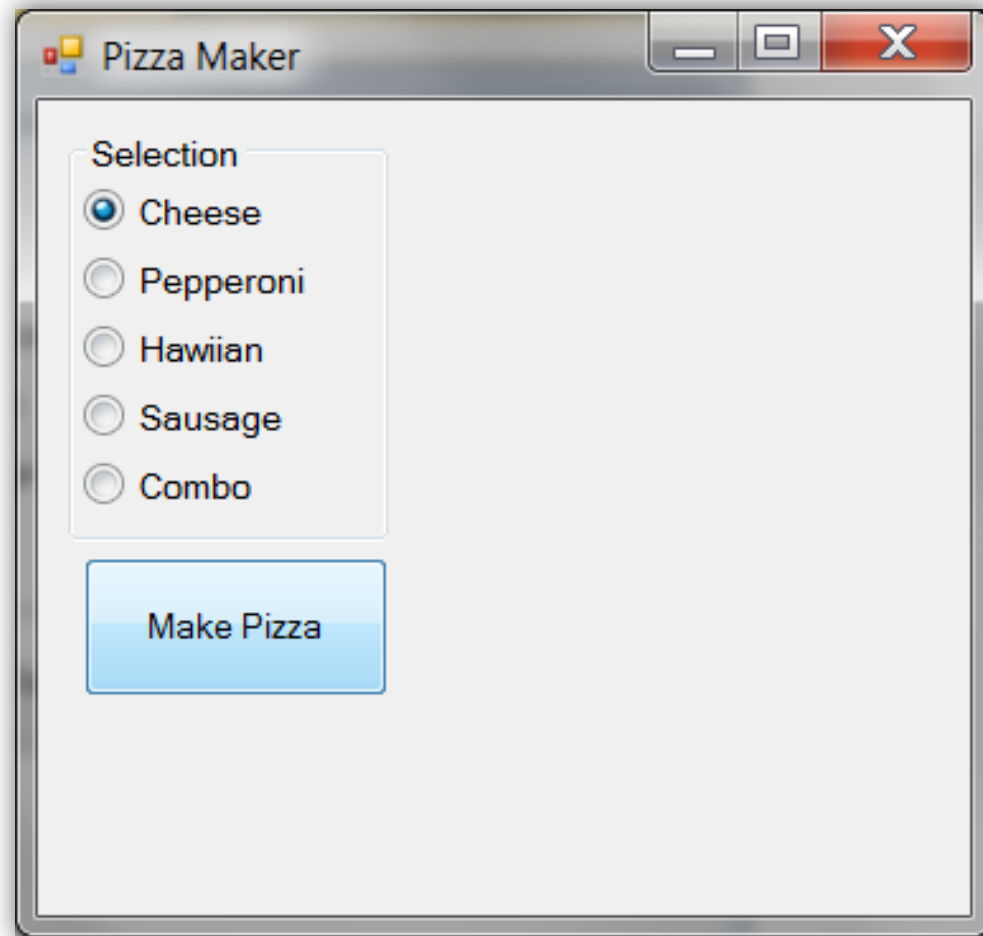
solo con ingredienti
freschi

www.letspizza.it

Detailed Interface

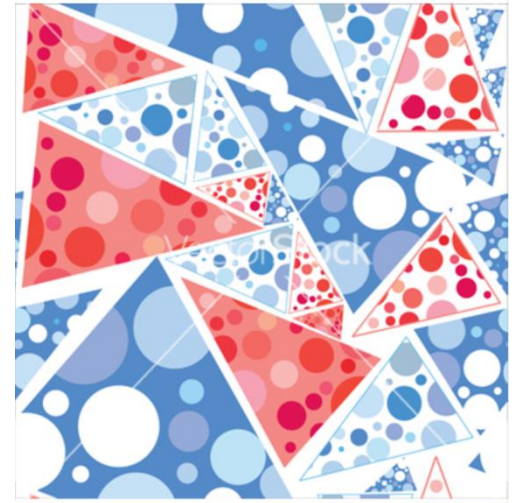


Simplified Interface

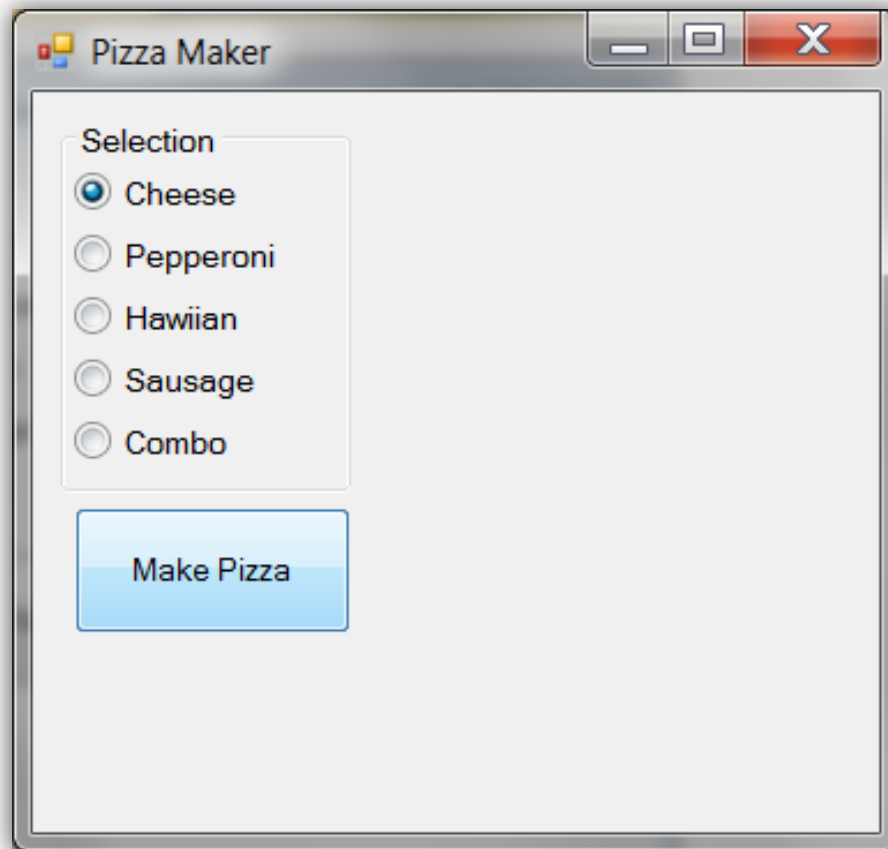


Abstract the Interface

- ▶ An advantage of OOD is classes can be designed for reuse.
- ▶ Reusable classes have interfaces more abstract than concrete.



Specific Interface



Reusable Interface

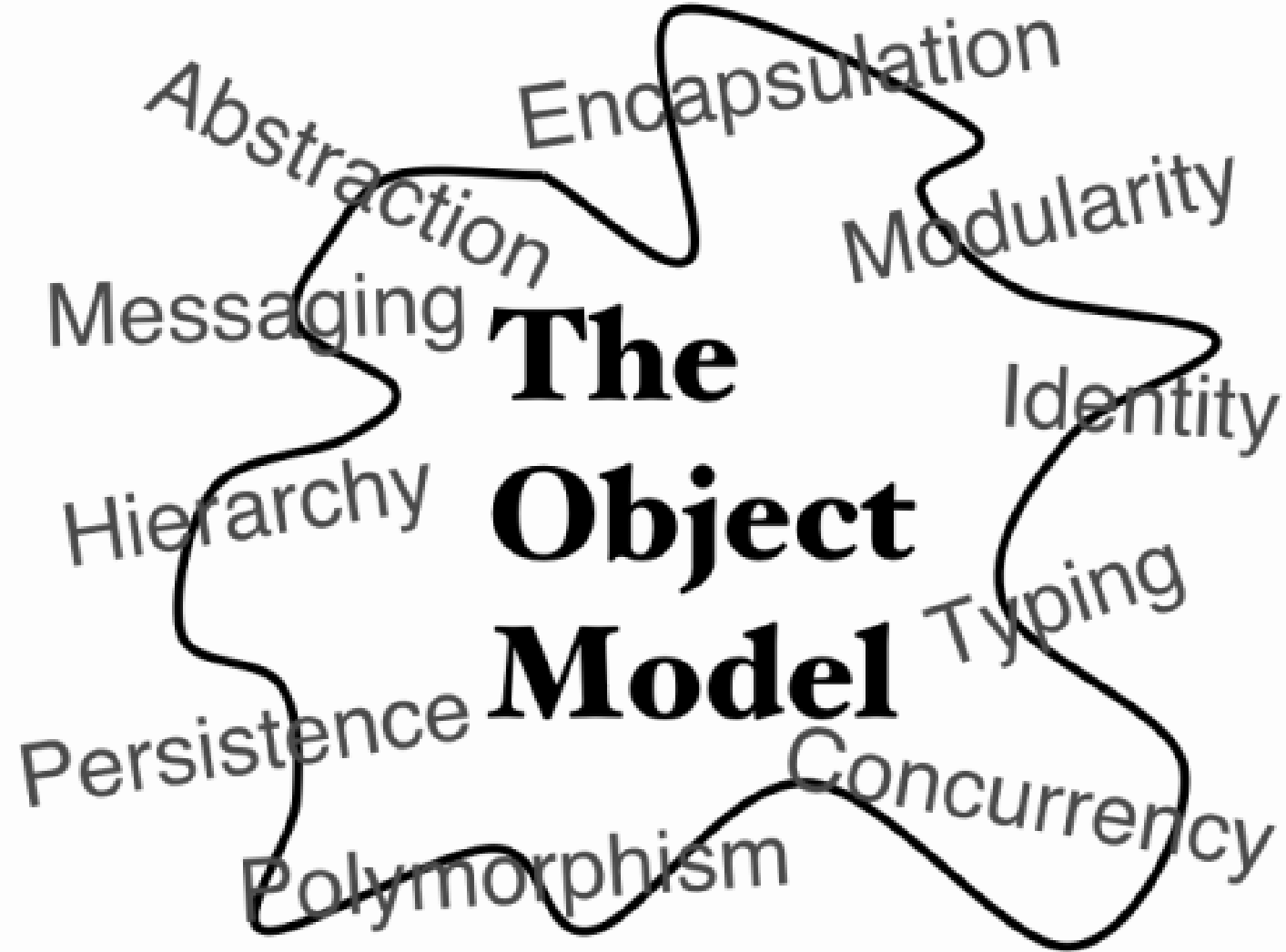
The image shows a software window titled "Baker Maker" with standard Windows-style window controls (minimize, maximize, close). Inside the window, there is a section titled "Baked Goods" containing six checkboxes arranged in two rows and three columns:

Baked Goods		
<input type="checkbox"/> Bagle	<input type="checkbox"/> Cup Cake	<input checked="" type="checkbox"/> Pie
<input type="checkbox"/> Doughnut	<input type="checkbox"/> Pizza	<input type="checkbox"/> Cookie

Below the "Baked Goods" section is a "Selection" section containing five radio buttons, with "Apple" selected:

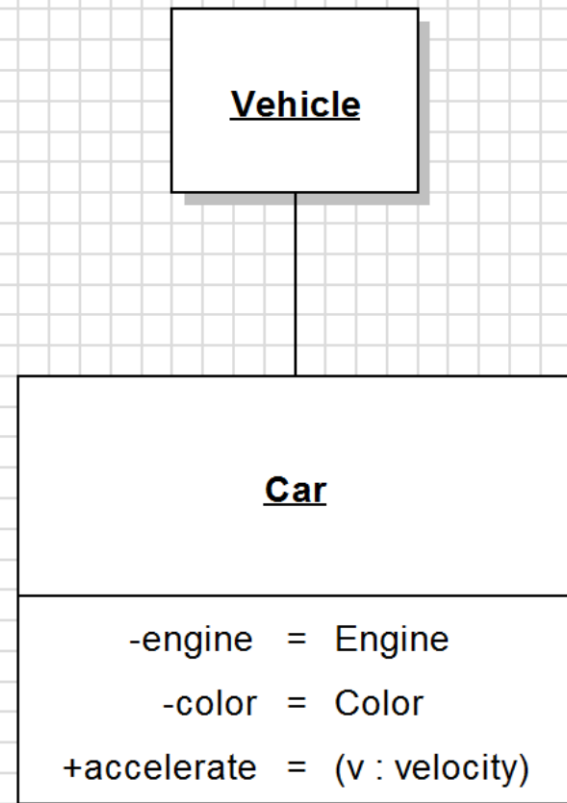
- ☒ Apple
- ☐ Berry
- ☐ Cherry
- ☐ Peach
- ☐ Pecan

At the bottom center of the window is a large "Make" button.



UML Representation

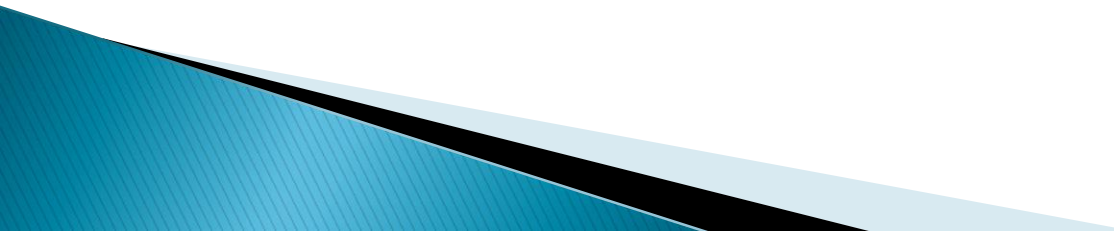
- Abstract class Vehicle
- Class Car has 2 private variables indicated with the - minus sign
- Class Car has 1 public function indicated by the + sign.



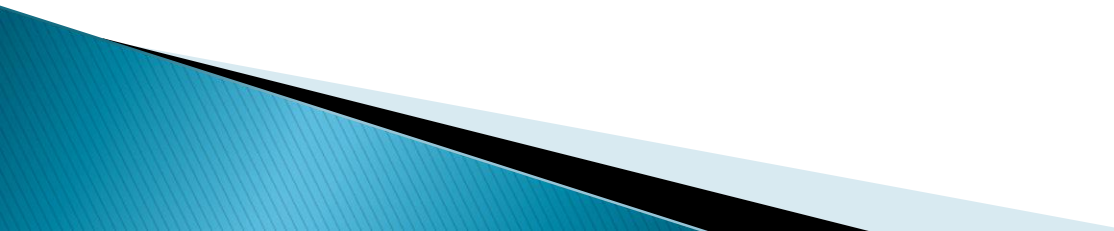
Code Implementation

```
abstract Vehicle  
{  
};
```

```
class Car : Vehicle  
{  
    private Engine engine;  
    private Color color;  
    public void accelerate(Velocity v);  
};
```



Analogy

- ▶ An object is to its class, as a car is to the automobile factory.
 - ▶ Classes are static. They define the capabilities or behavior of an object. They don't get work done.
 - ▶ Objects are entities. They get work done.
- 

State

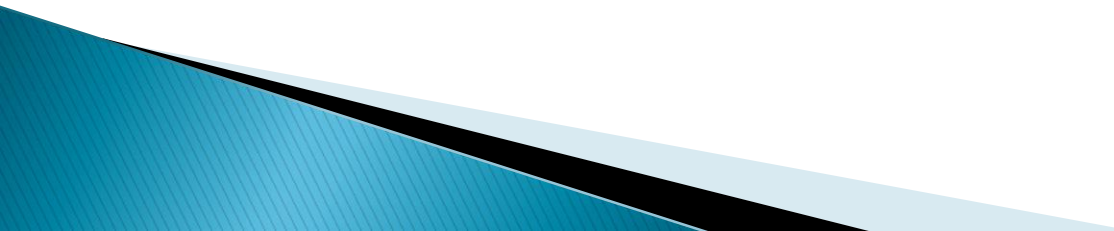
- ▶ This is what an object knows. It is a set of properties, or variables which can take different values at different times in the objects life (dynamic).

State Example

- ▶ An Employee object should generate a paycheck request if the Employee object has a state variable which indicates the Employee is active.



Behavior

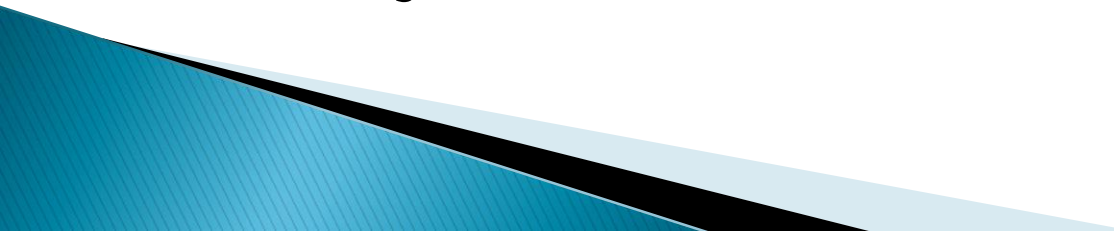
- ▶ This is the set of actions that an object knows how to take. In general, we can group the methods of a class into five categories:
 1. Modifier Change the state of the object
 2. Selector Accesses, but does not change the state
 3. Iterator Operates across all parts of an object
 4. Constructor Creates an object and initializes its state
 5. Destructor Frees the state and destroys the object cleanly
- 

Behavior Example

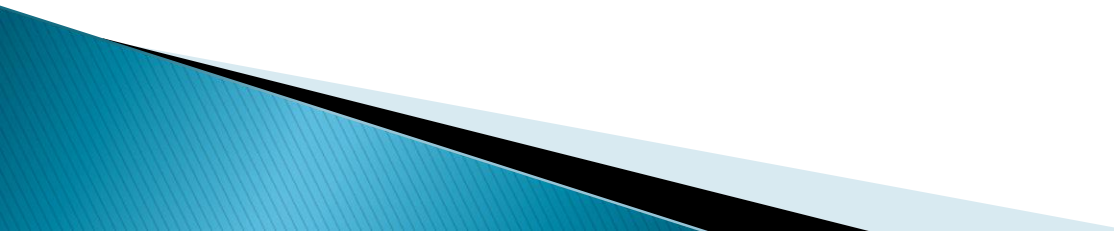
- ▶ Your Pet object knows a set of tricks (behavior) that you can ask it (message it) to perform. C++ calls the “tricks” that an object knows *member functions*.



OO Analysis Techniques

- ▶ Behavior Analysis: Focuses on behavior, vs properties.
 - ▶ Domain Analysis: Study existing systems that do similar things.
 - ▶ Use-Case Analysis: Construct scenarios based on actions that must be performed.
 - ▶ Pattern Scavenging: Look for patterns among existing classes or projects.
 - ▶ Classical Analysis: Recognize entities that share common properties and form a class (or object).
- 

OO Design Techniques

- ▶ Identify the classes and objects at a given level of abstraction.
 - ▶ Identify the semantics of these classes and objects.
 - ▶ Identify the relationships among these classes and objects.
 - ▶ Implement these classes and objects.
- 

OOA + OOD

- ▶ OOA feeds OOD which then can be implemented with OOP. In practice, these separate activities are usually done in an iterative fashion, with more analysis and less implementation up front, then more design, and finally more concentration on implementation.

Modeling the System

- ▶ The central goal of OOAD: Study and understand a software solution via a model of that system.
 - ▶ Like aerodynamic engineers build models of aircraft yet to be built, OOAD models will let you explore design alternatives and test your understanding of the system.
 - ▶ OOAD is faster than constructing the system from scratch.
- 