The Object-Oriented Thought Process

Chapter 1

Introduction to Object-Oriented Concepts

O-O Language Criteria

- Encapsulation
- Inheritance
- Polymorphism
- Composition

Object Wrappers

Even when there are legacy concerns, there is a trend to wrap the legacy systems in object wrappers.

- Wrappers are used to wrap:
  - Legacy code
  - Non-portable code
  - 3rd party libraries
  - etc,.

What is an Object?

In its basic definition, an object is an entity that contains both data and behavior.

- This is the key difference between the more traditional programming methodology, procedural programming, and O-O programming.

Procedural Programming

In procedural programming, code is placed into methods.

- Ideally these procedures then become "black boxes", inputs come in and outputs go out.
  - Data is placed into separate structures, and is manipulated by these methods.

Unpredictable?

First, this means that access to data is uncontrolled and unpredictable.

- Second, because you have no control over who has access to the data, testing and debugging is much more difficult.

Nice Packages

Objects address these problems by combining data and behavior into a nice, complete package.

- Objects are not just primitive data types, like integers and strings.

Data and Behaviors

Procedural programming separates the data of the program from the operations that manipulate the data.

- For example, if you want to send information across a network, only the relevant data is sent with the expectation that the program at the other end of the pipe knows what to do with it.

O-O Programming

The fundamental advantage of O-O programming is that the data and the operations that manipulate the data are both contained in the object.

- For example, when an object is transported across a network, the entire object, including the data and behavior, goes with it.

Object Data

The data stored within an object represents the *state* of the object.

– In O-O programming terminology, this data is called *attributes*.

Object Behaviors

The *behavior* of an object is what the object can do.

- In procedural languages the behavior is defined by procedures, functions, and subroutines.

Object Behaviors

In O-O programming terminology these behaviors are contained in *methods,* and you invoke a method by sending a *message* to it.

What Exactly Is a Class?

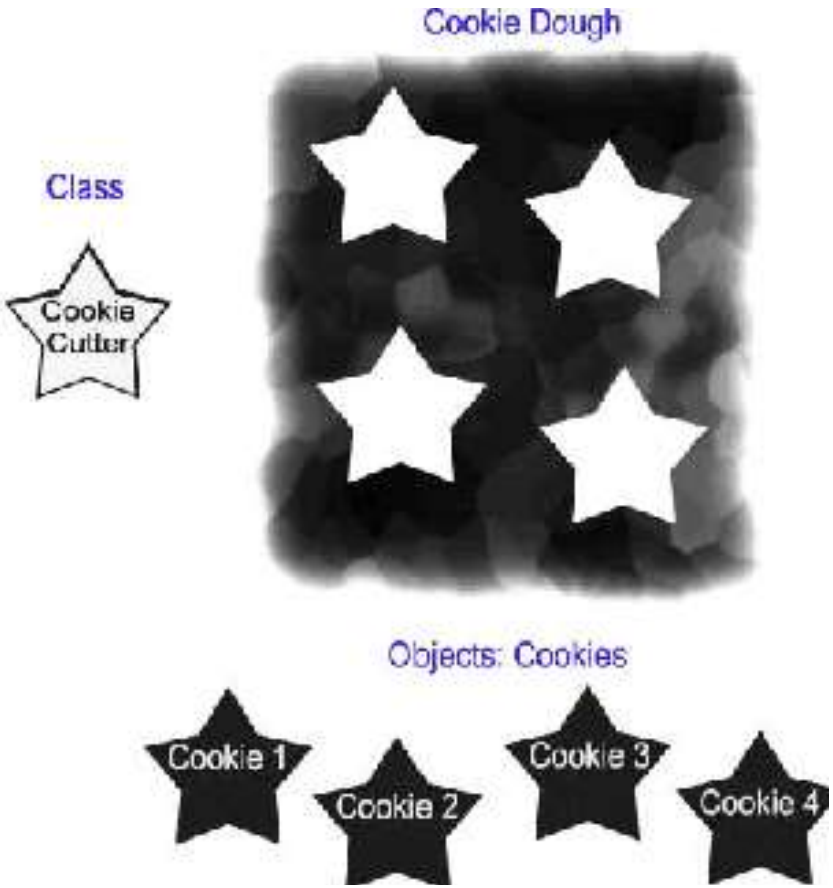
A *class* is a blueprint for an object.

- When you instantiate an object, you use a class as the basis for how the object is built.

What Exactly Is a Class?

An object cannot be instantiated without a class.

- Classes can be thought of as the templates, or cookie cutters, for objects as seen in the next figure.



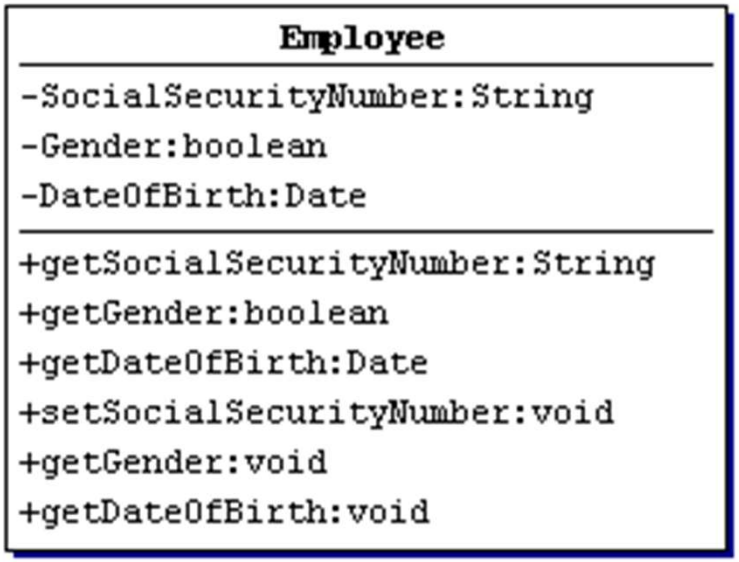
Higher Level Data Types?

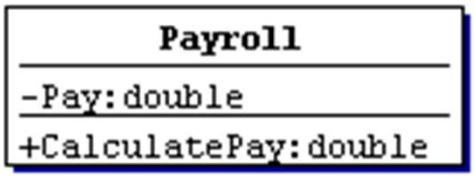
A class can be thought of as a sort of higher-level data type.

- For example, just as you create an integer or a float:

```
int x;
float y;
```

Modeling a Class in UML



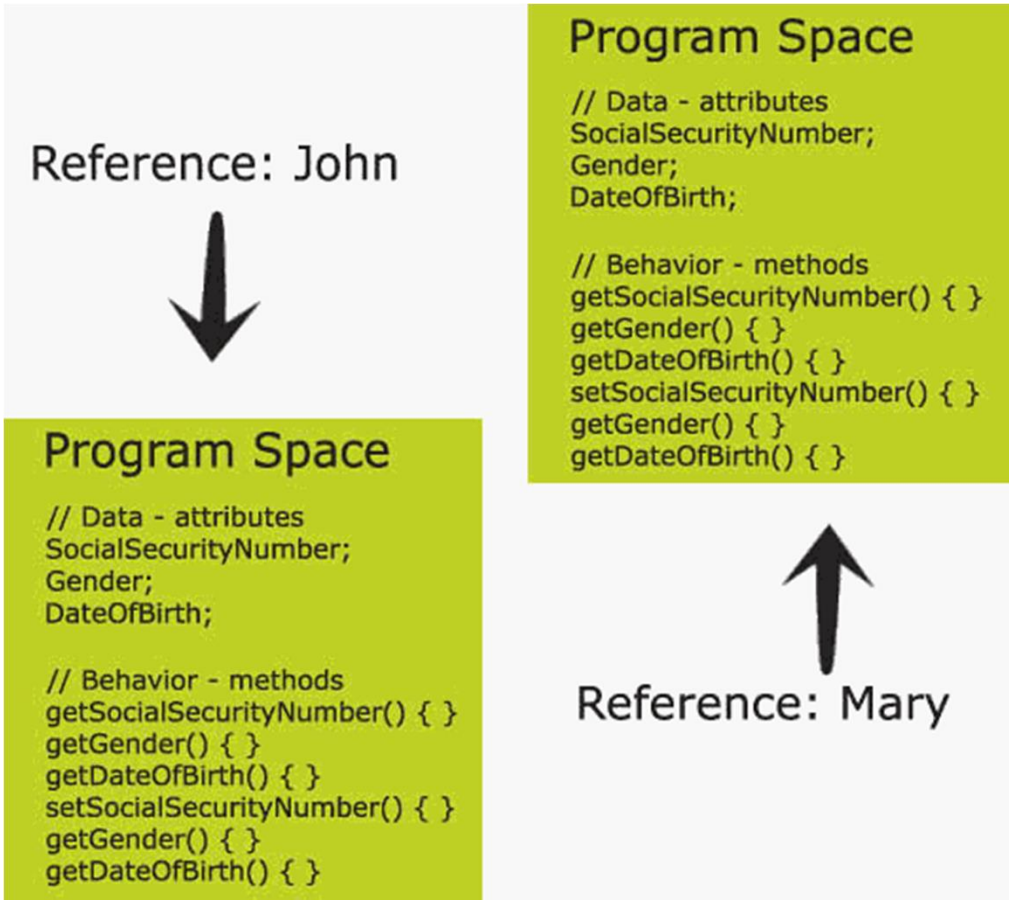


Method Signatures

The following is all the user needs to know to effectively use the methods:

1. The name of the method
2. The parameters passed to the method
3. The return type of the method

Program Spaces



Encapsulation

One of the primary advantages of using objects is that the object need not reveal all its attributes and behaviors.

- In good O-O design (at least what is generally accepted as good), an object should only reveal the interfaces needed to interact with it.

Encapsulation

Details not pertinent to the use of the object should be hidden from other objects.

- This is called *encapsulation*.

Interfaces

Any behavior that the object provides must be invoked by a message sent using one of the provided interfaces.

- The interface should completely describe how users of the class interact with the class.
- methods that are part of the interface are designated as public.

Interfaces

Interfaces do not normally include attributes, only methods.

- As discussed earlier in this module, if a user needs access to an attribute, then a method is created to return the attribute.

Implementations

Only the public attributes and methods are considered the interface.

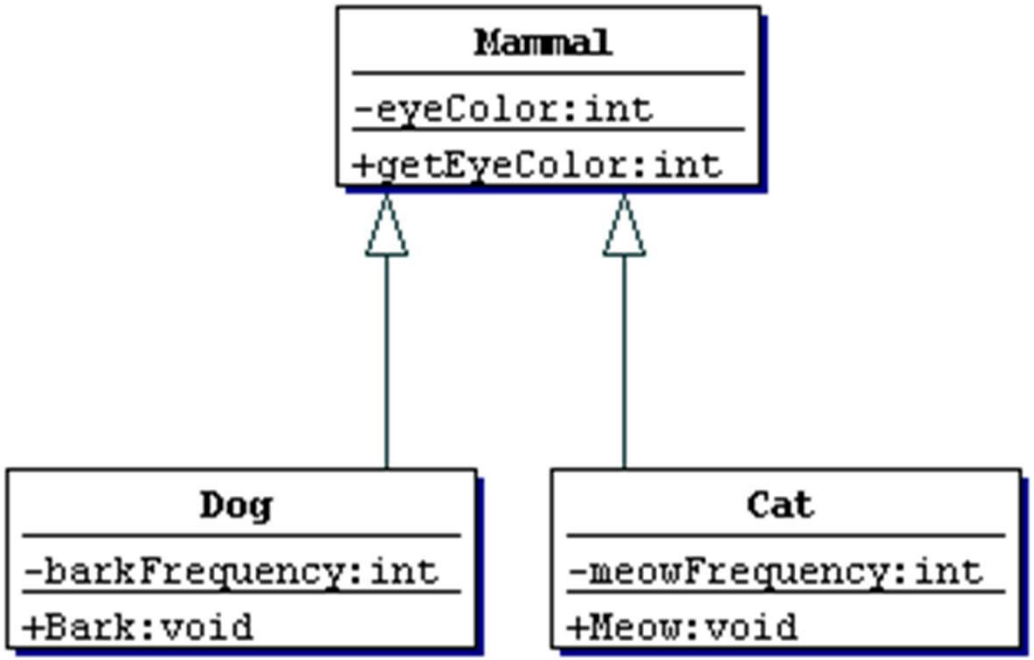
- The user should not see any part of the implementation interacting with an object solely through interfaces.
- In previous example, for instance the Employee class, only the attributes were hidden.
- Thus, the implementation can change and it will not affect the user's code.

Inheritance

Inheritance allows a class to inherit the attributes and methods of another class.

- This allows you to create brand new classes by abstracting out common attributes and behaviors.

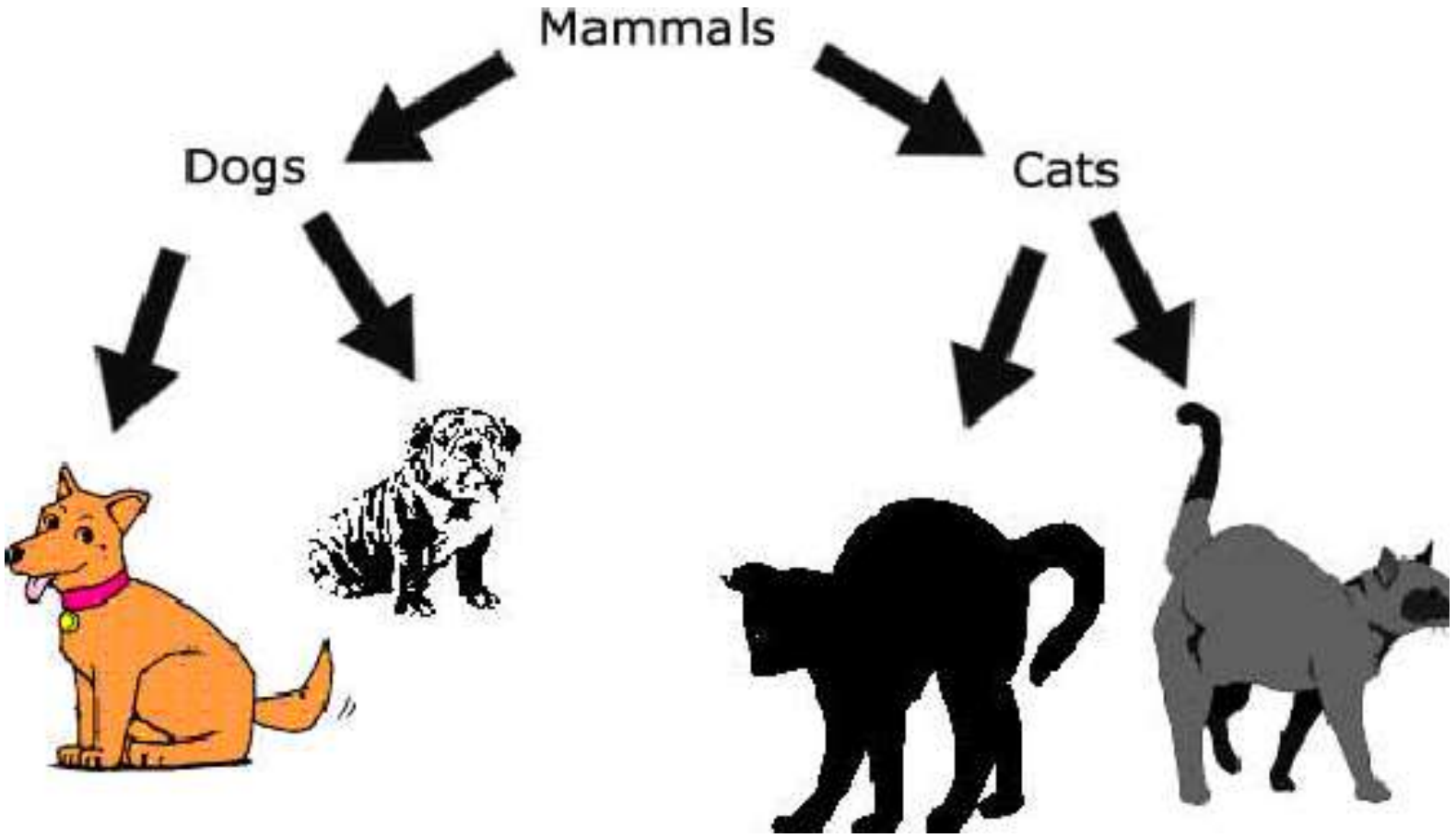
Mammal Hierarchy



Superclasses and Subclasses

- The superclass, or parent class, contains all the attributes and behaviors that are common to classes that inherit from it.

Mammal Hierarchy



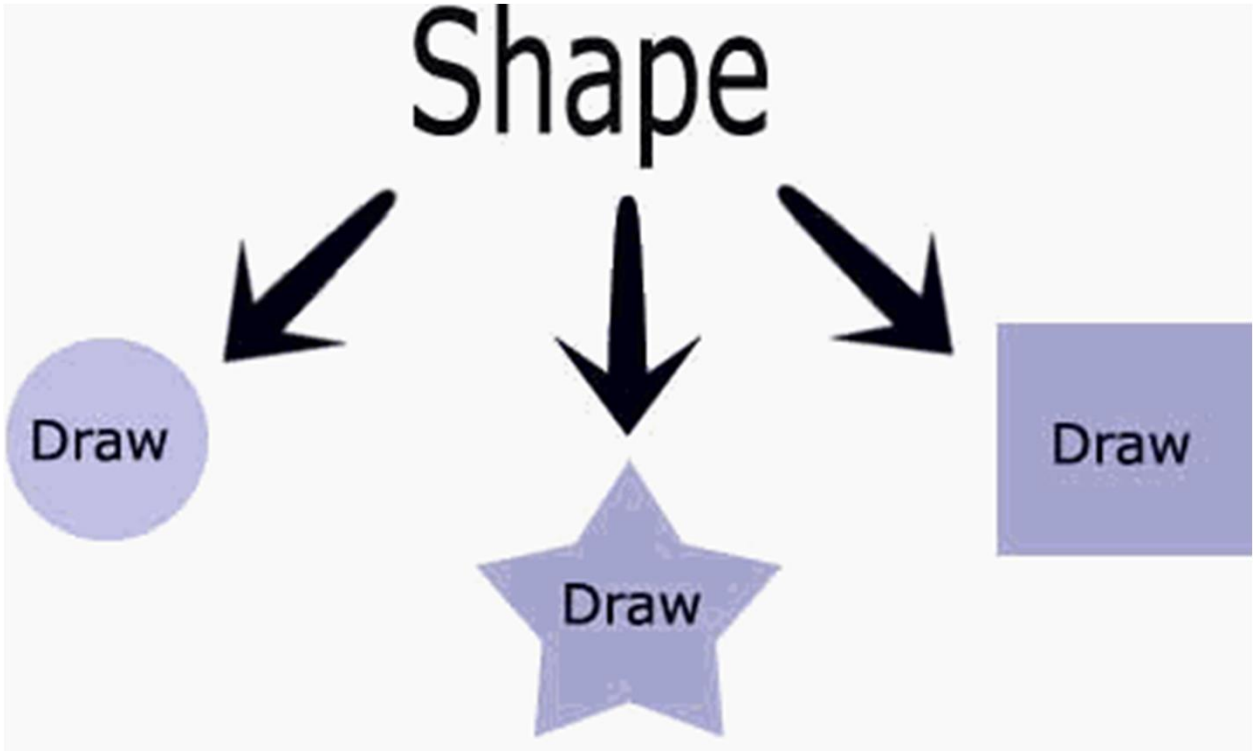
Is-a Relationships

In the Shape example, Circle, Square, and Star all inherit directly from Shape.

- This relationship is often referred to as an *is-a relationship* because a circle *is a* shape.

Polymorphism

Polymorphism literally means many shapes.



Polymorphism

The Shape object cannot draw a shape, it is too abstract (in fact, the Draw() method in Shape contains no implementation).

- You must specify a concrete shape.
- To do this, you provide the actual implementation in Circle.

Polymorphism

In short, each class is able to respond differently to the same Draw method and draw itself.

- This is what is meant by polymorphism.

Polymorphism

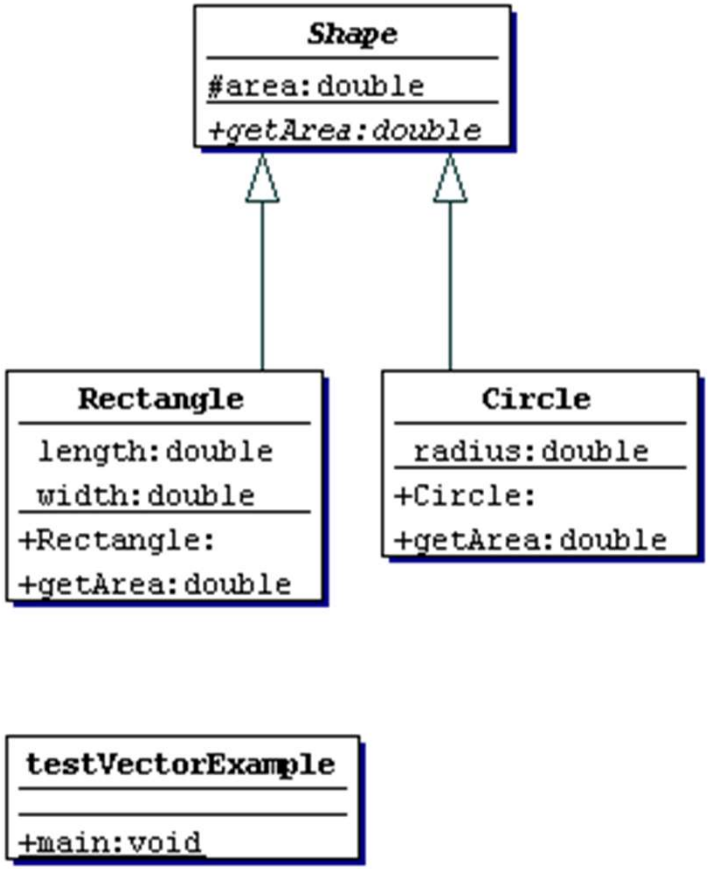
When a method is defined as abstract, a subclass must provide the implementation for this method.

- In this case, Shape is requiring subclasses to provide a getArea() implementation.

Polymorphism

- If a subclass inherits an abstract method from a superclass, it *must* provide a concrete implementation of that method, or else it will be an abstract class itself (see the following figure for a UML diagram).

Shape UML Diagram



Composition

It is natural to think of objects as containing other objects.

- A television set contains a tuner and video display.

Composition

A computer contains video cards, keyboards, and drives.

- Although the computer can be considered an object unto itself, the drive is also considered a valid object.

Composition

In fact, you could open up the computer and remove the drive and hold it in your hand.

- Both the computer and the drive are considered objects. It is just that the computer contains other objects such as drives.
- In this way, objects are often built, or *composed*, from other objects: This is *composition*.

Has-a Relationships

While an inheritance relationship is considered an Is-a relationship for reasons already discussed, a composition relationship is termed a Has-a relationship.

Has-a Relationships

- Using the example in the previous section, a television Has-a a tuner and Has-a video display.
- A television is obviously not a tuner, so there is no inheritance relationship.

Has-a Relationships

In the same vein, a computer Has-a video card, Has-a keyboard, and Has-a disk drive.

- The topics of inheritance, composition and how they relate to each other is covered in great detail later in the course.