



Object Oriented Design



Object Orientation

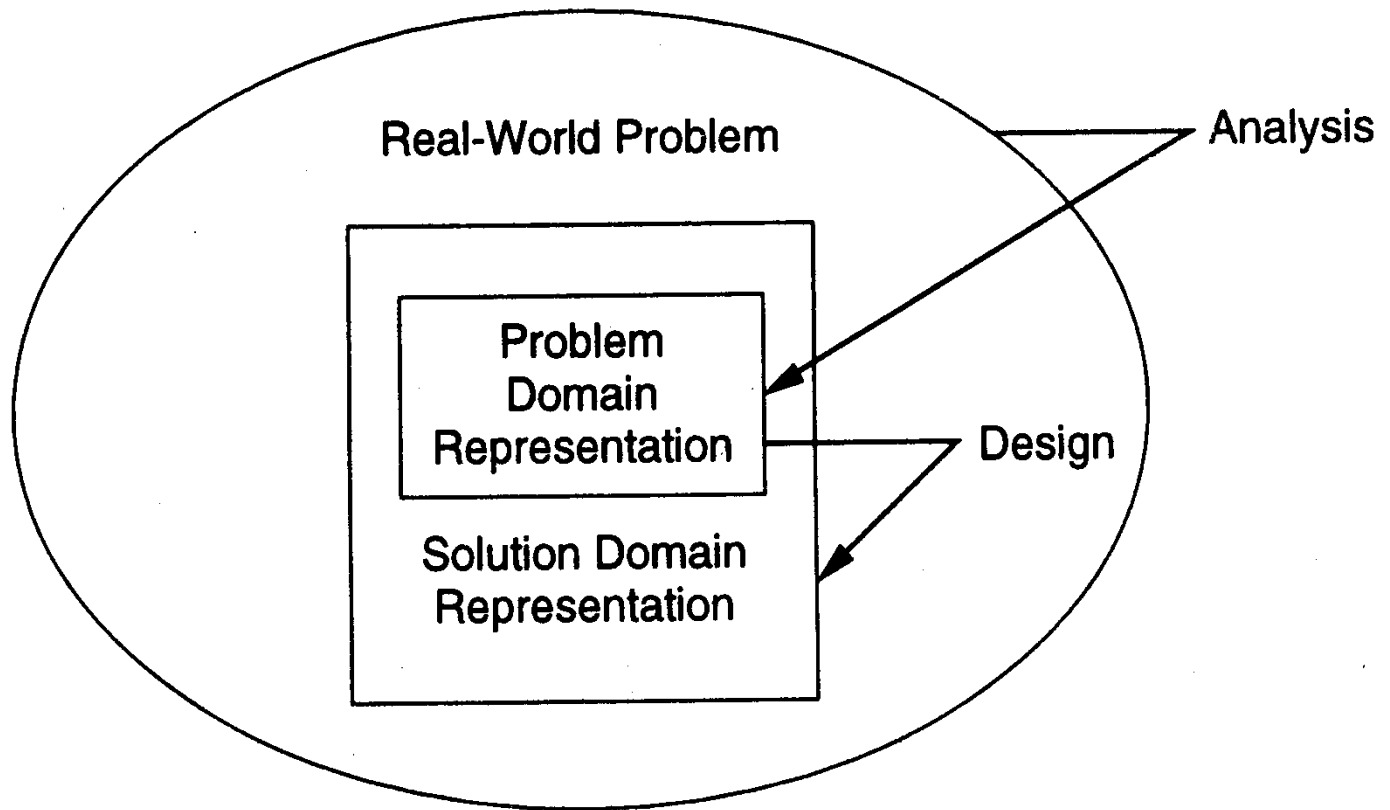
- Traditional procedural systems separate data and procedures, and model these separately
- Object orientation –views data and functions together; data abstraction is the basis
- The purpose of OO design is to define the classes in the system to be built and their relationships
 - A class is a blue print for an object.



OOA and OOD

- OO techniques can be used in analysis (requirements) as well as design
- The methods and notations are similar
- In Object Oriented Analysis (OOA) we model the problem, while in Object Oriented Design (OOD) we model the solution
- Often OOA structures are subsumed in the solution domain structures
- The line between OOA and OOD is not fixed

OOA and OOD...





OO Concepts

- Encapsulation – grouping of related ideas into one unit which we can refer to by a single name
 - Eg. procedures, functions, packages
- In OO, object is the unit; encapsulates state and provides interface to access and modify
 - **Object** is an instance of a class



OO Concepts..

- Information hiding – use encapsulation to restrict external visibility
- OO encapsulates the data, provides limited access, visibility
- Information hiding can be provided without OO – is an old concept



OO Concepts...

- State retention – functions, procedures do not retain state; an object is aware of its past and maintains state
- Identity – each object can be identified and treated as a distinct entity
- Behavior – state and services together define the behavior of an object, or how an object responds



OO Concepts..

- Messages – through which a sender object conveys to a target object a request
- For requesting O1
 - must have – a handle for O2
 - name of the operation
 - information on operations that O2 requires
- General format `O2.method(args)`



OO Concepts..

- Classes – a class is a stencil from which objects are created; defines the structure and services. A class has
 - An interface which defines which parts of an object can be accessed from outside
 - Body that implements the operations
 - Instance variables to hold object state
- A class defines the attributes and operations
- Objects and classes are different; class is a type, object is an instance
- State and identity is of objects



OO Concepts – access

- Operations in a class can be
 - Public: accessible from outside
 - Private: accessible only from within the class
 - Protected: accessible from within the class and from within subclasses
- There are some constructor and destructor operations
 - Used to modify attributes



Inheritance

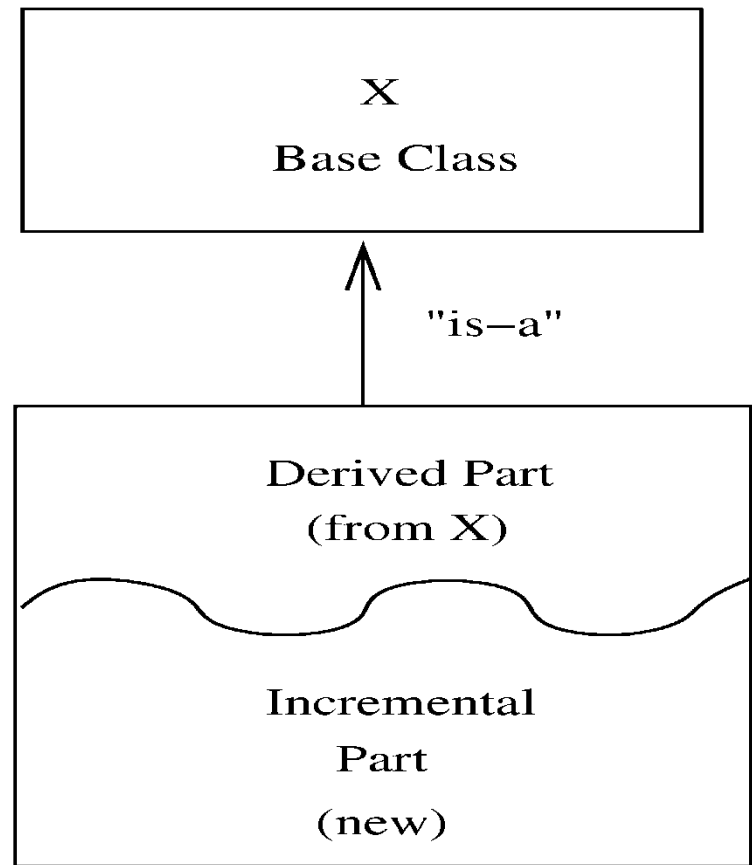
- Inheritance is unique to OO
 - not there in function-oriented languages/models
- Inheritance by class B from class A is the facility by which B implicitly gets the attributes and operations of A as part of itself
- Attributes and methods of A are reused by B
- When B inherits from A, B is the *subclass* or *derived* class and A is the *base* class or *superclass*



Inheritance..

- A subclass B generally has a derived part (inherited from A) and an incremental part (is new)
- Hence, B needs to define only the incremental part
- Creates an “is-a” relationship
 - objects of type B are also objects of type A

Inheritance...



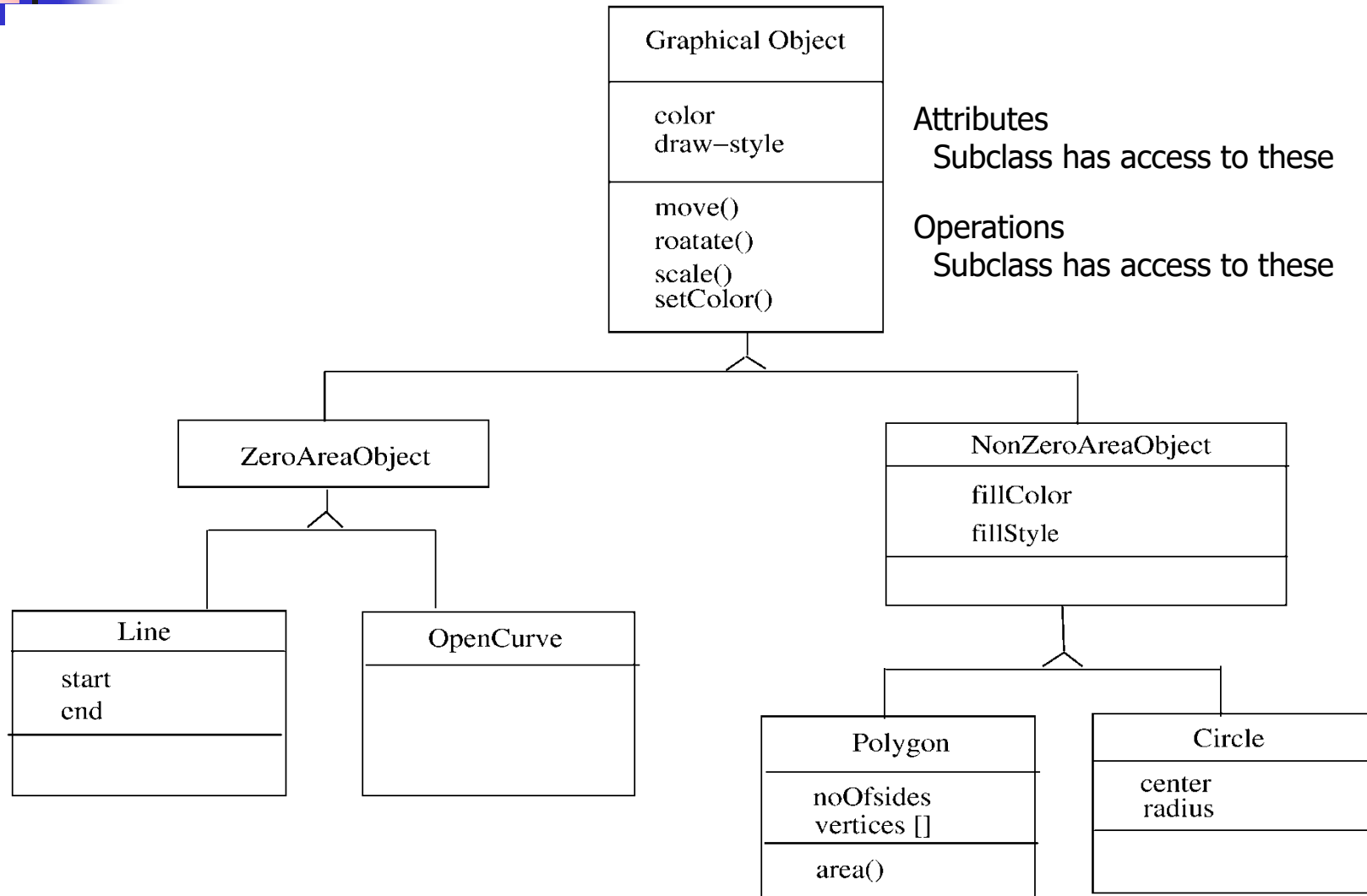
Y – Derived class
OO Design



Inheritance...

- The inheritance relationship between classes forms a class hierarchy
- In models, hierarchy should represent the natural relationships present in the problem domain
- In a hierarchy, all the common features can be accumulated in a superclass
- An existing class can be a specialization of an existing general class – is also called generalization-specialization relationships

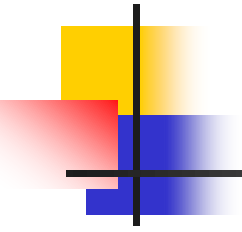
Hierarchy Class Example



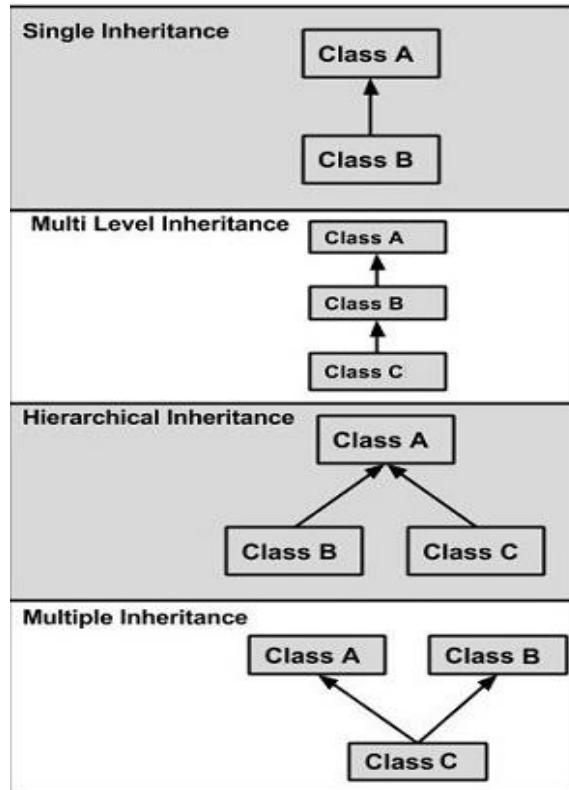


Inheritance...Types

- Single inheritance – a subclass inherits from only one superclass
 - Class hierarchy is a tree
- Multiple inheritance – a class inherits from more than one class
 - Can cause runtime conflicts
 - Repeated inheritance - a class inherits from a class but from two separate paths

- 
-
- **Single inheritance** enables a derived class to **inherit** properties and behavior from a **single** parent class. It allows a derived class to **inherit** the properties and behavior of a base class, thus enabling code reusability as well as adding new features to the existing code.

Single vs Multiple





Inheritance...

- Strict inheritance – a subclass takes all features of parent class
 - Only adds features to specialize it
- Non-strict: when some of the features have been redefined
- Strict inheritance supports “is-a” cleanly and has fewer side effects



Inheritance and Polymorphism

- Inheritance brings polymorphism, i.e. an object can be of different types
- An object of type B is also an object of type A
- Hence an object has a static type and a dynamic type
 - Implications on type checking
 - Also brings dynamic binding of operations which allows writing of general code where operations do different things depending on the type



END
