Object-Oriented Design Principles

Oracle® Certified Professional, Java® SE 7 Programmer

Module 3

Module 3 - Objectives

- Write code that declares, implements, and/or extends interfaces
- Choose between interface inheritance and class inheritance
- Develop code that implements "is-a" and/or "has-a" relationships.
- Apply object composition principles
- Design a class using the Singleton design pattern
- Write code to implement the DAO pattern
- Design and create objects using a factory, and use factories from the API

Implementing an interface

- To implement an interface, you must simply state that your class implements the interface, and provide overriding methods for the implementation of the interface.
- An interface can **extend** another interface. It does not implement it.
- Remember classes **implement** interfaces, and interfaces **extend** interfaces.

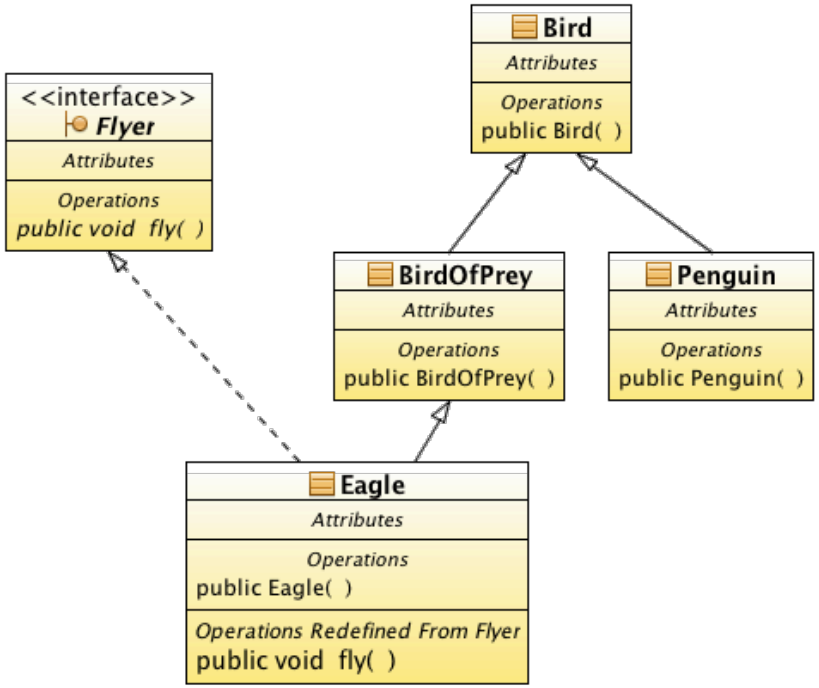
Inheritance

- There are two main mechanisms for inheritance in Java; class and interface.
- Java supports only single-class inheritance.
- All classes extend from Object.
- Java supports multiple interface inheritance.
- Promotes class reuse through specialization
- Uses polymorphism

“is-a” relationships

- This is based on class inheritance, or interface implementation.
- An Eagle IS-A Bird.
- An Eagle IS-A Bird of Prey
- An Eagle IS-A Flyer
- A Penguin IS-NOT a Flyer.

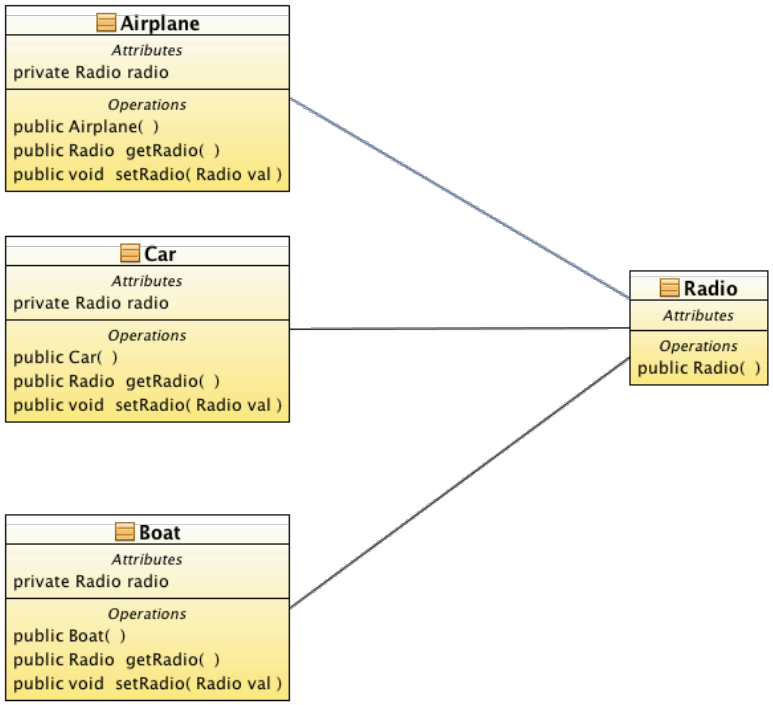
“is-a” relationships (cont.)



“has-a” relationships

- This is based on usage rather than inheritance, or interface implementation.
- A Car HAS-A Radio
- An Airplane HAS-A Radio
- A Boat HAS-A Radio

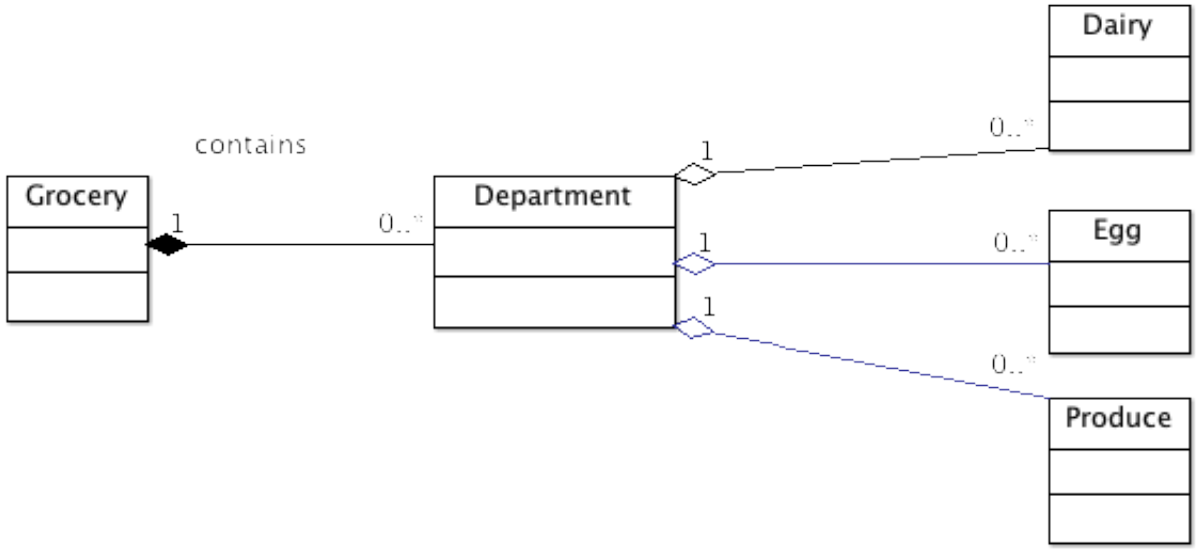
“has-a” relationships



Object Composition

- A means of constructing more complex objects from simple building blocks.
- A composition “has-a” relationship between the main Object and its composed parts; e.g. a car has tires, radio, steering wheel, etc.

Object Composition (cont.)



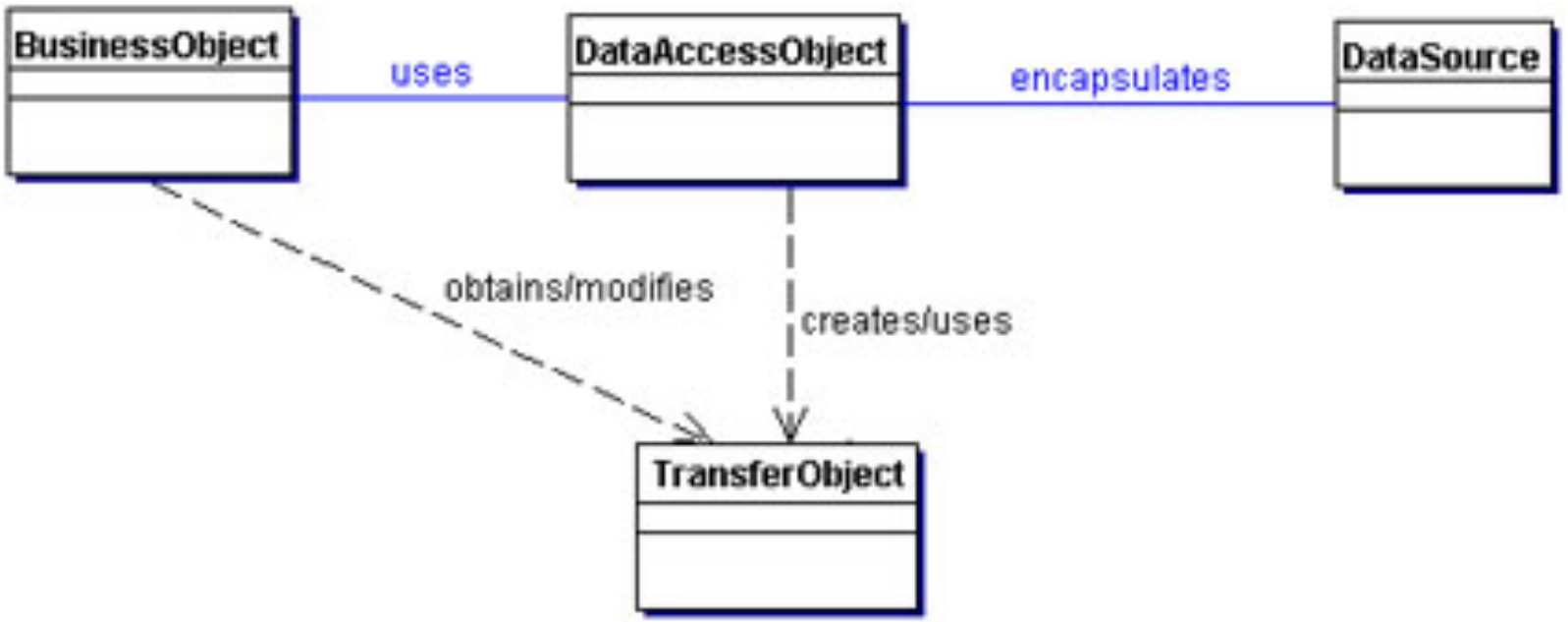
Singleton Design Pattern

- The typical Singleton design uses a private constructor, and has a static method to get an instance of the object.
- Since the constructor is private, the object is implicitly final.
- Runtime.getRuntime(); is a singleton.

Data Access Object (DAO) Design Pattern

- This is a means of abstracting the data access from the implementation.
- Uses Transfer objects to maintain data.
- Traditionally used with a database, but is a more general design for use with any data store.

Data Access Object (DAO) Design Pattern (cont.)



Core J2EE Pattern

Factory Pattern

- A GoF Creational Pattern.
- Designs a method for obtaining an instance of an object from an interface.
- A concrete implementation is used to create the object.
- Class.newInstance(); is a a reflective example for creating an object in Java.

creative commons

Attribution-NonCommercial-ShareAlike 3.0 Unported