Introduction to Object-Oriented Programming

- Objects and classes
- Abstract Data Types (ADT)
- Encapsulation and information hiding
- Aggregation
- Inheritance and polymorphism

Pure Object-Oriented Languages

Five rules [Source: Alan Kay]:

- Everything in an object.
- A program is a set of objects telling each other what to do by sending messages.
- Each object has its own memory (made up by other objects).
- Every object has a type.
- All objects of a specific type can receive the same messages.

Java breaks some of these rules in the name of efficiency.

The Object Concept

- An object is an *encapsulation* of data.

- An object has
  - identity (a unique reference),
  - state, also called characteristics
  - behavior

- An object is an instance of an *abstract data type*.
- An abstract data type is implemented via a *class*.

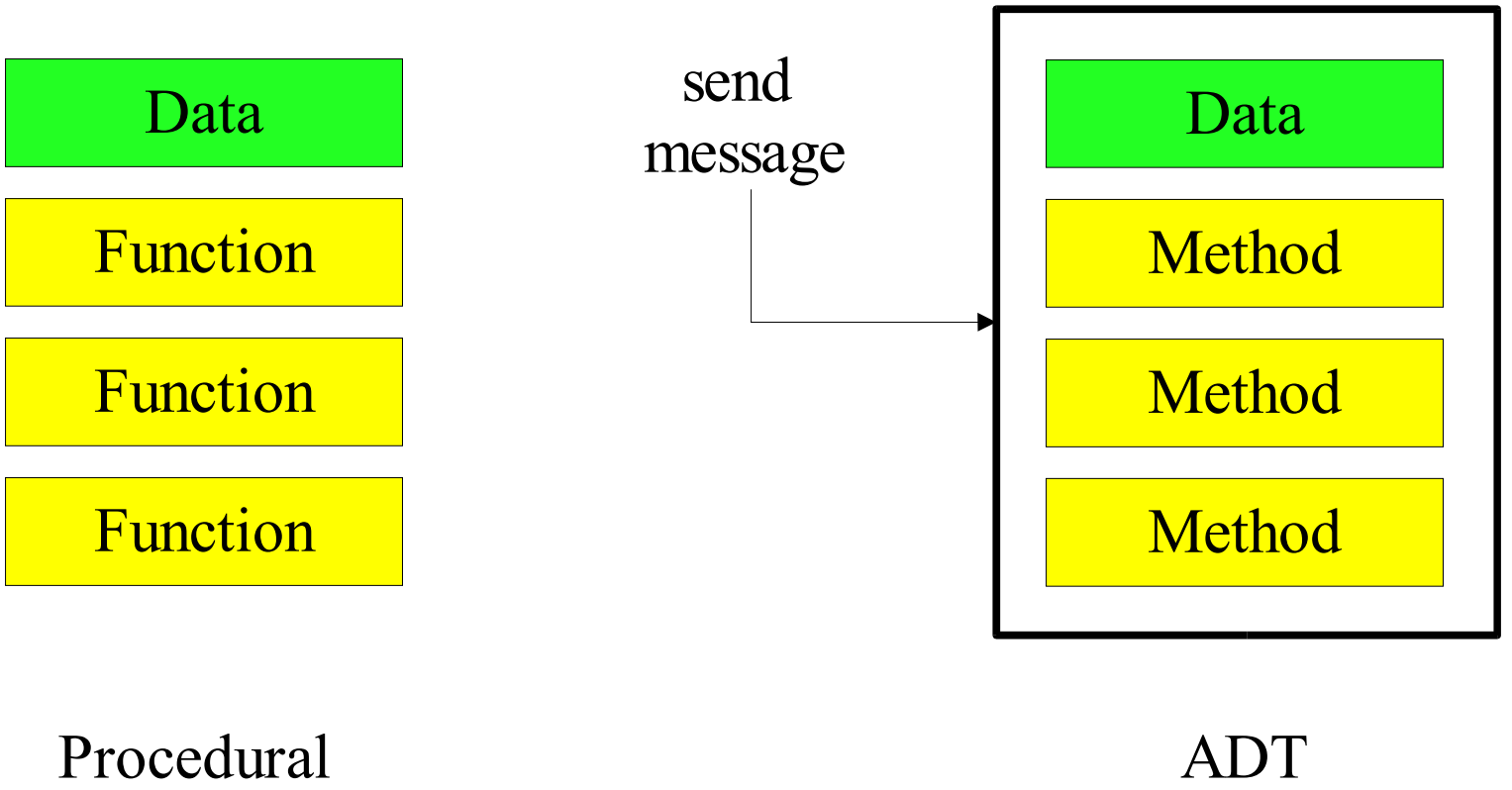
Abstract Data Type (ADT)

- An ADT is a collection of *objects* (or *values*) and a corresponding set of *methods*.
- An ADT encapsulates the data representation and makes data access possible at a higher level of abstraction.

- Example 1: A set of vehicles with operations for starting, stopping, driving, get km/liter, etc..
- Example 2: A time interval, start time, end time, duration, overlapping intervals, etc.

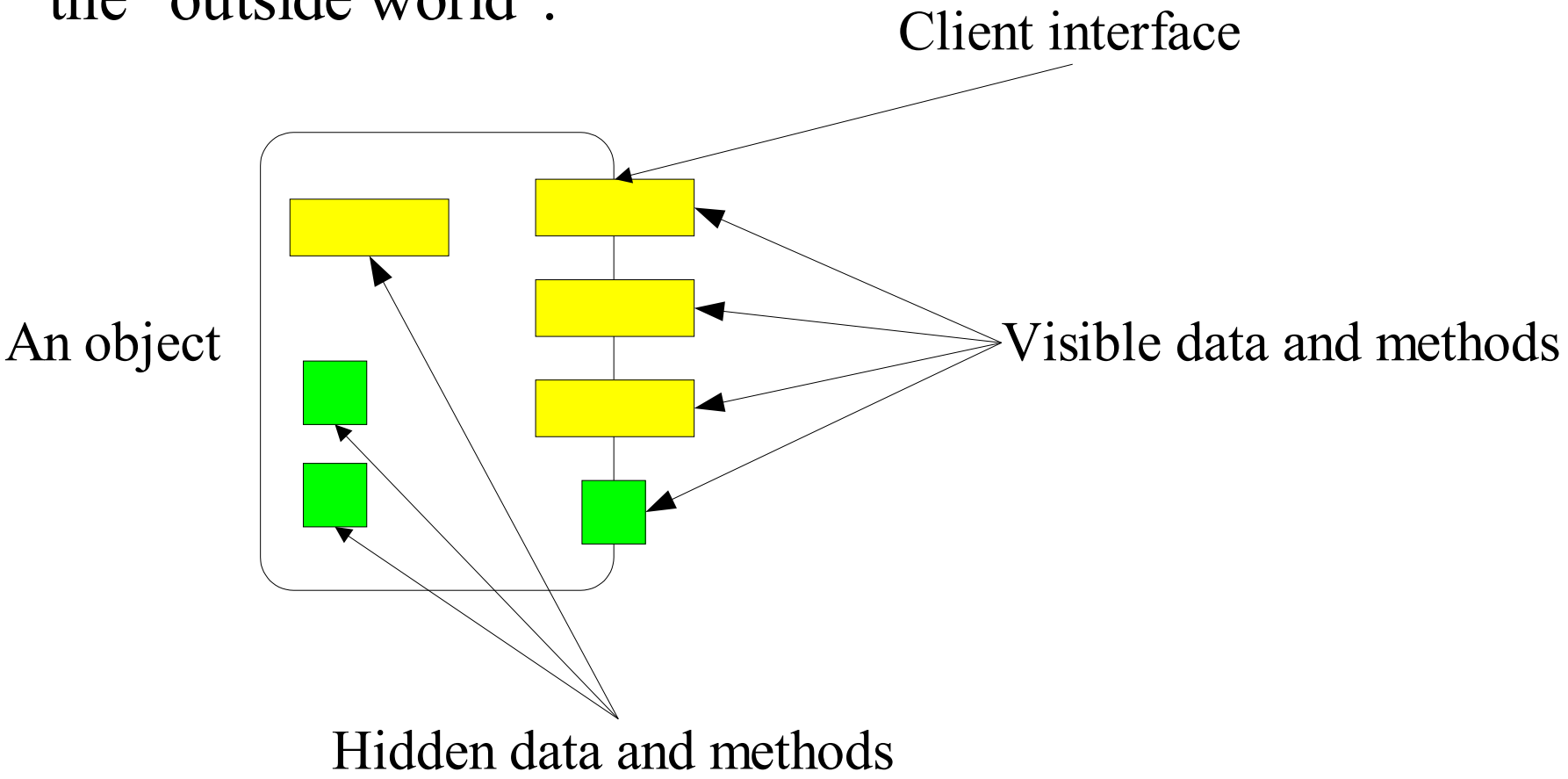
Encapsulation and Information Hiding

- Data can be encapsulated such that it is invisible to the "outside world".
- Data can only be accessed via methods.



Encapsulation and Information Hiding, cont.

- What the "outside world" cannot see it cannot depend on!
- The object is a "fire-wall" between the object and the "outside world".
- The hidden data and methods can be changed without affecting the "outside world".



Class vs. Object

Class

- A description of the *common properties* of a set of objects.
- A concept.
- A class is a part of a program.

- Example 1: Person

- Example 2: Album

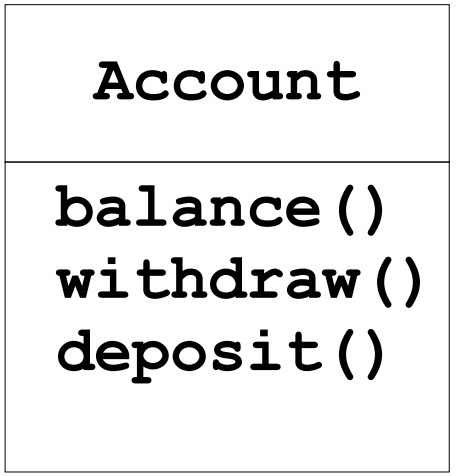
Object

- A representation of the *properties* of a single instance.
- A phenomenon.
- An object is part of data and a program execution.

- Example 1: Bill Clinton, Bono, Viggo Jensen.
- Example 2: A Hard Day's Night, Joshua Tree, Rickie Lee Jones.

Type and Interface

- An object has type and an interface.



Type

Interface

- To get an object `Account a = new Account()`
- To send a message `a.withdraw()`

Instantiating Classes

- An instantiation is a mechanism where objects are created from a class.
- Always involves storage allocation for the object.
- A mechanism where objects are given an initial state.

Static Instantiating

- In the declaration part of a program.
- A static instance is implicitly created

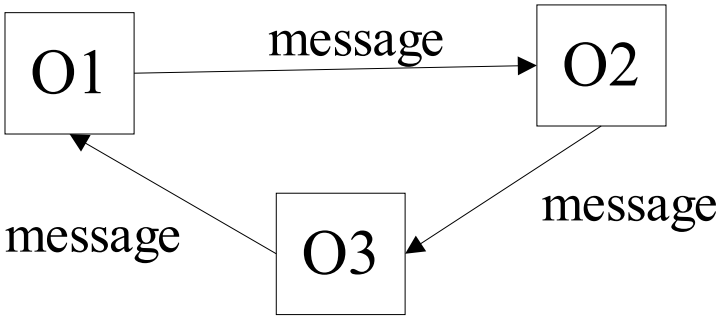
Dynamic Instantiating

- In the method part of a program.
- A dynamic instance is created explicitly with a special command.

Interaction between Objects

- Interaction between objects happens by *messages* being send.
- A message activates a method on the calling object.

- An object O1 interacts with another object O2 by calling a method on O2 (must be part of the client interface).
  - "O1 sends O2 a message"
- O1 and O2 must be *related* to communicate.
- The call of a method corresponds to a procedure call in a non-object-oriented language such as C or Pascal.



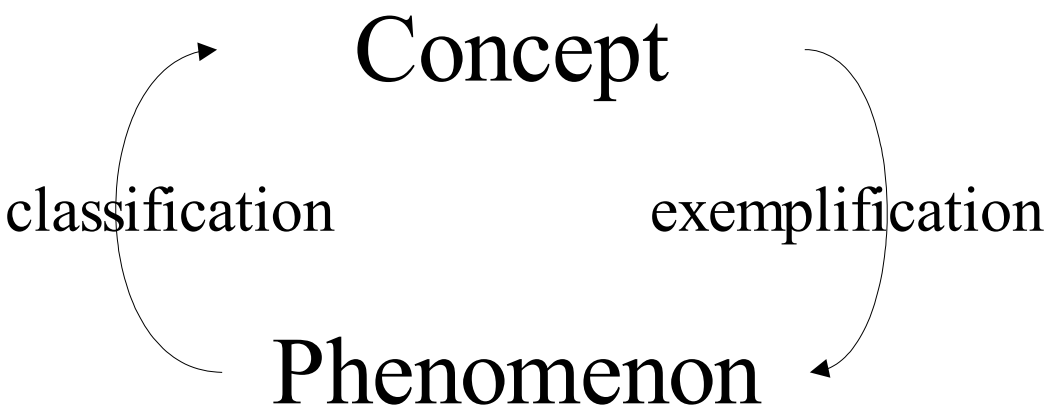
Phenomenon and Concept

- A *phenomenon* is a thing in the "real" world that has individual existence.
- A *concept* is a generalization, derived from a set of phenomena and based on the common properties of these phenomena.

- Characteristics of a concept
  - A name
  - *Intension*, the set of properties of the phenomenon
  - *Extension*, the set of phenomena covered by the concept.

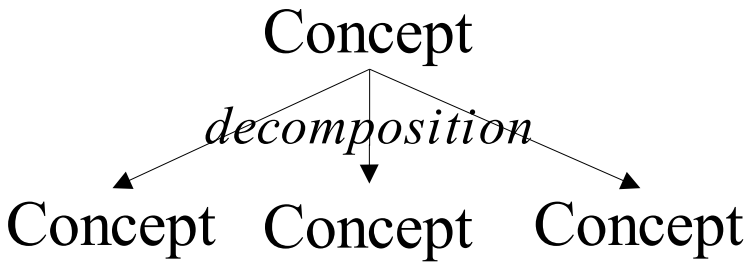
Classification and Exemplification

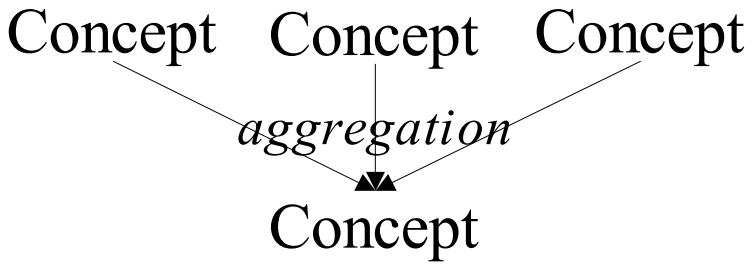
- A *classification* is a description of which phenomena that belongs to a concept.
- An *exemplification* is a phenomenon that covers the concept



Aggregation and Decomposition

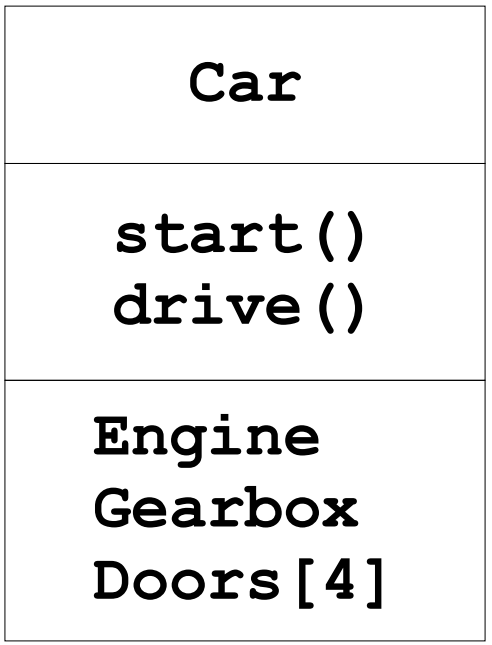
- An *aggregation* consists of a number of (sub-)concepts which collectively is considered a new concept.
- A *decomposition* splits a single concept into a number of (sub-)concepts.



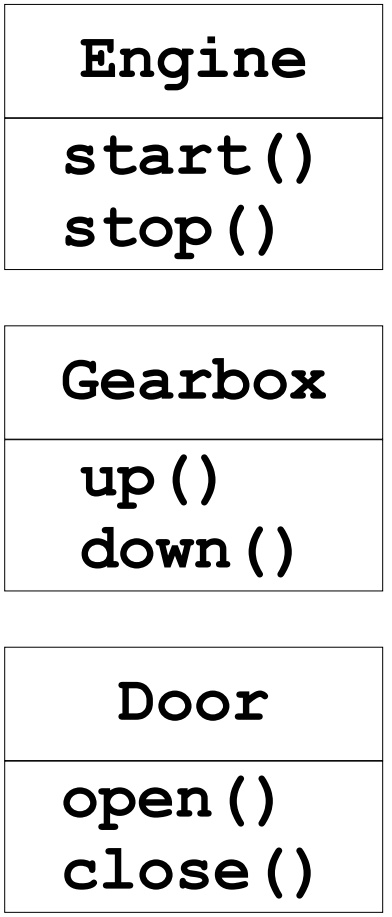


Aggregation and Decomposition, Example

- Idea: make new objects by combining existing objects.
- *Reusing the implementation!*



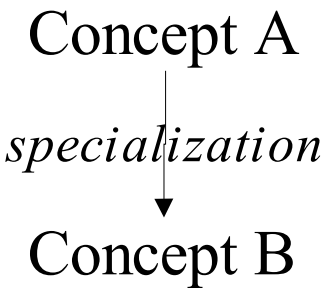
new class

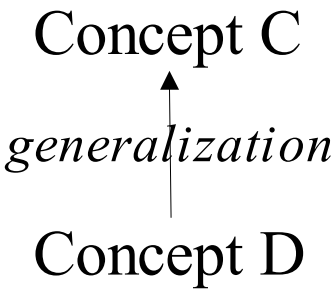


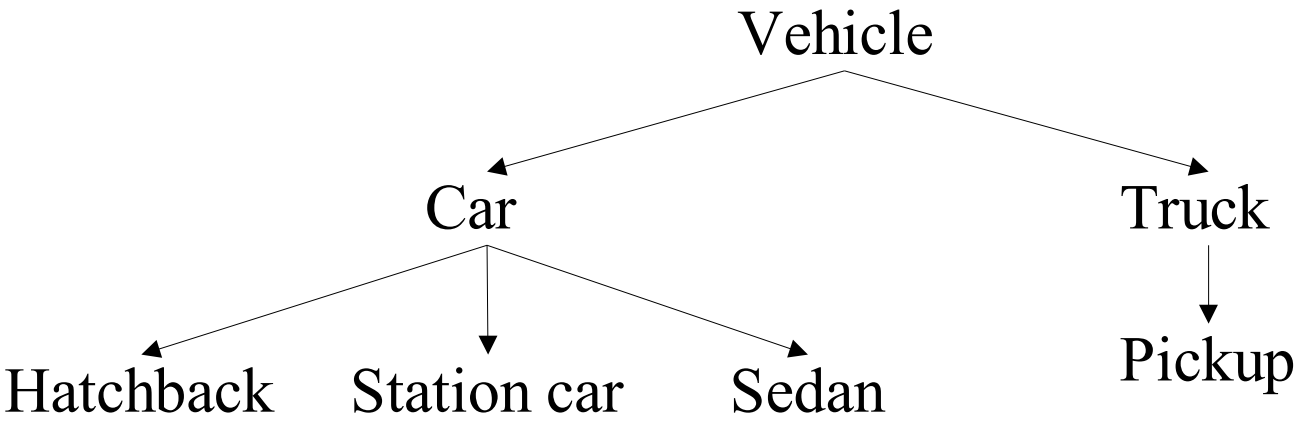
existing classes

Generalization and Specialization

- *Generalization* creates a concept with a broader scope.
- *Specialization* creates a concept with a narrower scope.
- *Reusing the interface!*

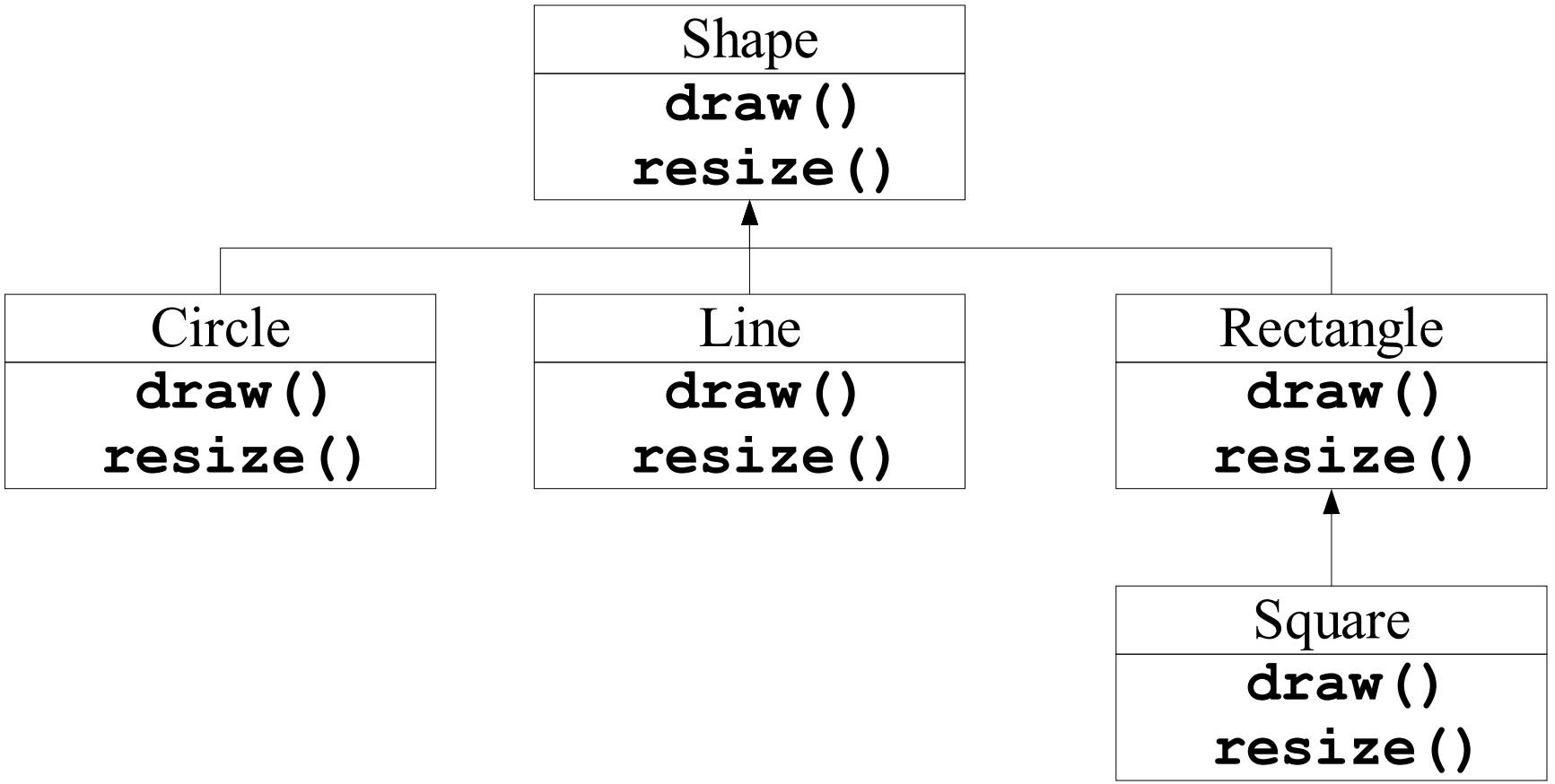






Generalization and Specialization, Example

- *Inheritance*: get the interface from the general class.
- Objects related by inheritance are all of the same type.



Code Example

```
void doSomething(Shape s){
  s.draw();  // "magically" calls on specific class
  s.resize();
}
Circle c = new Circle();
Line l = new Line();
Rectangle r = new Rectangle();

doSomething(c);          // dynamic binding
doSomething(l);
doSomething(r);
```

- *Polymorphism*: One piece of code works with all shape objects.
- *Dynamic binding*: How polymorphism is implemented.

Structuring by Program or Data?

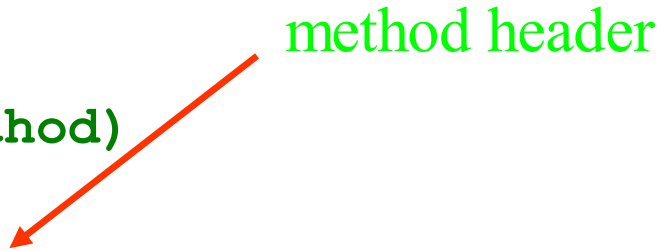
- What are the actions of the program vs. which data does the program act on.
- *Top-down:* Stepwise program refinement
- *Bottom-up*: Focus on the stable data parts then add methods

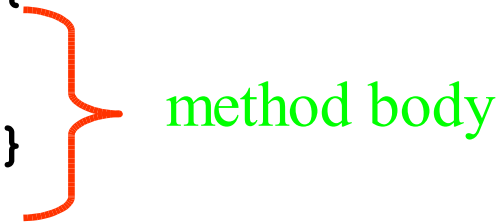
- Object-oriented programming is bottom-up. Programs are structure with outset in the data.
  - C and Pascal programs are typically implemented in a more top-down fashion.

Java Program Structure

```
// comment on the class
public class MyProg {
  String s = ”Viggo”;

  /**
   * The main method (comment on method)
   */
  public static void main (String[] args){
    // just write some stuff
    System.out.println ("Hello World");  }
}
```

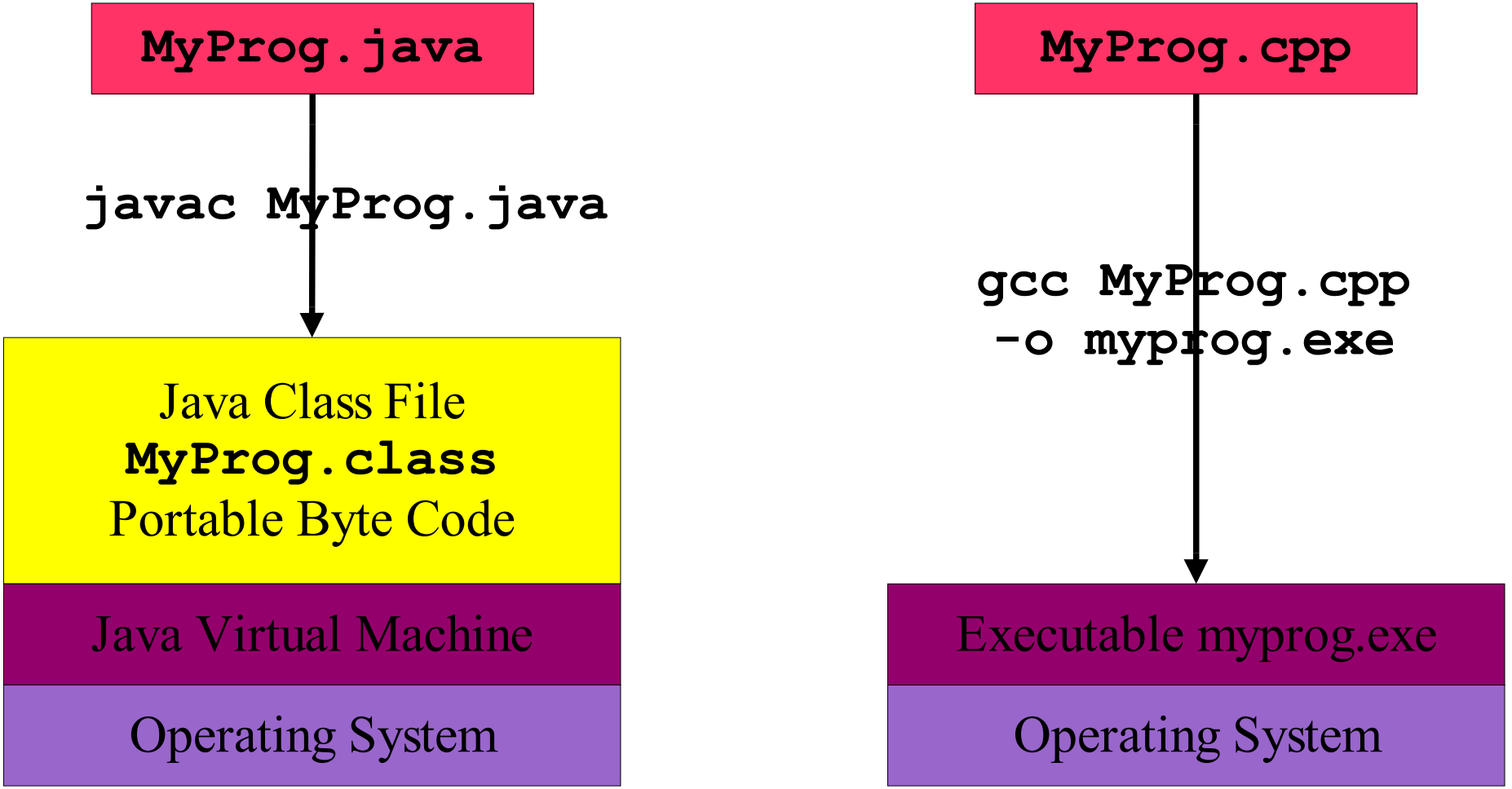




Java Class Example **Car**

```
/** A simple class modeling a car. */
public class Car {
    // instance variables
    private String make;  private String model;
    private double price;
    // String representation of the car
    public Car(String m, String mo, double p) {
        make = m; model = mo; price = p;
    }
    // String representation of the car
    public String toString() {
        return "make: " + make + " model: "
         + model + " price: " + price;
    }
}
```

Byte Code vs. Executable



History of Java

- 1990 Oak (interactive television, big failure)
- 1994 Java (for the Internet)
  - Main feature: "Write Once, Run Any Where"
    => wrap the operating system so they all look the same

Designed for

- A fresh start (no backward compatibility)
- "Pure" OOP: C++ Syntax, Smalltalk style
- Improvements over C++ much harder to write a bad program
- Internet programming
  - Very hard to create a virus
  - Run in a web browser (and at the server)
- There is a speed issue (from Java 1.3 and up much better)

Difference from C/C++

- Everything resides in a class
  - variables and methods
- Garbage collection
- Error and exception handling handling
- No global variables or methods
- No local static variables
- No separation of declaration and implementation (no header files).
- No explicit pointer operations (uses references)
- No preprocessor (but something similar)
- Has fewer "dark corners"
- Has a much larger standard library

Summary

- Classes are "recipes" for creating objects
- All objects are instances of classes
- An ADT is implemented in a class

- Aggregation and decomposition
  - “has-a” relationship
- Generalization and specialization
  - “is-a” or “is-like-a” relationship
- Encapsulation
  - Key feature of object-oriented programming
  - Separation of interface from implementation
  - It is not possible to access the private parts of an object