

# Introduction to Object-Oriented Programming

---

- Objects and classes
- Encapsulation and information hiding
- Mental exercises
  - Classification and exemplification
  - Aggregation and decomposition
  - Generalization and specialization
- Inheritance
- Polymorphism and dynamic binding
- Java
  - An object-oriented programming language
  - Program example
  - History of Java
  - Comparison to C/C+

# Objects and Classes

---



Mammal  
Two-legs  
Very large brains  
Omnivorous (plants + meat)



Mammal  
Tusks  
Four legs  
Herbivorous (plant eater)

# The Object Concept

---

- An object is an *encapsulation* of data.
- An object has
  - identity (a unique reference)
    - ◆ Memory address
    - ◆ Social security number (cpr), employee number, passport number
  - state, also called characteristics (variables or attributes)
    - ◆ hungry, sad, drunk, running, alive
  - behavior (functions also called methods)
    - ◆ eat, drink, wave, smile, kiss
- An object is an instance of a *class*.

# The Class Concept

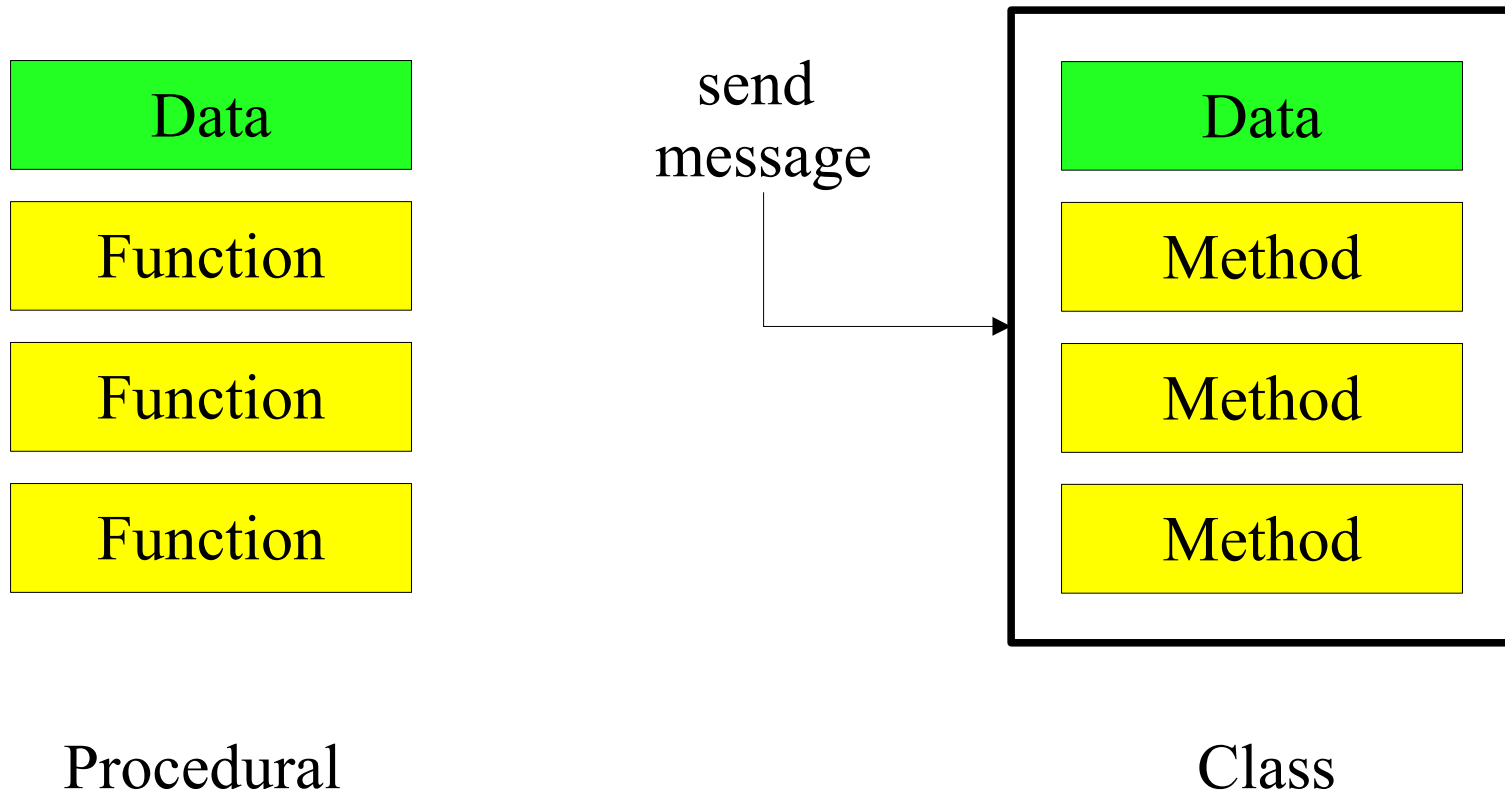
---

- A class is a collection of *objects* (or *values*) and a corresponding set of *methods*.
- A class encapsulates the data representation and makes data access possible at a higher level of abstraction.
- A class is often called an *Abstract Data Type (ADT)*.
  
- Example 1: A vehicle with methods for starting, stopping, driving, get km/liter, etc.
- Example 2: A time interval with methods for start time, end time, duration, overlapping intervals, etc.
- Example 3: A string, upper case, compare, lower case, etc.
  - `str.equals(otherStr)` – class/Java style
  - `strcmp(str, otherStr)` – C style

# Encapsulation and Information Hiding

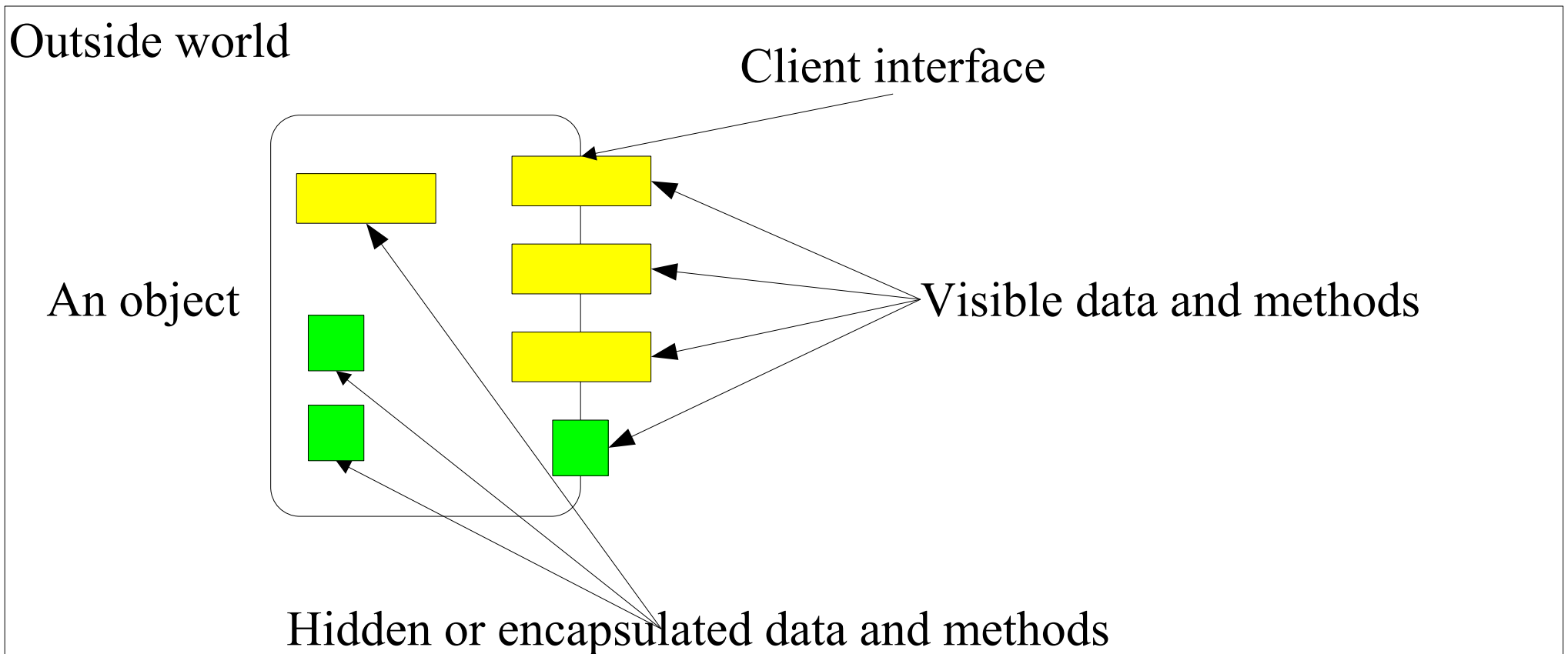
---

- 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 effecting 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.

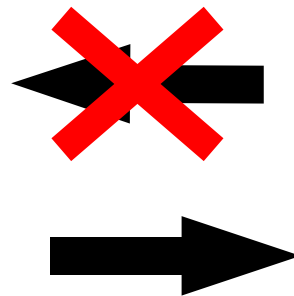
# Connection between Object and Class

---

- In object-oriented programming language we write classes
  - The text files we create contain classes!
  - Static (and “One”)
- Objects are created *from* classes
  - A class contains a “recipe” on how to make objects
  - Dynamic (and “Many”)

## Class

250 g digestive biscuits  
125 g soft brown sugar  
125 g butter  
50 g raisins  
3 tablespoons cocoa  
1 egg  
few drops vanilla essence



## Objects



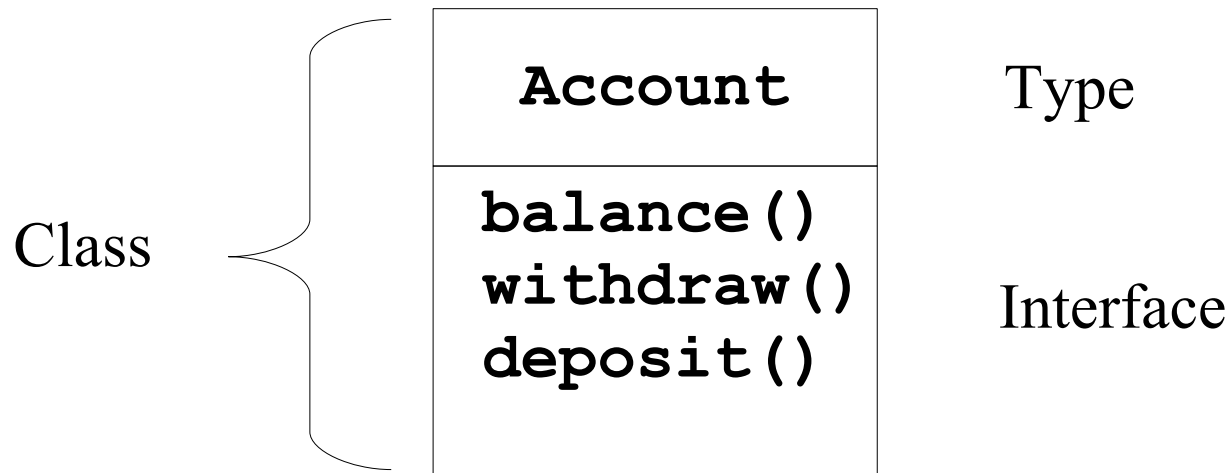
source <http://www.filflora.com>



# Type and Interface

---

- An object has a type and an interface.



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

# 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

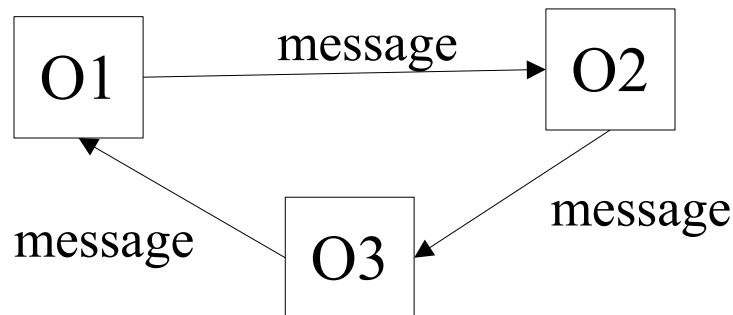
## *Dynamic Instantiating*

- 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 visible interface).
  - “O1 sends O2 a message”
- O1 and O2 must be *related* to communicate.
- The call of a method corresponds to a function (or procedure) call in a non-object-oriented languages such as C or Pascal.



# Phenomenon and Concept

---

- A *phenomenon* is a thing in the “real” world that has individual existence.
  - an object
- A *concept* is a generalization, derived from a set of phenomena. It is based on the common properties of these phenomena.
  - a class
- 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, Examples

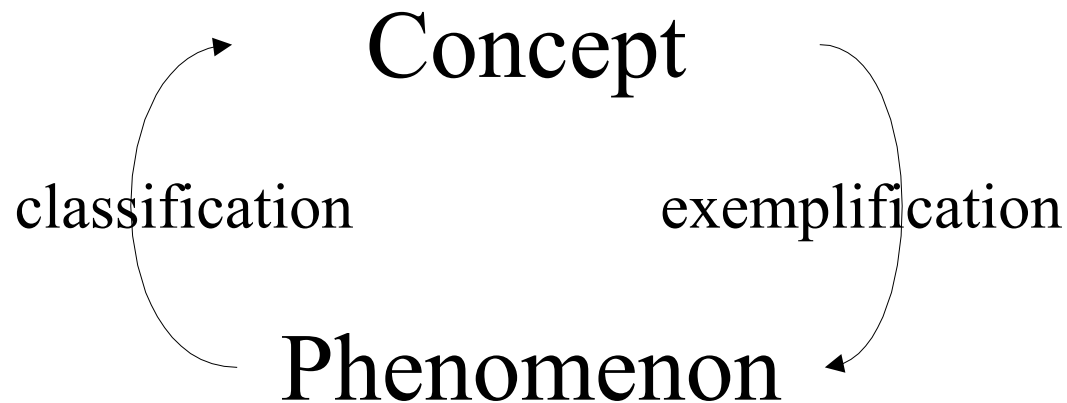
---

- hat, 23, 34, mouse, telephone, book, 98, 45.34, hello
  - numbers: 23, 34, 98, 45.34
  - words: hat, mouse, telephone, book, hello
  
- mouse, tyrannosaurus rex, allosaurus, elephant, velociraptor
  - dinosaur: tyrannosaurus rex, allosaurus, velociraptor
  - mammal: mouse, elephant

# Classification and Exemplification, cont.

---

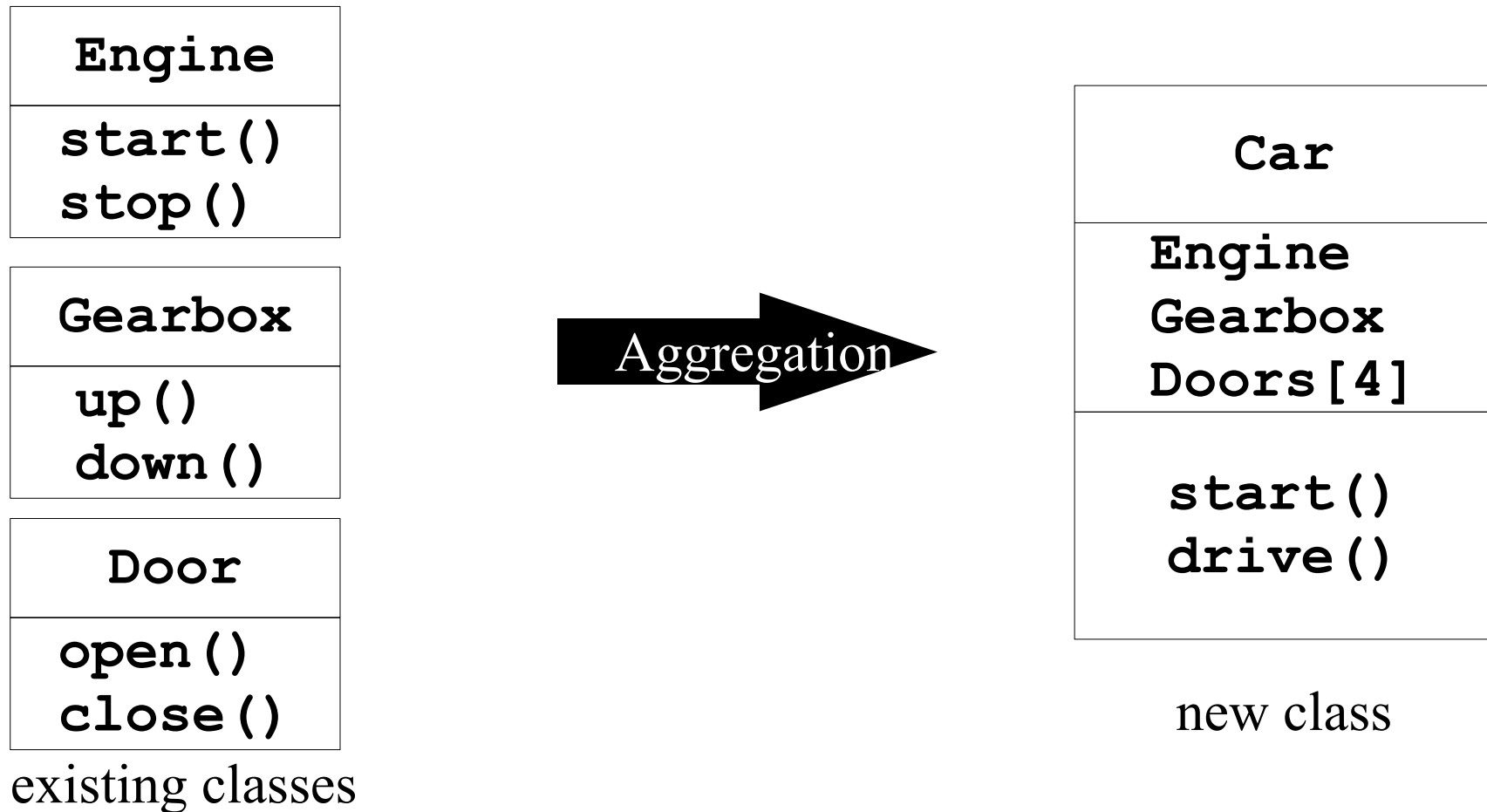
- A *classification* is a description of which phenomena that belongs to a concept.
- An *exemplification* is a phenomenon that covers the concept.



# Aggregation and Decomposition, Example

---

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

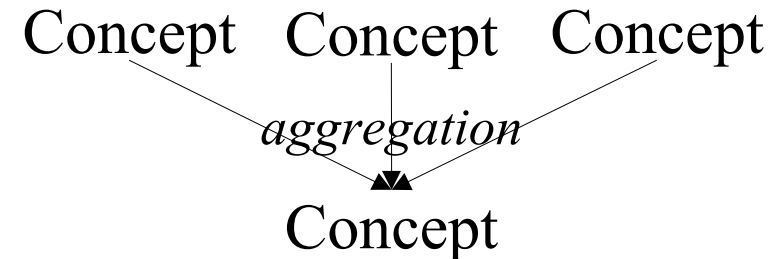
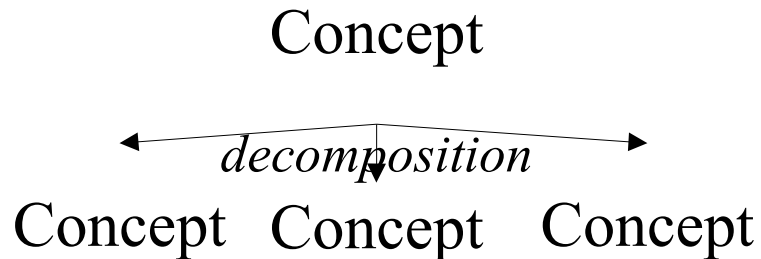


- **Car** “has-a” **Gearbox** and **Car** “has-an” **Engine**

# Aggregation and Decomposition, cont.

---

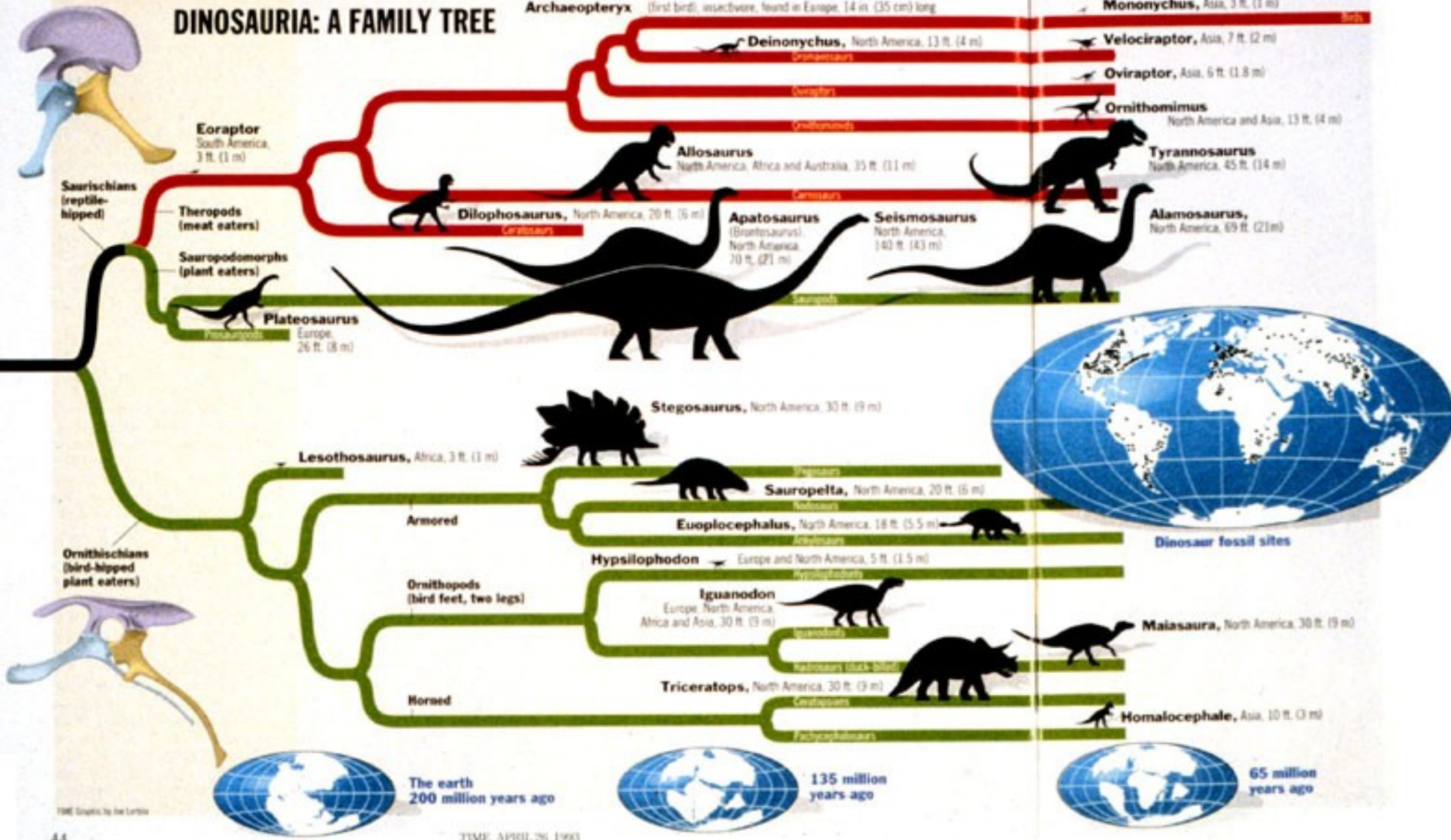
- An *aggregation* consists of a number of (sub-)concepts that collectively is considered a new concept.
- A *decomposition* splits a single concept into a number of (sub-)concepts.





# Generalization and Specialization, Example

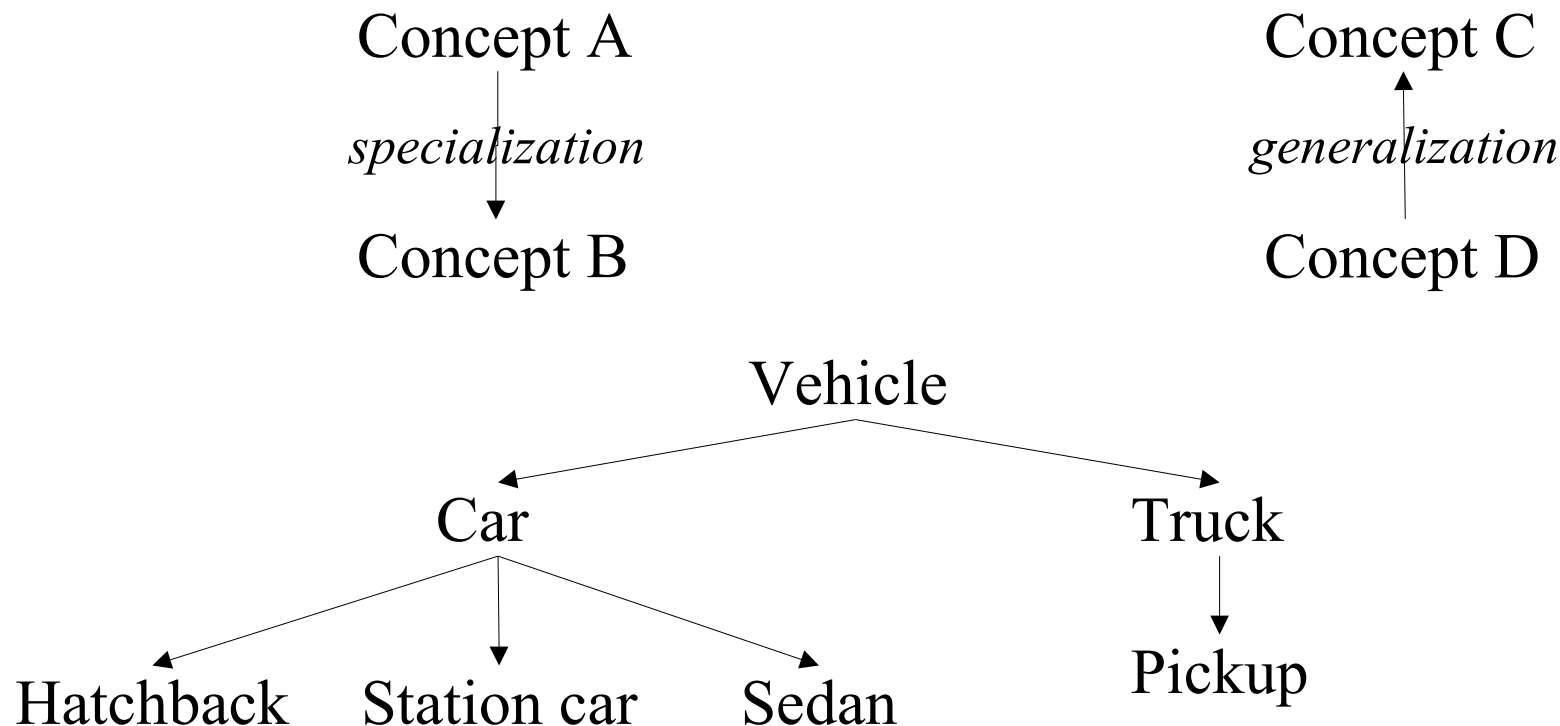
## DINOSAURIA: A FAMILY TREE



# Generalization and Specialization, cont.

---

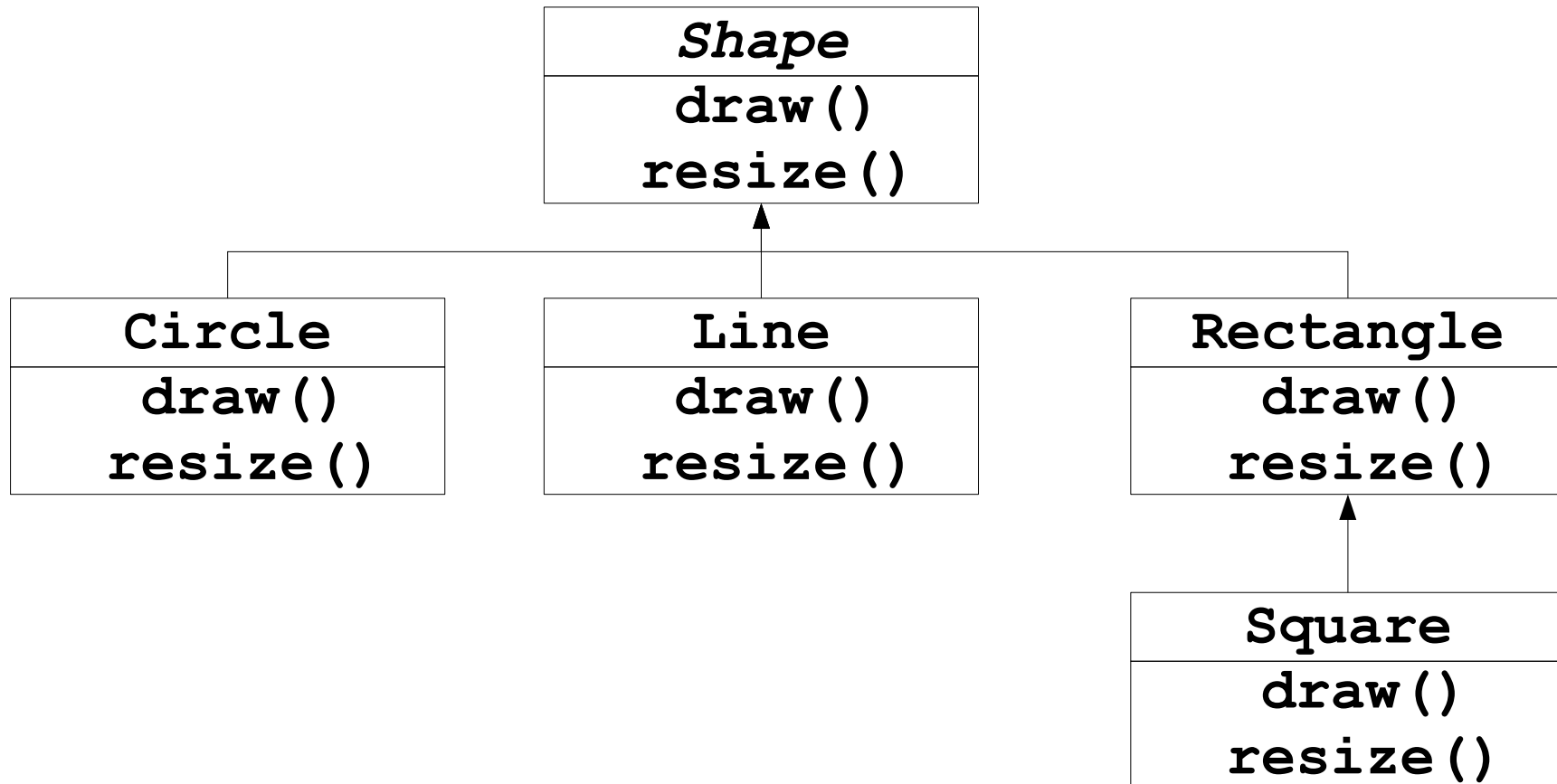
- *Generalization* creates a concept with a broader scope.
  - Larger extension, smaller intension
- *Specialization* creates a concept with a narrower scope.
  - Smaller extension, larger intension
- *Reusing the interface!*



# Generalization and Specialization, Example

---

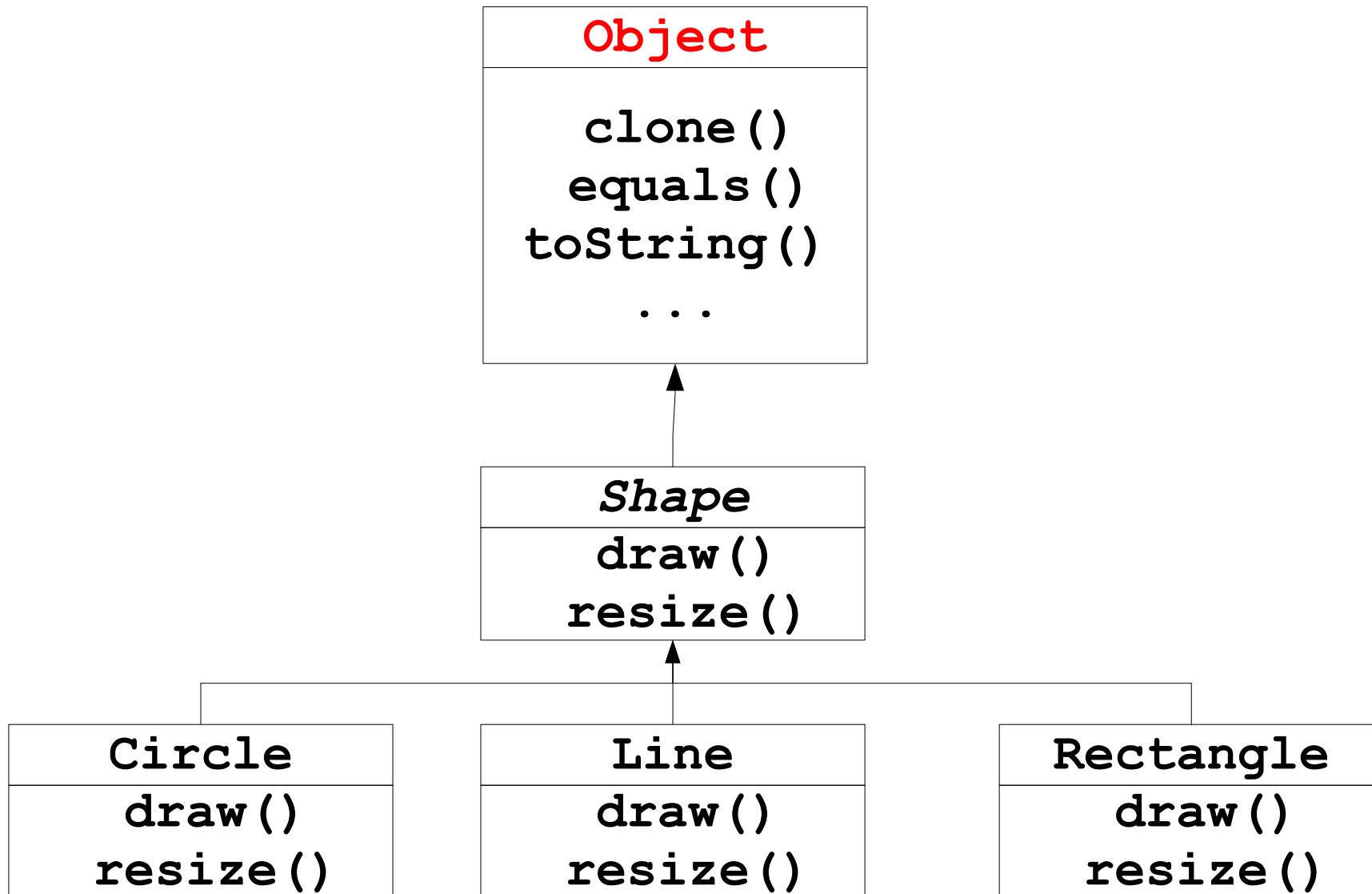
- *Inheritance*: get the interface from the general class.
- Objects related by inheritance are all of the same type.



- **Square “is-a” Shape** or **Square “is-like-a” Shape**

# Generalization and Specialization in Java

---



# Polymorphism and Dynamic Binding

---

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

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

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

# Benefit Generalization and Specialization

---

- Take previous **Shape** class hierarchy
  - Remove inheritance
  - Remove general class **Shape**

<b>Rectangle</b>
<b>draw()</b> <b>resize()</b>

<b>Square</b>
<b>draw()</b> <b>resize()</b>

<b>Circle</b>
<b>draw()</b> <b>resize()</b>

<b>Line</b>
<b>draw()</b> <b>resize()</b>

# Code Example, Revisited

---

```
void doSomething(Circle c) {  
    c.draw();  
    c.resize();  
}
```

```
void doSomething(Line l) {  
    l.draw();  
    l.resize();  
}
```

```
Circle c = new Circle();  
Line l = new Line();  
Rectangle r = new Rectangle();
```

```
doSomething(c);  
doSomething(l);  
doSomething(r);
```

```
void doSomething(Rectangle r) {  
    r.draw();  
    r.resize();  
}
```

```
void doSomething(Square s) {  
    s.draw();  
    s.resize();  
}
```

Similar code  
is repeated

# Java Class Example Car

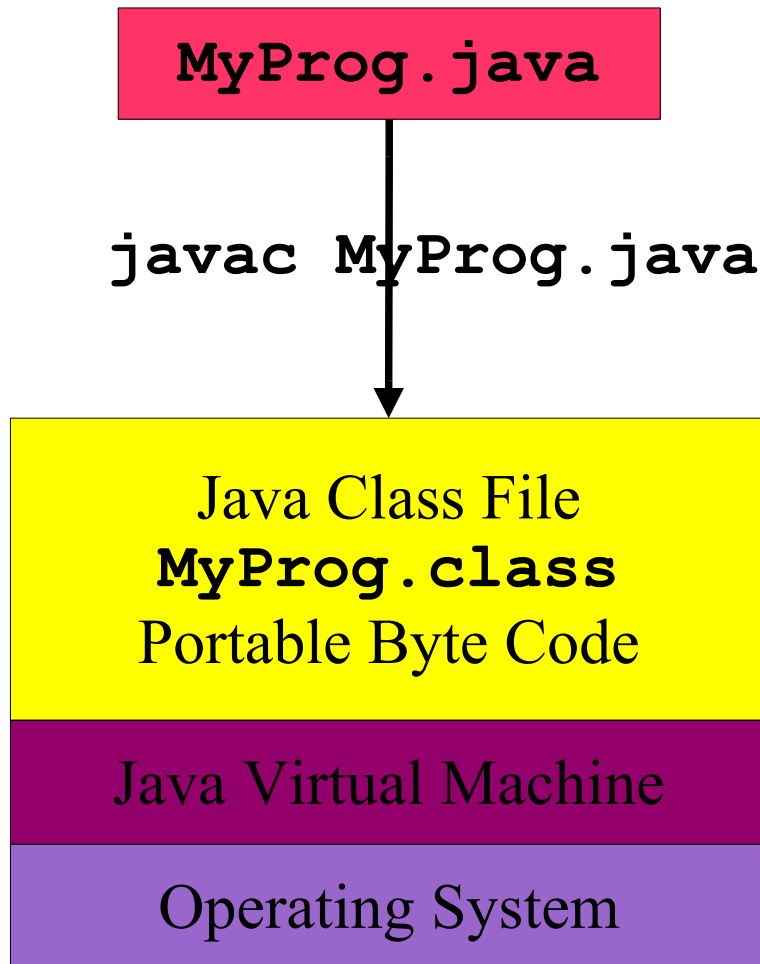
---

```
/** A simple class modeling a car. */
public class Car {
    // instance variables
    private String make;
    private String model;
    private double price;
    // constructor
    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;
    }
}
```

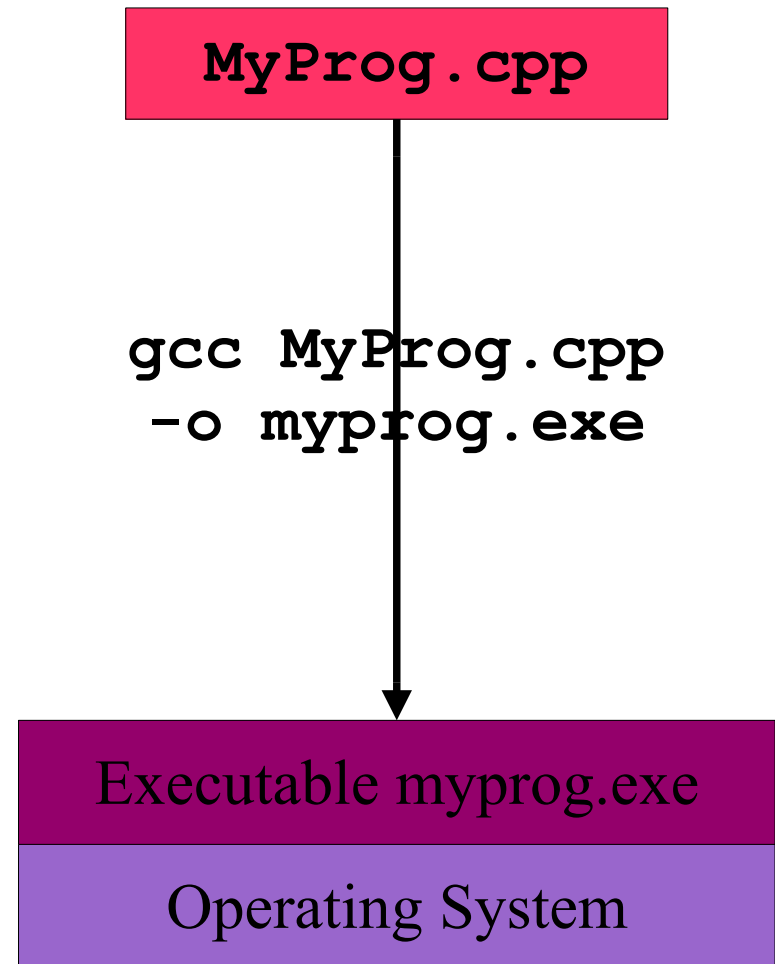


# Byte Code vs. Executable

---



Java/C# world



C++ world

# 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)
- C# Microsoft's “Java-Killer” project release 2001
  - Language very similar to Java
  - Common-Language Runtime (CLR) supports 60+ languages

# Difference from C/C++

---

- Everything resides in a class
  - variables and methods
- Garbage collection
  - Bye bye **malloc()**, **free()**, and **sizeof()**
- Error and exception handling handling
- No global variables or methods
- No local static variables
- No separation of declaration and implementation
  - Bye bye header files
- No explicit pointer operations (uses references)
- Has fewer “dark corners”
- Has a much larger standard library
  - Java Developer Kit or JDK

# Summary

---

- A class is a “recipe” for creating objects
- All objects are instances of classes
- Encapsulation
  - Key feature of object-oriented programming
  - Separation of interface from implementation
  - It is not possible to access the hidden/encapsulated parts of an object
- Aggregation and decomposition
  - “has-a” relationship
- Generalization and specialization (inheritance)
  - “is-a” or “is-like-a” relationship
- Polymorphism/dynamic binding
  - Softening static typing

# Java Program Structure

---

```
// comment on the class
```

```
public class MyProg {
```

```
    String s = "Viggo";
```

variable



```
/**
```

```
 * The main method (comment on method)
```

```
*/
```

```
public static void main (String[] args) {
```

```
    // just write some stuff
```

```
    System.out.println ("Hello World"); }
```

method header



method body



```
}
```

# Common Mistakes and Errors

---

// what is ugly here?

```
public class main {  
    public static void main(String[] args) {  
        System.out.println("Hello World");  
    }  
}
```

// what is wrong here?

```
public class MyClass {  
    public void static main(string[] args) {  
        system.out.println("Hello World");  
    }  
}
```

// what is ugly here?

```
public class MyClass {  
    public static void main(String[] args) {  
        System.out.println("Hello World");  
    }  
};
```

# 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.

# 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.