Introduction to Standard C++

Intermezzo: Object Oriented Analysis and Design

Massimiliano Culpo[1]

[1]CINECA - SuperComputing Applications and Innovation Department

07.04.2014

Table of contents

Object oriented programming : what's in that?

Imperative programming : reach your goal changing a program state

Procedural programming : steps to reach the problem solution

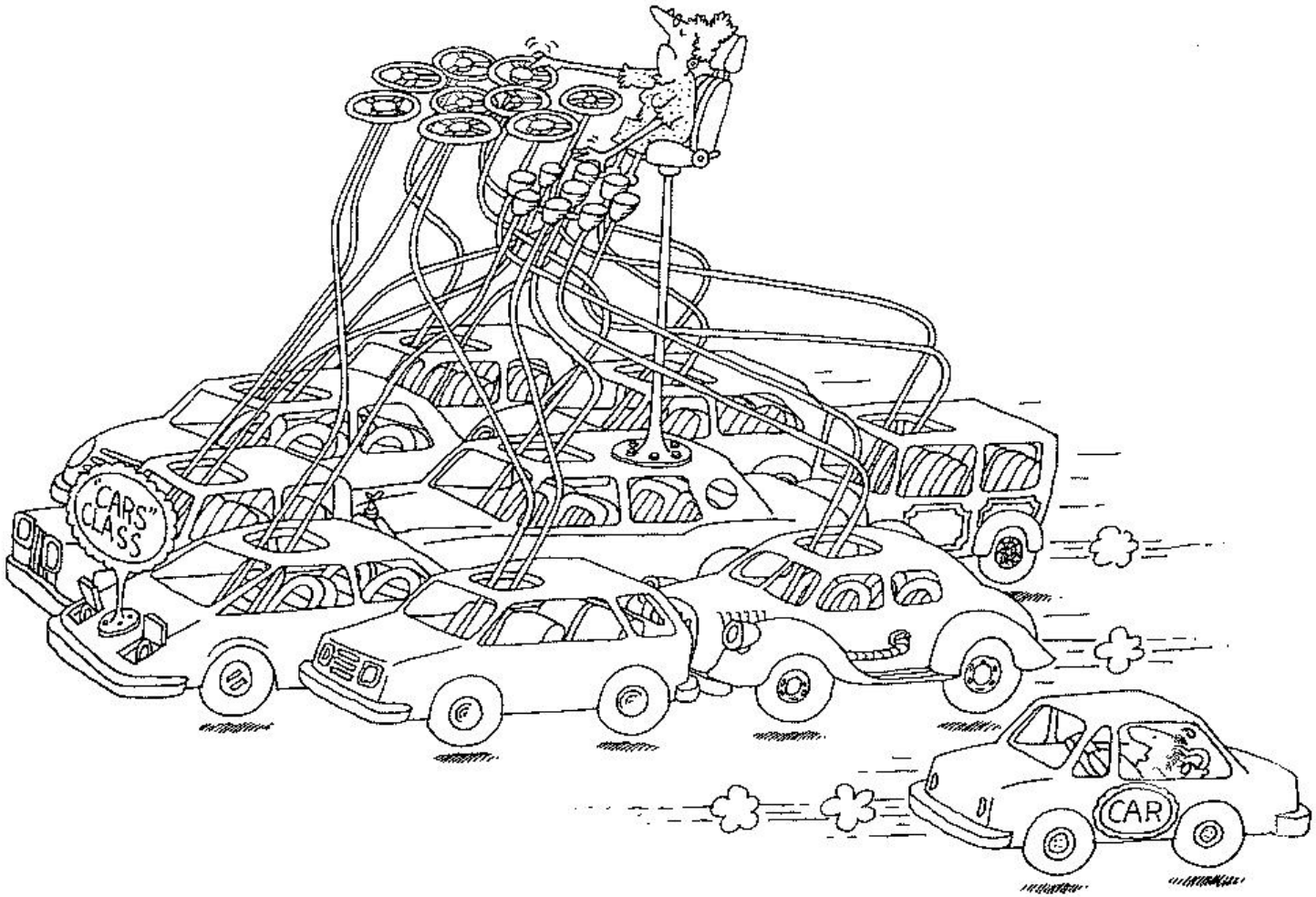
1. Separation between data structure and algorithms
2. A program is usually broken down in a series of functions
3. Code cluttered with low-level details of algorithms

OO programming : problem in terms of interactions among entities

1. Aggregation between data and functions operating on it
2. A program is broken down in a series of interacting classes
3. Algorithms may be expressed into high-level domain language
4. Fosters flexibility and permits to manage complexity

Table of contents

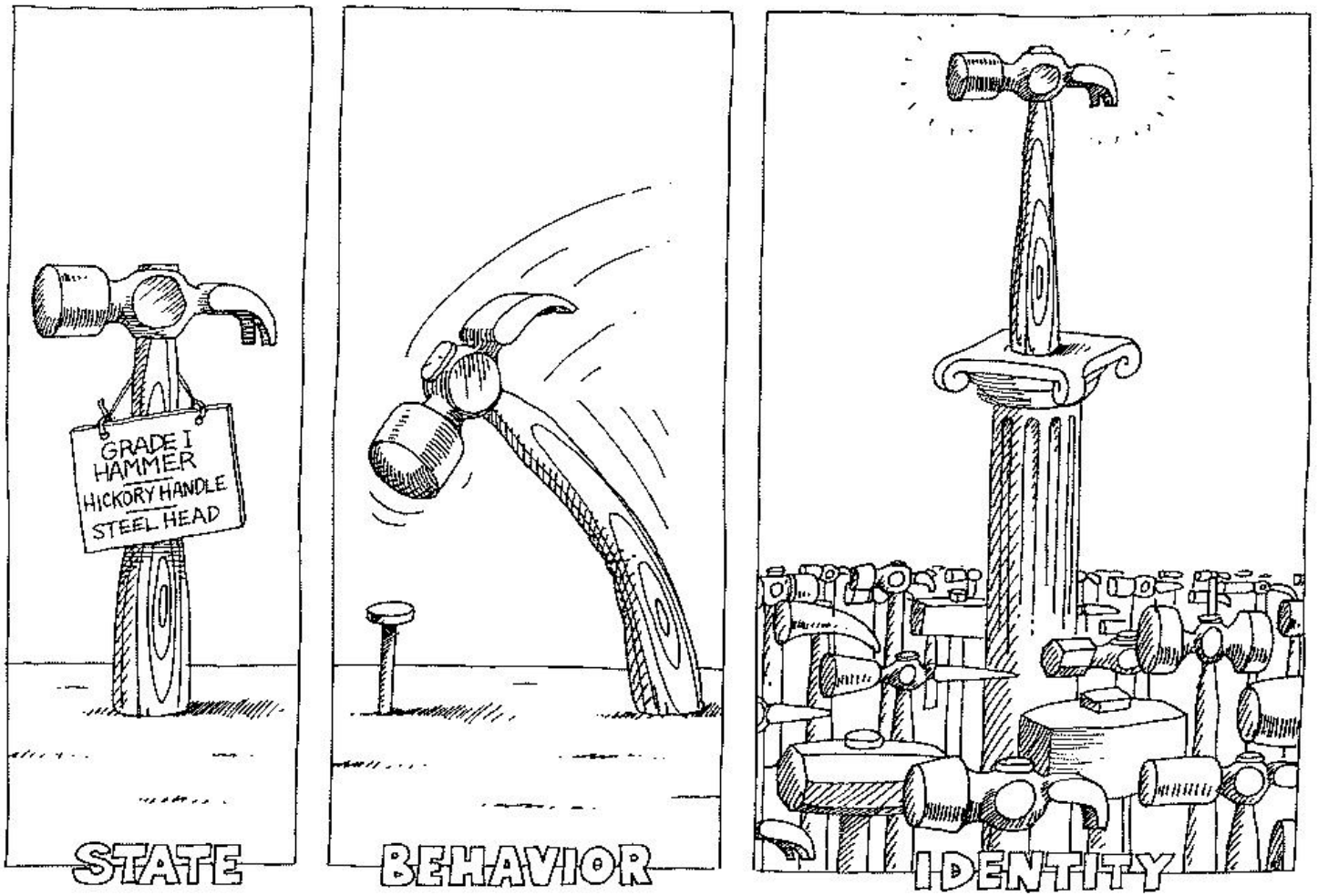
OO Concept - **Class**



OO Concept - **Class**

```
/**
 * A class defines a set of objects that
 * share behavior and structure
 */
class Car {
public:
   Car & change_speed(Speed new_speed);
   Car & turn(Angle degree);
   /* Behavior */
private:
   Engine m_engine;
   Tires m_tires;
   /* Structure */
};
```

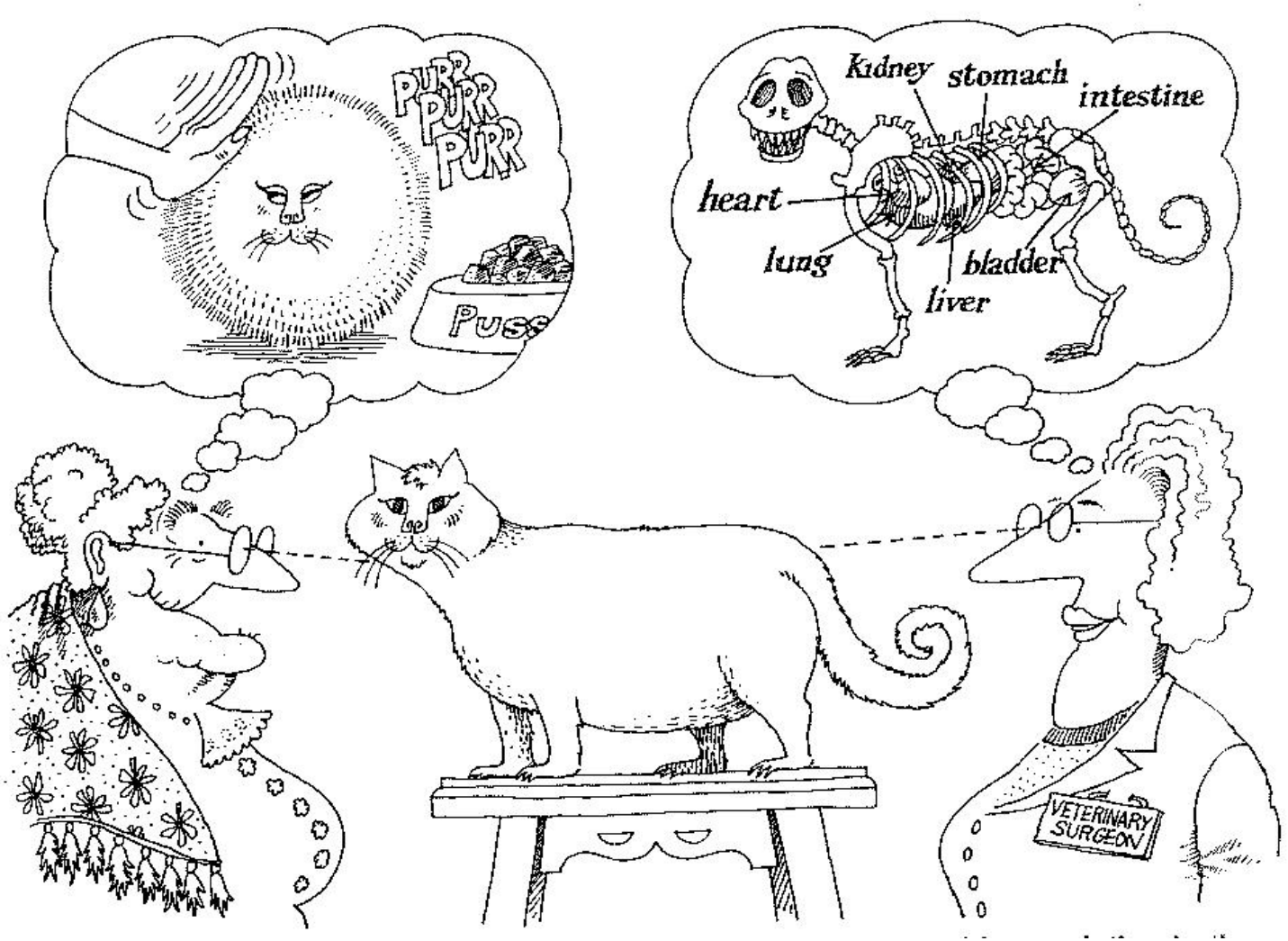
OO Concept - **Object**



OO Concept - **Object**

```
/**
 * An object is an instance of a class
 */
class Hammer {
public:
   Hammer(Handle,Head);
   void hit(Needle needle);
private:
   Handle m_handle;
   Head m_head;
};
/* Each object has a unique identity */
Hammer hammer1(wooden_handle,steel_head);
Hammer hammer2(rubber_handle,titanium_head);
```

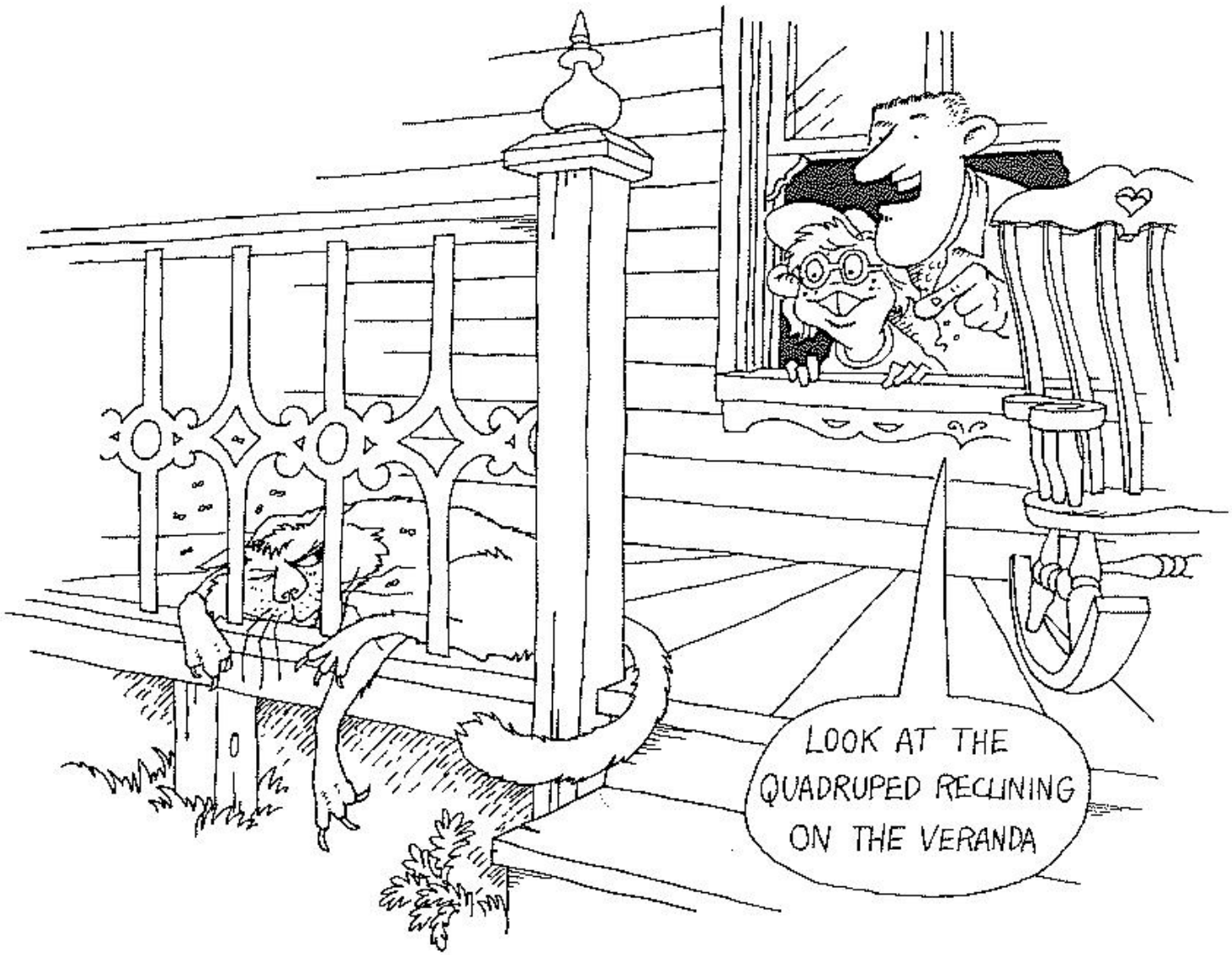
OO Concept - **Abstraction**



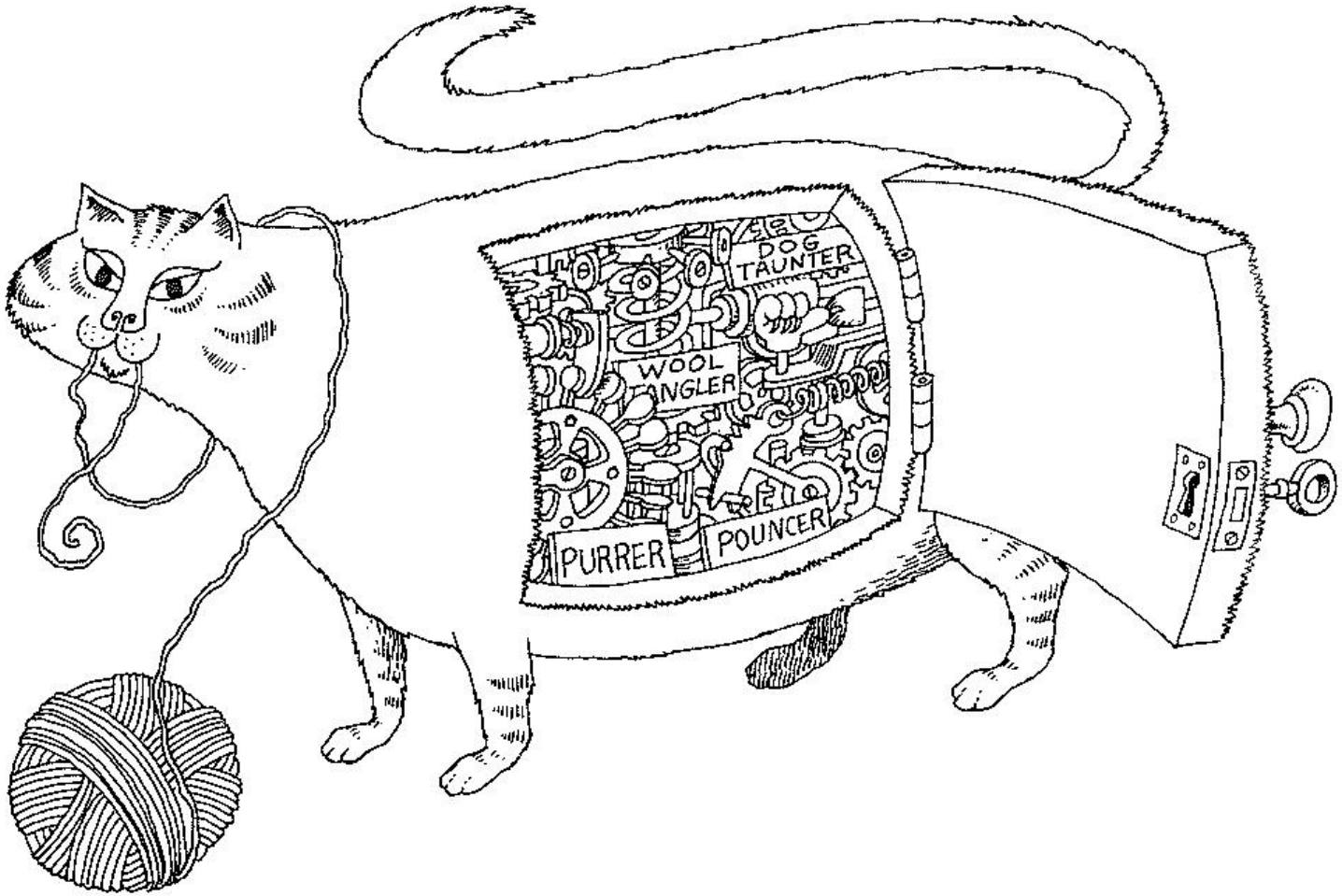
OO Concept - **Abstraction**

```
/**
 * Classes are abstraction of real entities
 * as they reduce the information to what is
 * important for the problem at hand
 */
struct Cat { /* For the granny */
  void purr();
  void eat();
};
struct Cat { /* For the veterinary surgeon */
  void breath();
  void heart_rate();
  /* ... */
}
```

OO Concept - **Abstraction**



OO Concept - **Encapsulation**



OO Concept - **Encapsulation**

```
/**
 * Encapsulate means hiding the internal
 * state from the client
 */
}
class Cat {
public:/* What the client knows */
  void purr();
  void eat();
private:/* What the client shouldn't know */
  void tongue_out();
  Stomach m_stomach;
  Tongue m_tongue;
};
```

OO Concept - **Modularity**

OO Concept - **Modularity**

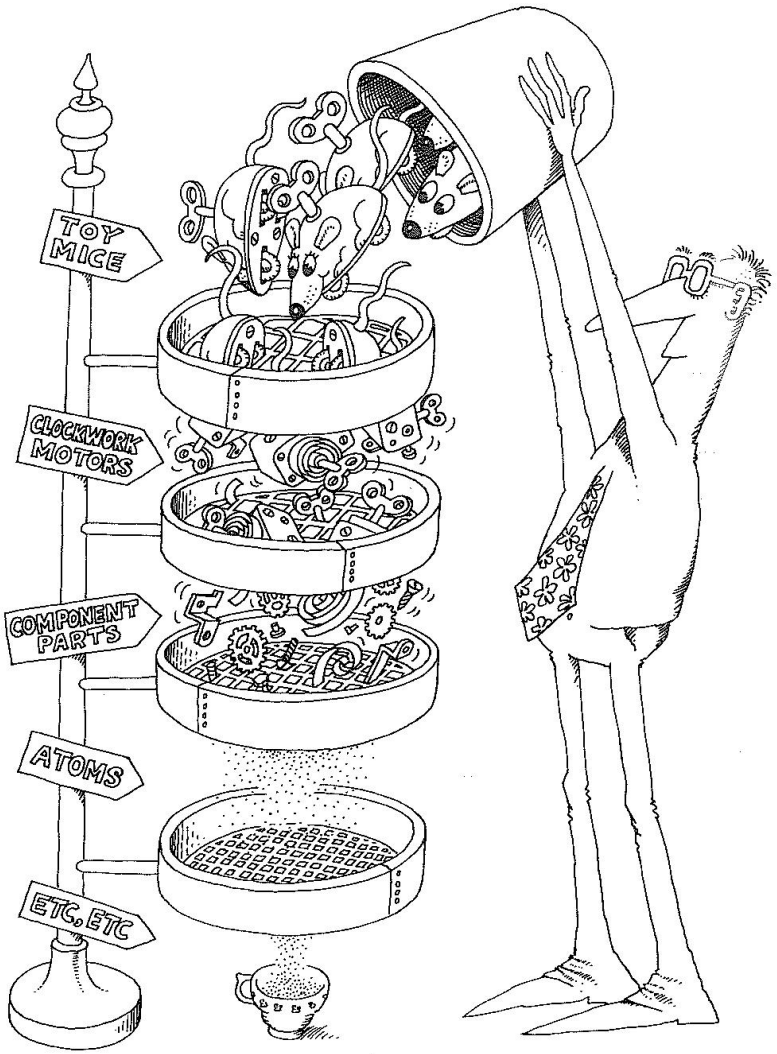
```
/**
 * A system is decomposed into parts
 * (modules) that are weakly coupled
 */
struct Purrer {
  /* Public interface */
  void execute();
  /* Internal state */
}
struct Cat {
  void purr() { m_purrer.execute(); }
private:
  Purrer m_purrer;
};
```

OO Concept - **Inheritance**

OO Concept - **Inheritance**

```
/**
 * A class may extend the behavior of
 * another class inheriting from it
 */
struct Car { /* Behavior */
  virtual Car & change_speed(Speed speed);
  virtual Car & turn(Angle degree);
  /* Other methods here */
};
struct RaceCar : public Car {
  /* Change only what needs change */
  Car & change_speed(Speed speed);
  Car & turn(Angle degree);
}
```

OO Concept - **Hierarchy**



OO Concept - **Hierarchy**

```
/**
 * Abstractions are ordered into hierarchies
 * according to the relationship among them
 */
class ClockworkMotor : public ToyMotor {
  /* State and behavior defined here */
}

class ToyMouse : public MechanicalToy {
  void charge();
private:
  ClockworkMotor m_motor;
}
```

OO Concept - **Type check**

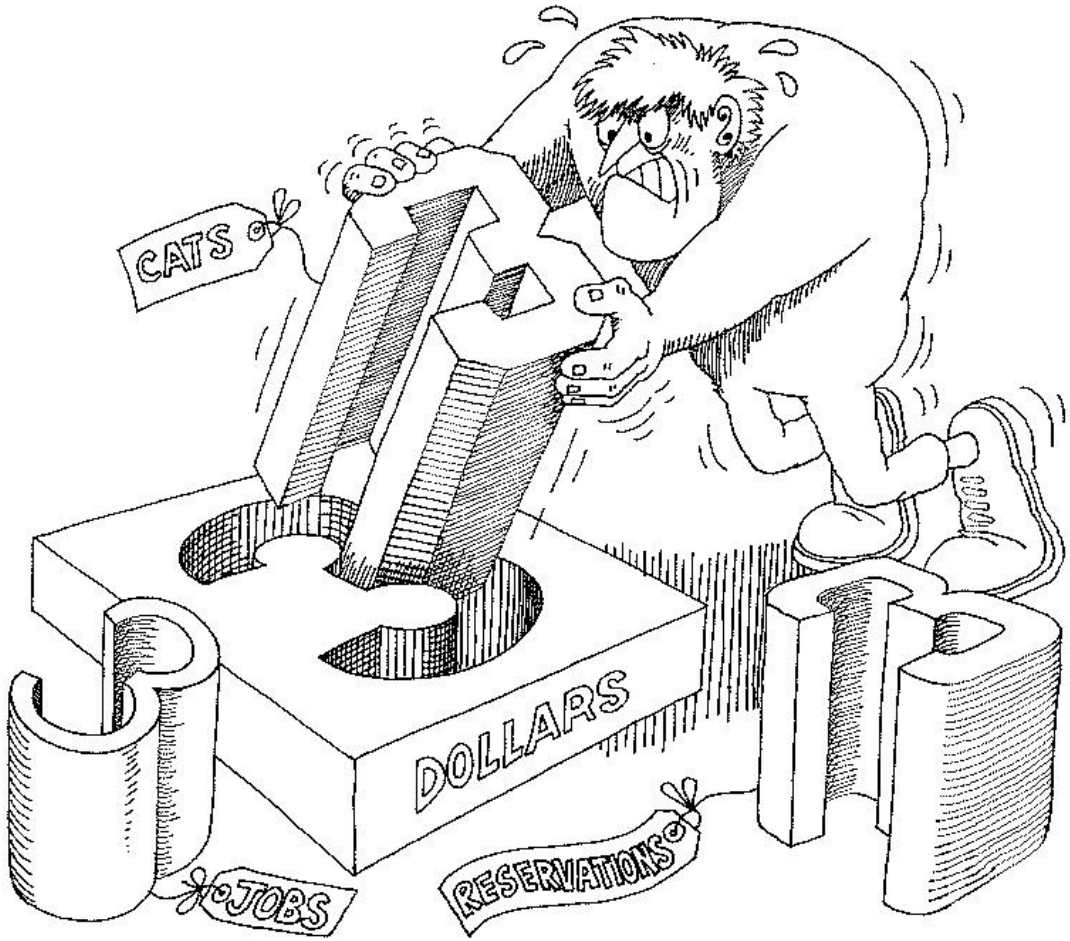


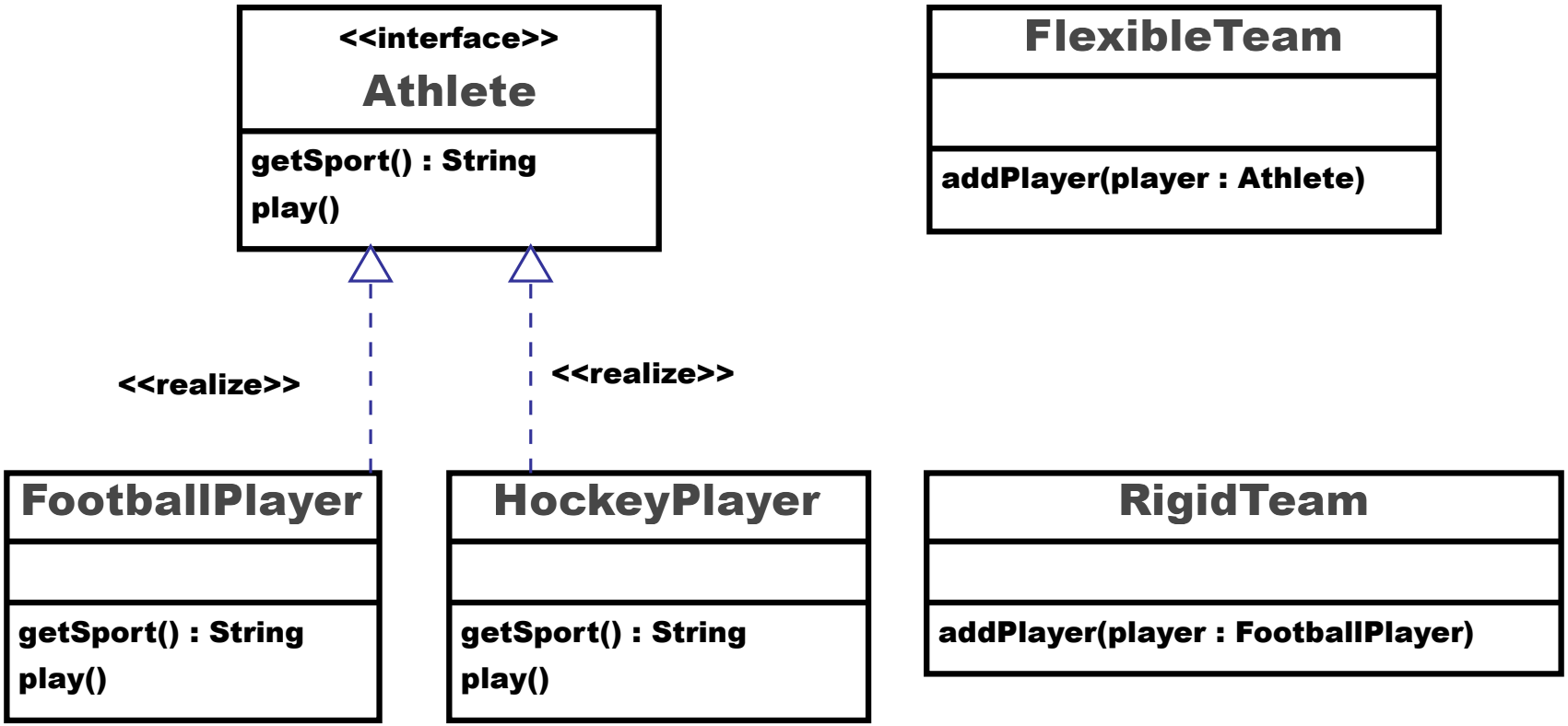
Table of contents

What is a design principle?

Design principle

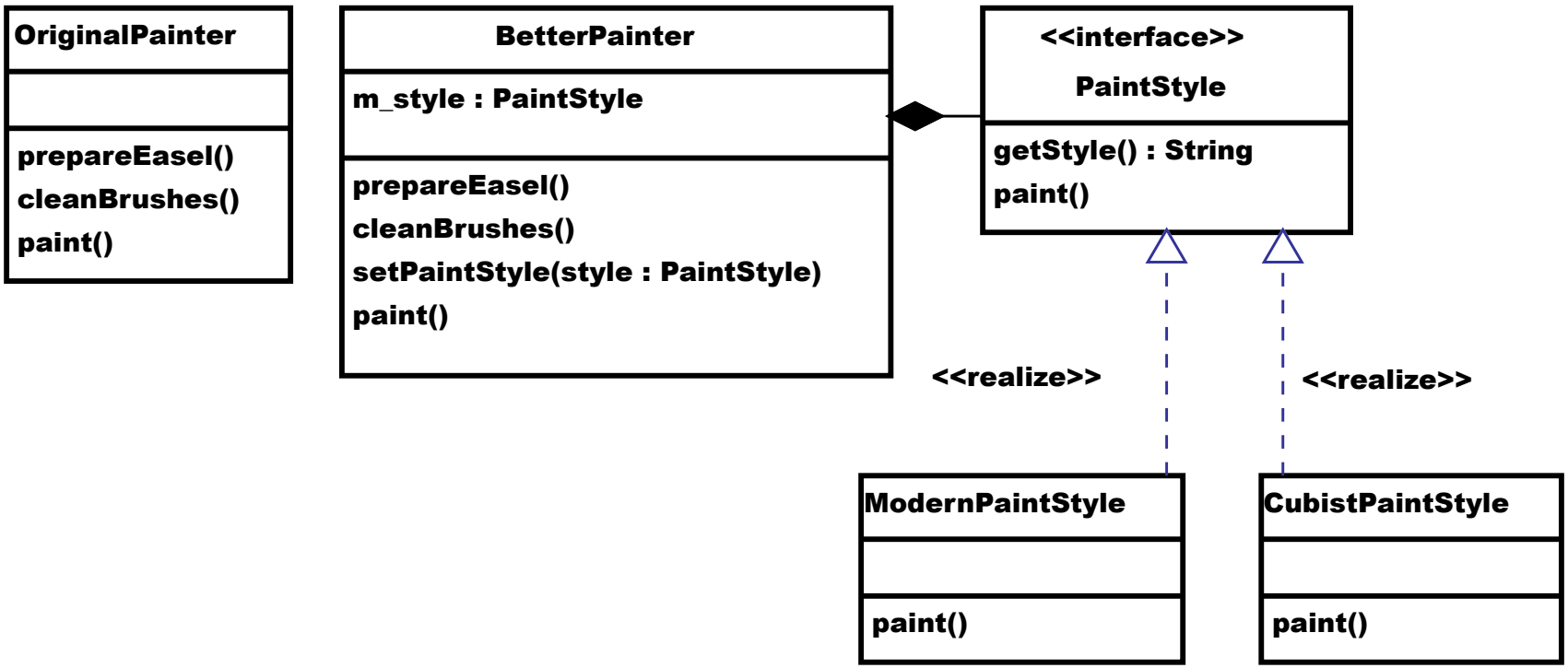
Basic tool or technique that can be applied to designing or writing code to make that code **more maintainable, flexible or extensible**, *"Head First, Object Oriented Analysis and Design"*

OOD principle - **Code to an interface**



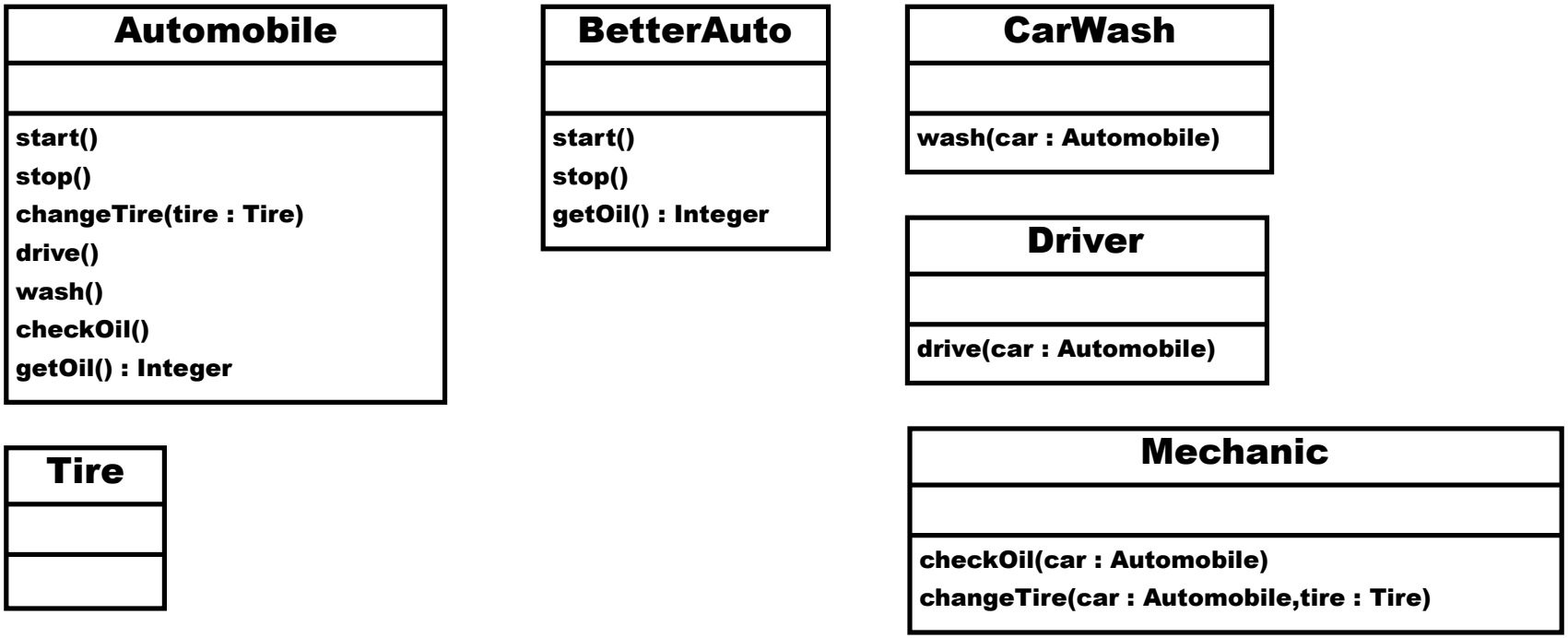
By coding to an interface your code will work with **all** *of the interface subclasses - even the ones that haven't been created yet*

OOD principle - **Encapsulate what varies**



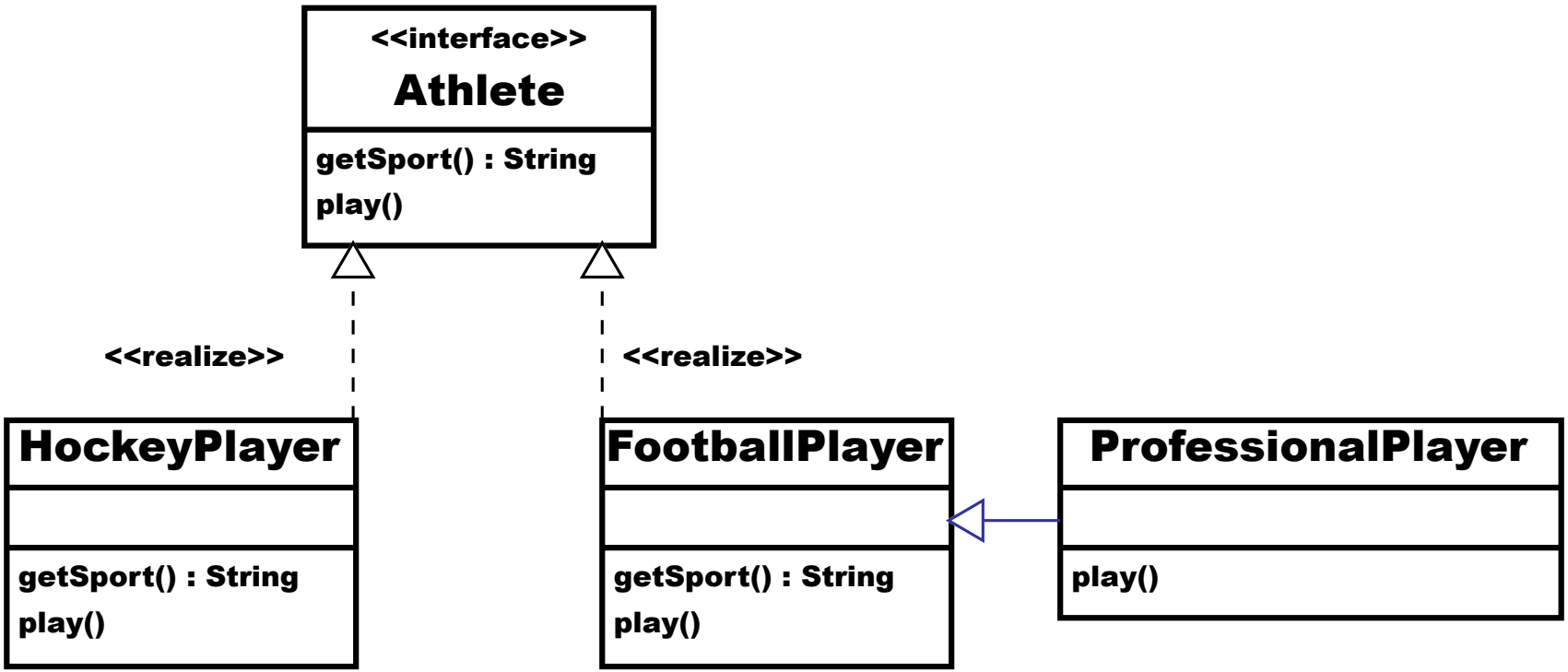
Anytime you have behavior that is likely to change, move that behavior **away from what won't change** *very frequently. This way you'll get stable and extensible code*

OOD principle - **Have only one reason to change**



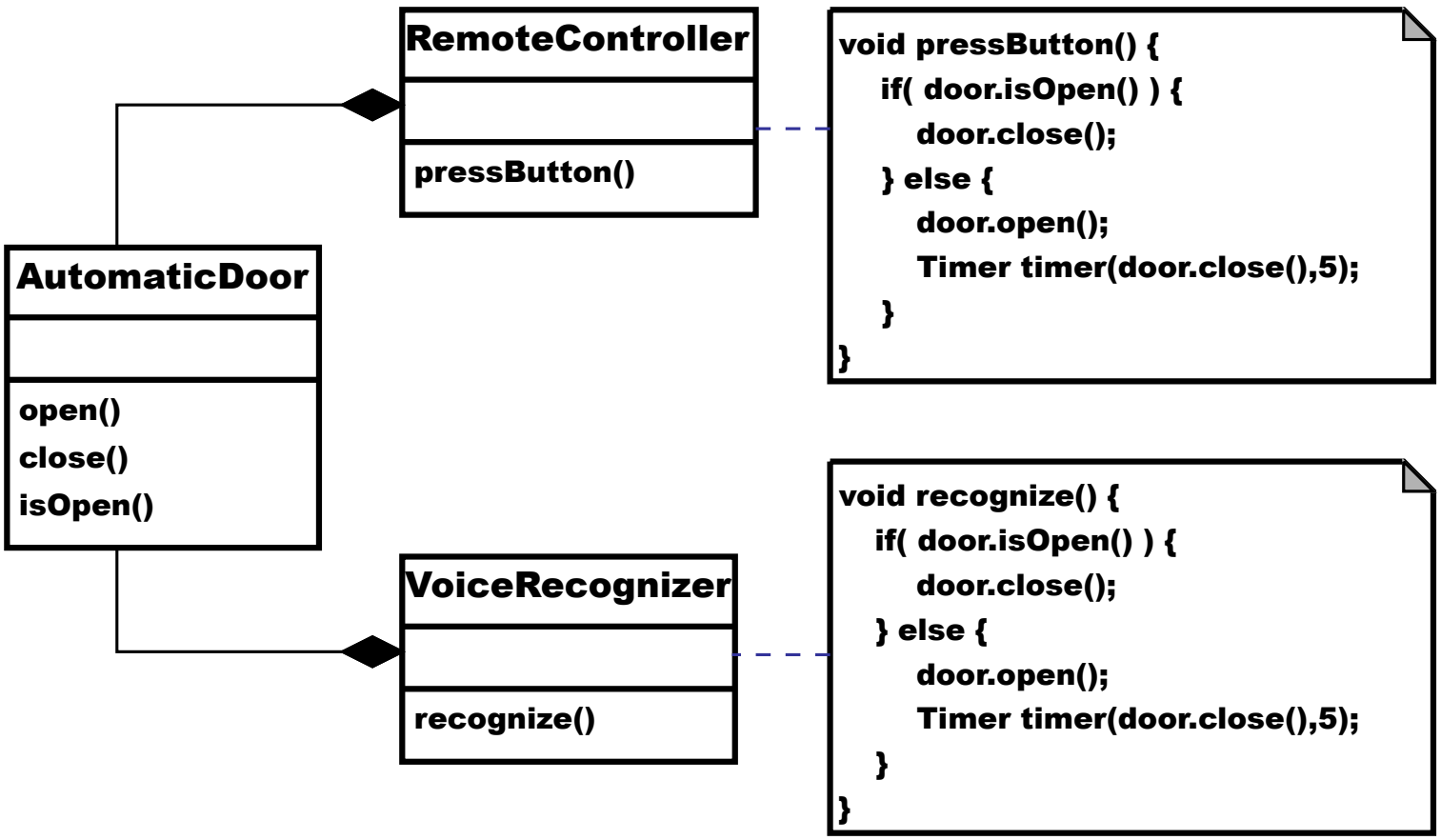
The easiest way to make your software resilient to change is to make sure that each class **has only one reason to change**. *The chances that a class is going to change are minimized* **reducing the number of things that can change**

OOD principle - **The Open-Closed Principle (OCP)**



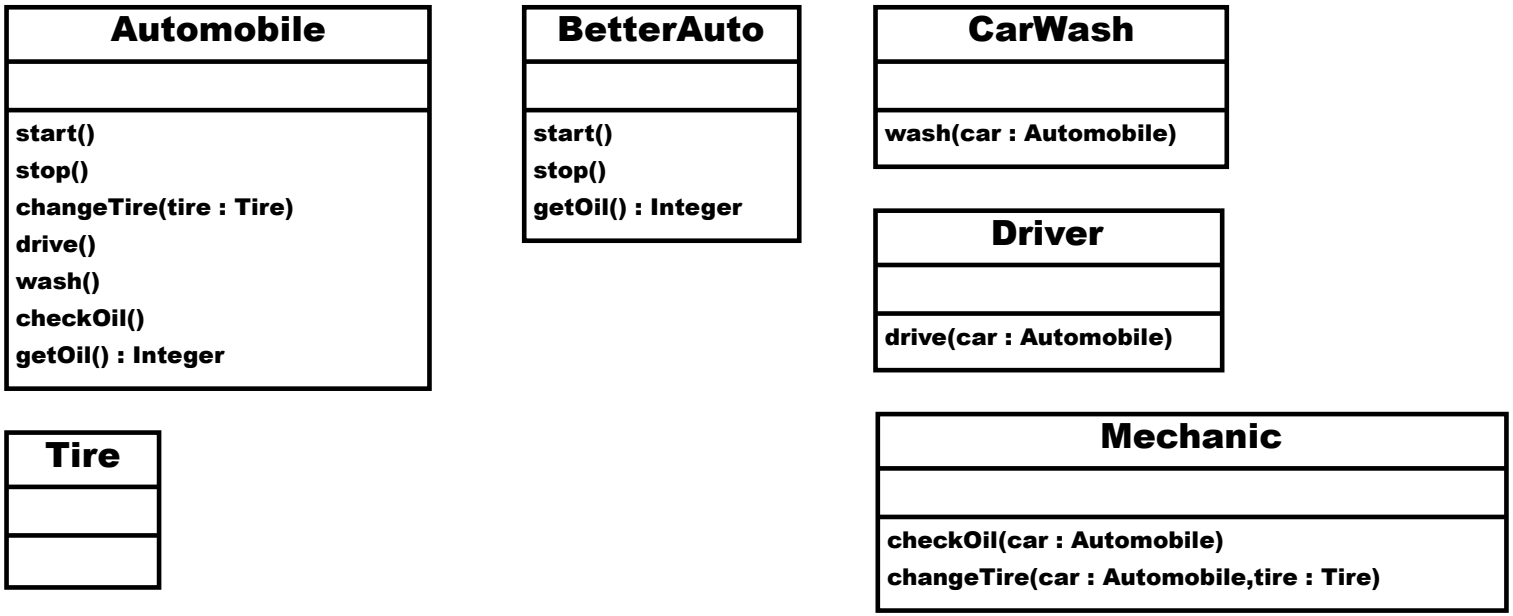
A flexible code is one that allows changes but does not require modifications to existing code. Code classes that are **open for extension and closed for modifications**

OOD principle - **Don't Repeat Yourself (DRY)**



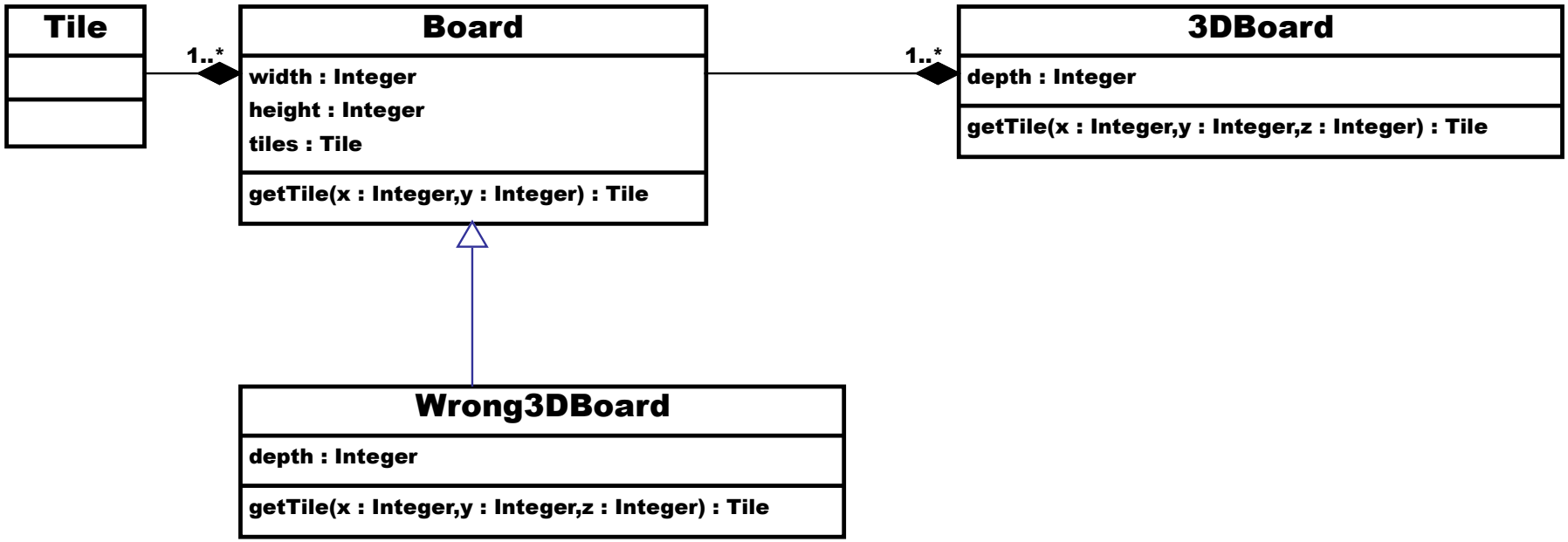
Avoid duplicate code by abstracting out things that are common and **placing those things in a single location**

OOD principle - **Single Responsibility Principle (SRP)**



Every object in your system should have a **single responsibility**. *All its services should be* **focused on carrying out that responsibility**.

OOD principle - **Liskov Substitution Principle (LSP)**



When inheriting from a base class, you must be able to substitute your sub-class for the base class without altering the semantic.
In short: **subtypes must be substitutable for their base types**

OOD principle - **Liskov Substitution Principle (LSP)**

Delegation

Delegate behavior to another class when you don't want to change the behavior, but the **implementation it's not your responsibility**

Composition

Composition permits to reuse behavior from one or more classes. Your object **completely owns the composed objects**, *and they do not exist outside of their usage in your object*

Aggregation

Aggregation is the same thing as composition except that **aggregated objects exist outside of your object**

Table of contents - Appendices


Alexandrescu, A.
Modern C++ Design: Generic Programming and Design Patterns Applied.
Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2001.


Gamma, E., Helm, R., Johnson, R., and Vlissides, J.
Design Patterns: Elements of Reusable Object-oriented Software.
Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1995.


McLaughlin, B. D., Pollice, G., and West, D.
Head First Object-Oriented Analysis and Design: A Brain Friendly Guide to OOA&D (Head First).
O'Reilly Media, Inc., 2006.