Object-Oriented Programming

Huseyin Ergin

THE UNIVERSITY OF ALABAMA®
College of Engineering
Computer Science

PROGRAMMING?

- **Just writing some text in a specific format**

```java
/* HelloWorld.java
 */

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

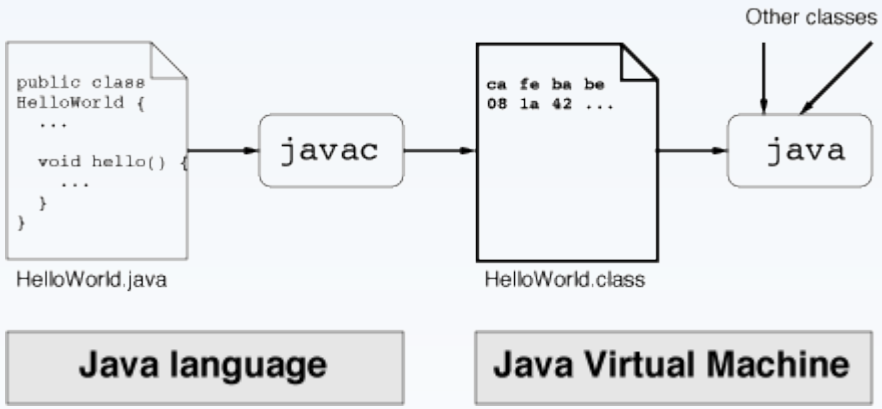
```cpp
// my first program in C++
#include <iostream>

int main()
{
  std::cout << "Hello World!";
}
```

```python
# HelloWorld.py
print "Hello World"
```

PROGRAMMING LANGUAGES

SO, WHAT IS TRICK HERE?

- **A compiler/interpreter does the actual job!**
- **Read text file, interpret the contents, output another file more understandable by machines, less understandable by us ☺**



MODERN IDES

- Previous slide is not a modern day practice.
  - At least only computer engineers likes to do it.
- Now we have IDEs (Integrated Development Environment)
  - Providing
    - Text editors
    - Compilers
    - And many other useful features

IDE SAMPLES

AlternativeFuelPatrolRouting - Microsoft Visual Studio

File Edit View Project Build Debug Team Tools Test Analyze Window Help

Debug x64 Start

Toolbox

Search Toolbox

- Other Widgets
- Images & Media
- Layouts
- Composite
- Advanced
- Form Widgets
- Text Fields
- Time & Date
- Other Layouts
- General

There are no usable controls in this group. Drag an item onto this text to add it to the toolbox.

IProblem.cs | ProblemBase.cs | VehicleType.cs | vehicles.csv | ComputationalStudyForm.cs

AlternativeFuelPatrolRouting | AlternativeFuelPatrolRouting.Implementations.Pr | problemCharacteristics

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using AlternativeFuelPatrolRouting.Interfaces;
using AlternativeFuelPatrolRouting.Model.Problem;
using AlternativeFuelPatrolRouting.Model;
using AlternativeFuelPatrolRouting.Util;

namespace AlternativeFuelPatrolRouting.Implementations
{
    public abstract class ProblemBase : IProblem
    {
        protected ProblemCharacteristics problemCharacteristics;
        public ProblemCharacteristics ProblemCharacteristics { get { return problemCharacteristics; } }

        protected List<Hotspot> hotspots;
        public List<Hotspot> Hotspots { get { return hotspots; } }

        protected List<Vehicle> vehicles;
        public List<Vehicle> Vehicles { get { return vehicles; } }

        protected List<ChargingStation> chargingStations;
        public List<ChargingStation> ChargingStations { get { return chargingStations; } }

        protected List<AdminStation> adminStations;
        public List<AdminStation> AdminStations { get { return adminStations; } }

        protected List<Road> roads;
        public List<Road> Roads { get { return roads; } }

        public ProblemBase()
        {
            hotspots = new List<Hotspot>();
            vehicles = new List<Vehicle>();
            chargingStations = new List<ChargingStation>();
            adminStations = new List<AdminStation>();
            roads = new List<Road>();
            problemCharacteristics = new ProblemCharacteristics();

            problemCharacteristics.addCharacteristic(new Variable(VariableID.ALPHA, "Alpha: Travel speed miles pe
            problemCharacteristics.addCharacteristic(new Variable(VariableID.KAPPA, "Kappa", "120.0"));
            problemCharacteristics.addCharacteristic(new Variable(VariableID.T_BAR, "T_Bar", "480.0"));
```

Team Explorer - Home

Home | AlternativeFuelPatrolRouting

Project: Changes, Branches, Sync, Settings

Solutions: New... | Open...

AlternativeFuelPatrolRouting.sln

Solution Explorer | Team Explorer

Properties

Error List | Task List | Output | Find Symbol Results | Code Metrics Results | Call Hierarchy

Ready Ln 10 Col 1 Ch 1 INS AlternativeFuelPatrolRouting master

IDE SAMPLES

Java - ATM/src/org/conner/demo/atm/Driver.java - Eclipse

File Edit Source Refactor Navigate Search Project Run Window Help

Package Explorer

- ATM
  - src
    - org.conner.demo.atm
  - Referenced Libraries
  - doc
    - index-files
    - org
    - allclasses-frame.html
    - allclasses-noframe.html
    - constant-values.html
    - deprecated-list.html
    - help-doc.html
    - index.html
    - overview-tree.html
    - package-list
    - script.js
    - stylesheet.css
  - build.xml
- ATM-JUnit
- cs200tryout
- HackerRank

Driver.java | Card.java | Customer.java | DebitCard.java | Transaction.java | ATM.java

```
1  package org.conner.demo.atm;
2
3  /**
4   * The driver of the ATM demo
5   * @author chansen
6   *
7   */
8  public class Driver {
9
10
11     public static void main(String[] args) {
12         ATM a = new ATM();
13         a.init();
14     }
15
16 }
17
```

Task List | Outline

- org.conner.demo.atm
- Driver
  - main(String[]) : void

Ant

- ATM [ATM/build.xml]
  - build [default]
  - build-eclipse-compiler
  - build-project
  - build-refprojects
  - build-subprojects
  - clean
  - cleanall
  - Driver
  - init
  - init-eclipse-compiler
  - Main

Problems | Javadoc | Declaration | Console | Debug | Expressions | Error Log

Driver [Java Application] C:\Program Files\Java\jre1.8.0_73\bin\javaw.exe (Mar 23, 2017, 7:24:15 PM)

```
[1] - Deposit Amount

[3] - Withdraw Amount

[4] - Exit

Enter selection:
```

WHAT IS A PROGRAM?

- **Just some sequential lines to instruct the computer what to do!**

```
int a = 5
int b = 6
int c = a + b

print c
```

WHAT IS A PROGRAMMING LANGUAGE?

- Provides the developer useful constructs
  - Defining variables
  - Looping
  - Conditionals
  - Many more…
- And advanced constructs
  - Data management
  - Memory management
  - Networking
  - Scheduling

WHICH LANGUAGE

- **According to Google*:**
  - **The fastest language is C++**

Benchmark	Time [sec]	Factor
C++ Opt	23	1.0x
C++ Dbg	197	8.6x
Java 64-bit	134	5.8x
Java 32-bit	290	12.6x
Java 32-bit GC*	106	4.6x
Java 32-bit SPEC GC	89	3.7x
Scala	82	3.6x
Scala low-level*	67	2.9x
Scala low-level GC*	58	2.5x
Go 6g	161	7.0x
Go Pro*	126	5.5x

  - **BUT**
    - **If you don't optimize your code, you may end up slower than other languages**
      - **See C++ Dbg vs other languages**

*Loop Recognition in C++/Java/Go/Scala by Robert Hundt - Google

OF COURSE…

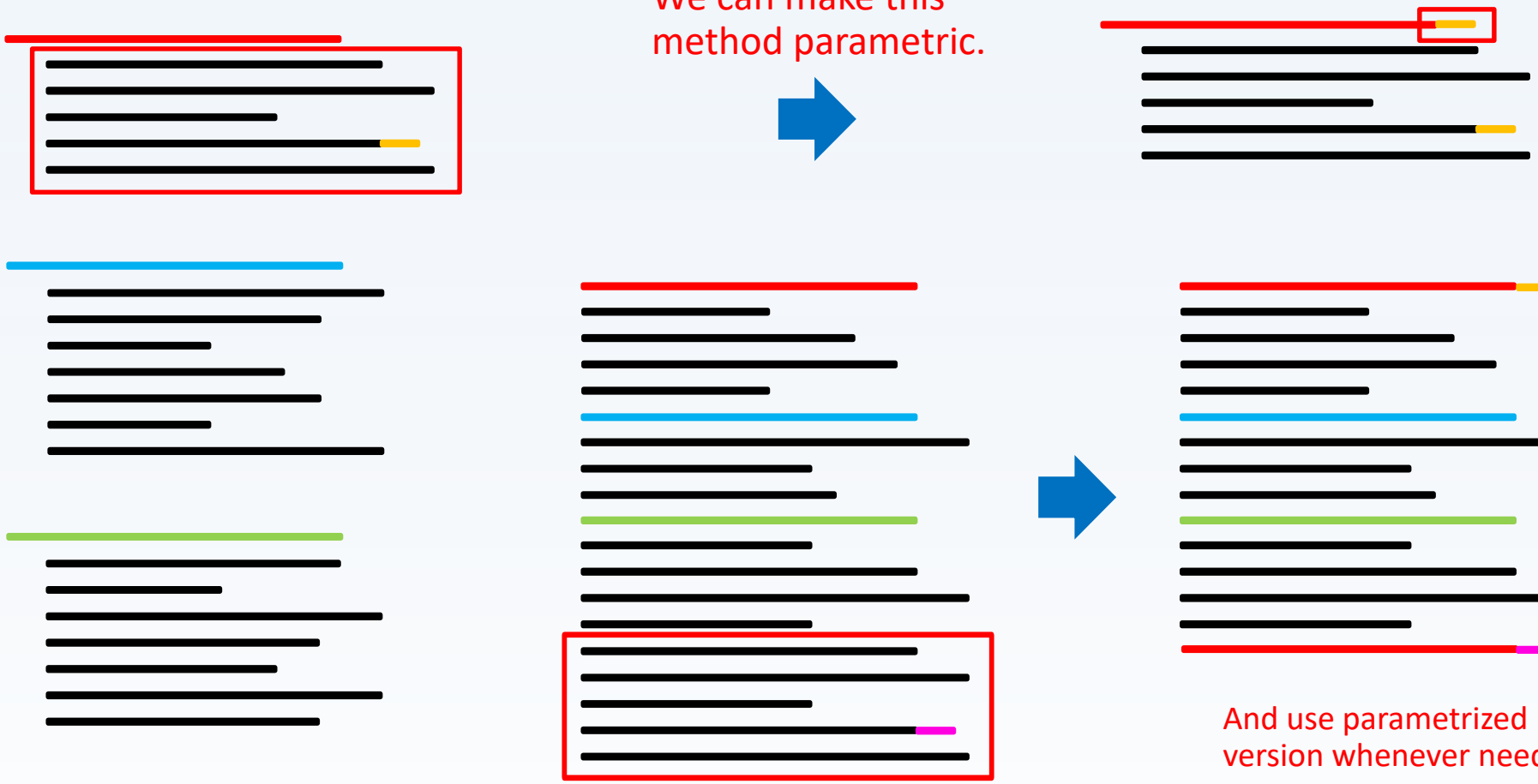
- The programs are not small any more!
  - Chrome browser: 17 millions LOC (lines of code)
  - Office 2013: 45 millions LOC
  - Facebook: 60 millions LOC
- You can't just put all the lines sequentially and expect someone to understand!
  - First, it is impossible.
  - Second, it is torture.
- Of course we need extra structures to handle the complexity

WHEN THINGS GET BIGGER - MODULARIZE

We can define methods in order to make our program mode readable.

MODULARIZE EVEN MORE

We can make this method parametric.



And use parametrized version whenever needed.

FUN PART

SPOILER ALERT: YOU WILL SEE SOME CODE IN ACTION!

ENOUGH OF THE LINES, GIVE ME SOMETHING

```
1
2   print("I am a dog")
3   print("I have 4 legs")
4   print("My sound is: woof woof")
5
```

I want to do this 3 times?

```
1
2   print("I am a dog")
3   print("I have 4 legs")
4   print("My sound is: woof woof")
5
6   print("I am a dog")
7   print("I have 4 legs")
8   print("My sound is: woof woof")
9
10  print("I am a dog")
11  print("I have 4 legs")
12  print("My sound is: woof woof")
13
```

```
1
2   def dogInformation():
3       print("I am a dog")
4       print("I have 4 legs")
5       print("My sound is: woof woof")
6
7   dogInformation()
8   dogInformation()
9   dogInformation()
10
```

MODULARIZING - PARAMETRIZED

```python

def dogInformation():
    print("I am a dog")
    print("I have 4 legs")
    print("My sound is: woof woof")

dogInformation()
dogInformation()
dogInformation()

```

```python

def dogInformation():
    print("I am a dog")
    print("I have 4 legs")
    print("My sound is: woof woof")
    print("My name is Rexx")

dogInformation()
dogInformation()
dogInformation()

```

```python

def dogInformation(name):
    print("I am a dog")
    print("I have 4 legs")
    print("My sound is: woof woof")
    print("My name is " + name)

dogInformation("Rexx")
dogInformation("Bingo")
dogInformation("Rocky")

```

MODULARIZING – MORE PARAMETRIZED

```python

def chickenInformation(name):
    print("I am a chicken")
    print("I have 2 legs")
    print("My sound is: cluck cluck")
    print("My name is " + name)

chickenInformation("Angel")
chickenInformation("Coco")

def dogInformation(name):
    print("I am a dog")
    print("I have 4 legs")
    print("My sound is: woof woof")
    print("My name is " + name)

dogInformation("Rexx")
dogInformation("Bingo")
dogInformation("Rocky")

```

```python

def animalInformation(type, legCount, sound, name):
    print("I am a " + type)
    print("I have " + legCount + " legs")
    print("My sound is: " + sound)
    print("My name is " + name)

animalInformation("chicken", 2, "cluck cluck", "Angel")
animalInformation("chicken", 2, "cluck cluck", "Coco")

animalInformation("dog", 4, "woof woof", "Rexx")
animalInformation("dog", 4, "woof woof", "Bingo")
animalInformation("dog", 4, "woof woof", "Rocky")

```

We still have a lot of repetitions.

EPIPHANY

- **Wouldn't it be nice if there is a special structure that:**
  - **Knows what kind of an animal it is.**
  - **Knows how many legs it has.**
  - **Can make sound by itself.**
  - **Has a name.**
- **So that, I don't have to write them every time.**

```
1
2   def animalInformation(type, legCount, sound, name):
3       print("I am a " + type)
4       print("I have " + legCount + " legs")
5       print("My sound is: " + sound)
6       print("My name is " + name)
7
8   animalInformation("chicken", 2, "cluck cluck", "Angel")
9   animalInformation("chicken", 2, "cluck cluck", "Coco")
10
11  animalInformation("dog", 4, "woof woof", "Rexx")
12  animalInformation("dog", 4, "woof woof", "Bingo")
13  animalInformation("dog", 4, "woof woof", "Rocky")
14
```

EPIPHANY

So, we are basically talking about a dog?

It has a name, knows how many legs it has, can make sound itself.

EPIPHANY

- **So we need to mimic the real life in our applications.**
- **Because a program is for the developer to write/understand easily, not for the computer.**

"First, we want to establish the idea that a computer language is not just a way of getting a computer to perform operations but rather it is a novel formal medium for expressing ideas about methodology.

Thus, programs must be written for people to read, and only incidentally for machines to execute."

Harold Abelson*

*Author of the renowned **Structure and Interpretation of Computer Programs** book (with Jay Sussman)

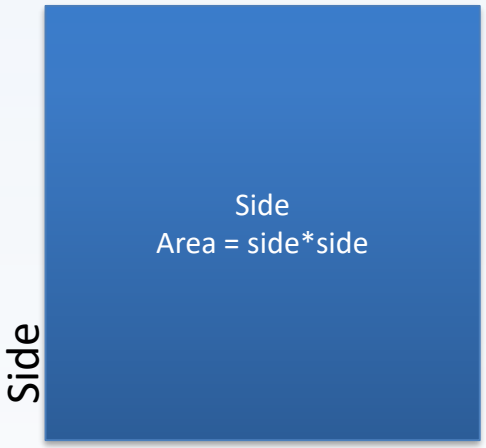
EPIPHANY

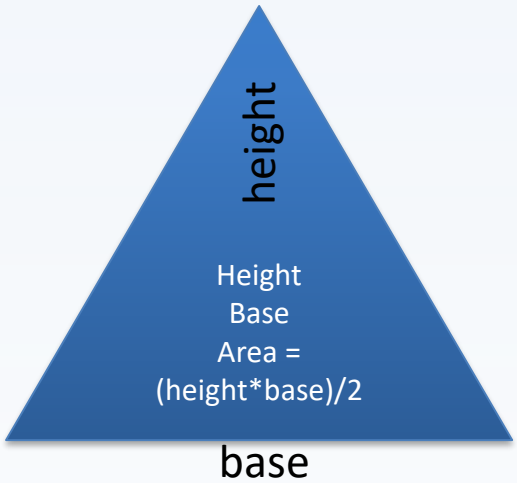
- Wouldn't it be nice if there is a special structure that:
  - Knows what kind of an animal it is.
  - Knows how many legs it has.
  - Can make sound by itself.
  - Has a name.

- We already have a system for this.
  - Object-oriented programming.
    - A.k.a.
      - Object-oriented methodology
      - Object-oriented paradigm
      - Object-oriented design and analysis

OBJECT-ORIENTED PROGRAMMING

- **Basically, we have objects that know:**
  - **Its attributes**
  - **Its operations**





- **Whenever, I need a square or triangle, I will use them.**
  - **I ask for information and they answer.**

OBJECT IDENTIFICATION

- Car, glass, bike, bus
  - All can be objects
- Dog, cat, person, laptop
  - Again, can be objects
- Running, walking, making a sound etc… ?
  - No, these are not objects!
  - These are actions (methods/operations) that can be done by an object
    - Like a Rectangle can compute its own area
    - A people can run.
- Side, height, numberOfLegs etc… ?
  - No, these are not objects!
  - These are properties (attributes) that can be owned by an object
    - Like side of a Rectangle.
    - Numberoflegs of a Person.

EVERYTHING CAN BE AN OBJECT

- It just depends on the context!
- Distance between two cities can be an object
  - If it is the main focus of the problem.

- Relationship between two people can be an object
  - If we are solving a problem about this.

LET'S IDENTIFY SOME OBJECTS

- Company XYZ is a manufacturing company that produces cartoon action figurines for big entertainment companies.
- This company is using an inventory and tracking system.
- The inventory system keeps track of how many of each figurine is stored in each warehouse.
- Figures are stored in cases.
- Clients order the figurines and the cases are eventually shipped to clients.

OBJECTS IDENTIFIED

- Company XYZ is a manufacturing company that produces cartoon action figurines for big entertainment companies.
- This company is using an inventory and tracking system.
- The inventory system keeps track of how many of each figurine is stored in each warehouse.
- Figures are stored in cases.
- Clients order the figurines and the cases are eventually shipped to clients.

ANY MORE?

- Company XYZ is a manufacturing company that produces cartoon action figurines for big entertainment companies.
- This company is using an inventory and tracking system.
- The inventory system keeps track of how many of each figurine is stored in each warehouse.
- Figures are stored in cases.
- Clients order the figurines and the cases are eventually shipped to clients.

ANOTHER EXAMPLE

- An ATM needs to allow a customer to identify themselves
  - Each customer has a debit card and PIN
- Customers should be presented with some kind of menu to help direct them.
- Customers can perform two transactions:
  - They should be able to deposit funds
  - They should be able to withdraw funds upto $200
    - These funds must be withdrawn in units of $20
- The ATM should tell some banking software to update the customers' account at the end of transaction
- The ATM should also give the customer some record of the transaction.

OBJECTS IDENTIFIED

- An ATM needs to allow a customer to identify themselves
  - Each customer has a debit card and PIN
- Customers should be presented with some kind of menu to help direct them.
- Customers can perform two transactions:
  - They should be able to deposit funds
  - They should be able to withdraw funds upto $200
    - These funds must be withdrawn in units of $20
- The ATM should tell some banking software to update the customers' account at the end of transaction
- The ATM should also give the customer some record of the transaction.

This PIN number object seems redundant.

We can make it a property of either customer or debit card.

This small design decision will make a lot of difference.

OBJECT-ORIENTED LANGUAGES

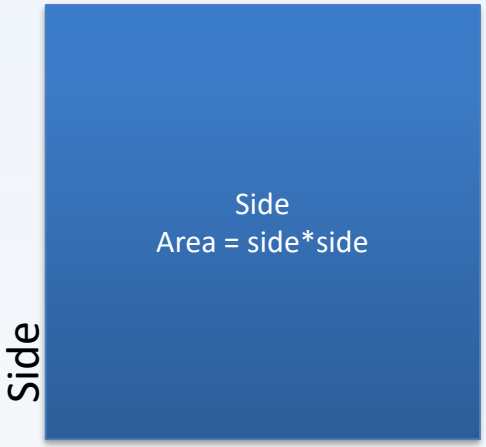
- Most modern languages support object-orientation now
  - Java
  - C++ (C with classes)
  - C#
  - Python
  - Even Fortran
  - Any many more

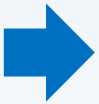
A REPRESENTATION SYSTEM

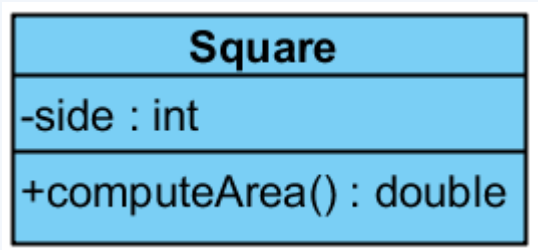
- Various languages support object-orientation.
- So, there should be a way to represent this system regardless of the language we use.

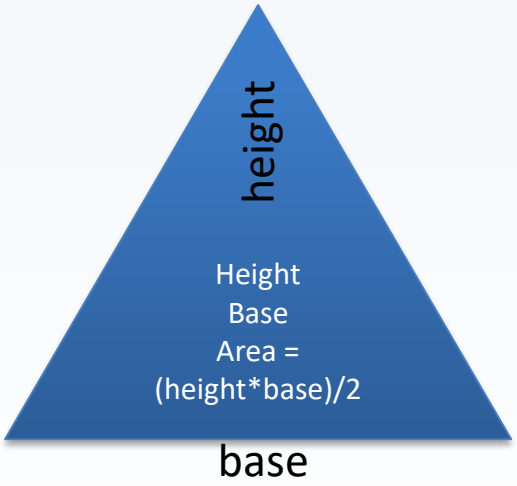
- UML is here.

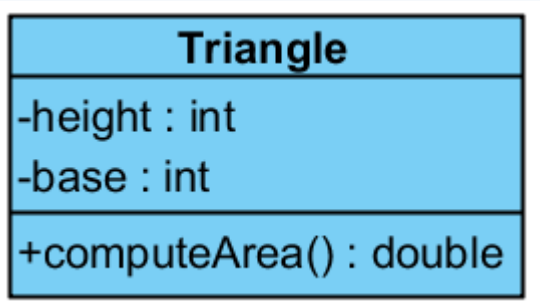
UML BASICS - CLASS



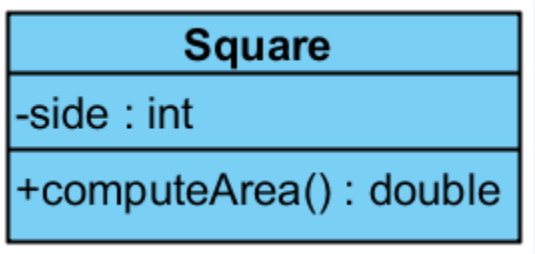








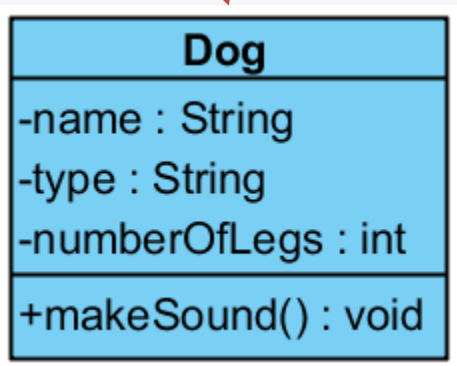
CLASS IN C#




```
namespace OR_Seminar
{
    public class Square
    {
        int side;

        public Square(int side)
        {
            this.side = side;
        }

        public double computeArea()
        {
            return side * side;
        }
    }
}
```

WAIT

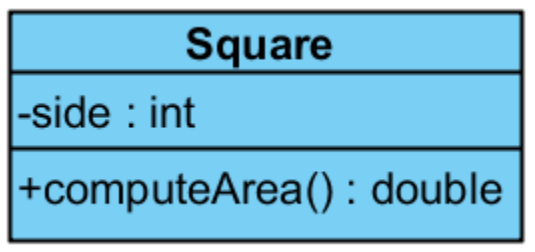
- We were talking about objects, now what the heck is a class?
- Time to define the relation between these two.



- Basically, we get the properties that all the dogs have and define a Dog class with them.
  - A class is a generalization of objects.
  - And objects are just instances of the class.

OBJECTS FROM CLASSES

- **Constructor**




```
namespace OR_Seminar
{
    public class Square
    {
        int side;

        public Square(int side)
        {
            this.side = side;
        }

        public double computeArea()
        {
            return side * side;
        }
    }
}
```

- **Now we can create objects**
  - **From classes**
  - **Using a constructor**

```
namespace OR_Seminar
{
    class Program
    {
        static void Main(string[] args)
        {
            Square square1 = new Square(5);
            Square another = new Square(10);
        }
    }
}
```

CONCEPTS & DESIGN PRINCIPLES IN OBJECT-ORIENTED PROGRAMMING

- **Concepts:**
  - **Encapsulation**
  - **Inheritance**
  - **Polymorphism**
  - **Cohesion**
  - **Coupling**
- **Principles**
  - **Open-Closed Principle**
  - **Don't Repeat Yourself Principle**
  - **Single Responsibility Principle**
  - **Liskov Substitution Principle**
  - **Interface Seggregation Principle**
  - **Dependency Inversion Principle**

If we don't use these concepts and obey these principles, we aren't coding in proper object-oriented way.

ENCAPSULATION

- **Protecting your information from being used incorrectly.**

```java

public class Airplane {
    public int speed;

    public Airplane() { }
}

```

```java

public class Main {

    public static void main(String[] args) {
        Airplane plane1 = new Airplane();
        plane1.speed = 100;
        System.out.println(plane1.speed);
    }

}

```

PROBLEM?

What if speed if out of limits?

What if we want some adjustments before setting the speed?

ENCAPSULATION

```
1
2  public class Airplane {
3      public int speed;
4
5      public Airplane() { }
6
7      public void setSpeed(int newSpeed) {
8          this.speed = newSpeed;
9          this.speed = this.speed - 9; // adjustment for rainy weather
10     }
11 }
12
```

```
1
2  public class Main {
3
4      public static void main(String[] args) {
5          Airplane plane1 = new Airplane();
6          plane1.speed = 100;
7          System.out.println(pl
8
9          plane1.setSpeed(100);
10         System.out.println(pl
11     }
12
13 }
14
```

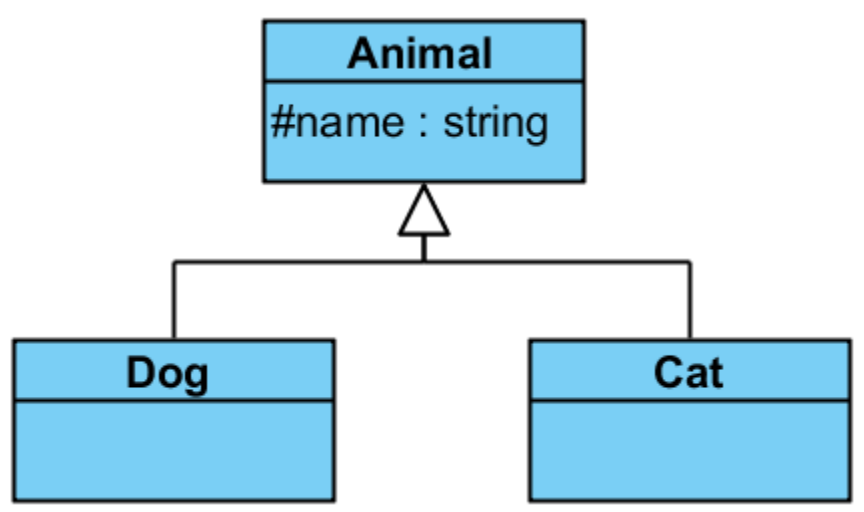
SOLUTION?

```
1
2  public class Airplane {
3      private int speed;
4
5      public Airplane() { }
6
7      public void setSpeed(int newSpeed) {
8          this.speed = newSpeed;
9          this.speed = this.speed - 9; // adjustment for rainy weather
10     }
11 }
12
```

INHERITANCE

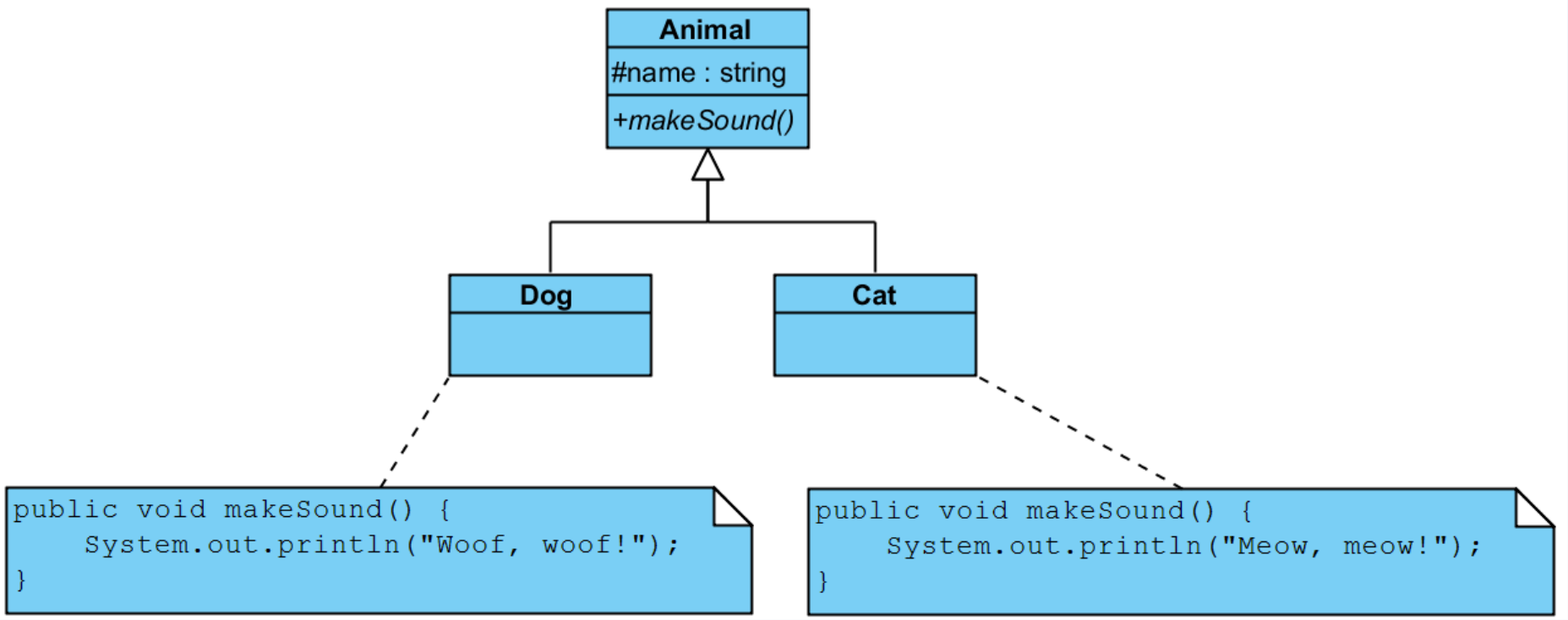
- A class' inheriting properties and operations from another class.

- Both Dogs and Cats have names, right?
  - Then, why shouldn't we create a base class Animal for them?



POLYMORPHISM

- **Having different forms of same operations.**



POLYMORPHISM

```java

public abstract class Animal {
    protected String name;
    public abstract void makeSound();
}

```

```java

public class Dog extends Animal {

    @Override
    public void makeSound() {
        System.out.println("Woof, woof!");
    }

}

```

```java

public class Cat extends Animal {

    @Override
    public void makeSound() {
        System.out.println("Meow, meow!");
    }

}

```

COHESION

- A class should do one thing really well, and shouldn't try to do or be someone else.

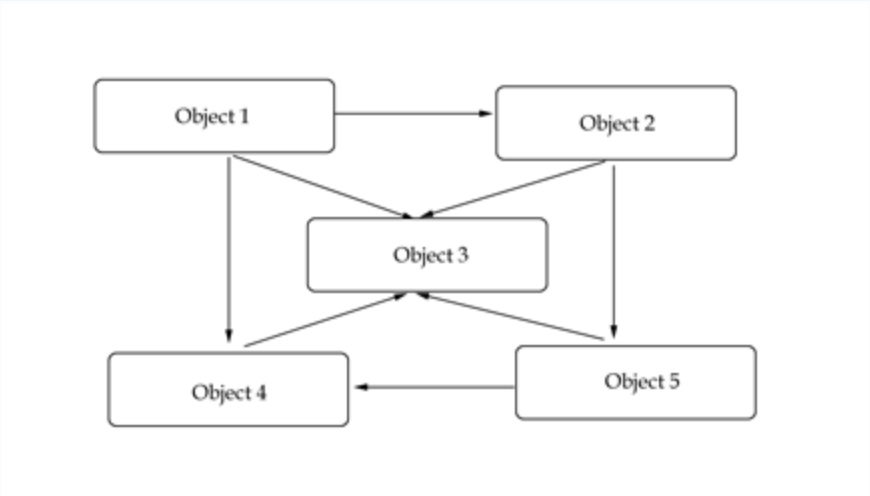
- Strong cohesion means: all methods of a class are more or less related.
  - A Math class which can compute sqrt, power, exp, cos etc.

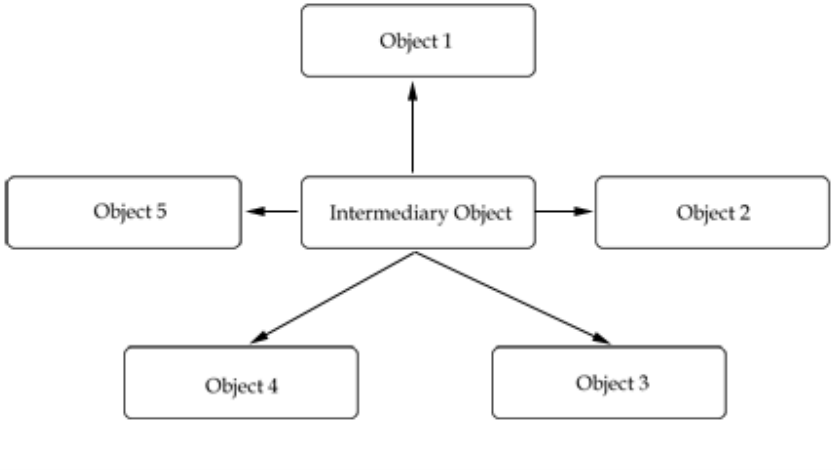
- If a class has methods for:
  - Printing a document
  - Sending an email
  - Working with trigonometric functions
  - How should we name it? Complicated, right?

```
public class Magic
{
    public void PrintDocument(Document d) { … }
    public void SendEmail(string recipient,
        string subject, string text) { … }
    public void CalculateDistanceBetweenPoints(
        int x1, int y1, int x2, int y2) { … }
}
```

COUPLING

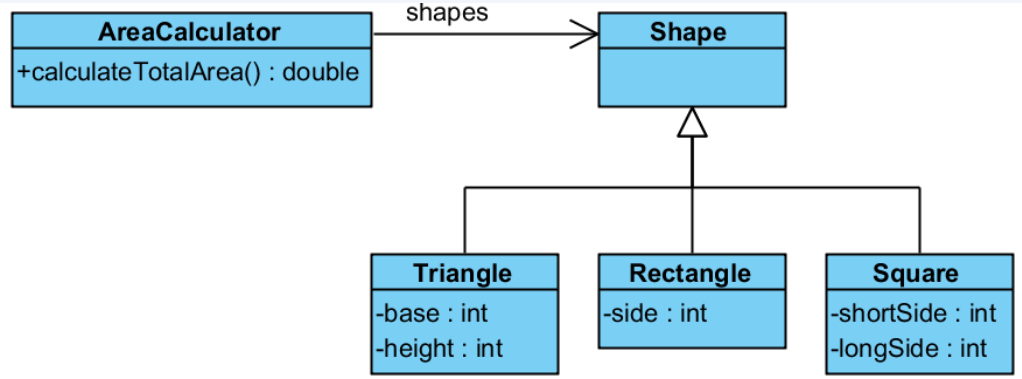
- **The extent to which classes depend on one another.**
  - **A class should work independently without being coupled too much to other classes.**
  - **This helps us making them modules and available on demand.**





OPEN-CLOSED PRINCIPLE

- **Classes should be open to extension, but closed for modification.**

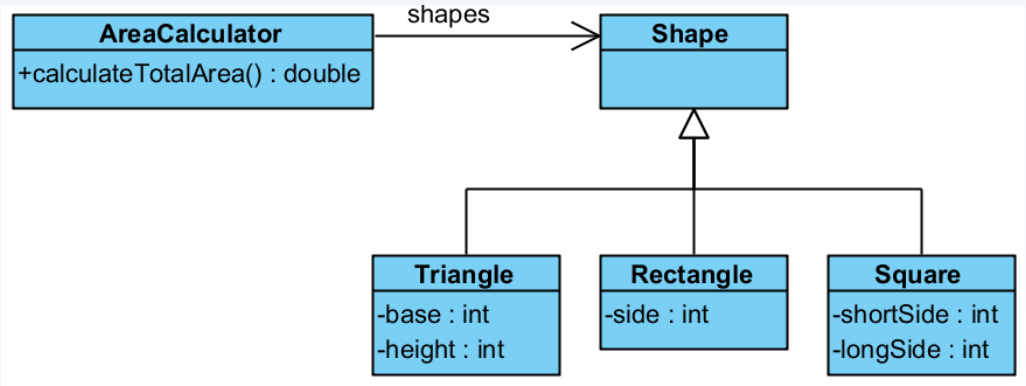


PROBLEM?

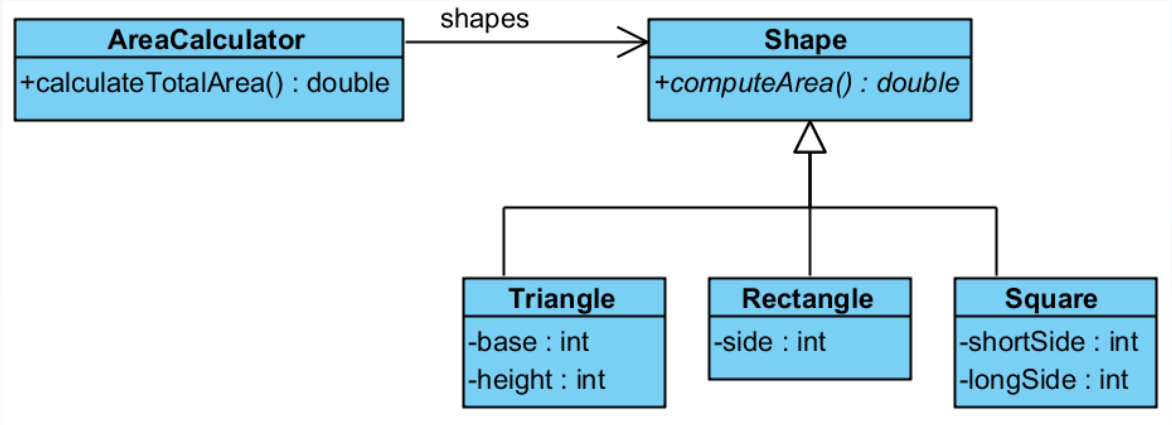
Whenever we add a new shape, we should modify the calculateTotalArea method.

```
public double calculateTotalArea() {
    double result = 0;

    // VIOLATES the OCP principle
    for (Shape shape : shapes) {
        if (shape instanceof Triangle) {
            result += ((Triangle) shape).base * ((Triangle) shape).height / 2;
        } else if (shape instanceof Square) {
            result += ((Square) shape).length * ((Square) shape).length;
        } else if (shape instanceof Rectangle) {
            result += ((Square) shape).length * ((Square) shape).length;
        }
    }

    return result;
}
```

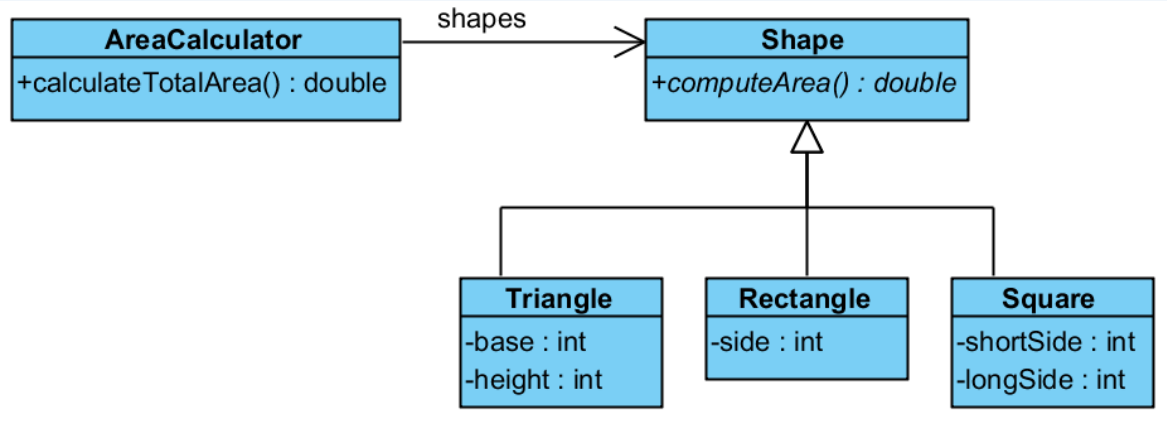
OPEN-CLOSED PRINCIPLE



SOLUTION?



OPEN-CLOSED PRINCIPLE




```java
public double calculateTotalArea() {
    double result = 0;

    // VIOLATES the OCP principle
    for (Shape shape : shapes) {
        if (shape instanceof Triangle) {
            result += ((Triangle) shape).base * ((Triangle) shape).height / 2;
        } else if (shape instanceof Square) {
            result += ((Square) shape).length * ((Square) shape).length;
        } else if (shape instanceof Rectangle) {
            result += ((Square) shape).length * ((Square) shape).length;
        }
    }

    return result;
}
```

```java
    public double calculateTotalArea() {
        double result = 0;

        // Fixing the OCP principle violation
        for (Shape shape : shapes) {
            result += shape.computeArea();
        }

        return result;
    }
```

DON'T REPEAT YOURSELF PRINCIPLE

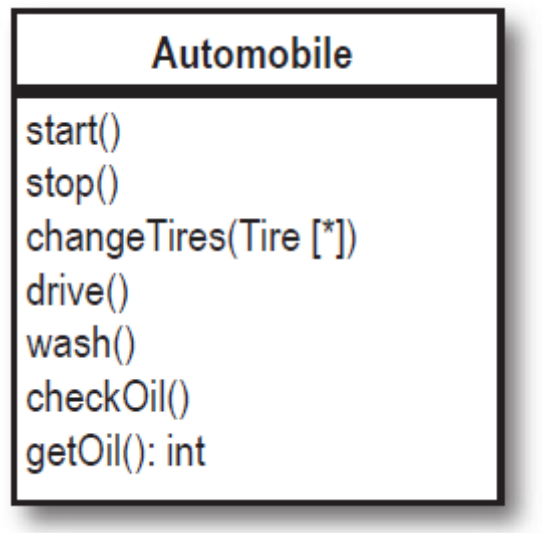
- **Avoid duplicate code by abstracting common things out and placing them in a single location.**

```java

public class Mechanic {
    public void serviceCar() {
        System.out.println("Checking in the customer!");
        System.out.println("Servicing car now!");
        System.out.println("Doing final check!");
        System.out.println("Preparing the bill!");
        System.out.println("Servicing car now!");
    }

    public void serviceBike() {
        System.out.println("Checking in the customer!");
        System.out.println("Servicing car now!");
        System.out.println("Doing final check!");
        System.out.println("Preparing the bill!");
        System.out.println("Servicing car now!");
    }
}

```

```java

public class Mechanic {
    public void serviceCar() {
        checkin();
        System.out.println("Servicing car now!");
        checkout();
    }

    public void serviceBike() {
        checkin();
        System.out.println("Servicing car now!");
        checkout();
    }

    void checkin() {
        System.out.println("Checking in the customer!");
    }

    void checkout() {
        System.out.println("Doing final check!");
        System.out.println("Preparing the bill!");
        System.out.println("Servicing car now!");
    }
}

```

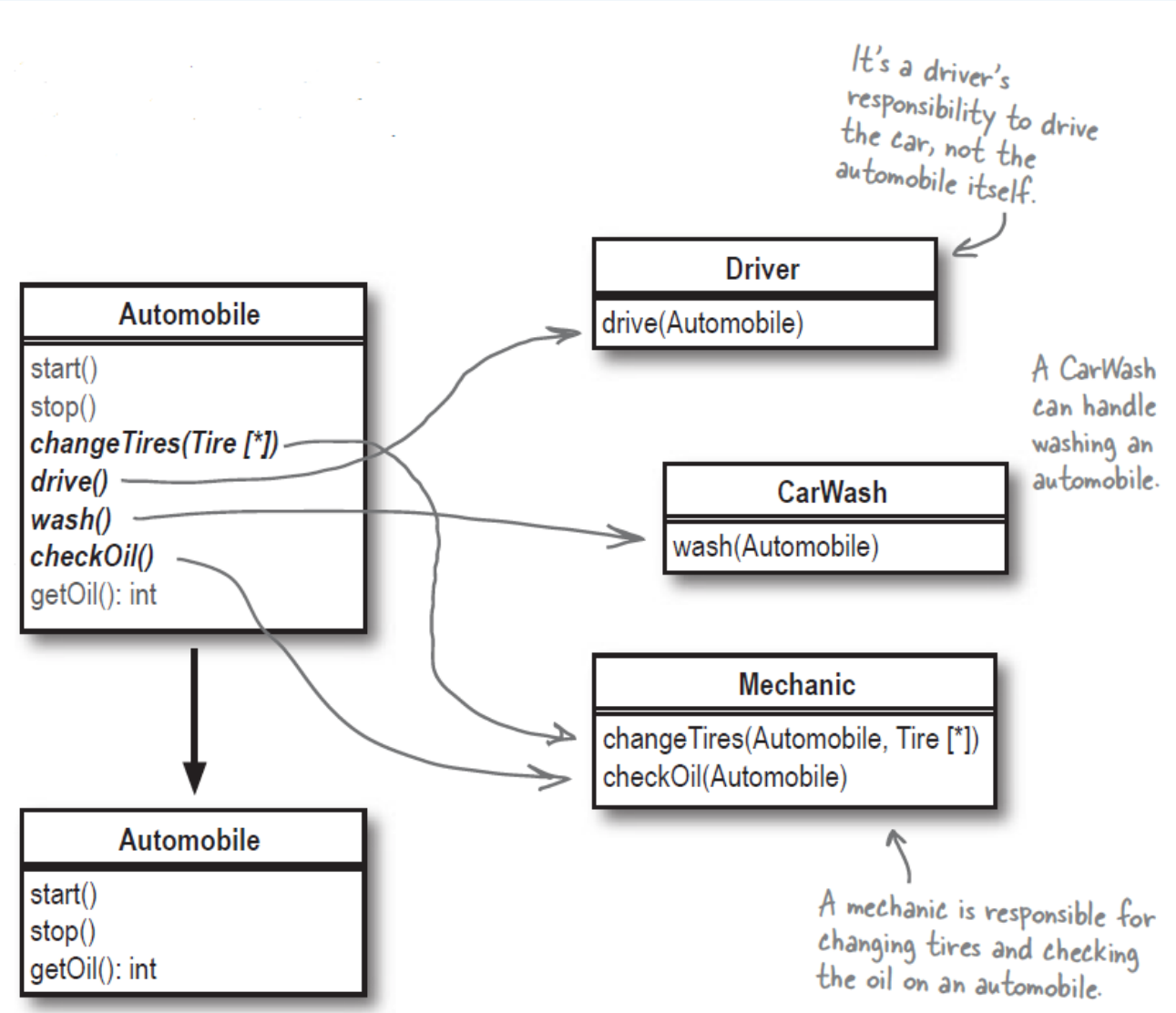
SINGLE RESPONSIBILITY PRINCIPLE

- **Every object in the system should have one responsibility**
  - **Each object has only one reason to change**
  - **Doesn't mean it should only have one method**



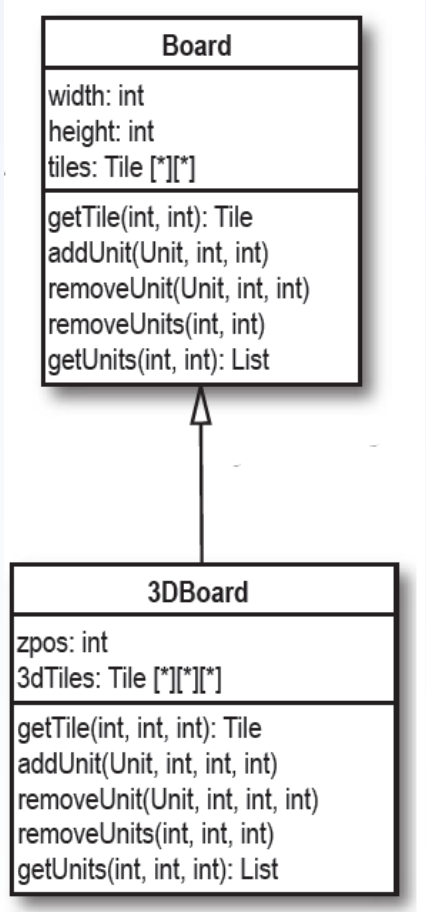
- **Usually the violation of this principle can be identified by asking:**
  - **Can _____ _____ itself?**
    - **Example: Can Automobile changeTires itself?**
    - **If it is no, it is violating SRP.**

SINGLE RESPONSIBILITY PRINCIPLE



LISKOV SUBSTITUTION PRINCIPLE

- **Sub-classes must be substitutable for their base classes.**
  - **If you inherit from a wrong base class, then you can't do this.**

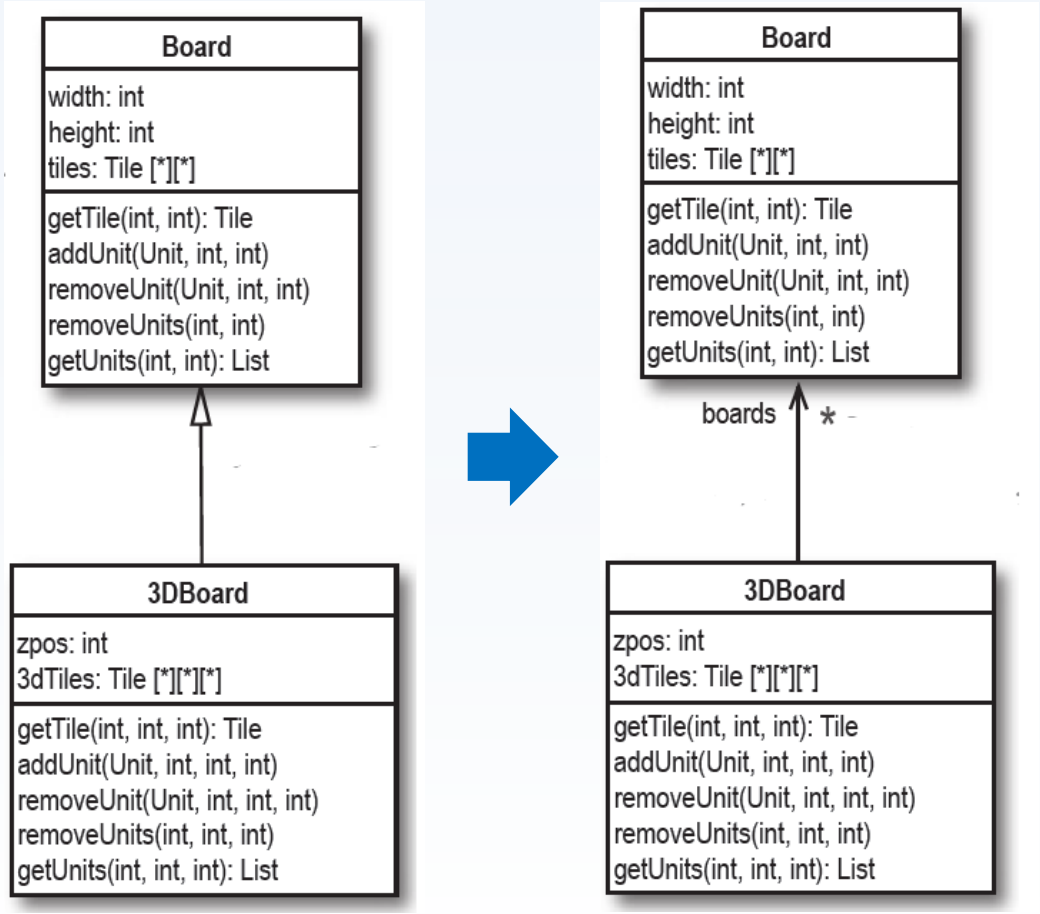


PROBLEM?

We inherit from the board, but we hardly use its methods or properties. Because they don't match with 3D board.

LISKOV SUBSTITUTION PRINCIPLE

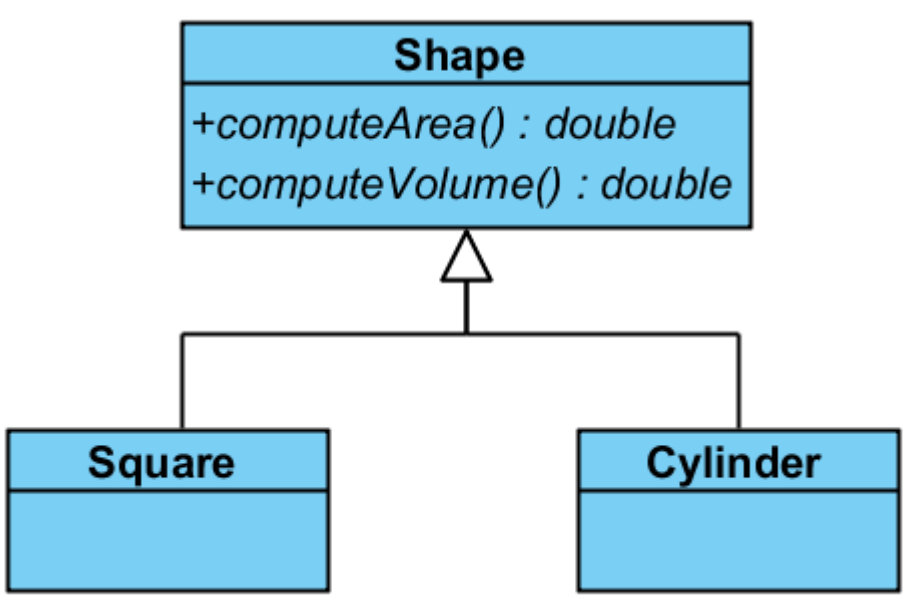
- **Sub-classes must be substitutable for their base classes.**
  - **If you inherit from a wrong base class, then you can't do this.**



SOLUTION?

INTERFACE SEGGREGATION PRINCIPLE

- **A class should never be forced to have some unnecessary methods.**

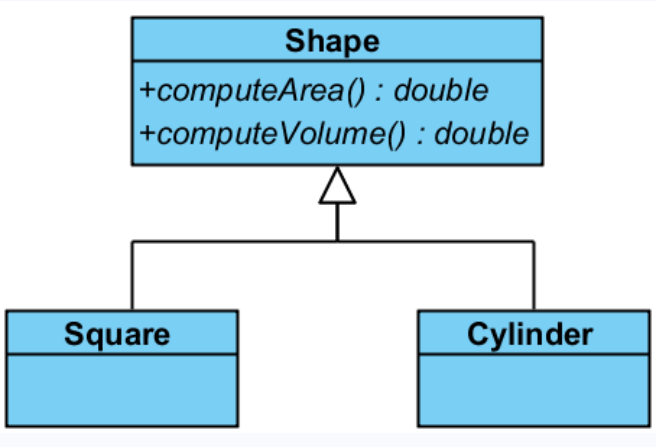


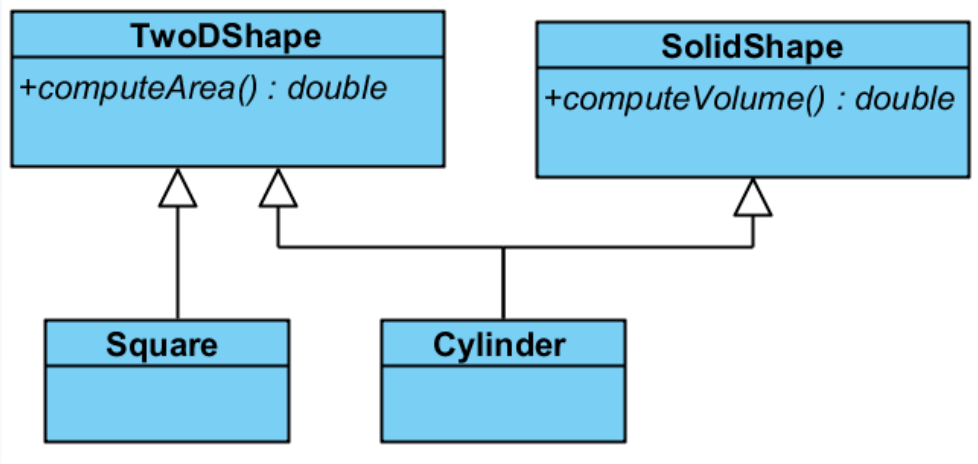
PROBLEM?

We force Square to have a computeVolume method, which it doesn't have!

INTERFACE SEGGREGATION PRINCIPLE

SOLUTION?



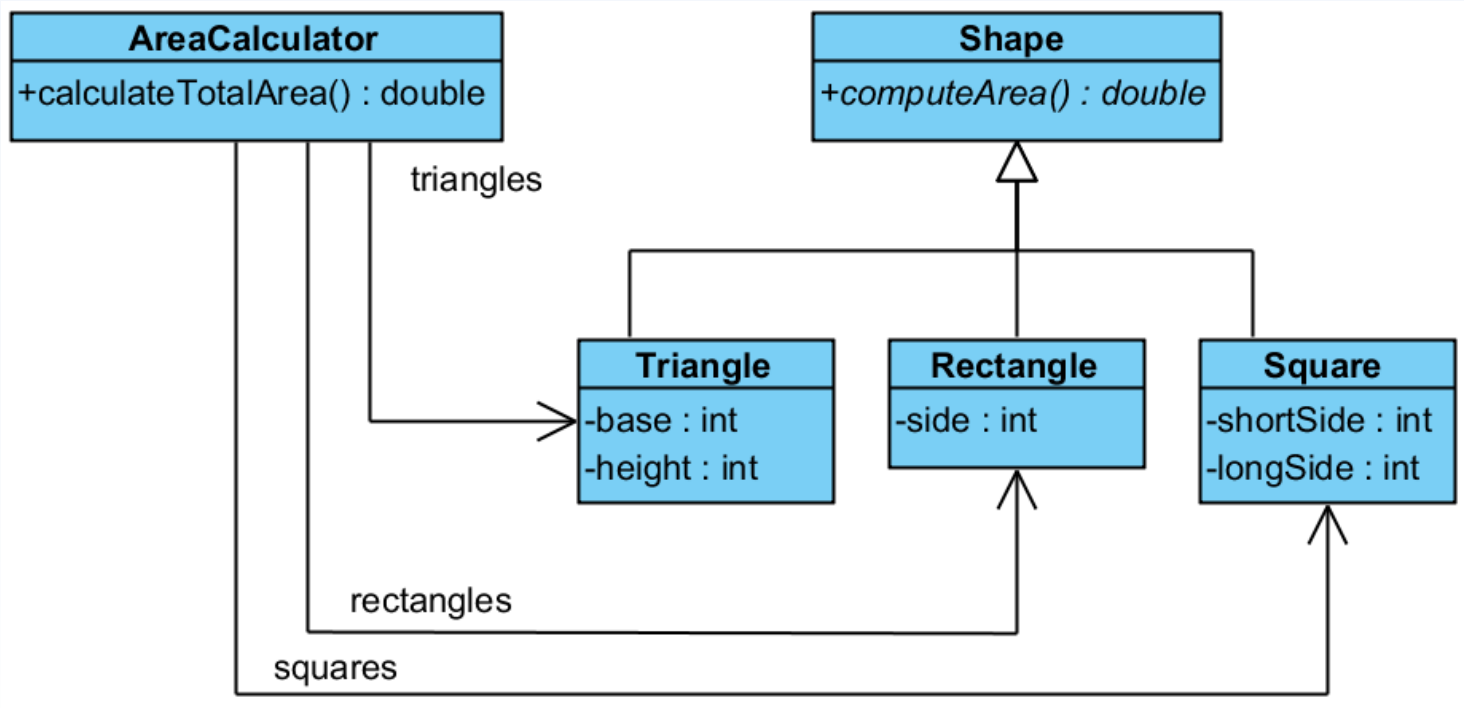


DEPENDENCY INVERSION PRINCIPLE

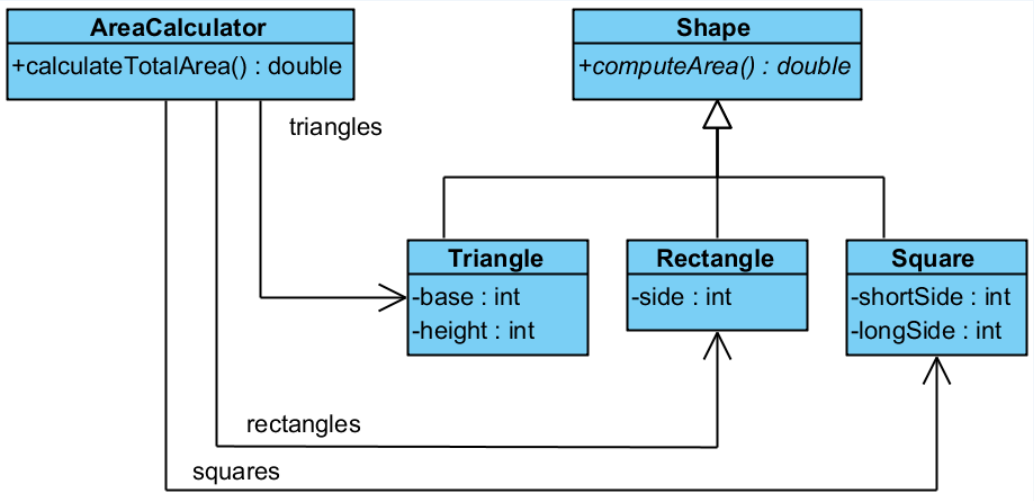
- **Entities must depend on abstractions/interfaces rather than actual classes.**
  - **So that they can be decoupled.**

PROBLEM?

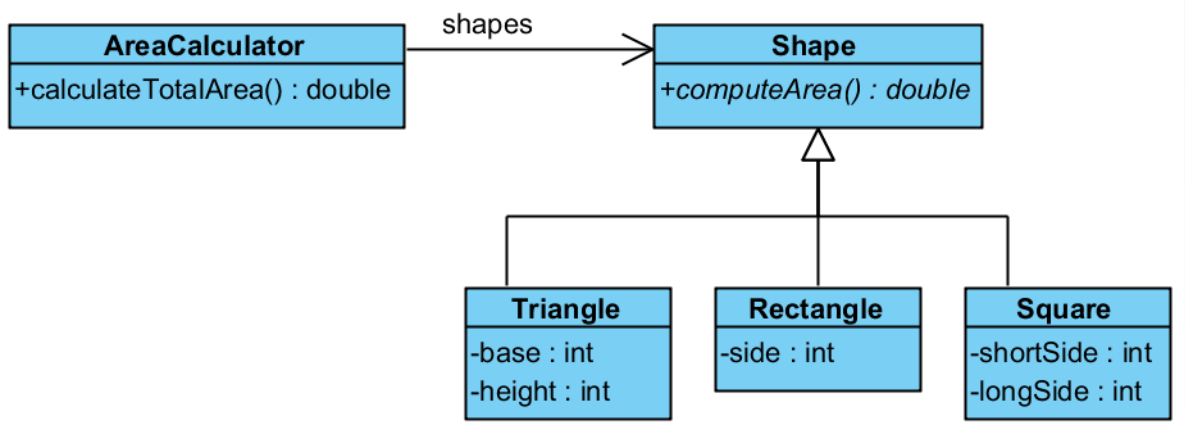
We have to treat each shape separately. Because we have links to actual classes.



DEPENDENCY INVERSION PRINCIPLE



SOLUTION?



WHAT CAN WE ACHIEVE FROM THIS? SOME IDEAS

- Read data from various sources but our program can stay the same
- Easily switch between different algorithms on the same data
- Make any module work independent from others
- Make the output of the project independent from the data or the algorithm itself
- Test the correctness of each class independently
- Model the problem in a human-readable way
- And most importantly, reuse and maintain your application better.
  - Future-proof.

OTHER TOOLS & TECHNOLOGIES IN ORDER TO IMPROVE YOUR PROGRAMMING

- Versioning (github, bitbucket etc.)
  - Version your code in order to access any change you made (and backup)
- Unit tests
  - Test each unit independently, be sure it is doing whatever it is supposed to do.
  - Most languages have support for this. Look for it.
- Documentation (Doxygen, Javadoc etc.)
  - Always comment your code. You can even produce automatic documentation from these comments.
- Diagramming
  - Your code may not be understandable, but your diagrams will be. Learn UML Class diagram and use it in your projects.

CONCLUSION

- **Object-oriented programming provides very flexible structures for our programs.**
  - **It can be applied in many languages, as long as the language supports object-orientation.**
- **If we obey the principles, it will be an actual system.**
  - **Otherwise, it is just the same code with classes and additional complexity.**
- **Object-oriented system is not a perfect system and it has its own flaws. But it is still the best system.**
- **Always strive for the best design.**

QUESTIONS?

- **Thanks for listening…**
- **For offline questions, find my contact info here: www.objectivelook.net**

SOME RESOURCES

- Object-oriented programming with C# (The book itself is nice and free, chapter 20 is OOP): http://www.introprogramming.info/english-intro-csharp-book/read-online/
- For new starters to OOP, this book is fun: https://www.amazon.com/Head-First-Object-Oriented-Analysis-Design/dp/0596008678
- Detailed explanation, nicely done, 2 pages (Java):
  - https://www.ntu.edu.sg/home/ehchua/programming/java/J3a_OOPBasics.html
  - https://www.ntu.edu.sg/home/ehchua/programming/java/J3b_OOPInheritancePolymorphism.html
- Same as above but with C++:
  - https://www.ntu.edu.sg/home/ehchua/programming/cpp/cp3_OOP.html
- Even though, there are a lot of resources. I suggest to work with someone who you can ask questions immediately. Because OOP requires a change of mindset.