

CS 535 Object-Oriented Programming & Design  
Fall Semester, 2011  
Doc 2 More OO Introduction  
Sept 1 2011

Copyright ©, All rights reserved. 2011 SDSU & Roger Whitney, 5500 Campanile Drive, San Diego, CA 92182-7700 USA. OpenContent (<http://www.opencontent.org/openpub/>) license defines the copyright on this document.

## References

Object-Oriented Design Heuristics, Chapter 2

Designing Object-Oriented Software, Wirfs-Brock, Wilkerson, Wiener

# Helper method

Method in class that

- Does not access any field (data member, instance variables)

- Just uses parameters

# Data Class

```
class Point {  
    private int x;  
    private int y;  
  
    public void setX(int newX) {  
        x = newX;  
    }  
  
    public int getX() {  
        return x;  
    }  
  
    public void setY(int newY) {  
        y = newY;  
    }  
  
    public int getY() {  
        return y;  
    }  
}
```

Class with

get/set methods

constructor

No or very few other methods

# Assignment Results

Classes	
Data Classes	
Methods	
Helper methods	

# Relevant Heuristics

2.8 A class should capture one and only one key abstraction

2.9 Keep related data and behavior in one place

2.10 Spin off nonrelated information into another class

# java.lang.Math

```
package java.lang;  
import java.util.Random;
```

```
public final strictfp class Math {  
    public static double abs(double a) {  
        return (a <= 0.0D) ? 0.0D - a : a;  
    }  
}
```

```
    public static double toDegrees(double angrad) {  
        return angrad * 180.0 / PI;  
    }  
}
```

etc.

So what do we lose  
doing this?

# Heuristic

A method to help solve a problem, commonly informal

"rules of thumb"



## Heuristic 2.3

Minimize the number of messages in the protocol of a class

"The problem with large public interfaces is that you can never find what you are looking for"

Is this a design issue or a tool issue?

What do you do when the class does not have the method you need?

## 2.1 All data should be hidden within its class

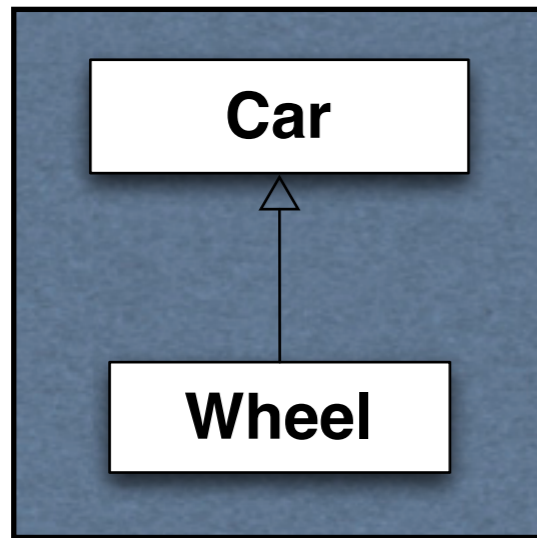
```
public class Foo {  
    public int x;  
    public int y;  
}
```

```
public class Foo {  
    private int x;  
    private int y;  
  
    public int getX() {return x;}  
    public int getY() {return y;}  
  
    public void setX(int newX){  
        x = newX  
    }  
  
    public void setY(int newY){  
        y = newY  
    }  
}
```

# Role Versus Class

Is Mother a subclass of a Person class or an instance of it?

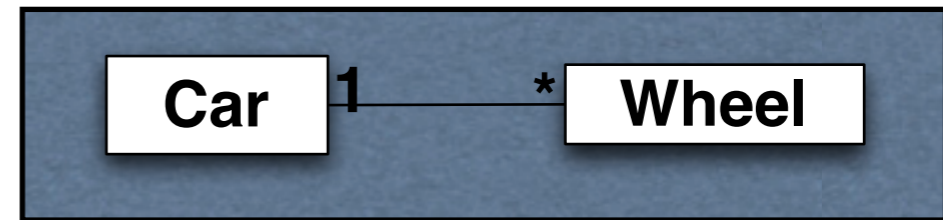
# Inheritance verses Data Members



```
class Car { ... }
```

```
class Wheel extends Car { ... }
```

Wheel is a type of car



```
class Car {
    Wheel[] tires;
    ...
}
```

Car has wheels

# Coupling

Strength of interaction between objects in system

"Unnecessary object coupling needlessly decreases the reusability of the coupled objects"

"Unnecessary object coupling also increases the chances of system corruption when changes are made to one or more of the coupled objects"

## Design Goal

The interaction or other interrelationship between any two components at the same level of abstraction within the system be as weak as possible

# Types of Coupling

## Nil Coupling

No interaction between two classes

## Export Coupling

One class uses the public interface of another

## Overt Coupling

One class uses implementation details of another class with permission

## Covert Coupling

One class uses implementation details of another class without permission

# Cohesion

Degree to which the tasks performed by a single module are functionally related

Each element in the module should be essential to the module's purpose

# Coupling & Cohesion Heuristics

Classes should only exhibit nil or export coupling with other classes

A class should capture one and only one key abstraction

Keep related data and behavior in one place

Spin off nonrelated information into another class



# Design Process

# One OO Design Process

## Exploratory Phase

Who is on the team?

What are their tasks, responsibilities?

Who works with whom?

## Analysis Phase

Who's related to whom?

Finding sub teams

Putting it all together

# Exploratory Phase

Who is on the team?

What are the goals of the system?

What must the system accomplish?

What objects are required to model the system and accomplish the goals?

Finding the initial list of classes for the system

# Exploratory Phase

What are their tasks, responsibilities?

What does each object have to know in order to accomplish its tasks?

What steps toward accomplishing each goal is it responsible for?

Candidate list of fields and methods

# Exploratory Phase

Who works with whom?

With whom will each object collaborate in order to accomplish each of its responsibilities?

What is the nature of the objects' collaboration?

How do the objects interact

# Analysis Phase

Who's related to whom?

Determine which classes are related via inheritance

Finding abstract classes

Determine class contracts

# Analysis Phase

Finding sub teams

Divide responsibilities into subsystems

Designing interfaces of subsystems and classes

# Analysis Phase

Putting it all together

Construct protocols for each class

Produce a design specification for each class and subsystem

Write a design specification for each contract