gigigo

Composition VS Inheritance

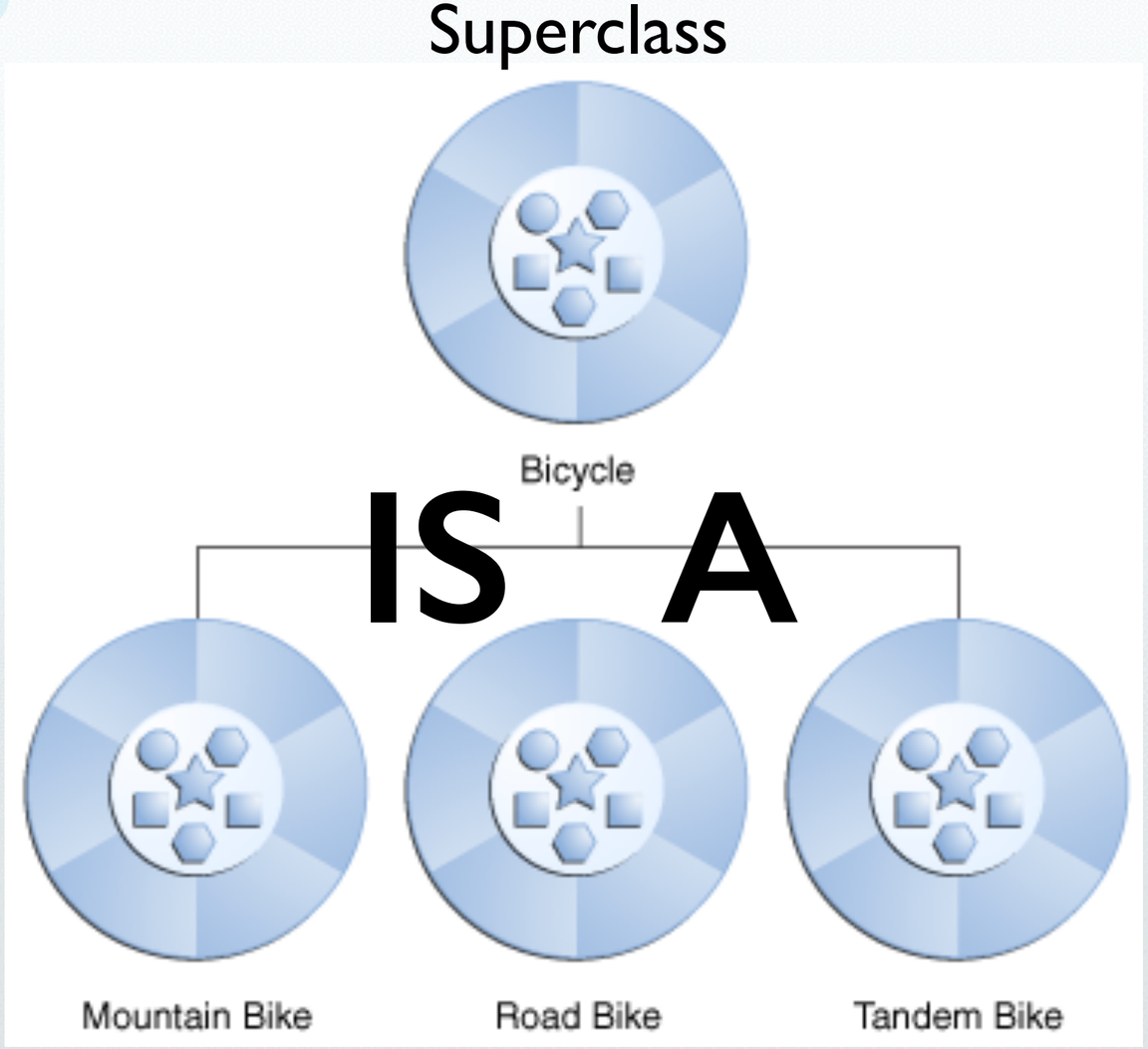
Which should I use and why?

Inheritance

1 Inheritance: what's that?

Inheritance is when an object or class is based on another object or class, using the same implementation or behavior. It is a mechanism for **code reuse and to allow independent extensions** of the original software via public classes and interfaces.

1 Inheritance: how's that?



- We often fail one of the basics assumptions about inheritance: subclass and superclass must have a "is a" relation, so MTB is always a Bike but a bike is not always a MTB.

- Regarding this, MTB is coupled to Bike. Think about that:

  · Changes in bike affects MTB
  · MTB depend on Bike behavior.
  · Both share multiple aspects
  · They have a strong relation

1 Inheritance: so, is inheritance evil?

- No, not really. But you MUST think about where to use is, and why.

You should be aware of using inheritance because…

- Inheritance is attractive, allow code reuse in an evident and fast way
- Inheritance is powerful and I can play with polymorphism.
- Inheritance allow override methods, I always can override if I need it
- Inheritance with abstract classes and base clases allow to develop software in an easy guided way

All those things are one main reasons that guide us to use inheritance in the wrong moment and guide us to a poor design

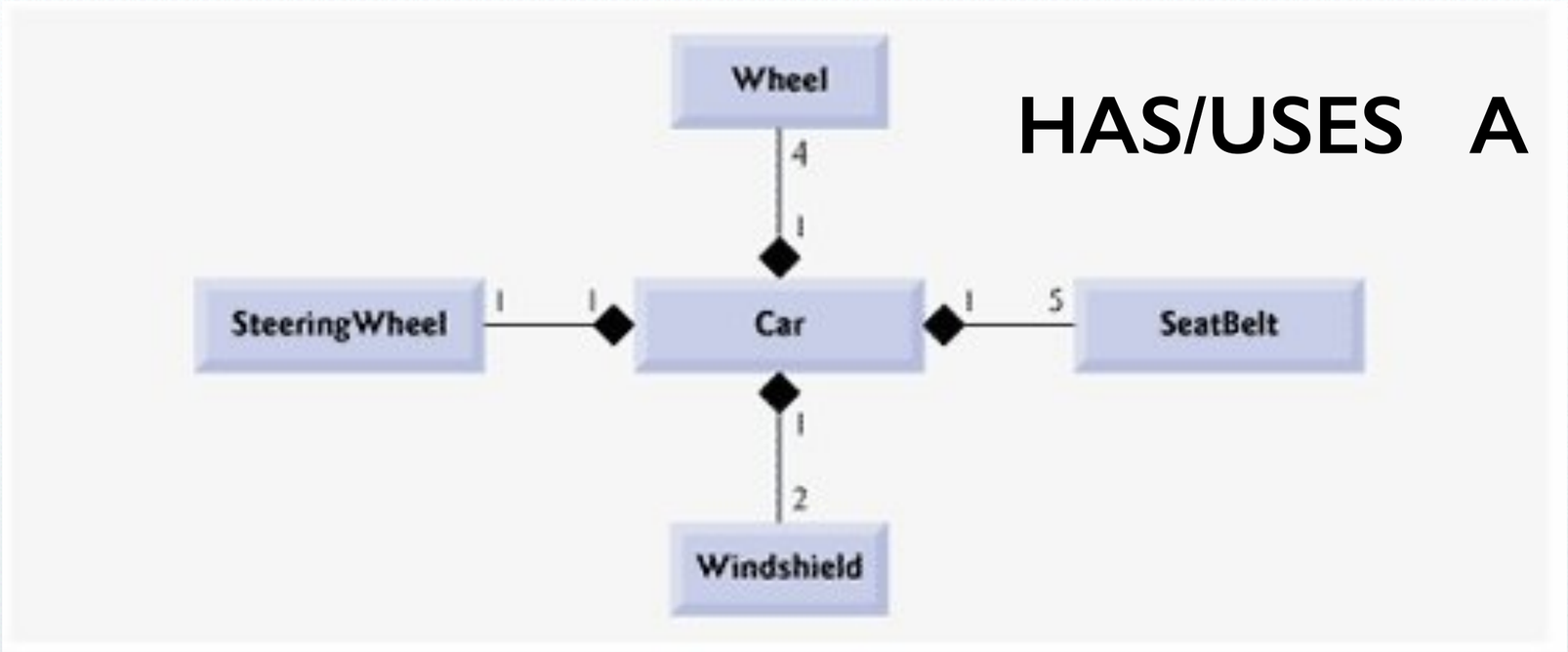
2

Composition

Composition: what's that?

Composition means to have an instance of one class that is **containing instances** of other classes that implement the **desired functionality**

2 Composition: how's that?



Composition makes use of his collaborators and delegates into them the tasks that was supposed to be done by using inheritance. In this way the resulting class is much more flexible at compile and runtime.

3

Comparing both

3 Compare yourself: Types of software

Type	Inheritance Driven Design	Composition Driven Design
Development "Init"	Faster	Slower
Design	Easy and poor	More complex
Side Effects	Many, and ease to get	Minimal
Change-friendly	Not really, the more you change the closer to spaghetti software you get	Easy to change
Development "one year later"	Many people made is own interpretation of inheritance	People have followed a similar composition strategy.
Testability	Difficult to maintain because of overrides and superclass tests	Easy to maintain and extend
Easy extension	Extension of software requires inheritance and changes in superclasses	You get extension by composing small pieces and developing new ones

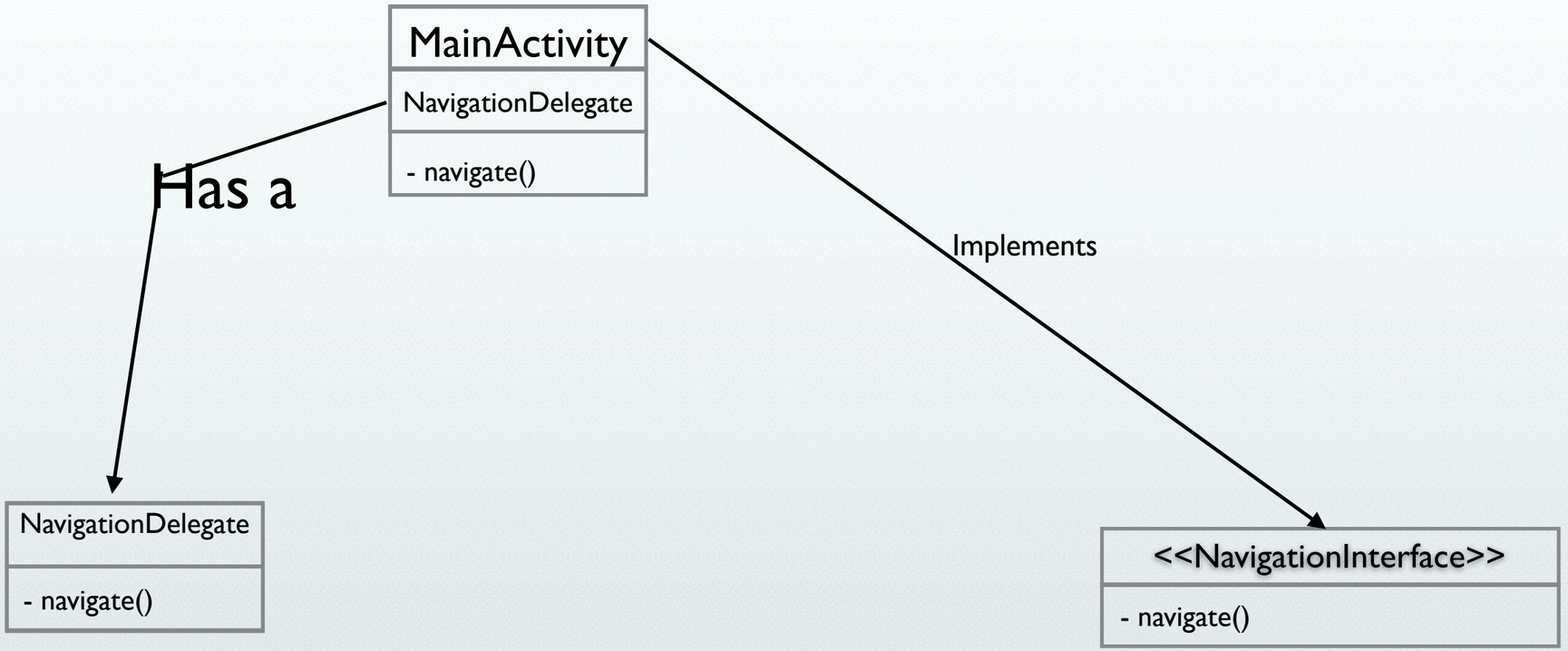
Inheritance lead us to software that promise to be robust and easy to change and maintain but empiric cases have shown us that this promise is broken once and again

3 More, interesting things…

- In class composition, there are 0 or more inner objects. In class inheritance, there is exactly one inner object.
- Different syntax: Inheritance uses the extends keyword. Composition, on the other hand, involves creating an object inside a class.
- As we will see later, polymorphism and dynamic binding does not apply to class composition.

So… What's about polymorphism? —> Interface to rescue

Interface rol on composition



4

Then… when to use inheritance

4 Inheritance should only be used when

1- Both classes are in the same logical domain

2- The subclass is a proper subtype of the superclass

3- The superclass's implementation is necessary or appropriate for the subclass

4- The enhancements made by the subclass are primarily additive.

Ex:

Higher-level domain modeling
Frameworks and framework extensions

4 Signals about inheritance bad usage

1 - Overrides start to grow without control and sense.

2 - “Is a” relation has been broken

3 - Extension of software requires constant changes of same base classes

4 - Hierarchy level is too high and you are starting to get loose.

If some of the previous signals appear in your software go ahead, redesign your code and get ready for favor this time composition over inheritance.

5

Common error cases

5 Common error cases

Trying to model behavior like business objects (bad code reuse)

We often take too complex decisions too early
(EX: getJson() as method in a BaseRequest class).

This kind of information is not clear at the first stages of software development. You are taking too important decisions that will tie you to non-flexible software.

Create Base components and methods that will are advocate to be unused or used with different aim of the original one. Abstract behavior is one of the most difficult tasks of Soft Eng.

Make too complex flows in clases that don't really need any lifecycle.

6

Conclusion

6 Conclusion

Inheritance is not bad, but regarding the type of software we usually develop, inheritance gives more problems than profit we can take from it.

Just think about each case carefully, maybe inheritance is the best. But if you are not sure about that, give a try to composition + interfaces.

7

Practice

7 The inherited activities party

Flow of work

- Create an activity that manages Dependency Injection (Mock)

- Create an activity extending the first one that fits the Design guidelines of the app (Mock)

- Create an activity that knows how to navigate between activities

- Create an activity that supports rotation and changes his UI

- Create an activity that supports only rotation. Are you having any problems?

- Create an activity that supports navigation and Dependency Injection. Problems again?

Now translate this into Composition (Try using dynamic composition. Setting behaviors at runtime)