Programming Language Design 2024

Modularisation; Object-oriented programming languages

Hans Hüttel

4 March 2024

Part I

About today

Learning goals – Modules

- To be able to understand and argue for the reasons for modularizing programs
- To be able to understand and explain the pros and cons of the main approaches to modularization at implementation level: text inclusion, linking and shared modules
- To be able to give an overview of the main ideas for supporting modularization at the language level, including the notions of signatures and headers

Learning goals – Object-oriented languages

- To be able to classify programming languages according to paradigms based on data flow, execution ordering and structuring, respectively
- To understand and explain object-oriented languages that use simple inheritance, multiple inheritance and prototypes, respectively, and understand and explain the pointer models underlying these three notions

Part II

Some problems for discussion

How is information hiding supported?

Some of the reasons for introducing modularity that are mentioned in the video content for this session are

- Abstract away implementation details from the rest of program.
- Reuse code.

Which *language constructs* support these reasons for introducing modularity?

Supporting information hiding

- Using methods and only methods to read from and write to fields
- Using local variables in methods
- Using access modifiers (private/public/protected)

Note that none of these ideas are particular to the object-oriented programming paradigm!

The central notion is that of abstract datatypes that require us to ”speak the language of the datatype” and not its implementation.

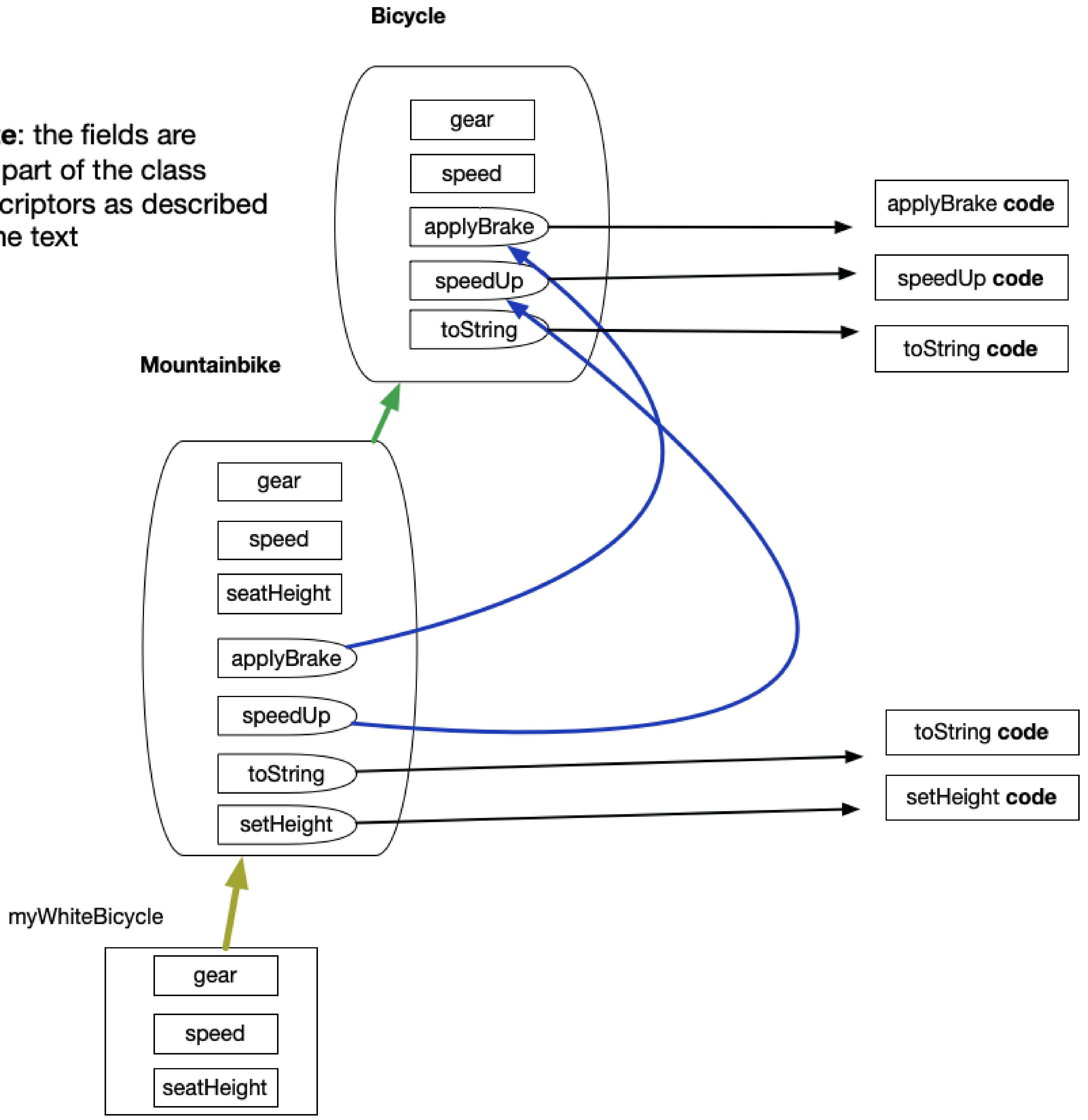
Inheritance – but how?

In the plan for this session available from Absalon I list two Java classes; I have left out the actual code in the method bodies. Draw a pointer model in the same way as was done in Part II of the podcast to show how the classes and the objects are related.

```java
class Bicycle {

// the Bicycle class has two fields

public int gear;

public int speed;

// the Bicycle class has one constructor

public Bicycle(int gear, int speed)

{  ... }

// the Bicycle class has three methods

public void applyBrake(int decrement)

{  ... }

public void speedUp(int increment)

{  ... }

// toString() method to print info of
Bicycle

public String toString()

{  ... }

}
```

```java
class MountainBike extends Bicycle {

// the MountainBike subclass adds one more
field

public int seatHeight;

// the MountainBike subclass has one
constructor

public MountainBike(int gear, int speed,
int startHeight)

{  ... }

// the MountainBike subclass adds one more
method

public void setHeight(int newValue)

{  ... }

// overriding toString() method of Bicycle
to print more info

@Override public String toString()

{  ... }

}
```

Note: the fields are *not* part of the class descriptors as described in the text



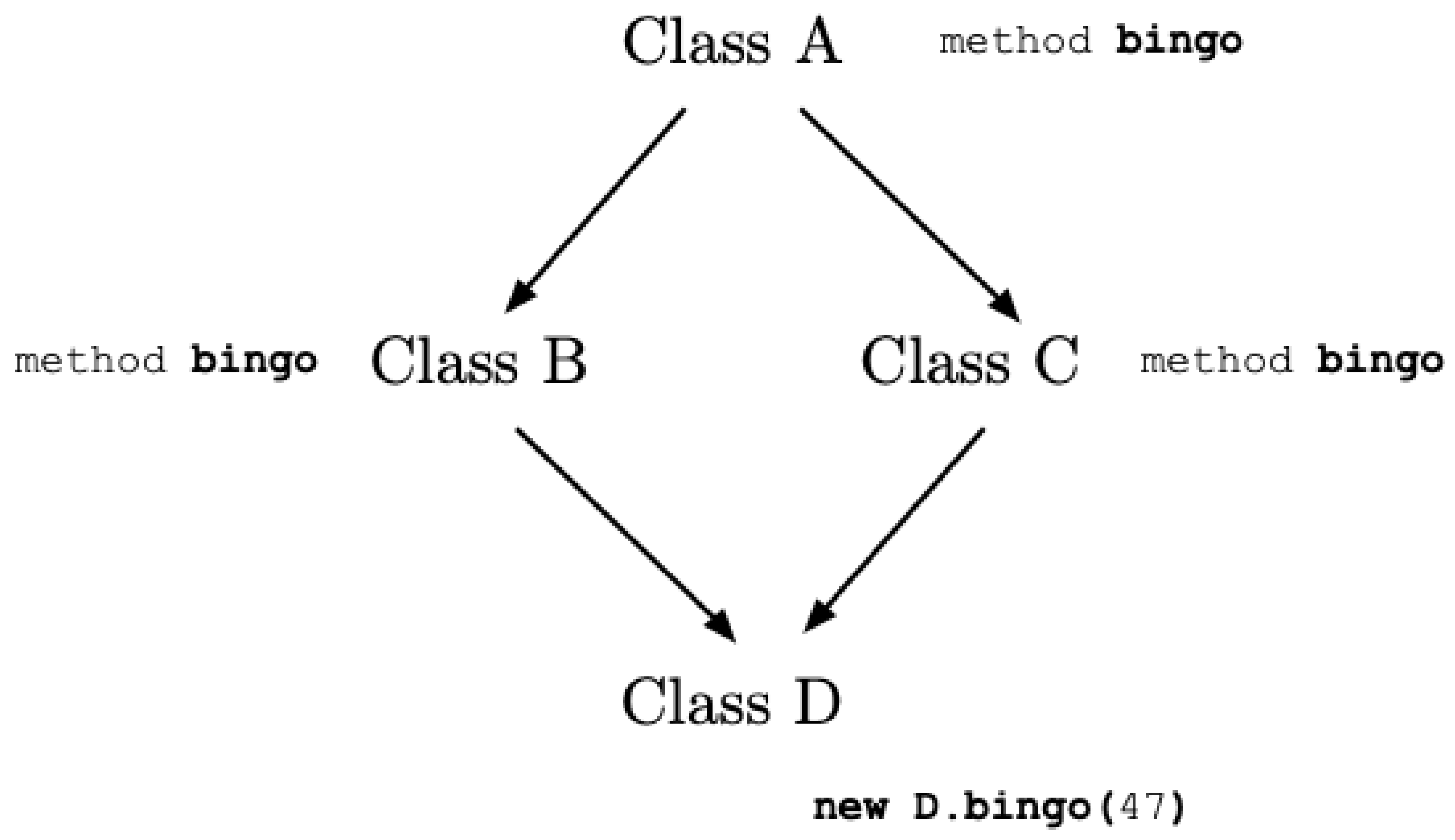
The diamond problem

In the text for today we run into the diamond problem and I claim that different languages have different approaches to dealing with it.

What was the diamond problem? How does your favourite object-oriented language with multiple inheritance deal with it?

The diamond problem

This is a problem that appears when we allow multiple inheritance.



If there is a `bingo` method in *A* that *B* and *C* both redefine, and *D* does not redefine it, which version of `bingo` should *D* inherit?

The diamond problem

The Wikipedia page about this (`https://en.wikipedia.org/wiki/Multiple_inheritance`) has a good overview of the many different approaches taken to resolve this.

The diamond problem

Some highlights:

- C++ requires that one must specify which class one inherits each feature from.
- Kotlin requires that one must specify which class one inherits a feature from in case of ambiguity.
- Go does not allow this – the compiler will complain.
- Java (as from version 8.0) requires that D must redefine the ambiguous feature. Older versions of Java do not have multiple inheritance at all.

Part III

About today (summing up)

An evaluation of the session

- What has gone well so far?
- What did not go so well?
- Is there a particular topic/problem that we should follow up on?