

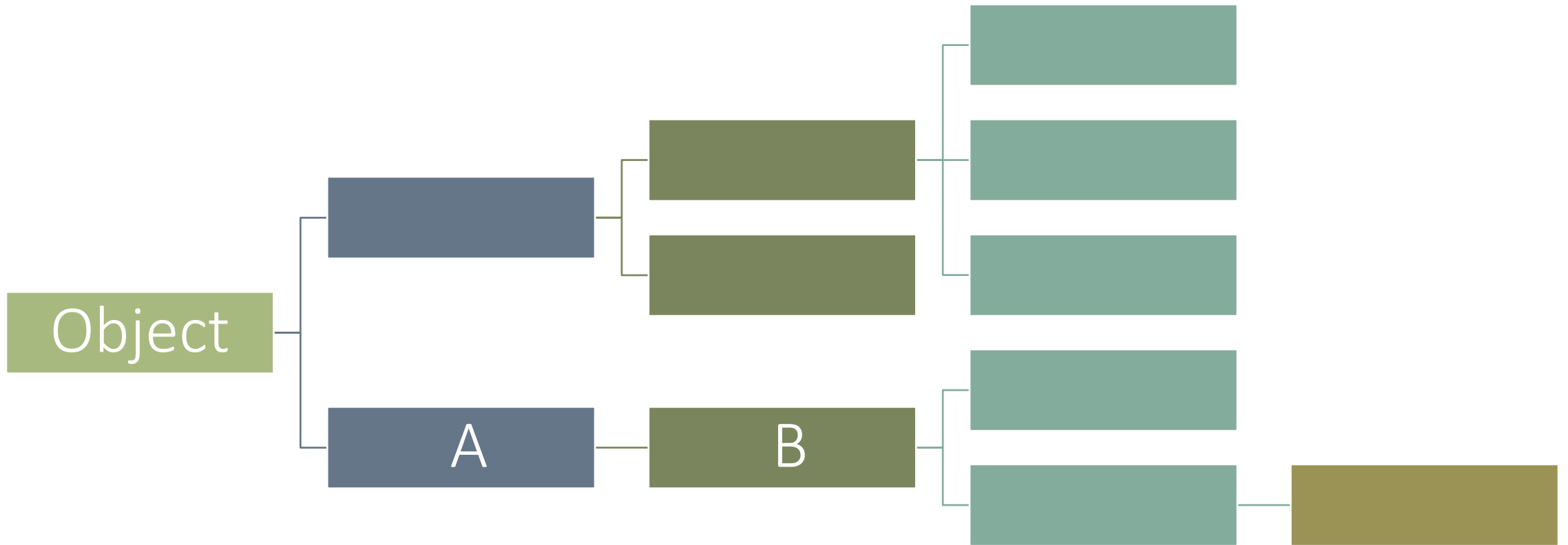
A row of colorful drinks in plastic cups with straws and fruit garnishes on a bar counter. The drinks include orange juice, red juice, and light green juice, all garnished with lemons, limes, and raspberries. The background shows a person in a grey t-shirt and a red and white striped shirt, and a green van.

Java Programming 2 Interfaces

Mary Ellen Foster

MaryEllen.Foster@glasgow.ac.uk

Inheritance in Java

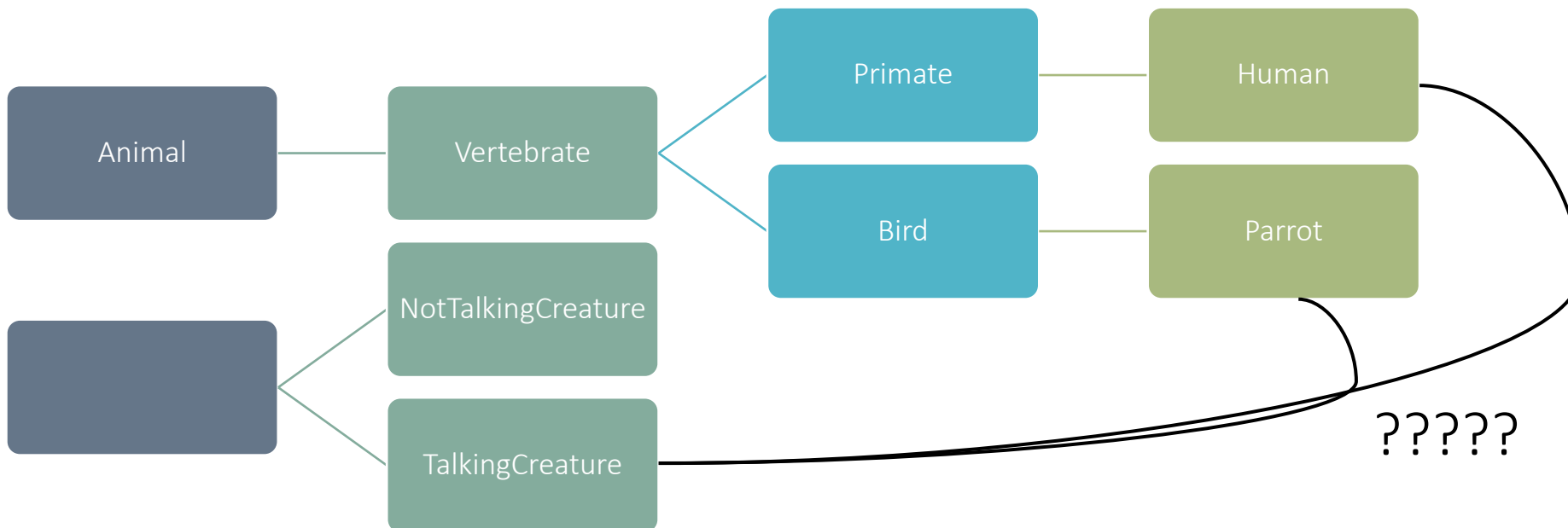


What about *multiple* inheritance?

Example:

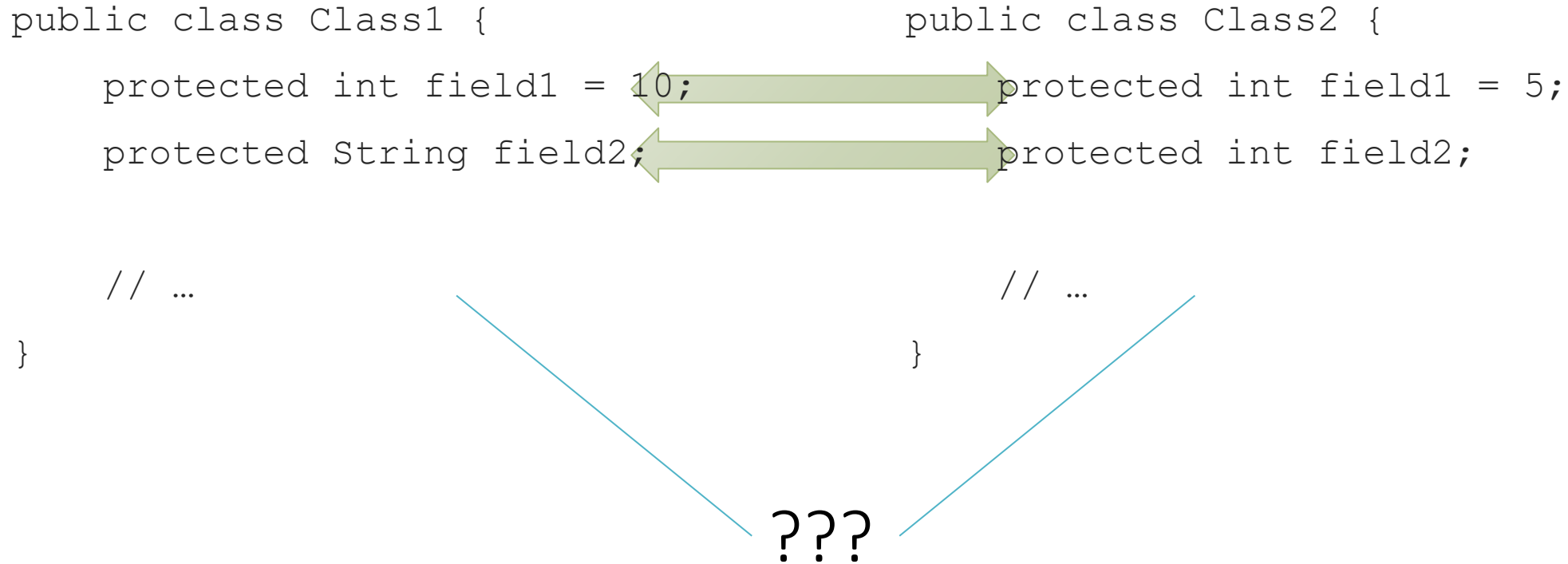
A Person is both a Primate and a TalkingCreature

A Parrot is both a Bird and a TalkingCreature



Multiple inheritance of state

Inheriting fields from multiple classes



Multiple inheritance of implementation

Inheriting method definitions from multiple classes

```
public class Class1 {  
    protected int field1;
```

```
    public void doSomething() {  
        this.field1 = 5;  
    }  
    // ...  
}
```

```
public class Class2 {  
    protected int field1;
```

```
    public void doSomething() {  
        this.field1 = 10;  
    }  
    // ...  
}
```

???

Multiple inheritance of type

Ability of a class to implement more than one interface (i.e., method signature only)

```
public interface Interface1 {  
    public void doSomethingElse();  
}
```

```
public interface Interface2 {  
    public void doSomething();  
}
```

```
public class MyClass implements Interface1, Interface2 {  
    public void doSomething() {  
        // ...  
    }  
  
    public void doSomethingElse() {  
        // ...  
    }  
}
```

Interface overview

Interfaces represent class relationships **outside the main inheritance hierarchy**

A class can implement any number of interfaces (including zero)

An interface specifies a public API

Implementation is irrelevant – method signatures only

It is a **contract** that all implementing classes must honour

Interfaces in Java

Similar to a class, but ...

- Declared with `interface` keyword

- All (non-static) methods are implicitly `public abstract`

- All fields are implicitly `public static final`

 - i.e., constants*

- Has no* instance-level fields or methods

 - Eliminates issues with multiple inheritance of state and implementation*

```
public interface TalkingCreature {  
    void speak(String s);  
}
```

```
public interface List {  
    int size();  
    boolean isEmpty();  
    boolean contains (Object o);  
    boolean remove (Object o);  
    void clear();  
    // ...  
}
```

* Except for default methods (introduced in Java 8) ... see slide #13 for details

Implementing an interface

Use `implements` keyword

Conventionally, comes after `extends`

Provide a definition for all methods declared in each interface

... or else declare class as `abstract`

All method implementations must be `public`

Classes can implement multiple interfaces (comma-separated)

```
public class Person
    extends Primate
    implements TalkingCreature {
    public void speak (String s) {
        // ...
    }
}

public class Parrot
    extends Bird
    implements TalkingCreature {
    public void speak (String s) {
        // ...
    }
}
```

Interface inheritance

Interfaces can `extend` other interfaces

They can even extend **multiple** other interfaces!

... Because they avoid issues of multiple inheritance

Comma-separated list of parent interfaces

Interfaces cannot extend classes, and vice versa

Conflicting methods

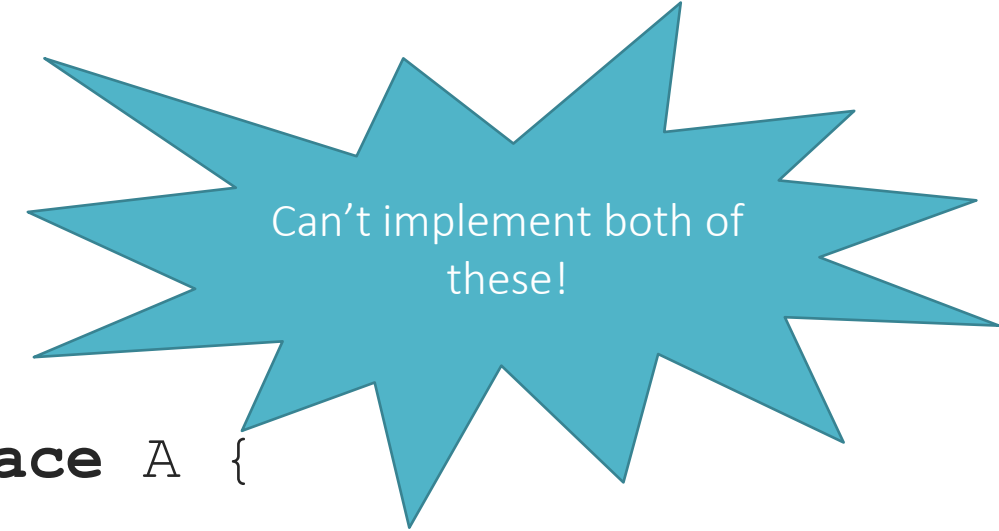
If two interfaces have methods with the same name and parameters, but **different return types**, you cannot

- Implement them both in a class

- Inherit from both in an interface

If the two methods have **identical signatures** then it will work

- (... but possibly be confusing)



```
interface A {  
    int doSomething();  
}
```

```
interface B {  
    String doSomething();  
}
```

Using an interface as a type

You can use an interface name anywhere you use any other data type name

- Variable declarations
- Method parameters
- etc

You cannot directly create an instance of an interface (through `new`)

```
TalkingCreature c = new Person();  
public void listen (TalkingCreature c)  
{  
    // ...  
}
```

Complication: default methods

Interface methods can be declared as **default**

- Allows an implementation of the method to be provided

Implementing classes can ...

- Not mention the method – default behaviour is inherited

- Redeclare the method – becomes abstract (like a normal interface method)

- Redefine the method – overriding behaviour as usual

Why was this added?

- To allow interfaces to be extended/updated without breaking all existing implementations of the interface

Interface vs. abstract class

INTERFACE

- Cannot be instantiated
- Has no constructor
- All methods are public
- Contain only constant fields
- Classes can implement **multiple** interfaces

ABSTRACT CLASS

- Cannot be instantiated
- Has a constructor
- Methods can have any access modifier
- Contain constant and “normal” fields
- Classes can have **at most one** parent class