Object Oriented Languages

We are now going to spend some more time looking at OO languages as they are important and have a wide variety of variants amongst themselves

Object Oriented Languages

We are now going to spend some more time looking at OO languages as they are important and have a wide variety of variants amongst themselves

It's a big family

Object Oriented Languages

We are now going to spend some more time looking at OO languages as they are important and have a wide variety of variants amongst themselves

It's a big family

Many people have the implicit assumption that if you know Java then you know all about OO

Object Oriented Languages

We are now going to spend some more time looking at OO languages as they are important and have a wide variety of variants amongst themselves

It’s a big family

Many people have the implicit assumption that if you know Java then you know all about OO

This is far from the truth: the Java way of doing OO is just one way of many

Object Oriented Languages

It is sometimes said that an OO language is typified by

> *“Abstraction, Encapsulation, Inheritance, Polymorphism”*

Object Oriented Languages

Abstraction: a high level view, where irrelevant details are hidden. Helps programming at a “higher” concept level. E.g., A `Dog` can `bark()` but you don’t need to know how it does that to use that code in your program

Object Oriented Languages

Abstraction: a high level view, where irrelevant details are hidden. Helps programming at a “higher” concept level. E.g., A `Dog` can `bark()` but you don’t need to know how it does that to use that code in your program

Encapsulation: structure and implementation kept hidden from the programmer behind a well-defined interface. Allows changing details of implementation at a lower level without affecting the higher levels. A way of enforcing abstraction

Object Oriented Languages

Inheritance: allowing properties to be shared in a hierarchy, thus avoiding re-implementation across related objects. Generally using classes to describe the hierarchy

Object Oriented Languages

Inheritance: allowing properties to be shared in a hierarchy, thus avoiding re-implementation across related objects. Generally using classes to describe the hierarchy

Polymorphism: treating different objects in different ways depending on their type/class. Allow the same “idea” to apply in different ways, e.g., a `Dog` and a `Cat` can both `eat()`, but in different ways

Object Oriented Languages

We shall see the several ways that this statement (AEIP) is wrong!

Object Oriented Languages

We shall see the several ways that this statement (AEIP) is wrong!

Or, at least, a very limited viewpoint

Object Oriented Languages

We shall see the several ways that this statement (AEIP) is wrong!

Or, at least, a very limited viewpoint

But for some people, they are *defining features* of what it means to be OO

Object Oriented Languages

We shall see the several ways that this statement (AEIP) is wrong!

Or, at least, a very limited viewpoint

But for some people, they are *defining features* of what it means to be OO

Thus excluding a wide range of very useful languages!

Object Oriented Languages

Similarly, many people think that OO is about *classes*

Object Oriented Languages

Similarly, many people think that OO is about *classes*

And so say the first step in OO design is to map out your class hierarchy

Object Oriented Languages

Similarly, many people think that OO is about *classes*

And so say the first step in OO design is to map out your class hierarchy

This is also misleading: OO is actually about *messaging objects*

Object Oriented Languages

Similarly, many people think that OO is about *classes*

And so say the first step in OO design is to map out your class hierarchy

This is also misleading: OO is actually about *messaging objects*

Classes are secondary, and sometimes not there at all!

Object Oriented Languages

Similarly, many people think that OO is about *classes*

And so say the first step in OO design is to map out your class hierarchy

This is also misleading: OO is actually about *messaging objects*

Classes are secondary, and sometimes not there at all!

And there are OO languages with classes, but no inheritance

Object Oriented Languages

> *OOP to me means only messaging, local retention and protection and hiding of state-process, and extreme late-binding of all things.*

Alan Kay (invented the term “object oriented”)

> *The notion of object oriented programming is completely misunderstood. It’s not about objects and classes, it’s all about messages.*

Alan Kay

Object Oriented Languages

From ISO/IEC 2382:2015 Information technology - Vocabulary

> *object-oriented:*
> *pertaining to a technique or a programming language that supports objects, classes, and inheritance*
>
> *Note 1 to entry: Some authorities list the following requirements for object-oriented programming: information hiding or encapsulation, data abstraction, message passing, polymorphism, dynamic binding, and inheritance.*

Object Oriented Languages

Exercise Another (better?) characterisation is

- Single Responsibility Principle
- Encapsulation
- Abstraction
- Minimal Coupling

Read about this

Object Oriented Languages

Exercise You will also find SOLID

- Single Responsibility Principle
- Open-Closed principle
- Liskov substitution principle
- Interface segregation principle
- dependency inversion principle

Read about this

Exercise For later. Come back and revisit SEAM and SOLID when we have gone though OO in detail

Object Oriented Languages

> *It was obvious to me 20-some years ago that OOP wasn't a panacea. That's the reason C++ supports several design and programming styles.*
>
> *In the first edition of "The C++ Programming Language," I didn't use the phrase "object-oriented programming" because I didn't want to feed the hype. One of the problems with OOP is exactly that unscrupulous people have hyped it as a panacea. Overselling something inevitably leads to disappointments.*

Bjarne Stroustrup, Feb 2000 (24 years ago!)

Object Oriented Languages

Language historians put the emergence of the idea of objects and classes in a purpose-designed language perhaps as far back as 1962 with Simula, a discrete event simulation language, and more definitely in 1967 with Simula 67

Object Oriented Languages

Language historians put the emergence of the idea of objects and classes in a purpose-designed language perhaps as far back as 1962 with Simula, a discrete event simulation language, and more definitely in 1967 with Simula 67

People had been exploring the ideas using Lisp, but this seems to be the first language designed to be OO

Object Oriented Languages

Language historians put the emergence of the idea of objects and classes in a purpose-designed language perhaps as far back as 1962 with Simula, a discrete event simulation language, and more definitely in 1967 with Simula 67

People had been exploring the ideas using Lisp, but this seems to be the first language designed to be OO

Simula looks like a mixture of Pascal and Java, and has been described as “Algol plus classes”

Object Oriented Languages

Simula has constructs like objects, classes, subclasses and methods that eventually followed through C++ directly into Java

Object Oriented Languages

Simula has constructs like objects, classes, subclasses and methods that eventually followed through C++ directly into Java

> *C++ is Simula in wolf's clothing*

Bjarne Stroustrup

Object Oriented Languages

Simula has constructs like objects, classes, subclasses and methods that eventually followed through C++ directly into Java

> *C++ is Simula in wolf's clothing*

Bjarne Stroustrup

However, it was with Smalltalk in 1972 that the OO concept really took off and influenced "modern" languages like C++ and Java

Object Oriented Languages

The original SmallTalk ideas of what defines OO were

- message passing
- isolation of objects
- polymorphism

Notice no mention of inheritance!

Object Oriented Languages

Feet

- Simula: ?

Object Oriented Languages

Feet

- Simula: ?
- Smalltalk: You send the message shoot to gun, with selectors bullet and myFoot. A window pops up saying Gunpowder doesNotUnderstand: spark. After several fruitless hours spent browsing the methods for Trigger, FiringPin and IdealGas, you take the easy way out and create ShotFoot, a subclass of Foot with an additional instance variable bulletHole

Object Oriented Languages

Reflection

Smalltalk was a very advanced version of OO: it introduced *metaclasses*, classes that determine the behaviour of other classes, thus enabling *reflection* in programs

Object Oriented Languages

Reflection

Smalltalk was a very advanced version of OO: it introduced *metaclasses*, classes that determine the behaviour of other classes, thus enabling *reflection* in programs

The concept of reflection, where a language can inspect and alter itself is dangerously close to the idea of self-modifying programs

Object Oriented Languages

Reflection

Smalltalk was a very advanced version of OO: it introduced *metaclasses*, classes that determine the behaviour of other classes, thus enabling *reflection* in programs

The concept of reflection, where a language can inspect and alter itself is dangerously close to the idea of self-modifying programs

Self-modifying programs are dangerous and hard to understand or control

Object Oriented Languages

Reflection

Smalltalk was a very advanced version of OO: it introduced *metaclasses*, classes that determine the behaviour of other classes, thus enabling *reflection* in programs

The concept of reflection, where a language can inspect and alter itself is dangerously close to the idea of self-modifying programs

Self-modifying programs are dangerous and hard to understand or control

But metaobject programming as a way to implement reflection puts a framework on this which makes it safe to use

Object Oriented Languages

Reflection

Smalltalk was a very advanced version of OO: it introduced *metaclasses*, classes that determine the behaviour of other classes, thus enabling *reflection* in programs

The concept of reflection, where a language can inspect and alter itself is dangerously close to the idea of self-modifying programs

Self-modifying programs are dangerous and hard to understand or control

But metaobject programming as a way to implement reflection puts a framework on this which makes it safe to use

But still very powerful

Object Oriented Languages

Reflection

A related idea is *reification*

Object Oriented Languages

Reflection

A related idea is *reification*

This is where a system can look at its own structure or behaviour

Object Oriented Languages

Reflection

A related idea is *reification*

This is where a system can look at its own structure or behaviour

Sometimes called *introspection*, and is often seen as making certain aspects first-class objects, for example first-class classes

Object Oriented Languages

Reflection

A related idea is *reification*

This is where a system can look at its own structure or behaviour

Sometimes called *introspection*, and is often seen as making certain aspects first-class objects, for example first-class classes

Debuggers can be viewed as reification; as can class-loaders in Java; and `eval` in Lisp

Object Oriented Languages

Reflection

A related idea is *reification*

This is where a system can look at its own structure or behaviour

Sometimes called *introspection*, and is often seen as making certain aspects first-class objects, for example first-class classes

Debuggers can be viewed as reification; as can class-loaders in Java; and `eval` in Lisp

Reflection is where the system can go in and modify things, too

Object Oriented Languages

Reflection

A related idea is *reification*

This is where a system can look at its own structure or behaviour

Sometimes called *introspection*, and is often seen as making certain aspects first-class objects, for example first-class classes

Debuggers can be viewed as reification; as can class-loaders in Java; and `eval` in Lisp

Reflection is where the system can go in and modify things, too

Time permitting, we will look in more depth at these ideas later

Object Oriented Languages

In Smalltalk, everything is an object, including control structures like `if`

Object Oriented Languages

In Smalltalk, everything is an object, including control structures like `if`

And everything is mediated by messages sent between objects (i.e., methods)

Object Oriented Languages

In Smalltalk, everything is an object, including control structures like `if`

And everything is mediated by messages sent between objects (i.e., methods)

Even addition is a message: `2 + 3` is the syntax that sends the message + with argument 3 to the object 2

Object Oriented Languages

In Smalltalk, everything is an object, including control structures like `if`

And everything is mediated by messages sent between objects (i.e., methods)

Even addition is a message: `2 + 3` is the syntax that sends the message + with argument 3 to the object 2

There is no artificial separation of primitive objects from other objects like in Java

Object Oriented Languages

In Smalltalk, everything is an object, including control structures like `if`

And everything is mediated by messages sent between objects (i.e., methods)

Even addition is a message: `2 + 3` is the syntax that sends the message + with argument 3 to the object 2

There is no artificial separation of primitive objects from other objects like in Java

This is more like `2.plus(3)` in Java-like syntax

Object Oriented Languages

Smalltalk prompted a lot of research into OO in the 70s and 80s

Object Oriented Languages

Smalltalk prompted a lot of research into OO in the 70s and 80s

And many different styles of OO were proposed including features called *prototyping* and *delegation*, and then Lisp-based languages featuring multiple inheritance and metaobject protocols

Object Oriented Languages

Smalltalk prompted a lot of research into OO in the 70s and 80s

And many different styles of OO were proposed including features called *prototyping* and *delegation*, and then Lisp-based languages featuring multiple inheritance and metaobject protocols

We shall be looking at these in turn, but we shall start with the most familiar kind of OO: that typified by having classes arranged in a hierarchy

Object Oriented Languages

Classes

Classes are things that gather together descriptions of the *structure* (how and which values are stored in the object, perhaps in memory, perhaps elsewhere) and the *behaviour* (the methods) of certain objects

Object Oriented Languages

Classes

Classes are things that gather together descriptions of the *structure* (how and which values are stored in the object, perhaps in memory, perhaps elsewhere) and the *behaviour* (the methods) of certain objects

For code reuse, many languages allow classes to have subclasses that inherit and extend the structure and/or behaviour of the parent class

Object Oriented Languages

Class Hierarchy

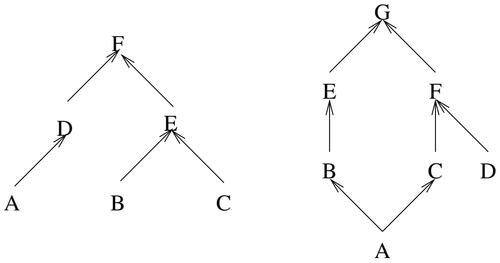
The *class hierarchy* is the relationship between classes

Object Oriented Languages

Class Hierarchy

The *class hierarchy* is the relationship between classes

This can be in a *tree*, where a class inherits from a single parent class (*single inheritance*); or a *directed acyclic graph* (DAG) when classes can inherit from more than one parent (*multiple inheritance*)



A Tree and a DAG

Object Oriented Languages

Class Hierarchy

Both trees and DAGs have an important property: no loops

Object Oriented Languages

Class Hierarchy

Both trees and DAGs have an important property: no loops

A loop would entail a class inheriting (possibly indirectly) from itself

Object Oriented Languages

Class Hierarchy

Both trees and DAGs have an important property: no loops

A loop would entail a class inheriting (possibly indirectly) from itself

Thus we do not allow loops in the class hierarchy

Object Oriented Languages

Class Hierarchy

In some languages, e.g., Lisp and Smalltalk, classes are first-class, being things that can be created and manipulated in the program

Object Oriented Languages

Class Hierarchy

In some languages, e.g., Lisp and Smalltalk, classes are first-class, being things that can be created and manipulated in the program

Classes are objects!

Object Oriented Languages

Class Hierarchy

In some languages, e.g., Lisp and Smalltalk, classes are first-class, being things that can be created and manipulated in the program

Classes are objects!

So they are actually instances of other classes

Object Oriented Languages

Class Hierarchy

In some languages, e.g., Lisp and Smalltalk, classes are first-class, being things that can be created and manipulated in the program

Classes are objects!

So they are actually instances of other classes

In languages, e.g., C++ and Java, classes are part of the program design but not first-class objects in the system

Object Oriented Languages

Class Hierarchy

In some languages, e.g., Lisp and Smalltalk, classes are first-class, being things that can be created and manipulated in the program

Classes are objects!

So they are actually instances of other classes

In languages, e.g., C++ and Java, classes are part of the program design but not first-class objects in the system

Exercise But (later) look up `java.lang.reflect`

Object Oriented Languages

Class Hierarchy

In some languages, e.g., Lisp and Smalltalk, classes are first-class, being things that can be created and manipulated in the program

Classes are objects!

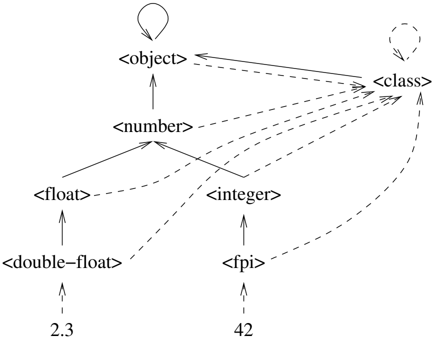
So they are actually instances of other classes

In languages, e.g., C++ and Java, classes are part of the program design but not first-class objects in the system

Exercise But (later) look up `java.lang.reflect`

A language will have a default hierarchy of those classes that come with the language

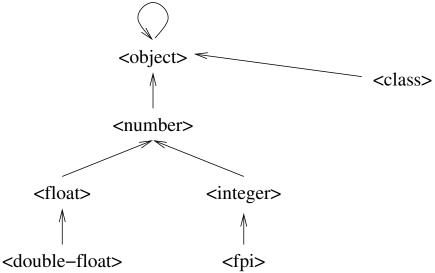
Object Oriented Languages



A Small Part of the EuLisp Class Hierarchy (simplified)

There are *two* hierarchies in this diagram

Object Oriented Languages

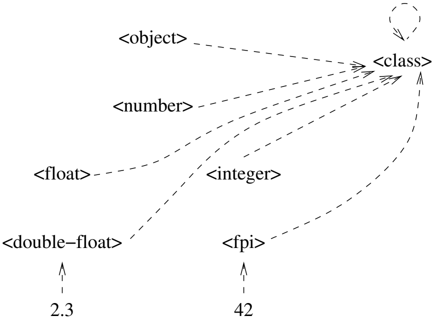


Inheritance hierarchy

Solid arrow is *inherits from*/*subclass*/*extends*/ *refines*/*specialises*/*subset*

This is the normal “class inheritance diagram”

Object Oriented Languages



Instance hierarchy

Dotted arrow is *instance of*/*member of*/*is a*;

Object Oriented Languages

Every object is an *instance* of a class (dotted arrow); sometimes called a *member* of that class.

Object Oriented Languages

Every object is an *instance* of a class (dotted arrow); sometimes called a *member* of that class.

Exercise Think about the relationship between set theory and classes and objects: members and instances; subsets and subclasses

Object Oriented Languages

So, for example, the integer 42 is an instance of the class `<fpi>`

Object Oriented Languages

So, for example, the integer 42 is an instance of the class `<fpi>`

E.g., the class `<fpi>` is an instance of the class `<class>`

Object Oriented Languages

So, for example, the integer 42 is an instance of the class `<fpi>`

E.g., the class `<fpi>` is an instance of the class `<class>`

A *subclass* will *inherit* (solid arrow) from its parent *superclass* (or superclass*es*)

Object Oriented Languages

So, for example, the integer 42 is an instance of the class `<fpi>`

E.g., the class `<fpi>` is an instance of the class `<class>`

A *subclass* will *inherit* (solid arrow) from its parent *superclass* (or superclass*es*)

It inherits both *structure*/*attributes* (how values in the instances are stored); and *behaviour* (the methods)

Object Oriented Languages

So, for example, the integer 42 is an instance of the class `<fpi>`

E.g., the class `<fpi>` is an instance of the class `<class>`

A *subclass* will *inherit* (solid arrow) from its parent *superclass* (or superclass*es*)

It inherits both *structure*/*attributes* (how values in the instances are stored); and *behaviour* (the methods)

Of course, it may override or add to either: generally you add or override methods, but just add to attributes

Object Oriented Languages

E.g., `<fpi>` inherits from `<integer>`

And `<class>` inherits from `<object>`

Object Oriented Languages

E.g., `<fpi>` inherits from `<integer>`

And `<class>` inherits from `<object>`

`<object>` inherits from itself

Object Oriented Languages

E.g., `<fpi>` inherits from `<integer>`

And `<class>` inherits from `<object>`

`<object>` inherits from itself

This is safe to do, as `<object>` has no structure or behaviour

Object Oriented Languages

E.g., `<fpi>` inherits from `<integer>`

And `<class>` inherits from `<object>`

`<object>` inherits from itself

This is safe to do, as `<object>` has no structure or behaviour

And the class `<object>` is an instance of the class `<class>`

Object Oriented Languages

E.g., `<fpi>` inherits from `<integer>`

And `<class>` inherits from `<object>`

`<object>` inherits from itself

This is safe to do, as `<object>` has no structure or behaviour

And the class `<object>` is an instance of the class `<class>`

Of course, the class `<class>` is an instance of itself

Object Oriented Languages

So there are two kinds of relationships between objects: instance and inherits

Object Oriented Languages

So there are two kinds of relationships between objects: instance and inherits

And two kinds of object: classes and non-classes

Object Oriented Languages

So there are two kinds of relationships between objects: instance and inherits

And two kinds of object: classes and non-classes

We can make instances of classes, but not of non-classes

Object Oriented Languages

So there are two kinds of relationships between objects: instance and inherits

And two kinds of object: classes and non-classes

We can make instances of classes, but not of non-classes

Other kinds of OO dispense with one or both of these relationships

Object Oriented Languages

So there are two kinds of relationships between objects: instance and inherits

And two kinds of object: classes and non-classes

We can make instances of classes, but not of non-classes

Other kinds of OO dispense with one or both of these relationships

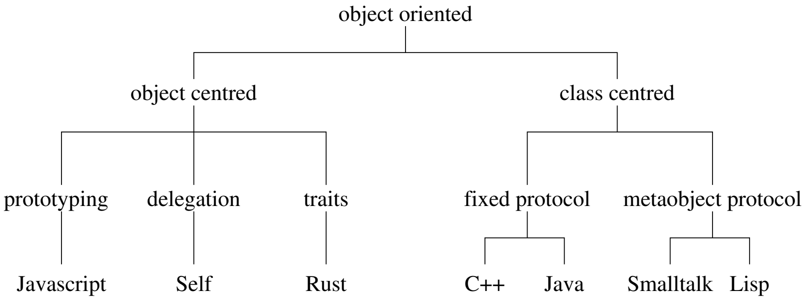
Or one of these kinds of object: the classes

Object Oriented Languages

Exercise For Java, C++, Common Lisp, EuLisp and any others determine their initial class hierarchy

Object Oriented Languages

Kinds of OO Language

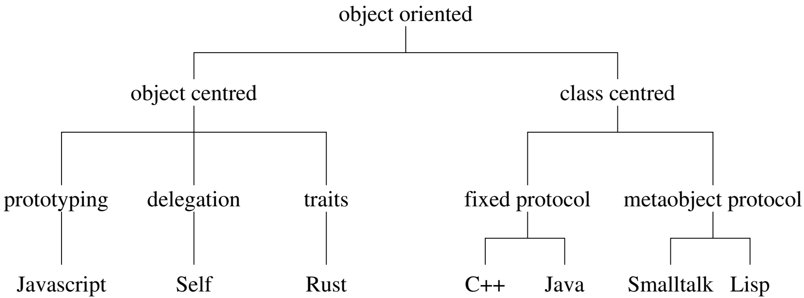


A classification of OO languages

Note: non-exclusive properties

Object Oriented Languages

Kinds of OO Language



object oriented

object centred

class centred

prototyping

delegation

traits

fixed protocol

metaobject protocol

Javascript

Self

Rust

C++

Java

Smalltalk

Lisp

A classification of OO languages

Note: non-exclusive properties

Exercise In this picture, determine which are instance links and which are inheritance links!

Object Oriented Languages

There is a wide variety of things that like to be called OO

Object Oriented Languages

There is a wide variety of things that like to be called OO

The basic idea they all share is the use of encapsulation of state within an object

Object Oriented Languages

There is a wide variety of things that like to be called OO

The basic idea they all share is the use of encapsulation of state within an object

While things like classes and inheritance are extras

Object Oriented Languages

There is a wide variety of things that like to be called OO

The basic idea they all share is the use of encapsulation of state within an object

While things like classes and inheritance are extras

Of course, these variants came about through lots of research and experimentation and have varying levels of success

Object Oriented Languages

There is a wide variety of things that like to be called OO

The basic idea they all share is the use of encapsulation of state within an object

While things like classes and inheritance are extras

Of course, these variants came about through lots of research and experimentation and have varying levels of success

As always, it's not a case of what is *better*, more what is *better for the application in hand*