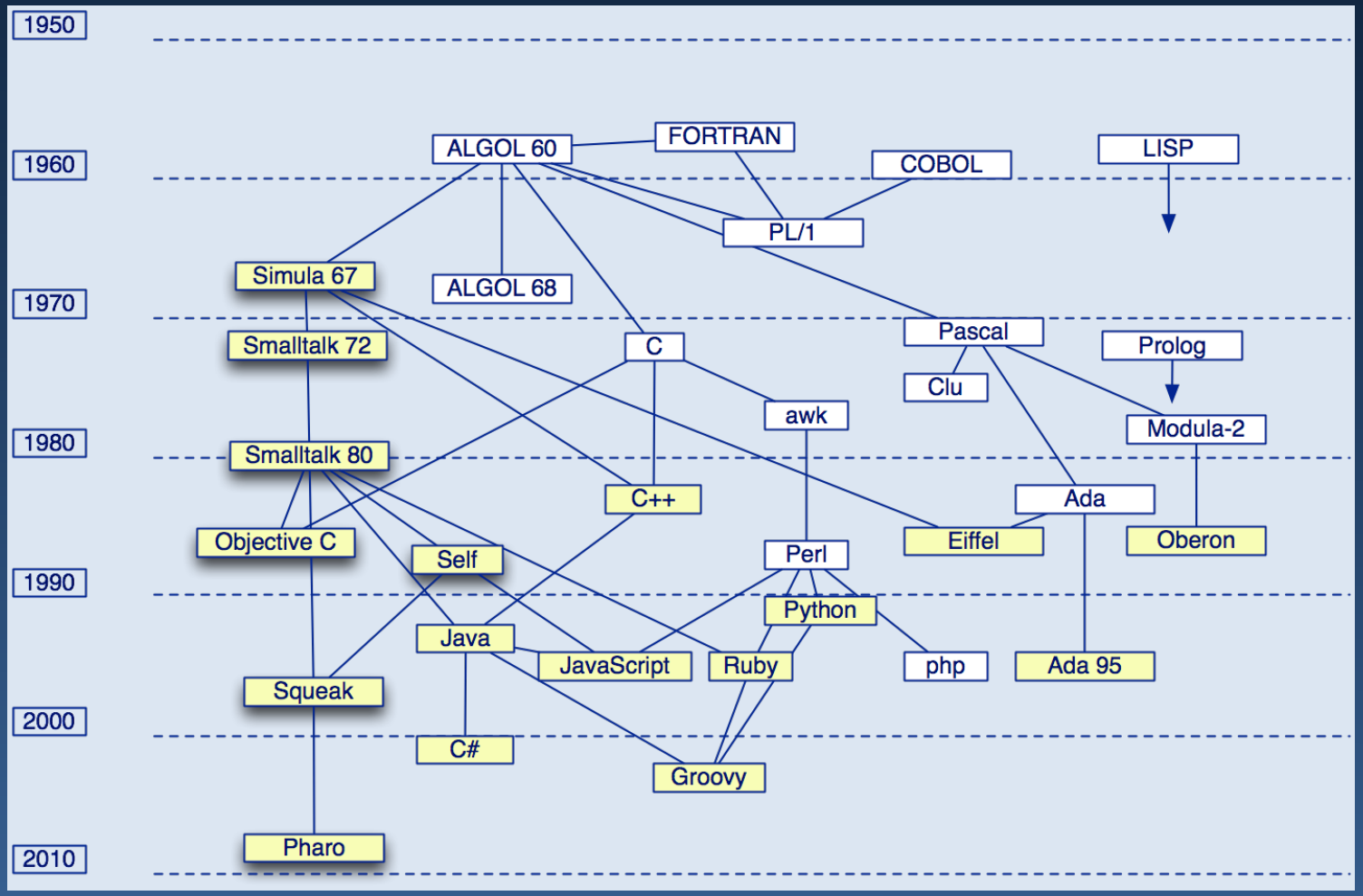
CS2510
MODERN PROGRAMMING LANGUAGES

Object-Oriented Programming 2

Prof. Peter Edwards
p.edwards@abdn.ac.uk

A Brief History of OOP



Simula

- Simula - developed in the 1960s at the Norwegian Computing Center in Oslo, by **Ole-Johan Dahl** and **Kristen Nygaard**.

- Syntactically, it is a superset of *Algol 60*.

Ole-Johan Dahl and Kristen Nygaard

- Simula 67 introduced **objects**, **classes**, **subclasses**, **virtual methods**, **coroutines**, **discrete event simulation**, and featured **garbage collection**.

- Considered to be the first **object-oriented** programming language.

Simula

- *Simula* was designed for performing simulations.
  - Simulation systems are modelled by a series of **state changes** that occur in **parallel** through the **interaction** of the **elements** of the system as they compete for restricted system resources.
- Each type of element of the system is composed of its internal state, as well as a set of procedures that define its own behaviour.
- *Simula* = Algol 60 + ***class*** concept
- Elements interact with one another as a system by calling some of the other elements' procedures.
- This is the baseline of what we now call *object-oriented programming*.

Smalltalk

- Smalltalk is an object-oriented, dynamically typed, reflective programming language.

- It was designed and created in part for educational use at the Learning Research Group of Xerox PARC by **Alan Kay**, **Dan Ingalls**, **Adele Goldberg**, Ted Kaehler, Scott Wallace, and others during the 1970s.

Dan Ingalls

Adele Goldberg

Alan Kay

Smalltalk

- ***Everything is an Object!***
  - Numbers, files, editors, compilers, points, tools, Booleans, messages …
- Everything happens by ***sending messages***
- Every object is an instance of one class.
- A class is also an object
  - Defines the structure and the behaviour of its instances.
- All objects are allocated from the heap.
  - Deallocation is implicit.
- Dynamic binding
  - Method being called upon an object is looked up by name at runtime.
  - Search the object itself, then its superclass, and then up through inheritance hierarchy to **`Object`** class.

Smalltalk

- Inheritance
  - A Smalltalk subclass inherits all of the instance variables, instance methods, and class methods of its superclass.
  - All subclasses are subtypes (nothing can be hidden)
  - No multiple inheritance.
  - *Overriding* is permitted.
- Polymorphism
  - Method overloading
  - Smalltalk is a *dynamically* typed language, so it exhibits *subtype polymorphism* by default.

Messages & Methods

- *Message* — which action to perform

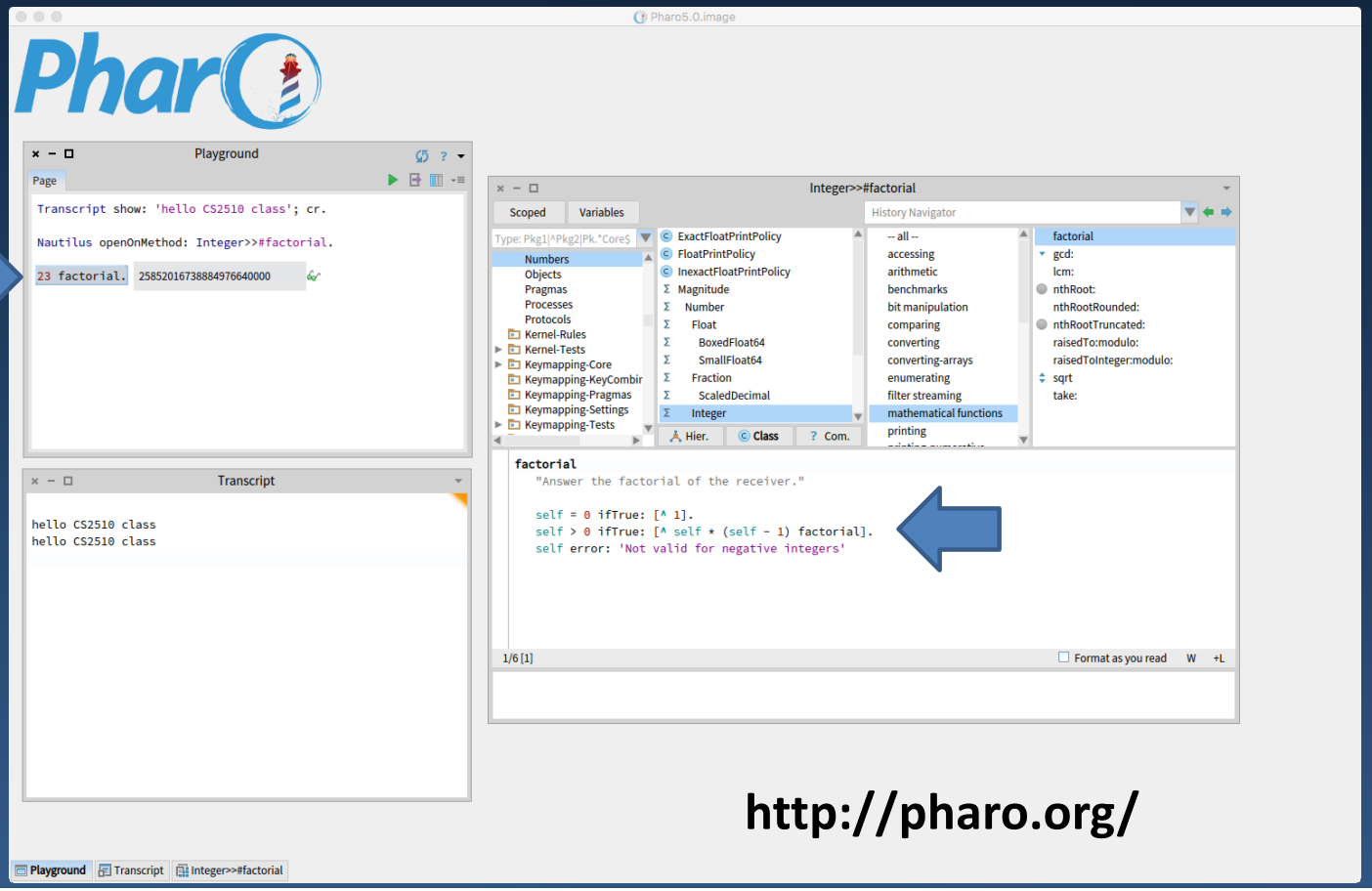
```
aWorkstation accept: aPacket
aMonster eat: aCookie
```

- *Method* — how to carry out the action

```
accept: aPacket
   (aPacket isAddressedTo: self)
      ifTrue:[
         Transcript show:
            'A packet is accepted by the Workstation ',
         self name asString ]
      ifFalse: [super accept: aPacket]
```

Smalltalk

Here we see the factorial message being sent to the object “23”



Remember - In Smalltalk EVERYTHING is an object.

Here is the implementation of the factorial method.

http://pharo.org/

UNIVERSITY OF ABERDEEN

Smalltalk

- Smalltalk-80 is a totally **reflective system**, implemented in Smalltalk-80 itself.
- Provides both **structural** and **computational** reflection.
- ***Structural reflection***
  - Classes/methods that define the system are themselves objects and fully part of the system that they help define.
  - The Smalltalk compiler compiles textual source code into method objects, typically instances of the **`CompiledMethod`** class.
  - System is extended by running Smalltalk-80 code that creates or defines classes and methods.

Smalltalk

- ***Computational reflection***
  - The ability to observe the computational state of the system.
  - The current activation of a method is accessible as an object named via a keyword: **`thisContext`**
  - By sending messages to **`thisContext`** a method activation can ask questions like: ”Who sent this message to me?"

Design Issues for OOP Languages

- The Exclusivity of Objects
- Are Subclasses Subtypes?
- Single & Multiple Inheritance
- Object Allocation & Deallocation
- Dynamic & Static Binding
- Nested Classes
- Initialization of Objects

The Exclusivity of Objects

- Everything is an Object!
  - Advantage: elegance and purity
  - Disadvantage: slow operations on simple objects
- Add objects to a complete typing system
  - Advantage: fast operations on simple objects
  - Disadvantage: results in a confusing type system (two kinds of entities)
- Include an imperative-style typing system for primitives but make everything else objects
  - Advantage: fast operations on simple objects and a relatively small typing system
  - Disadvantage: still some confusion because of the two type systems.

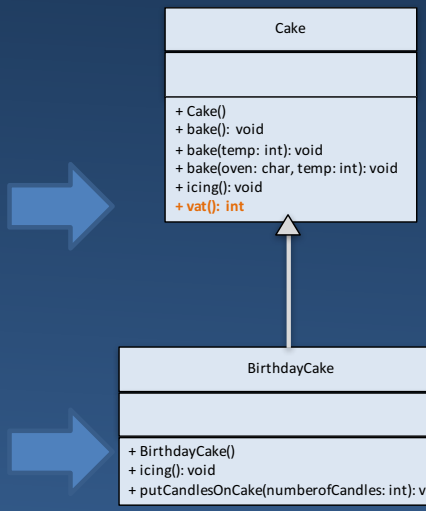
Are Subclasses Subtypes?

- Does an “is-a” relationship hold between a parent class object and an object of the subclass?
  - If a derived class *is-a* parent class, then objects of the derived class should behave the same way as the parent class object.
- A derived class is a subtype if it has an *is-a* relationship with its parent class
  - Subclass can only add variables and methods and override inherited methods in “compatible” ways.

Single vs Multiple Inheritance

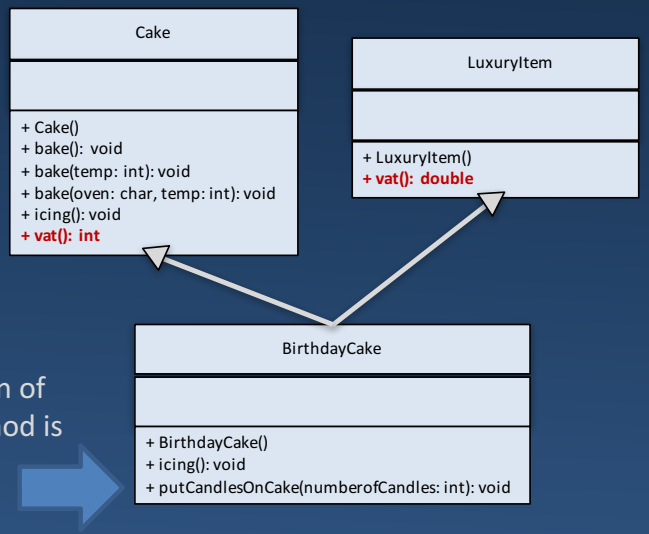
- Multiple inheritance allows a new class to inherit from two or more classes.
- Consider this example:

This is our **Cake** class from earlier; a new method `vat()` has been added - returns an int.



Subclass inherits 3 `bake()` methods, overrides `icing()` method, implements `putCandlesOnCake(int)` and **inherits `vat()` method**.

Subclass now inherits from two base (parent) classes - multiple inheritance.



Which version of `vat()` method is inherited??

Multiple Inheritance

- Disadvantages of multiple inheritance:
  - Language and implementation complexity (in part due to name collisions)
  - Potential inefficiency - dynamic binding costs more with multiple inheritance (but not much)
- Advantage:
  - Sometimes it is quite convenient and valuable.
- Supported in certain OO languages:
  - C++, CLOS, Eiffel …
  - Java doesn't permit multiple inheritance via classes, but does permit via *interfaces* (more on this later!)

Allocation & DeAllocation of Objects

- From where are objects allocated?
  - Allocated from the run-time stack?
  - Explicitly create on the heap (via `new`)?
- If they are all heap-dynamic, references can be uniform through a pointer or reference variable.
  - Simplifies assignment - dereferencing can be implicit.
- Is deallocation explicit (via `delete`) or implicit?

Dynamic & Static Binding

- Should all binding of messages to methods be dynamic?
  - If none are, you lose the advantages of dynamic binding.
  - If all are, it is inefficient
- Maybe the design should allow the user to specify?

Nested Classes

- If a new class is needed by only one class, there is no reason to define so it can be seen by other classes.
  - Can the new class be nested inside the class that uses it?
  - In some cases, the new class is nested inside a subprogram rather than directly in another class.
- Other issues:
  - Which facilities of the nesting class should be visible to the nested class and vice versa?

Initialization of Objects

- Are objects initialized to values when they are created?
  - Implicit or explicit initialization
- How are parent class members initialized when a subclass object is created?