CS4125
SYSTEMS ANALYSIS
SPRING SEMESTER 2010-2011

J.J. Collins
Dept of CSIS
University of Limerick

Lecture D (W2L2)

The Object-Oriented (OO) paradigm – Part 2.

Labs and Tutorials .... REVISED

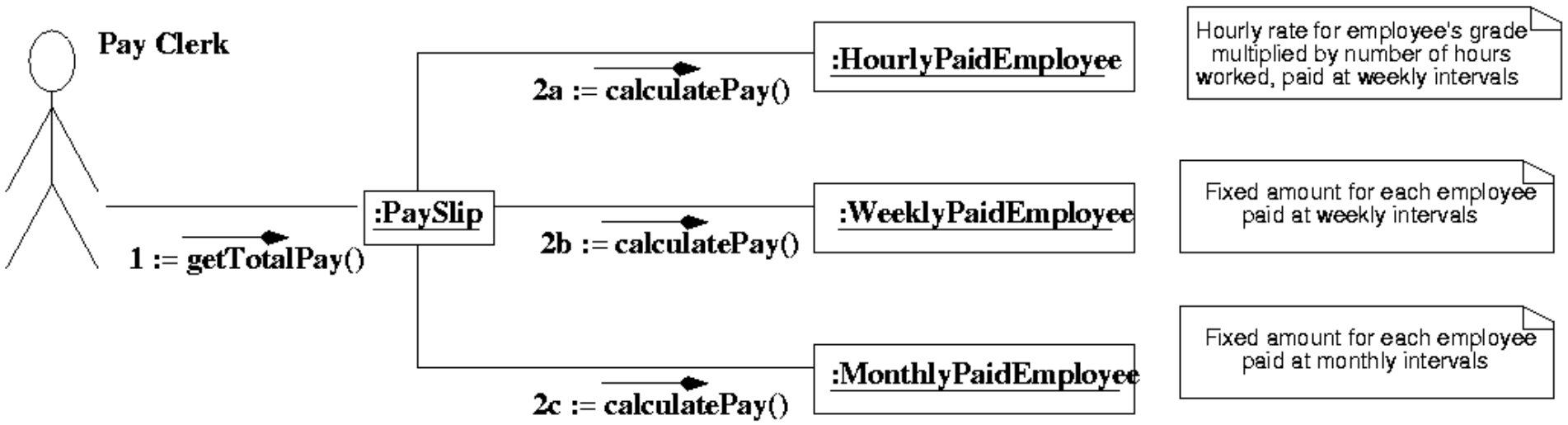
- Tutorials:
  - Tuesday 13:00-14:00 in SG19 - 2nd years
  - Tuesday 16:00-17:00 in S117 - 3rd + 4th years
  - Wednesday 16:00-17:00 in CSG25 – 3rd + 4th years
- Labs:
  - Wednesday 09:00-10:00 in CS244 – 3rd + 4th years
  - Wednesday 13:00-14:00 in CS244 - 3rd + 4th years
  - Wednesday 15:00-16:00 in CS244 – 2nd years
    - Assuming that project teams are year-oriented
- Starting Week 3
- TAs: Howell Jordan (tutorials) and Anne Meade (labs)
  - Both are LGSSE PhD postgrads.
  - Both are very competent in the area of software design.

Polymorphism

- Polymorphism: “one interface, multiple methods”.
- In OO languages, polymorphism supported at compile time and run time.
- Compile time: function and operator overloading.
- Runtime polymorphism: known as overriding, late binding or dynamic binding.
- Same message being passed to different objects, each object responding in a different way.

Polymorphism

- Illustrated using a Communication diagram



Polymorphism

- Runtime polymorphism dependent on dynamic (late) binding.
- Dynamic (late) binding: determining which code to execute at run time.
- Look at Java coding fragments from Mughal and Rasmussen at end of handout.
- C++:
  - Use the keyword virtual
  - A virtual function is a function that is declared as virtual in a base class and redefined in one or more derived classes.
  - Virtual functions are special because when one is accessed using a pointer of the base class to an object of a derived class, C++ determines which function to call at run-time, based on the type of object pointed to.

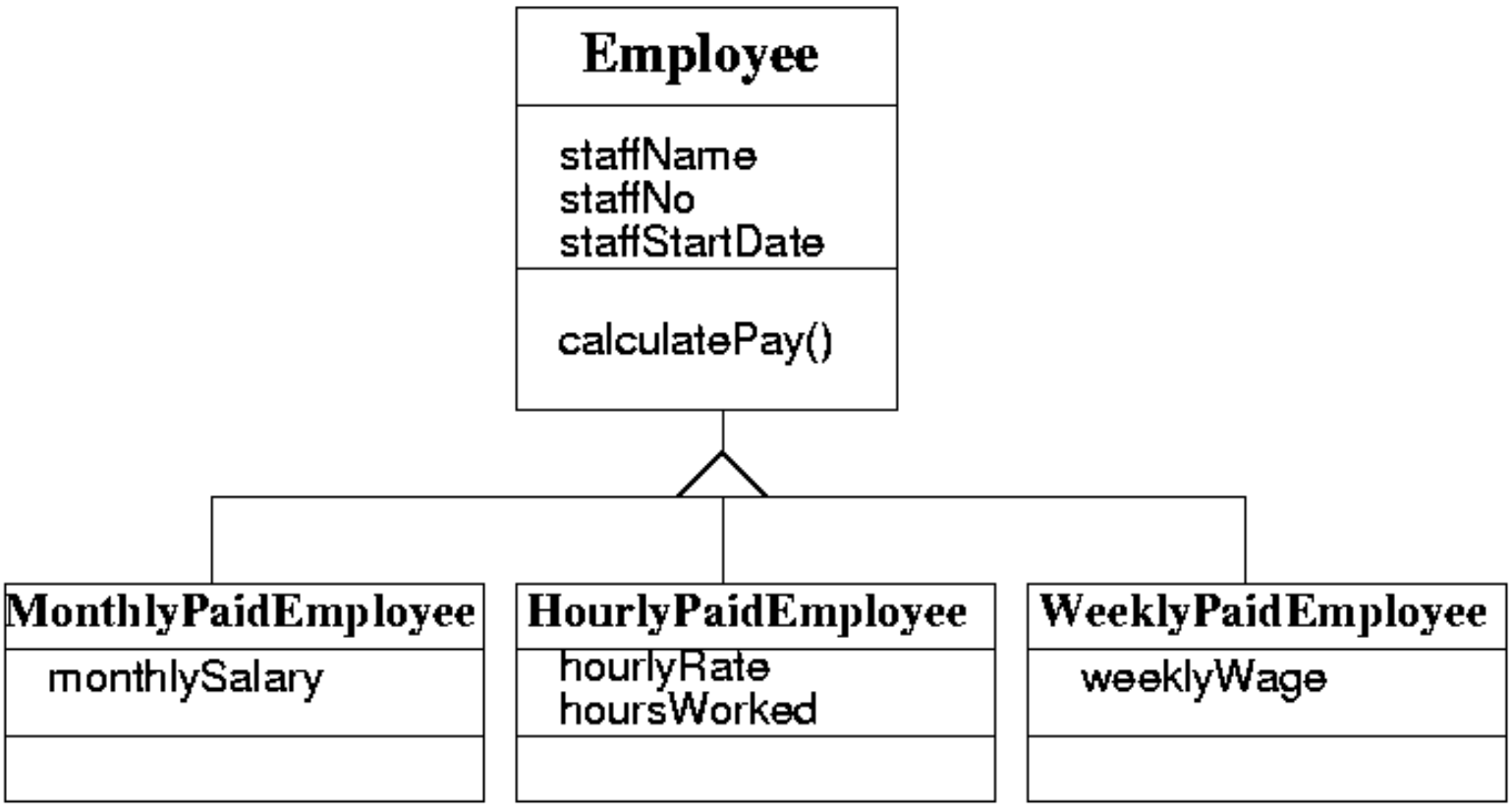
Polymorphism

1. Employee *p;
2. MonthlyPaidEmployee mE;
3. WeeklyPaidEmployee wE;
4. HourlyPaidEmployee hE;
5. p = &mE;
6. p->calculatePay();
7. p = &hE;
8. p->calculatePay();
9. p = &hE;
10. p->calculatePay();

Polymorphism

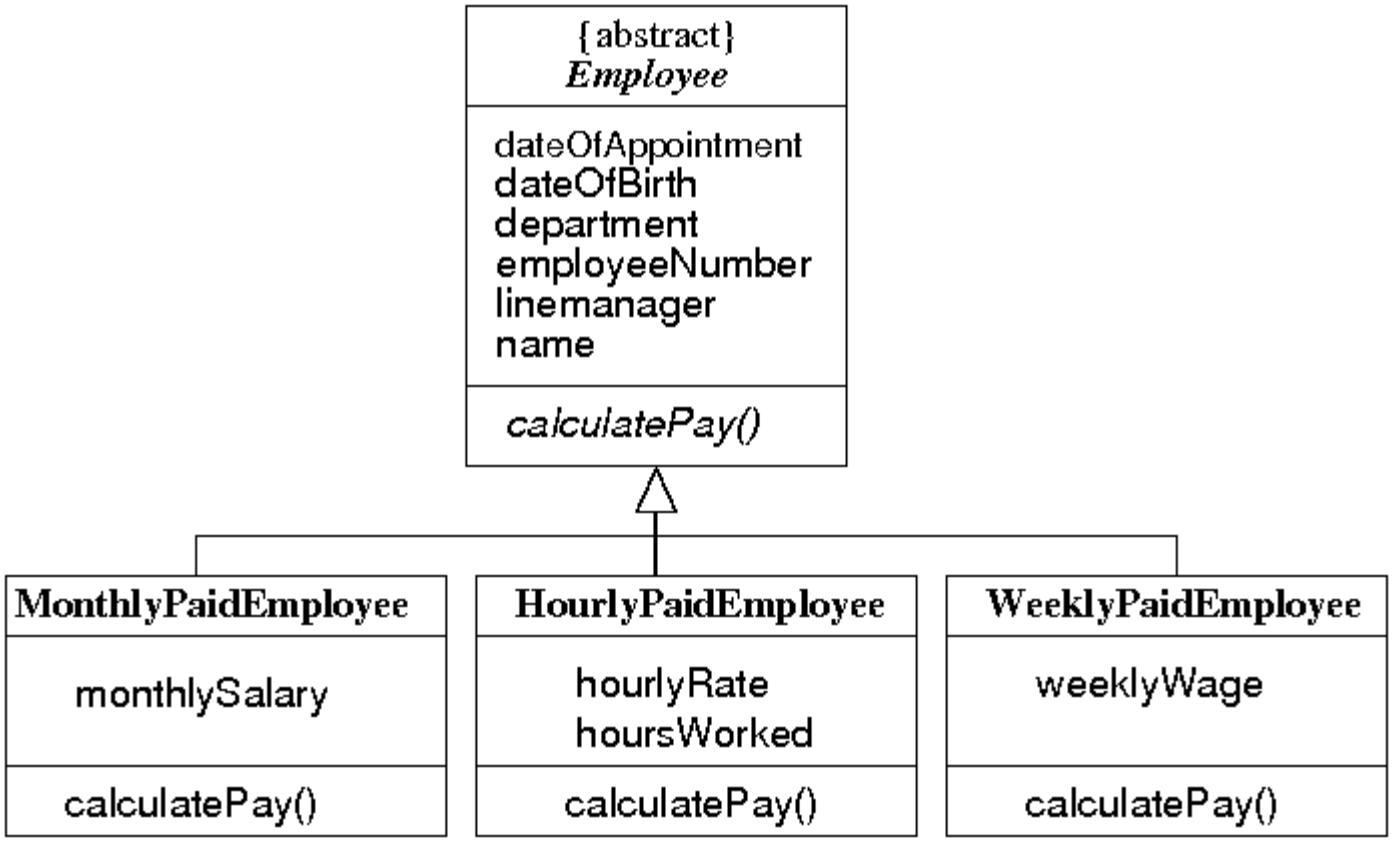
- We have seen
  - List of employees
    - where each element is instantiated from a different subclass in the inheritance hierarchy
  - List of shapes and drawables
    - Where each element is instantiated from a subclass in the inheritance hierarchy, or a class that implements the interface
  - Talked about a list of students
    - where each element in the list is instantiated from:
      - different subclasses in the student taxonomic hierarchy AND/OR
      - Classes that implement an explicit student interface
    - i.e EUundergraduate, nonEUundergraduate, EUlinkIn, nonEUlinkIn
- Can call operations on elements in the list without having to specify which implementation to execute
  - E.g. item.calculatePay(), item.drawShape(), item.calculateFee();
- Makes programming easy
  - Without polymorphism, would have to .....

Polymorphism (or lack of)



All subclasses inherit the implementation of the method calculatePay() in the superclass.

Polymorphism



Italics imply that the element is abstract

Summary

- **Key Concepts**
  - Classes and Objects
  - Inheritance – generalisation.
  - Polymorphism – compile time (static) and run time (dynamic).
  - Templates (discussed later).
- Reading
  - Chapter 4 from Bennett, McRobb, and Farmer

Historical Note: Evolution of Design

- In the beginning: no design - spaghetti programming.
- The old days: Structured Systems Analysis and Design Methodology (SSADM), with a focus on modelling the processes in the system –
  - functional/procedural design. Data Flow Diagrams (DFDs) capture the processes and the flow of data between processes.
  - DFDs continuously refined and eventually, each process specified using pseudocode or control flow charts.
  - Associated with procedural (structured) programming languages such as Pascal, Fortran, Cobol, Basic, C, etc.
- Currently: *database (DB)* community standardised design using entity relationship (ER) diagrams. Focus on data and information. Most methods utilise ER diagrams for the data modelling element of the systems analysis and design phase.

Historical Note: Evolution of Design

- Last decade: Object oriented design (OOD) using UML - object oriented programming (OOP) using C++, Java, SmallTalk, Simula. Concept of patterns. Focus on all aspects of 5 component model, weak on environment.
- Now: Component and framework design - extends OOD and OOP languages,
  - Sun's J2EE / Java 6, Microsoft .Net
  - OMG's Component Composition Model (CCM)

Origins of the OO Paradigm

- Early work on computer simulation - event driven programming, very difficult for 3GLs. Led to development of Simula.
- Structure program as a set of independent software agents, each agent represents an agent in the real world.
- Resolves tension between application domain model and the model of the software - convergent engineering.
- Spread of GUIs: difficult to design or control in a procedural way.

Origins of the OO Paradigm

- Model transitions: in structured approaches, the models of the process that are developed during the analysis phase (e.g. DFDs) have only an indirect relationship to the models developed during design phase (e.g. structure charts, SSADM, update process models).
- Object oriented analysis avoids these transitions.
- In UML, use cases and class diagrams constitute the basis of design.
- Reusable software.

Reading

- Bennett, McRobb, and Farmer: chapter 4