Principles of Object-Orientation

- Encapsulation
- Inheritance
- Polymorphism

CE202 Software Engineering, Autumn term

Dr Cunjin Luo, School of Computer Science and Electronic Engineering, University of Essex

Object-orientation

- So far we have briefly touched on how analysis and design (and implementation) can make use of objects
  - Eg. objects in activity diagrams, class diagrams, etc
- In this lecture we will explore common object-oriented principles
- As we get more into UML we will make more use of objects

Advantages of O-O

- Can save effort
  - Reuse of generalized components cuts work, cost and time
- Can improve software quality
  - Encapsulation increases modularity
  - Sub-systems less coupled to each other
  - Better translations between analysis and design models and working code
  - Objects are good for modelling what happens in the real world
  - Can be used throughout the software lifecycle ie. requirements -> design -> implementation -> testing

OO Analysis & Design: mechanisms of abstraction

- Fundamentally: the same abstraction mechanisms as object-oriented programming:
  - Encapsulation: classes and objects
    - Other possible modularization techniques: interfaces, packages/namespaces
  - Inheritance
    - Generalization/specialization
    - Subtyping
    - Subclassing
  - Polymorphism
    - Overloading
    - Dynamic binding

Encapsulation

Objects

An object is:

"an abstraction of something in a problem domain, reflecting the capabilities of the system to

- keep information about it,
- interact with it,
- or both."

Coad and Yourdon (1990)

Objects

"Objects have state, behaviour and identity."

Booch (1994)

- *State*: the condition of an object at any moment, affecting how it can behave
- *Behaviour*: what an object can do, how it can respond to events and stimuli
- *Identity*: each object is unique

Examples of Objects

Object	Identity	Behaviour	State
A person	**'Hussain Pervez.'**	**Speak, walk, read.**	**Studying, resting, qualified.**
A shirt	**My favourite button white denim shirt.**	**Shrink, stain, rip.**	**Pressed, dirty, worn.**
A sale	**Sale no #0015, 18/05/05.**	**Earn loyalty points.**	**Invoiced, cancelled.**
A bottle of ketchup	***This* bottle of ketchup.**	**Spill in transit.**	**Unsold, opened, empty.**

Can you suggest other behaviours and states for these objects?

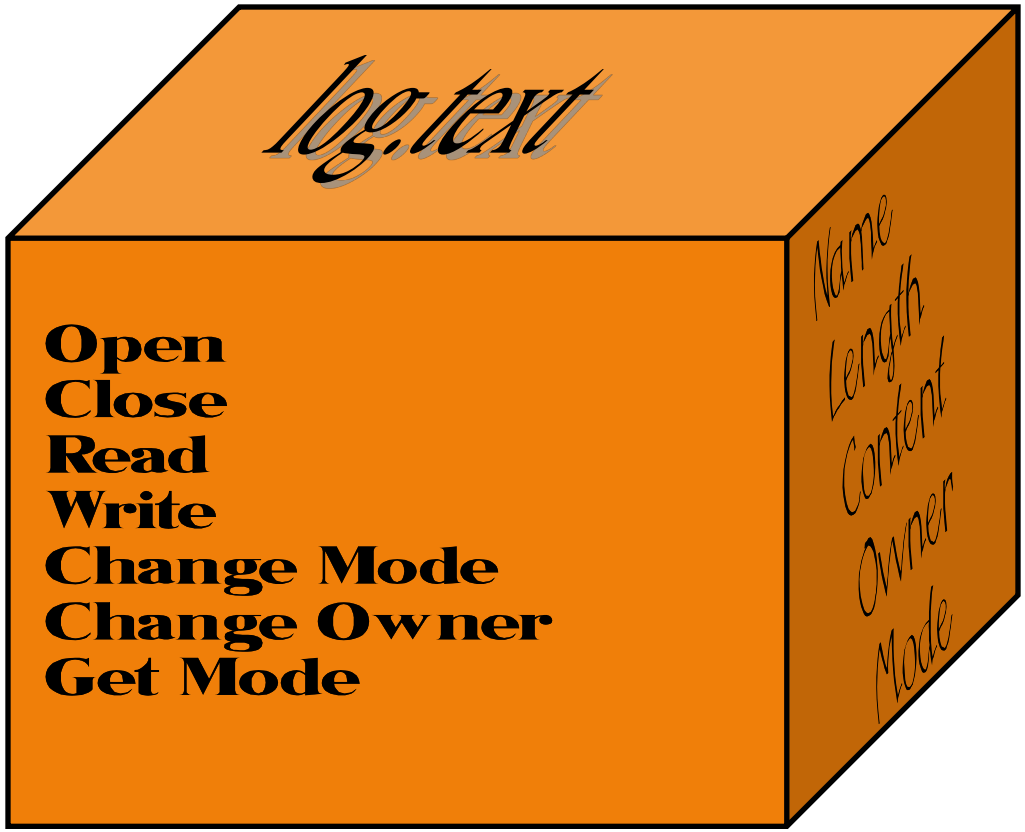
How can external events and object behaviour both result in a change of state?

How can state restrict the possible behaviours of an object?

Examples of objects (cont.)

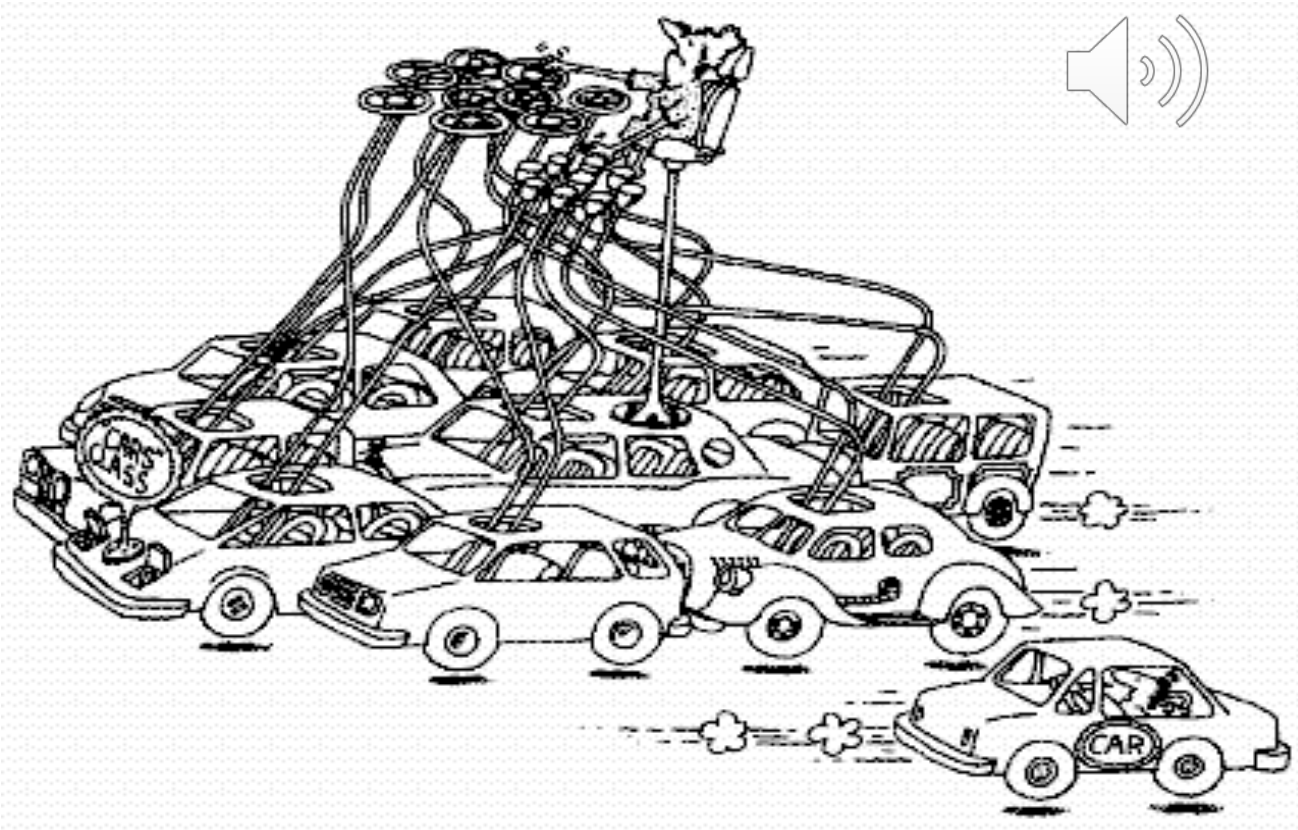
- Examples:
  - Time point
    - Data: 16:45:00, Feb. 21, 1997
    - Operations: add time interval, calculate difference from another time point,
  - Acts. For example: Measurement of a patients' fever
    - Data: 37.1C, by Deborah, at 10:10 am
    - Operations: Print, update, archive
  - File
    - Data: log.txt, -rwx------, Last read 21:07 June 1, 1999, …
    - Operations: read, write, execute, remove, change directory, …
  - A communication event (time, length, phone-number, …)
  - Transaction in a bank account (withdraw $15, time, …)
  - Elements of ticket machine dispenser: ticket, balance, zone, price, …

Object: Example



Class: Abstraction Over Objects

- A class represents a set of objects that share a common structure and a common behavior.



Class and Instance

- All objects are *instances* of some *class*
- A Class is a description of a set of objects with similar:
  - features (attributes, operations, links);
  - semantics;
  - constraints (e.g. when and whether an object can be instantiated).

OMG (2009)

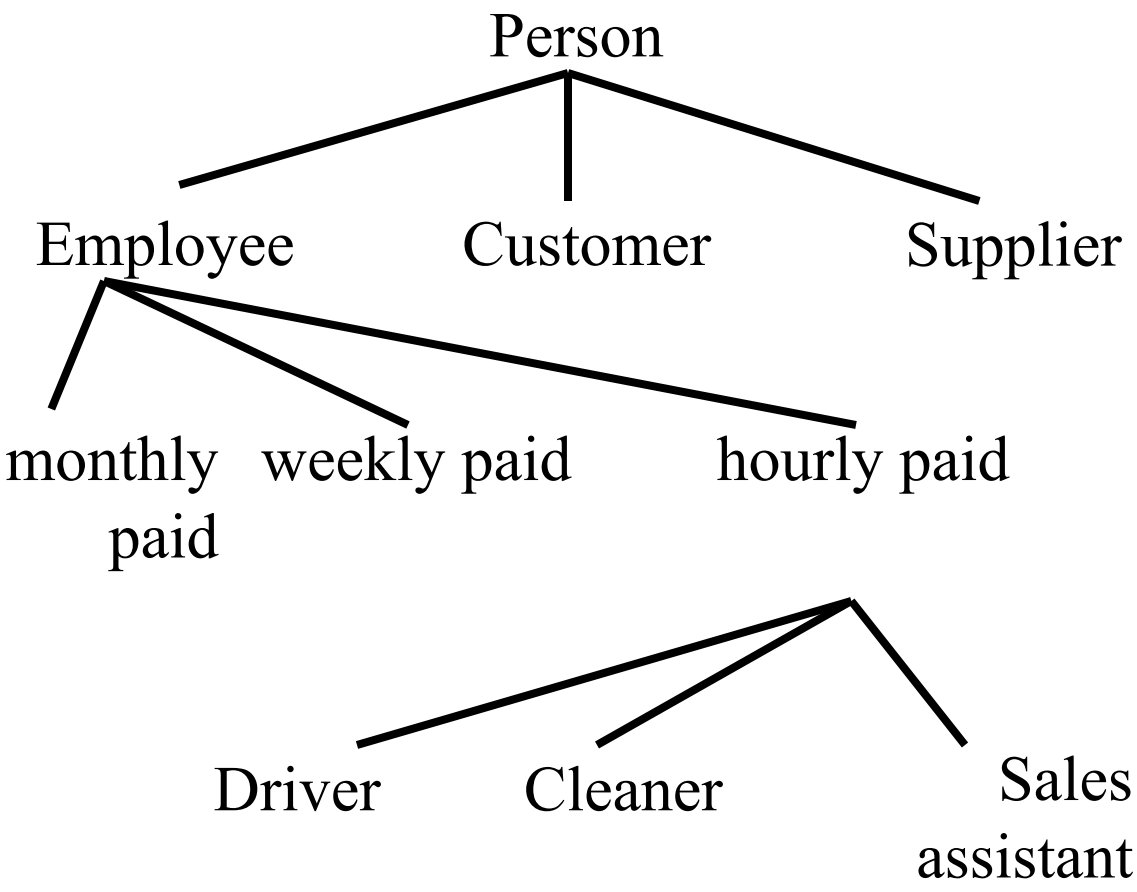
Class and Instance

- An object is an instance of some class
- So, instance = object
  - but also carries connotations of the class to which the object belongs
- Instances of a class are similar in their:
  - *Structure*: what they *know*, what information they hold, what links they have to other objects
  - *Behaviour*: what they *can do*

Generalization and Specialization

- Classification is hierarchic in nature
- For example, a person may be an employee, a customer, a supplier of a service
- An employee may be paid monthly, weekly or hourly
- An hourly paid employee may be a driver, a cleaner, a sales assistant

Specialization Hierarchy



More general (superclasses)

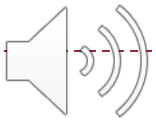
More specialized (subclasses)

Generalization and Specialization

- More general bits of description are *abstracted out* from specialized classes:

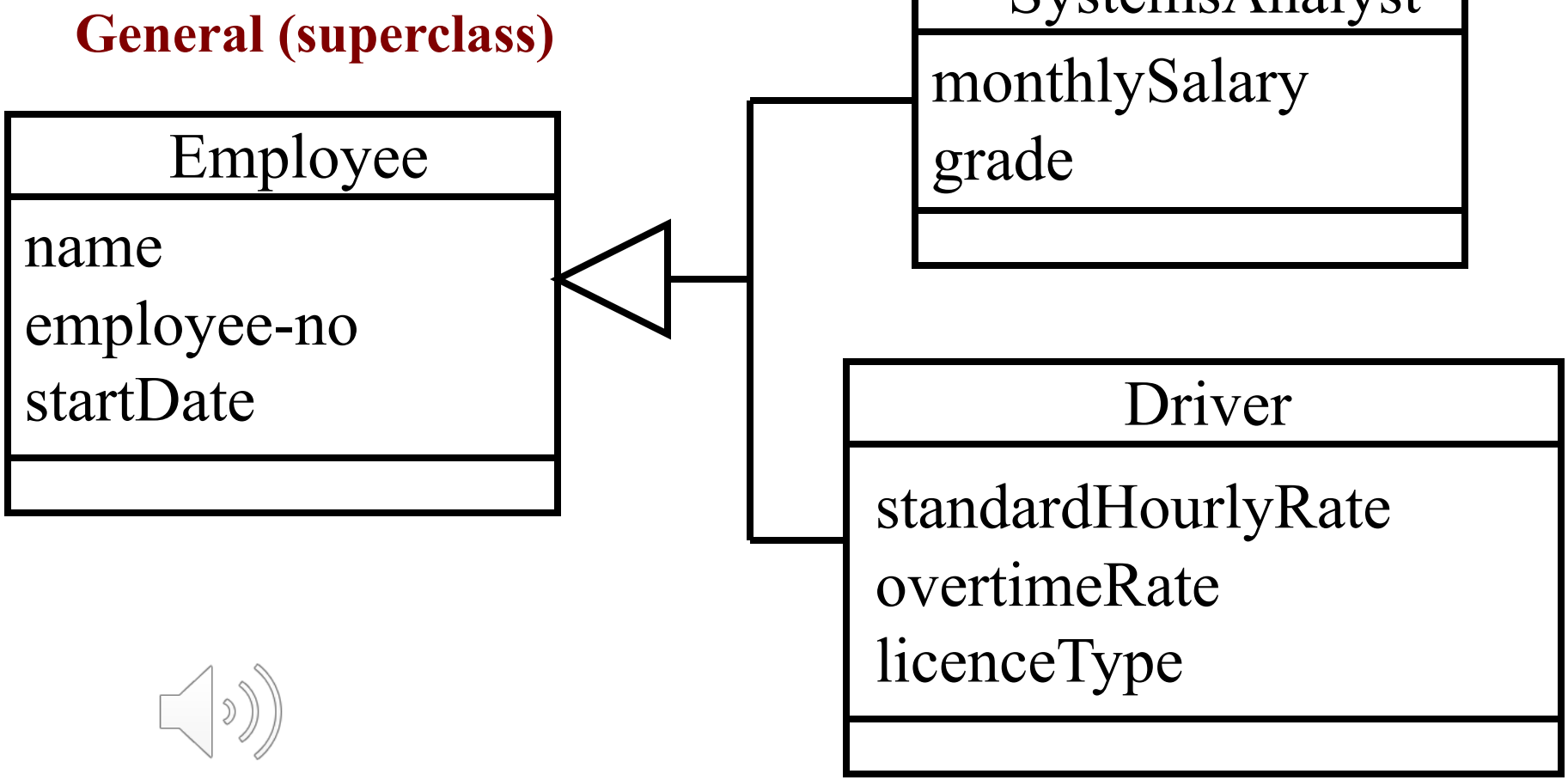
SystemsAnalyst
name employee-no startDate monthlySalary grade

Driver
name employee-no startDate standardHourlyRate overtimeRate licenceType

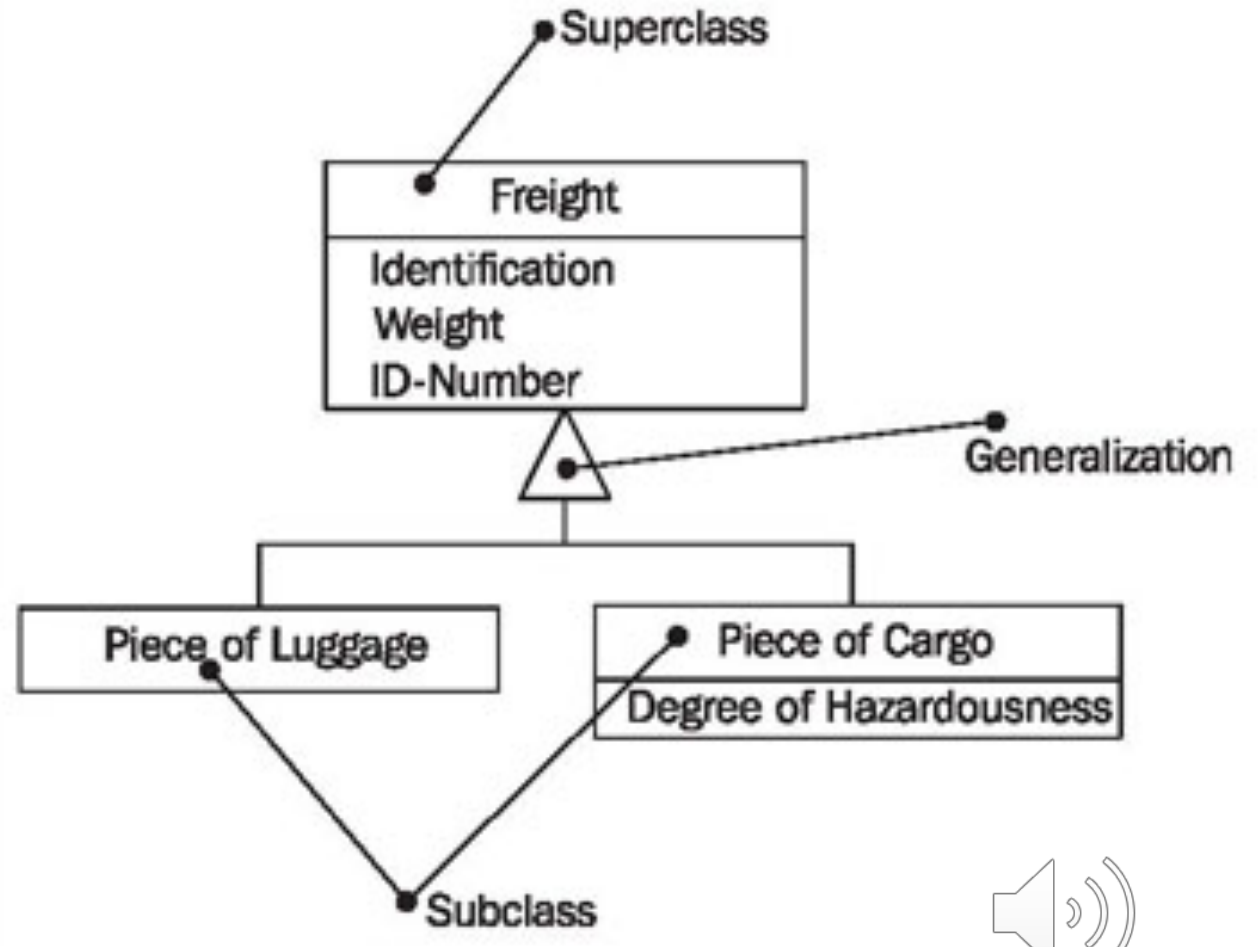


Specialized (subclasses)

General (superclass)

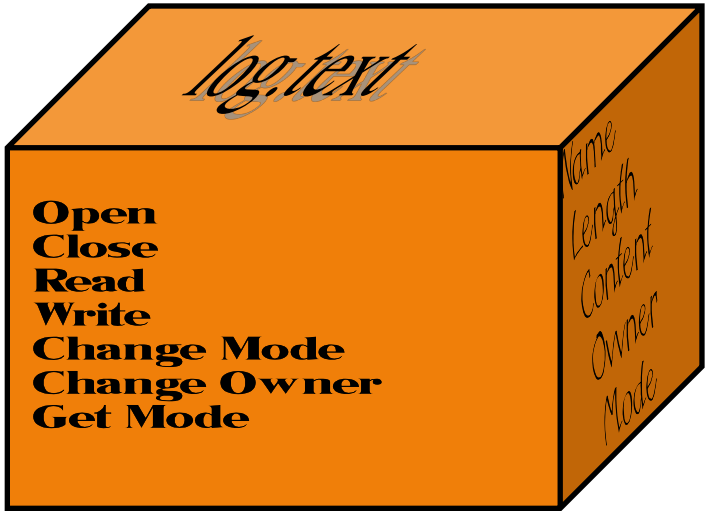


Object-oriented modelling notation: genericity in class diagrams



Encapsulation definition

- Encapsulation: an object's data is located with the operations that use it



Message-passing

- Several objects may collaborate to fulfil each system action/use-case
- "Record CD sale" could involve:
  - A CD stock item object
  - A sales transaction object
  - A sales assistant object
- These objects communicate by sending each other messages
- We will model these messages when we look at sequence and collaboration diagrams

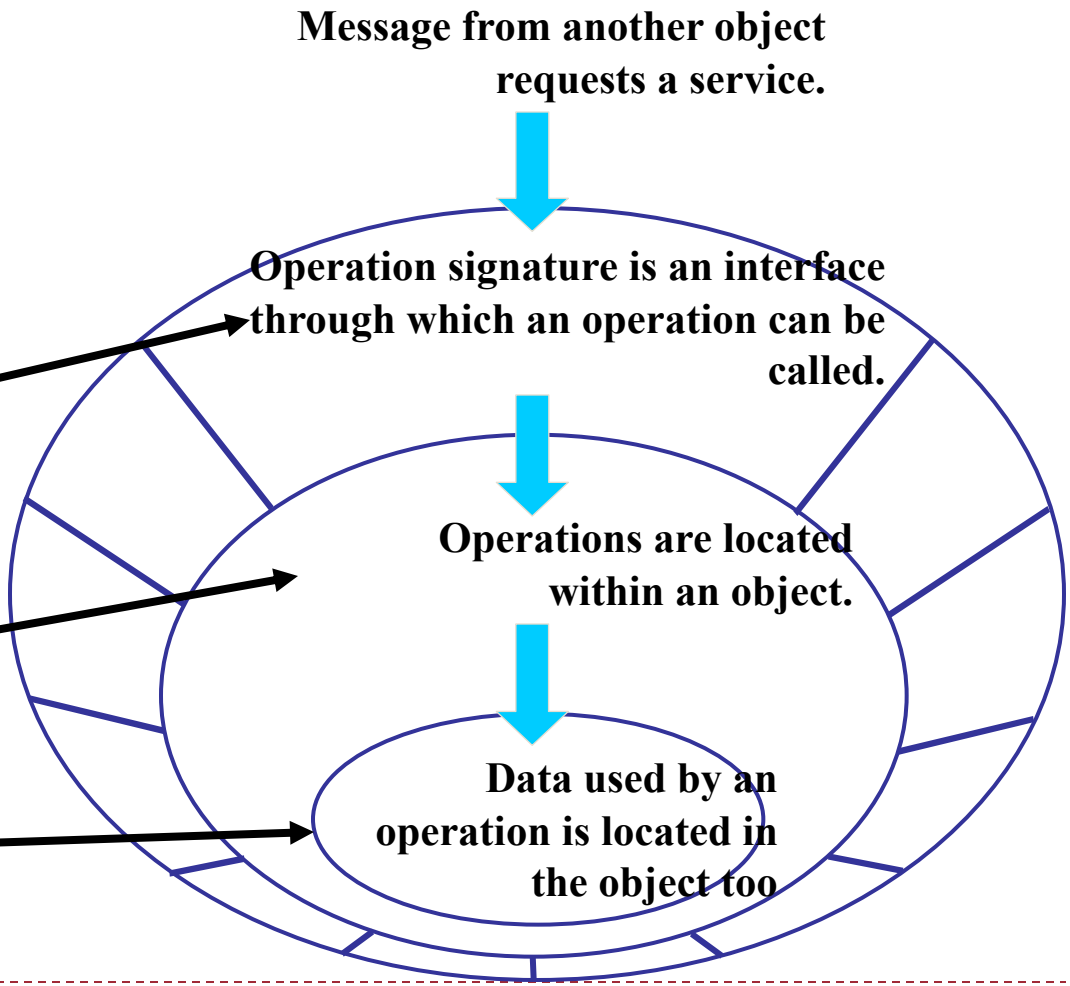
Message-passing and Encapsulation

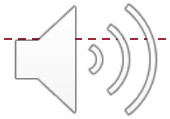
'Layers of an onion' model of an object:

An outer layer of operation signatures…

…gives access to middle layer of operations…

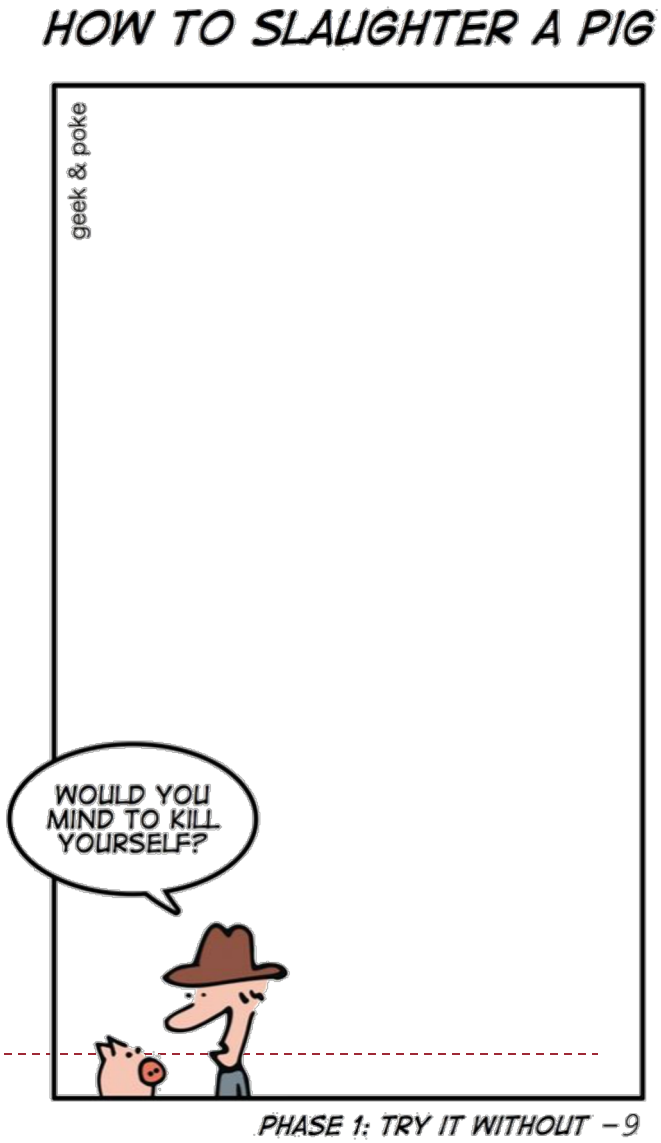
…which access an inner core of data





Classes

- Kinds of classes:
  - Abstract (also: Java interface)
    - No instances!
    - Serve as an interface, or a common base with similar behavior
    - Example: `Collection` (Smalltalk, Java)
  - Singleton
    - One instance!
    - Example: `True` (Smalltalk)
    - Language support for singletons:
      - Self: No classes! All objects are singletons
      - BETA: Objects may be defined as singletons
  - Concrete/effective
    - Any number of instances



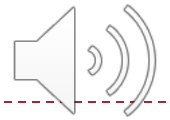
Classes in O-O programming languages

- C++

```
class Employee: public Person {
private:                        // visible to none
  Date birthday;                // data member
public:                         // visible to all
  void hire(Reason why)         // function member
  { /* ... */ }
};
```

- Java:

```
class Employee extends Person {
  private Date birthday;                // field, visible to none
  public void Hire(Reason why) {        // method, visible to all
  { /* ... */  }
}
```



Classes in OOPLs (Cont.)

- Smalltalk:

```
Person subclass: Employee
  instanceVariables:  'birthday'  "instance variable, visible to none"
  classVariables:  ' '
  poolDictionaries:  ' '

hire: why          "method, visible to all"
  "..."
```

- Eiffel:

```
class EMPLOYEE
inherit
   PERSON
feature {NONE}                  -- visible to none
   DATE birthday;               -- attribute
feature {ALL}                   -- visible to all
   hire(why: REASON) is -- routine
         do
            -- ...
         end
end -- class EMPLOYEE
```

Information Hiding

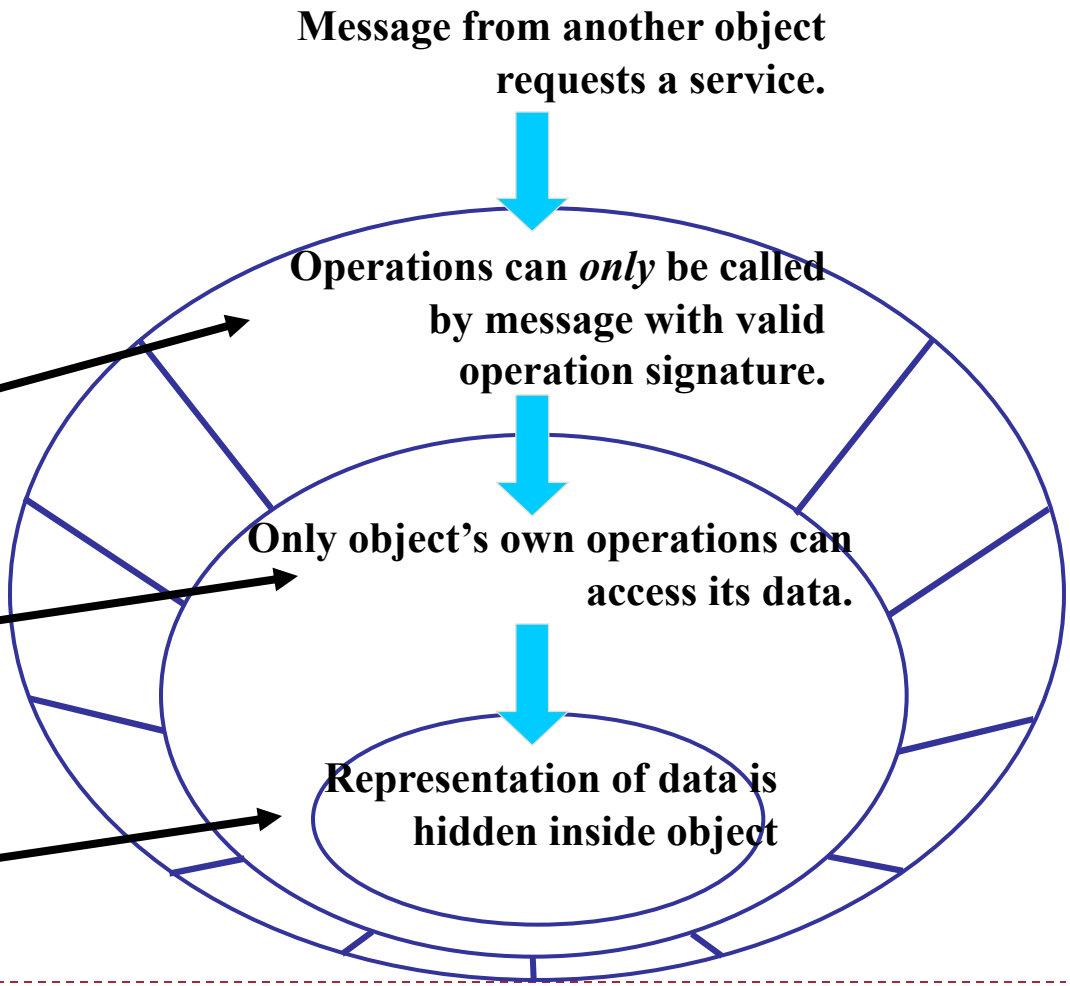
Information Hiding: the onion model

'Layers of an onion' model of an object:

Only the outer layer is visible to other objects…

…and it is the only way to access operations…

…which are the only way to access the hidden data



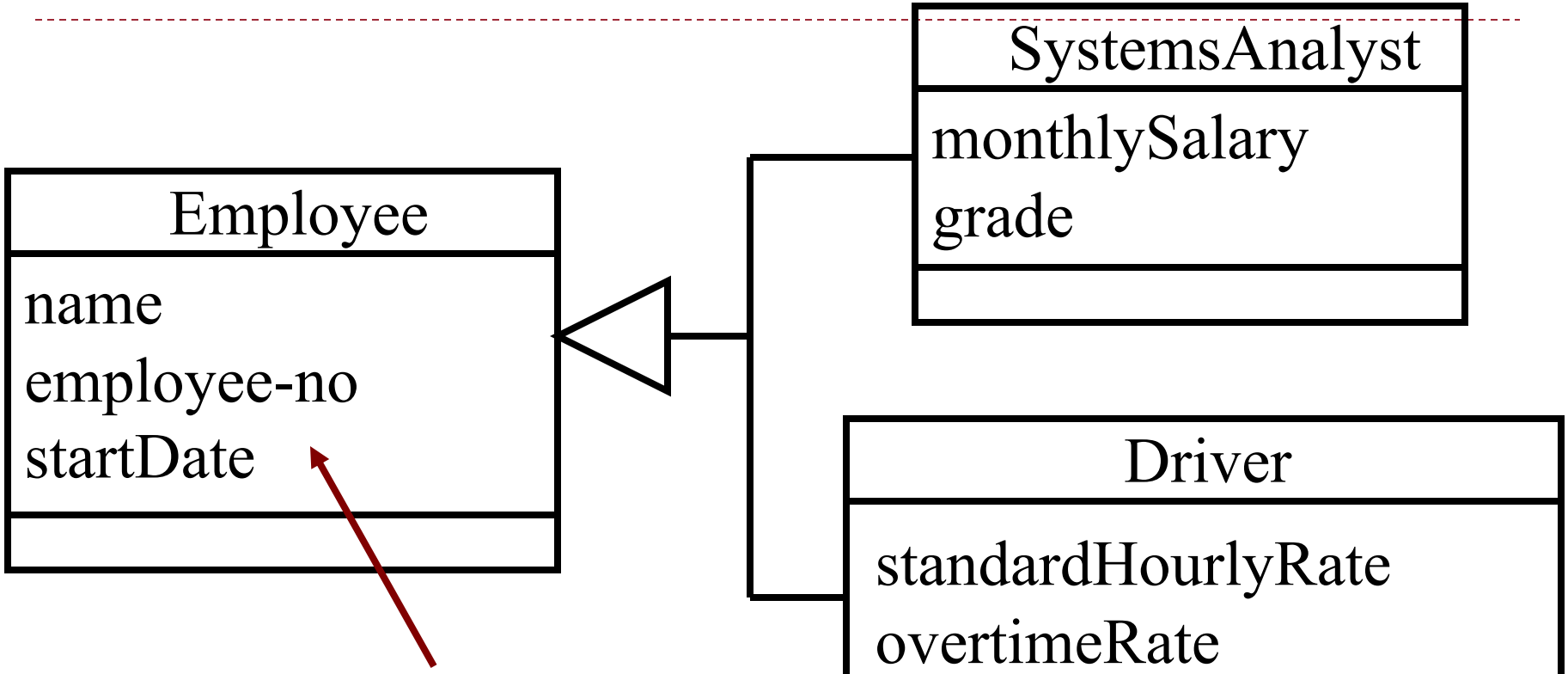
Note: Information Hiding Vs. Encapsulation

- The terms are sometimes confused and used interchangeably
  - Some people say "encapsulation" with reference to what we described as information hiding
- We shall adhere to the interpretation in these slides
  - Encapsulation: an object's data is located with the operations that use it
  - **Information hiding: only an object's Interface is visible to other objects**

Inheritance

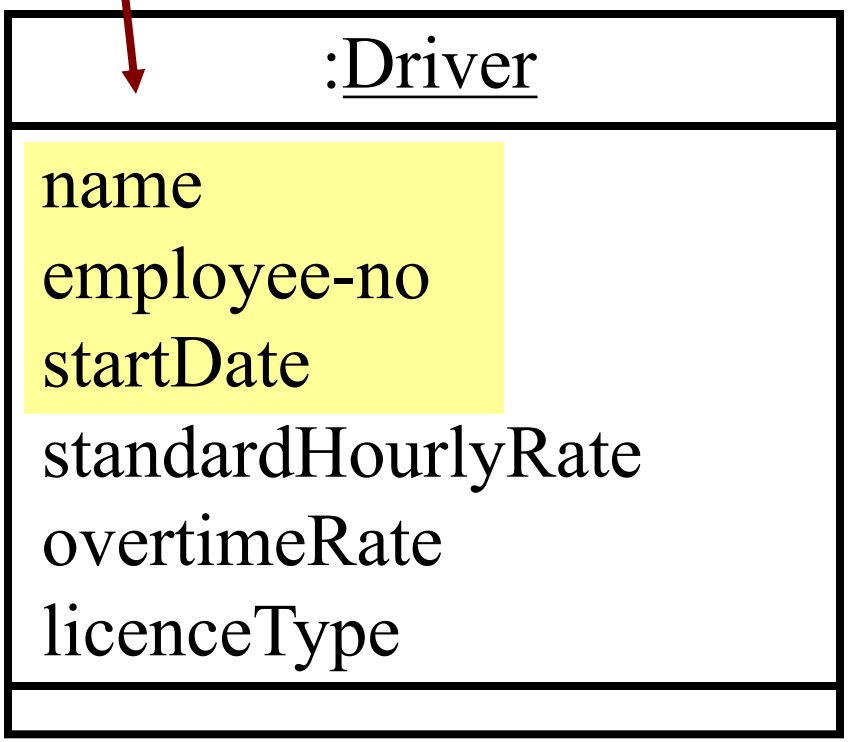
Inheritance

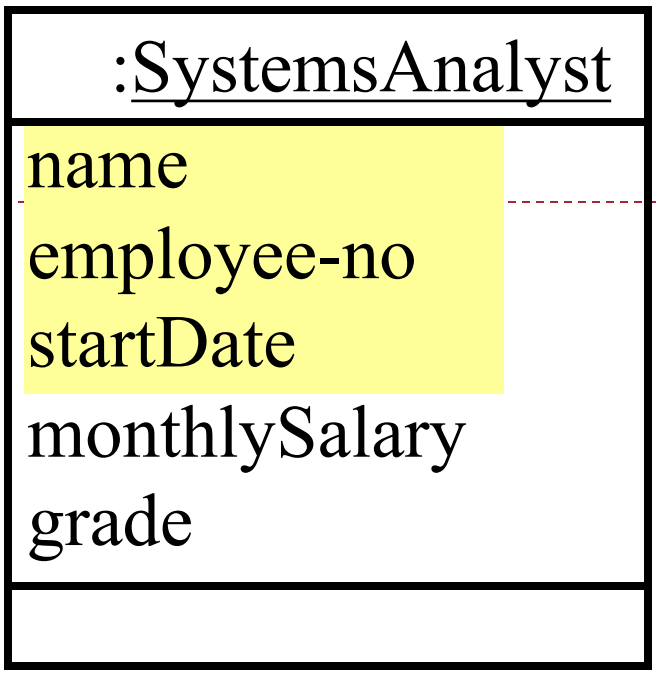
- The *whole* description of a superclass applies to *all* its subclasses, including:
  - Information structure (including associations)
  - Behaviour
- Often known loosely as *inheritance*
- (But actually inheritance is how an O-O programming language *implements* generalization / specialization)



All characteristics of the superclass are *inherited* by its subclasses

***Instances* of each subclass include the characteristics of the superclass (but not usually shown like this on diagrams)**



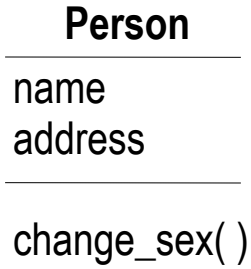


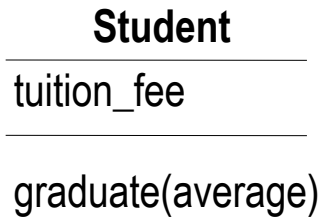
Inheritance

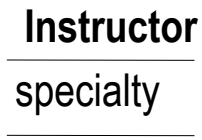
- Represents separate notions:
  - Generalization: ‘is-kind-of’ relation
    - Instances of the specialized class are a subcategory of the generalized class
  - Subtyping (in Java: ‘implements’)
    - The subtypes supports all the operations on supertype
  - Subclassing (in Java: extends)
    - A mechanism of code reuse

Inheritance *and* generalization

- Superclass is a generalization of Subclass
  - Also: subclass is a specialization of superclass
- Also known as: is-kind-of relation:
  - Staff is-kind-of Person
  - Instructor is-kind-of Person

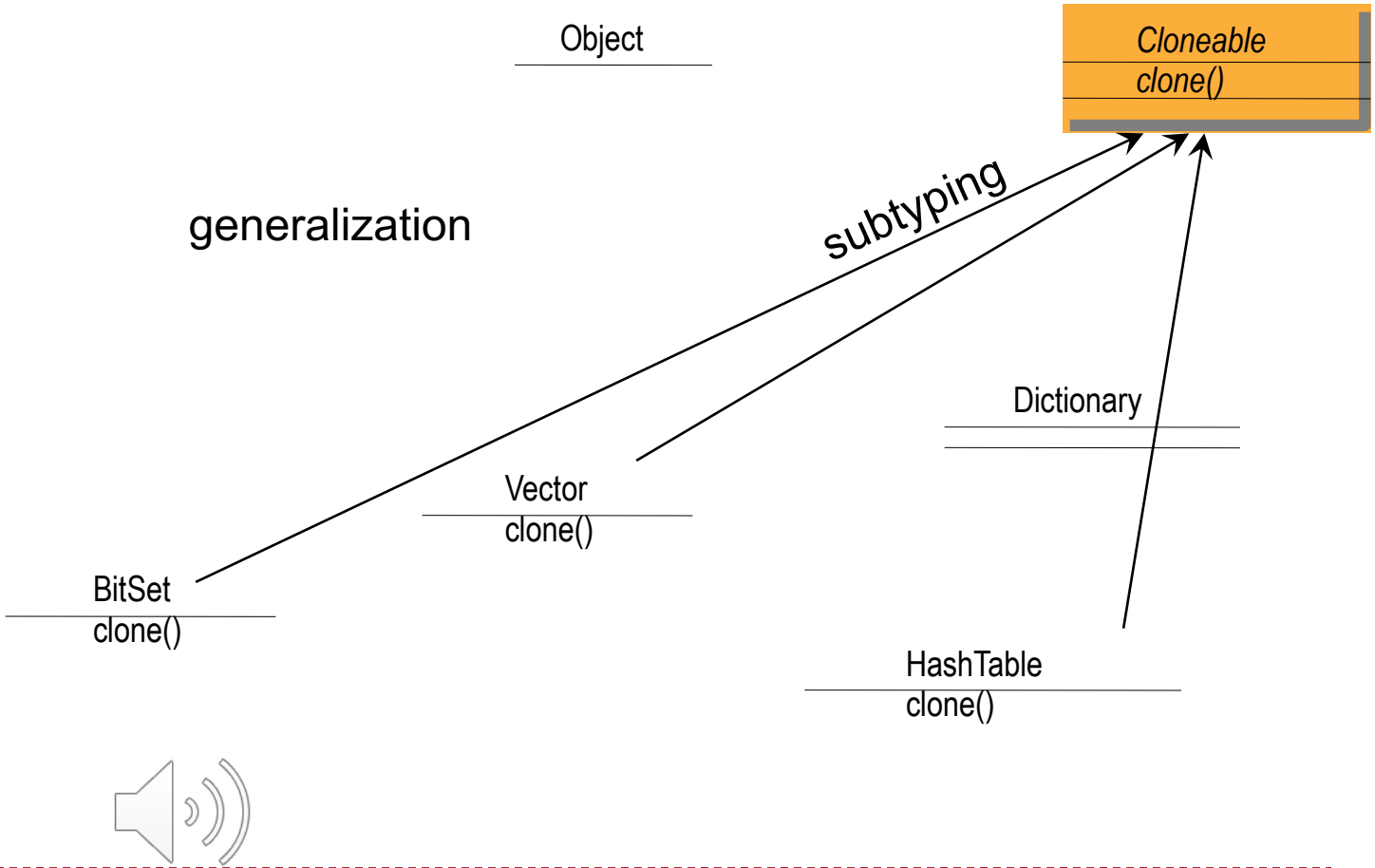






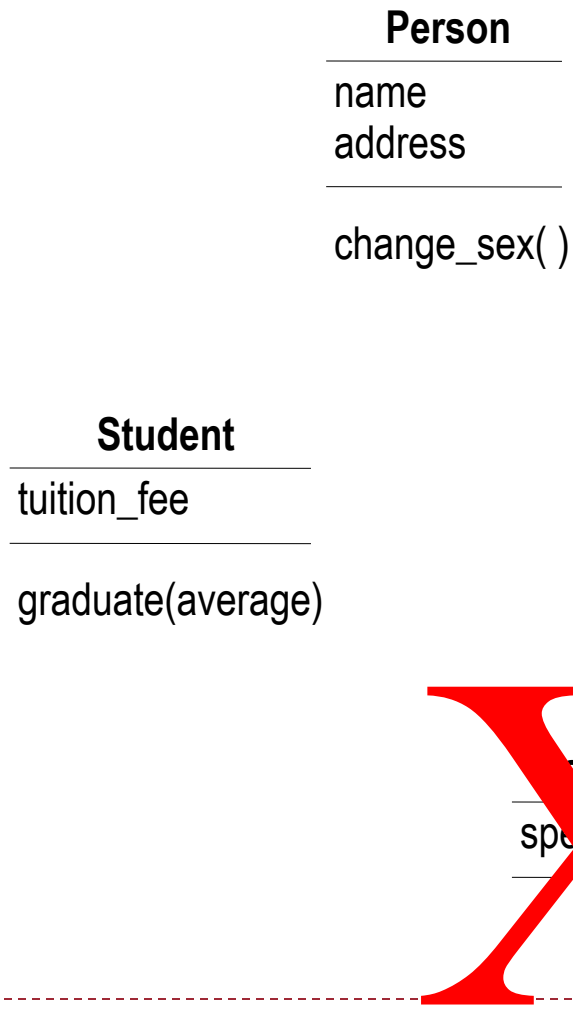
Inheritance *and* subtyping

- Subtype supports the same operations as supertype



Inheritance *and* subclassing

- Inheritance can be used as a crude mechanism of reuse
- A very problematic and dangerous tactic



In this session we have looked at

- What is an object
- What is a class
- Advantages of OO
- OO mechanisms of
  - Encapsulation/Information Hiding
  - Inheritance
- In the class we will look at Polymorphism
  - Overloading
  - Dynamic binding
  - Genericity

Exercises: OO inheritance

- What rules describe the relationship between a subclass and its superclass?
- For each one of the following class pairs, determine the appropriate kind of inheritance relation between them:
  - Person, Parent
  - Person, Mammal
  - Bird, FlyingObject
- What is the difference between generalization and sub-typing?

Exercises: OO inheritance

- What rules describe the relationship between a subclass and its superclass?
- For each one of the following class pairs, determine the appropriate kind of inheritance relation between them:
  - Person, Parent
  - Person, Mammal
  - Bird, FlyingObject
- What is the difference between generalization and sub-typing?

Summary

- OO Analysis & Design has same mechanisms as OO programming
- OO provides many benefits
- Difference between Objects and Classes
- Looked at core OO concepts:
  - Encapsulation
    - Information Hiding
  - Inheritance
    - Generalization/specialization
  - Polymorphism
    - Overloading
    - Dynamic binding
    - Genericity

Further reading

- Object-orientation
  - Chapter 4, Bennett
  - Coad, P & Yourdan, E (1990) Object-oriented analysis. Prentice-Hall.
  - Booch, G (1994) Object-oriented analysis and design with applications. Menlo Park.
- Information hiding & abstraction:
  - Wirfs-Brock, Rebecca, Wilkerson, Brian, and Wiener, Lauren. 'Designing Object-Oriented Software'. Prentice-Hall, 1990.
  - Parnas, 1985, Communications of the ACM.
- Look at Java Interfaces
  - Eg. http://docs.oracle.com/javase/tutorial/java/concepts/interface.html
  - Look at Java abstraction
    - Eg. http://javarevisited.blogspot.co.uk/2010/10/abstraction-in-java.html
- Exemplar based object-orientation:
  - 'An exemplar based Smalltalk'. OOPSLA 1986 Proceedings.