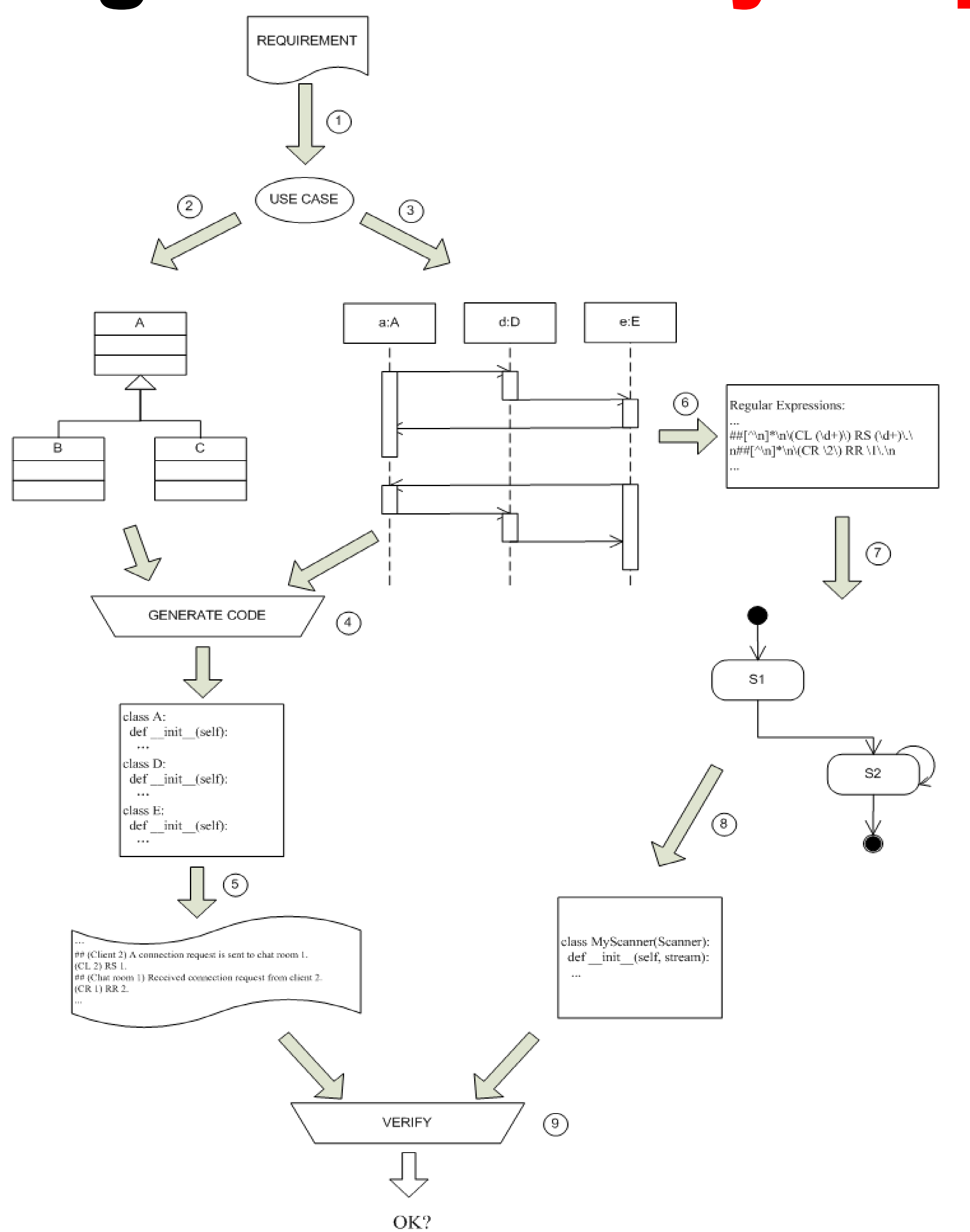
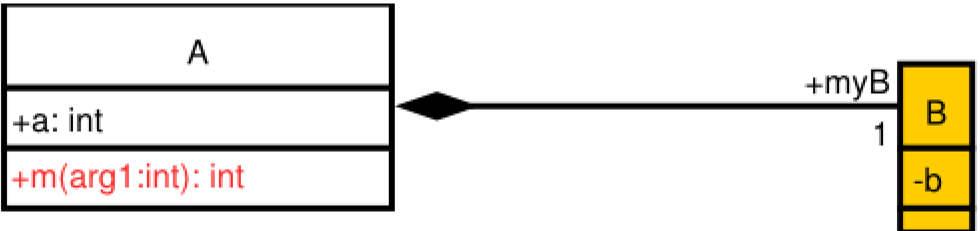
Quality of Design

Design must satisfy Requirements ~ testing



Design must be Consistent

(i..e., no inconsistencies)



ainstance:A

binstance:B

cinstance:C

m("hello")

p()

Design must satisfy (type) Constraints

- Liskov Substitutability Principle
  - State-space, behaviour
  - In presence of inheritance
- Closed Behaviour

Quality of Design

- What is a good object-oriented design?
- How to determine whether a given design is good?
- What are the quality characteristics (qualitative)?
- What are the quality metrics (quantitative)?

A closer look at Designs

- Domains of classes
  - Not all classes are equal
- Encumbrance
  - Measure for class sophistication
  - Law of Demeter
- Class cohesion
  - Mixed – instance/domain/role

Classes used in a Human Resources (HR) System

- Employee
- Date / Time
- Salary
- Performance Review
- Job Position
- Job Offer
- Recruitment
- Currency
- Bonus
- Location/Office

Classes used in an Inventory System

- Equipment
- Bar code
- Loan History
- Date/Time
- Employee
- Location/Office
- Repair Order
- Repair History
- Purchase Order
- Currency

Classes used in an Accounting System

- Client
- Account
- Invoice
- Date / Time
- Currency
- Employee
- Bar code
- Delivery
- Pickup

Similarities

<u>HR</u>

- Employee
- Date / Time
- Salary
- Performance Review
- Job Position
- Recruit
- Currency
- Location/Office

<u>Inventory</u>

- Equipment
- Bar code
- Loan History
- Date/Time
- Employee
- Location/Office
- Repair Order
- Purchase Order
- Currency

<u>Accounting</u>

- Client
- Account
- Invoice
- Date / Time
- Currency
- Employee
- Bar code
- Delivery
- Pickup

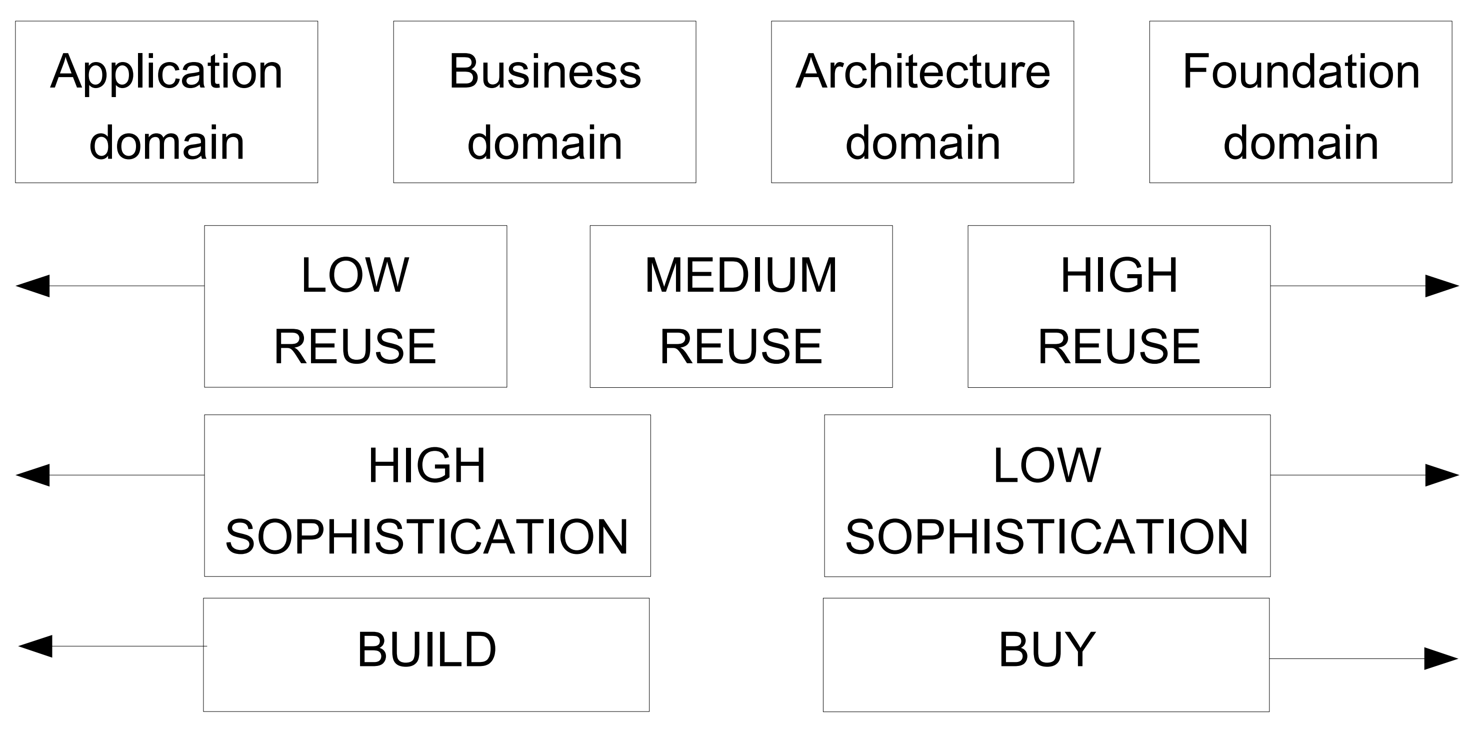
Domains of Classes

- Foundation Domain
  - classes useful for all businesses, architectures, applications
    - Fundamental: int, boolean, ... (data types)
    - Structural: stack, tree, ... (container data structures)
    - Semantic: date, time, height (adds meaning, often units)
- Architecture Domain
  - classes useful for a single architecture
    - Networking: port, socket, ...
    - Database: transaction, rollback, ...
    - User Interface: window, button, ...

Domains of Classes

- Business Domain
  - classes useful for one type of business
    - Attribute: BodyTemperature of patient
    - Role: Patient
    - Relationship: PatientSupervision
- Application Domain
  - classes useful for one application
    - MNIPatientTemperatureMonitor (event recognizer)
    - RoyalVictoriaHospitalInventoryPurchaseOrder

Reusability & Sophistication



Encumbrance

- Quantitative measure for the distance of a class to the foundation domain (i.e., its sophistication)
- Encumbrance : take a class C and measure the number of classes C depends on, recursively ...
- First introduce direct and indirect class reference sets

Class Reference Set

- Direct class reference set refers to the set of classes that a given class C directly refers to (via inheritance, association, ...), call these C1, C2, C3, ...

- Refers to ...
  - Inherits from Ci
  - Has attribute of type Ci
  - Has method with argument/return value of type Ci

Class Reference Set

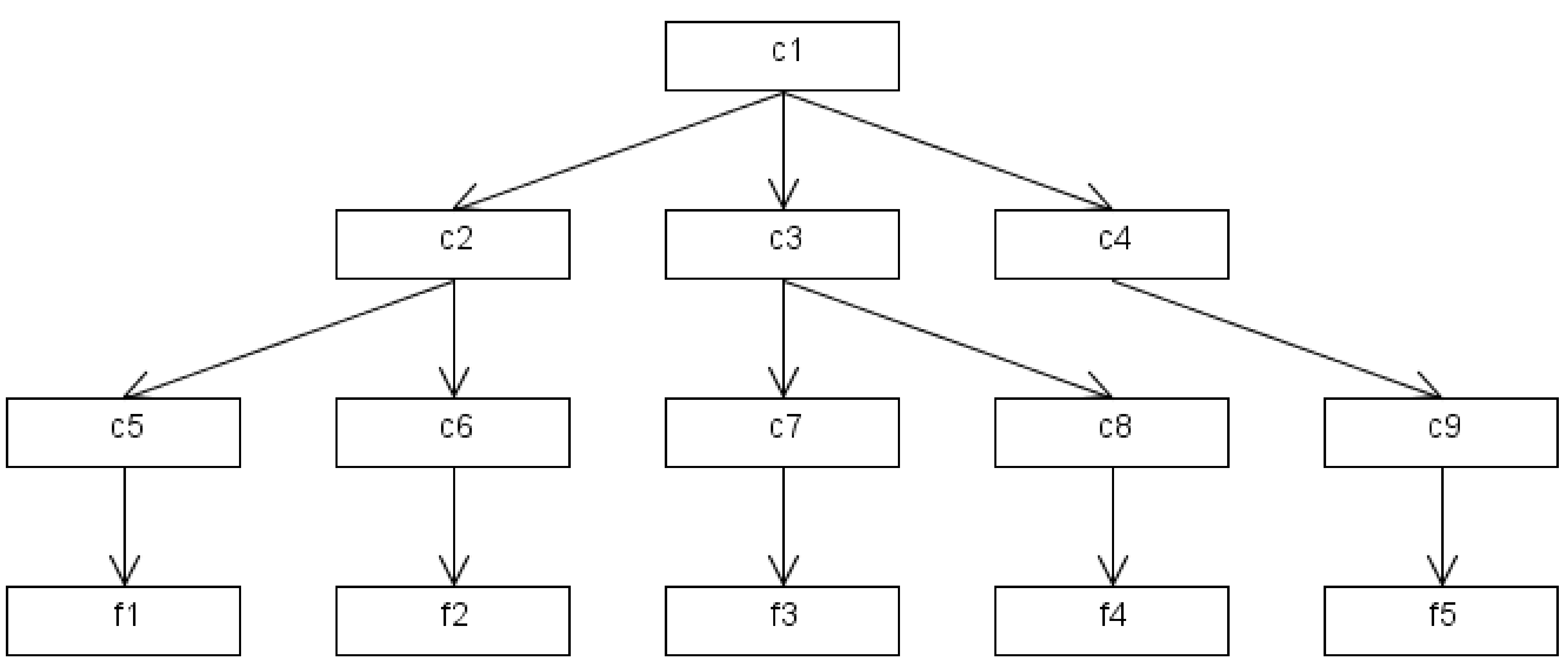
- Indirect class reference set of C is the union of its direct class reference set DCRS(C) and the indirect class reference sets of the elements of DCRS(C).

- Indirect Class Reference Set has a recursive definition ... so when does it stop?
- ... the direct class reference set of classes of the foundation domain is the empty set.

Enumbrance

- Direct and indirect class reference sets lead to direct and indirect encumbrance, which is just the size of the respective class reference set

Simple Example



Low/High Encumbrance

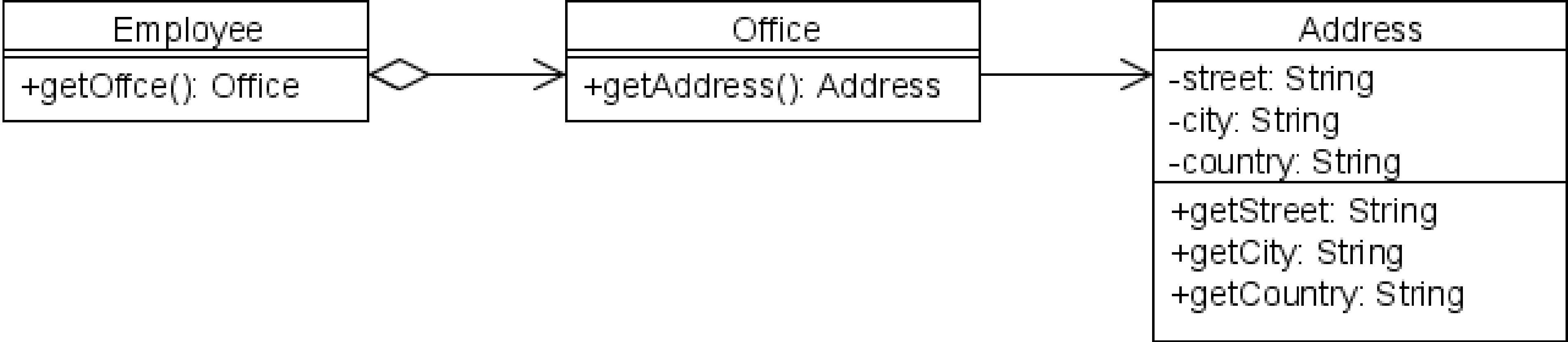
- A foundation class should have low encumbrance.
- An application class should have high encumbrance.
- A good indication of a problem in the design is:
  - High indirect encumbrance in the Foundation domain.
  - Low indirect encumbrance in an Application domain.

Law of Demeter

- The Law of Demeter limits the size of direct class reference sets.
- It states that if an object o1 refers to a object o2 through some method m of o1, then o2 must be:
  - the object itself (so o2 is actually o1)
  - an object referred to by the arguments of m
  - an object referred to by an attribute of o1
  - an object created by m
  - an object referred to by a global variable
- Weak law (vs. strong law): attribute of superclass
- In summary, an object should only send messages to objects it can directly reference.

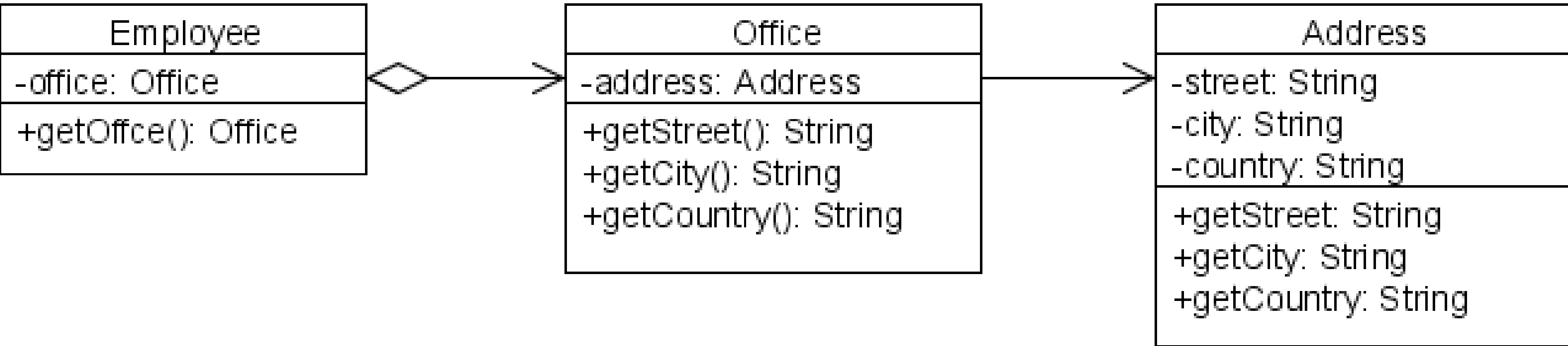
  Only talk to your immediate friends!

Example




```
String employeeStreet = this.office.getAddress().getStreet();
```

Example Fixed



String employeeStreet = this.office.getStreet();

Cohesion

- Measure of interrelatedness of features (attributes and methods) in an external interface of a class.

- Low (bad) cohesion
  - set of features that don't belong together
- High (good) cohesion
  - set of features that all contribute to the implementation

Three types of Cohesion

- Mixed-Instance Cohesion
  - Really really bad!
- Mixed-Domain Cohesion
  - Really bad!
- Mixed-Role Cohesion
  - Bad!

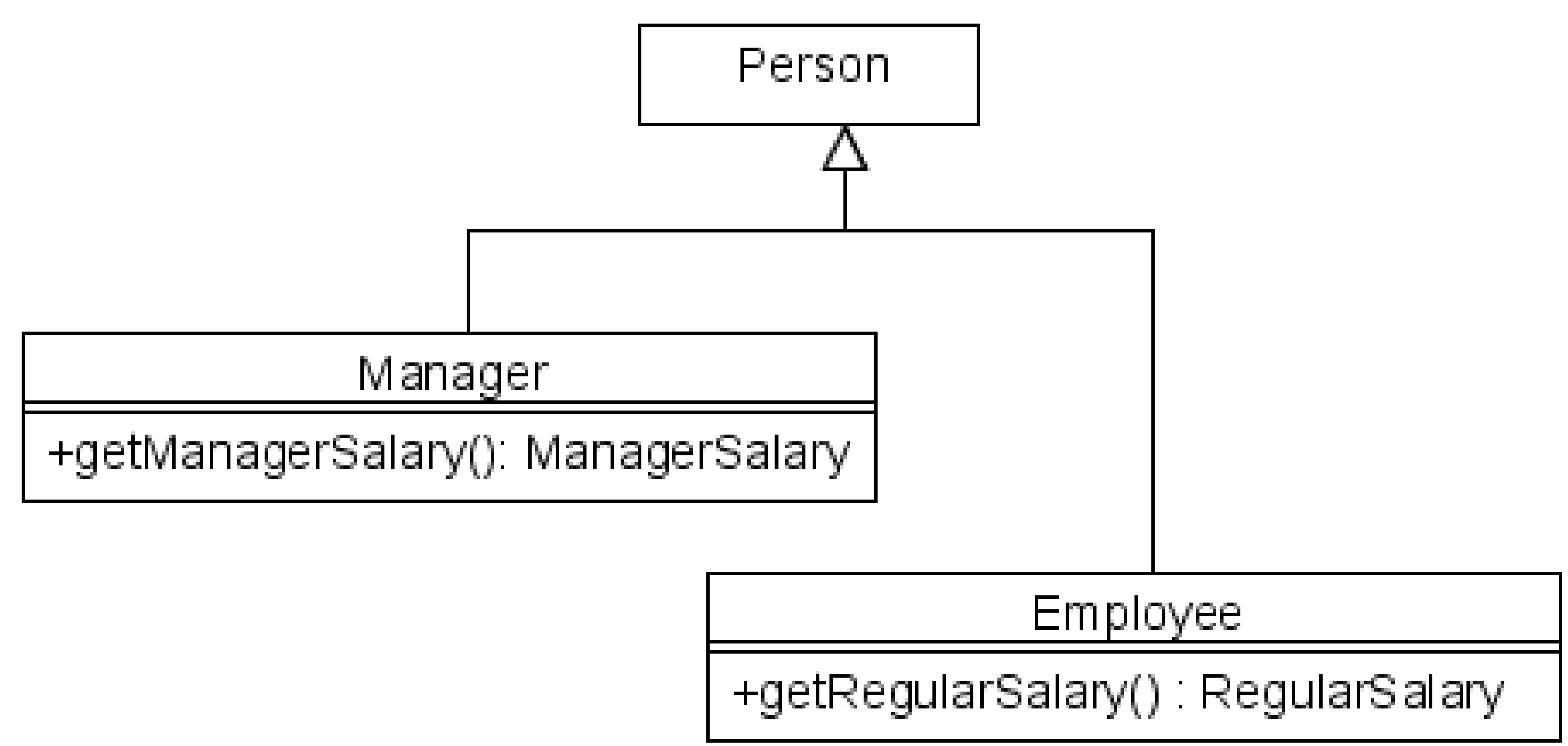
Mixed-Instance Cohesion

A class with mixed-instance cohesion has some features that are undefined for some objects of the class.

- Suppose you work at a company that has managers and non-managers.
- Managers receive a ManagerSalary and other employees receive a RegularSalary.
- Imagine employees are implemented using a Person class.
  - That class has a getManagerSalary() and a getRegularSalary() method which returns both types of salary.
- For each Person instance, we have features that won't be used.
  - Thus, Person is too broad

How to solve this?

- Usually means that there is a class hierarchy missing.
  - in our case, we should have classes Manager and Employee that inherit from a superclass Person.
- Now we won't have any unused features.



Extrinsic vs. Intrinsic

- The class B is extrinsic to A if A can be fully defined with no notion of B.
  - For example, Dog is extrinsic to Person, because in no sense does “Dog” capture some characteristic of Person.
- B is intrinsic to A if B captures some characteristic inherent to A.
  - For example, Dog is intrinsic to DogOwner, because “Dog” captures some characteristic of DogOwner.

Mixed-Domain Cohesion

A class with mixed-domain cohesion contains an element that directly encumbers the class in an extrinsic class of a different domain.

- In other words, a class should only encumber classes in other domains if they are intrinsic.
- For example, Invoice and Currency are two classes that exist in different domains.
- Since Currency is intrinsic to Invoice, there is no mixed-domain cohesion.
- However, Invoice and Printer, which also exist in different domains, would present mixed-domain cohesion since Printer is extrinsic to Invoice.

Mixed-Role Cohesion

A class C with mixed-role cohesion contains an element that directly encumbers the class with an extrinsic class that lies in the same domain as C.

- In other words, a class should only encumber classes if they are intrinsic.
- Lets go back to our example with Dog and Person.
- Although both classes exist in the same domain, they are not intrinsic
- As such, Dog should not encumber Person.
- Although this is the less serious cohesion problem, you must take it into account when designing for reusability.