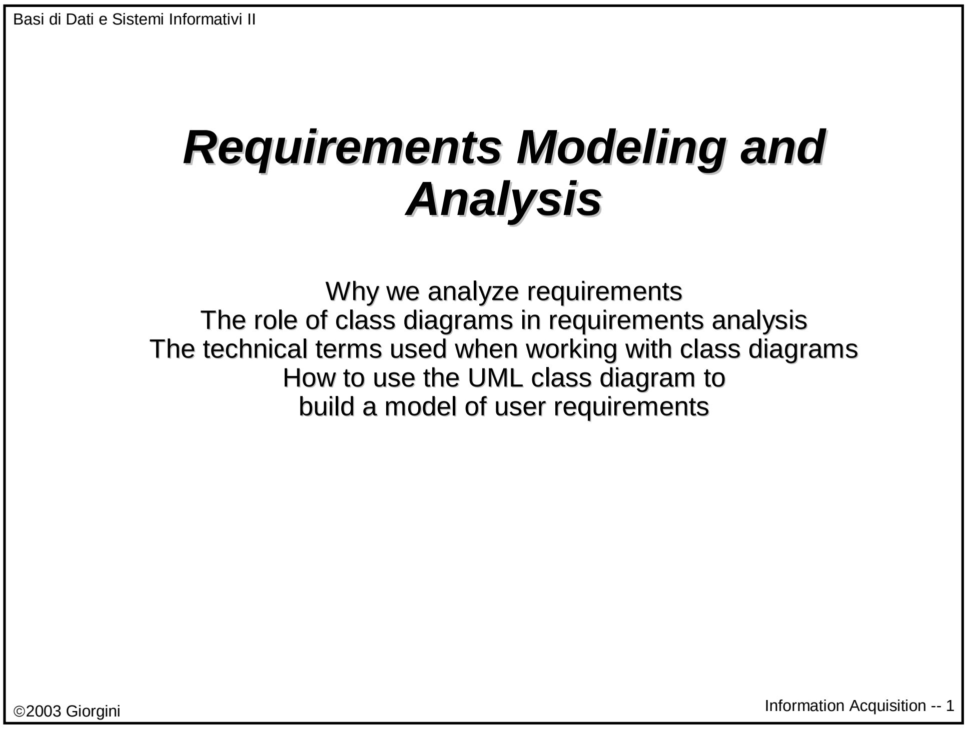
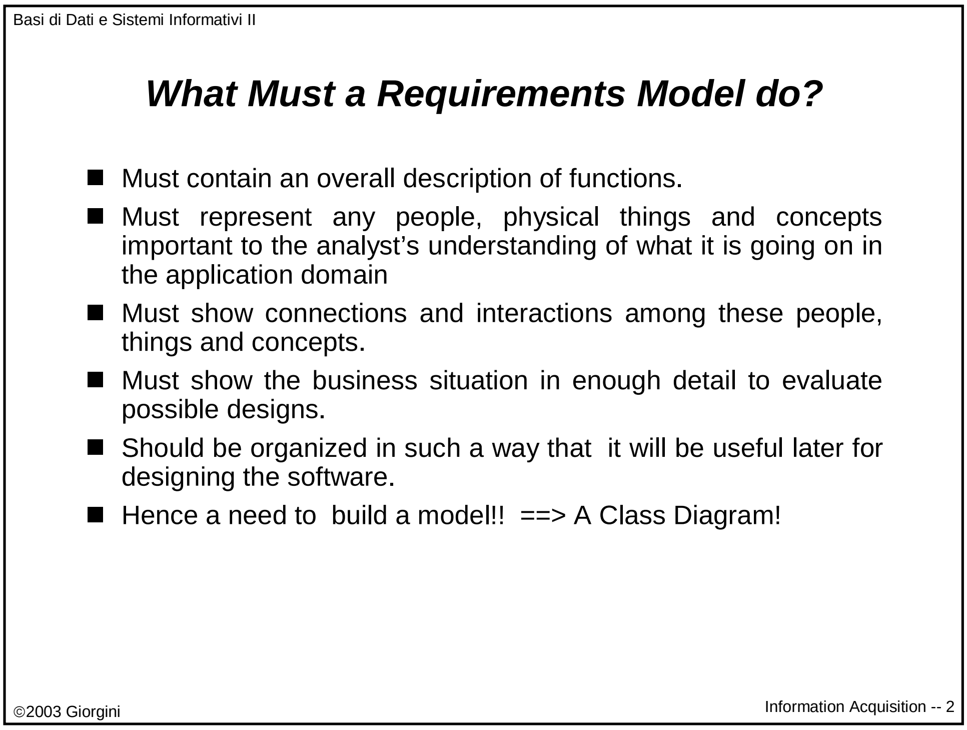


Requirements Modeling and Analysis

Why we analyze requirements
The role of class diagrams in requirements analysis
The technical terms used when working with class diagrams
How to use the UML class diagram to
build a model of user requirements



What Must a Requirements Model do?

- Must contain an overall description of functions.
- Must represent any people, physical things and concepts important to the analyst's understanding of what it is going on in the application domain
- Must show connections and interactions among these people, things and concepts.
- Must show the business situation in enough detail to evaluate possible designs.
- Should be organized in such a way that it will be useful later for designing the software.
- Hence a need to build a model!! ==> A Class Diagram!

Classes

- A class describes a group of objects with
  - ✓similar properties (attributes),
  - ✓common behavior (operations),
  - ✓common relationships to other objects,
  - ✓and common semantics.

Finding Classes from Use Cases

- Look for nouns and noun phrases
- They are only retained if they help to explain the nature or structure of the application domain.
- Deleting classes
  - ✓ Beyond the scope of the system
  - ✓ Refers to the system as a whole
  - ✓ Duplicates another class
  - ✓ Too vague
  - ✓ Too specific
  - ✓ Too tied up with physical inputs and outputs
  - ✓ Attribute
  - ✓ Operation

Finding Classes: Other sources

- Reviewing background information
- User representatives
- Analysis patterns

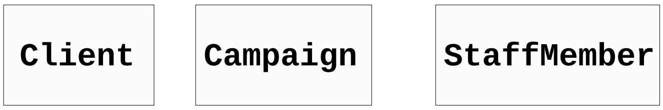
Classes

- *For example*, Agate will want to store information about all its staff members:
  - ✓current
  - ✓staff members who will be employed in the future.

- The object class `StaffMember` is a way of
  - ✓organizing all these object instances and
  - ✓defining the set of attributes and operations that apply to all staff

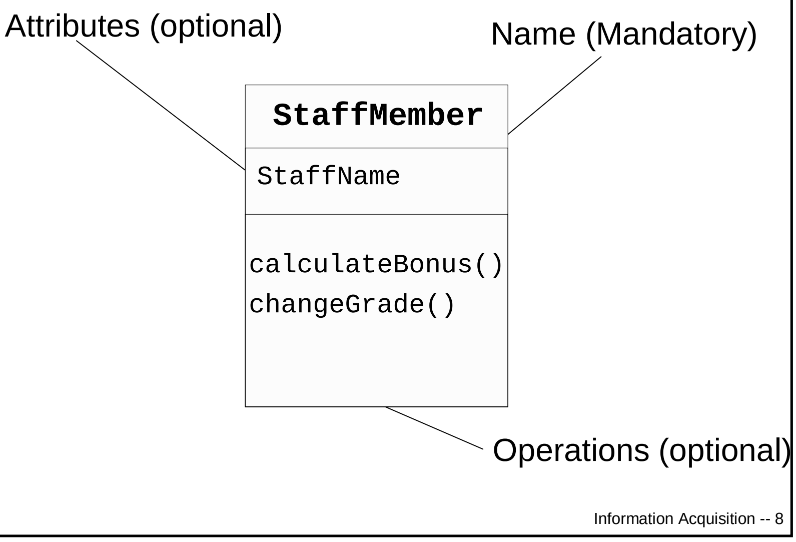
Names

- Every class must have a distinct name



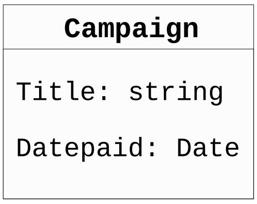
- In our Agate system, we shall use instances of these classes
  - ✓ Eg, when we Assign staff to work on a campaign, we shall use the classes `campaign` and `staffmember`
  - ✓ There will be one instance of `campaign` and several instances of `staffmember`

Basic Notation



Attributes and Operations

- Each object class will have ***attributes*** and ***operations***
- At this stage, operations may be more difficult to identify than attributes
- Attributes are the data we store about instances of the object
  - ✓ Each attribute has a *type*
- For example, `campaign` has attributes `Title` and `Datepaid`.

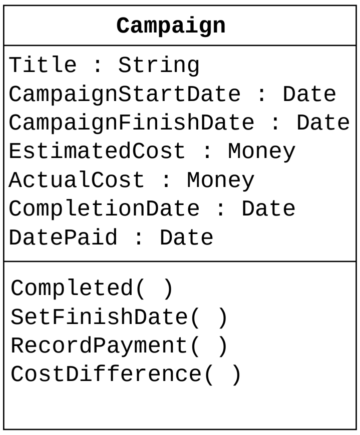


Operations

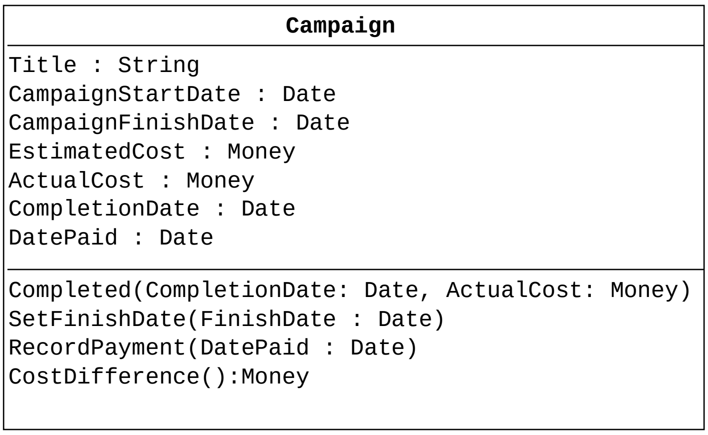
- Sometime found as actions <u>verbs in use case</u> descriptions (goal and task in the SR models)
- Some operations will carry out processes to change or do calculations with the attributes of an object.
- For example, the directors of Agate might want to know the difference between the estimated cost and the actual cost of a campaign
  - ✓ campaign would need an operation called *`CostDifference`*

Operations

- Some operations return a value, and the return value has to have a data type, like the attributes
- What do you think the return data type of `CostDifference( )` will be?



```
Campaign
Title : String
CampaignStartDate : Date
CampaignFinishDate : Date
EstimatedCost : Money
ActualCost : Money
CompletionDate : Date
DatePaid : Date
Completed( )
SetFinishDate( )
RecordPayment( )
CostDifference( )
```



```
Campaign
Title : String
CampaignStartDate : Date
CampaignFinishDate : Date
EstimatedCost : Money
ActualCost : Money
CompletionDate : Date
DatePaid : Date
Completed(CompletionDate: Date, ActualCost: Money)
SetFinishDate(FinishDate : Date)
RecordPayment(DatePaid : Date)
CostDifference():Money
```

Visibibilty

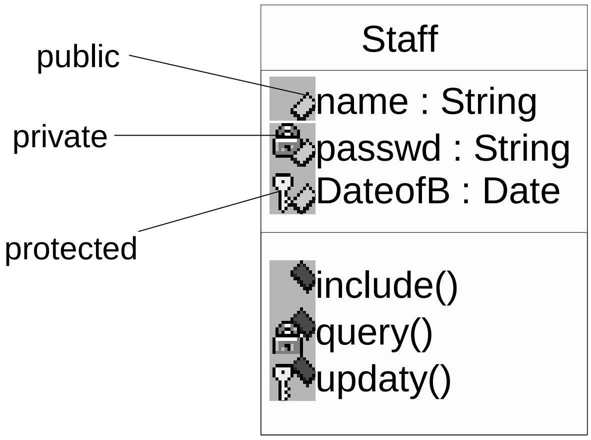
- Classifier: Classes, interfaces, components, nodes, use cases, subsystems

+ public: any outside classifier with visibility to the given classifier can use the feature

# protected: any descendant of the classifier can use the feature

- private: Only the classifier itself can use the feature

Rose Visibility



Relationships

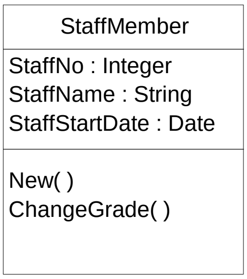
- Classes and objects do not exit in isolation from one another
- A relationship is a connection among things:
  - ✓ Generalization
  - ✓ Association
    - Aggregation
    - Composition
  - ✓ Dependency
  - ✓ Realization

Generalization Relationship

- A relationship between a general thing (called the <u>superclass</u> or <u>parent</u>) and more specific thing (called <u>subclass</u> or <u>child</u>).
- The child will inherit all the structure and behaviour of the parent.
- The child may add new structure and behaviour, or may modify the behaviour of the parent.

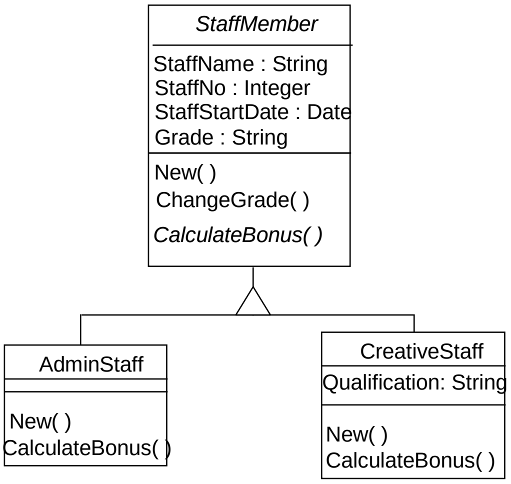
Generalization

- It may be that in a system like Agate’s we need to distinguish between different types of staff:
  - ✓ creative staff and administrative staff
  - ✓ and to store different data about them



- For example
  - ✓ administrative staff cannot be assigned to work on or manage a campaign
  - ✓ Creative staff have qualifications which we need to store
  - ✓ Creative staff are paid a bonus based on the work they have done
  - ✓ administrative staff are paid a bonus based on a percentage of salary

Generalization



Generalization

- The triangle linking the classes shows inheritance
- There will not be any instances of the class `StaffMember` in the system, they will all be either `AdminStaff` or `CreativeStaff`
  - ✓ `StaffMember` is an *abstract class*
- However, all instances of `AdminStaff` and `CreativeStaff` will have a `StaffNo`, `StaffName` and `StaffStartDate`. `CreativeStaff` will also have a `Qualification`

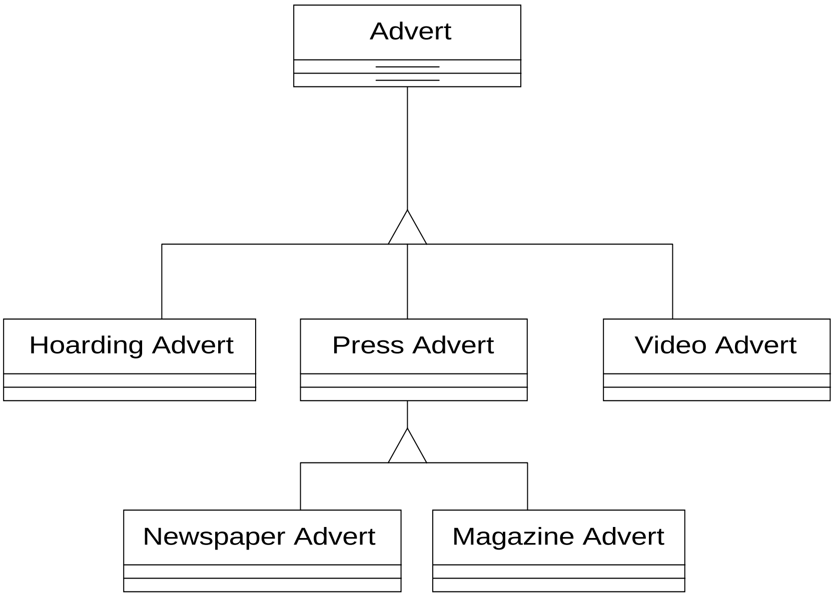
Generalization

- Similarly, the operation *`CalculateBonus( )`* will be declared in `StaffMember`, but it will be *overridden* in each of the sub-classes
- In the case of `AdminStaff`, it will use data from the `StaffGrade` which that member of staff is on to find out their salary rate and calculate the bonus
- In the case of `CreativeStaff`, it will use data from the campaigns that the member of staff has worked on to calculate the bonus
- When the same operation is implemented differently in different classes, each class is said to have its own ***method*** of implementing the operation

Finding Inheritance

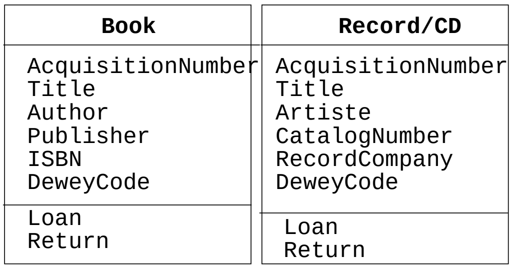
- Sometimes we find inheritance top-down:
  - ✓ we have a class, and we realize that we need to break it down into sub-classes which have different attributes and operations
- Here is a quote from a director at Agate:

  “*Most of our work is on adverts for the press, that's newspapers and magazines, for advertising hoardings, and for videos.*"



Finding Inheritance

- Sometimes we find inheritance bottom-up:
  - ✓ we have several classes and we realize that they have attributes and operations in common, so we group those attributes and operations together in a common super-class.
- Define a suitable base class and redraw this diagram



Book

AcquisitionNumber
Title
Author
Publisher
ISBN
DeweyCode

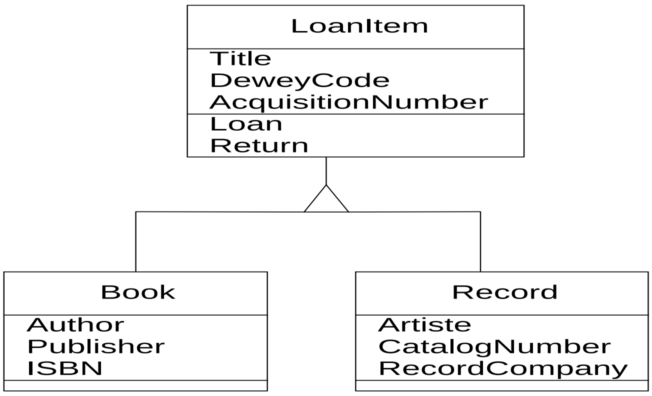
Loan
Return

Record/CD

AcquisitionNumber
Title
Artiste
CatalogNumber
RecordCompany
DeweyCode

Loan
Return

Finding Inheritance



LoanItem

Title
DeweyCode
AcquisitionNumber

Loan
Return

Book

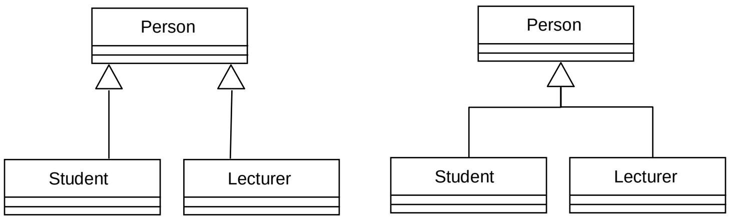
Author
Publisher
ISBN

Record

Artiste
CatalogNumber
RecordCompany

Generalization Notation

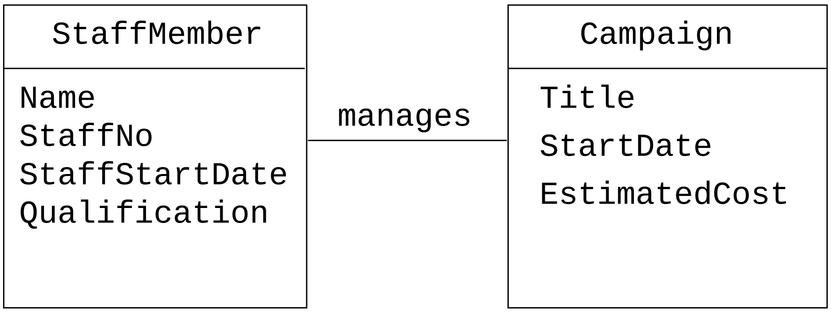
or this



Association Relationship

- It is a structural relationship, specifying that objects of one thing are connected to objects of another
- Have already seen that classes need to be linked to other classes in some way:
  - ✓ a staff member manages each campaign
- which can be converted into associations between classes.

Association


StaffMember

Name
StaffNo
StaffStartDate
Qualification

manages

Campaign

Title
StartDate
EstimatedCost

We have to determine the multiplicity of the associations

Association: Multiplicity

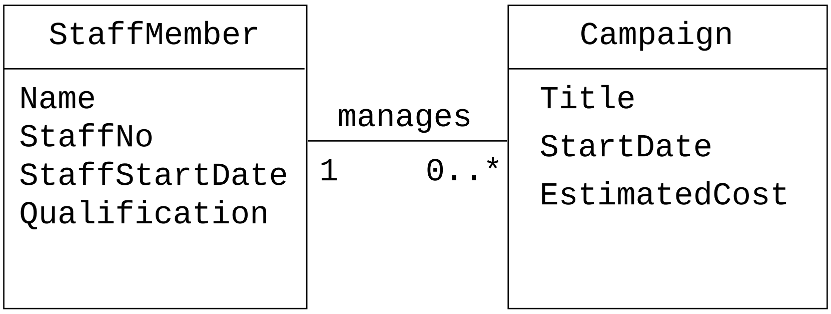
- Can a campaign exist without a member of staff to manage it?
- If yes, then the association is optional at the Staff end - zero or one
- If a Campaign cannot exist without a member of staff to manage it
  - ✓ then it is not optional
  - ✓ if it must be managed by one and only one member of staff then we show it like this - exactly one
- What about the other end of the association?
- Does every member of staff have to manage exactly one Campaign?
- No. So the correct multiplicity is zero or more.
  - ✓ Kerry Dent, a more junior member of staff, doesn’t manage any campaigns
  - ✓ Pete Bywater manages two

Multiplicity

- Multiplicity can be shown in the following ways:

Optional (0 or 1)	0..1
Exactly one	1
Zero or more	0..* = *
One or more	1..*
A range of values	1..6
A set of ranges	1..3,7..10,15,19..*

Association: Multiplicity



StaffMember

Name
StaffNo
StaffStartDate
Qualification

manages

1

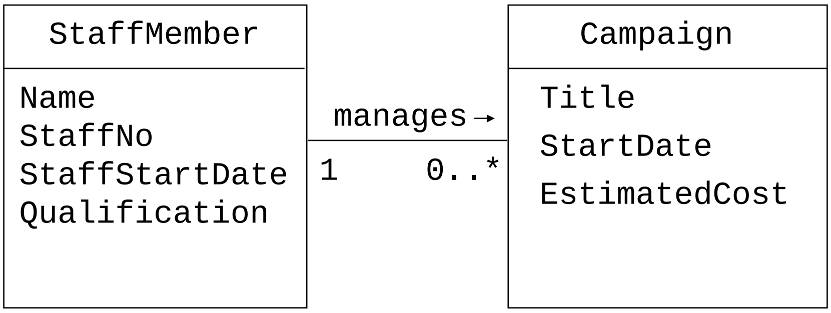
0..*

Campaign

Title
StartDate
EstimatedCost

Direction of Association

- The direction of the association label can be shown



StaffMember

Name
StaffNo
StaffStartDate
Qualification

manages→

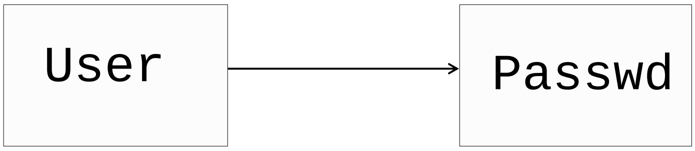
1 0..*

Campaign

Title
StartDate
EstimatedCost

Association Navigation

- Given a plain, unadorned association between two classes, it is possible to navigate from objects of one kind to objects of the other kind.
- However, there are circumstances in which you´ll want to limit navigation to just one direction.

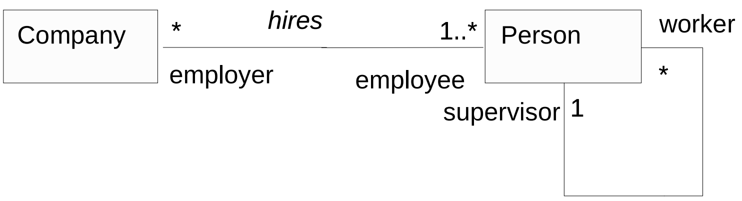


User

Passwd

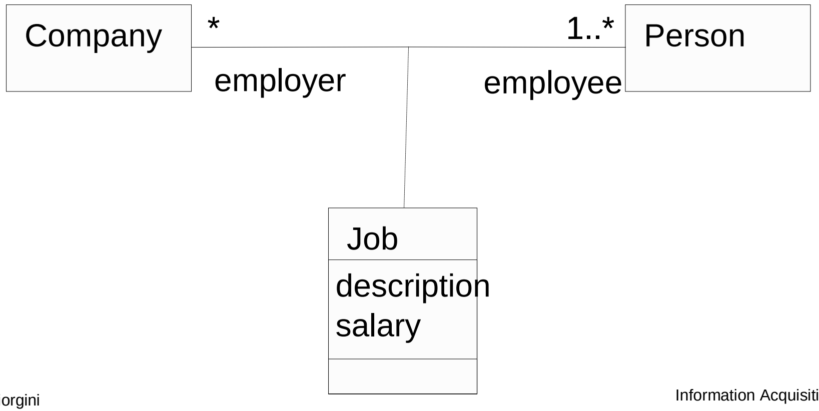
Association and Role

- You can explicitly name the role of a class in an association.
- The same class can play the same or different roles in other associations.



Association Class

- In an association between two classes, the association itself might have properties.

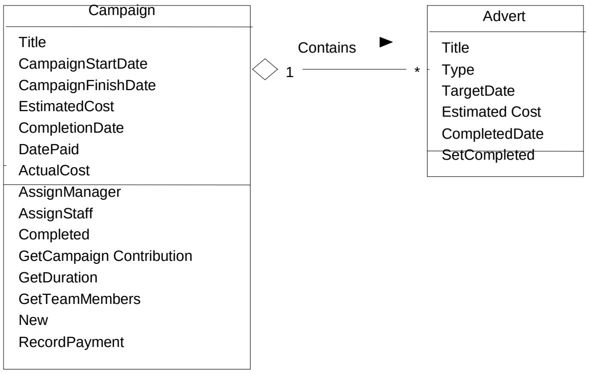


Association Class

- Attributes usually are placed in the class they describe.
- However, they might also be placed in an association class (ex. salary)

Aggregation Relationship

- “Has-a” or Whole/part relationship.
  - ✓one part represents a large thing (the “whole”), which consists of smaller things (the “parts”).



Campaign

Title
CampaignStartDate
CampaignFinishDate
EstimatedCost
CompletionDate
DatePaid
ActualCost

AssignManager
AssignStaff
Completed
GetCampaign Contribution
GetDuration
GetTeamMembers
New
RecordPayment

Contains ►

1

*

Advert

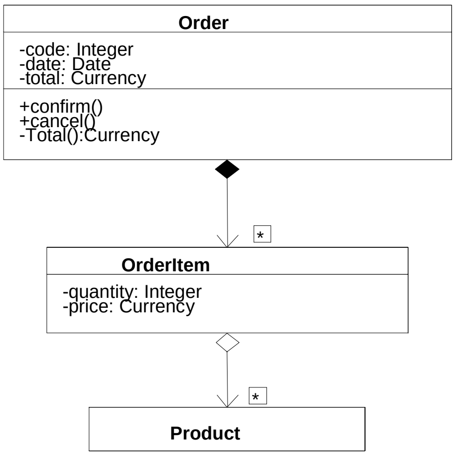
Title
Type
TargetDate
Estimated Cost
CompletedDate

SetCompleted

Composition Relationship

- It is a form of aggregation
- Strong ownership and coincident lifetime as part of the whole
- Once created they live and die with it
- The whole is responsible for the disposition of its parts, i.e. the composite must manage the creation and destruction of its parts.

Composition



Order

-code: Integer
-date: Date
-total: Currency

+confirm()
+cancel()
-Total():Currency

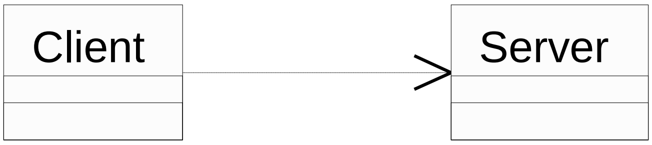
OrderItem

-quantity: Integer
-price: Currency

Product

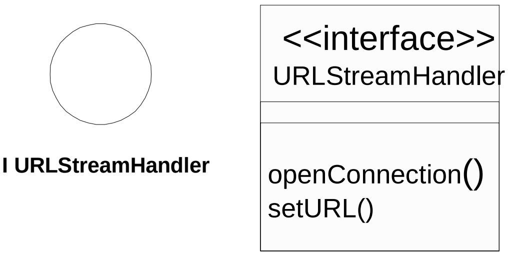
Dependency Relationship

- A dependency is a using relationship.
- Apply dependencies when you want to show one thing using another.
- Specifies that a change in the specification of one thing (server) may affect another thing that uses it (client), but not necessary the reverse.



Interfaces

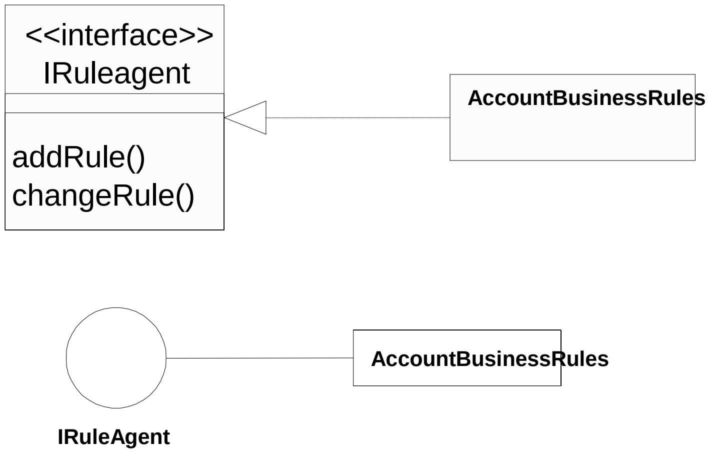
- A collection of operations that are used to <u>specify</u> a service of a class or a component.
- Graphically it is rendered as a circle. In its expanded form, an interface may be rendered as a stereotyped class.
- Separates the specification of a contract from its implementation



Realization Relationship

- One class specifies a contract that another class guarantees to carry out.
- Used in the context of interfaces to specify the relationship between an interface and the class or component that provides an operation or service for it.
- An interface may be realized by many classes, or a class may realize many components.

Realization



<<interface>>
IRuleagent

addRule()
changeRule()

AccountBusinessRules

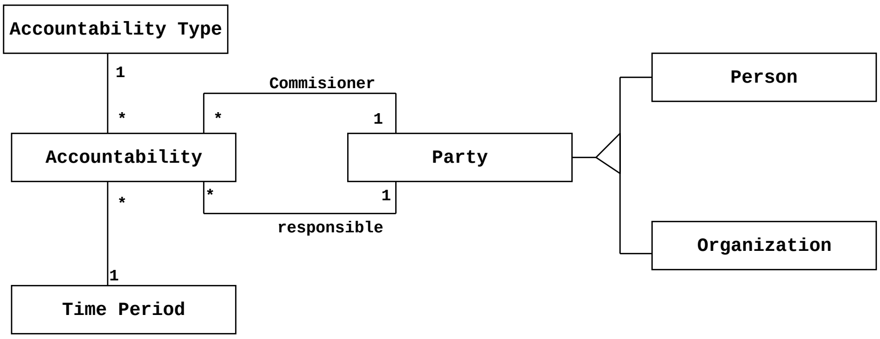
AccountBusinessRules

IRuleAgent

Analysis Patterns

- A structure of classes and associations that is found to occur over and over again in many different modelling situations.
- Each pattern can be used to communicate a general understanding about how to model a particular set of requirements
- Since a pattern may consist of whole structures of classes, the abstraction takes places at a higher level than is normally possible using generalization alone.

Analysis Patterns: an example



Additional Readings

- [Booch99] Booch, G. et all. The Unified Modeling Language User Guide. Chapters 4, 5, 8, 9, 10. Addison-Wesley.
- [Fowler97] Fowler, M. Analysis Patterns: Reusable Object Models, Addison-Wesley.
- [Bellin97] Bellin, D et all. The CRC Card Book. Addison-Wesley.