7SENG003W Advanced Software Design

Lecture 2 : Classes, Objects and Messages

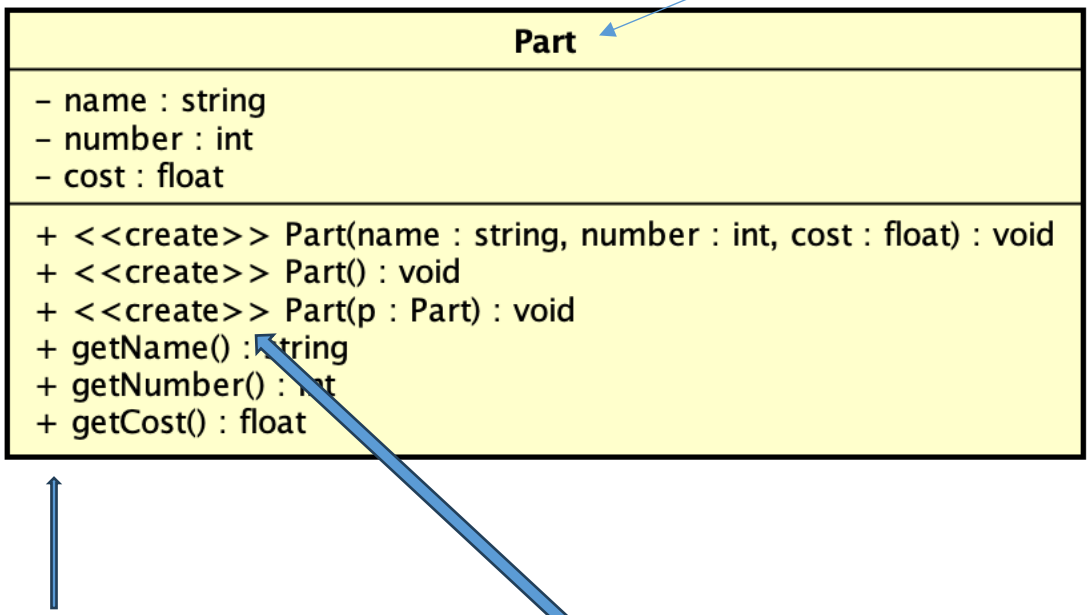
Simon Courtenage
University of Westminster

Aims for today

- Objects and classes
  - In Java and UML
  - How to relate objects using links
  - How to relate classes using associations – including how to add information to associations to make associations more specific
- Discuss how objects send messages to each other
  - How to show that diagrammatically

Classes

- The Part class in UML



Class name

attributes

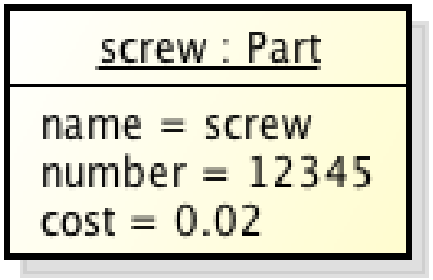
methods

Access levels:
'-' = private
'+' = public

<<..>> = stereotype – adds additional information

Objects

- An object is created when we instantiate a class
- In UML



- Shows identity of object (and optionally its class)
- Shows attributes and their state
- No need to show methods since these are defined by the class
- We model objects in order to show how different objects interact with each other
- Diagrams => Interaction diagrams
  - E.g.., communication diagram

Relationships between objects

- Recall the introduction of the CatalogueEntry class to hold information about types of parts

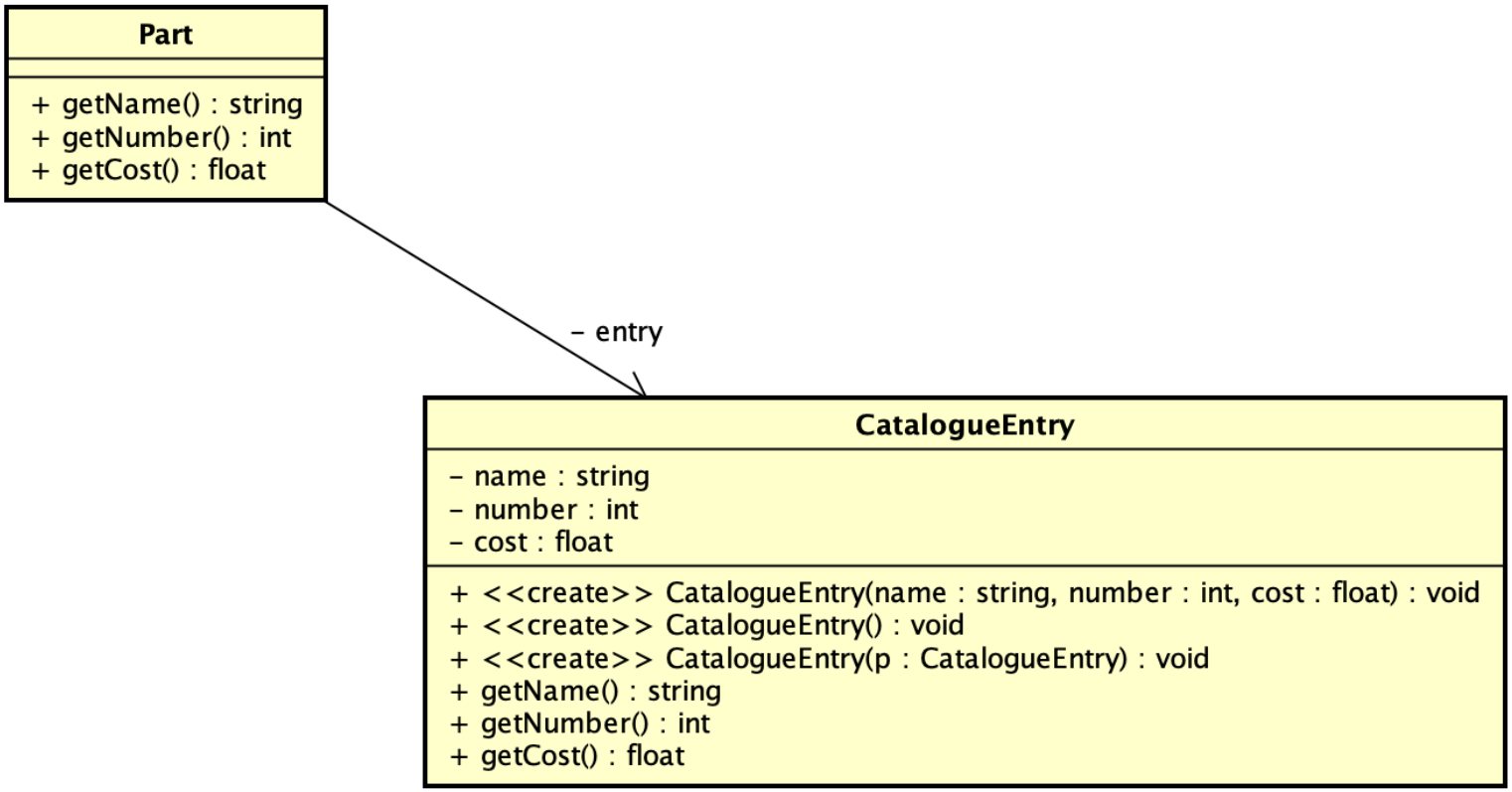
Here, the Part class says that objects created from this class will store CatalogueEntry objects

```
3   public class Part {
4       private CatalogueEntry entry;
5
6       Part(CatalogueEntry c) {
7           entry = c;
8       }
9       Part(Part p) {
10          entry = p.entry;
11      }
12
13      public String getName() {
14          return entry.getName();
15      }
16      public int getNumber() {
17          return entry.getNumber();
18      }
19      public double getCost() {
20          return entry.getCost();
21      }
22  }
```

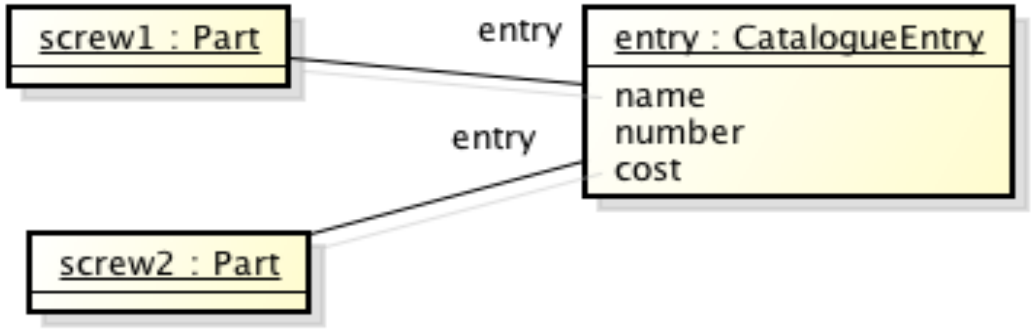
```
7   public class CatalogueEntry {
8       private String name;
9       private int number;
10      private double cost;
11
12      CatalogueEntry(String nm, int num, double c) {
13          name = nm;
14          number = num;
15          cost = c;
16      }
17      CatalogueEntry() { // default constructor
18          name = "";
19          number = -1;
20          cost = 0.0;
21      }
22
23      public String getName() {
24          return name;
25      }
26      public int getNumber() {
27          return number;
28      }
29      public double getCost() {
30          return cost;
31      }
32  }
```

How do we create a Part object? And what is the situation regarding objects after we have created it?

Updated class diagram



Modeling Object Relationships in UML



- Example of **object diagram**, showing how different objects are connected together so they can communicate/collaborate
- The relationship between the objects is an example of a *link relationship*
  - A link is a relationship between two objects, representing the fact that one object knows the identity of the other
  - If uni-directional then only one object knows about the other, but if bi-directional, then they both know of the other's existence
  - here, the direction is left unspecified (straight line with no arrows)

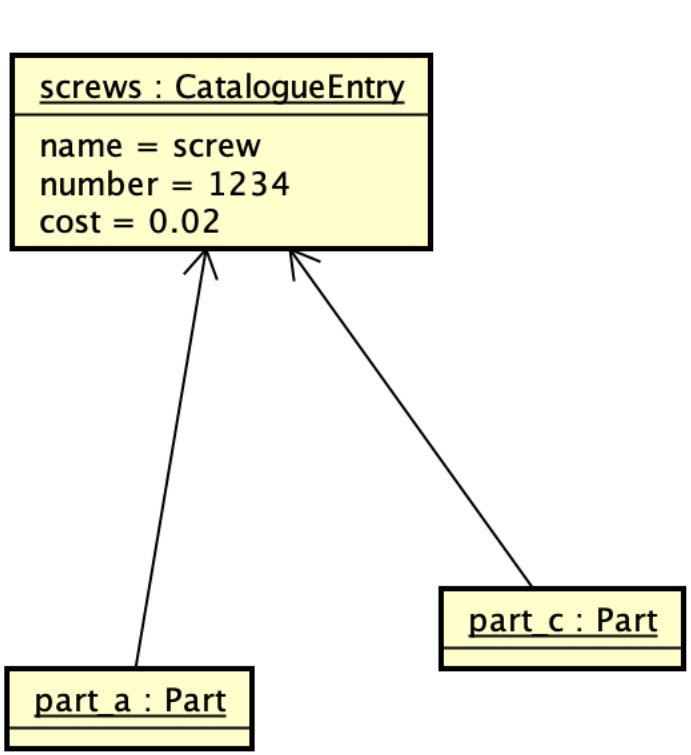
Creating Part objects

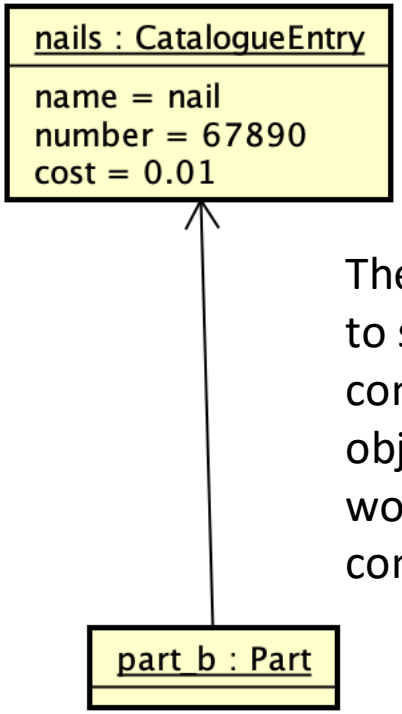
```
13      public static void main(String[] args) {
14          // create objects using 'new' operator and appropriate constructor
15          CatalogueEntry screws = new CatalogueEntry( nm:"screw", num:1234, c:0.02);
16          CatalogueEntry nails = new CatalogueEntry( nm:"nail", num:67890, c:0.01);
17
18          Part part_a = new Part( c:screws);
            Part part_b = new Part( c:nails);
20          Part part_d = new Part( p:part_a);
21
22          System.out.println("part_d part name is " + part_d.getName());
23
24      }
25  }
```

- This code creates two CatalogueEntry objects and then passes them to the constructors of Part objects
- What is the end result concerning object relationships?

Object diagram

```
13      public static void main(String[] args) {
14          // create objects using 'new' operator and appropriate constructor
15          CatalogueEntry screws = new CatalogueEntry( nm:"screw", num:1234, c:0.02);
16          CatalogueEntry nails = new CatalogueEntry( nm:"nail", num:67890, c:0.01);
17
18          Part part_a = new Part( c:screws);
            Part part_b = new Part( c:nails);
20          Part part_d = new Part( p:part_a);
21
22          System.out.println("part_d part name is " + part_d.getName());
23
24      }
25  }
```





The purpose of this diagram is to show the structural connections between the objects. In order for objects to work together, they must be connected in some way.

(Note: in (my version of) Astah, this diagram is a class diagram)

Message Passing (Communication Diagram)

- Links can be used to simply connect objects with no restriction placed on their direction – to be used simply as a kind of communications channel between two objects

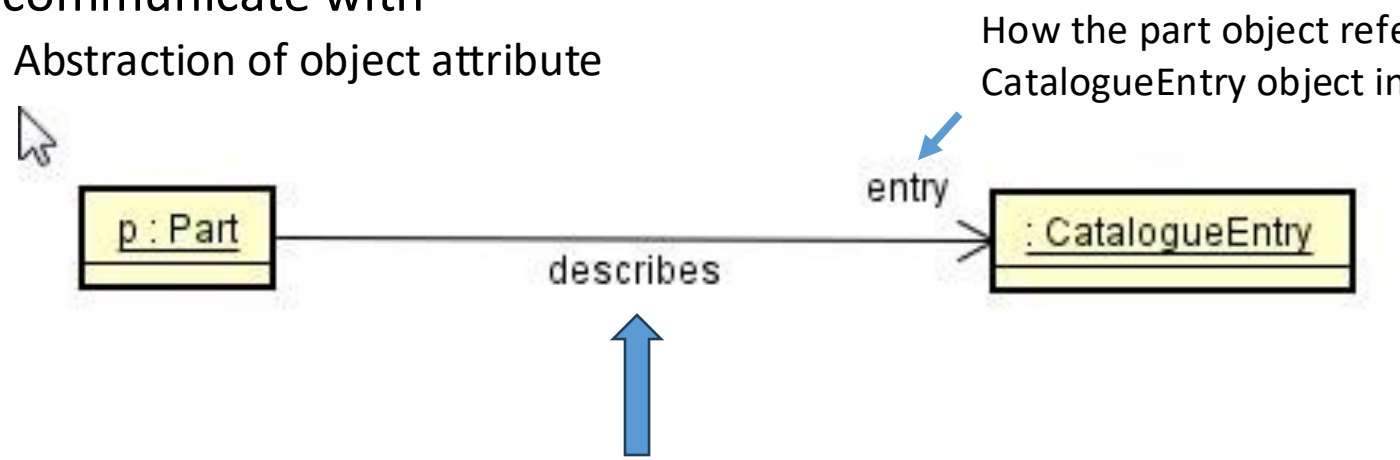
- In UML, objects communicate by **sending messages** – a message is a request from one object to another object to perform some service
- If an object has a link to another, it can send messages to the other object
- E.g., a link between Part and CatalogueEntry allows the Part object to send the getCost message to the CatalogueEntry Object
- The message should be one that the receiving object should understand! (I.e., correspond to a method in its public interface)

Links are represented as attributes in classes in code

- An attribute stores the identity of an object

```
3   public class Part {
        private CatalogueEntry entry;
5
```

- A Link is a communications channel – identity of the thing you want to communicate with
  - Abstraction of object attribute

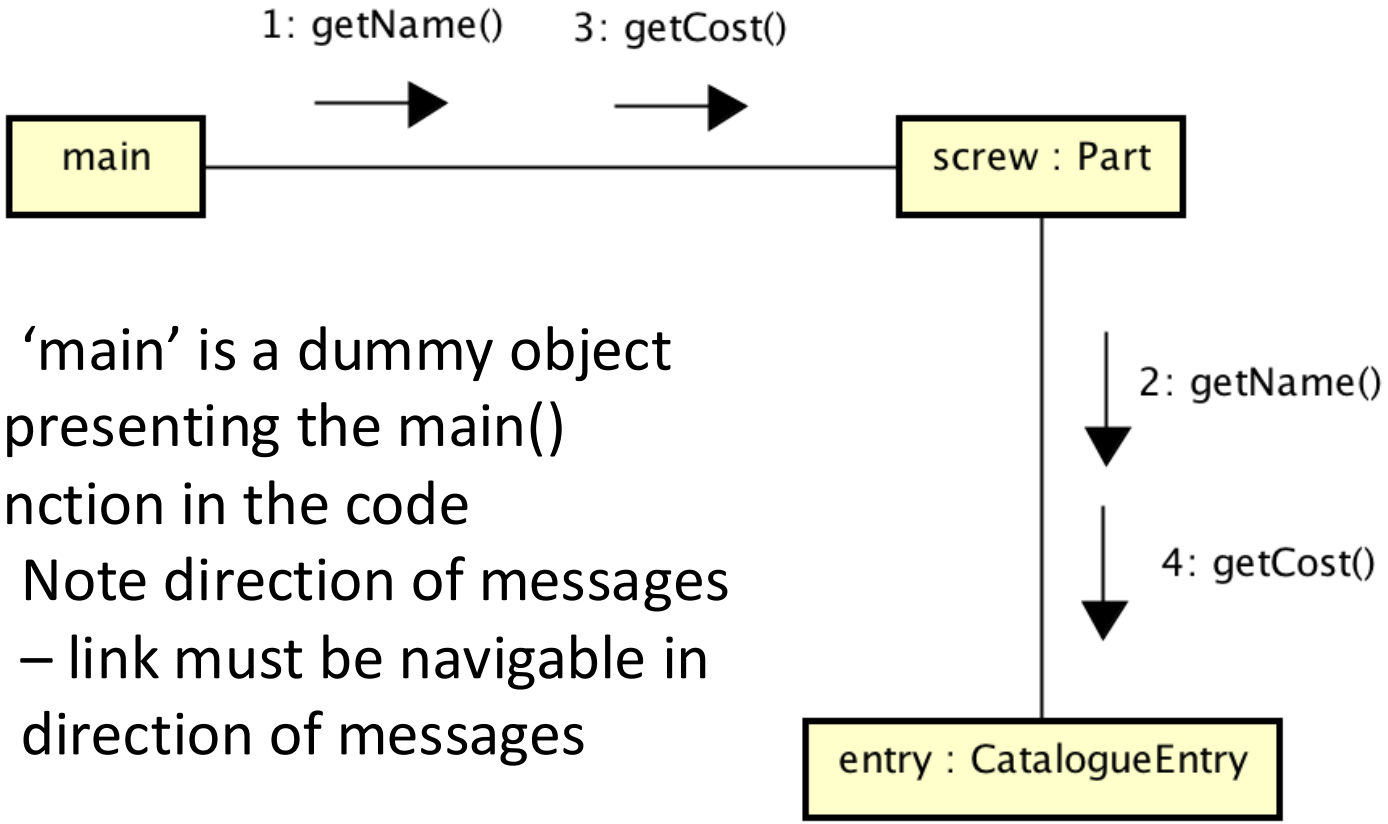


Sending messages in code

```
11  public class Inventory {
12
13      public static void main(String[] args) {
14          // create objects using 'new' operator and appropriate constructor
15          CatalogueEntry screws = new CatalogueEntry( nm:"screw", num:1234, c:0.02);
16
17          Part screw = new Part( c:screws);
18
19          System.out.println("part_d is " + screw.getName() +
20                  " and costs " + screw.getCost());
21
22
23      }
24  }
```

What messages are being sent and in what order? (starting from main() function)

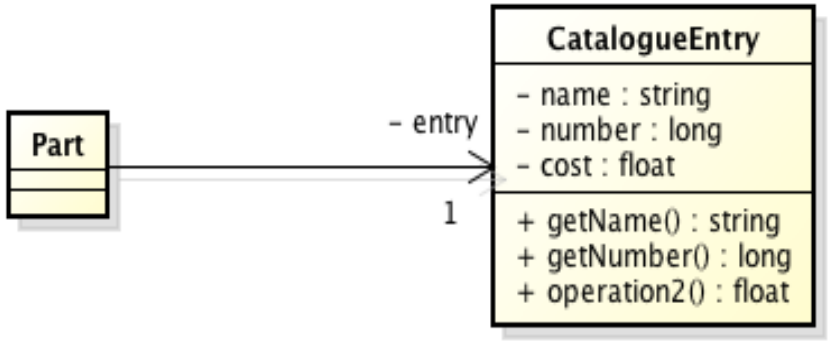
Communication Diagram example



- ‘main’ is a dummy object representing the main() function in the code
- Note direction of messages
  – link must be navigable in direction of messages

Class Relationship - Association

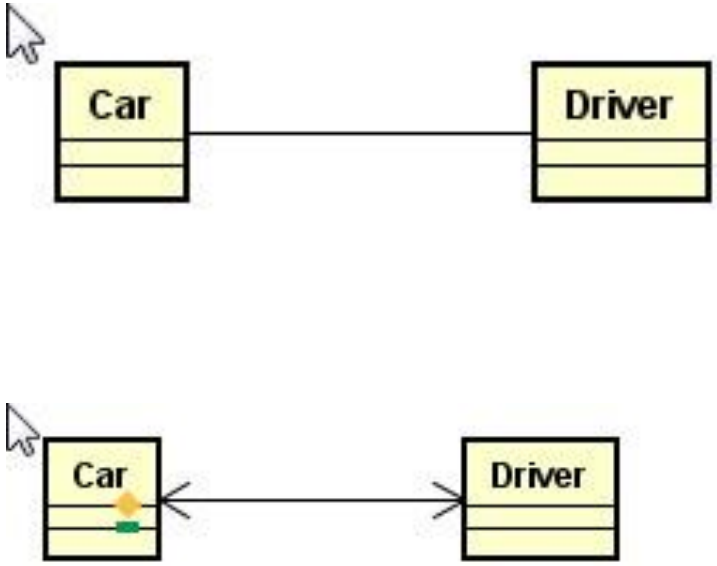
- If objects of one class are linked to objects of another class, then we should represent this fact at the level of classes, not just at the level of objects



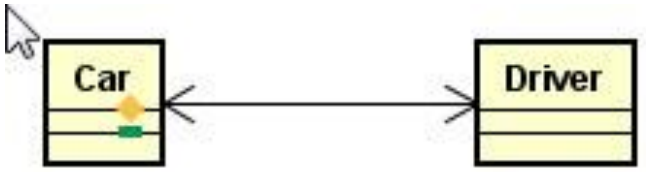
- In this case, the links are uni-directional (from Parts to CatalogueEntry) so the class relationship is uni-directional (in the same direction)
- Note 1: many Part objects link to one CatalogueEntry object = ONE association from ONE Part class to ONE CatalogueEntry class
- Note 2: we represent attributes with class type using associations, not with attributes in the class 'box'
- Note 3: if Part object sends getName() message to CatalogueEntry object, then getName() must be in public interface of CatalogueEntry class

Bi-directional associations

- These two diagrams differ only in the navigability of the association
- The first has no information about navigation
- The second has bi-directional navigation
  - Bi-directional navigation implies a pair of attributes – one in each class – that are inverses of each other



Implementing bi-directional associations




```
11  public class Driver {
12      private Car car;
13      String name;
15      Driver(String n) {
16          name = n;
17      }
19      void setCar(Car c) {
20          car = c;
        }
        String getName() { return name; }
        Car getCar()     { return car; }
23  }
```

```
11  public class Car {
12      Driver driver;
13      String model;
15      Car(String m, Driver d) {
16          model = m;
17          driver = d;
        }
        Driver getDriver() { return driver; }
        String getModel()  { return model; }
20  }
```

```
13      public static void main(String[] args) {
14          Driver d = new Driver( n:"Jo");
15          Car c = new Car( m:"VW Polo",d); // set up association in one direction
16
17          d.setCar(c); // now set up association in the other direction
18      }
```

Containment

- Most association relationships between classes are instances of some form of containment, where one object ‘contains’ another
  - E.g., shopping basket and product
- UML has two ways to represent ‘containment’-type relationships
  - Aggregation and Composition
  - Both are specific forms of association – specifying more information about the nature of the association relationship
  - Differences are in whether or not sharing is allowed and in dependency in terms of lifetime

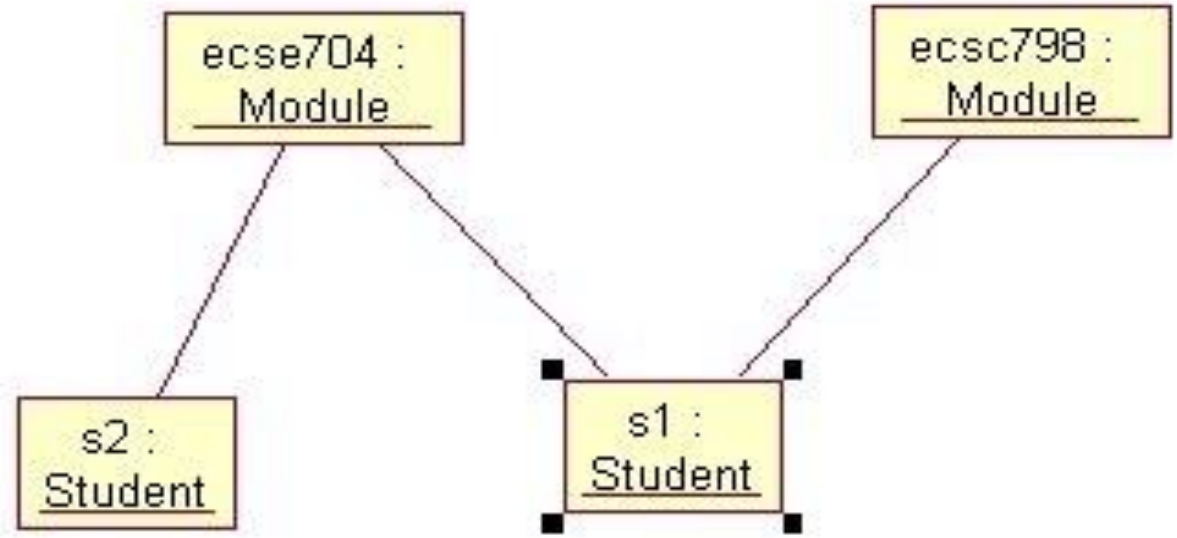
AGGREGATION

- Association is a fairly weak relationship – it just shows that two classes are related
- A slightly stronger version of association is aggregation – which denotes a form of “ownership”

- There isn’t much difference between “ordinary” association and aggregation – in fact, aggregation has been called “modelling placebo” for this reason

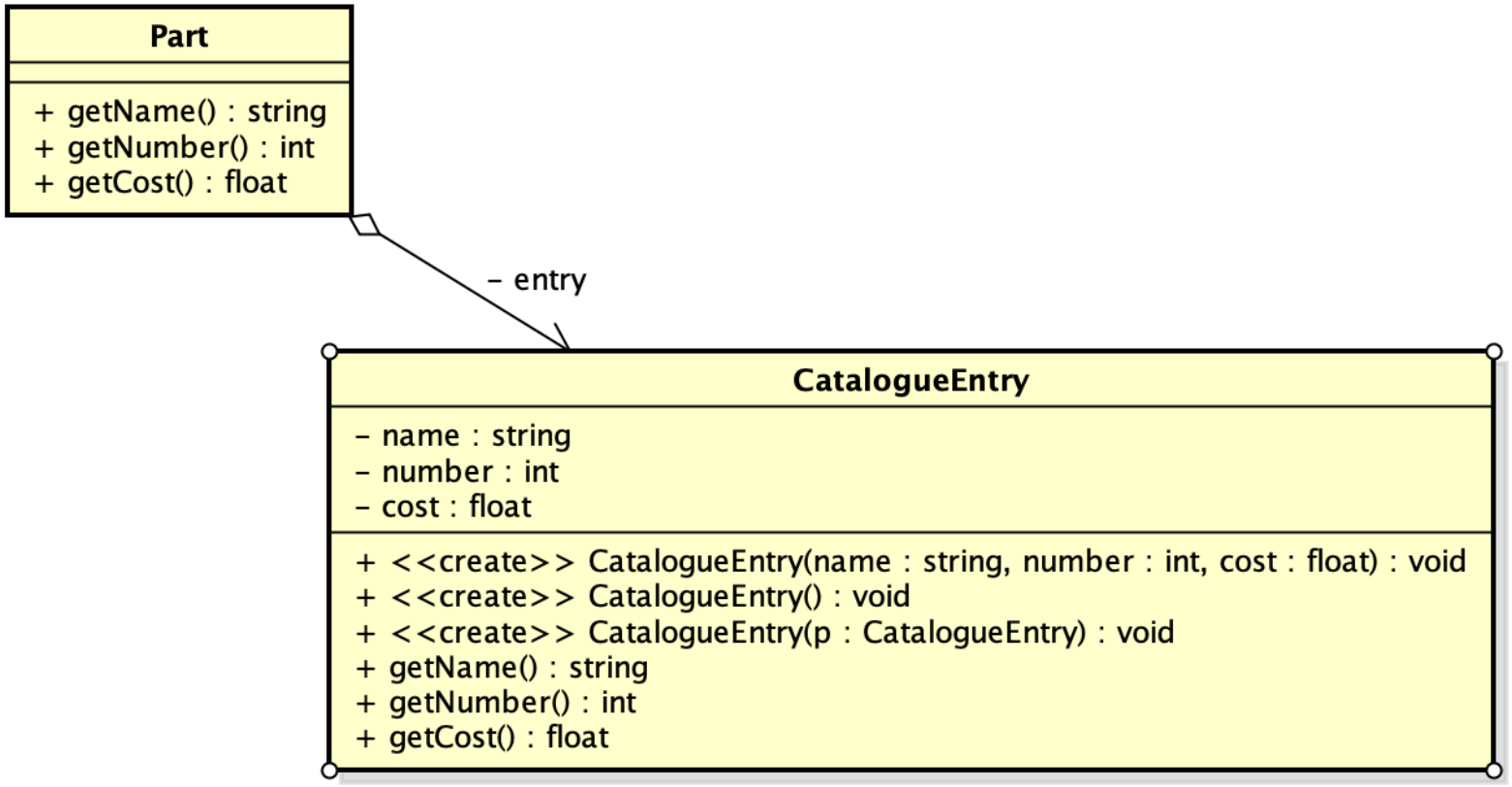
AGGREGATION - PROPERTIES

- Aggregation means that objects of one class “own” or possess the identity of object(s) from another class
- They may share the 'owned' object and if they are destroyed, it doesn't mean the 'owned' object is destroyed



Aggregation code example

- The Part – CatalogueEntry relationship is an example of Aggregation, since many Part objects may have links to the same CatalogueEntry object. So we can change the class diagram to show this.



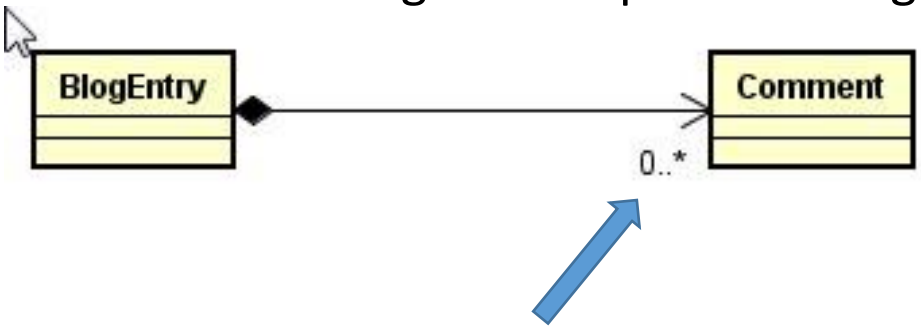
COMPOSITION

- Composition is another version of association that denotes ownership – stronger than aggregation
- No sharing
- The 'owned' object is destroyed when the 'owning' object is destroyed

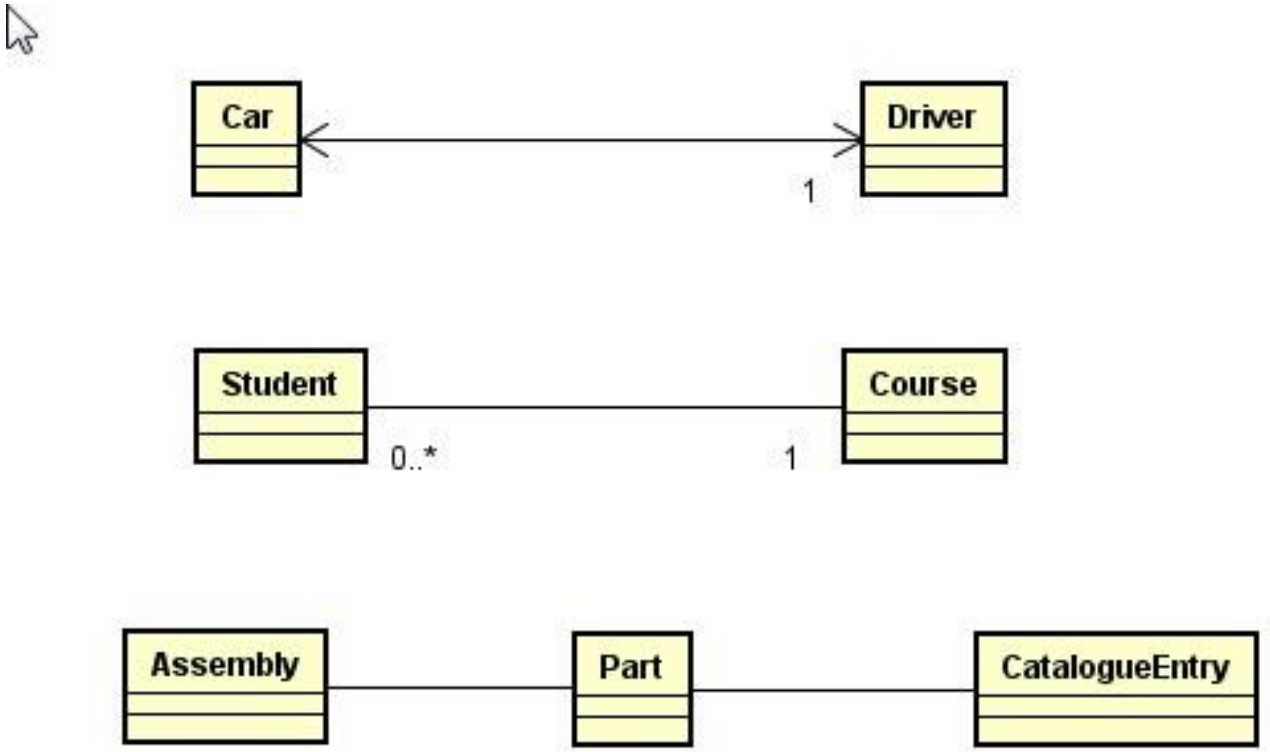
Multiplicities

- We can also specify on an association the degree of the relationship
- How many comments does a blog entry have?

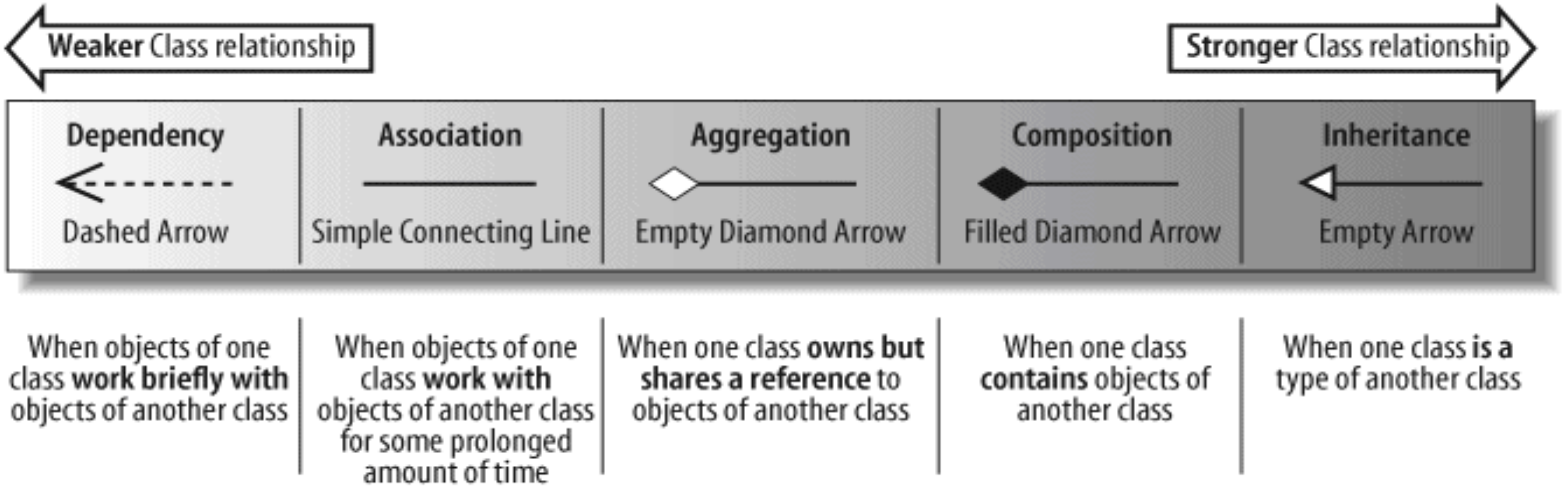
- It might have none (0), 1 or many – there are lots of possibilities
- How can we show on the diagram the possible range of comments?



Multiplicity - examples



CLASS RELATIONSHIPS



(From *Learning UML 2.0*, Miles and Hamilton, pub. O'Reilly, 2008)

Further reading

- https://www.visual-paradigm.com/guide/uml-unified-modeling-language/uml-aggregation-vs-composition/ Association vs Aggregation vs Composition

Updated lecture plan

1. Classes and objects
2. Message passing and object interaction
3. Use Case modelling – modelling requirements
4. Use Case Realisation – turning Use Cases into OO Designs
5. Inheritance and polymorphism
6. READING WEEK
7. Inheritance and polymorphism (contd.)
8. Design patterns and Design Heuristics – refining designs
9. Design Patterns (contd.)
10. Statecharts
11. Software Quality Assurance
12. Testing and Performance Evaluation