7SENG003W Advanced Software Design

Lecture 2 : Classes, Objects and Messages

Simon Courtenage
University of Westminster

Aims for today

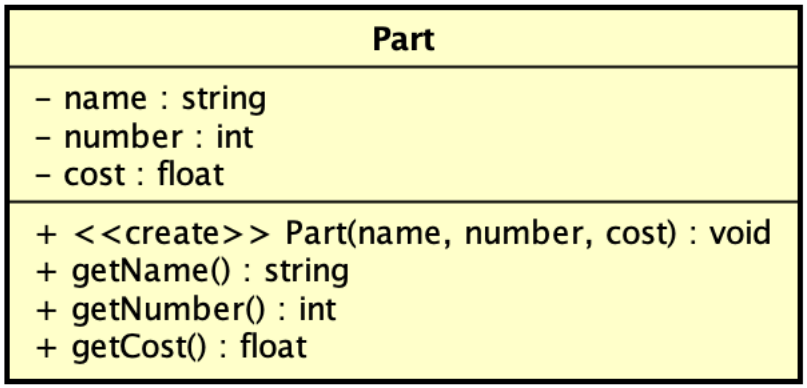
- Objects and classes
  - In C# and UML
  - How to relate objects using links
  - How to relate classes using associations – including how to add information to associations to make associations more specific
- Discuss how objects send messages to each other
  - How to show that diagrammatically

Classes

- The Part class in C#

```
1   using System;
2   namespace HelloWorld
3   {
4       public class Part
5       {
6           private string name;
7           private int number;
8           private double cost;
9
10          public string Name {
11              get { return name; }
12          }
13          public int Number { get { return number; } }
14          public double Cost { get { return cost; } }
15
16          public Part(string nm, int num, double c)
17          {
18              this.name = nm;
19              this.number = num;
20              this.cost = c;
21          }
22      }
23  }
24
```

- The Part class in UML

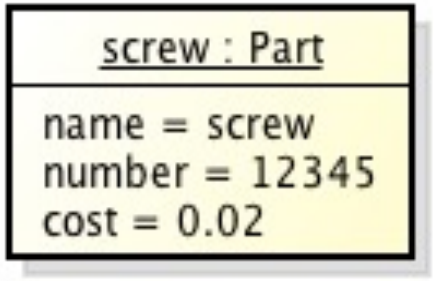


Objects

- An object is created when we instantiate a class
- In C#

```
var p = new Part("screw",12345,0.02);
```

- In UML



- 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

```
1   using System;
2   namespace Inventory
3   {
4       public class CatalogueEntry
5       {
6           private string name;
7           private int number;
8           private double cost;
9
10          public string Name
11          {
12              get { return name; }
13          }
14          public int Number { get { return number; } }
15          public double Cost { get { return cost; } }
16
17          public CatalogueEntry(string nm, int num, double c)...
23
24          public CatalogueEntry()...
30      }
31  }
32
```

```
1   using System;
2   namespace Inventory
3   {
4       public class Part
5       {
6           CatalogueEntry Entry { get; }
7           public string Name   { get { return Entry.Name;  } }
8           public int    Number { get { return Entry.Number;} }
9           public double Cost   { get { return Entry.Cost;  } }
10
11          public Part(CatalogueEntry c)
12          {
13              Entry = c;
14          }
15
16          public Part(Part p) : this(p.Entry) { }
17      }
18  }
19
```

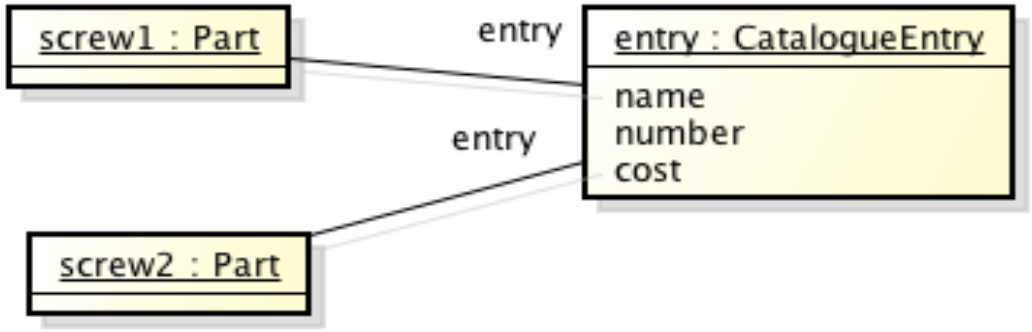
How do we create a Part object? And what is the situation regarding objects after we have created it?

Creating Part objects

```
2
3   namespace Inventory
4   {
5       class Program
6       {
7           static void Main(string[] args)
8           {
9               var screw = new CatalogueEntry("screw", 12345, 0.01);
10              var p = new Part(screw);
11              var p2 = new Part(screw);
12              Console.WriteLine($"A {p.Name} costs £{p.Cost}");
13              Console.WriteLine($"A {p2.Name} costs £{p2.Cost}");
14          }
15      }
16  }
17
```

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

Object Relationships in UML



- Example of communication 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)

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

- 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 attributes

- An attribute stores the identity of an object

```
public class Part
{
    CatalogueEntry Entry { get; }
    // rest of class...
```

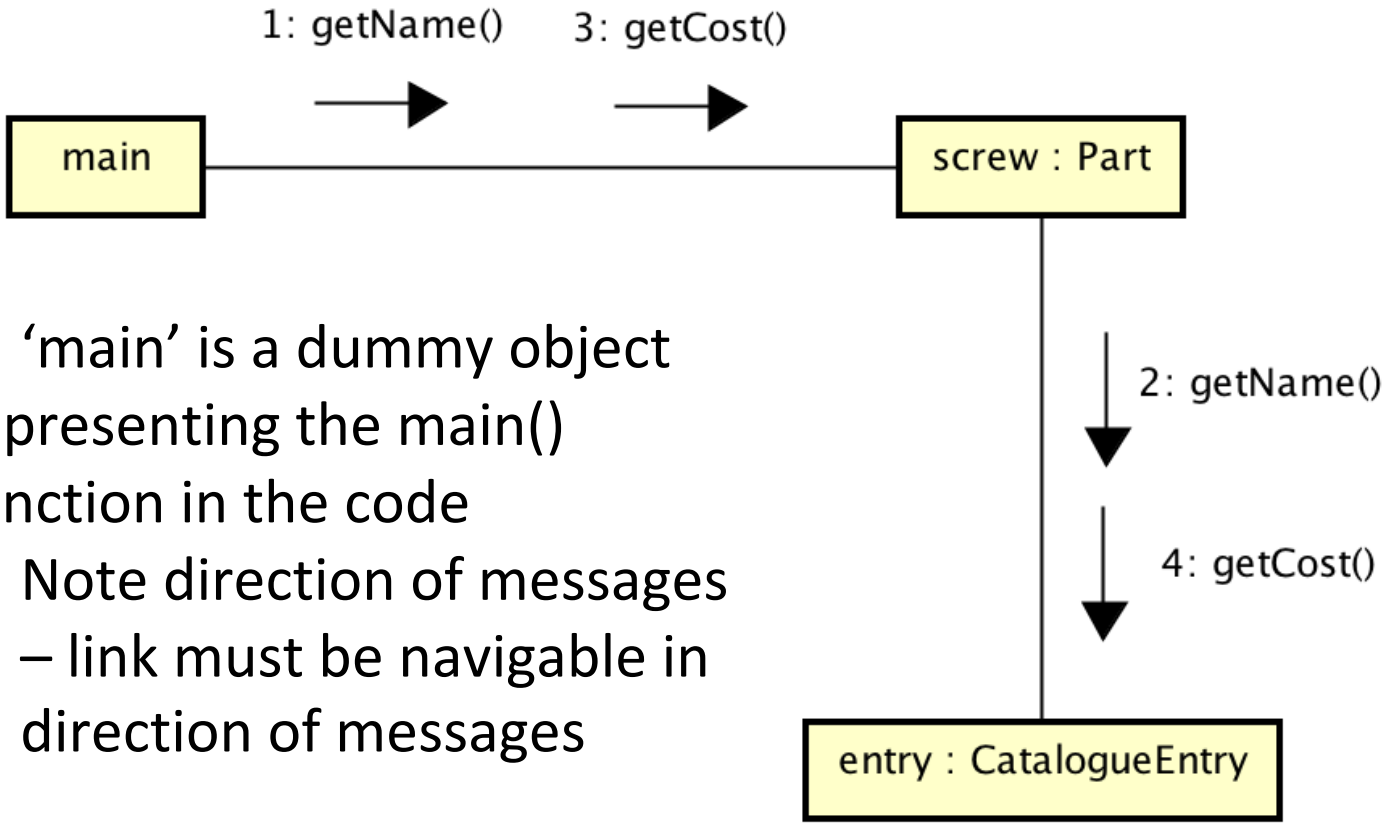
- A Link is a communications channel – identity of the thing you want to communicate with
  - Abstraction of object attribute

Sending messages in code

```
2
3   namespace Inventory
4   {
5       class Program
6       {
7           static void Main(string[] args)
8           {
9               var screw = new CatalogueEntry("screw", 12345, 0.01);
10              var p = new Part(screw);
11              var p2 = new Part(screw);
12              Console.WriteLine($"A {p.Name} costs £{p.Cost}");
13              Console.WriteLine($"A {p2.Name} costs £{p2.Cost}");
14          }
15      }
16  }
17
```

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

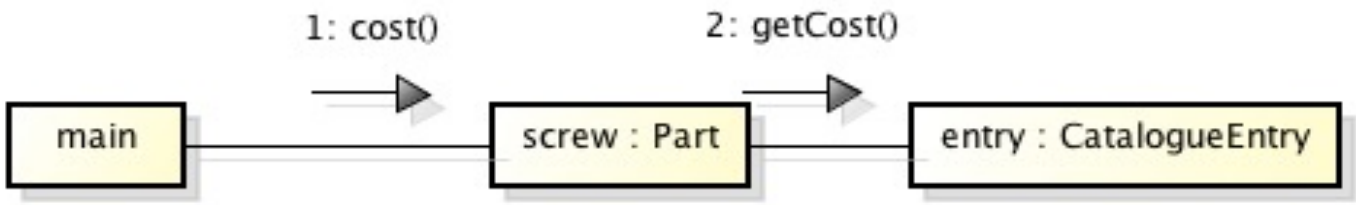
Message Syntax

- Messages have a name, such as “cost”, and are written in a syntax similar to function calls
- The name of the message will typically match one of the operations defined for the object receiving the message

- Links are viewed as **communication channels**
  - If an object is connected to another object via a link, it can send messages to that object
  - Note: the link must be navigable in the same direction that the message is being sent
  - If there is no link between two objects, then no messages can be sent

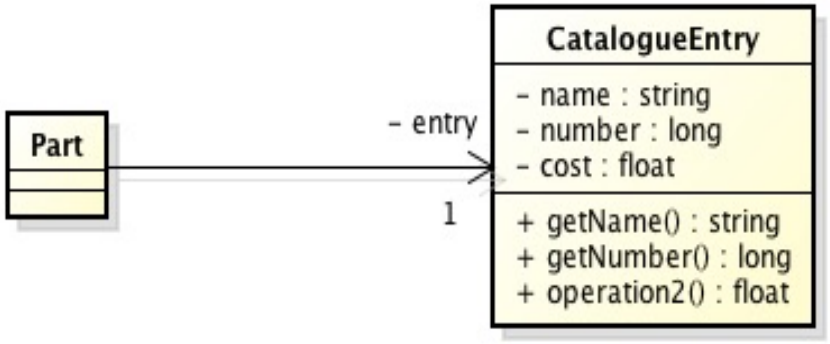
Sharing Responsibility - Delegation

- Data is distributed across the network of objects
  - An object will sometimes receive a message asking it do something, but doesn't itself hold all the data needed to respond
  - E.g, the part object when it receives the cost() message. The cost data is stored in the catalogue entry object
    - So the Part object must send a message to the Catalogue Entry object to retrieve the cost data in order to generate its own response



Class Relationships

- 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


```
4     public class Car
5     {
6         private string model;
7         private Driver driver;
8         public Driver MyDriver { get;  }
9         public string Model { get { return model; }  }
10        public Car(string m,Driver d)
11        {
12            this.model = m;
13            this.driver = d;
14        }
15    }
```

```
4     public class Driver
5     {
6         private string name;
7         private Car car;
8         public string MyName { get; }
9         public Car MyCar {
10            get { return car;  }
11            set { car = value; }
12        }
13        public Driver(string n) { this.name = n; }
14    }
```

```
5     class Program
6     {
7         static void Main(string[] args)
8         {
9             Driver d = new Driver("Charlie");
10            Car c = new Car("Ford Fiesta", d);
11            d.MyCar = c;
12        }
13    }
```

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

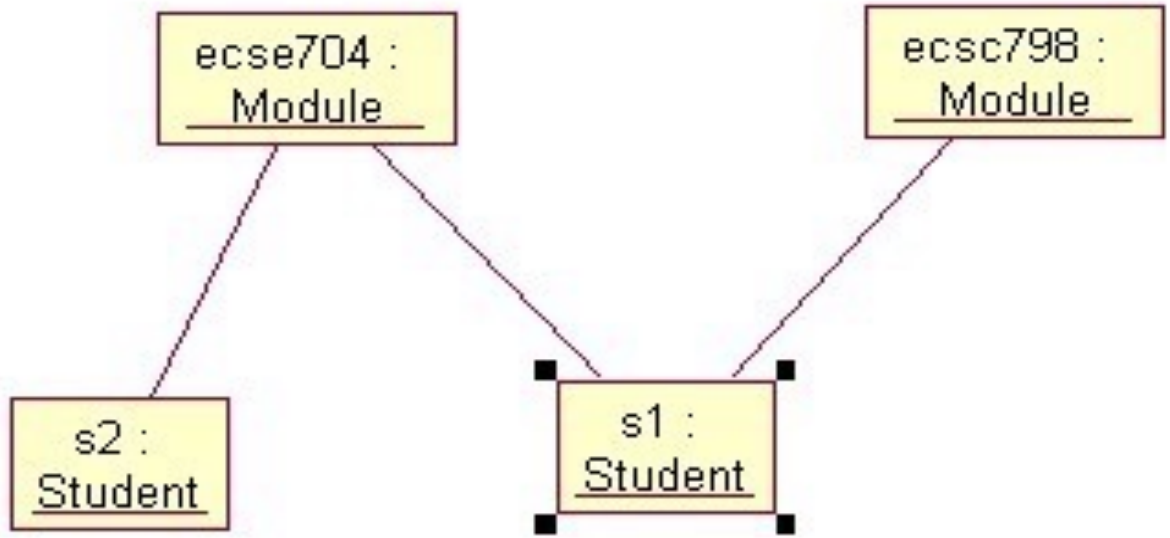
AGGREGATION

- 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

```
1   using System;
2   namespace Inventory
3   {
4       public class Part
5       {
6           CatalogueEntry Entry { get; }
7           public string Name   { get { return Entry.Name;  } }
8           public int    Number { get { return Entry.Number;} }
9           public double Cost   { get { return Entry.Cost;  } }
10
11          public Part(CatalogueEntry c)
12          {
13              Entry = c;
14          }
15
16          public Part(Part p) : this(p.Entry) { }
17      }
18  }
19
```

COMPOSITION

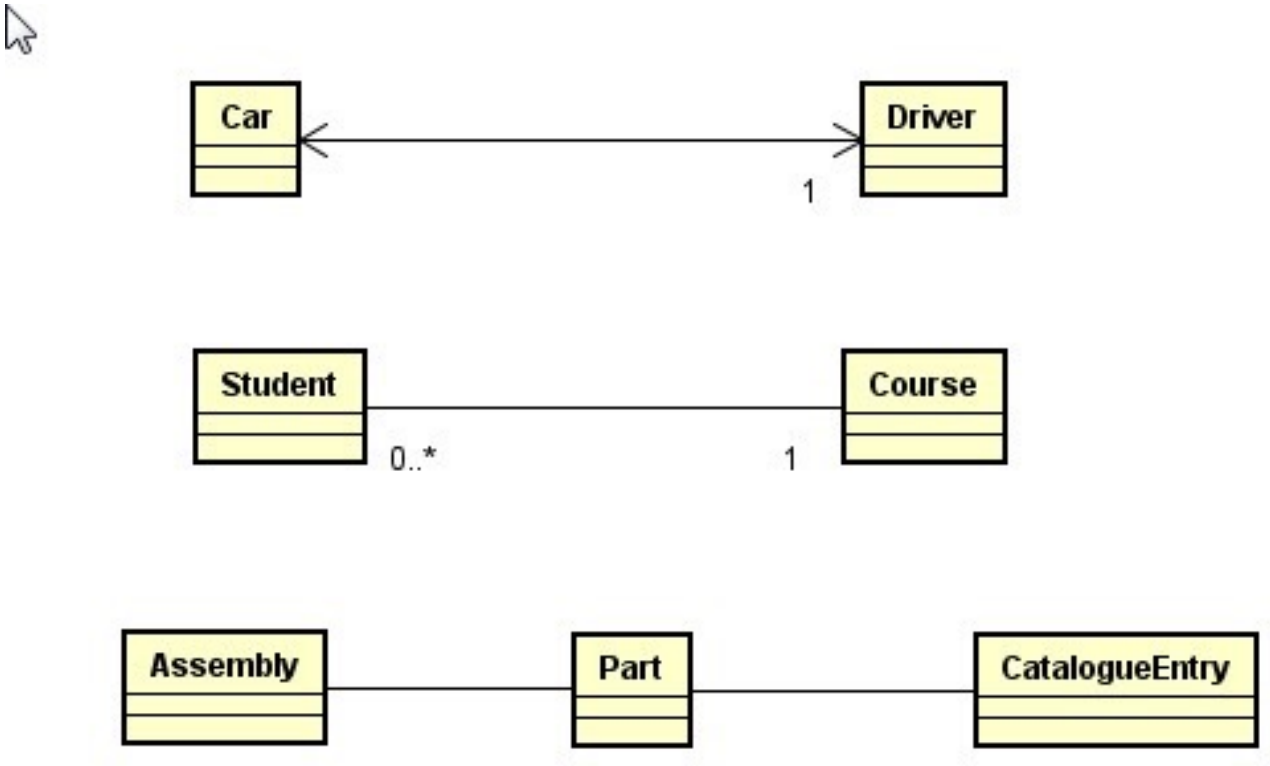
- Composition is another version of association that denotes ownership – stronger than aggregation
- No sharing
- The 'owned' object is destroyed when the 'owning' object is destroyed

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?

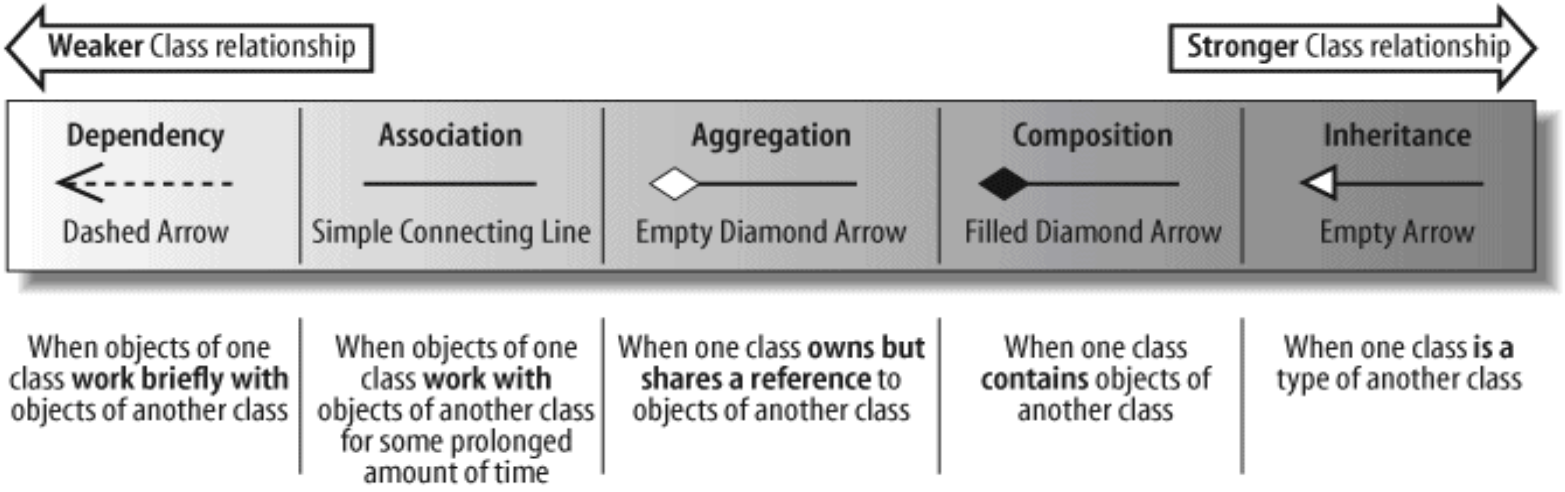
Multiplicity - examples



Exercise - Class diagrams

- The university is developing a new timetabling system. This system will allow everyone (staff and students) in the university to access their timetable, as well as allow room bookings to be managed consistently and without conflict.
- The system will need to model courses, modules on courses, students' module registrations (to know how many students are on a particular module), lecturers who are module leaders, as well as locations and their capacity.
- Create a class diagram that represents as much of the information above as possible. What operations and attributes do you think are required to generate a timetable for a student? Try to be as specific as possible about such things as aggregation or composition, and multiplicities.

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