



1B1a

Object-Oriented Concepts

1 Copyright © 2001, Graham Roberts Department of Computer Science 


Agenda

- The programming process.
- Review of OO ideas.
- UML – diagram notation.

2 Copyright © 2001, Graham Roberts Department of Computer Science 

Note


Here, and throughout 1b1a, we are dealing with programming in the *small scale*.

3 Copyright © 2001, Graham Roberts Department of Computer Science 

Development Activities


- Requirements Gathering
- Requirements Analysis
- Analysis
- Design
- Implementation
- Testing (Verification & Validation)
- Delivery
- Maintenance

} System Function
Use Cases
OOA/OOD
OOP

4 Copyright © 2001, Graham Roberts Department of Computer Science 


Requirements Gathering

- Determine what the program should do.
- Communicate the *requirements* clearly to all concerned.
 - Get feedback.
- Make them unambiguous and correct.
 - Usually impossible...
- Categorise as essential, important or optional.

5 Copyright © 2001, Graham Roberts Department of Computer Science 

Analysis (Outsides)

- Describe how the program should behave:
 - Tasks, inputs, processing, output
 - But not how these are programmed.
- Identify possible classes (responsibilities and collaborations).
- Create conceptual *model* of program.

6 Copyright © 2001, Graham Roberts Department of Computer Science 

Design (Insides)

- Determine how the program can be implemented.
- Refine classes and relationships.
- Methods, method call sequences, data structures, algorithms.
- Infrastructure.

7

Copyright © 2001, Graham Roberts

Department of Computer Science



Implementation

- Transform design to code.
- Fill in all detail until a complete program is built.

8

Copyright © 2001, Graham Roberts

Department of Computer Science



Testing

- Check that the program works according to the specification.
- Run program on test data, try to do things that may cause errors.
- Testing is about finding (and fixing) errors.
- You cannot use testing to “prove” a program is correct!

9

Copyright © 2001, Graham Roberts

Department of Computer Science



Delivery

- Install/Deploy program.
- Write documentation, train users.
- Verify program works and is useable (acceptance testing).

10

Copyright © 2001, Graham Roberts

Department of Computer Science



Maintenance

- Care of program for rest of its lifetime (possibly many years).
- Fix errors.
- Add features.
- Port to new systems.

11

Copyright © 2001, Graham Roberts

Department of Computer Science



Organising the Core Activities

- Could follow through each activity one after the other?

Won't work!

- Need iteration, exploration and experimentation:

“Analyse a little,
Design a little,
Code a little”

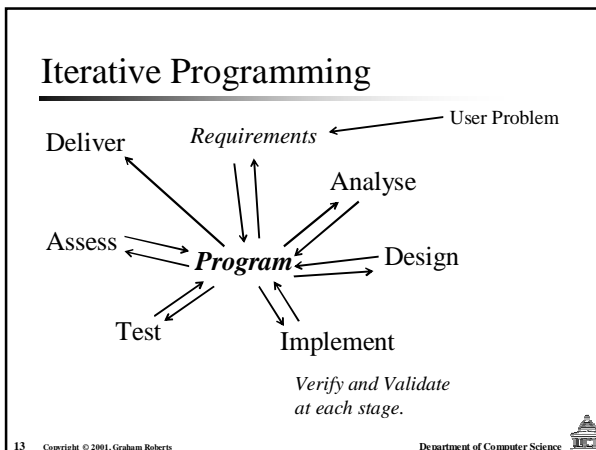
- A code centric approach...

12

Copyright © 2001, Graham Roberts

Department of Computer Science





Good Design/Programming

... is hard to do!

- Trying to predict the future
 - It will work (won't it?)
 - I can program this (can't I?)
 - I don't make mistakes (do I?)
 - I do good quality work (don't I?)

14 Copyright © 2001, Graham Roberts Department of Computer Science

So,

- How do you go about designing and implementing a object-oriented program?
- How do you know the program code you are writing is:
 - Correct?
 - Good quality?
 - Robust?
 - Reliable?
- We want to start addressing these issues.
 - You will never stop learning about the answers throughout your professional career.

15 Copyright © 2001, Graham Roberts Department of Computer Science

Key issues

- Knowing how to test your program.
- Really knowing how to test your program.
- Knowing how to design and implement your program.
- Knowing how to judge the quality of your work (including self assessment).

All are hard!

16 Copyright © 2001, Graham Roberts Department of Computer Science

Consider the Programming Process

- Programming is, first and foremost, about understanding the problem.
 - But this is very difficult...
- Programming then solves the problem, by translating a representation of the solution into programming language notation.
 - This is not much easier!

17 Copyright © 2001, Graham Roberts Department of Computer Science

Questions?

18 Copyright © 2001, Graham Roberts Department of Computer Science

What's this Software Engineering thing?

- Software Engineering (SE) is often used as a label to cover development process in general.
- We don't do any engineering...
 - Design/programming is a still largely a craft.
- Really only applicable to large scale development, safety-critical systems.

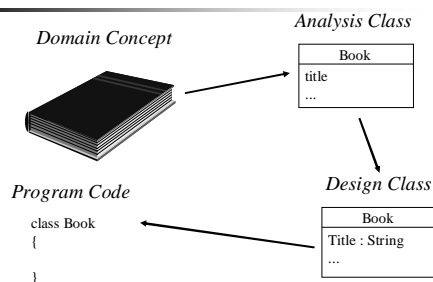


Points of View

- Could be that:
 - Programming is seen as a part of the overall SE process.
 - SE provides the context and framework that programming fits into.
 - Programming is activity that solves small scale problems, while SE deals with the big problems.



SE Centred

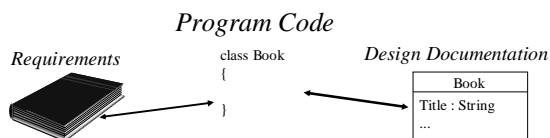


Or...

- Code is what really matters.
- It must embody the real design.
- It is what gets delivered.
- It is the *working* solution.



Code Centred



Reality?

- Programming is *the* central activity.
- Everything else supports programming.
- Code is the only thing that matters.

But

- This does NOT imply that design and documentation are neglected.

What do you think?



Hmmm...

- Deliberately been controversial to get *you* thinking about the issues!
- Truth is that design/programming is heavily skill-based.
 - Not easy.
 - No excuse for sloppy work.

25 Copyright © 2001, Graham Roberts

Department of Computer Science



Questions?

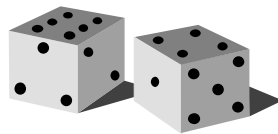
26 Copyright © 2001, Graham Roberts

Department of Computer Science



A (Very Simple) Example

Initial Idea:



“A dice game in which a player rolls two die. If the total is seven the player wins, otherwise the player loses.”

27 Copyright © 2001, Graham Roberts

Department of Computer Science



Activities

- Requirements — functionality and use cases.
- Analysis — conceptual model.
- Design — *class diagram* and *interaction diagrams*.
- Implementation — Java code.

28 Copyright © 2001, Graham Roberts

Department of Computer Science



Define Requirements

- R1. Play a game (essential)
- R2. Record result in table (important)
- R3. Display dice animation (optional)

29 Copyright © 2001, Graham Roberts

Department of Computer Science



Identify Behaviour (Use Case)

U1. Play a game

Actors: Player

Description: Player rolls the dice. If the dice total is seven the player is a winner, otherwise a loser.

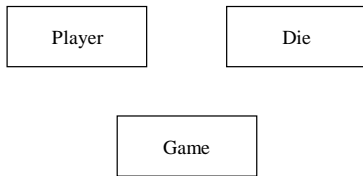
30 Copyright © 2001, Graham Roberts

Department of Computer Science



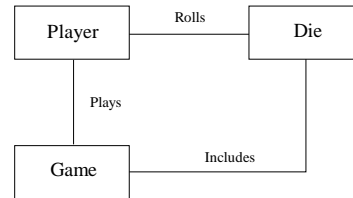
Conceptual Model (Analysis)

- Identify potential classes



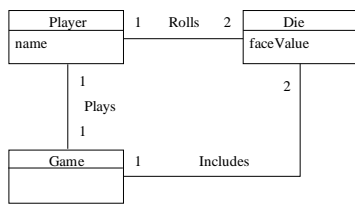
Conceptual Model (2)

Add relationships (associations):



Conceptual Model (3)

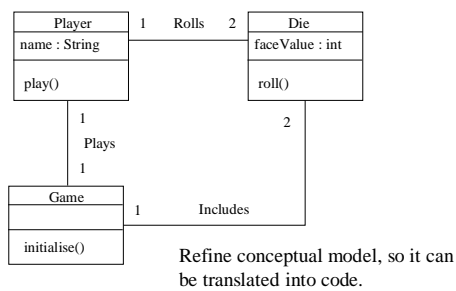
Begin to fill in details:



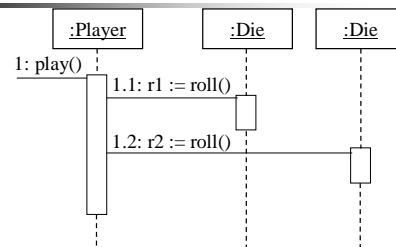
Design

- Could:
 - Refine conceptual model into detailed *class diagram(s)*.
 - Generate *object interaction diagrams* and *state diagrams* as needed.
 - Or:
 - Directly write the code.
- Which approach should be taken?
What issues help you make a choice?

Class Diagram

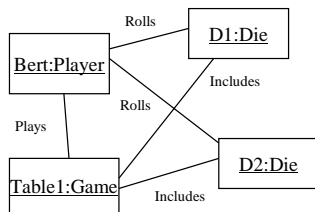


Interaction Diagram



Show that objects can call methods to play game.

Object Diagram



“Snapshot” of program when running.

37 Copyright © 2001, Graham Roberts

Department of Computer Science



Code — Player.java

```

public class Player
{
    private String name ;
    public void play()
    { ... }
}
  
```

38 Copyright © 2001, Graham Roberts

Department of Computer Science



Code — Die.java

```

public class Die
{
    private int faceValue ;
    public void roll()
    { ... }
}
  
```

39 Copyright © 2001, Graham Roberts

Department of Computer Science



Code — Game.java

```

public class Game
{
    private Player name ;
    private Die[] dice ;
    public static void main()
    { ... }
}
  
```

40 Copyright © 2001, Graham Roberts

Department of Computer Science



Design before Coding?

- Both the design diagrams/documentation and the code should be *consistent* descriptions of the same program.
- Whether you design then code, or directly code, you are still be working with the *same* description.
- That description must be accurate and sufficient.

41 Copyright © 2001, Graham Roberts

Department of Computer Science



Do what is necessary

- It's not really a question of one approach or the other but of choosing your strategy to provide the most effective and reliable development process *for the current context*.
- Do what is required to do an effective job.
 - Learning how to choose what to do when, why and how is the hard part.

42 Copyright © 2001, Graham Roberts

Department of Computer Science



Implementation

- May need to:
 - Translate the knowledge in the class and interaction diagrams to code.
- Or
 - Code class/interaction knowledge directly.
 - And generate and review diagrams from code (tools can do this automatically).

43 Copyright © 2001, Graham Roberts

Department of Computer Science



Predicting the future (again)

- Design is about predicting how your proposed system will work in the future once you have constructed it.
- Prediction is difficult and error prone...
- Always look for ways to reduce the risk.

44 Copyright © 2001, Graham Roberts

Department of Computer Science



Questions?

45 Copyright © 2001, Graham Roberts

Department of Computer Science



Key OO Concepts

- Object and Class
- Instance and Instantiation
- Methods and method Calls
- Abstraction and Encapsulation
- Association, Inheritance and Dynamic Binding
- Polymorphism

46 Copyright © 2001, Graham Roberts

Department of Computer Science



Object

- Represents an entity.
- Has an identity.
- Encapsulates data (attributes) as its state.
- Performs operations when requested.
- Has a public interface.
- Has a private internal representation.

`:Player`

47 Copyright © 2001, Graham Roberts

Department of Computer Science



Class

- Defines the structure and behaviour of a particular kind of object.
- Acts as a template or blueprint.
- An object must be an *instance* of one (and only one) class.
- A class may have many instance objects.

| |
|-------------------------------|
| Name |
| Instance Variable Definitions |
| Method Definitions |

| |
|---------------|
| Player |
| name : String |
| play() |

48 Copyright © 2001, Graham Roberts

Department of Computer Science



Encapsulation

- Objects are strongly *encapsulated*.
- From the outside an object can only be manipulated via its *public* interface.
- The internal state is private and can only be changed by the object itself.
- Supports information hiding.

49 Copyright © 2001, Graham Roberts

Department of Computer Science



Role of Encapsulation

“Encapsulation is a technique for minimising interdependencies among separately written modules by defining strict external interfaces. The external interface acts as a contract between a module and its clients. If clients only depend on the interface, modules can be re-implemented without affecting the client. Thus the effects of changes can be confined.” Synder 1986

50 Copyright © 2001, Graham Roberts

Department of Computer Science



Instantiation

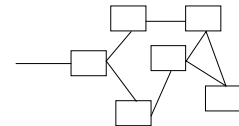
- The activity of creating an object given its class.
- The object’s data structure (state) must be initialised.

51 Copyright © 2001, Graham Roberts

Department of Computer Science



Message Passing



- Objects interact by message passing (method calls).
- A message *requests* an object to perform an operation or service.
- A message consists of a name and any arguments.

52 Copyright © 2001, Graham Roberts

Department of Computer Science



Method

- Implementation in code of an operation or responsibility.
- Called in response to a message sent to a specific object.
- Has full access to the object’s state.
- Matching a message name to a method is called *binding*.

53 Copyright © 2001, Graham Roberts

Department of Computer Science



Dynamic Binding

- The *same* message can be sent to different objects.
- Each object can bind the message to a different method, as defined by its class.
- This reduces the *coupling* between objects
- It enables *pluggability* and *substitutability*.

54 Copyright © 2001, Graham Roberts

Department of Computer Science



Abstraction

- Abstraction separates the *essential* from the full detail.
- Classes represent *abstractions*.
- They provide a selective and simplified view of the concepts being represented as objects.

55 Copyright © 2001, Graham Roberts

Department of Computer Science



Structure and Relationships

- Classes are used to describe the structure of a system.
- To do this classes must be related or connected with one another.
- There are three key relationships:
 - Association
 - Aggregation (composition)
 - Inheritance

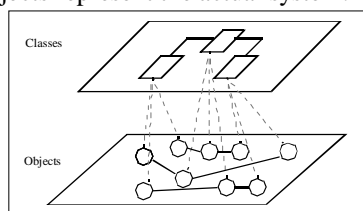
56 Copyright © 2001, Graham Roberts

Department of Computer Science



Classes and Objects

- Classes define the structure and behaviour of a system.
- Objects represent the actual system.



57 Copyright © 2001, Graham Roberts

Department of Computer Science



Questions?

58 Copyright © 2001, Graham Roberts

Department of Computer Science



UML Notation

- Already seen various icons and diagrams.
- Expressed using the Unified Modelling Language (UML) notation.
- Established standard.
- Visit www.uml.org.



59 Copyright © 2001, Graham Roberts

Department of Computer Science



Diagrams

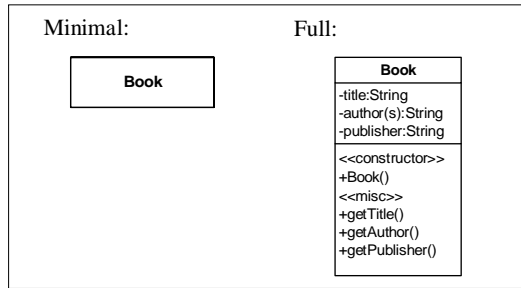
- A *class diagram* describes the static structure of a system in terms of classes and their relationships.
- An *object diagram* is a snapshot of a system at some point during its execution, showing objects and their links.

60 Copyright © 2001, Graham Roberts

Department of Computer Science

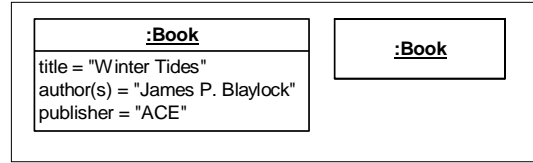


The Class Icon



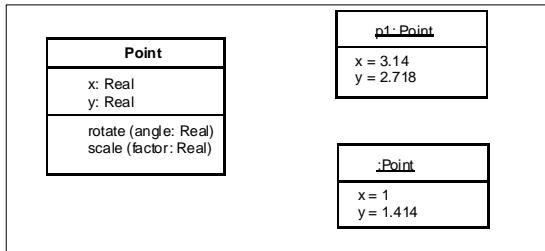
Object Icons

- Labelled `ObjectName : Class`
- Object name often omitted.



Class and Object

- Not usually shown on same diagram.

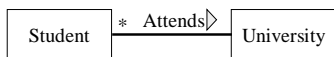


Association

- Models a relationship between two classes.

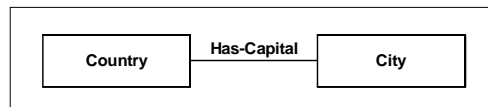
For example, the association *works-for* between a person class and a company class.

Association



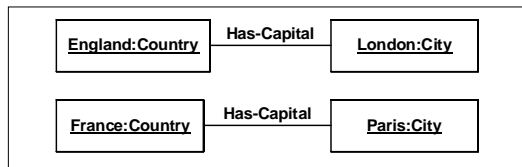
- Associations between classes describe the *structural relationships* in a system.
- And determine the *links* between objects.

Associations (1)



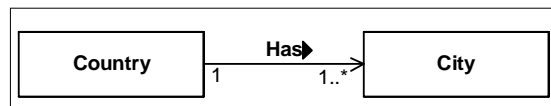
Links

- An association corresponds to links between instance objects.
- A link is an instance of an association.



Associations (2)

- Multiplicity of objects can be denoted on the associations:

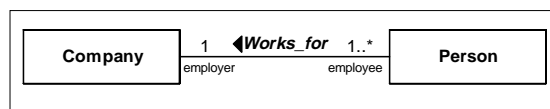


Multiplicity

- 1 - one
- 1..* - one or more (many)
- 0..* - zero or more
- * - many
- 1..5 - one to five
- 2,4,8 - two, four or eight

Associations (3)

- Role names can be added to associations to make role of each class clear.



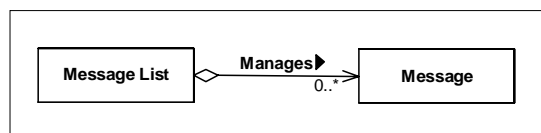
Aggregation (Composition)

- Represents a *part-whole* relationship.
- A stronger form of association.
- The life-time of the whole dictates the life-time of the parts.

For example, an engine is *part-of* a car.

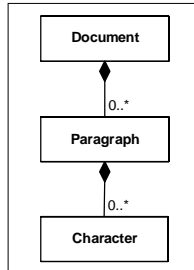
Aggregation

- A part-whole relationship.



Composition

- Stronger form of ownership and control.



Questions?

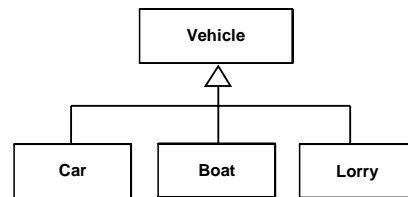
Inheritance

- Models the “kind-of” relationship between classes.
- Specifies that one class is an extension of another class.

For example, a bus is a *kind-of* vehicle.

Inheritance (2)

- Several classes can inherit from the same superclass.



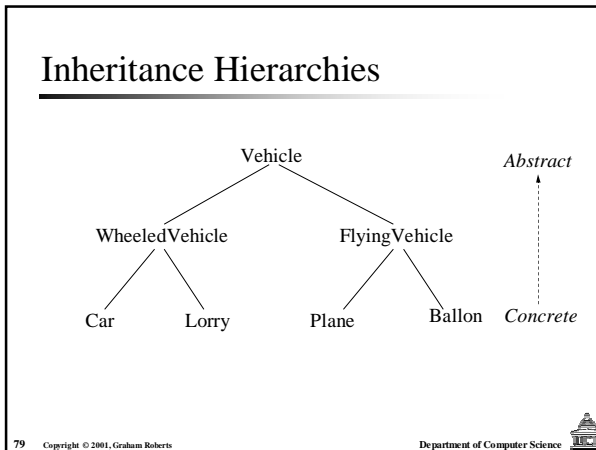
Subclass and Superclass



- A *superclass* may be inherited by a *subclass*.
- The subclass gains all the properties of the superclass and can add more.

Generalisation & Specialisation

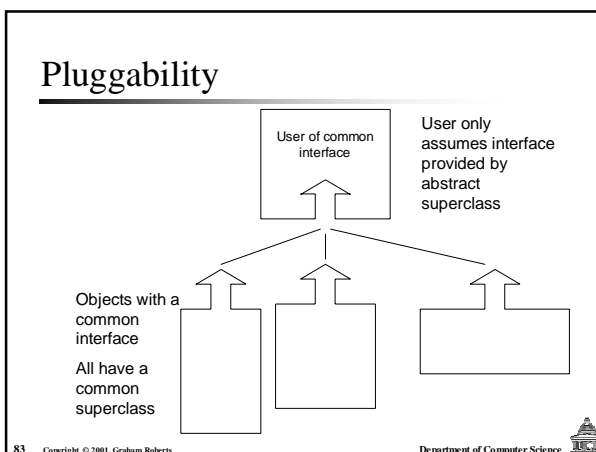
- A superclass is a generalisation.
- A subclass is a specialisation.



- ### Role of Inheritance
- Allows classification hierarchies.
 - Enables the use of common interfaces.
 - Enables implementation sharing (by extension, not copy and edit).
- 80 Copyright © 2001, Graham Roberts Department of Computer Science

- ### Polymorphism
- Enabled by dynamic binding and inheritance.
 - Permits one message to be sent to several kinds of objects.
 - Allows interchange of objects that share a common public interface.
- 81 Copyright © 2001, Graham Roberts Department of Computer Science

- ### Polymorphism and Inheritance
- A superclass can define a common interface.
 - Subclasses inherit the common interface and specialise the corresponding methods.
 - A subclass object can be used where a superclass object has been specified.
- 82 Copyright © 2001, Graham Roberts Department of Computer Science



- ### Old Code can Call New Code
- New pluggable components can be added *without* changing the users of the components.
 - Code designed to use the common interfaces remains unchanged.
- 84 Copyright © 2001, Graham Roberts Department of Computer Science

Being Object-Oriented

- Exploit the combination of:
 - objects
 - classes
 - encapsulation
 - inheritance
 - polymorphism
 - dynamic binding
 - pluggability

OO = Object-Oriented



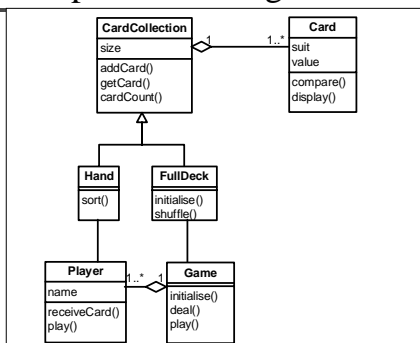
Classes, objects and systems

- The *structure* of a complex system is described by lots of classes.
- And *dynamically* realised by many objects.

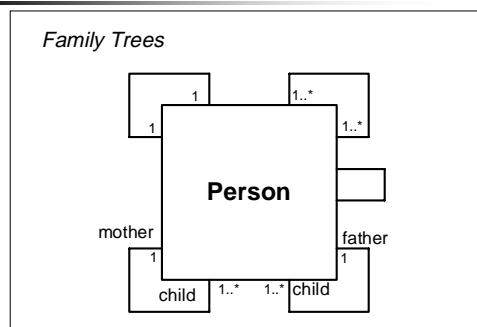
Structure and behaviour must be precisely defined.



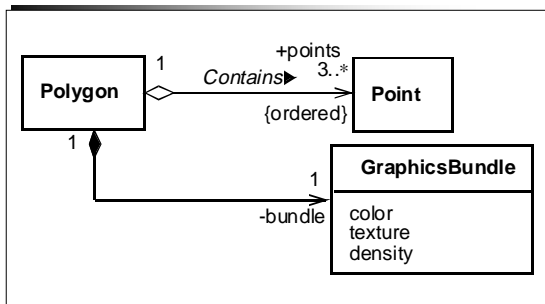
An Example Class Diagram



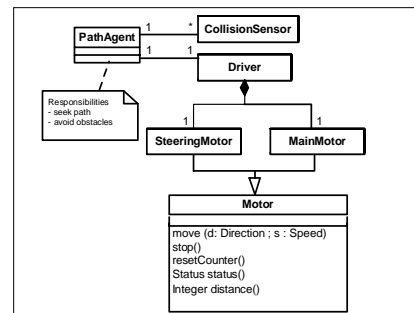
Example diagram (1)



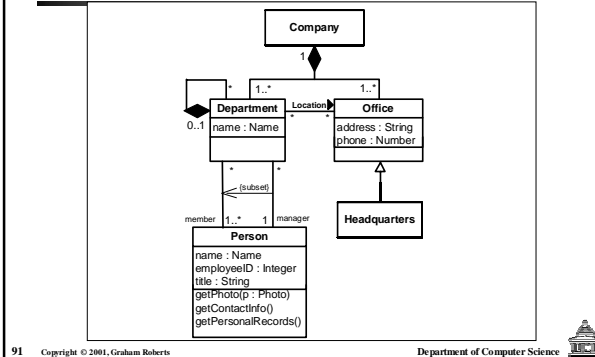
Example diagram (2)



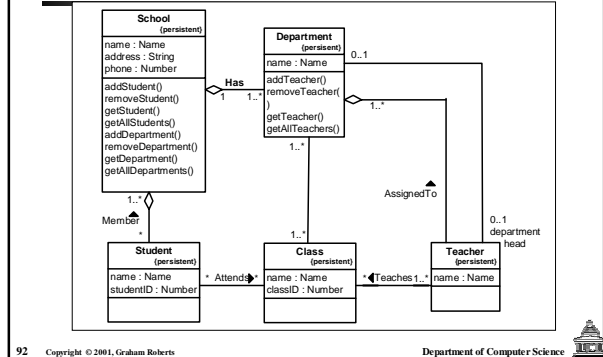
Example diagram (3)



Example diagram (4)



Example diagram (5)



Summary

- Covered a lot of ground.
 - Iterative development activities.
 - OO concepts in detail.
 - Core UML notation.
- Spend time reviewing all this.
- Do background reading.