Lecture #1 Summary

- Software Engineering is not Programming
- We must approach software projects paying careful attention to what might go wrong!

Program Complexity:

Code Reuse:

Maintenance and Expansion:

People make mistakes:

Overall Philosophy

- *If the overall structure of the computer code and the organisation of the data mimics the real world, or better how we think about and describe the real world, then there is a better chance that the program will be (a) easier to develop, (b) easier to extend, (c) contain parts that can be reused for other purposes*.

Overall Philosophy

- *Our code should be implemented in a manner that minimises the chances of errors arising because different people are developing different parts of the code. The way in which a piece of code (e.g. a function) can be reliably used will be strictly enforced to avoid errors due to misunderstandings.*

Lecture #2

- Object Oriented Design
- Classes in C++
- C revision: pointers and references; passing arguments to functions and function return values
- Student Directed Time

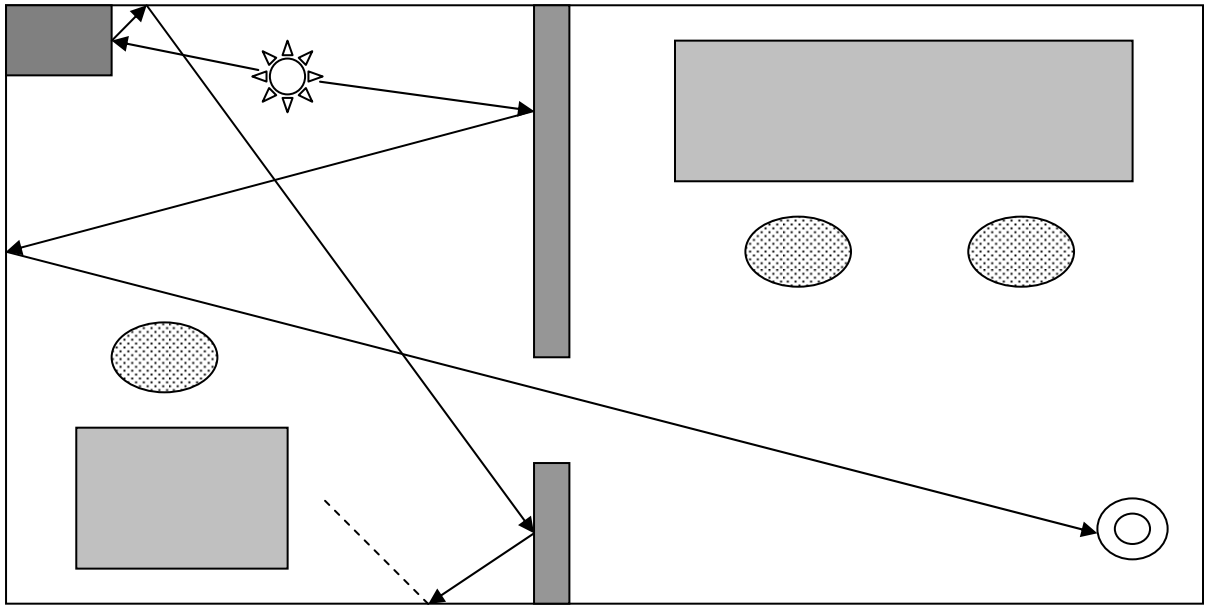
Object Oriented Design

- Define the basic sorts of *objects* that are involved in the problem.

  What characterises them generically?

  What can be done to or by these things?

Object Oriented Design



 Receiver  Transmitter

Object Oriented Design

Get geometry data

For all rays leaving transmitter {

Until this ray arrives at receiver or is too weak {

Get the next intersection of this ray with an obstacle {

Evaluate how the obstacle changes the ray's direction and strength or splits it into multiple rays

}

If this ray has arrived at receiver add to the receiver's signal strength

}

Output the results

Object Oriented Design

Look at the “language”;

- A ray is being thought of as a thing with a strength and direction.
- Obstacles are any things that change ray’s properties.
- Clearly rays and obstacles interact.
- For example I could say *make a change* to this particular ray because it hit this particular obstacle.
- Now what actually happens depends upon the particular obstacle – but at this level all we need to realise is that **obstacles, whatever they are, change rays**.

Object Oriented Design

- Have I considered at all how any of this will be represented in detail in a code? No and yet I have an initial software design. How I have I done this – by considering what sort of “things” there are and how they interact.

- More usefully I have placed no restrictions on what obstacles might occur. Let’s say we have just walls to start with and design and implement a working program. Then the client wants to add tables – oh dear we have to do a big recode – well not if we could somehow “program in” the generality of the above”.

- There you go, we have found a place to start!

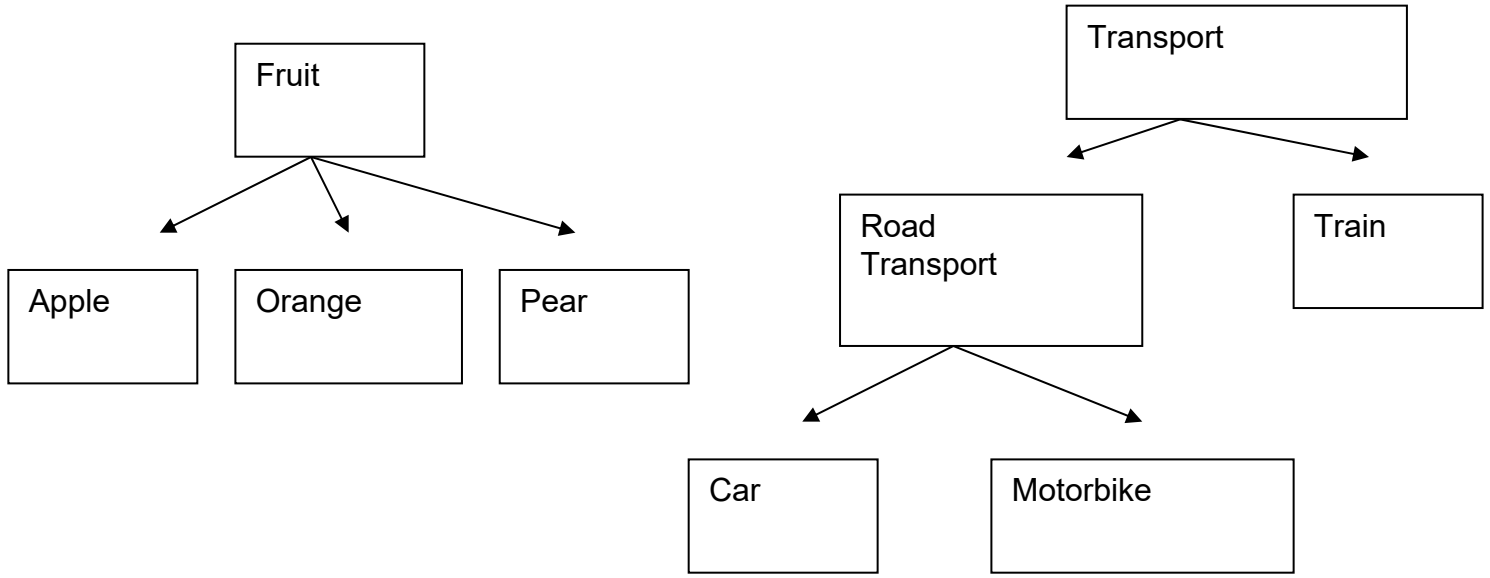
Object Oriented Design

- **Objects – usually nouns**
  - Rays, obstacles
  - Apple, orange, pear – different objects?
  - Integer, floating point number, complex number – different objects?
  - Motorbike, car, train – different objects?

Object Oriented Design

- **Hierarchies of objects**
  - Determine the relationships between different *objects*
  - Obviously there is something similar about apple, orange, pear, but not car and orange
  - Clearly we have hierarchies of inter-related objects

Object Oriented Design



- As this is how we think of the object relationships, we want our programs to be structured with such hierarchies

Object Oriented Design

- The *object* is an noun - *motorbike*