


# 1B1b

## Hints about doing the Mini-Project properly

1 Copyright © 2004, Graham Roberts Department of Computer Science 


## Think Object-Oriented

- Identify classes and objects.
  - Objects encapsulate state or a representation.
  - And provide services or methods.
  - Classes describe how objects are implemented.
- Tasks are performed by objects calling each others methods.

2 Copyright © 2004, Graham Roberts Department of Computer Science 


## An example

- The London Underground Problem
  - Write a program that will find a route between any two stations on the London Underground network.

3 Copyright © 2004, Graham Roberts Department of Computer Science 

## Getting Started


- Brainstorm!!!
- We need:
  - Data structures to store a representation of the underground.
  - An algorithm to find a route using the data structure.
- AND we want to take an object-oriented point of view.

4 Copyright © 2004, Graham Roberts Department of Computer Science 

## Algorithm Ideas


- Try finding a route on an underground map:
  - Locate start station
  - Follow line in one direction
  - Do we go past end station?
  - If yes, then done, otherwise back-track to start and go in the other direction.

People searching for a route will take short cuts – the program will need to do things step-by-step.

5 Copyright © 2004, Graham Roberts Department of Computer Science 

## Algorithm Ideas (2)

- What if the station is on a different line?
  - Do a recursive search onto the new line.
    - At each station, if it is not the destination,
    - Pick each line in turn, search that line in both directions.

6 Copyright © 2004, Graham Roberts Department of Computer Science 

### Algorithm Ideas (3)

- Wait, can't this be done using graphs or adjacency matrices or something?
- Yes, and would probably be more efficient than recursive (depth-first) search.
- Then research the alternatives!
  - We will carry on with recursive search for now.

7

Copyright © 2004, Graham Roberts

Department of Computer Science



### Classes - Role Play (CRC method)

- Identify some initial classes/objects to work with:
  - Station, Line, RouteFinder
- Pick a task:
  - Find a route? – Too complicated to start with!
  - Try something simpler: What Line is a Station on.

8

Copyright © 2004, Graham Roberts

Department of Computer Science



### Role Play (2)

- Talk through a scenario:
  - Get a station object. How?
  - Ask it what line it is on. Call the getLine method.
- Implications:
  - A station knows what line it is on. An instance variable is needed.
  - We need a collection of all stations to get a station from.

9

Copyright © 2004, Graham Roberts

Department of Computer Science



### Role Play (3)

- Review
    - A station can be on more than one line.
      - Needs a variable to store a collection of lines.
    - If we have a collection of stations, how is a given station identified?
      - By its name!
      - A station needs to store its name in a variable.
- (Note, the need for a name may be obvious but we should only add it once the need has been established.)

10

Copyright © 2004, Graham Roberts

Department of Computer Science



### Station Class – 1<sup>st</sup> go

```
class Station
{
    private String myName;
    private Line myLine;

    public Line getLine()
    { return myLine; }

    public String getName()
    { return myName; }
}
```

11

Copyright © 2004, Graham Roberts

Department of Computer Science



### Next Scenario

- What is the next station on the line?
  - In fact, what is a line?
    - A collection of stations,
    - and a line has a name.
  - Could provide a getNextStation method.

12

Copyright © 2004, Graham Roberts

Department of Computer Science



## Line

```
class Line
{
    private ArrayList stations;
    private String name;

    public Station getNextStation(String
currentStation, boolean forward) { ... }
}
```

← Returns null if no station?

13 Copyright © 2004, Graham Roberts

Department of Computer Science



## The Data Structure

- We can now see that it consists of:
  - A collection of lines,
  - where each line is a collection of stations
  - Also want a collection of stations to make it easy to find a station to start with.

14 Copyright © 2004, Graham Roberts

Department of Computer Science



## Searching

- Could Line implement searching?
  - find (Station destination) ;
- Yes – the recursive algorithm could be implemented quite easily.
- A separate RouteFinder class would not be needed.
- Searching would then take place by the objects calling each others methods.

15 Copyright © 2004, Graham Roberts

Department of Computer Science



## Searching (2)

- Note the OO point of view:
  - Object cooperate to represent the underground and do the searching.
  - Objects encapsulate information and provide services.

16 Copyright © 2004, Graham Roberts

Department of Computer Science



## RouteFinder?

- An alternative is to encapsulate the algorithm in an object.
- Advantages:
  - We can change algorithm by using different route finder objects.
  - The algorithm can be implemented independently of how lines and stations are implemented.

17 Copyright © 2004, Graham Roberts

Department of Computer Science



## RouteFinder (2)

- Disadvantages:
  - The encapsulation of Line and Station may need to be reduced to allow the RouteFinder access.
  - However, there are more advanced ways of structuring the program to avoid the problem.
- On balance a RouteFinder is a better solution but requires more sophistication.

18 Copyright © 2004, Graham Roberts

Department of Computer Science



## Progressing

- Now have enough to start coding a working prototype.
- New issues will be encountered:
  - How are all the lines and stations created and initialised?
  - Input/Output
- But a feasible solution is emerging.

19 Copyright © 2004, Graham Roberts

Department of Computer Science



## Progressing (2)

- Prototyping involves experimenting.
- Some things won't work – throw them out!
- You'll get new ideas and perhaps find better solutions – use them!
- Don't let things get messy – spend time cleaning up and throwing out.
- Test your code!

20 Copyright © 2004, Graham Roberts

Department of Computer Science



## Class Checklist

- What instance variables, what types?
  - Are they all private?
- Constructors and initialisation
  - How is an object initialised?
- What public methods?
  - What services do objects provide?

21 Copyright © 2004, Graham Roberts

Department of Computer Science



## Review Classes

- What can be eliminated to keep the class as simple as possible but no simpler?
- What have we discovered that may matter in the future?
- Are the public methods reducing encapsulation unnecessarily?

22 Copyright © 2004, Graham Roberts

Department of Computer Science



## Summary

- Think Object-Oriented!
- Had a taste of the thinking and designing process.
- All the time you are searching for viable solutions and balancing the conflicting issues.
- Simplify, Simplify, Simplify.
- BUT Simple != Trivial

23 Copyright © 2004, Graham Roberts

Department of Computer Science

