Case Study: Modelling using Objects

Murray Cole

Modelling the world with data

Specifying systems is one of the hardest tasks in SE

Key issues in object-oriented modelling are

- which classes do we need
- how do they relate to each other

One simple approach is take a textual description and look for noun-phrases, then discard those which seem vague, out of scope, or duplicate.

An Example

The Informatics Teaching Office maintains a record of every student taking Informatics courses. This includes basic contact and identification information on the student, the student's Director of Studies, and the student's programme of study. The programme of study is a record of which courses the student is taking that year.

An Example

The *Informatics Teaching Office* maintains a record of every *student* taking *Informatics courses*. This includes *basic contact and identification information* on the student, the student's *Director of Studies*, and the student's *programme of study*. The programme of study is a *record* of which courses the student is taking that *year*.

- Choose "Student", "Director of Studies" and "Course"
- Student and Director share some properties - superclass "Person"?
- Note down attributes of each class

Validating the Model

To add confidence that we have captured the right information, consider some operations which will be required.

- Do we have the right fields?
- Can we support the operations and keep data consistent?
- Can we do this easily (and flexibly)?

One simple tool is construct a demo driver program which exercises the model.

```
class ITOdemo {
   public static void main(String[] args) {
        Director d1, d2;
        Student  s1, s2, s3;
        Course   c1, c2, c3;
        d1 = new Director("Murray", "Cole", "mic", 505154);
        d2 = new Director("Paul", "Jackson", "pbj", 505131);
        s1 = new Student("Freddy", "Furby", "ff", 208729);
        s2 = new Student("Pika", "Chu", "pc", 293923);
        s3 = new Student("Thomas", "TankEngine", "tt", 1);

        c1 = new Course("Apld Txting");
        c2 = new Course("Track Maintenance");
        c3 = new Course("Sitting around");
```

```
        s1.setDirector(d2);
        s2.setDirector(d1);
        s3.setDirector(d2);

        s1.addCourse(c1); s1.addCourse(c2);
        s2.addCourse(c3); s2.addCourse(c2);
        s3.addCourse(c2);

        d1.showDirectees(); d2.showDirectees();
        s2.showCourses();

        s1.addCourse(c3); s1.showCourses();
        c3.changeTitle("Hard work in the Library");
        s1.showCourses();
    }
}
```

```
class Course {
    private String title;
    public Course(String title) {
        this.title = title;
    }

    public String getTitle() {
        return title;
    }

    public void changeTitle (String newTitle) {
        title = newTitle;
    }
}
```

```
class Person {
    private String forename;
    private String surname;
    private String email;

    public Person (String fname, String sname,
                   String email) {
      this.forename = fname;
      this.surname  = sname;
      this.email    = email;
    }

    public String toString () {
        return forename + " " + surname;
    }
}
```

```
class Student extends Person {
    private int matricno;
    private Director dos;
    private Vector programme;

    public Student(String fname, String sname,
                   String email, int matricno) {
        super (fname, sname, email);
        this.matricno = matricno;
        programme  = new Vector();
    }

    public void addCourse(Course course) {
        programme.addElement(course);
    }
```

```
    public void setDirector(Director mydos) {
        dos = mydos;
        mydos.assignDirectee(this);
    }

    public void showCourses() {
        Iterator i  = programme.iterator();
        Course c;

        while (i.hasNext()) {
            c = (Course) i.next();
            System.out.println(c.getTitle());
        }
        System.out.println();
    }
}
```

```
class Director extends Person {
    private int extension;
    private Vector directees;

    public Director(String forename, String surname,
                    String email, int extension) {
        super(forename, surname, email);
        directees = new Vector();
    }

    public void assignDirectee(Student student) {
        directees.addElement(student);
    }
```

```
    public void showDirectees() {
        Iterator i  = directees.iterator();
        Student s;

        while (i.hasNext()) {
          s = (Student) i.next();
          System.out.println(s); // invokes "toString"
        }
        System.out.println();
    }
}
```

Exercise

Complete the following method (as a member of `Director`), which should print out all the courses being taken by all the DoS's directees. Don't worry about duplicates.

```
public void listCourses() {



}
```

```
public void listCourses() {
    Iterator ds  = directees.iterator();
    while (ds.hasNext()) {
       ((Student) ds.next()).showCourses();
    }
}
```

Note the cast to `Student`, allowing us to invoke `showCourses()`