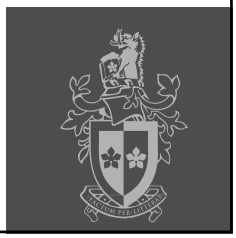
SWINBURNE UNIVERSITY OF TECHNOLOGY

SWE30003
Software Architecture and Design

Lecture 5
Object Design (Part I) –

Responsibility-Driven OO Design



1

Logistical matters

- Weekly submissions – A & Q
  - Week 2: 362 and 341 out of 459;
  - Week 3: 397 and 375 out of 459;
  - Week 4: 399 and 390 out of 459;
  - Week 5:
  - Week 6:
  - Note that this is a hurdle requirement
  - No late submission
  - xx students have not submitted anything to date!!
- Assignment 1: … not long to due date

2

2

Question to Answer from Week 4

A good programmer in these times does not just write programs, *he/she builds a working vocabulary*. Provide examples and explain this statement.

3

3

Principal References

- Rebecca Wirfs-Brock, Brian Wilkerson and Lauren Wiener, *Designing Object-Oriented Software*, Prentice Hall, 1990, Chapters 3 to 5.
- Bernd Bruegge and Allen H. Dutoit, *Object-oriented Software Engineering*, Prentice Hall, 2001, Chapter 5.
- Shari L. Pfleeger, *Software Engineering: Theory and Practice* (2nd Edition), Prentice Hall, 2001, Chapters 5 and 6.
- Ian Sommerville, *Software Engineering* (9th Edition), Addison Wesley, 2013, Chapter 7.

4

4

Object-Oriented Design

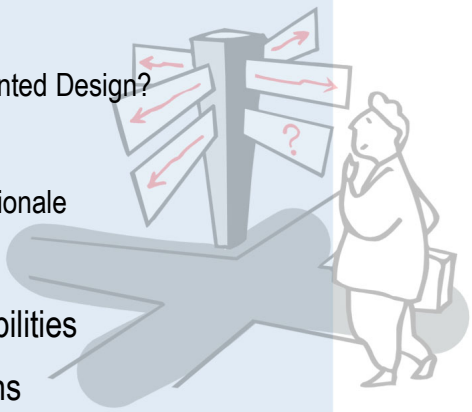
OOD != OOP

5

5

Roadmap

- **Preliminaries**
- Objects and Classes
  - What is Object-Oriented Design?
- Finding Classes
  - Class Selection Rationale
- CRC Cards
- Identifying Responsibilities
- Finding Collaborations



6

6

Decomposition

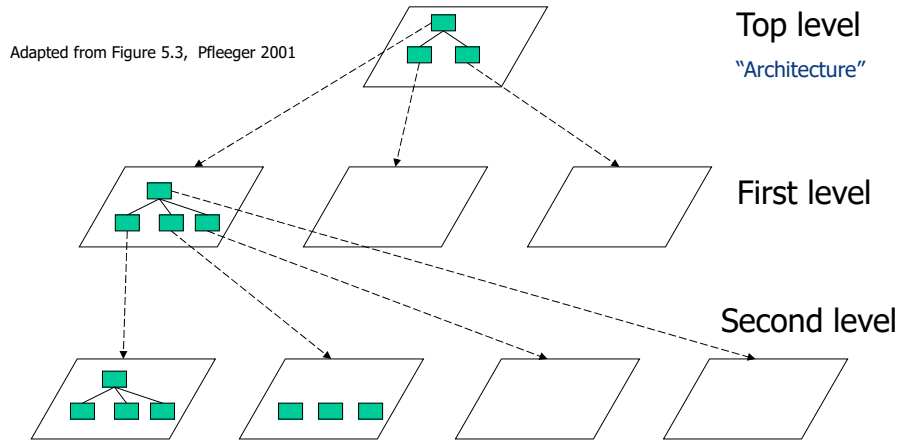
Decomposition is a process that helps to manage the *complexity* of the system

- ☐ Starting with a high-level view of the system.
- ☐ Creating the low-level details of the features of the system in turn.
- ☐ Stop when we are satisfied with the level of detail.
- ☞ Also known as *Divide-and-Conquer*.

7

7

Decomposition (cont.)



8

8

Coupling and Cohesion

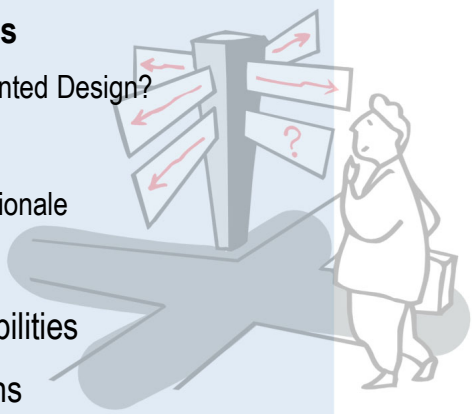
- Two measures of how well the parts of a system are *tied together* to form the system itself.
- They also give good indication of how "good" the design is
- Coupling:
  - measure the *inter-dependency between system parts*
- Cohesion:
  - measure the *intra-dependency of a system part*
- Goal: Weak Coupling and Strong Cohesion
  - ☞ *Good Object-oriented design attempts to achieve this!*

9

9

Roadmap

- Preliminaries
- **Objects and Classes**
  - What is Object-Oriented Design?
- Finding Classes
  - Class Selection Rationale
- CRC Cards
- Identifying Responsibilities
- Finding Collaborations



10

10

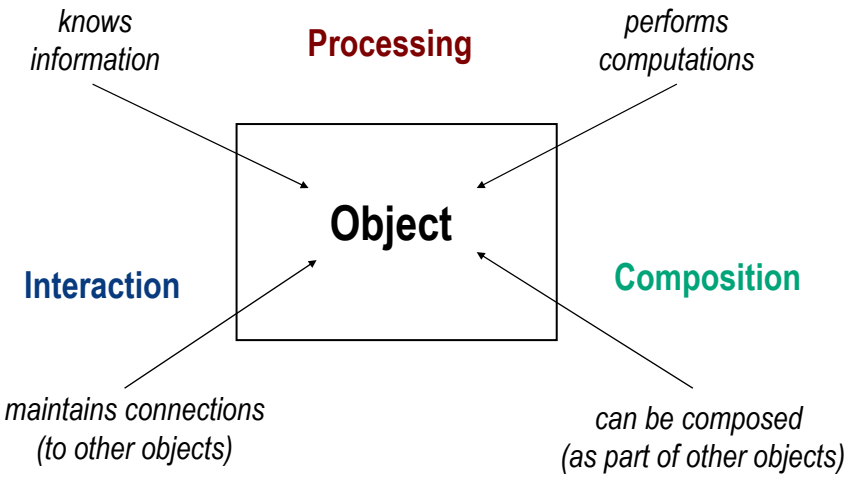
What is an Object?

- Analysis and modeling view:
  - an *abstract model of a real world entity* that has an independent *identity*.
  - E.g., Computer, Person, Car
- Design and implementation view
  - a software component that contains *a collection of data and related procedures*.
- An object
  - represents something in real world that has an *independent identity*.
  - is something that you can interact with:
    - receive and response to messages.
  - has *behaviour* which may depend on its internal *state* which may change by responding to messages.
- Object vs Class

11

11

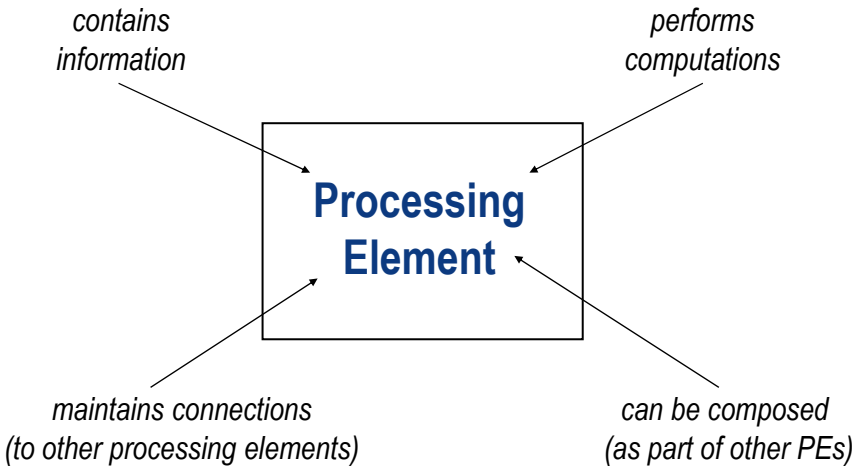
What is an Object?



12

12

In fact…



13

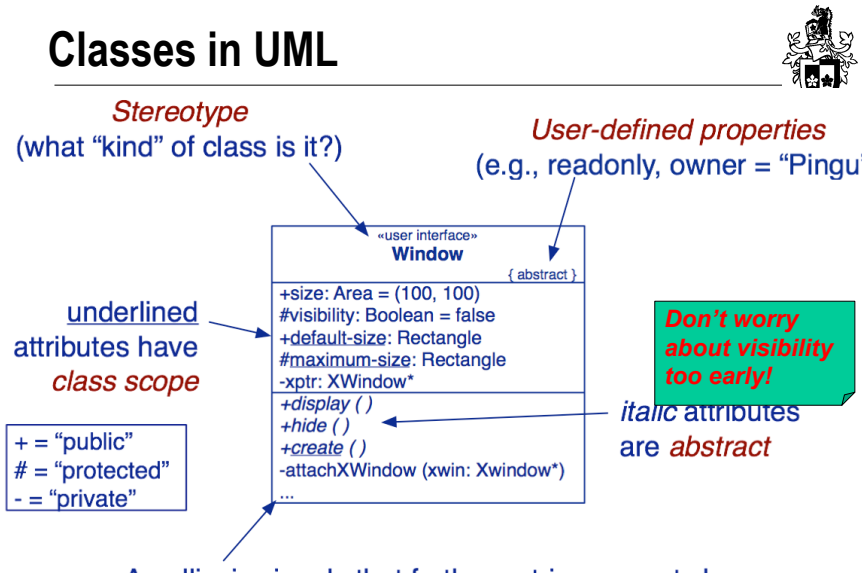
13

What is a Class?

- A class is a description of a *group of objects* with
  - common data attributes
  - common operations
  - common relationships to objects; and
  - common semantics (i.e. "meaning")
- Classes are used to *create* objects:
  - Such objects are called *instances* of their creating class
- Object-Oriented Design often focuses on *identifying classes*, not objects! … but,
- Analysis & dynamic behaviour in terms of objects[14]

14

Classes in UML



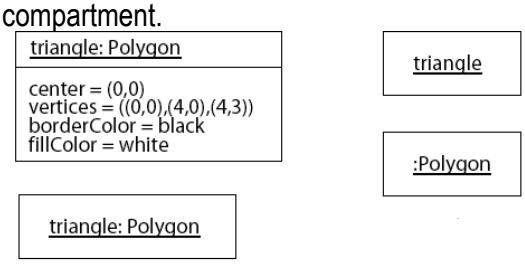
*** *In high-level design, less concerned about details (methods, attributes, etc)!!!*

15

15

Objects in UML

Objects are shown as rectangles with their name and type underlined in one compartment, and attribute values, optionally, in a second compartment.



At least one of the name or the type must be present.

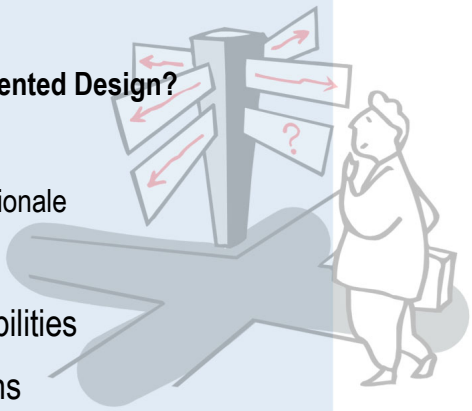
*** *In high-level design, less concerned about details (methods, attributes, etc)!!!*

16

16

Roadmap

- Preliminaries
- Objects and Classes
  - **What is Object-Oriented Design?**
- Finding Classes
  - Class Selection Rationale
- CRC Cards
- Identifying Responsibilities
- Finding Collaborations



17

17

Object-oriented Design

- A *method* of design encompassing the process of *object-oriented decomposition*.
- A *notation* for depicting the (detailed) characteristics of the system under design
- View the system as the *interactions* of different *objects*.
  - Problem: many OO design approaches focus too much on the static relationships…

18

18

Object-oriented Design (cont.)

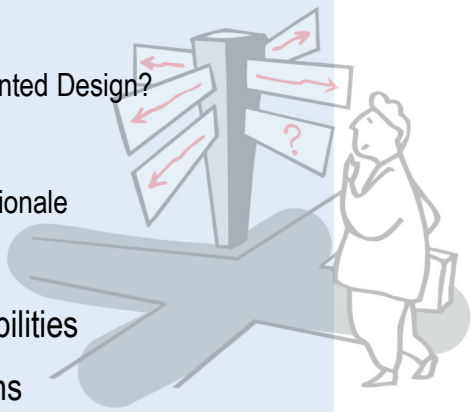
- Goal:
  - To find the important *object-based abstractions* of the system
  - What happens if we find the wrong abstraction?
    - ☞ Iterate and correct the model
- Questions (of importance)
  - How can we *identify* objects (or classes)?
  - How do objects *interact*?
  - What *roles* and *responsibilities* do objects perform?
  - ***… before writing down in UML/program!***

19

19

Roadmap

- Preliminaries
- Objects and Classes
  - What is Object-Oriented Design?
- **Finding Classes**
  - Class Selection Rationale
- CRC Cards
- Identifying Responsibilities
- Finding Collaborations



20

20

Why Object-Oriented Design?

***Functional* Decomposition:**

Decompose according to the functions a system is supposed to perform.

- *Good in a "waterfall" approach: stable requirements and one monolithic function*

However

- Naive: modern systems perform *more than one function*
- Maintainability: system functions evolve ⇒ redesign affect whole system
- Interoperability: interfacing with other system is difficult

21

21

Why Object-Oriented Design (cont.)?

***Object-Oriented* Decomposition:**

Decompose according to the objects a system is supposed to manipulate.

- Better for complex and evolving systems

However

- *How to find the objects?*

22

22

Iteration in Object-Oriented Design

- The result of the design process is *not a final product:*
  - design decisions may be *revisited*, even after implementation
  - design is rarely linear but *iterative*
- The design process is *not algorithmic*:
  - a design method provides *guidelines*, not fixed rules
  - a good *sense of style* often helps produce clean, elegant designs — designs that make a lot of sense from the engineering standpoint

Responsibility-driven design is an (analysis and) design technique that works well in combination with various methods and notations.

23

23

Case Study: Drawing Editor

The drawing editor is an interactive graphics editor. With it, users can create and edit drawings composed of lines, rectangles, ellipses and text.

Tools control the mode of operation of the editor. Exactly one tool is active at any given time.

Two kinds of tools exist: the selection tool and creation tools. When the selection tool is active, existing drawing elements can be selected with the cursor. One or more drawing elements can be selected and manipulated; if several drawing elements are selected, they can be manipulated as if they were a single element. Elements that have been selected in this way are referred to as the current selection. The current selection is indicated visually by displaying the control points for the element. Clicking on and dragging a control point modifies the element with which the control point is associated.

When a creation tool is active, the current selection is empty. The cursor changes in different ways according to the specific creation tool, and the user can create an element of the selected kind. After the element is created, the selection tool is made active and the newly created element becomes the current selection.

The text creation tool changes the shape of the cursor to that of an I-beam. The position of the first character of text is determined by where the user clicks the mouse button. The creation tool is no longer active when the user clicks the mouse button outside the text element. The control points for a text element are the four corners of the region within which the text is formatted. Dragging the control points changes this region. The other creation tools allow the creation of lines, rectangles and ellipses. They change the shape of the cursor to that of a crosshair. The appropriate element starts to be created when the mouse button is pressed, and is completed when the mouse button is released. These two events create the start point and the stop point.

The line creation tool creates a line from the start point to the stop point. These are the control points of a line. Dragging a control point changes the end point.

The rectangle creation tool creates a rectangle such that these points are diagonally opposite corners. These points and the other corners are the control points. Dragging a control point changes the associated corner.

The ellipse creation tool creates an ellipse fitting within the rectangle defined by the two points described above. The major radius is one half the width of the rectangle, and the minor radius is one half the height of the rectangle. The control points are at the corners of the bounding rectangle. Dragging control points changes the associated corner.

24

24

The Initial Exploration

1. Find the *classes* in your system.
2. Determine the *responsibilities* of each class.
3. Determine how objects *collaborate* with each other to fulfill their responsibilities.

These will be discussed in this lecture.

25

25

The Detailed Analysis

1. *Factor* common responsibilities to build class hierarchies
2. *Streamline* collaborations between objects:
   - Is message traffic heavy in parts of the system?
   - Are there classes that collaborate with everybody?
   - Are there classes that collaborate with nobody?
   - Are there groups of classes that can be seen as subsystems?
3. Apply *design heuristics* to improve specific design aspects

These will be discussed in the next lecture.

26

26

Finding Classes

Start with the requirements:

What are the goals of the system being designed, its expected inputs, and desired responses?

1. Look for *noun phrases*:
   - separate into obvious classes, uncertain candidates, and nonsense.
   - *** *Note: we did something similar to create a domain model! – extend/refine and revise here.*

27

27

Finding Classes (cont.)

2. Refine to a list of *candidate classes*. Some guidelines are:
   - Model *physical objects* — e.g. disks, printers
   - Model *conceptual entities* — e.g. windows, files
   - Choose *one word for one concept* — what does it mean within the system
   - Be wary of *adjectives* — is it really a separate class?
   - Be wary of *missing or misleading subjects* — rephrase in active voice
   - Model *categories of classes* — delay modeling of inheritance
   - Model *interfaces* to the system — e.g., user interface, program interfaces
   - Model attribute *values*, not attributes — e.g., Point vs. Centre

28

28

Drawing Editor: Noun Phrases

The drawing editor is an interactive graphics editor. With it, users can create and edit drawings composed of lines, rectangles, ellipses and text.

Tools control the mode of operation of the editor. Exactly one tool is active at any given time.

Two kinds of tools exist: the selection tool and creation tools. When the selection tool is active, existing drawing elements can be selected with the cursor. One or more drawing elements can be selected and manipulated; if several drawing elements are selected, they can be manipulated as if they were a single element. Elements that have been selected in this way are referred to as the current selection. The current selection is indicated visually by displaying the control points for the element. Clicking on and dragging a control point modifies the element with which the control point is associated.

When a creation tool is active, the current selection is empty. The cursor changes in different ways according to the specific creation tool, and the user can create an element of the selected kind. After the element is created, the selection tool is made active and the newly created element becomes the current selection.

29

29

The text creation tool changes the shape of the cursor to that of an I-beam. The position of the first character of text is determined by where the user clicks the mouse button. The creation tool is no longer active when the user clicks the mouse button outside the text element. The control points for a text element are the four corners of the region within which the text is formatted. Dragging the control points changes this region. The other creation tools allow the creation of lines, rectangles and ellipses. They change the shape of the cursor to that of a crosshair. The appropriate element starts to be created when the mouse button is pressed, and is completed when the mouse button is released. These two events create the start point and the stop point.

The line creation tool creates a line from the start point to the stop point. These are the control points of a line. Dragging a control point changes the end point.

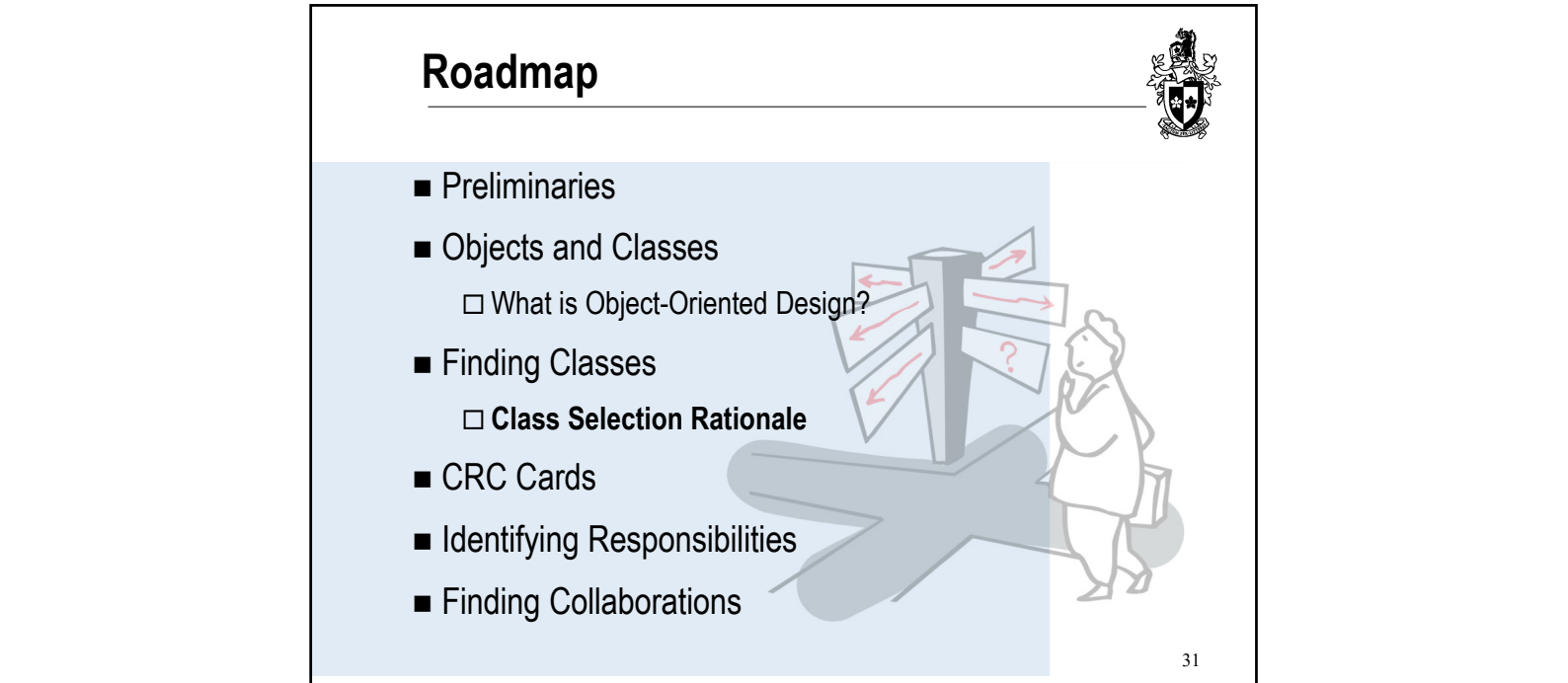
The rectangle creation tool creates a rectangle such that these points are diagonally opposite corners. These points and the other corners are the control points. Dragging a control point changes the associated corner.

The ellipse creation tool creates an ellipse fitting within the rectangle defined by the two points described above. The major radius is one half the width of the rectangle, and the minor radius is one half the height of the rectangle. The control points are at the corners of the bounding rectangle. Dragging control points changes the associated corner.

30

30

Roadmap



- Preliminaries
- Objects and Classes
  - What is Object-Oriented Design?
- Finding Classes
  - **Class Selection Rationale**
- CRC Cards
- Identifying Responsibilities
- Finding Collaborations

31

31

Class Selection Rationale

Model physical objects:

- ~~mouse button~~ [event or attribute]

Model conceptual entities:

- ellipse, line, rectangle
- Drawing, Drawing Element
- Tool, Creation Tool, Ellipse Creation Tool, Line Creation Tool, Rectangle Creation Tool, Selection Tool, Text Creation Tool
- Text, Character
- Current Selection

32

32

Class Selection Rationale (II)

Choose one word for one concept:

- Drawing Editor ⇒
  ~~editor, interactive graphics editor~~
- Drawing Element ⇒ ~~element~~
- Text Element ⇒ ~~text~~
- Ellipse Element, Line Element, Rectangle Element
  ⇒ ~~ellipse, line, rectangle~~

33

33

Class Selection Rationale (III)

Be wary of adjectives:

- Ellipse Creation Tool, Line Creation Tool, Rectangle Creation Tool, Selection Tool, Text Creation Tool
  - *all have different requirements*
- ~~bounding rectangle, rectangle, region~~ ⇒ Rectangle
  - *common meaning, but different from Rectangle Element!*
- Point ⇒ ~~end point, start point, stop point~~
- Control Point
  - *more than just a coordinate*
- corner ⇒
  ~~associated corner, diagonally opposite corner~~
  - *no new behaviour*

34

34

Class Selection Rationale (IV)

Be wary of sentences with missing or misleading subjects:

- "The current selection is indicated visually by displaying the control points for the element."
  - *by what? Assume Drawing Editor ...*

Model categories:

- Tool, Creation Tool

Model interfaces to the system: — no good candidates here ...

- ~~user~~ — *don't need to model user explicitly*
- ~~cursor~~ — *cursor motion handled by operating system*

35

35

Class Selection Rationale (V)

Model values of attributes, not attributes themselves:

- ~~height of the rectangle, width of the rectangle~~
- ~~major radius, minor radius~~
- ~~position~~ — *of first text character; probably Point attribute*
- ~~mode of operation~~ — *attribute of Drawing Editor*
- ~~shape of the cursor, I beam, crosshair~~ — *attributes of Cursor*
- ~~corner~~ — *attribute of Rectangle*
- ~~time~~ — *an implicit attribute of the system*

36

36

Candidate Classes

Preliminary analysis yields the following candidates:

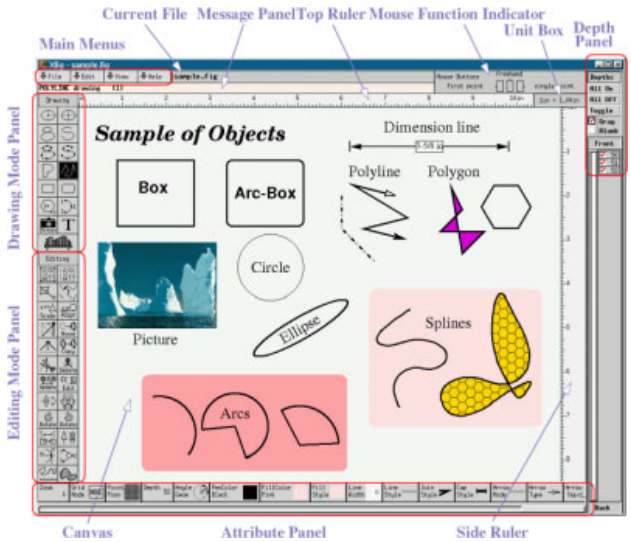
Character	Line Element
Control Point	Point
Creation Tool	Rectangle
Current Selection	Rectangle Creation Tool
Drawing	Rectangle Element
Drawing Editor	Selection Tool
Drawing Element	Text Creation Tool
Ellipse Creation Tool	Text Element
Ellipse Element	Tool
Line Creation Tool	

Expect the list to evolve as design progresses.

37

37

xfig



38

38

A Quicker Starting Point …

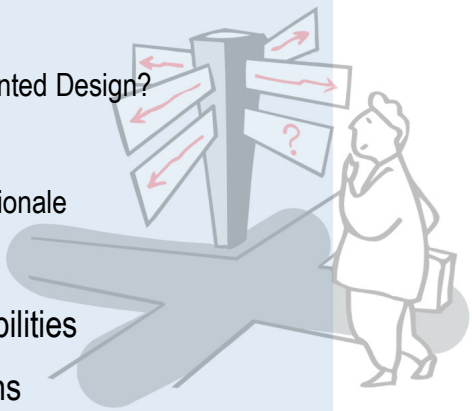
1. The Domain Model from Requirements Specification -- but very coarse/high-level
2. Refined/detailed … with more classes
3. Try the Hotel Management System … (key?, receptionist?, payment? …)

39

39

Roadmap

- Preliminaries
- Objects and Classes
  - What is Object-Oriented Design?
- Finding Classes
  - Class Selection Rationale
- **CRC Cards**
- Identifying Responsibilities
- Finding Collaborations



40

40

Class Responsibility Collaboration Cards

Use CRC cards to record candidate classes:

Text Creation Tool	*subclass of Tool*
Editing Text	

Record the candidate *Class Name* and *superclass* (not yet known)

Record each *Responsibility* and the *Collaborating classes*

- compact, easy to manipulate, easy to modify or discard!
- easy to arrange, reorganize
- easy to retrieve discarded classes

41

41

CRC Cards (cont.)

CRC cards are *not* a specification of a design.

They are a *tool* to *explore* possible designs.

- Prepare a CRC card for *each candidate class*
- Get a team of Developers to *sit around a table* and distribute the cards to the team
- The team *walks through scenarios*, playing the roles of instances of the identified (candidate) classes.

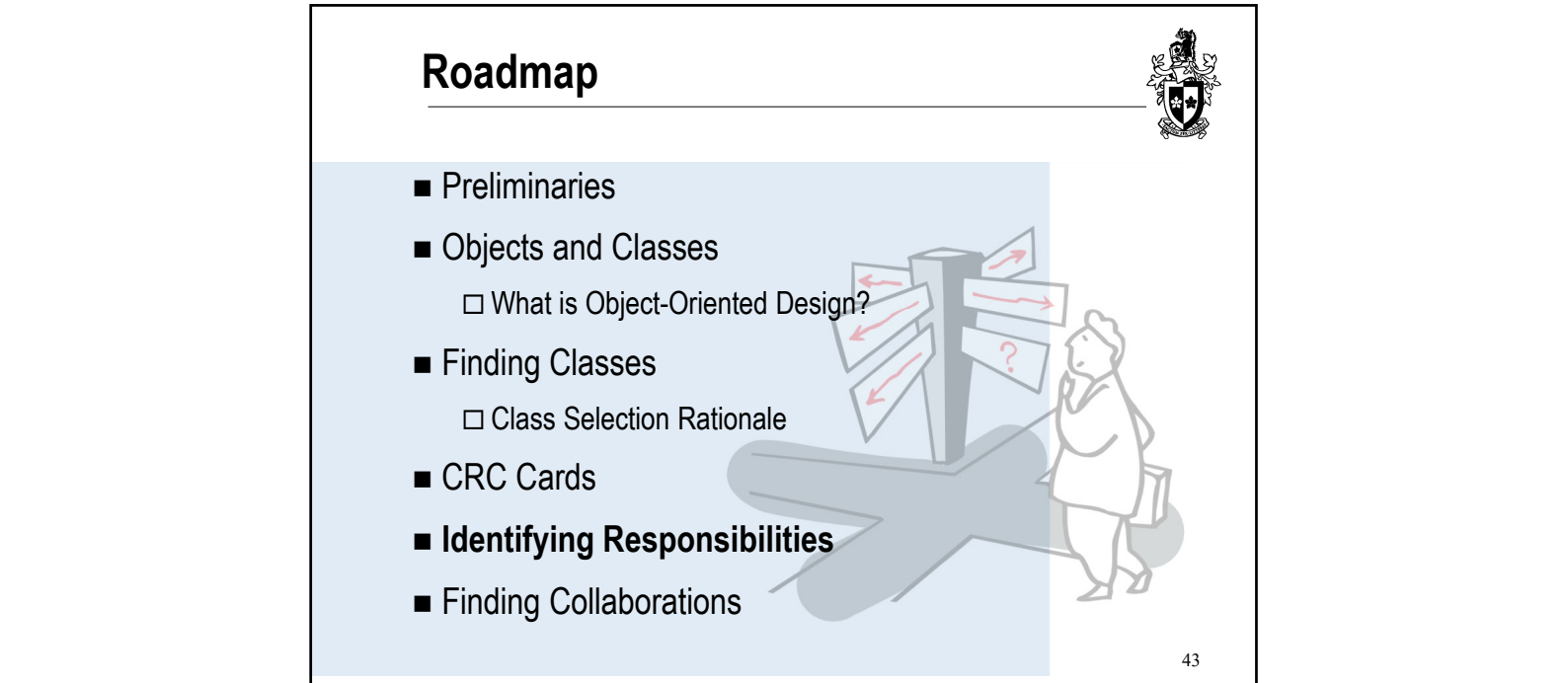
This exercise will uncover:

- *unneeded* classes and responsibilities, and
- *missing* classes and responsibilities.

42

42

Roadmap



- Preliminaries
- Objects and Classes
  - What is Object-Oriented Design?
- Finding Classes
  - Class Selection Rationale
- CRC Cards
- **Identifying Responsibilities**
- Finding Collaborations

43

43

Responsibilities

What are responsibilities?

- the *knowledge* an object maintains and provides,
- the *actions* it can perform.

Responsibilities represent the *public services* an object may provide to clients (but not the way in which those services may be implemented)

- specify *what* an object does, not *how* it does it.
- Do not describe the interface yet, only *conceptual responsibilities*.

44

44

Identifying Responsibilities

- Study the requirements specification:
  - highlight *verbs* and determine which represent responsibilities
  - perform a *walk-though* of the system
    - explore as many scenarios as possible
    - identify *actions resulting from input* to the system
- Study the candidate classes:
  - class names ⇒ roles ⇒ responsibilities
  - recorded purposes on class cards ⇒ responsibilities

45

45

How to assign responsibility?

Pelrine's Laws:

- *"Don't do anything you can push off to someone else."*
- *"Don't let anyone else play with you."*

46

46

Assigning Responsibilities

- *Evenly distribute* system intelligence
  - avoid procedural centralization of responsibilities
  - keep responsibilities close to objects rather than their clients
  - ☞ *Note: this is not always possible (or even desirable)!*
- State responsibilities as *generally* as possible
  - "draw yourself" vs. "draw a line/rectangle etc."
  - leads to sharing
- Keep *behavior* together with any *related information*
  - principle of *encapsulation*.

47

47

Assigning Responsibilities (cont.)

- Keep information about one thing in *one place*
  - if multiple objects need access to the same information
    1. a new object may be introduced to manage the information, or
    2. one object may be an obvious candidate, or
    3. the multiple objects may need to be collapsed into a single one
- *Share* responsibilities among related objects
  - break down complex responsibilities
- Maintaining information is a responsibility
  - But: avoid "data-holder" only classes!
- E.g., Drawing Editor, Drawing, Drawing Element

48

48

Relationships Between Classes

Additional responsibilities can be uncovered by examining relationships between classes, especially:

- The *"Is-Kind-Of"* Relationship:
  - classes sharing a *common attribute* often share a *common superclass*
  - common super-classes suggest *common responsibilities*

☞ To create a new Drawing Element, a Creation Tool must:

1.	accept user input	*implemented in subclass*
2.	determine location to place it	*generic*
3.	instantiate the element	*implemented in subclass*

49

49

Relationships Between Classes (cont.)

- The *"Is-Analogous-To"* Relationship:
  - *similarities* between classes suggest as-yet-undiscovered superclasses
- The *"Is-Part-Of"* Relationship:
  - *distinguish* (don't share) responsibilities of *part* and of *whole*

Difficulties in assigning responsibilities suggest:

- *missing classes* in design, or — e.g., Group Element,
- *free choice* between multiple classes.

50

50

Example Relationships

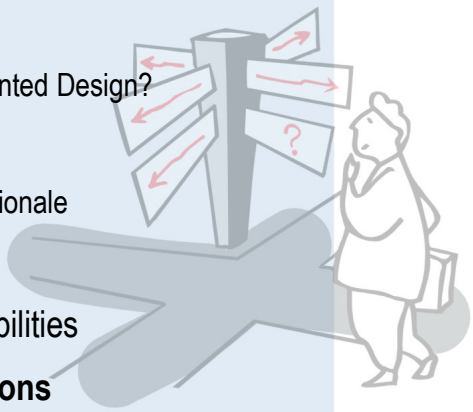
- Drawing Element *is-part-of* Drawing
- Drawing Element *has-knowledge-of* Control Points
- Rectangle Tool *is-kind-of* Creation Tool

51

51

Roadmap

- Preliminaries
- Objects and Classes
  - What is Object-Oriented Design?
- Finding Classes
  - Class Selection Rationale
- CRC Cards
- Identifying Responsibilities
- **Finding Collaborations**



52

52

Collaborations

What are collaborations?

- *collaborations* are *client requests* to servers needed to fulfill responsibilities (**one directional**!)
- collaborations reveal *control and information flow*
- collaborations can uncover *missing responsibilities*,
- analysis of communication patterns can reveal *misassigned* responsibilities.

53

53

Finding Collaborations

For each responsibility:

1. Can the class *fulfill* the responsibility *by itself?*
2. If not, *what does it need*, and from what other class can it obtain what it needs?

For each class:

1. What does this class *know*?
2. What *other classes* need its information or results? Check for collaborations.
3. Classes that *do not interact* with others should be *discarded*. (Check carefully!)

54

54

Listing Collaborations

Drawing	
Knows which elements it contains	
Maintains order of elements	Drawing Element

55

55

Summary

An application = a set of interacting *objects*

An object = an implementation of one or more *roles*

A role = a set of related *responsibilities*

A responsibility = an obligation to perform a task or know certain information

A collaboration = an *interaction* of objects *via* roles

56

56

Questions for Review

1. How do classes differ from objects?
2. Coupling and cohesion are two important concepts to consider when designing a software system. How can the requirements for each be achieved and what needs to be considered with the knowledge that the software will expand over time?
3. Given the fact that future releases may be ported to different platforms, how can the software be designed in a way that anticipates this change?
4. Why is it important to differentiate interfaces from implementation? What happens if this is not adhered to?
5. When assigning responsibilities for objects in OO design, why is it recommended to avoid 'data holder' only classes?

57

57

Question to Answer – week 5 (for week 6)

The spec of the "Question to Answer" is under the corresponding assignment setup, which will be released this evening after the lecture.

58

58

Required Reading Lecture 6

- Robert C. Martin, *UML for Java Programmers*, Prentice Hall, 2003, Chapter 11 (available from Canvas).
- Bertrand Meyer, *Applying "Design by Contract"*, IEEE Computer, October 1992, (available from Canvas).

59

59