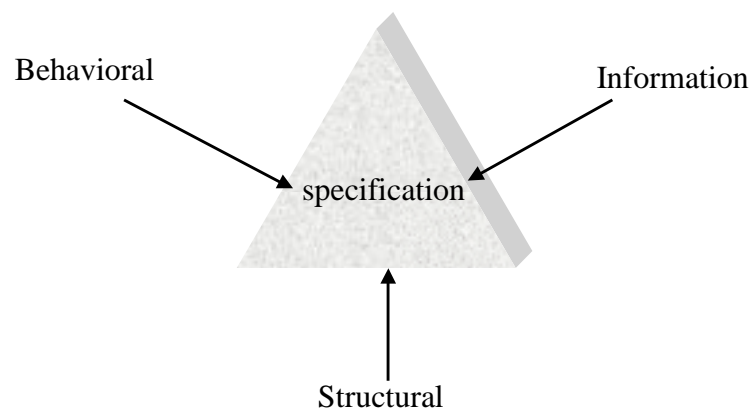


Chapter 5

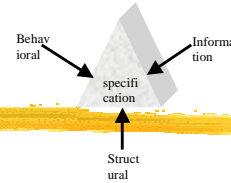
Evaluating Class Designs

Section 9.2:

Class Design: Perspectives

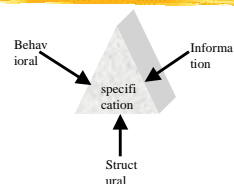


3 Perspectives



- Behavioral:
 - Emphasizes *actions* in system
- Structural:
 - Emphasizes relationships among components
- Information:
 - Emphasizes role of information/data/state and how it's manipulated

Examples of each perspective for Project 1



- Behavioral (actions):
 - Shapes move
 - Click mouse to score
- Structural (relationships):
 - Circles and Squares are kinds of shapes
 - 2 windows contain shapes; 1 contains controls
- Information (state):
 - What's user's score?
 - How many shapes?
 - What are colors, sizes, locations of each shape?

Behavioral Perspective

Consider some action in a program...

What object...

- | initiates action?

What objects...

- | help perform action?
- | are changed by action?
- | are interrogated during action?

Behavioral Perspective

Consider mouse click action

What object...

- | initiates action? ⇒ **User**

What objects...

- | help perform action?
⇒ **Globals (OnMouseEvent)**
- | are changed by action?
⇒ **Possibly some shape**
- | are interrogated during action?
⇒ **ShapeManager (sees if mouse lies in any shape)**

Behavioral Classes

- **Actor** (does something)
⇒ Game object with Start() & Stop()
- **Reactor** (system events, external & user events)
⇒ mouse click reactor, buttons
- **Agent** (messenger, server, finder, communicator)
⇒ Communicator lets user choose file to load old game
- **Transformer** (data formatter, data filter)

Structural Perspective

- What objects...
 - are involved in relationship?
 - are necessary to sustain (implement, realize, maintain) relationship?
- What objects not in relationship...
 - are aware of and exploit relationship?

Structural Perspective

"2 windows contain shapes; 1 contains controls"

■ *What objects...*

- are involved in relationship?

 - ⇒ **Frames, Shapes, Controls**

- are necessary to realize relationship?

 - ⇒ **ShapeManager (maps shapes to frames)**
ControlArea (holds buttons, sliders, ...)

■ *What objects not in relationship...*

- are aware of and exploit relationship?

 - ⇒ **Clock associates with ShapeManager,**
which distributes ticks to all Shapes

Structural Classes

■ **Acquaintance** (symmetric, asymmetric)

- ShapeManager might be asymmetric
(it knows about shapes and frames, but
frames/shapes are unaware of ShapeManager)

■ **Containment** (collaborator, controller)

- ControlArea class contains buttons, textboxes,...

■ **Collection** (peer, iterator, coordinator)

- ShapeManager might have iterator so external
user can traverse all shapes

Information Perspective

What objects...

- represent the data or state?
- read data or interrogate state?
- write data or update state?

Data Versus State

Data

• Definition:
Info processed by system

• Example:
Text you type into a word processor

State

• Definition:
Info used by system to perform processing

• Example:
Variables specifying font, size, width

Information Perspective

What objects...

- represent the state?
 - ⇒ **Circle, Rectangle give color, size, location**
- read data or interrogate state?
 - ⇒ **ShapeManager might ask Shape if it contains x,y mouse coordinate**
- write data or update state?
 - ⇒ **ShapeManager might tell Shapes to move**

Summary

- Try creating Table 9-5 for Project 1!

Behavioral Perspective

- Actors: InterfaceManager
- Reactors: DisplayManager, EventManager
- Agent:
 - Servers: FileReader
 - Communicator: CheckBox,Slider, TextBox, Button

Behavioral Perspective

- Transformer:
 - Filter: Filter
 - Formatter: Plotter

Structural Perspective

- Acquaintance:
 - Symmetric:
 - | None
 - Asymmetric:
 - | EventManager with FileReader, Timer, DisplayManager
 - | InterfaceManager with DisplayManager
 - | DisplayManager with FileReader

Structural Perspective

- Containment:
 - Collaborator:
 - | User Manager contains Frame, Panel, Sliders, CheckBox, TextBox, Buttons
 - | DisplayManager contains Canvas, Color, Symbol, Plotter, Filter
 - Controller: EventManager controls DisplayManager and InterfaceManager
- No collection objects

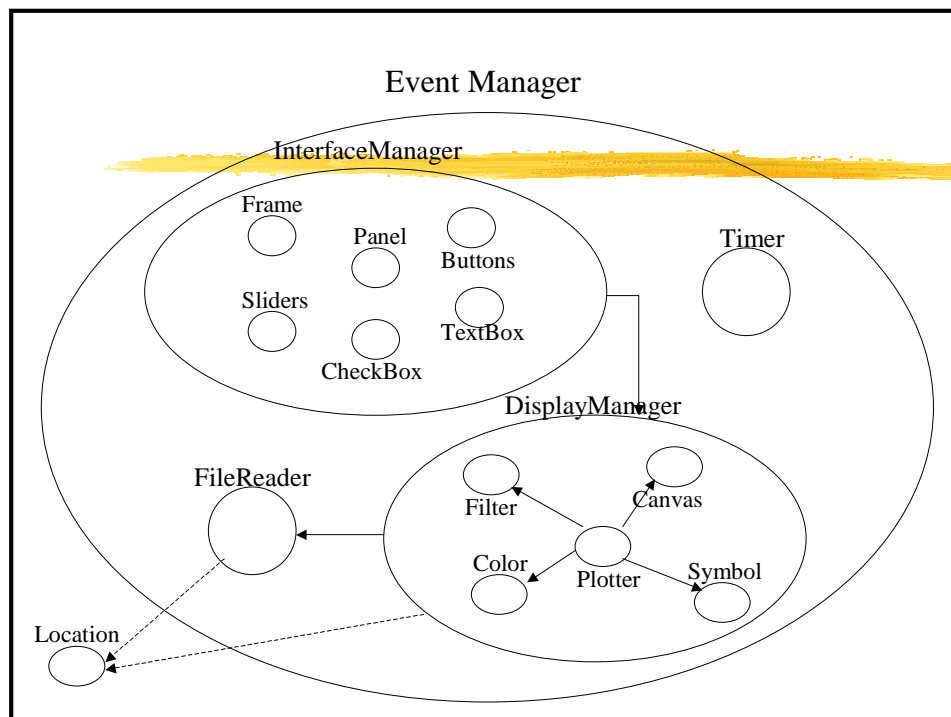
Information Perspective

■ Data:

- Source: FileReader
- Sink: Plotter
- Result: Location
- Synchronizers: Timer, EventManager

■ State:

- There are is real state information stored



Section 9.2, page 389:

Evaluating a Class Design

- Evaluation needed to accept, revise or reject a class design
- Five aspects to be evaluated:
 - Abstraction: useful?
 - Responsibilities: reasonable?
 - Interface: clean, simple?
 - Usage: "right" set of methods?
 - Implementation: reasonable?

Tests to determine adequacy of Abstraction

- **Identity:**
Are class & method names simple & suggestive?
- **Clarity:**
Can purpose of class be given in brief, dictionary-style definition?
- **Uniformity:**
Do operations have uniform level of abstraction?

Good or Bad Abstractions?

- *class Date:*

Date represents a specific instant in time, with millisecond precision.

- *class TimeZone:*

- TimeZone represents a time zone offset, and also figures out daylight savings.

Tests to determine adequacy of Responsibilities

- **Clear:**

Does class have specific responsibilities?

- **Limited:** Do responsibilities should fit abstraction (no more/less)?

- **Coherent:**

Do responsibilities make sense as a whole?

- **Complete:**

Does class completely capture abstraction?

Tests to determine adequacy of Interface

- **Naming:**
Do names clearly express intended effect?
- **Symmetry:** Are names & effects of pairs of inverse operations clear?
- **Flexibility:**
Are methods adequately overloaded?
- **Convenience:**
Are default values used when possible?

Example of Poor Naming:

```
class ItemList
{ ...
public:
    void Delete(Item item);
        // take Item's node out of list and delete Item
    void Remove(Item item);
        //take Item's node out of the list but do not delete Item
    void Erase(Item item);
        //keep Item's node in List, but with no information
};
```

Hard to remember difference!

Tests to determine adequacy of Usage

- Examine how objects of the class are used in different contexts (see next slide...)
- Incorporate all operations that may be useful in these contexts

Original Location Class:

```
class Location {  
private:  
    int xCoord, yCoord; //coordinates  
  
public:  
    Location(int x, int y);  
    int xCoord(); //return xCoord value  
    int yCoord(); //return yCoord value  
};  
  
//usage  
Location point(100,100);  
...  
point = Location( point.xCoord()+5, point.yCoord()+10 ); //shift point
```

It's so complex!

Revised Location Class:

```
class Location {
private:
    int xCoord, yCoord; //coordinates
public:
    Location(int x, int y);
    int XCoord(); //return xCoord value
    int YCoord(); //return yCoord value
    void ShiftBy(int dx, int dy); // shift by relative coordinates
};

//usage
Location point(100,100);
....
point.ShiftBy(5, 10); //shift point
```

Implementation

- Least important, mostly easily changed aspect to be evaluated
- Complex implementation may mean
 - Class not well conceived
 - Class has been given too much responsibility

Section 4.6:

A More Complex Static Aggregation

Consider object with a fixed number of more complicated internal parts.

StopWatch Class Interface

```
class StopWatch{
```

```
private:
```

```
    Button  startButton;  
    Button  stopButton;  
    Clock   clock;  
    Counter clockCount;  
    Message clockDisplay;  
    Panel   buttonPanel;  
    Canvas  canvas;
```

```
public:
```

```
    StopWatch(Frame& frame,  
              Location where,  
              int interval = 1000);  
    void ButtonPushed(char*  
                      buttonName);  
    void Tick();  
    int ElapsedTime();  
    ~StopWatch();  
};
```

StopWatch Class Implementation

```
void Stopwatch::ButtonPushed(char* buttonName)
{ if (startButton.IsNamed(buttonName))
  clock.Start();
  else if (stopButton.IsNamed(buttonName))
  clock.Stop();
}

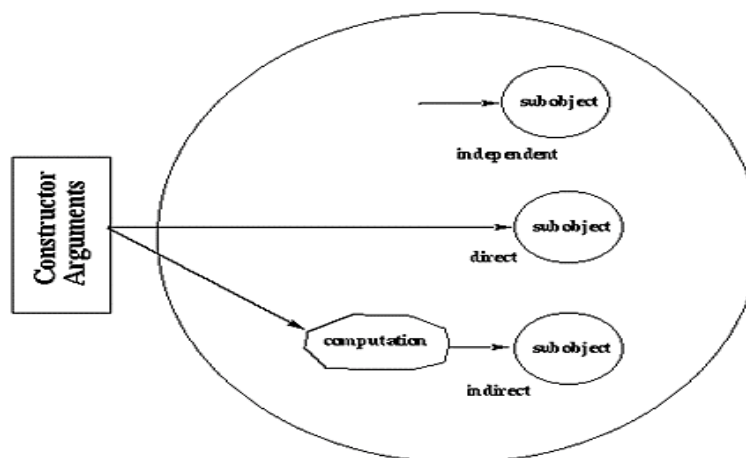
void Stopwatch::Tick(){ clockCount.Next();}

int Stopwatch::ElapsedTime() { return clockCount.Value(); }

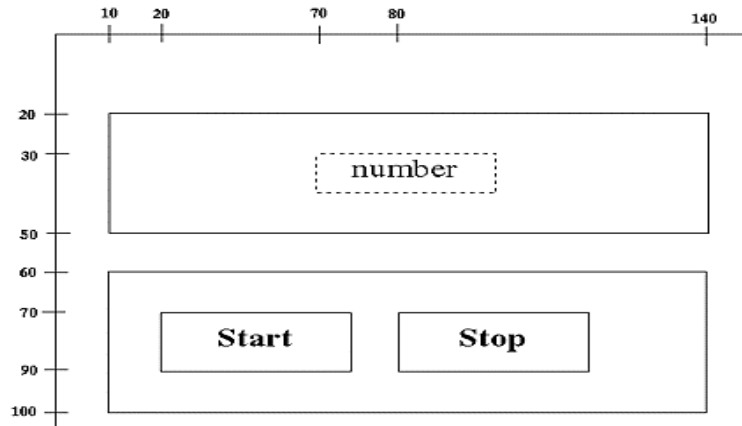
Stopwatch::~~Stopwatch() {}
```

Section 4.6, page 165:

Subobject Construction



Layout of the Stopwatch User-Interface Subobjects



Placement of User Interface Subobjects

Let \underline{x} =where.Xcoord() and \underline{y} =where.Ycoord():

Message: Location(60,10) in Canvas
(shape set automatically)

StartButton: Location(10,10) in Panel
Shape(50,20)

StopButton: Location(70,10) in Panel
Shape(50,20)

Canvas: Location(w+10, y+20) in Frame
Shape(130, 30)

Panel: Location(x+10, y+60) in Frame
Shape(130, 40)

Constructor form

global objects;

```
ClassName::ClassName(args) :  
    constructor(args),  
    constructor(args),  
    ...  
{ constructor body }
```

StopWatch Constructor (1 of 3)

```
//StopWatch Class Constructor  
  
Location StartButtonLocn = Location( 10,10);  
Shape StartButtonShape = Shape ( 50,20);  
Location StopButtonLocn = Location( 70,10);  
Shape StopButtonShape = Shape ( 50,20);  
Location ButtonPanelLocn = Location( 10,60);  
Shape ButtonPanelShape = Shape (130,40);  
Location CanvasLocn = Location( 10,20);  
Shape CanvasShape = Shape (130,30);  
Location ClockDisplayLocn = Location( 60,10);
```

StopWatch Constructor (2 of 3)

```
StopWatch::StopWatch(Frame& frame, Location where,
                    int interval):
// sub-object constructor list

    counter(0),

    clock("StopWatchClock", interval),

    buttonPanel( frame, "ButtonPanel",
        Location(where.Xcoord()+ButtonPanelLocn.Xcoord(),
            where.Ycoord()+ButtonPanelLocn.Ycoord()),
        ButtonPanelShape),

    startButton("Start", StartButtonLocn, StartButtonShape),

    stopButton ("Stop", StopButtonLocn, StopButtonShape),
```

StopWatch Constructor (3 of 3)

```
        canvas ( frame, "StopWatchCanvas",
            Location(where.Xcoord() + CanvasLocn.Xcoord(),
                where.Ycoord() + CanvasLocn.Ycoord()),
            CanvasShape),

        clockDisplay( "0", ClockDisplayLocn)

{
    // constructor body
    buttonPanel.Add(stopButton);
    buttonPanel.Add(startButton);
    canvas.Clear();
    clockCount.ConnectTo(clockDisplay);
    clockDisplay.DisplayIn(canvas);
    clock.Start();
}
```