Interaction Diagrams and Responsibility Assignment
Oct 23, 2006

Recap: CRC Cards Process
1. Initial class list
2. Refined class list
3. while (More use case scenarios left)
4. do
5. Take a use case scenario
6. Assign responsibilities to classes
7. Find super- and sub-classes
8. Find collaborators
9. od
Recap: Class Diagrams

- Class diagrams represent design structure
- Three parts: name, attribute, operations
- Visibility, attribute type, multiplicity
- Association, association multiplicity
- Generalization i.e. interface impl, subclassing
- Composition i.e. class A contains class B
- Applied information hiding, DSM, layering

Today’s Lecture

- Interaction Diagrams Basics & Notations
- Responsibility Assignment Principles
Example: Switch-Motor System

- Scenario: Motor can run, when the Switch state is on
- Class list: Switch and Motor
- R1: Store the decision condition
- R2: Make a decision
- R3: Run the motor

Class Diagram View

```
Switch
  -state: bool
  +getState(): bool

Motor
  +run()
  -switch: Switch
```

Interaction Diagram View

Switch-Motor: CRC Card View

- Scenario: Motor can run, when the Switch state is on
- Class list: Switch and Motor
- R1: Store the decision condition, R2: Make a decision, and R3: Run the motor

<table>
<thead>
<tr>
<th>Class Name: Switch</th>
<th>Class Name: Motor</th>
</tr>
</thead>
<tbody>
<tr>
<td>Subclasses:</td>
<td>Subclasses:</td>
</tr>
<tr>
<td>Super classes:</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Responsibilities</th>
<th>Collaborators</th>
</tr>
</thead>
<tbody>
<tr>
<td>Store decision</td>
<td></td>
</tr>
<tr>
<td>condition</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Responsibilities</th>
<th>Collaborators</th>
</tr>
</thead>
<tbody>
<tr>
<td>Make a decision</td>
<td></td>
</tr>
<tr>
<td>Run the motor</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Switch</td>
</tr>
</tbody>
</table>
Class Diagram View

Impact of Change: Multi-State Switch

Change: Switch can have more than two states.
**Change Escapes Module Boundary**

Switch

- state:int

+ getState():int

Change: Switch can have more than two states.

Motor

- switch:Switch

+ run():void

Impact of Change

```java
void run()
if (switch.getState()) /* Make a decision */
/* run the motor */
```

**Information Expert**

- Which class has the necessary information?
- Example – class list: Switch and Motor
- R1: Store the decision condition, R2: Make a decision, and R3: Run the motor

<table>
<thead>
<tr>
<th>Class Name: Switch</th>
<th>Subclasses:</th>
<th>Super classes:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Responsibilities</td>
<td>Collaborators</td>
<td></td>
</tr>
<tr>
<td>Store decision</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Condition</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Make a decision</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Class Name: Motor</th>
<th>Subclasses:</th>
<th>Super classes:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Responsibilities</td>
<td>Collaborators</td>
<td></td>
</tr>
<tr>
<td>Run the motor</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Switch</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
**Improved Class Diagram View**

```
Switch
  - state: bool
  + isOn(): bool

Motor
  - switch: Switch
  + run(): void

void run()
if (switch.isOn())
  /* run the motor */
```

**Implementation Changes Encapsulated**

```
Switch
  - state: int
  + isOn(): bool

Motor
  - switch: Switch
  + run(): void

void run()
if (switch.isOn())
  /* run the motor */
```

**Change:** Switch can have more than two states.

**Remember: Information Hiding**
Example: User-Level Process Management

- Main scenario: A process will maintain necessary information for it to run such as program counter (PC), reference to program segment (.ps), reference to data segments (.ds), registers (regs)
- Scenario of interest: functionality to create an identical copy of a process should be provided
- Class list: process, client
- R1: store information PC, .ps, .ds. Regs
- R2: create copy of process

User-Level Process Management: CRC Card View

- Class list: process, client
- R1: store information pc, .ps, .ds. regs
- R2: create copy of process
### Change: Register-based to Stack-based Machine

- **Class list:** process, client
- **R1:** store information pc, .stack, .top, .ps, .ds.
- **R2:** create copy of process

<table>
<thead>
<tr>
<th>Class Name: Process</th>
<th>Subclasses:</th>
<th>Super classes:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Responsibilities</td>
<td>Store information pc, .stack, .top, .ds. regs</td>
<td>Collaborators</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Class Name: Client ...</th>
<th>Subclasses:</th>
<th>Super classes:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Responsibilities</td>
<td>Create copy of pc, .ps, .ds, .regs</td>
<td>Collaborators</td>
</tr>
<tr>
<td></td>
<td>Spawn another process</td>
<td>Process</td>
</tr>
</tbody>
</table>

**Impact on all clients**

---

### Creator: Which class has the initializing data?

- **Class list:** process, client
- **R1:** store information pc, .ps, .ds. regs
- **R2:** create copy of process

<table>
<thead>
<tr>
<th>Class Name: Process</th>
<th>Subclasses:</th>
<th>Super classes:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Responsibilities</td>
<td>Store information pc, .ps, .ds. regs</td>
<td>Collaborators</td>
</tr>
<tr>
<td>Create copy of pc, .ps, .ds, .regs and spawn another process</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Class Name: Client ...</th>
<th>Subclasses:</th>
<th>Super classes:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Responsibilities</td>
<td>Create copy of pc, .ps, .ds, .regs</td>
<td>Collaborators</td>
</tr>
</tbody>
</table>

**Change Impact Encapsulated**
Adding GUI to the Sorting Application

- **Which class should handle the Sort Event?**

<table>
<thead>
<tr>
<th>Class Name: UI</th>
<th>Class Name: SortingApp</th>
</tr>
</thead>
<tbody>
<tr>
<td>Subclasses:</td>
<td>Subclasses:</td>
</tr>
<tr>
<td>Super classes:</td>
<td>Super classes:</td>
</tr>
<tr>
<td>Responsibilities:</td>
<td>Responsibilities:</td>
</tr>
<tr>
<td>Handle Sort Event</td>
<td>Create sorting, storage instance</td>
</tr>
<tr>
<td>Collaborators:</td>
<td>Collaborators</td>
</tr>
<tr>
<td>SortingApp</td>
<td>ISort, IStorage etc.</td>
</tr>
</tbody>
</table>

Adding GUI to the Sorting Application

```java
SortButton
... 
/addActionListener(ActionListener l)

SortingApplication
storage:IStorage
sort:ISort
+sort(…);IStorage
/actionPerformed(ActionEvent e)
```

```java
javax.swing.DefaultButtonModel
... 
+addActionListener(ActionListener l)

java.awt.event.ActionListener
... 
+actionPerformed(ActionEvent e)
```
Summary

- Interaction Diagrams
- Responsibility Assignment Principles
  - Information Expert
  - Creator
  - Controller
  - Polymorphism
  - Pure Fabrication
  - Indirection
  - Protected Variations