

Refactoring

Martin Fowler.

Refactoring / Improving the Design of Existing Code.

Addison Wesley, 1999.

Refactoring

Refactoring is the process of changing a software system in such a way that it does not alter the external behavior of the code yet improves its internal structure. It is a disciplined way to clean up code that minimizes the chances of introducing bugs. In essence when you refactor you are improving the design of the code after it has been written.

-- Fowler

Background

- William F. Opdyke. *Refactoring Object-Oriented Frameworks*. PhD dissertation, University of Illinois at Urbana-Champaign, 1992.
- Smalltalk
- Extreme Programming
- Refactoring tools

Process

- Design sanity tests ("OK" or failures)
- Detect design problems (*Bad Smells*)
- Systematically alter the source code of a program in order to improve its design
 - Correctness preserving
 - Small step at a time
 - Test with each step

Refactoring Testing

- Sanity test: way to quickly tell whether you have broken something
- Design tests before you start coding; add to them with each detected bug
 - Can't always be done a class at a time
- For Java, place the unit tests in each class's `main` method
- Junit, C++Unit, SUnit testing frameworks

“If it stinks, change it”

- What is a “bad smell” in code?
 - An *indication* of possible problems with code
- Why sniff for bad smells?
 - In order to identify:
 - A source of error
 - Design weaknesses
 - Code that is hard to understand and maintain
 - Code that is hard to enhance

Example: Duplicated Code

- Problems:
 - Public Enemy #1
 - Requires parallel changes implying defect persistence
 - Indication of design flaw
- Resolution
 - Extract and reference common code
 - If across classes, consider why two classes have similar responsibility?

Example: Long Method

- Problems
 - Hard to understand
 - Hard to maintain
 - More constraints, assumptions
- Detection
 - Section comments?
 - Too much nesting?
 - Too many temporaries?
- Resolution
 - Extract sections into another method
 - Use parameter object, method object

```
// find the max
for( int i=0; i < len(list); i++ )
{
    ...
}

// compute the histogram
for( int i=0; i < len(list); i++ )
{
    ...
}



---


max = Max(list);
hist = Histogram(list);
```

Example: Data Clumps

- Classic example:
 - Using 3 separate ints (R,G,B) to process color
- Problems
 - Method parameters take in R,G,B
 - Code processes components separately
- When data *clumps* together, consider as a candidate for a class.

Example: Type Abuse

- `switch`
 - Candidate for possible extension
 - May want to introduce OO (polymorphism)
- **Typecast**
 - Identify possible locations that can use type-safe containers, improve interfaces
- `instanceof`
 - Double dispatch or hack?

Examples: Change Smells

- Divergent Change
 - Updating part of class when database changes, other part when finance rule changes
- Shotgun Surgery
 - Change involves many locations
- Parallel Inheritance
 - A change to one branch, needs same change in another

Examples: Coupling Smells

- Feature Envy
 - A method is processing/interfaces too much with an foreign object
- Inappropriate Intimacy
 - Two classes are overly intertwined
- Message Chains / Middle Man
 - Law of Demeter: "Only talk to your friends who share your concerns"

Catalog of Refactorings

- Name
- Summary
- Motivation
- Mechanics
- Examples
- *Safety conditions*

Bad Smells

- **Duplicate code** - factor it
- **Long method** - split it
- **Large class** (too many instance variables)
- **Long parameter list** - pass just enough to navigate to what you need
- **Divergent change** (multiple feature categories)
- **Shotgun surgery** (*delocalization*)

Bad Smells - 2

- **Feature envy** (have to frequently visit other classes to get access to data) - move to another class
- **Data Clumps** (subsets of instance variables that are used together) - create new object
- **Primitive obsession** (create your own builtin)
- **switch statements** - use subtypes
- **Parallel inheritance hierarchies**

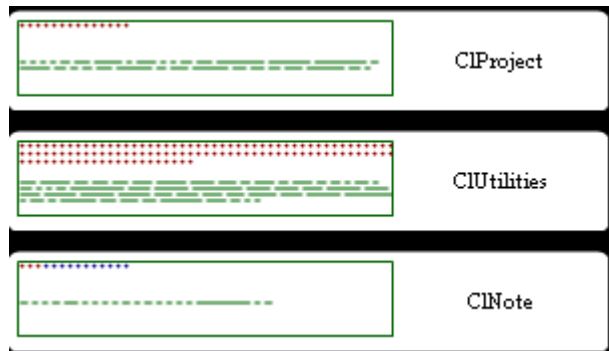
Bad Smells - 3

- **Lazy class** - eliminate
- **Speculative generality** (only used by test cases)
- **Temporary field** (instance variable used only in certain circumstances)
- **Message chains** (sequences of calls)
- **Middle man** (excessive delegation) - refer directly to delegatee

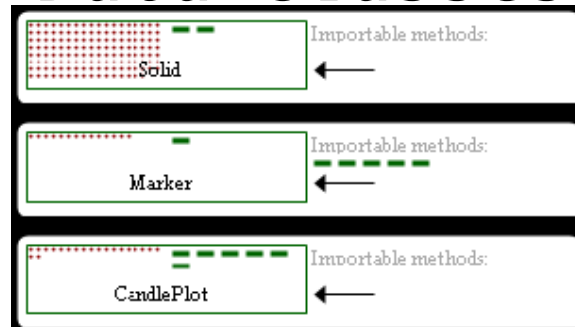
Bad Smells - 4

- **Inappropriate intimacy** (classes refer to each other too often)
- **Alternative classes with different interfaces** - rename methods to indicate similarities
- **Incomplete library class** - extend
- **Data class** - move behavior into class
- **Refused bequest**
- **Comments** - use self documenting names

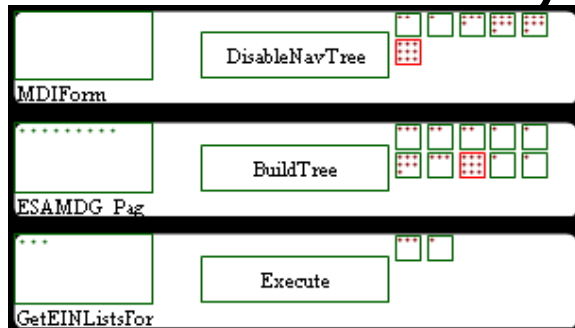
Large Classes



Data Classes

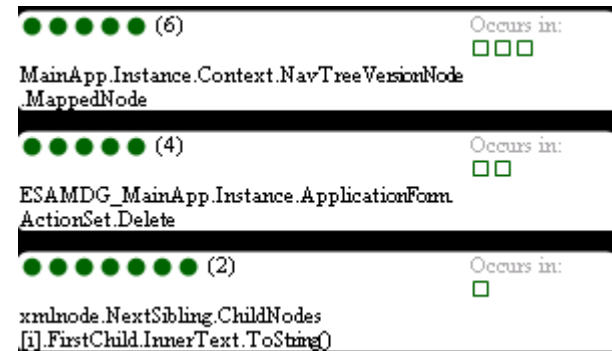


Feature Envy



Light-Weight Visualizations

Message Chains



Long Params



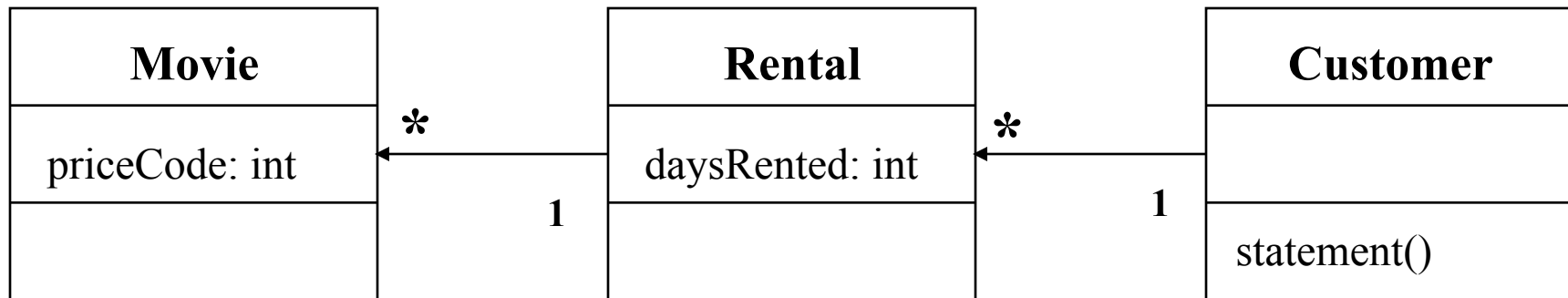
Examples

CIPProject.InitializeComponent() 2762	CIAutomation.exUnitTest() 639	CIPositionInfo.exUnitTest() 591	CIPreplannedThreat.InitializeComponent() 465	CITTypeEmitter.InitializeComponent() 448	CIMdfGenerator.exUnitTest() 397
CIOption.InitializeComponent() 364	CIControlManager.exUnitTest() 360	CIALQ184.InitializeComponent() 346	CIThreatResponse.InitializeComponent() 345	CIAutomation.ParseLatLon() 342	CIThreatEmitter.InitializeComponent() 341
CIFreezeManager.ProtectOrientForm() 325	CIThreat.InitializeComponent() 317	CIPriorityRank.InitializeComponent() 295	CIDisplayReportApp.GetOrientReport() 288	CIMainApp.Initialize() 276	CIEndScanner.FillTable() 273
CIDataGridBase.exUnitTest() 272	CIDispenseProgram.InitializeComponent() 271	CIILocation.InitializeComponent() 268	CICompareNotes.InitializeComponent() 264	CIReportForm.CIReportForm() 255	CIGenerateLoadFile.InitializeComponent() 249
CIIMdfGenerator.ReplaceParameters() 243	CIUnitConstructor.ConstructELU() 238	CIDisplayReportApp.GetDispenseEmitterData() 238	CIAccess.exUnitTest() 235	CIListViewBase.exUnitTest() 230	CINote.InitializeComponent() 228
CIHeaderGenerator.GenerateHeaderFile() 223	CIThreatClass.InitializeComponent() 223	CIUploadManager.UploadProjectToDatabase() 221	CITTypeEmitter.HandleItemChanged() 220	CICreateProject.InitializeComponent() 218	CIListViewBase.InitializeComponent() 213
CIRankThreat.InitializeComponent() 211	CIALQ131.InitializeComponent() 206	CIDisplayReportApp.GetNameEmitterData() 205	CITType.InitializeComponent() 204	CIALQ131Mapping.InitializeComponent() 204	CIProjectBrowser.InitializeComponent() 202
CIGMTThreat.InitializeComponent() 202	CIIMorph.InitializeComponent() 201	CIThreatEmitter.btUpdate_Click() 200	CICometDispensePattern.InitializeComponent() 200	CIPriorityMode.InitializeComponent() 199	CIALQ184Mapping.InitializeComponent() 198
CIUnitConstructor.exUnitTest() 198	CIAgeIn.InitializeComponent() 195	CIFinalPriority.InitializeComponent() 195	CIPriority.exUnitTest() 194	CIPriority.GetCombinationData() 193	CIEmitter.InitializeComponent() 187
CIUploadManager.UploadProjectToDatabaseAsNew() 187	CIEffectiveness.InitializeComponent() 184	CIJammerProgram.InitializeComponent() 181	CIPriority.ConstructPLUT() 180	CIUtilities.exUnitTest() 180	CIEndScanner.ParseEid() 175
CIDataGridControls.InitializeComponent() 171	CIMainApp.Main() 170	CIPriority.OrderFinalPriority() 169	CIPriorityGroup.InitializeComponent() 168	CIPriorityWeight.InitializeComponent() 165	CIProjectBrowser.btOpenProject_Click() 162
CIReportForm.PopUp_Clicked() 159	CICreateProject.btCreateProject_Click() 158	CIReportForm.Menu_Clicked() 157	CIUnitConstructor.GetGEIDData() 156	CIAbout.InitializeComponent() 154	CIIMdfGenerator.FrmLUTConstructor() 150

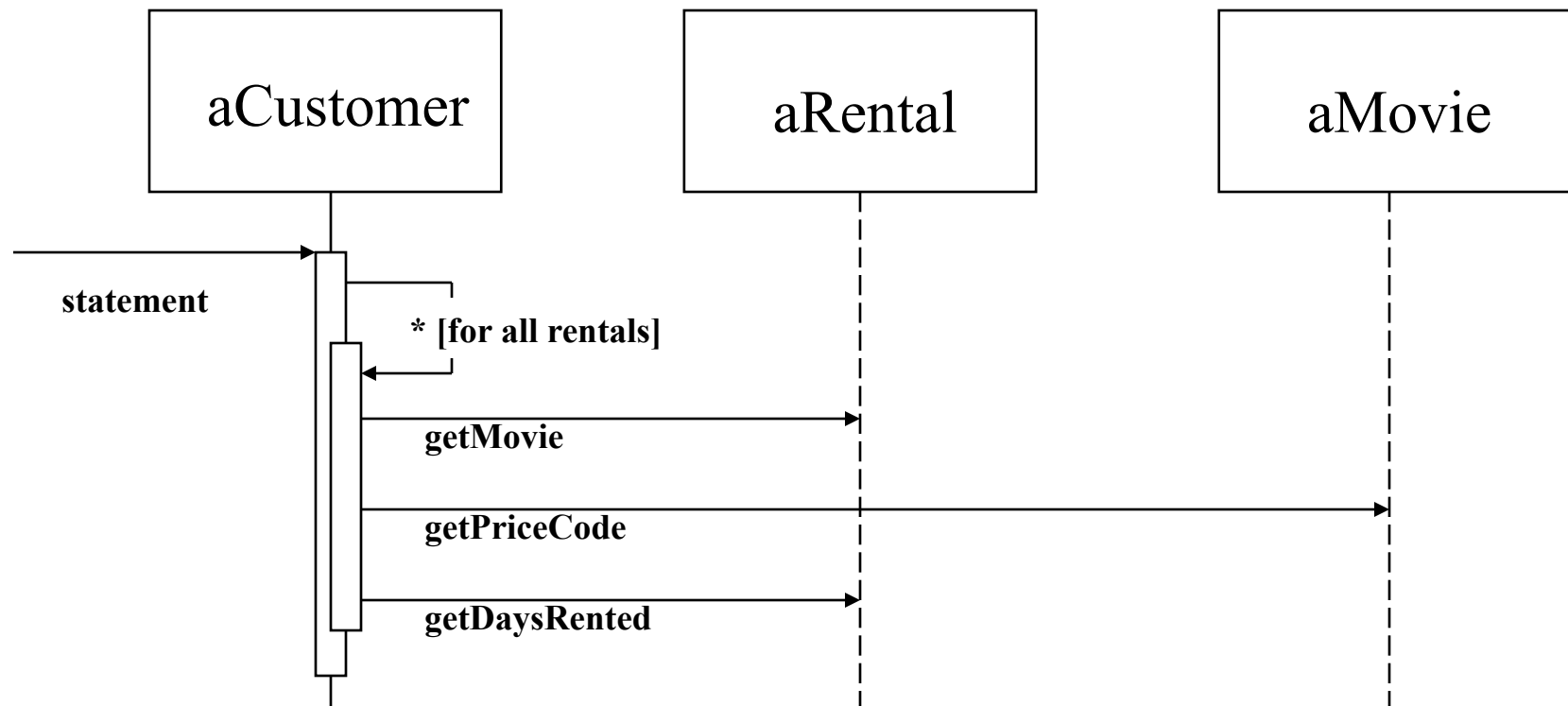
Case Study

- Software to support a video store
- Track rentals; print reports
- Refactor to improve design
- Add enhancements

Class Diagram



Interaction Diagram



Design Goals

- Abstraction
 - Information hiding
- Flexibility
 - Not the same as generality
- Clarity
- Irredundancy
 - "Thrifty" code

Refactoring 1

Decompose and Redistribute Statement Method

- **Bad smell:** `long method`
- **Refactoring:** `extract method`
 - **Handle local variables**
 - **Unmodified** (`each`) passed in as a parameter
 - **Modified** (`thisAmount`) returned as result

Refactoring 2

Rename Variables

Provide more descriptive names

- Avoid name conflicts to assure correctness

Guidelines

- Refactor while reading to gain understanding
- Try to eliminate the need for comments
- Try to eliminate the need for local variables in methods

Refactoring 3

Move Method

- `amountFor` is in `Customer`, but it doesn't use any `Customer` data
 - Move it to `Rental` (3a)
 - Give it a new name
 - Remove references to `aRental` from message invocations
 - Change method invocations to use new name (3b)
- Test after each small change

Refactoring 4

Replace Temp with Query

- Local variables that are only set once in a method act like constants
- Replace them with calls to the defining method
- This assumes the method call has no side effect
- May reduce performance
- Improves understandability
- Makes other refactorings easier

Refactoring 5

Extract FrequentRenterPoints Computation

- **Further simplify** `statement` **computation**
- **New method**
`(getFrequentRenterPoints)` **in**
`Rental`
- **Some algebraic simplification as well**
 - Removes side effects
 - Removes a local variable

Refactoring and Enhancements

- Refactorings leave functionality unchanged
 - This makes testing easier
- But sometimes refactorings can be used to prepare for an enhancement
 - For example, introduce an abstract class that will be subclassed as part of a new feature

Possible Enhancements to the Video Store Program

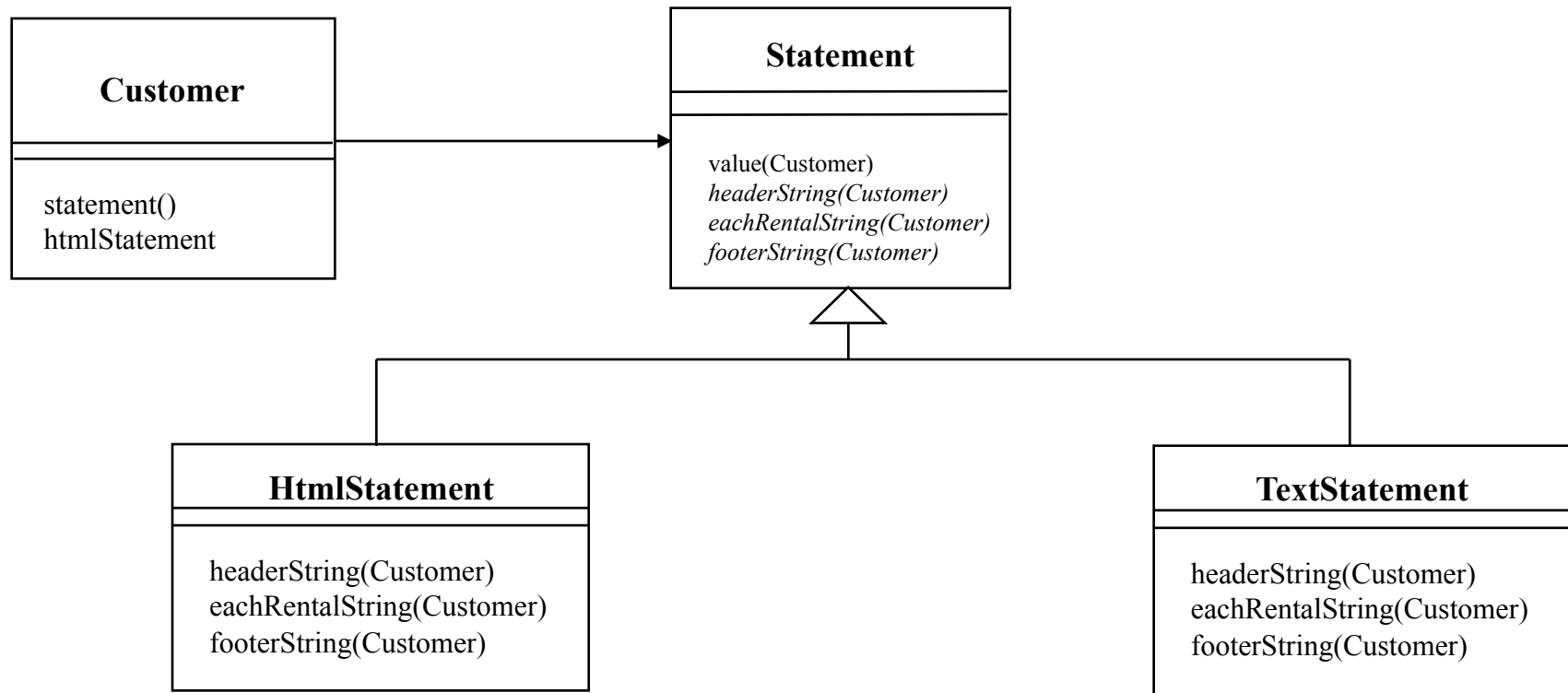
- HTML version of statement
- Altered charging rules
- New classes of movies

Interlude 6

Prepare to add `htmlStatement`

- One way in which this program might be extended is to add a second output format
- `htmlStatement` is a new method for formatting output in HTML instead of text
- It should not be concerned with the rules for computing charges, just with how to format
- Consequently, statements for computing charges should be factored into methods, even if this means duplicating loop code
- These methods will be callable from both `statement` and `htmlStatement`

Class Diagram



Refactoring 7: Steps

- Define a new `Statement` class with `TextStatement` and `HtmlStatement` subclasses
- Rename `statement` method to `value` within the `Statement` classes and pass a `Customer` as argument
- Add `getRentals` method to return an Enumeration of current `Rentals`
- Relax visibility of `getTotalCharge` and `getTotalFrequentRenterPoints`

Refactoring 7: Steps - 2

- Prepare methods for all format-specific functionality and replace existing code with calls to the new methods
- Remaining calling method can be pulled up into `Statement` class
- Declare abstract classes for format-specific methods

Refactoring 8:

Replace Conditional Logic on priceCode
with Polymorphism

- `switch` statements (or `else ifs`) are strong indicators of poor object-oriented coding practices
- They should be replaced by subclasses and polymorphic methods

Refactoring 8: Steps

- First, note that the `switch` statement is discriminating on the type of the movie, so it should really be in the `Movie` class
- Do the same with the `getFrequentRenterPoints` method
- Now the way is prepared for subclassing `Movie` objects into `Regular`, `Childrens`, and `NewRelease` movies

Refactoring 8: Steps - 2

- Unfortunately, this doesn't work. Can you see why?

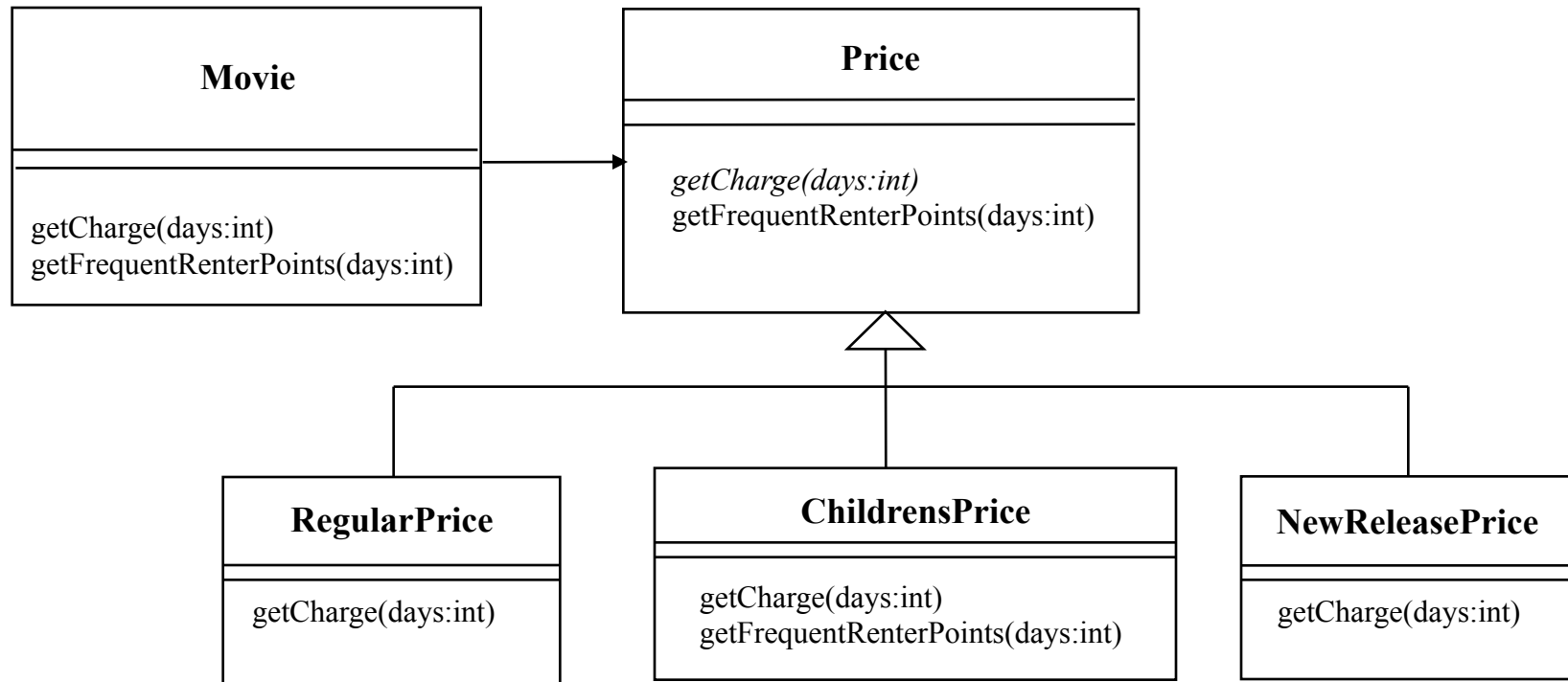
Refactoring 8: Steps - 2

- Unfortunately, this doesn't work. Can you see why?
- Movies in video stores can change their categories dynamically as, for example, when a `New Release` moves into the `Regular` category
- But an object can't change its class dynamically

Refactoring 8: Steps - 3

- There is a trick (called the *state* pattern) for overcoming this problem. It involves introducing a class solely for holding type information
- We will use the `Price` class with subclasses for `ChildrensPrice`, `NewReleasePrice`, and `RegularPrice`
- These classes provide methods for `getCharge` and `getFrequentRenterPoints`

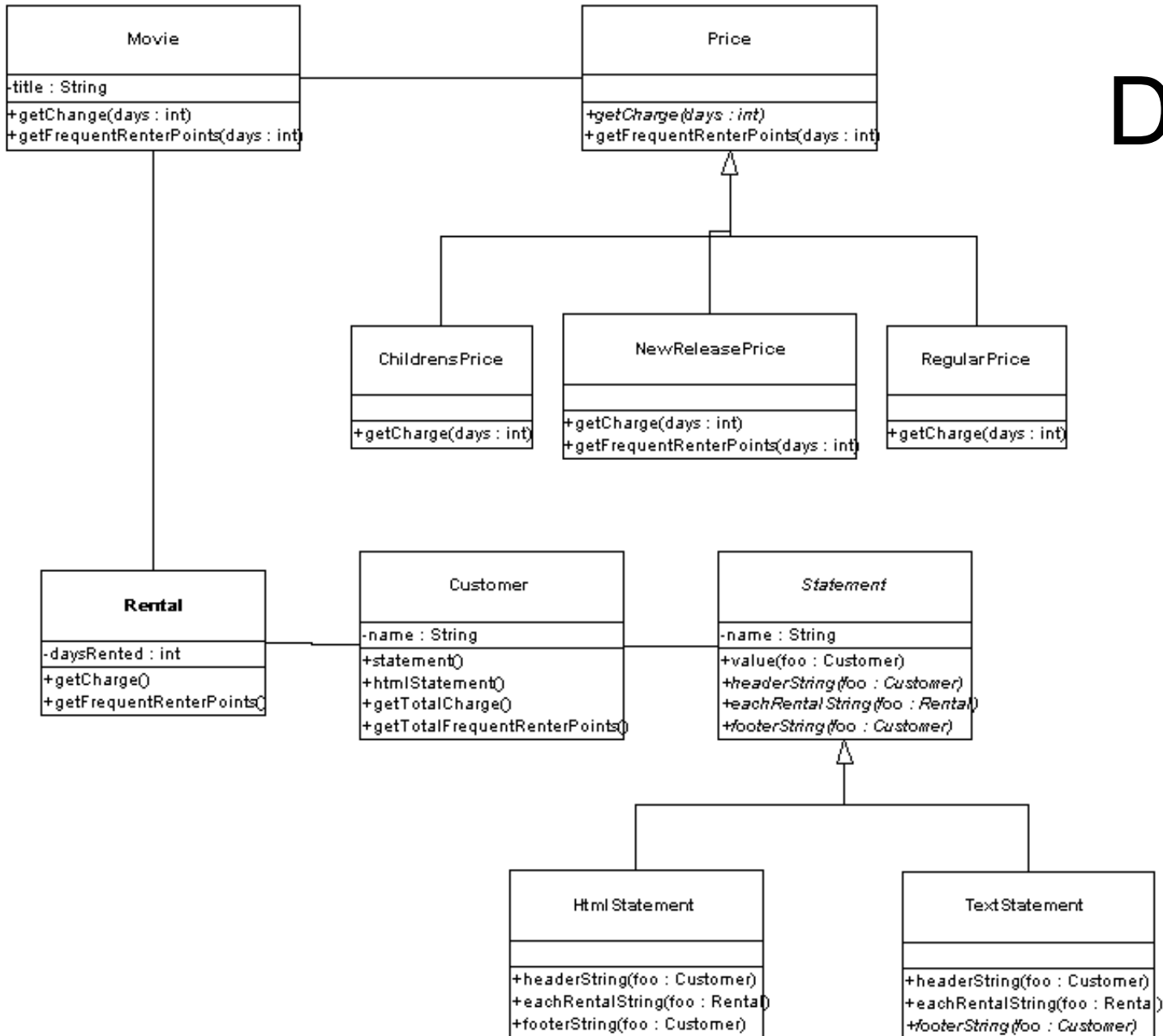
Class Diagram



Refactoring 8: Steps - 4

- Move type code behavior into `Price` class
- Move `switch` statement into `Price` class
- Replace `switch` statement with polymorphism
- Update `Movie` class to reflect changes

Class Diagram



Observations

- Ended up with many small methods
- Largely self documenting
- Classes correspond to concepts
- Changes made incrementally with frequent testing