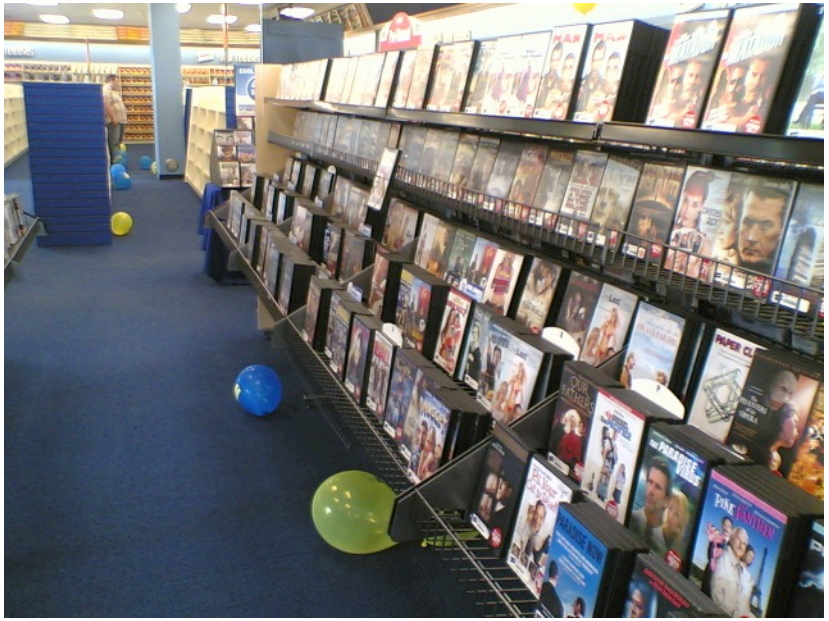


Refactoring – Exercise 1

Maria Grazia Pia

INFN Genova, Italy

Maria.Grazia.Pia@cern.ch

http://www.ge.infn.it/geant4/training/APC2014/

Exercise 1: The Video Store

Grab basic concepts

Get into the habit of refactoring

M. Fowler, Refactoring, Chapter 1

(translated from Java into C++)

Maria Grazia Pia, *INFN Genova*

Setting up a CernCM virtual machine on your laptop

- The exercise is performed on a Scientific Linux 6 platform.
- Install a hypervisor on your laptop
  - Follow the instructions at http://cernvm.cern.ch/portal/hypervisorconfig to configure your hypervisor
- Configure your CernVM virtual machine
  - http://cernvm.cern.ch/portal/cvmconfiguration
  - You may want to share a folder on your laptop devoted to refactoring exercises
- Create a context (i.e. a set of configuration options)
  - http://cernvm.cern.ch/portal/online/documentation/create-new-context
  - We will not use CVMFS, therefore you do not need to configure it
  - Define yourself as a user
  - In Preferences, select the Desktop option (don't forget to tick *Start X on startup*)
- Pair your virtual machine to the context that you defined
  - http://cernvm.cern.ch/portal/online/documentation/pairing-the-instance
- Login on your virtual machine

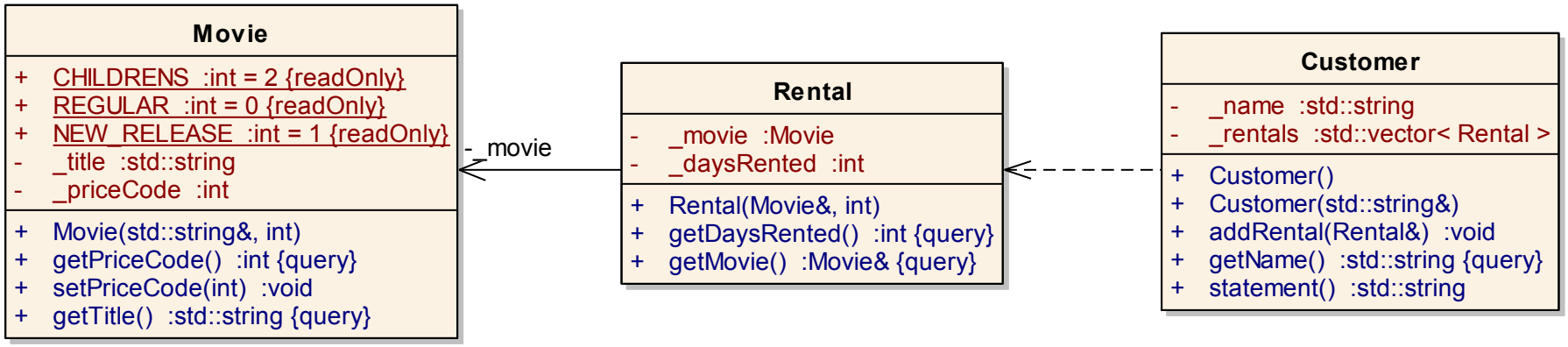
Download the tarball for the exercise

- **Create a directory** on your virtual machine for the exercise:
  mkdir APC2014ex1
  cd APC2014ex1
- **Download** the tarball with the original code ported to C++
  - http://www.ge.infn.it/geant4/training/APC2014/code1/original.tgz
  - wget http://www.ge.infn.it/geant4/training/APC2014/code1/original.tgz
- **Unpack** your tarball
  tar -zxf original.tgz
- Now you have an ***APC2014ex1/original*** directory on your virtual machine, which contains the starting point for the exercise

UML diagrams produced with Enterprise Architect (Sparx Systems)

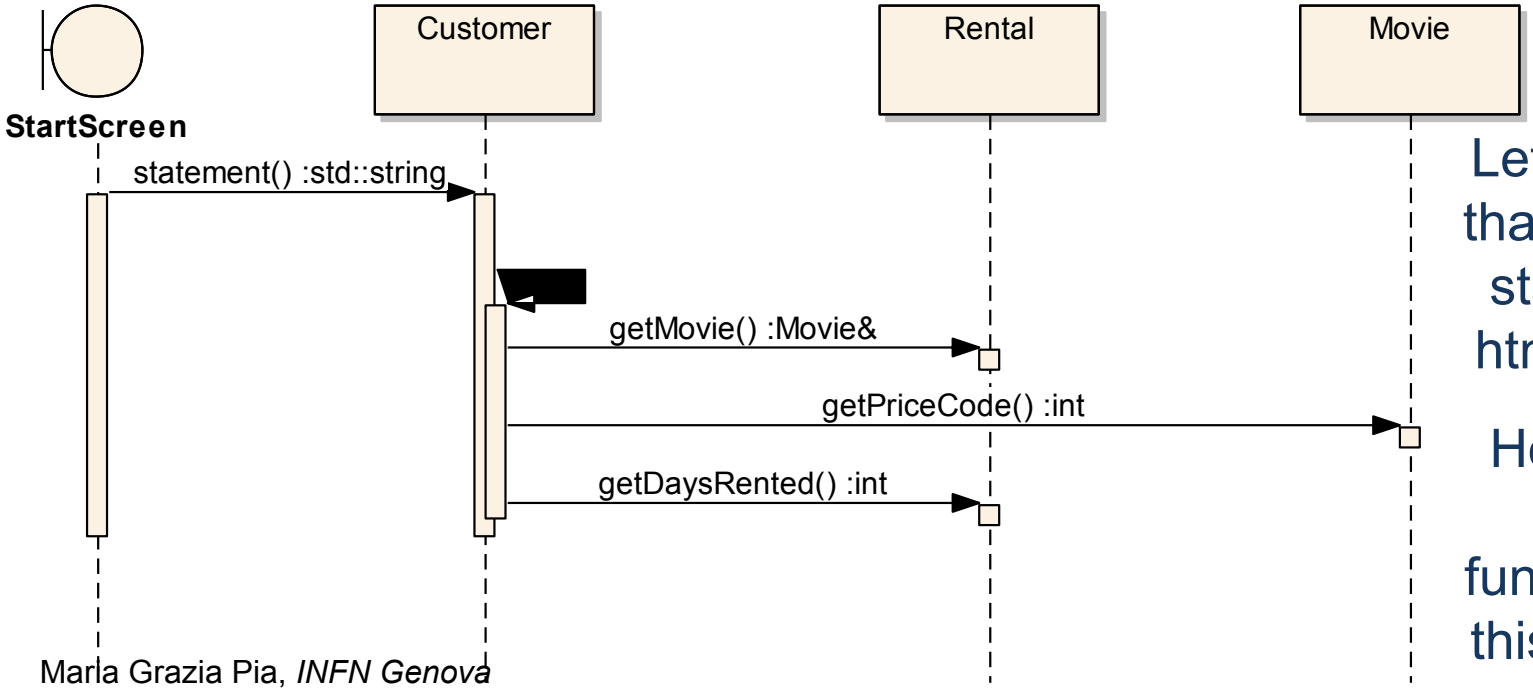
Original code

Example from Refactoring book



M. Fowler, Refactoring - Chapter 1: Refactoring, a First Example

Original code



Let's suppose that we want a statement in html format…

How can we add new functionality to this software?

A first look at the code

- **Not well designed** and certainly **not object oriented**
  - The long routine in the Customer class does far too much.
  - Many of the things that it does should really be done by the other classes
- **Difficult to change**
  - Suppose that they *want a statement printed in HTML*
  - It is impossible to reuse any of the behavior of the current statement method for an HTML statement.
  - One would end up with writing a whole new method that duplicates much of the behavior of statement.
  - But what happens when the charging rules change? You have to fix both statement and htmlStatement and ensure the fixes are consistent.
  - The users want to make changes to the way they classify movies, but they haven't yet decided on the change they are going to make.
  - The statement method is where the changes have to be made to deal with changes in classification and charging rules

```
std::string Customer::statement()
{
  double totalAmount = 0;
  int frequentRenterPoints = 0;
  std::vector< Rental >::iterator iter = _rentals.begin();
  std::vector< Rental >::iterator iter_end = _rentals.end();
  std::ostringstream result;
  result << "Rental Record for " << getName() << "\n";
  for ( ; iter != iter_end; ++iter ) {
    double thisAmount = 0;
    Rental each = *iter;

    // determine amounts for each line
    switch ( each.getMovie().getPriceCode() ) {
      case Movie::REGULAR:
        thisAmount += 2;
        if ( each.getDaysRented() > 2 )
          thisAmount += ( each.getDaysRented() - 2 ) * 1.5 ;
        break;
      case Movie::NEW_RELEASE:
        thisAmount += each.getDaysRented() * 3;
        break;
      case Movie::CHILDRENS:
        thisAmount += 1.5;
        if ( each.getDaysRented() > 3 )
          thisAmount += ( each.getDaysRented() - 3 ) * 1.5;
        break;
    }

    // add frequent renter points
    frequentRenterPoints++;
    // add bonus for a two day new release rental
    if ( ( each.getMovie().getPriceCode() == Movie::NEW_RELEASE )
         && each.getDaysRented() > 1 ) frequentRenterPoints++;

    // show figures for this rental
    result << "\t" << each.getMovie().getTitle() << "\t"
           << thisAmount << "\n";
    totalAmount += thisAmount;
  }
  // add footer lines
  result << "Amount owed is " << totalAmount << "\n";
  result << "You earned " << frequentRenterPoints
         << " frequent renter points";
  return result.str();
}
```

original Customer::statement()

Step 0: Tests

- One needs a solid set of **tests** for that section of code.
  - Risk of introducing bugs while modifying the code
- Tests must be **self-checking**
  - They either say "OK" (e.g. all the strings are identical to the reference strings)
  - or they print a list of failures (in this exercise: lines that turned out differently)
- Write a test to support your future refactoring
- Hint in http://www.ge.infn.it/geant4/training/APC2014/code1/original/fowler.cc
  - you may want to extend it, adding a few more test cases, reports on failures etc.
- Build the test fowler.cc
  - (if your test has another file name, modify the GNUmakefile accordingly):

```
cd original
gmake
```

- Run the test:

`./fowler` (it should end with "OK")

Refactoring in 15 steps

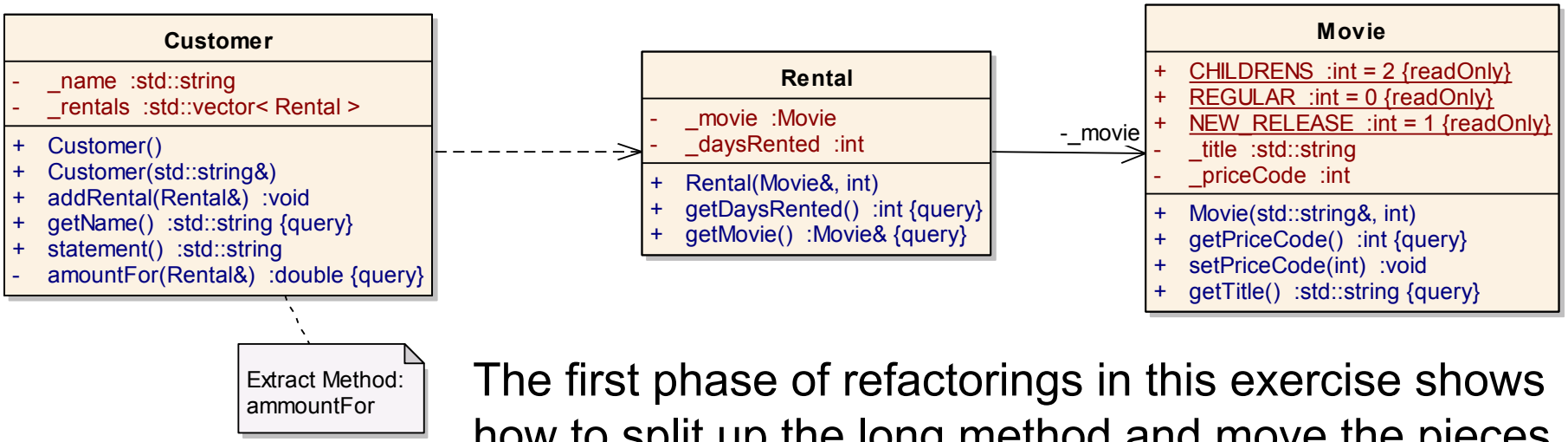
- **Try to do the suggested refactoring yourself**
- More extensive explanations in
  http://www.ge.infn.it/geant4/training/APC2014/exercise1.html
- At the end of each step, or if you encounter difficulties, you may want to have a look at the solutions
  - For each step N, the solution is in
    http://www.ge.infn.it/geant4/training/APC2014/code1/stepN/

Step 1: Extract Method

Bad smell:
the long metod in Customer
Decompose it in small pieces

M. Fowler, Refactoring - Chapter 1: Refactoring, a First Example

Step 1: Extract Method



The first phase of refactorings in this exercise shows how to split up the long method and move the pieces to better classes

Find a logical clump of code and use **Extract Method**
Candidate: **switch** statement to extract into its own method

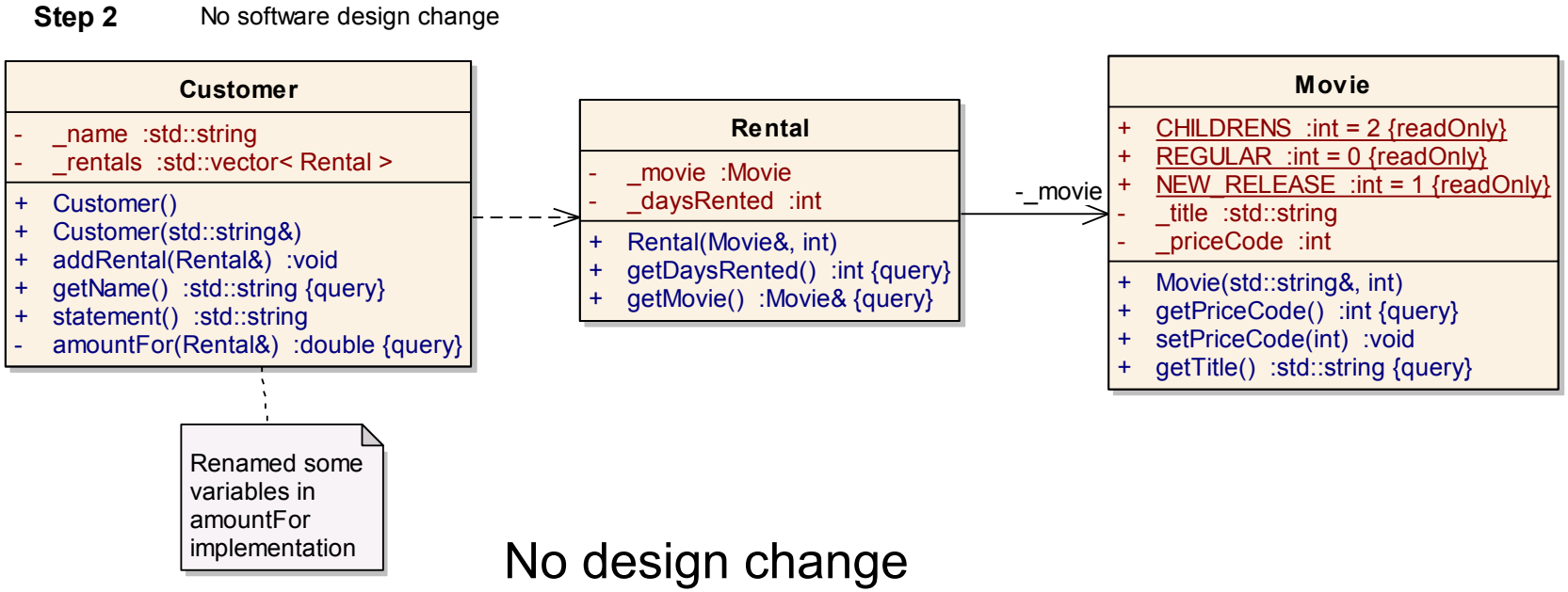
Solution http://www.ge.infn.it/geant4/training/APC2014/code1/step2/

```
std::string Customer::statement()
{
  double totalAmount = 0;
  int frequentRenterPoints = 0;
  std::vector< Rental >::iterator iter = _rentals.begin();
  std::vector< Rental >::iterator iter_end = _rentals.end();
  std::ostringstream result;
  result << "Rental Record for " << getName() << "\n";
  for ( ; iter != iter_end; ++iter ) {
    double thisAmount = 0;
    Rental each = *iter;

    thisAmount = amountFor( each );

    // add frequent renter points
    frequentRenterPoints++;
    // add bonus for a two day new release rental
    if ( ( each.getMovie().getPriceCode() == Movie::NEW_RELEASE )
         && each.getDaysRented() > 1 ) frequentRenterPoints++;

    // show figures for this rental
    result << "\t" << each.getMovie().getTitle() << "\t"
           << thisAmount << "\n";
    totalAmount += thisAmount;
  }
  // add footer lines
  result << "Amount owed is " << totalAmount << "\n";
  result << "You earned " << frequentRenterPoints
         << " frequent renter points";
  return result.str();
}

double Customer::amountFor( const Rental& each ) const
{
  double thisAmount = 0;
  switch ( each.getMovie().getPriceCode() ) {
    case Movie::REGULAR:
      thisAmount += 2;
      if ( each.getDaysRented() > 2 )
        thisAmount += ( each.getDaysRented() - 2 ) * 1.5;
      break;
    case Movie::NEW_RELEASE:
      thisAmount += each.getDaysRented() * 3;
      break;
    case Movie::CHILDRENS:
      thisAmount += 1.5;
      if ( each.getDaysRented() > 3 )
        thisAmount += ( each.getDaysRented() - 3 ) * 1.5;
      break;
  }
  return thisAmount;
}
```

Customer::statement()
after step 1

Extracted method amountFor

Step 2: Renaming Variables



No design change

Some of the variable names in amountFor could be better renamed

Is renaming worth the effort? Absolutely. Good code should communicate what it is doing clearly, and variable names are a key to clear code.

Solution:

http://www.ge.infn.it/geant4/training/APC2014/code1/step2/Customer.cc

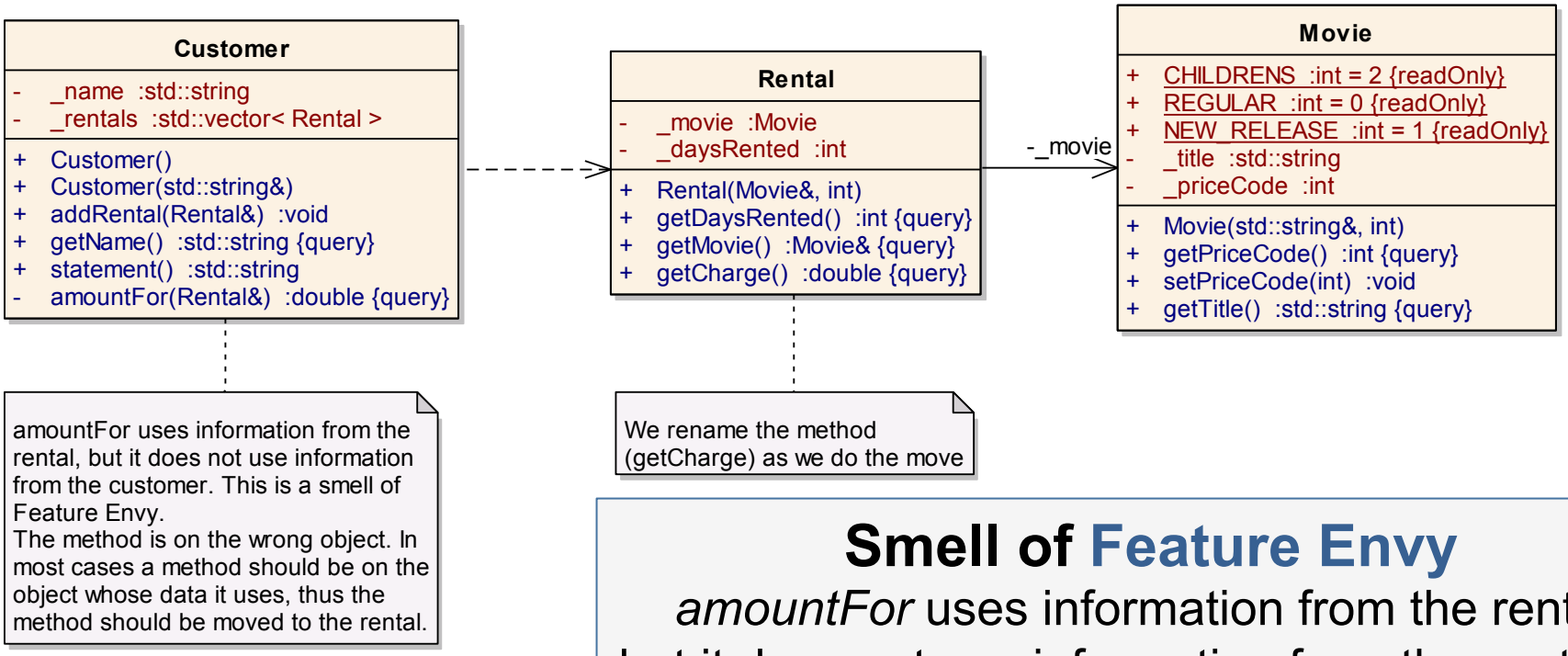
```
double Customer::amountFor( const Rental& aRental ) const
{
  double result = 0;
  switch ( aRental.getMovie().getPriceCode() ) {
    case Movie::REGULAR:
      result += 2;
      if ( aRental.getDaysRented() > 2 )
        result += ( aRental.getDaysRented() - 2 ) * 1.5;
      break;
    case Movie::NEW_RELEASE:
      result += aRental.getDaysRented() * 3;
      break;
    case Movie::CHILDRENS:
      result += 1.5;
      if ( aRental.getDaysRented() > 3 )
        result += ( aRental.getDaysRented() - 3 ) * 1.5;
      break;
  }
  return result;
}
```

Customer::statement()

Step 2: renamed variables

Step 3: Move Method

Step 3 - Move Method



Smell of Feature Envy

amountFor uses information from the rental, but it does not use information from the customer

In most cases a method should be on the object whose data it uses: the method should be moved to the rental

- Use **Move Method**
- Rename the method (*getCharge*) as we do the move
- Replace the body of *Customer::amountFor* to delegate to the new method

```
// Rental.cc
#include "Rental.hh"

double Rental::getCharge() const
{
  double result = 0;
  switch ( getMovie().getPriceCode() ) {
  case Movie::REGULAR:
    result += 2;
    if ( getDaysRented() > 2 )
      result += ( getDaysRented() - 2 ) * 1.5 ;
    break;
  case Movie::NEW_RELEASE:
    result += getDaysRented() * 3;
    break;
  case Movie::CHILDRENS:
    result += 1.5;
    if ( getDaysRented() > 3 )
      result += ( getDaysRented() - 3 ) * 1.5;
    break;
  }
  return result;
}
```

Step 3

The functionality of amountFor is moved to Rental and renamed getCharge

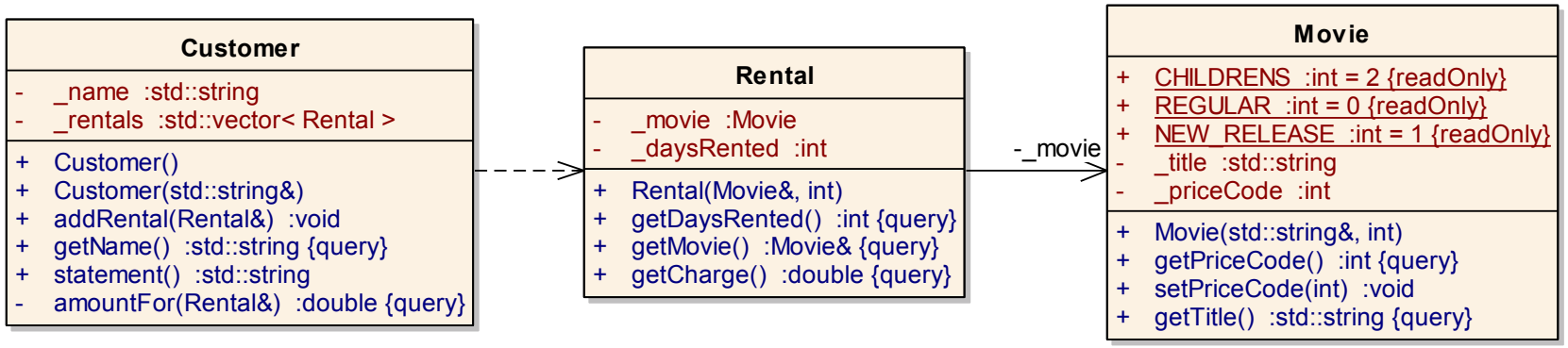
amountFor now delegates to Rental::getCharge

```
double Customer::amountFor( const Rental& aRental ) const
{
  return aRental.getCharge();
}
```

Step 4: Replace Temp with Query

Step 4 - Replace Temp with Query

No change to the software design



thisAmount is now redundant.

It is set to the result of *each.getCharge* and not changed afterward

Thus we can eliminate *thisAmount* by using

Replace Temp with Query

```
std::string Customer::statement()
{
  double totalAmount = 0;
  int frequentRenterPoints = 0;
  std::vector< Rental >::iterator iter = _rentals.begin();
  std::vector< Rental >::iterator iter_end = _rentals.end();
  std::ostringstream result;
  result << "Rental Record for " << getName() << "\n";
  for ( ; iter != iter_end; ++iter ) {

    Rental each = *iter;

    // add frequent renter points
    frequentRenterPoints++;
    // add bonus for a two day new release rental
    if ( ( each.getMovie().getPriceCode() == Movie::NEW_RELEASE )
         && each.getDaysRented() > 1 ) frequentRenterPoints++;

    // show figures for this rental
    result << "\t" << each.getMovie().getTitle() << "\t"
           << each.getCharge() << "\n";
    totalAmount += each.getCharge();
  }
  // add footer lines
  result << "Amount owed is " << totalAmount << "\n";
  result << "You earned " << frequentRenterPoints
         << " frequent renter points";
  return result.str();
}
```

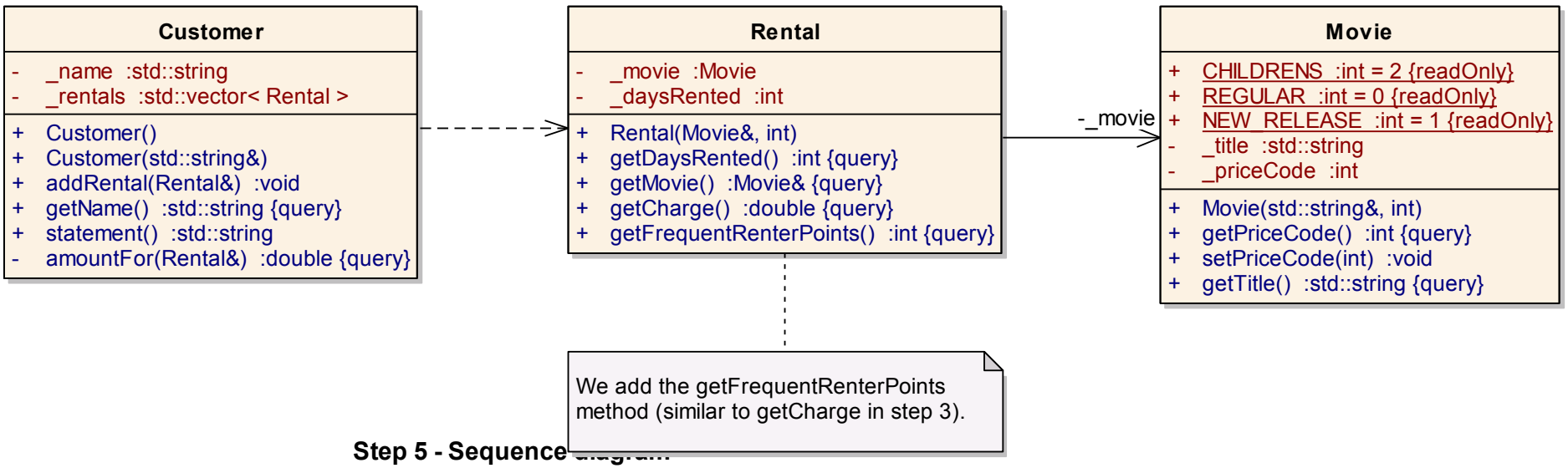
Step 4

thisAmount is replaced by calls to getCharge()

2 calls!

Step 5: Extracting Frequent Renter Points

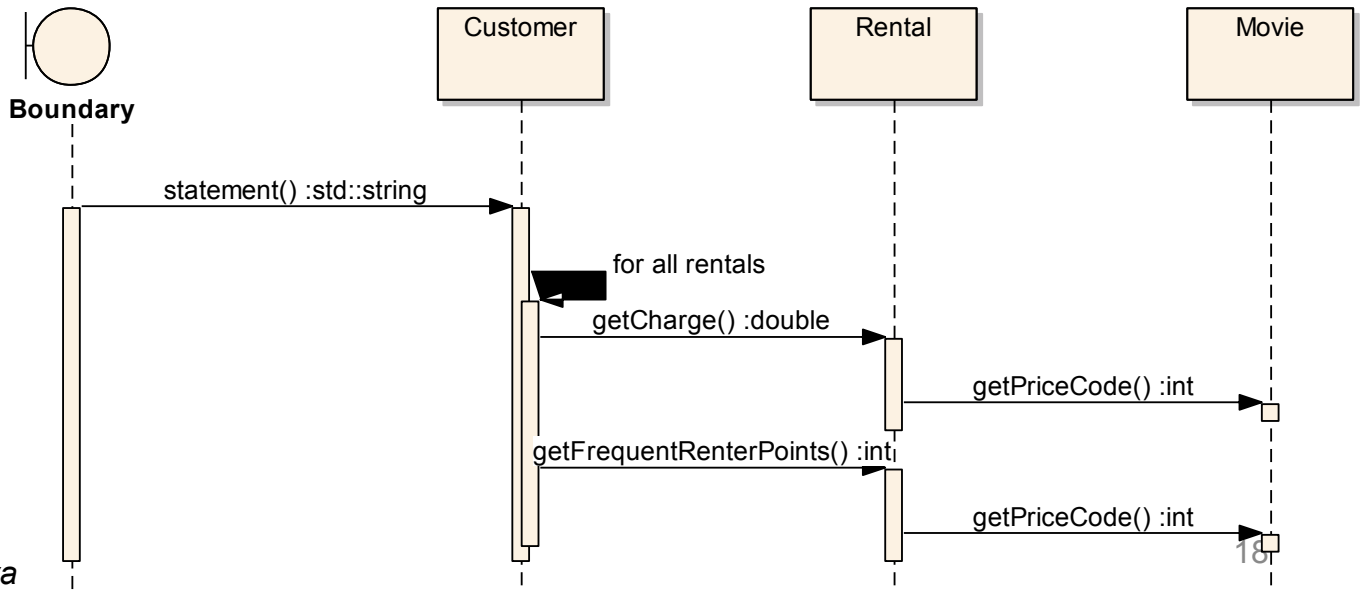
Step 5 - Extracting Frequent Renter Points



Step 5 - Sequence diagram

We do a similar thing for the frequent renter points.

Extract Method on the frequent renter points part of the code



```
std::string Customer::statement()
{
  double totalAmount = 0;
  int frequentRenterPoints = 0;
  std::vector< Rental >::iterator iter = _rentals.begin();
  std::vector< Rental >::iterator iter_end = _rentals.end();
  std::ostringstream result;
  result << "Rental Record for " << getName() << "\n";
  for ( ; iter != iter_end; ++iter ) {

    Rental each = *iter;

    frequentRenterPoints += each.getFrequentRenterPoints();

    // show figures for this rental
    result << "\t" << each.getMovie().getTitle() << "\t"
           << each.getCharge() << "\n";
    totalAmount += each.getCharge();
  }
  // add footer lines
  result << "Amount owed is " << totalAmount << "\n";
  result << "You earned " << frequentRenterPoints
         << " frequent renter points";
  return result.str();
}
```

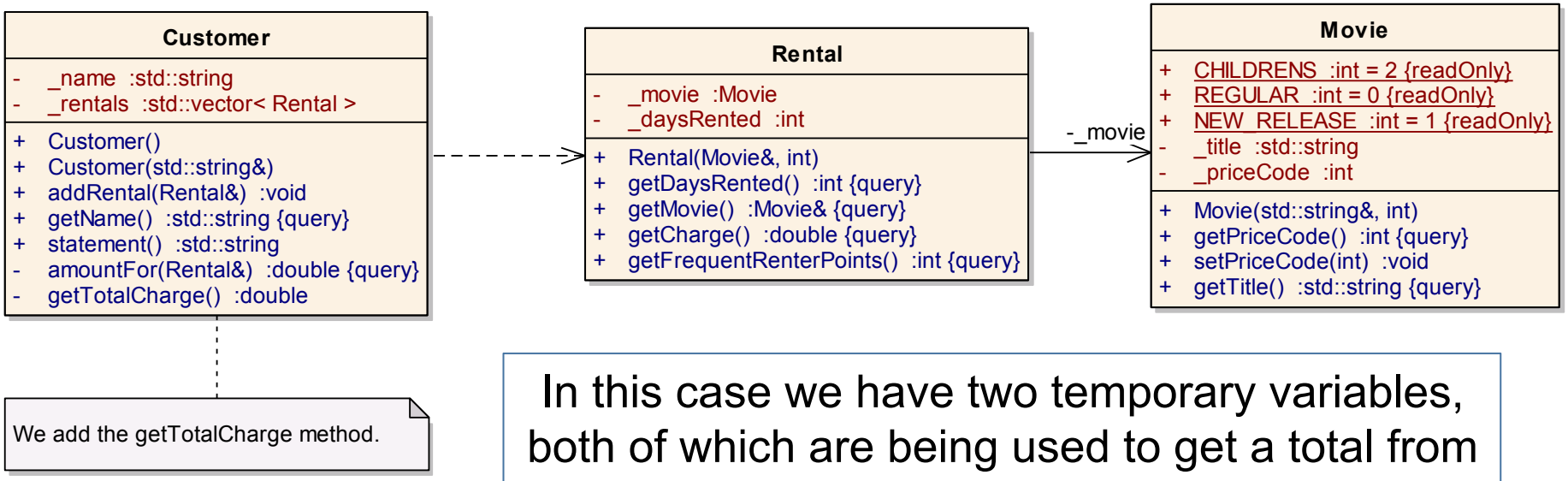
```
int Rental::getFrequentRenterPoints() const
{
  if ( ( getMovie().getPriceCode() == Movie::NEW_RELEASE )
       && getDaysRented() > 1 )
    return 2;
  else
    return 1;
}
```

Step 5

Extract Method with frequent renter points

Step 6: Removing Temps

Step 6 - Removing Temps



In this case we have two temporary variables, both of which are being used to get a total from the rentals attached to the customer. Both the ASCII and HTML versions require these totals.

Use **Replace Temp with Query** to replace *totalAmount* and *frequentRenterPoints* with query methods

Queries are accessible to any method in the class, thus encourage a cleaner design

We begin by replacing *totalAmount* with a *charge* method on *customer*

```
std::string Customer::statement()
{
  double totalAmount = 0;
  int frequentRenterPoints = 0;
  std::vector< Rental >::iterator iter = _rentals.begin();
  std::vector< Rental >::iterator iter_end = _rentals.end();
  std::ostringstream result;
  result << "Rental Record for " << getName() << "\n";
  for ( ; iter != iter_end; ++iter ) {

    Rental each = *iter;

    // add frequent renter points
    frequentRenterPoints += each.getFrequentRenterPoints();

    // show figures for this rental
    result << "\t" << each.getMovie().getTitle() << "\t"
           << each.getCharge() << "\n";

  }
  // add footer lines
  result << "Amount owed is " << getTotalCharge() << "\n";
  result << "You earned " << frequentRenterPoints
         << " frequent renter points";
  return result.str();
}
```

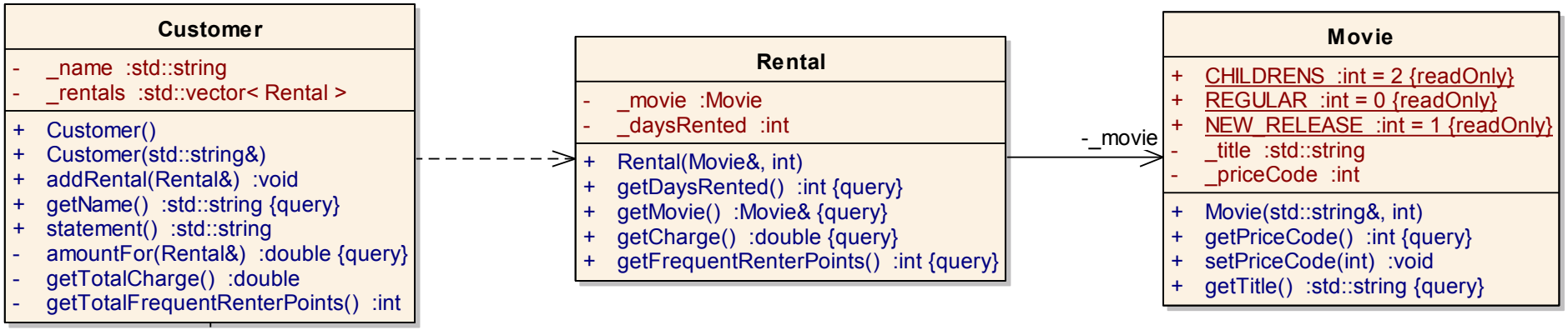
```
double Customer::getTotalCharge()
{
  double result = 0;
  std::vector< Rental >::iterator iter = _rentals.begin();
  std::vector< Rental >::iterator iter_end = _rentals.end();
  for ( ; iter != iter_end; ++iter ) {
    Rental each = *iter;
    result += each.getCharge();
  }
  return result;
}
```

Step 6

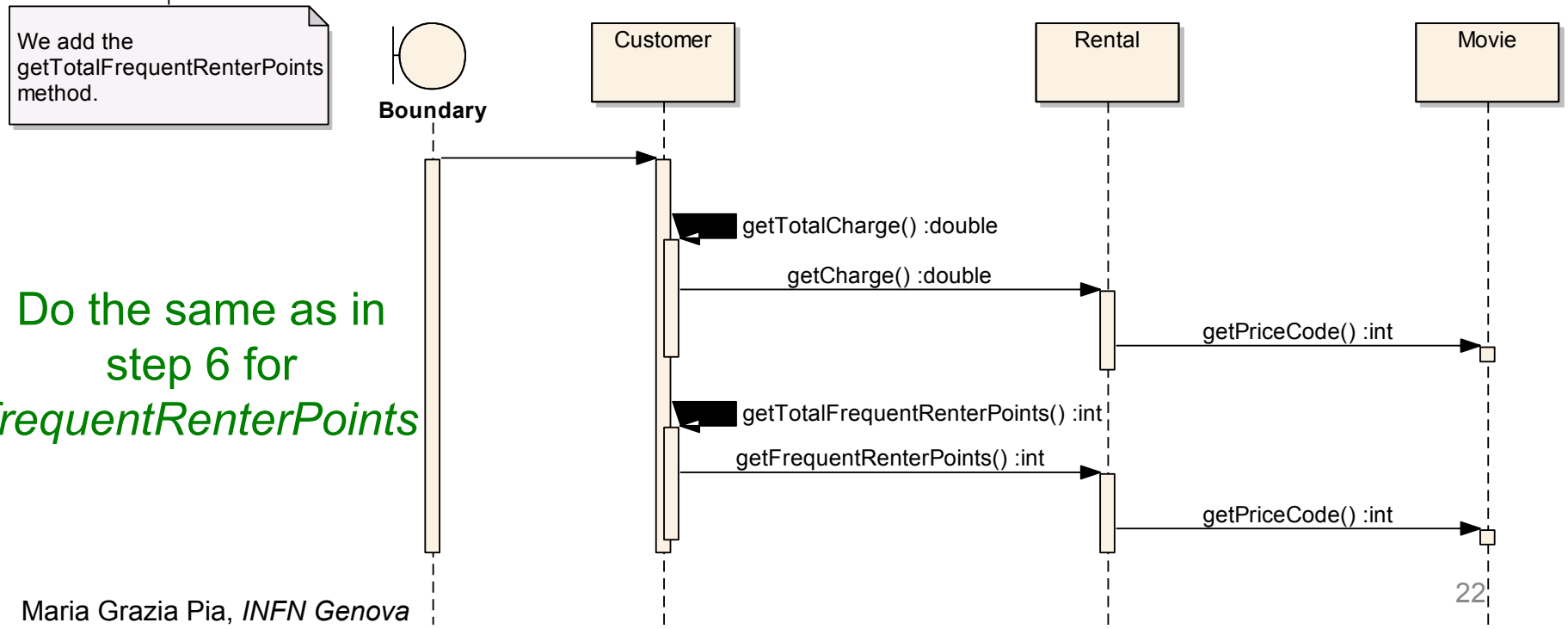
replaced
totalAmount
with query method

Step 7: Still about removing temps

Step 7 - Still about removing temps



Step 7 - Sequence diagram after extraction of totals



Do the same as in step 6 for *frequentRenterPoints*

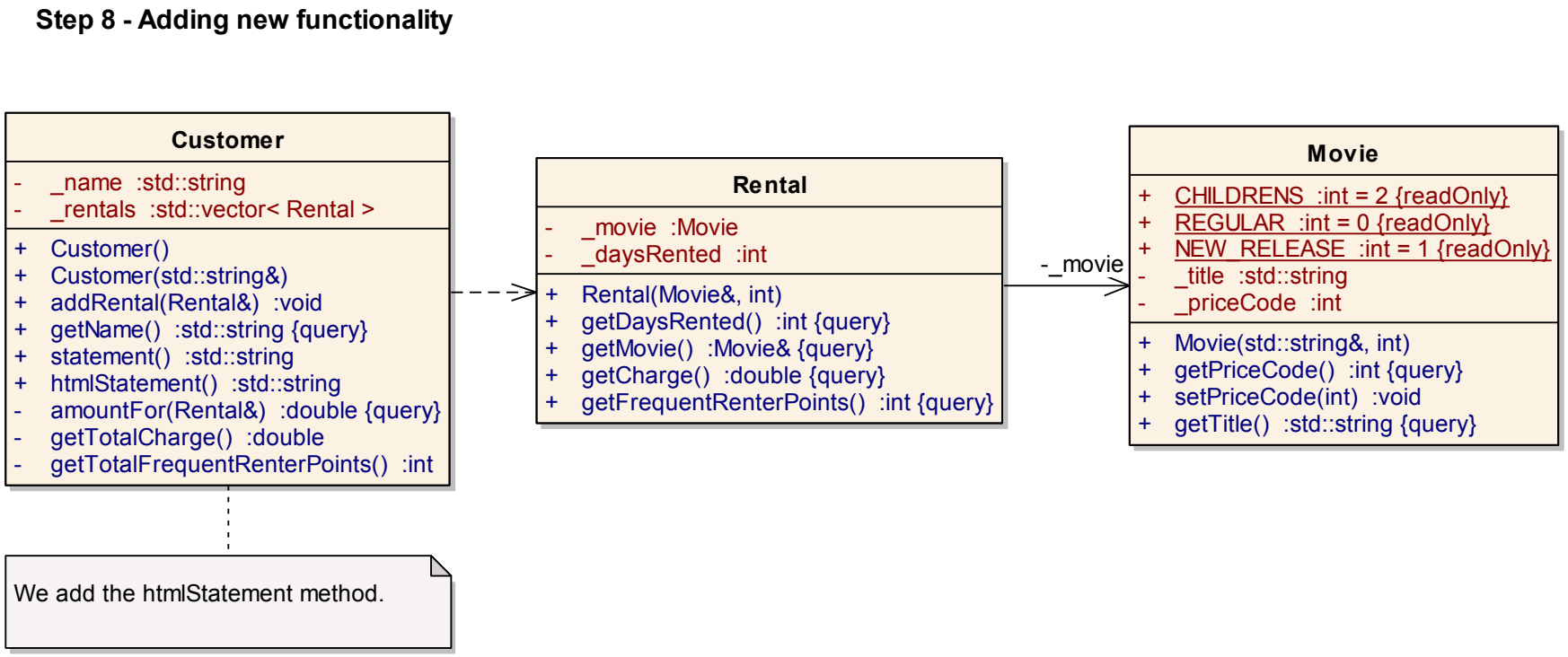
```
std::string Customer::statement()
{

  std::vector< Rental >::iterator iter = _rentals.begin();
  std::vector< Rental >::iterator iter_end = _rentals.end();
  std::ostringstream result;
  result << "Rental Record for " << getName() << "\n";
  for ( ; iter != iter_end; ++iter ) {

    Rental each = *iter;

    // show figures for this rental
    result << "\t" << each.getMovie().getTitle() << "\t"
           << each.getCharge() << "\n";
  }

  // add footer lines
  result << "Amount owed is " << getTotalCharge() << "\n";
  result << "You earned " << getTotalFrequentRenterPoints()
         << " frequent renter points";
  return result.str();
}

double Customer::amountFor( const Rental& aRental ) const
{
  return aRental.getCharge();
}
```

```
int Customer::getTotalFrequentRenterPoints()
{
  int result = 0;
  std::vector< Rental >::iterator iter = _rentals.begin();
  std::vector< Rental >::iterator iter_end = _rentals.end();
  for ( ; iter != iter_end; ++iter ) {
    Rental each = *iter;
    result += each.getFrequentRenterPoints();
  }
  return result;
}
```

Step 7

Query Method
also to replace
frequentRenterPoints

Step 8: Adding new functionality



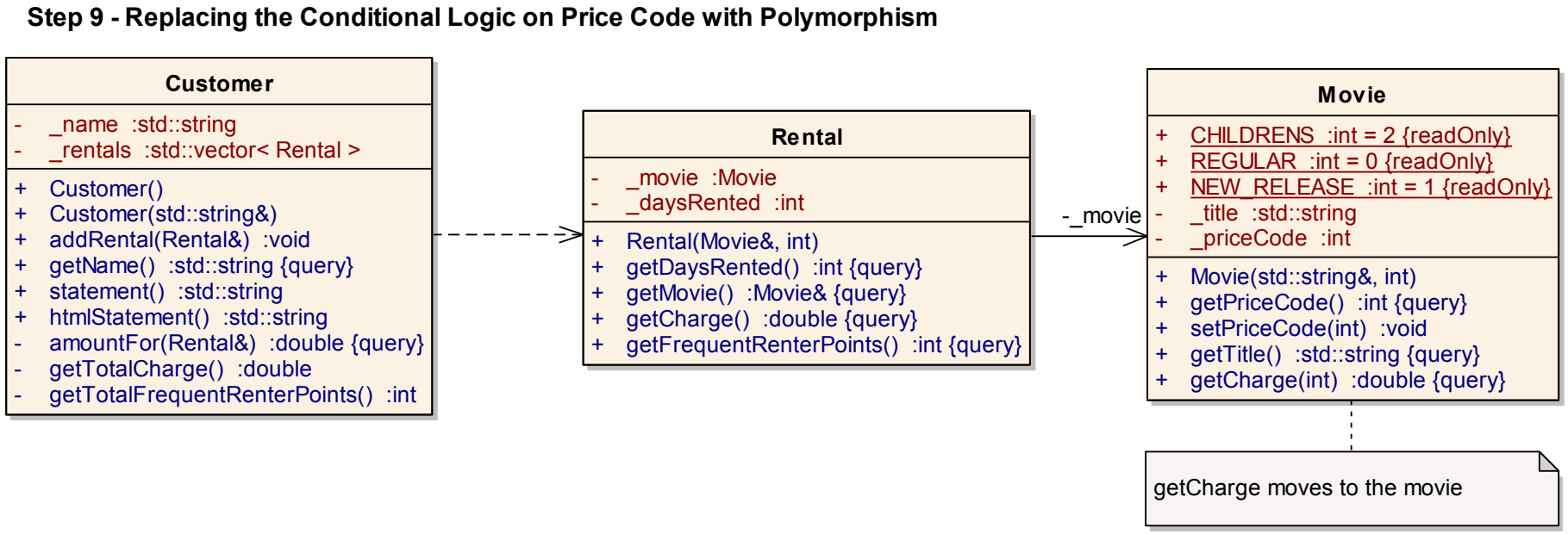
Add htmlStatement() to Customer

Step 8: added htmlStatement()

```
std::string Customer::htmlStatement()
{
  std:: vector< Rental >::iterator iter = _rentals.begin();
  std::vector< Rental >::iterator iter_end = _rentals.end();
  std::ostringstream result;
  result << "<H1>Rentals for <EM>" << getName() << "</EM></H1><P>\n";
  for ( ; iter != iter_end; ++iter ) {
    Rental each = *iter;
    // show figures for each rental
    result << each.getMovie().getTitle() << ": "
           << each.getCharge() << "<BR>\n";
  }
  // add footer lines
  result << "<P>You owe <EM>" << getTotalCharge() << "</EM><P>\n";
  result << "On this rental you earned <EM>"
         << getTotalFrequentRenterPoints()
         << "</EM> frequent renter points<P>";
  return result.str();
}
```

htmlStatement uses the methods we created in the previous steps

Step 9: Replacing the Conditional Logic on Price Code with Polymorphism



The first part of this problem is that switch statement.
It is a bad idea to do a switch based on an attribute of another object.
If you must use a switch statement, it should be on your own data, not on someone else's

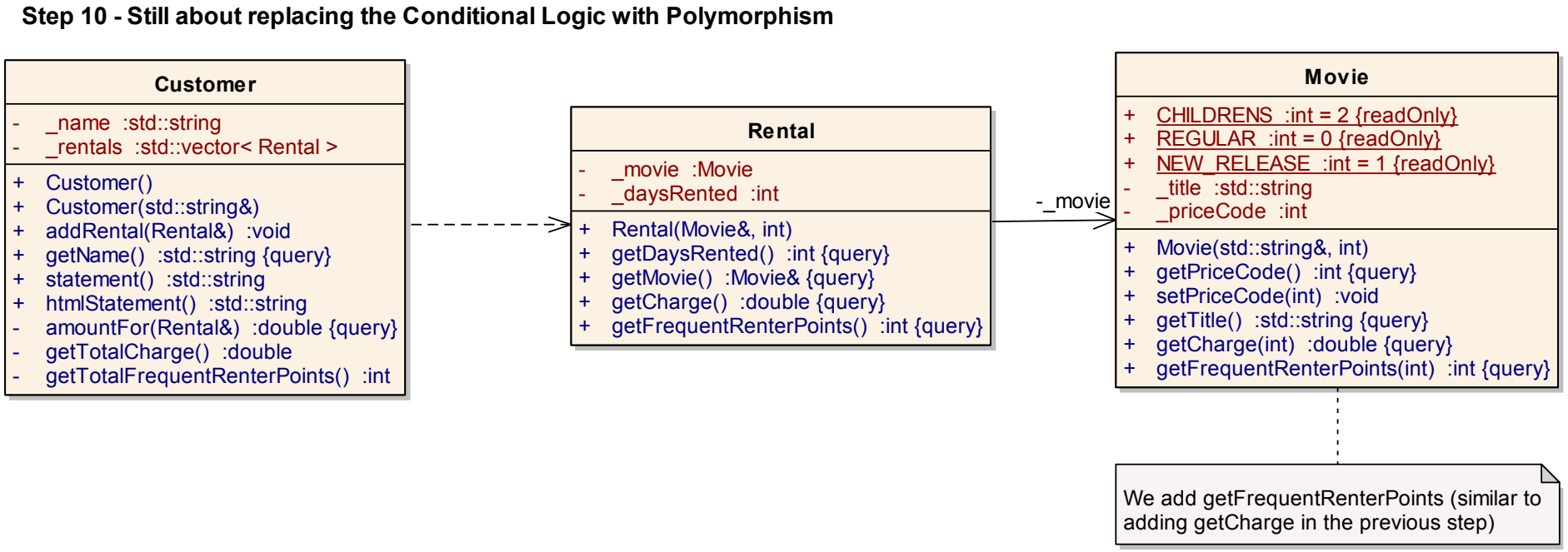
This implies that getCharge should move onto movie

```cpp
// Movie.cpp
#include "Movie.hh"

const int Movie::CHILDRENS;
const int Movie::REGULAR;
const int Movie::NEW_RELEASE;

double Movie::getCharge( int daysRented ) const
{
  double result = 0;
  switch ( getPriceCode() ) {
  case Movie::REGULAR:
    result += 2;
    if ( daysRented > 2 )
      result += ( daysRented - 2 ) * 1.5 ;
    break;
  case Movie::NEW_RELEASE:
    result += daysRented * 3;
    break;
  case Movie::CHILDRENS:
    result += 1.5;
    if ( daysRented > 3 )
      result += ( daysRented - 3 ) * 1.5;
    break;
  }
  return result;
}
```

Step 9

getCharge is moved to Movie

10: Still about replacing the Conditional Logic with Polymorphism



Do the same as in step 9
with the frequent renter point calculation

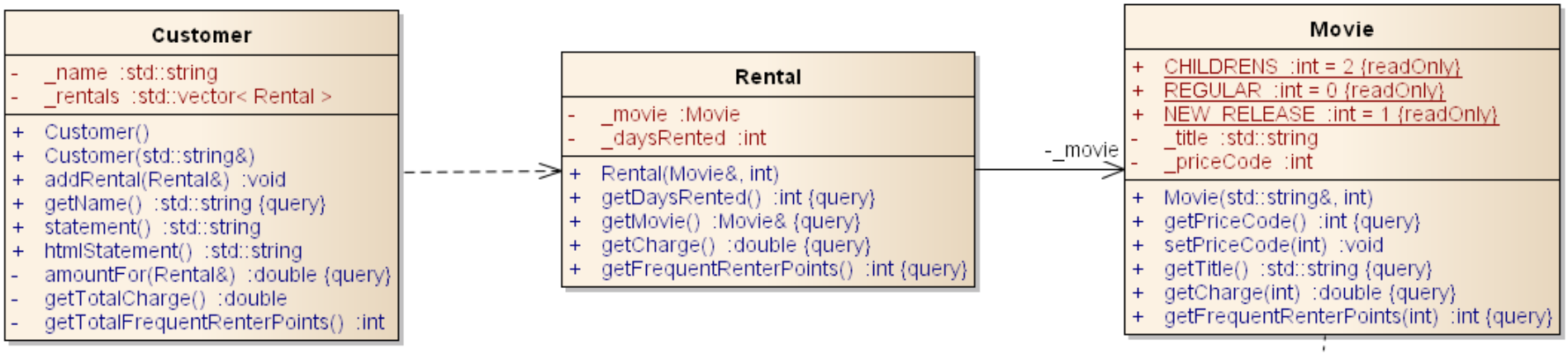
```cpp
// Movie.cpp
#include "Movie.hh"

const int Movie::CHILDRENS;
const int Movie::REGULAR;
const int Movie::NEW_RELEASE;

double Movie::getCharge( int daysRented ) const
{
  double result = 0;
  switch ( getPriceCode() ) {
  case Movie::REGULAR:
    result += 2;
    if ( daysRented > 2 )
      result += ( daysRented - 2 ) * 1.5 ;
    break;
  case Movie::NEW_RELEASE:
    result += daysRented * 3;
    break;
  case Movie::CHILDRENS:
    result += 1.5;
    if ( daysRented > 3 )
      result += ( daysRented - 3 ) * 1.5;
    break;
  }
  return result;
}


int Movie::getFrequentRenterPoints( int daysRented ) const
{
  if ( ( getPriceCode() == Movie::NEW_RELEASE ) && daysRented > 1 )
    return 2;
  else
    return 1;
}
```

Step 10

getFrequentRenterPoints is moved to Movie

Step 11: At last... Inheritance

Step 11 - Self Encapsulate Field



Replace the *switch* statement by using **polymorphism**

We have several types of movie that have different ways of answering the same question

This sounds like a job for **subclasses**

Self Encapsulate Field: all uses of the type code go through getting and setting methods

```
Movie::Movie( const std::string& title, int priceCode ):
_title( title )
{
setPriceCode( priceCode );
}
```

A movie can change its classification during its lifetime.
An object cannot change its class during its lifetime.
Solution: **use the State pattern**

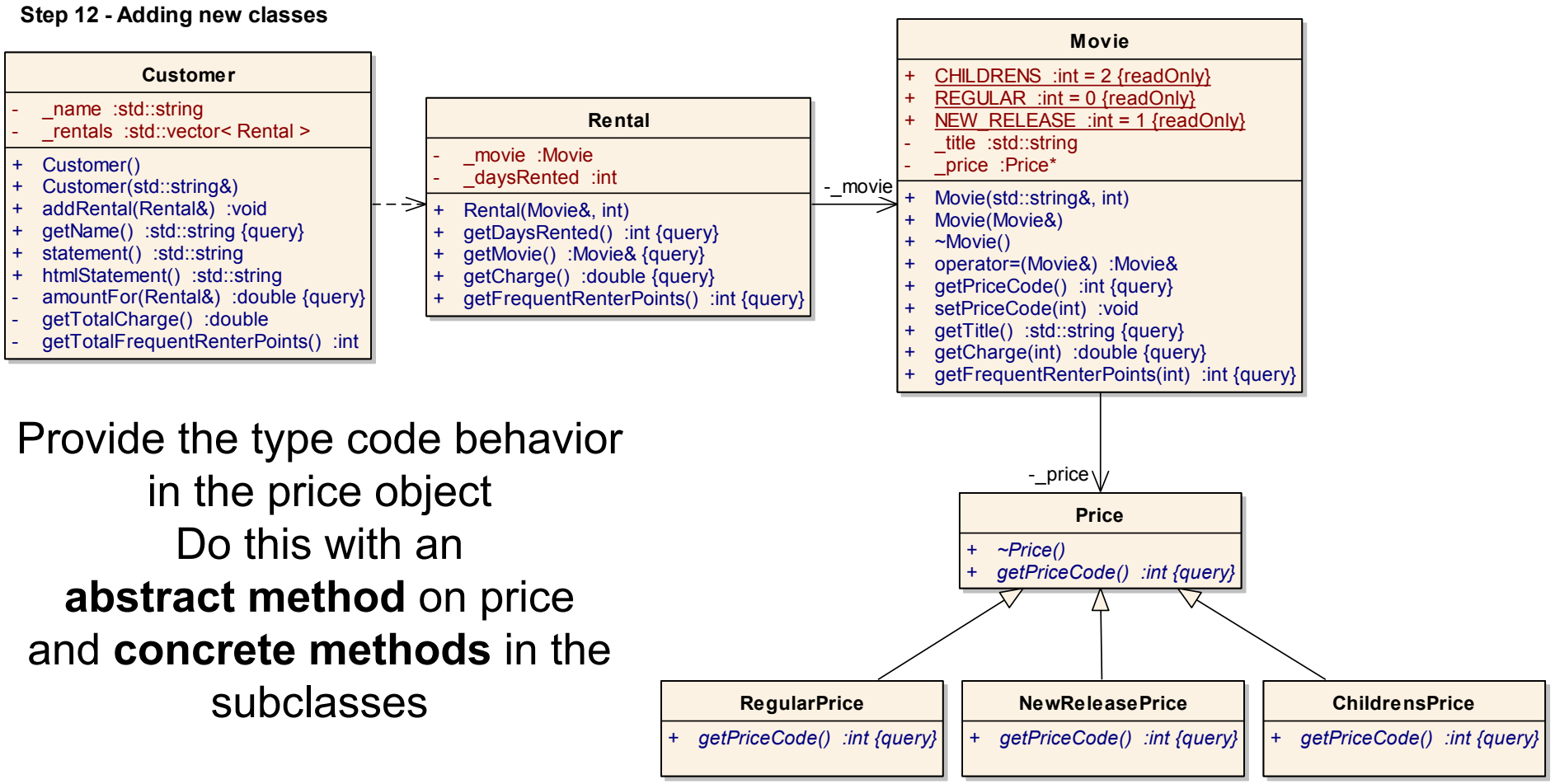
1. **Replace Type Code with State/Strategy**: **Self Encapsulate Field** on the type code
2. Move Method to move the switch statement into the price class
3. Replace Conditional with Polymorphism to eliminate the switch statement

```cpp
// Movie.cpp
#include "Movie.hh"

const int Movie::CHILDRENS;
const int Movie::REGULAR;
const int Movie::NEW_RELEASE;

Movie::Movie( const std::string& title, int priceCode )
  : _title( title )
{
  setPriceCode( priceCode );
}

double Movie::getCharge( int daysRented ) const
{
  double result = 0;
  switch ( getPriceCode() ) {
  case Movie::REGULAR:
    result += 2;
    if ( daysRented > 2 )
      result += ( daysRented - 2 ) * 1.5 ;
    break;
  case Movie::NEW_RELEASE:
    result += daysRented * 3;
    break;
  case Movie::CHILDRENS:
    result += 1.5;
    if ( daysRented > 3 )
      result += ( daysRented - 3 ) * 1.5;
    break;
  }
  return result;
}


int Movie::getFrequentRenterPoints( int daysRented ) const
{
  if ( ( getPriceCode() == Movie::NEW_RELEASE ) && daysRented > 1 )
    return 2;
  else
    return 1;
}
```

Step 11

The price code is encapsulated: only handled through the setPriceCode member function

Step 12: Adding new classes



Provide the type code behavior in the price object
Do this with an **abstract method** on price and **concrete methods** in the subclasses

Change the movie's accessor for the price code to use the new class

Add the new classes

```cpp
// ChildrensPrice.cc
#include "ChildrensPrice.hh"
#include "Movie.hh"

int ChildrensPrice::
getPriceCode() const
{
  return Movie::CHILDRENS;
}
```

```cpp
// NewReleasePrice.cc
#include "NewReleasePrice.hh"
#include "Movie.hh"

int NewReleasePrice::
getPriceCode() const
{
  return Movie::NEW_RELEASE;
}
```

```cpp
// RegularPrice.cc
#include "RegularPrice.hh"
#include "Movie.hh"

int RegularPrice::
getPriceCode() const
{
  return Movie::REGULAR;
}
```

Step 12

The three classes have concrete methods to calculate the price code, the base class has a pure virtual function

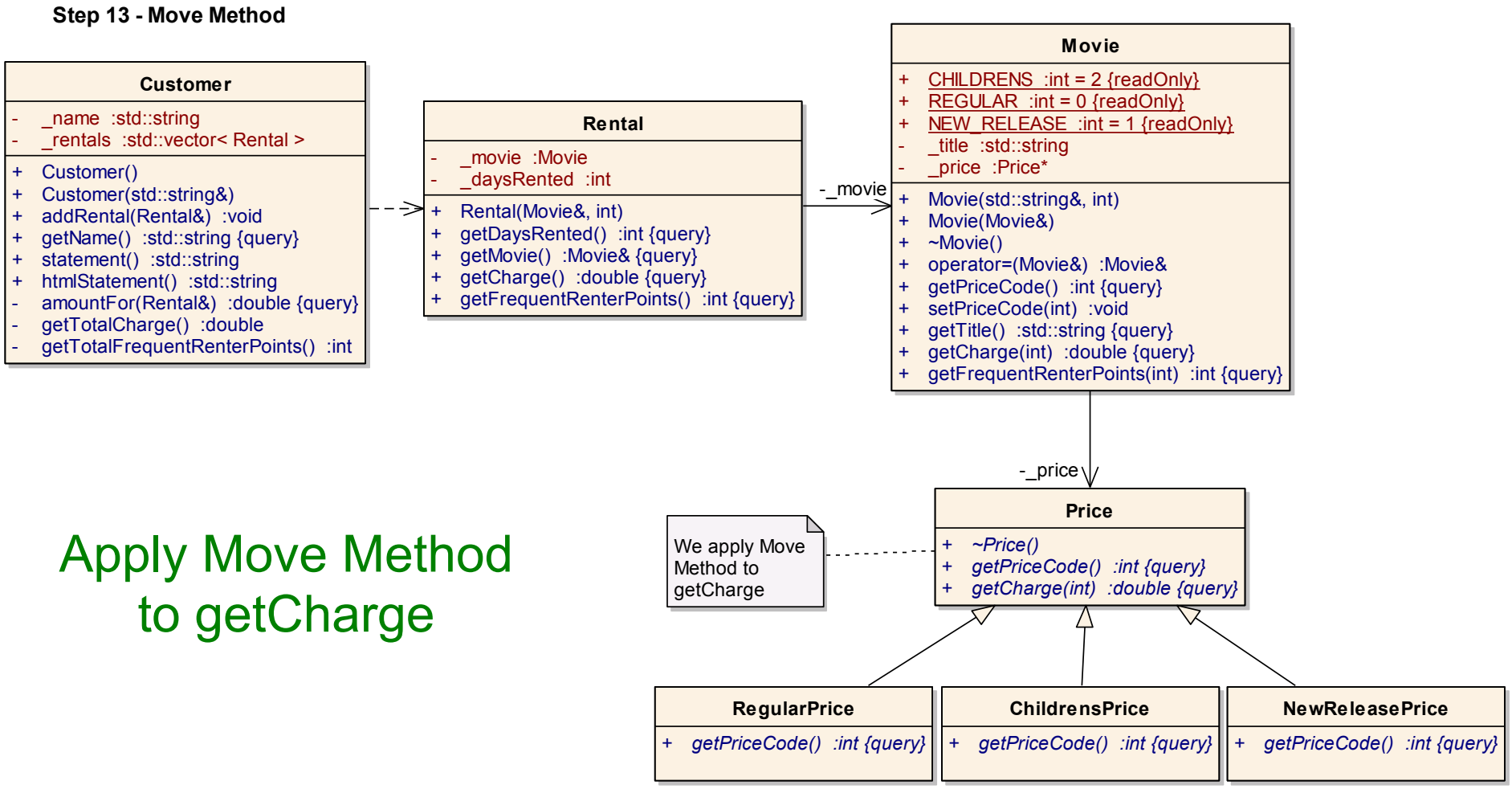
```
Movie::Movie( const std::string& title, int priceCode ):
  _title( title ),
  _price( 0)
{
  setPriceCode( priceCode );
}

Movie::Movie( const Movie& movie ):
  _title( movie._title ),
  _price( 0 )
{
  setPriceCode( movie.getPriceCode() );
}

Movie& Movie::operator=( const Movie& rhs )
{
  if ( this == &rhs )
    return *this;
  _title = rhs._title;
  setPriceCode( rhs.getPriceCode() );
  return *this;
}

int Movie::getPriceCode() const
{
  return _price->getPriceCode();
}

void Movie::setPriceCode( int arg )
{
  delete _price;
  switch ( arg ) {
  case REGULAR:
    _price = new RegularPrice;
    break;
  case CHILDRENS:
    _price = new ChildrensPrice;
    break;
  case NEW_RELEASE:
    _price = new NewReleasePrice;
    break;
  default:
    throw std::invalid_argument("Incorrect Price Code");
  }
}
```

Step 12

Copy constructor and **assignment operator** are needed to deal correctly with the Price* data member
(the default ones would do a shallow copy)

the accessor uses the new Price class

Step 13: Move Method



Apply Move Method to getCharge

```cpp
// Price.cc
#include "Price.hh"
#include "Movie.hh"

double Price::getCharge( int daysRented ) const
{
  double result = 0;
  switch ( getPriceCode() ) {
  case Movie::REGULAR:
    result += 2;
    if ( daysRented > 2 )
      result += ( daysRented - 2 ) * 1.5 ;
    break;
  case Movie::NEW_RELEASE:
    result += daysRented * 3;
    break;
  case Movie::CHILDRENS:
    result += 1.5;
    if ( daysRented > 3 )
      result += ( daysRented - 3 ) * 1.5;
    break;
  }
  return result;
}
```

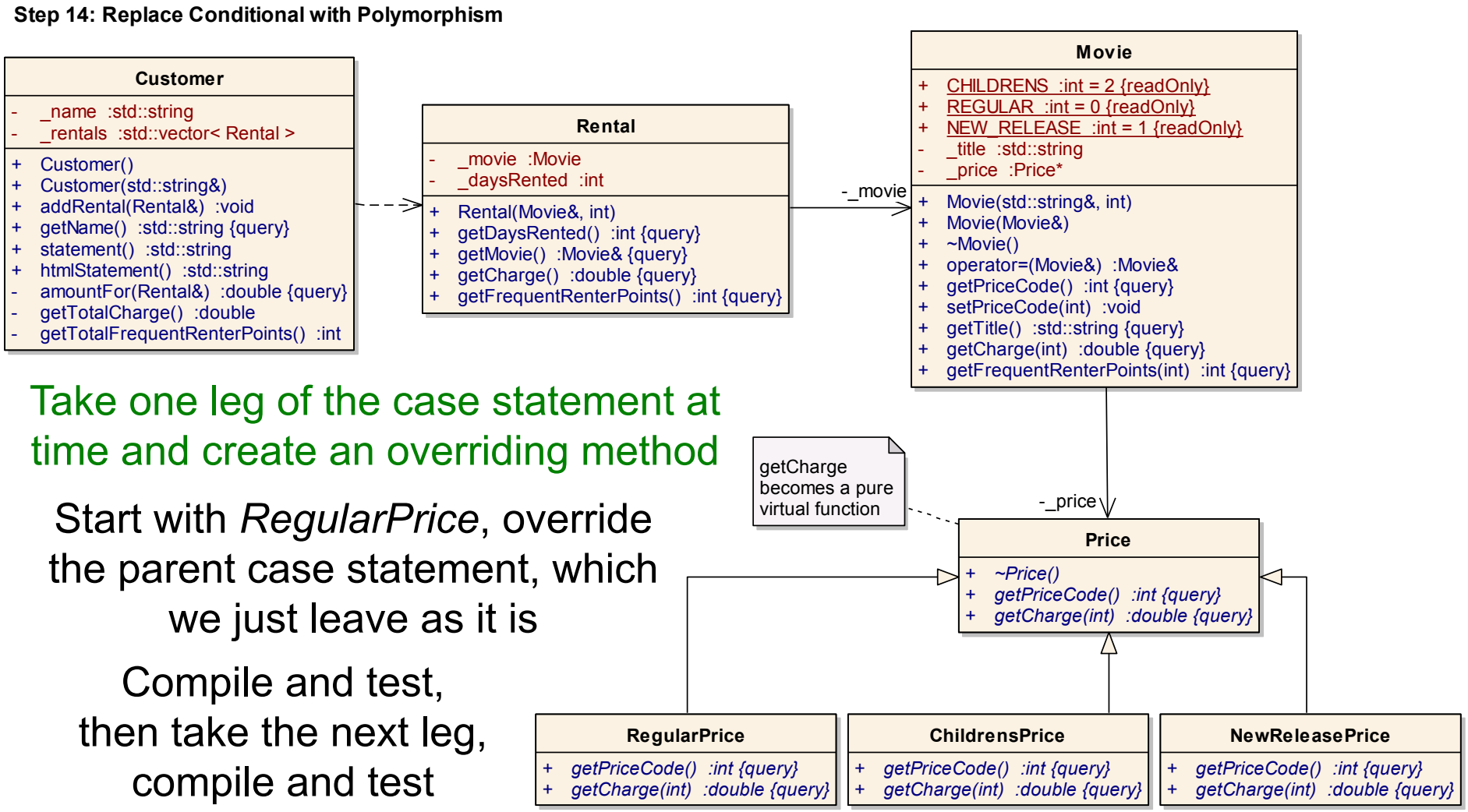
Step 13

getCharge is moved to Price

In Movie getCharge delegates the task to Price

```cpp
double Movie::getCharge( int daysRented ) const
{
  return _price->getCharge( daysRented );
}
```

Step 14: Replace Conditional with Polymorphism



Take one leg of the case statement at time and create an overriding method

Start with *RegularPrice*, override the parent case statement, which we just leave as it is

Compile and test, then take the next leg, compile and test

When done with all the legs, make *Price::getCharge* a pure virtual function

```cpp
// ChildrensPrice.cc
#include "ChildrensPrice.hh"
#include "Movie.hh"

int ChildrensPrice::
getPriceCode() const
{
  return Movie::CHILDRENS;
}

double ChildrensPrice::getCharge( int daysRented ) const
{
  double result = 1.5;
  if ( daysRented > 3 )
    result += ( daysRented - 3 ) * 1.5 ;
  return result;
}
```

```cpp
// RegularPrice.cc
#include "RegularPrice.hh"
#include "Movie.hh"

int RegularPrice::
getPriceCode() const
{
  return Movie::REGULAR;
}

double RegularPrice::getCharge( int daysRented ) const
{
  double result = 2;
  if ( daysRented > 2 )
    result += ( daysRented - 2 ) * 1.5 ;
  return result;
}
```

```cpp
// NewReleasePrice.cc
#include "NewReleasePrice.hh"
#include "Movie.hh"

int NewReleasePrice::
getPriceCode() const
{
  return Movie::NEW_RELEASE;
}

double NewReleasePrice::getCharge( int daysRented ) const
{
  return daysRented * 3;
}
```

The calculations in the switch are split in the three new classes

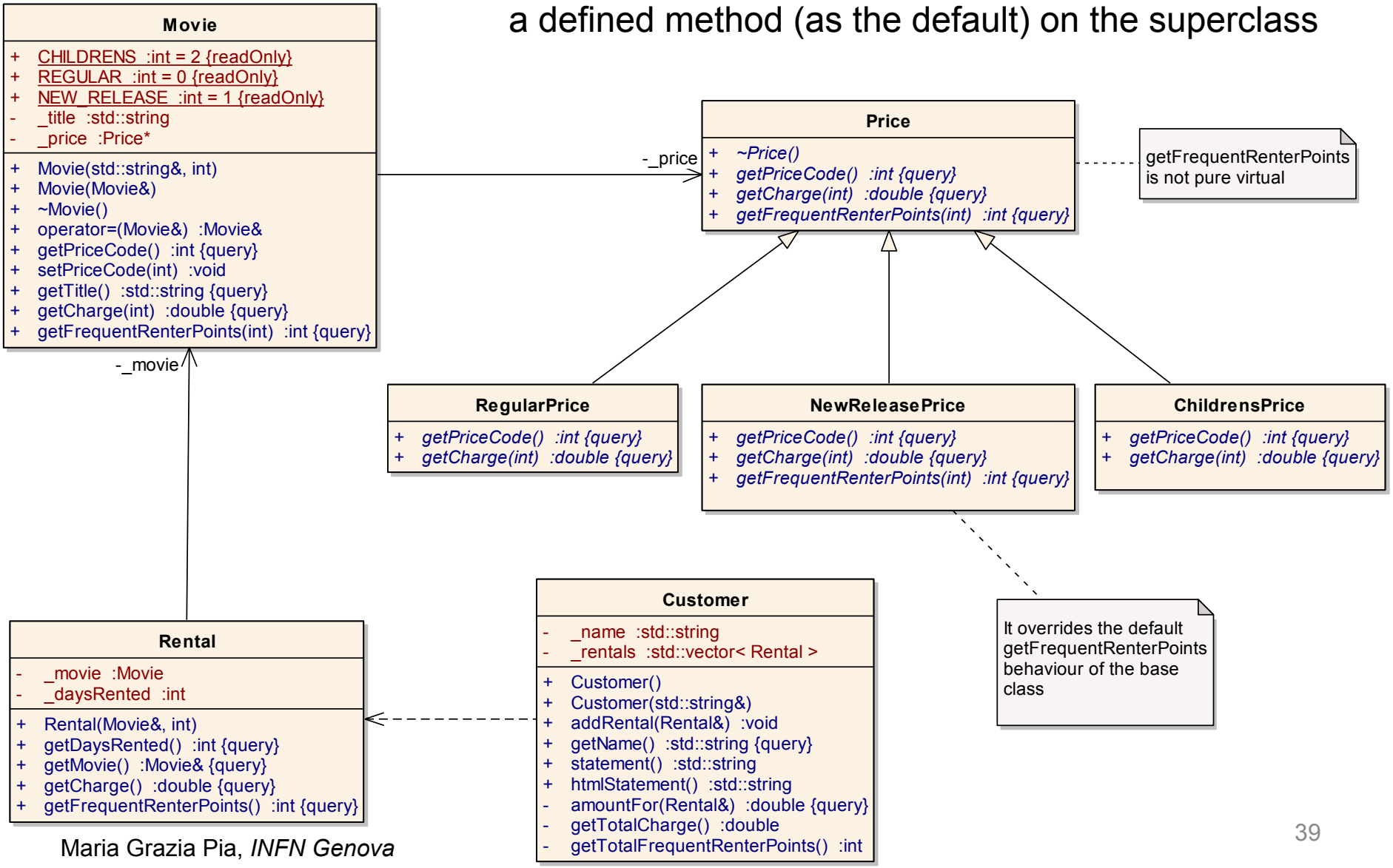
```cpp
// Price.hh
#ifndef PRICE_HH
#define PRICE_HH

class Price {
public:
  virtual ~Price();
  virtual int getPriceCode() const = 0;

  virtual double getCharge( int daysRented ) const = 0;
};

inline Price::~Price() {}

#endif // PRICE_HH
```

getCharge becomes pure virtual in the base class

Step 15

Apply the same procedure to *getFrequentRenterPoints*

In this case do not make the superclass method pure virtual
Create an overriding method for the new releases and leave a defined method (as the default) on the superclass



```
// Price.cc
#include "Price.hh"
#include "Movie.hh"

int Price::getFrequentRenterPoints( int daysRented ) const
{
  return 1;
}
```

The default implementation in the base class is good enough for ChildrensPrice and RegularPrice

```
int NewReleasePrice::getFrequentRenterPoints( int daysRented ) const
{
  return ( daysRented > 1 ) ? 2 : 1;
}
```

NewReleasePrice overrides it

Step 15: The End

Step 15 - Sequence diagram at the end of refactoring