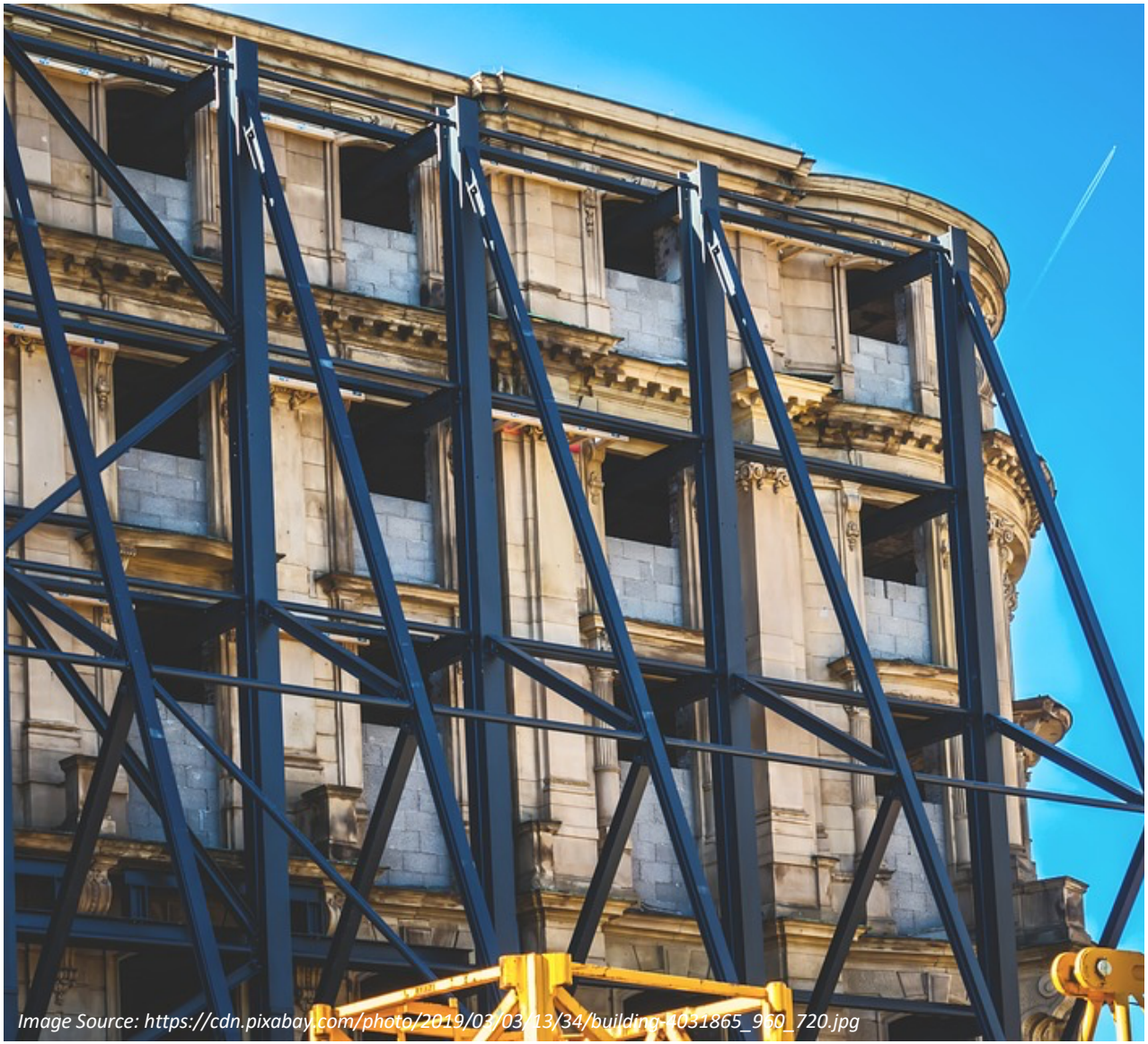
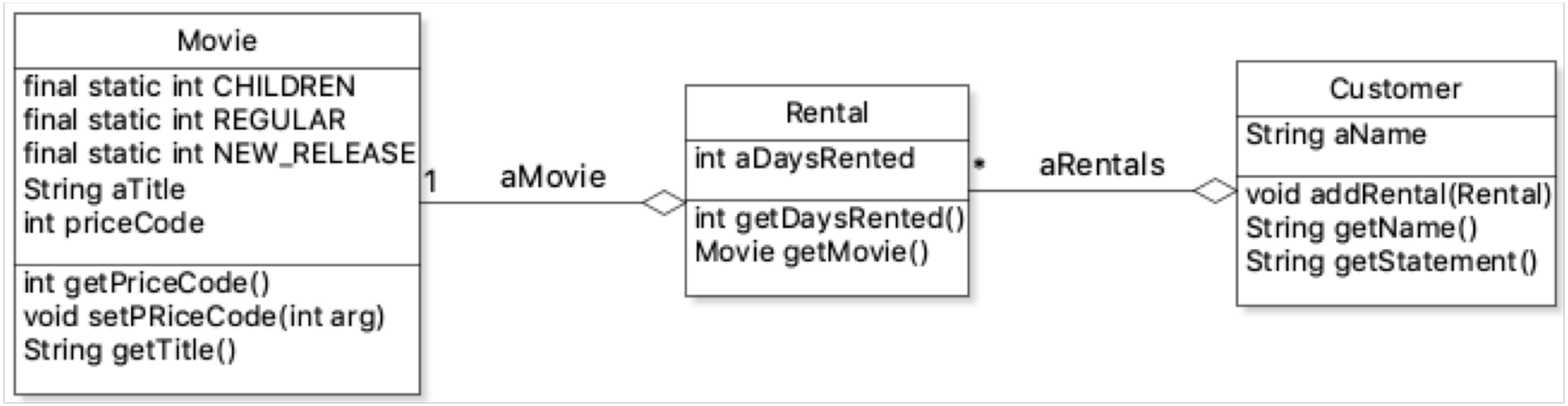



# M10
# Refactoring

Jin L.C. Guo

# Example


Movie
final static int CHILDREN
final static int REGULAR
final static int NEW_RELEASE
String aTitle
int priceCode
int getPriceCode()
void setPRiceCode(int arg)
String getTitle()
1
aMovie
Rental
int aDaysRented
int getDaysRented()
Movie getMovie()
*
aRentals
Customer
String aName
void addRental(Rental)
String getName()
String getStatement()

# What is refactoring

- 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.

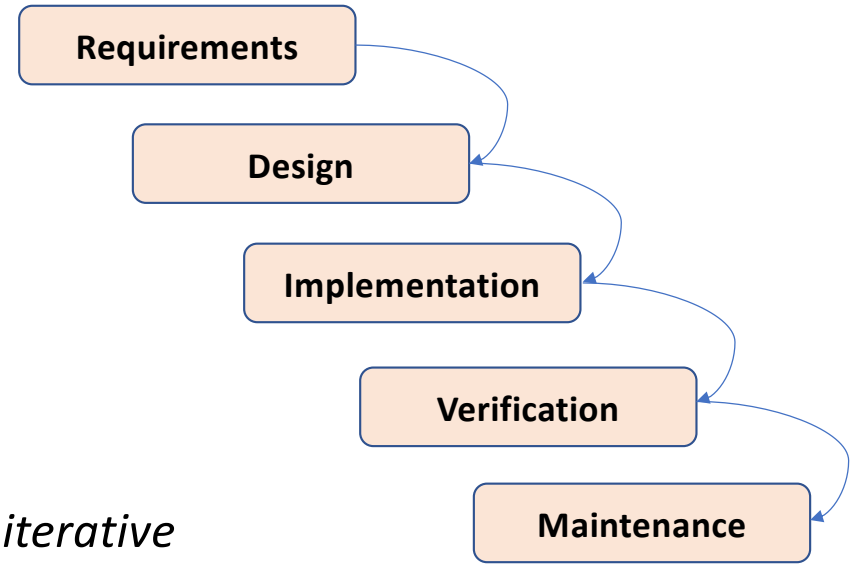
- A disciplined way to clean up code that minimizes the chances of introducing bugs.

# Design in Software Engineering

## *Software development process*

*Waterfall*

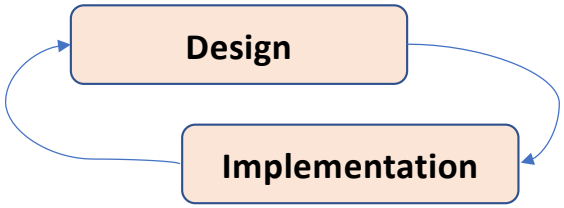
*Agile*




*Emphasize incremental and iterative development and delivery*

# Refactoring

The process of improving the design of the code after it has been written.



# When to refactor

- The rule of three

*The first time you do something, you just do it.*
*The second time you do something similar, you wince at the duplication, but you do the duplicate thing anyway.*
*The third time you do something similar, you refactor.*

# When to refactor

- When you add Function
- When you need to Fix a bug
- When you do code review

**To make the code**
**more understandable**
**in order to make changes**

# Code Smells

- Duplicated Code
- Long Method
- God Class
- Primitive obsession
- Temporary Field
- Long parameter list
- Comments

……

# First Step

Build solid tests

Run your tests frequently.

*When finding a bug, start by writing a unit test that exposes the bug.*

# Refactoring Techniques

- Composting Methods
  - Extract Method
  - Inline Method
  - Extract Variable
  - Inline Variable
  - Rename Variable
  - Introduce Parameter Object

  ……

# Inline Variable

```
double basePrice = _quantity * _itemPrice;
        if (basePrice > 1000)
            return basePrice * 0.95;
        else
            return basePrice * 0.98;
```

```
if (basePrice() > 1000)
            return basePrice() * 0.95;
        else
            return basePrice() * 0.98;

...
double basePrice() {
       return _quantity * _itemPrice;
}
```

# Introduce Parameter Object

```
amountInvoiced(Date startDate, Date endDate) {...}
amountReceived(Date startDate, Date endDate){...}
amountOverdue(Date startDate, Date endDate){...}
```

```
amountInvoiced(DateRange aDateRange) {...}
amountReceived(DateRange aDateRange){...}
amountOverdue(DateRange aDateRange){...}
```

# Refactoring Techniques

- Encapsulation
  - Extract Class
  - Inline Class
  - Encapsulate Collection
  - Hide Delegation
  - Remove Middle Man

  ……

# Encapsulate Collection

```
class Customer {
  private List<Rental> aRentals = new ArrayList<>();
  public void setRental(List<Rental> arg) {…}
  public List<Rental> getRental(Rental arg) {…}
```

```
class Customer {
  private List<Rental> aRentals = new ArrayList<>();
  public void addRental(Rental arg) {…}
  public void removeRental(Rental arg) {…}
```

# Refactoring Techniques

- Moving Features
  - Move Function
  - Move Field
  - Move Statement into Function
  - Move Statement to Callers
  - Slide Loop

  ……

# Split Loop

```
while (rentals.hasNext()) {
        Rental each = rentals.next();
        result += "\t" + each.getMovie().getTitle()+ "\t" +
            String.valueOf(each.calculateAmount()) + "\n";
        totalAmount += each.calculateAmount();
}
```

```
while (rentals.hasNext()) {
        Rental each = rentals.next();
        result += "\t" + each.getMovie().getTitle()+ "\t" +
            String.valueOf(each.calculateAmount()) + "\n";
}

while (rentals.hasNext()) {
        Rental each = rentals.next();
        totalAmount += each.calculateAmount();
}
```

# Refactoring Techniques

- Organizing Data
  - Splitting Variable
  - Rename Field
  - Replace Derived Variable with Query
  - Change Reference to Value

  ……

# Replace Derived Variable with Query

```
long get discountedTotal() {return aDiscountedTotal;}
void set discount(long pNumber) {
     aDiscountedTotal += aDiscount – pNumber;
     aDiscount = pNumber;
}
```

```
long get discountedTotal() {return aBaseTotal – aDiscount;}
void set discount(long pNumber) {
     aDiscount = pNumber;
}
```

# More Categories of Refactoring Techniques

- Simplifying Conditional Logic
- Refactoring APIs
- Dealing with Inheritance

# Final word

The true test of good code is how easy it is to change it.

-- Martin Fowler


Image source: https://farm6.staticflickr.com/5696/21227092111_0b17a57cb1_b.jpg

# Acknowledgement


- Contents and examples are adapted from the following resources:
  - *Refactoring Improving the Design of Existing Code by Martin Fowler*