Refactoring: Example

CPSC 501: Advanced Programming Techniques
Fall 2020

Jonathan Hudson, Ph.D
Instructor
Department of Computer Science
University of Calgary

Wednesday, August 5, 2020

UNIVERSITY OF CALGARY

Lets do something with all that

Example 1

- **Form Template Method**
  - Used when there is similar (but not identical) code in sibling classes
    - Their methods do similar steps in the same order
      - But the steps are different
  - Goal is **Template Method** design pattern
    - Identical code put into common superclass
    - Differing code put into subclasses

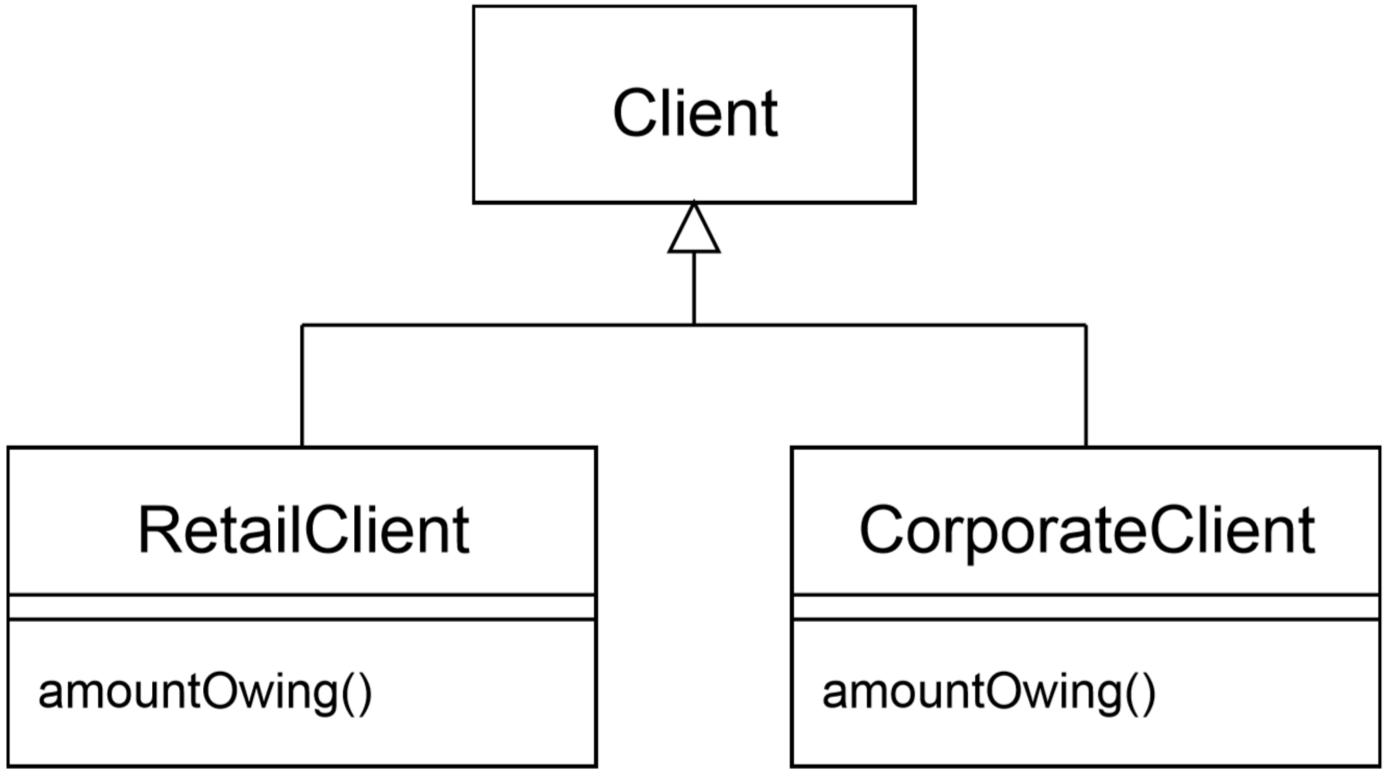
Example 1

- Original code:

```java
public class RetailClient extends Client {

    public double amountOwing(int daysWorked) {
        double base = daysWorked * dailyRate();
        double discount = base * discountRate();
        return base - discount;
    }
}

public class CorporateClient extends Client {

    public double amountOwing(int daysWorked) {
        double base = retainer + (daysWorked / 30.0) * monthlyRate();
        double discount = 500.0 + base * 0.02;
        return base - discount;
    }
}
```

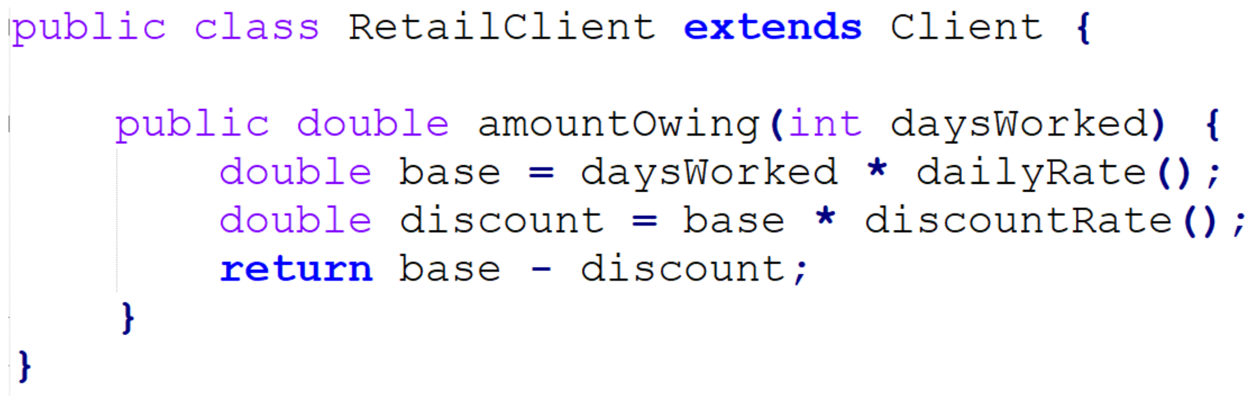
Example 1

- Original code:

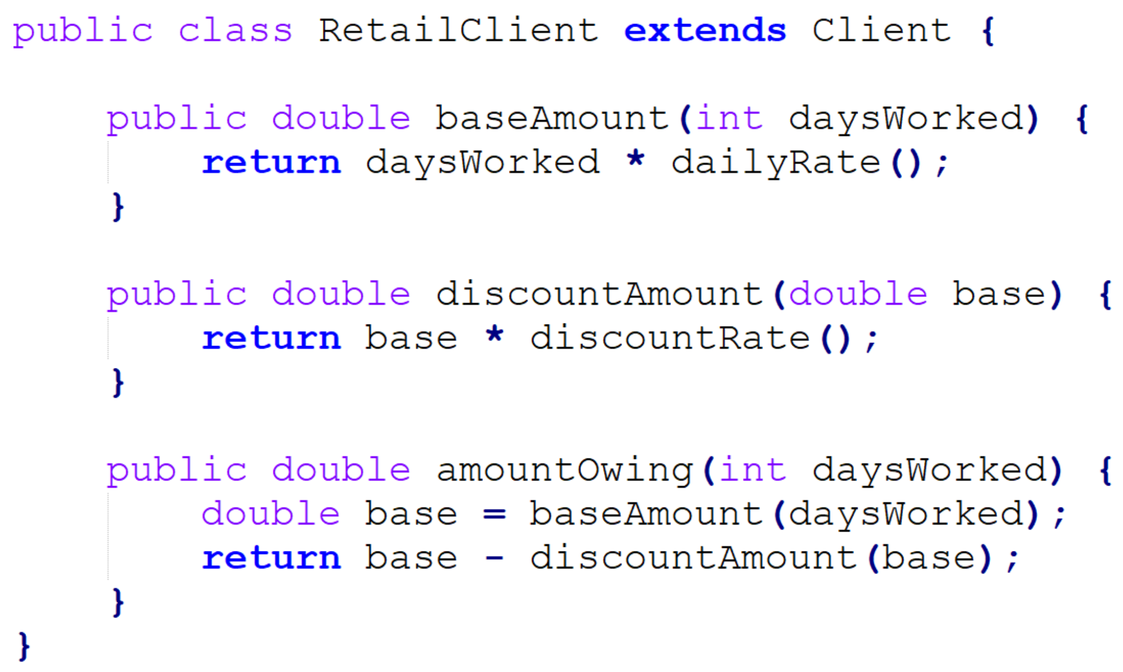


Example 1

- Mechanics:
  - Extract methods that are either identical or completely different



```java
public class RetailClient extends Client {

    public double amountOwing(int daysWorked) {
        double base = daysWorked * dailyRate();
        double discount = base * discountRate();
        return base - discount;
    }
}
```



```java
public class RetailClient extends Client {

    public double baseAmount(int daysWorked) {
        return daysWorked * dailyRate();
    }

    public double discountAmount(double base) {
        return base * discountRate();
    }

    public double amountOwing(int daysWorked) {
        double base = baseAmount(daysWorked);
        return base - discountAmount(base);
    }
}
```

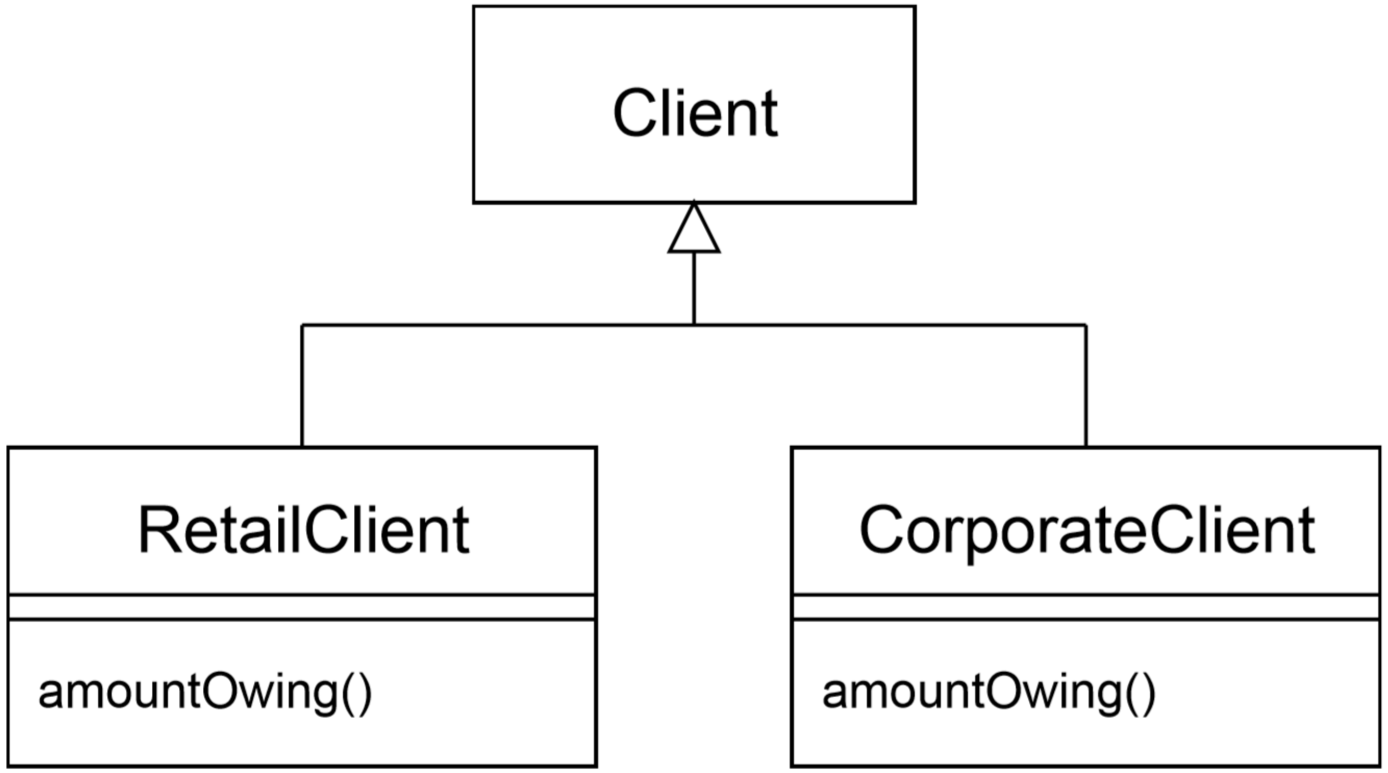
Example 1

```java
public class CorporateClient extends Client {

    public double amountOwing(int daysWorked) {
        double base = retainer + (daysWorked / 30.0) * monthlyRate();
        double discount = 500.0 + base * 0.02;
        return base - discount;
    }
}
```

```java
public class CorporateClient extends Client {

    public double baseAmount(int daysWorked) {
        return retainer + (daysWorked / 30.0) * monthlyRate();
    }

    public double discountAmount(double base) {
        return 500.0 + base * 0.02;
    }

    public double amountOwing(int daysWorked) {
        double base = baseAmount(daysWorked);
        return base - discountAmount(base);
    }
}
```

Example 1

- Original code:



Example 1

- Pull up the common method into the superclass, and declare differing methods as abstract

```java
public class RetailClient extends Client {

    public double baseAmount(int daysWorked) {
        return daysWorked * dailyRate();
    }

    public double discountAmount(double base) {
        return base * discountRate();
    }

    public double amountOwing(int daysWorked) {
        double base = baseAmount(daysWorked);
        return base - discountAmount(base);
    }
}
```

```java
public class CorporateClient extends Client {

    public double baseAmount(int daysWorked) {
        return retainer + (daysWorked / 30.0) * monthlyRate
    }

    public double discountAmount(double base) {
        return 500.0 + base * 0.02;
    }

    public double amountOwing(int daysWorked) {
        double base = baseAmount(daysWorked);
        return base - discountAmount(base);
    }
}
```

Example 1

- Pull up the common method into the superclass, and declare differing methods as abstract

```java
public class Client {

    public double amountOwing(int daysWorked) {
        double base = baseAmount(daysWorked);
        return base - discountAmount(base);
    }

    public abstract double baseAmount(int daysWorked);

    public abstract double discountAmount(double base);
}
```

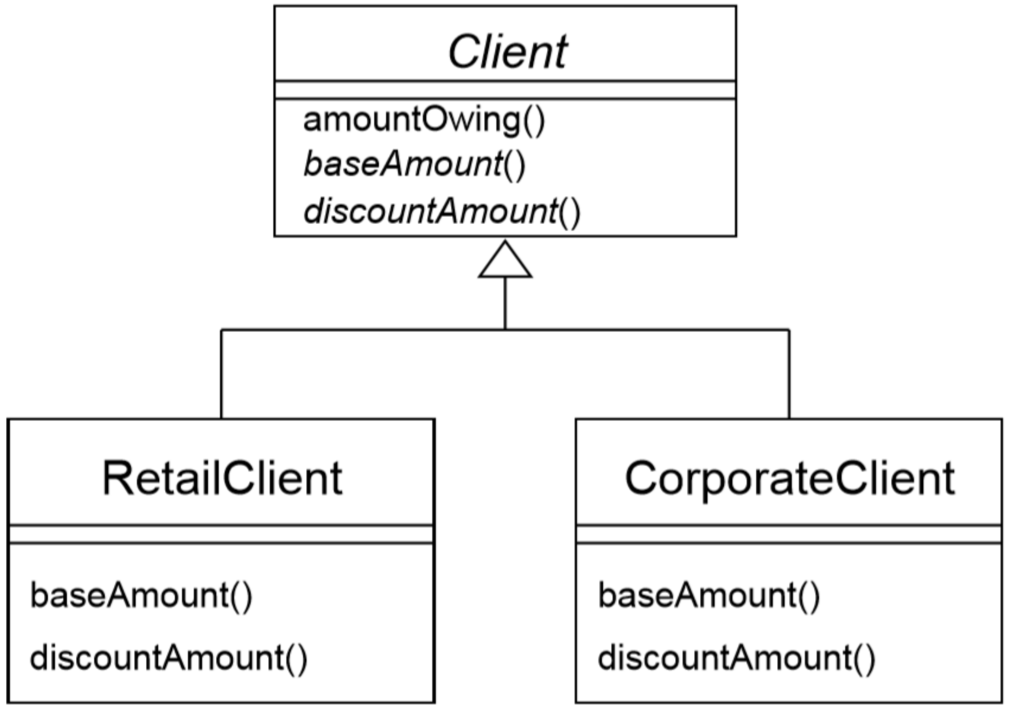
Example 1

- Remove pulled up methods from subclasses

```
public class RetailClient extends Client {

    public double baseAmount(int daysWorked) {
        return daysWorked * dailyRate();
    }

    public double discountAmount(double base) {
        return base * discountRate();
    }
}

public class CorporateClient extends Client {

    public double baseAmount(int daysWorked) {
        return retainer + (daysWorked / 30.0) * monthlyRate();
    }

    public double discountAmount(double base) {
        return 500.0 + base * 0.02;
    }
}
```

Example 1

- Result



Example 1

- Now easy to add new kinds of Clients
  - Create a new concrete subclass, overriding the abstract methods

How about something else

UNIVERSITY OF CALGARY

Example 2

- **Replace Type Code with Subclasses**
  - Allows you to remove switch statements, if followed by **Replace Conditional with Polymorphism**

Example 2

- Original code:

```
public class Account {

    private int type;
    static final int SAVINGS = 0;
    static final int CHEQUING = 1;

    public Account(int typeCode) {
        type = typeCode;
    }
}
```

Example 2

- Mechanics
  - Self-encapsulate the type code
    - If used by the constructor, replace constructor with factory method

```
public class Account {

    private int type;
    static final int SAVINGS = 0;
    static final int CHEQUING = 1;

    private Account(int typeCode) {
        type = typeCode;
    }

    public int getType() {
        return type;
    }

    public static Account create(int typeCode) {
        return new Account(typeCode);
    }
}
```

Example 2

- For each type code, create a subclass
  - Override the getType() method
  - Change the factory method

```
public class Savings extends Account {

    public int getType() {
        return Account.SAVINGS;
    }
}

public class Chequing extends Account {

    public int getType() {
        return Account.CHEQUING;
    }
}
```

Example 2

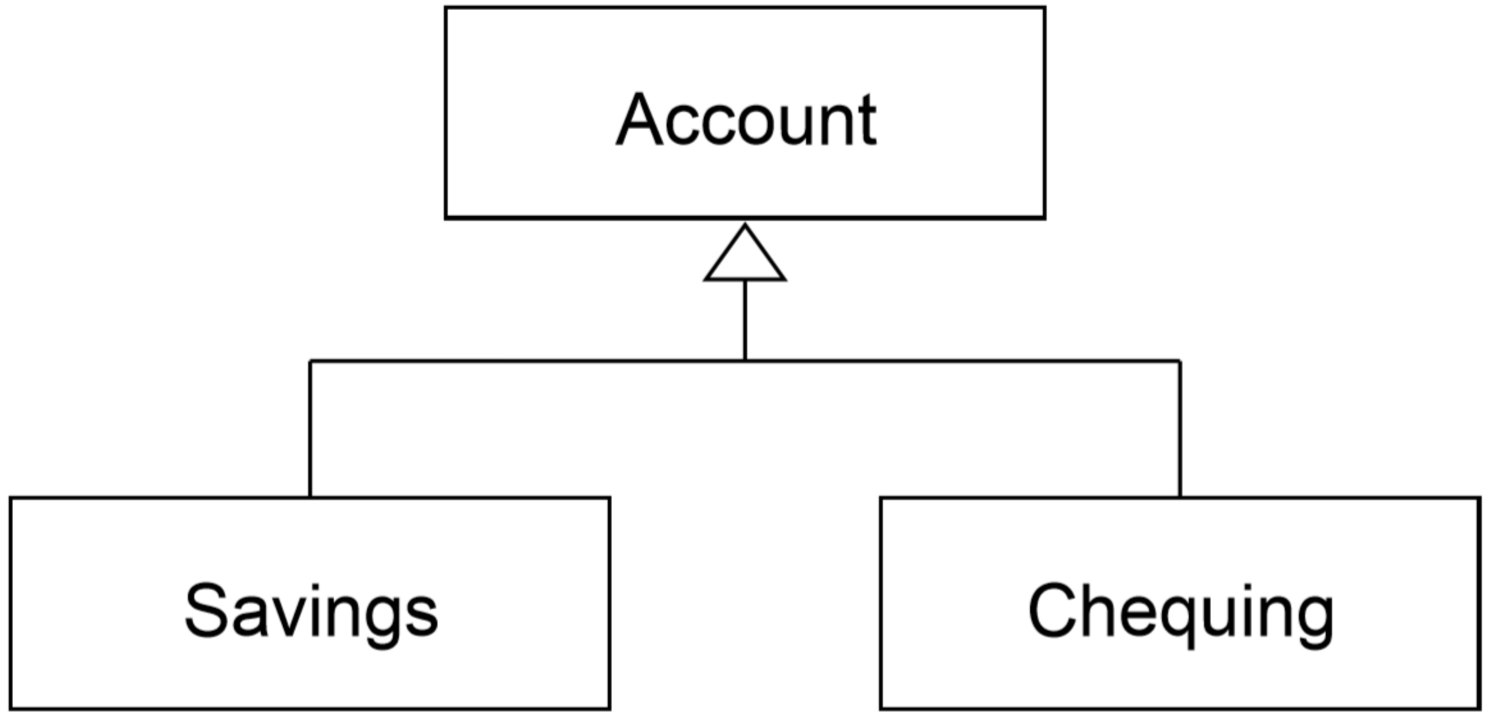
```
public class Account {

    private int type;
    static final int SAVINGS = 0;
    static final int CHEQUING = 1;

    private Account(int typeCode) {
        type = typeCode;
    }

    public int getType() {
        return type;
    }

    public static Account create(int typeCode) {
        switch (typeCode) {
            case SAVINGS:
                return new Savings();
            case CHEQUING:
                return new Chequing();
            default:
                throw new IllegalArgumentException("Bad type code");
        }
    }
}
```

Example 2

- Remove the type code field
    - Declare accessors as abstract

```
public class Account {

    static final int SAVINGS = 0;
    static final int CHEQUING = 1;

    public abstract int getType();

    public static Account create(int typeCode) {
        switch (typeCode) {
            case SAVINGS:
                return new Savings();
            case CHEQUING:
                return new Chequing();
            default:
                throw new IllegalArgumentException("Bad type code");
        }
    }
}
```

Example 2



Example 2

- Use **Push Down Method** and **Push Down Field** for features specific to a subclass
- If you have switch statements in methods other than the factory method, use **Replace Conditional with Polymorphism**

Onward to …
the next topic.

Jonathan Hudson
jwhudson@ucalgary.ca
https://pages.cpsc.ucalgary.ca/~hudsonj/

UNIVERSITY OF CALGARY