Advanced Software Development: Refactoring Examples

CPSC 501: Advanced Programming Techniques
Fall 2022

Jonathan Hudson, Ph.D
Assistant Professor (Teaching)
Department of Computer Science
University of Calgary

Friday, September 2, 2022

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

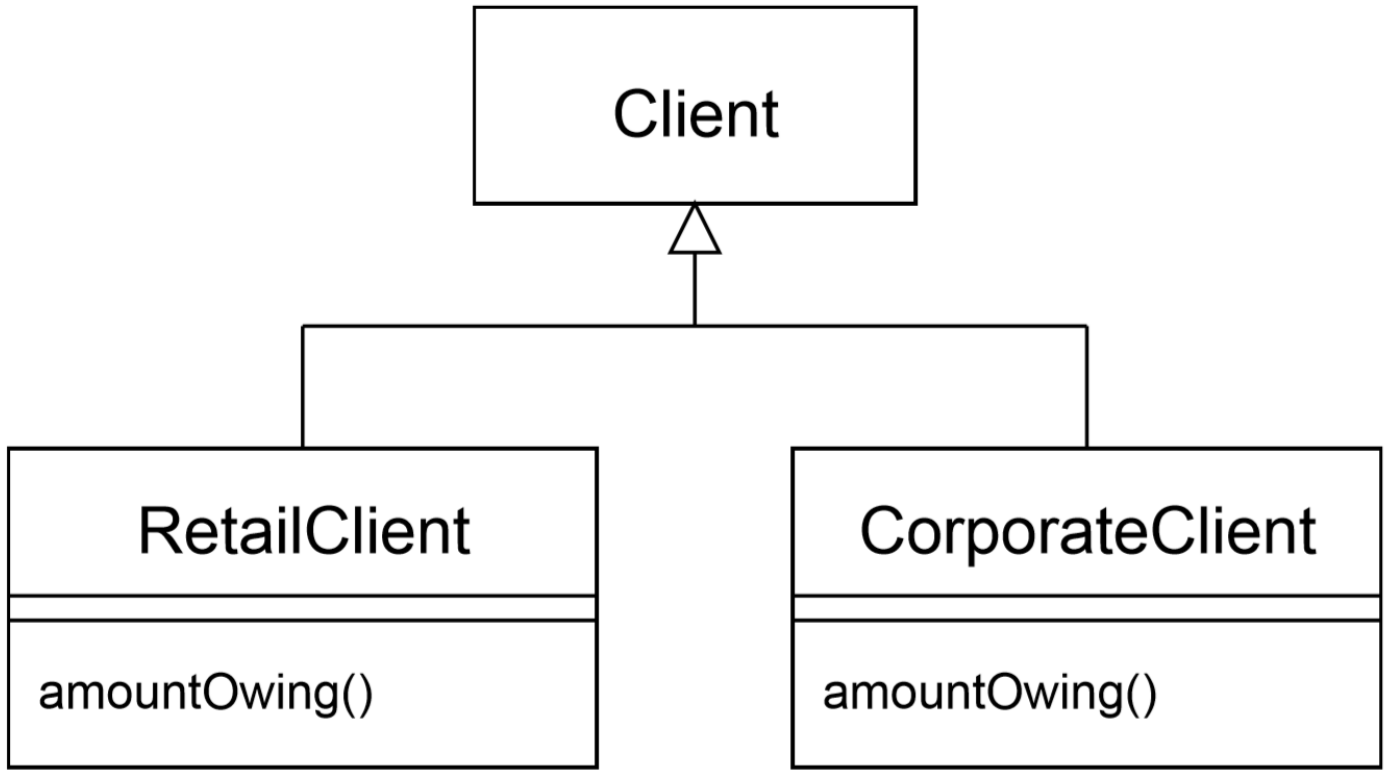
- Original code:

```
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;
    }
```

```
public class RetailClient  extends Client{

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

Example 1

- Original code:



Example 1

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

OLD

```
public class RetailClient  extends Client{

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

NEW

```
public class RetailClient extends Client {

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

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

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

Example 1

OLD

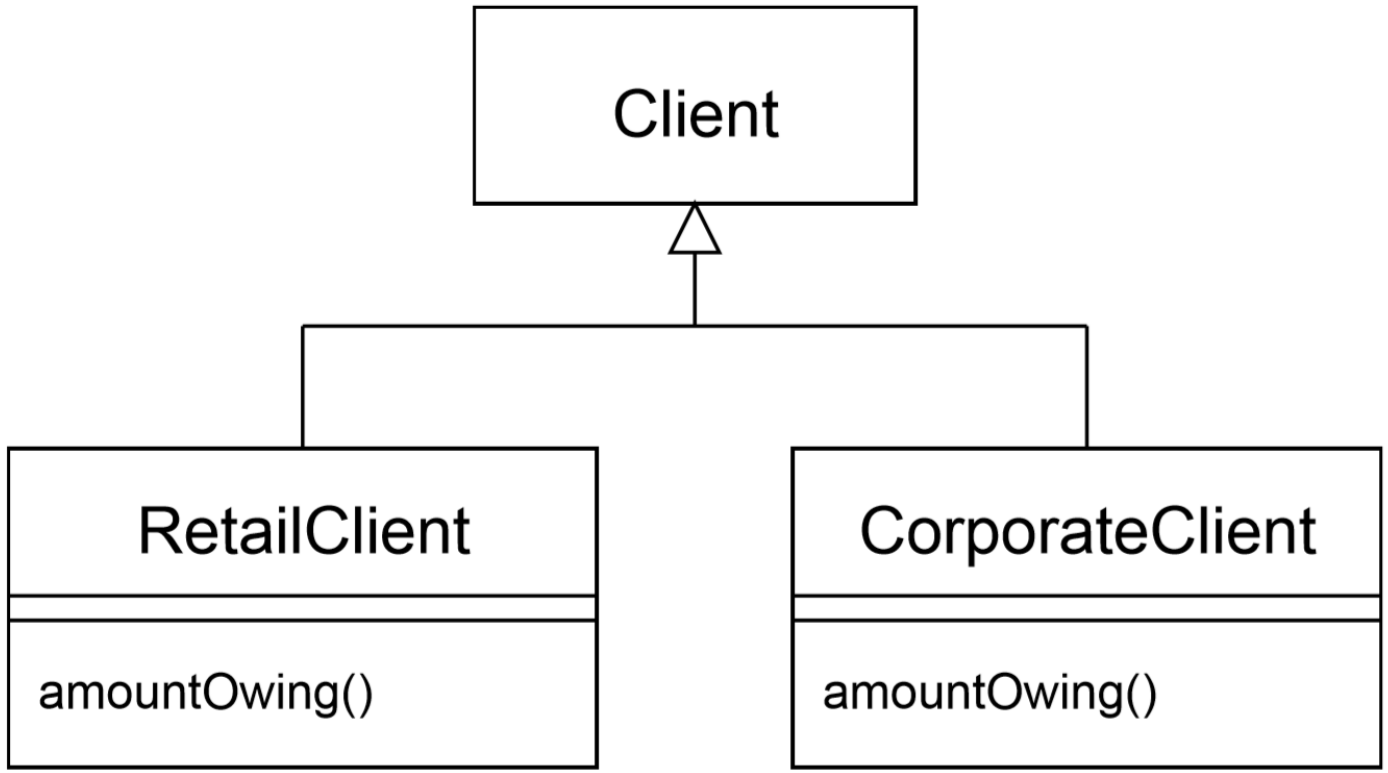
```
public class CorporateClient extends Client{

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

NEW

```
public class CorporateClient extends Client {

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

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

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

Example 1

- Original code:



Example 1

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

```
public class CorporateClient extends Client {

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

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

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

```
public class RetailClient extends Client {

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

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

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

Example 1

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

```
public abstract 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

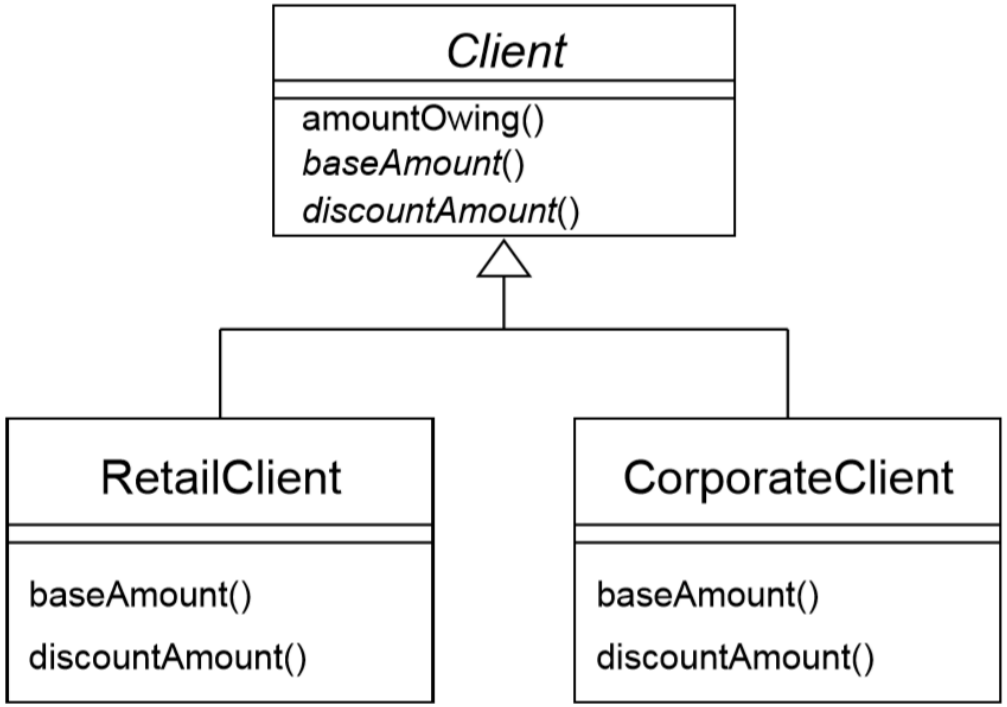
- Remove pulled up methods from subclasses

```
public class 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 class RetailClient  extends Client {

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

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

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

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 {

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

   private final int type;

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

   private Account(int typeCode) {
      type = typeCode;
   }
   public static Account create(int typeCode) {
      return new Account(typeCode);
   }
   public int getType() {
      return type;
   }
}
```

Example 2

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

```
public class Chequing extends Account {
   public Chequing() {
      super(Account.CHEQUING);
   }

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

```
public class Savings extends Account {
   public Savings() {
      super(Account.SAVINGS);
   }

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

Example 2

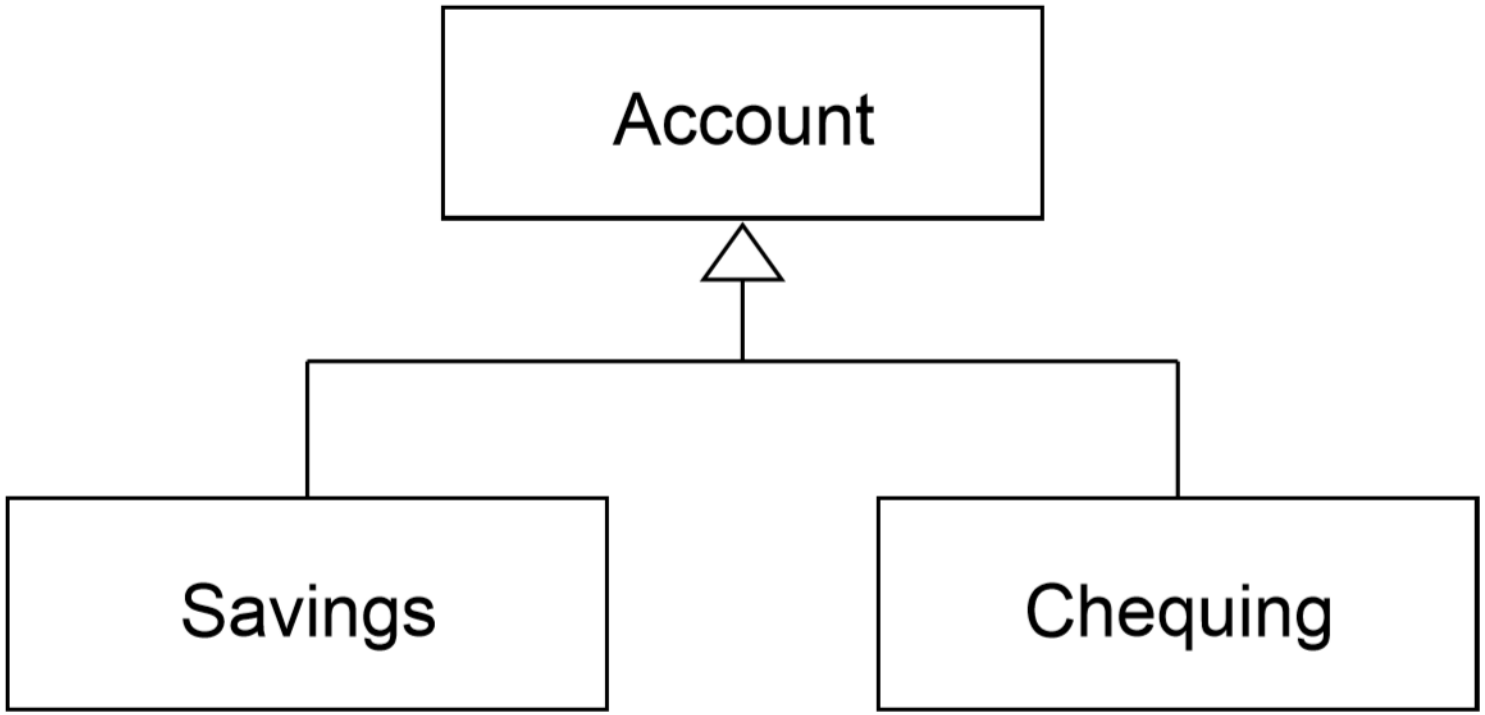
```
public class Account {

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

   private final int type;

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

   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!");

      }
   }

   public int getType() {
      return type;
   }
}
```

Example 2

- Remove the type code field
  - Declare accessors as abstract

```
public abstract class Account {
   static final int SAVINGS = 0;
   static final int CHEQUING = 1;

   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!");

      }
   }

   public abstract int getType();
}
```

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 …
Docker.

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

UNIVERSITY OF CALGARY