Refactoring Techniques

Composing Methods

Extract Method

```
void PrintOwing()
{
  this.PrintBanner();

  // Print details.
  Console.WriteLine("name: " + this.name);
  Console.WriteLine("amount: " + this.GetOutstanding());
}
```

```
void PrintOwing()
{
  this.PrintBanner();
  this.PrintDetails(this.GetOutstanding());
}

void PrintDetails(double outstanding)
{
  Console.WriteLine("name: " + this.name);
  Console.WriteLine("amount: " + this.outstanding);
}
```

Inline Method

```
class PizzaDelivery
{
  // ...
  int GetRating()
  {
    return MoreThanFiveLateDeliveries() ? 2 : 1;
  }
  bool MoreThanFiveLateDeliveries()
  {
    return numberOfLateDeliveries > 5;
  }
}
```

```
class PizzaDelivery
{
  // ...
  int GetRating()
  {
    return numberOfLateDeliveries > 5 ? 2 : 1;
  }
}
```

Extract Variable

```
void RenderBanner()
{
  if ((platform.ToUpper().IndexOf("MAC") > -1) &&
      (browser.ToUpper().IndexOf("IE") > -1) &&
       wasInitialized() && resize > 0 )
  {
    // do something
  }
}
```

```
void RenderBanner()
{
  readonly bool isMacOs = platform.ToUpper().IndexOf("MAC") > -1
  readonly bool isIE = browser.ToUpper().IndexOf("IE") > -1;
  readonly bool wasResized = resize > 0;

  if (isMacOs && isIE && wasInitialized() && wasResized)
  {
    // do something
  }
}
```

Inline Temp

```
bool HasDiscount(Order order)
{
  double basePrice = order.BasePrice();
  return basePrice > 1000;
}
```

```
bool HasDiscount(Order order)
{
  return order.BasePrice() > 1000;
}
```

Replace Temp with Query

```
double CalculateTotal()
{
  double basePrice = quantity * itemPrice;

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

* May have to trade off between code readability and reusibility VS performance

```
double CalculateTotal()
{
  if (BasePrice() > 1000)
  {
    return BasePrice() * 0.95;
  }
  else
  {
    return BasePrice() * 0.98;
  }
}
double BasePrice()
{
  return quantity * itemPrice;
}
```

Replace Method with Method Object

```
public class Order
{
  // ...
  public double Price()
  {
    double primaryBasePrice;
    double secondaryBasePrice;
    double tertiaryBasePrice;
    // Perform long computation.
  }
}
```

```
public class Order
{
  // ...
  public double Price()
  {
    return new PriceCalculator(this).Compute();
  }
}
```

```
public class PriceCalculator
{
  private double primaryBasePrice;
  private double secondaryBasePrice;
  private double tertiaryBasePrice;

  public PriceCalculator(Order order)
  {
    // Copy relevant information from the
    // order object.
  }

  public double Compute()
  {
    // Perform long computation.
  }
}
```

Split Temporary Variable

```
double temp = 2 * (height + width);
Console.WriteLine(temp);
temp = height * width;
Console.WriteLine(temp);
```

```
readonly double perimeter = 2 * (height + width);
Console.WriteLine(perimeter);
readonly double area = height * width;
Console.WriteLine(area);
```

Remove Assignments to Parameters

```
int Discount(int inputVal, int quantity)
{
  if (inputVal > 50)
  {
    inputVal -= 2;
  }
  // ...
}
```

* Update method parameter

```
int Discount(int inputVal, int quantity)
{
  int result = inputVal;

  if (inputVal > 50)
  {
    result -= 2;
  }
  // ...
}
```

Substitute Algorithm

```
string FoundPerson(string[] people)
{
  for (int i = 0; i < people.Length; i++)
  {
    if (people[i].Equals("Don"))
    {
      return "Don";
    }
    if (people[i].Equals("John"))
    {
      return "John";
    }
    if (people[i].Equals("Kent"))
    {
      return "Kent";
    }
  }
  return String.Empty;
}
```

```
string FoundPerson(string[] people)
{
  List<string> candidates = new List<string>() {"Don", "John", "

  for (int i = 0; i < people.Length; i++)
  {
    if (candidates.Contains(people[i]))
    {
      return people[i];
    }
  }

  return String.Empty;
}
```

Moving Features between Objects

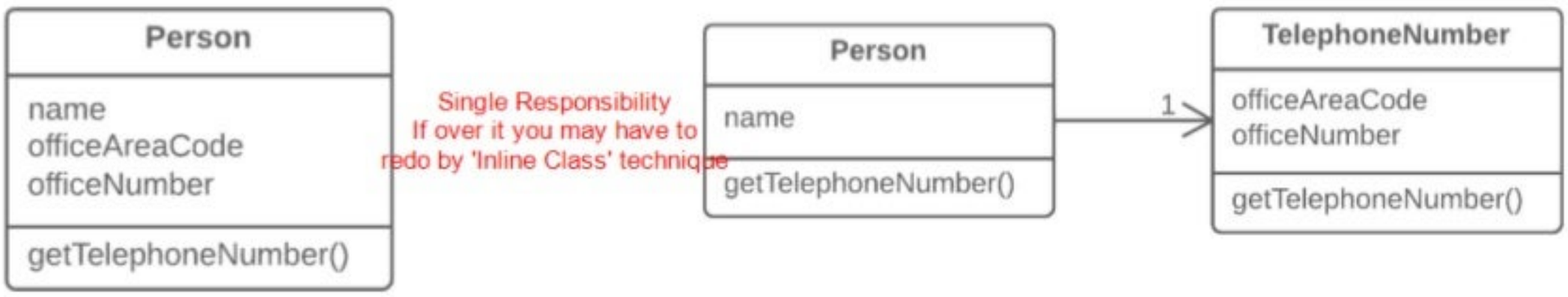
Move Method

Move method which used in another class than in its own class

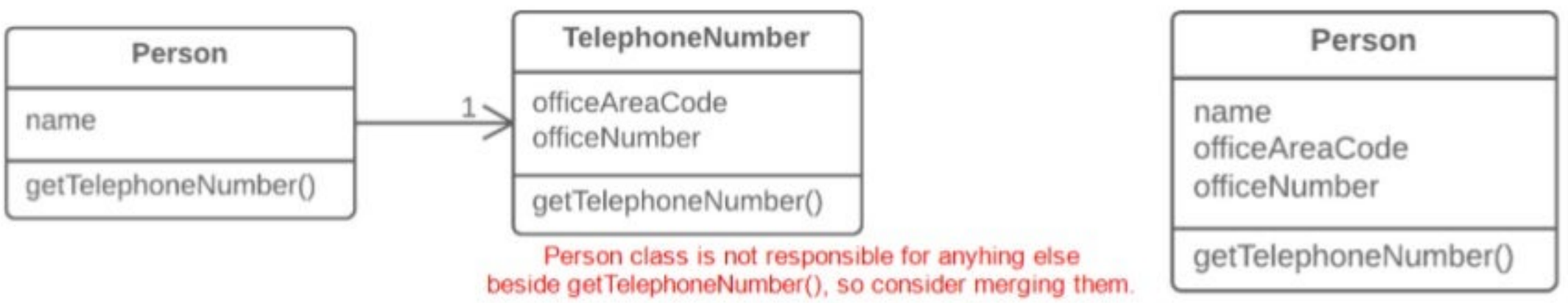
Move Field

Move fields which used in another class than in its own class.

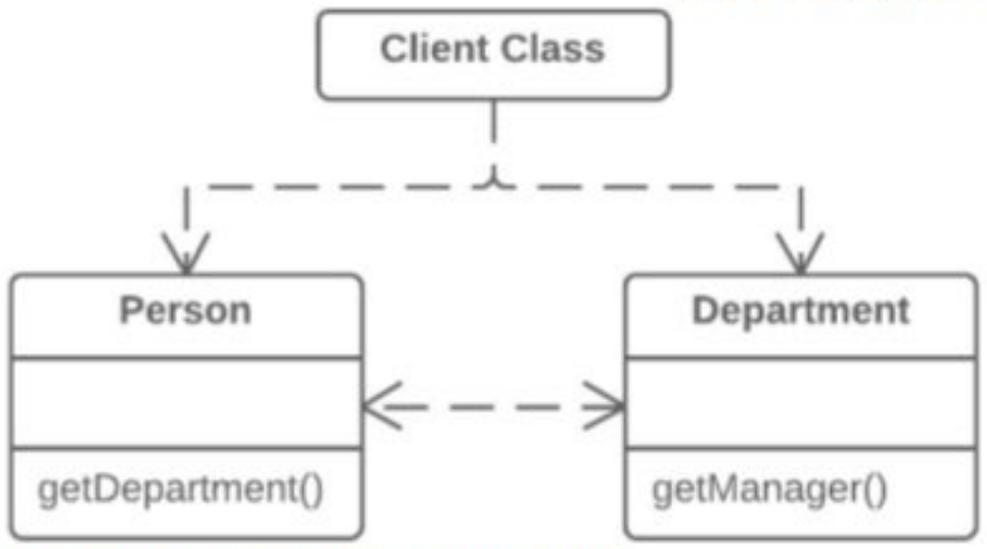
Extract Class



Inline Class



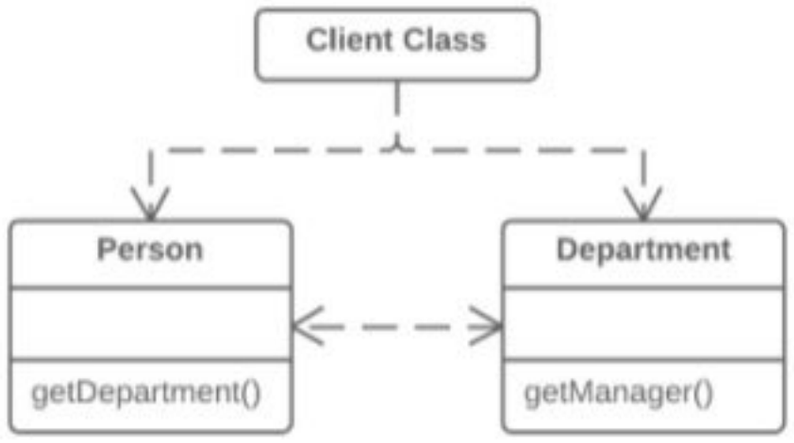
Hide Delegate



Client get access to Department via Person and call to Department directly

Create method on Person and delegate call to Department without Client knowing that.

Remove Middleman



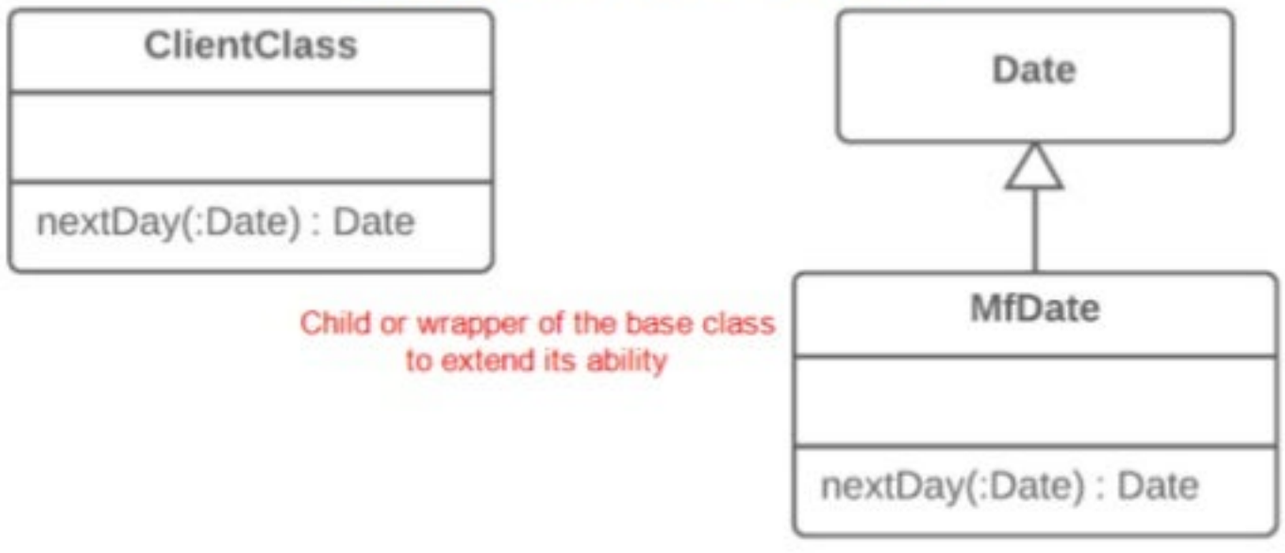
Class Person may have too many methods that delegate work to Department, so consider split them.

Introduce Foreign Method

```
class Report
{
  // ...
  void SendReport()
  {
    DateTime nextDay = previousEnd.AddDays(1);
    // ...
  }
}
```

```
class Report
{
  // ...
  void SendReport()
  {
    DateTime nextDay = NextDay(previousEnd);
    // ...
  }
  private static DateTime NextDay(DateTime date)
  {
    return date.AddDays(1);
  }
}
```

Introduce Local Extension



Organizing Data

Self-Encapsulate Field

```
class Range
{
  private int low, high;

  bool Includes(int arg)
  {
    return arg >= low && arg <= high;
  }
}
```

```
class Range
{
  private int low, high;

  int Low {
    get { return low; }
  }
  int High {
    get { return high; }
  }

  bool Includes(int arg)
  {
    return arg >= Low && arg <= High;
  }
}
```

Can make it readonly or have computation before return

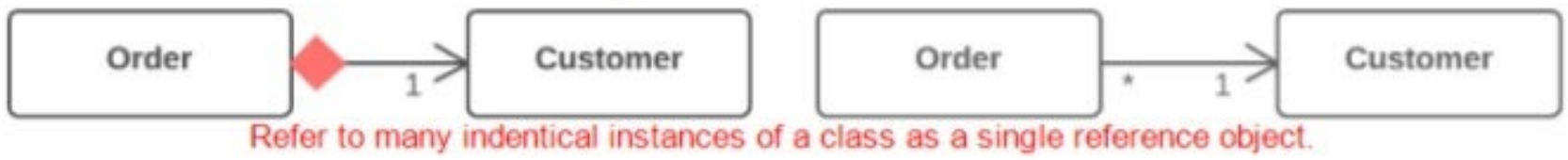
Encapsulate Field

```
class Person
{
  public string name;
}
```

C3 have shorthand property
public string Name { get; set; }

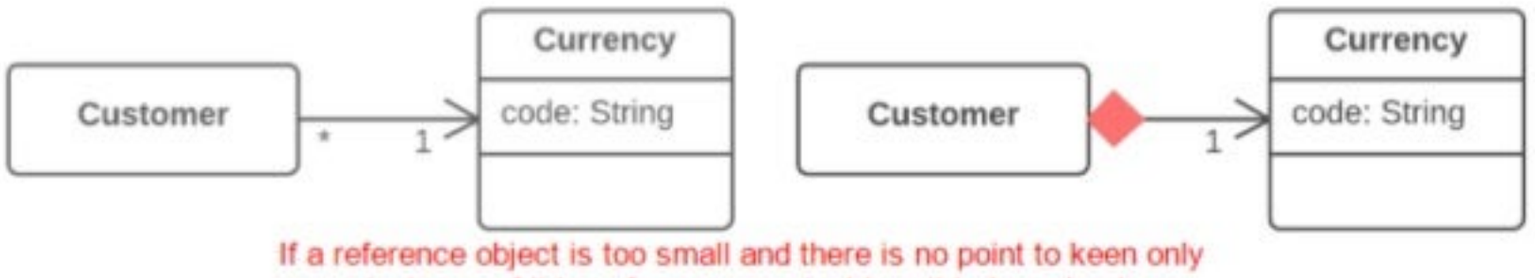
```
class Person
{
  private string name;

  public string Name
  {
    get { return name; }
    set { name = value; }
  }
}
```

Change Value to Reference



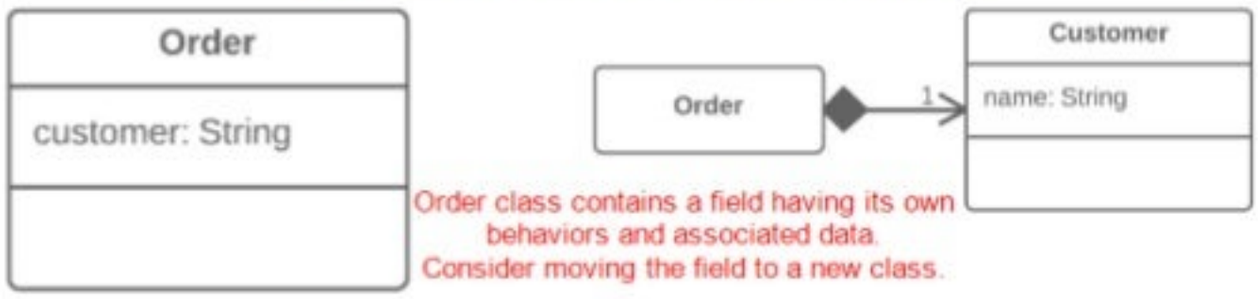
Refer to many indentical instances of a class as a single reference object.

Change Reference to Value



If a reference object is too small and there is no point to keen only one instance of it to safe resource, just turn it to be value type

Replace Data Value with Object



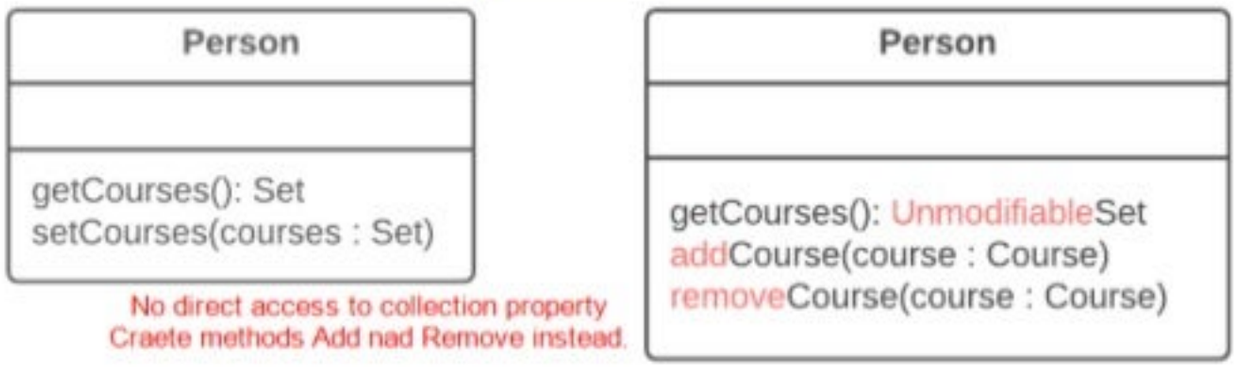
Replace Array with Object

```
string[] row = new string[2];
row[0] = "Liverpool";
row[1] = "15";
```

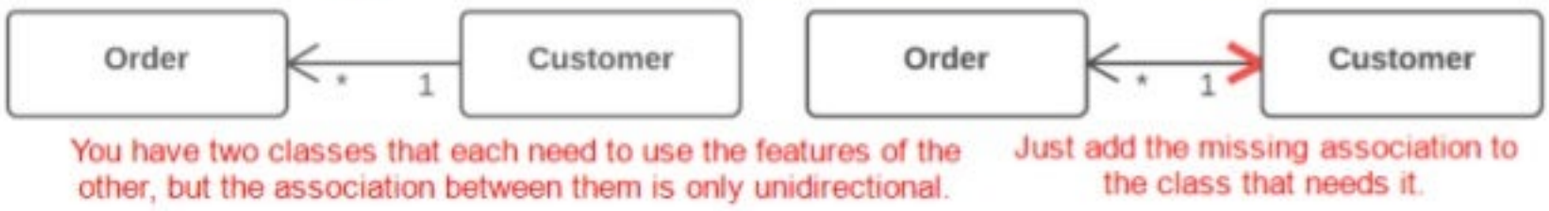
Easier than remembering which index represent which info.

```
Performance row = new Performance();
row.SetName("Liverpool");
row.SetWins("15");
```

Encapsulate Collection



Change Unidirectional Association to Bidirectional



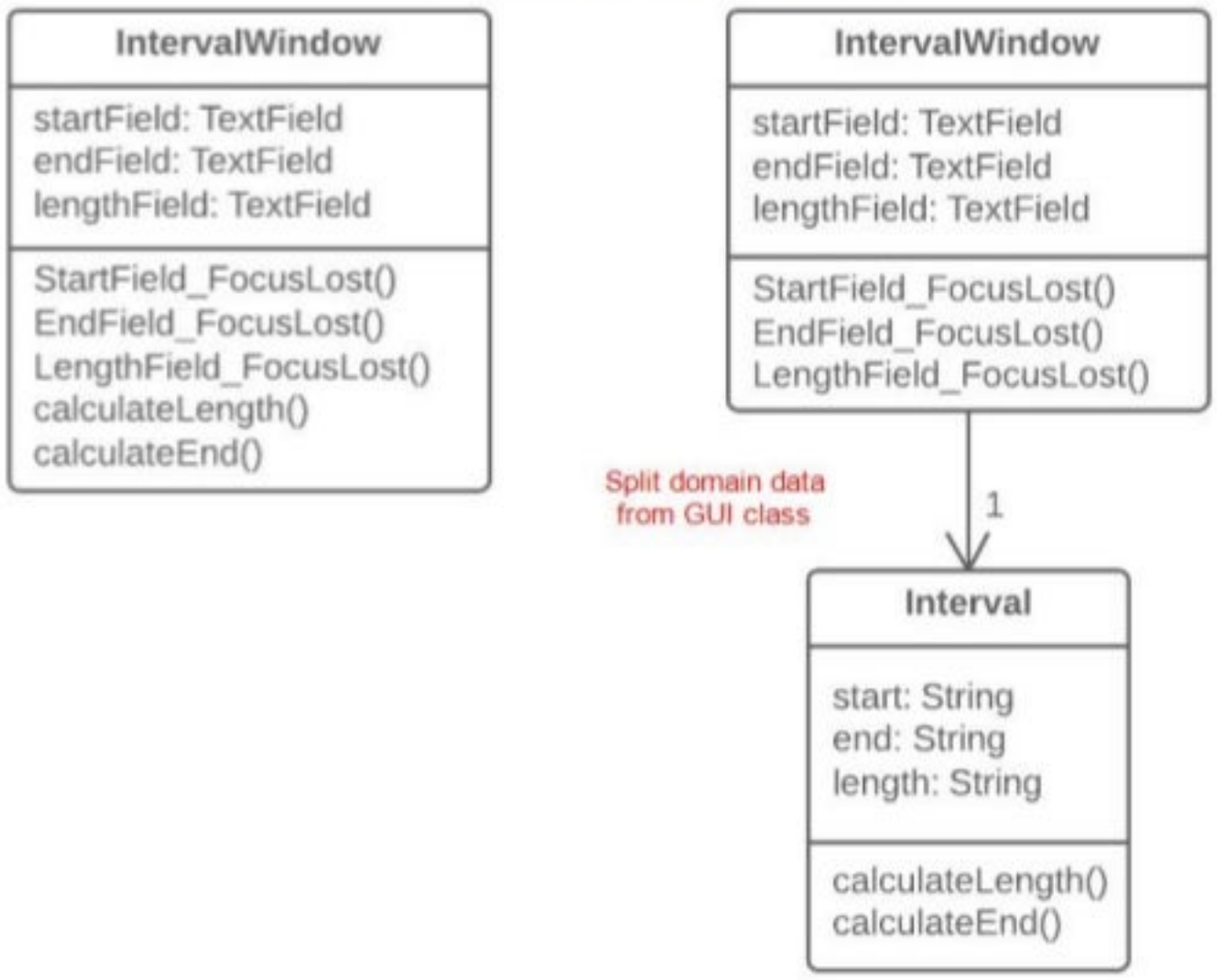
Change Bidirectional Association to Unidirectional

Replace Magic Number with Symbolic Constant

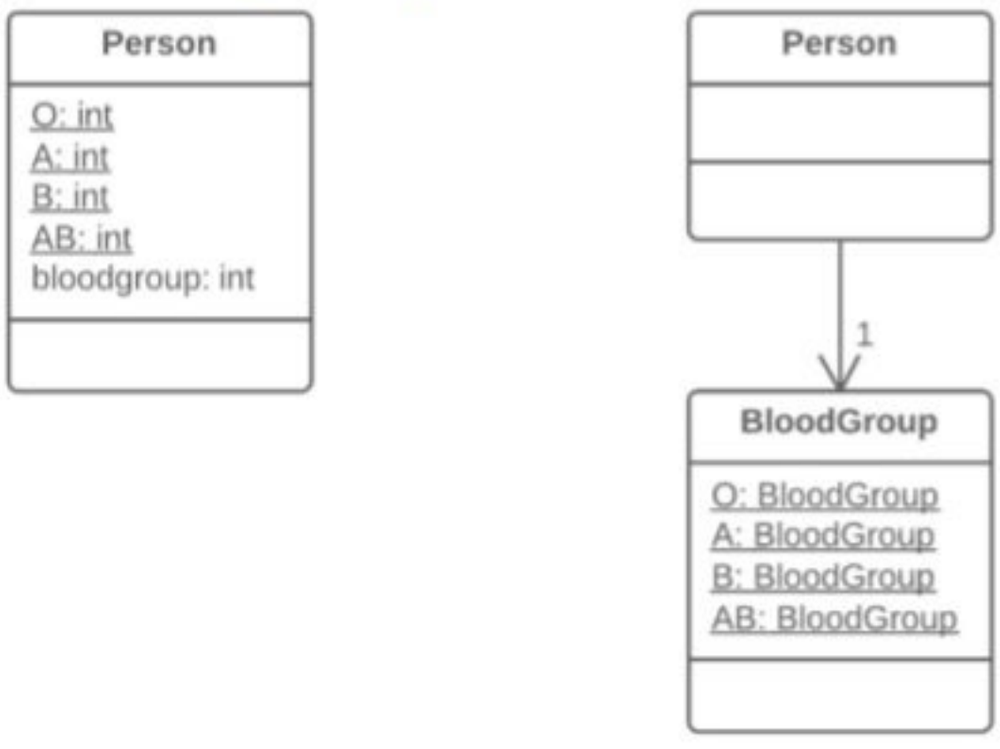
```
double PotentialEnergy(double mass, double height)
{
  return mass * height * 9.81;
}
```

```
const double GRAVITATIONAL_CONSTANT = 9.81;

double PotentialEnergy(double mass, double height)
{
  return mass * height * GRAVITATIONAL_CONSTANT;
}
```

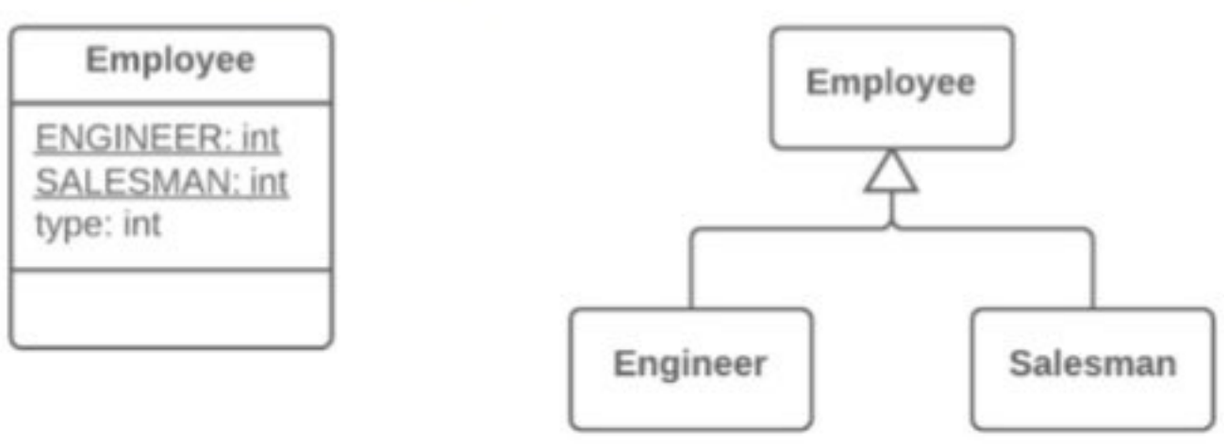
Duplicate Observed Data



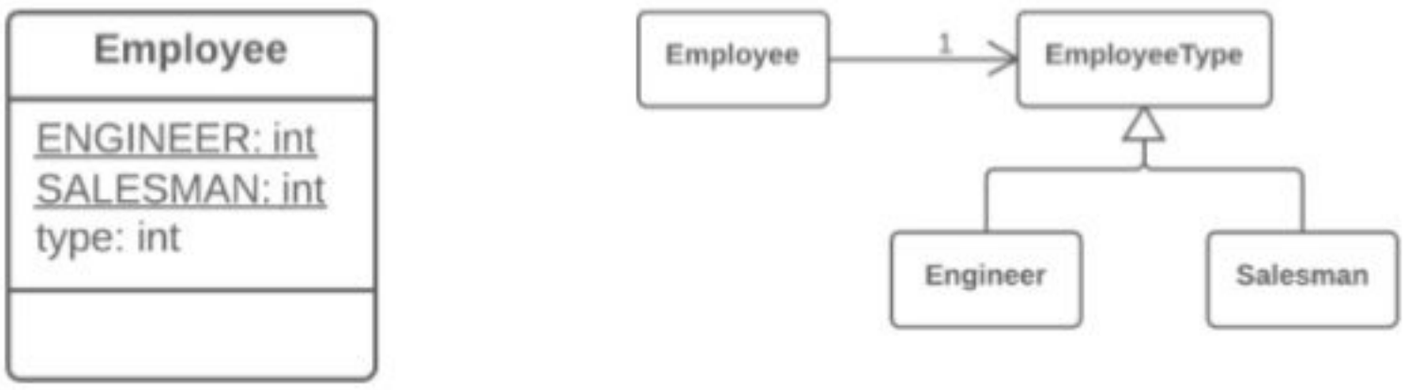
Replace Type Code with Class



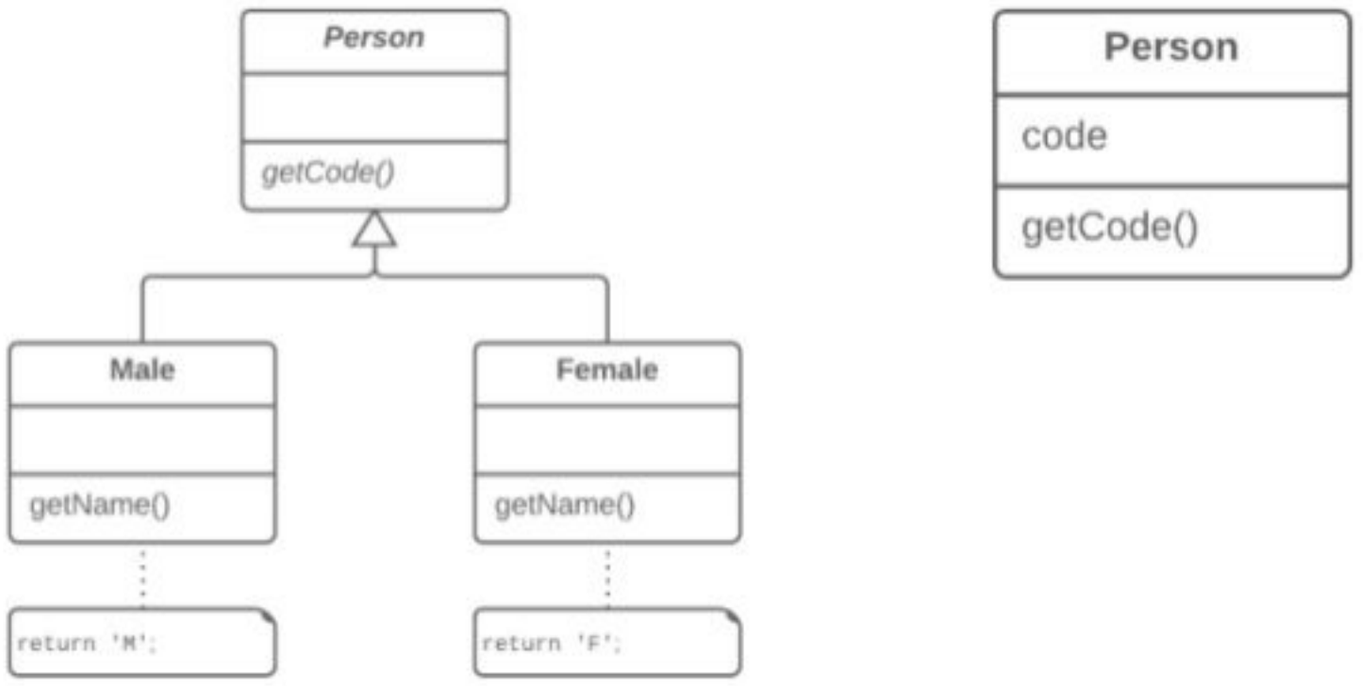
Replace Type Code with Subclasses



Replace Type Code with State/Strategy



Replace Subclass with Fields



Simplifying Conditional Expressions

Decompose Conditional

```
if (date < SUMMER_START || date > SUMMER_END)
{
  charge = quantity * winterRate + winterServiceCharge;
}
else
{
  charge = quantity * summerRate;
}
```

```
if (isSummer(date))
{
  charge = SummerCharge(quantity);
}
else
{
  charge = WinterCharge(quantity);
}
```

Consolidate Conditional Expression

```
double DisabilityAmount()
{
  if (seniority < 2)
  {
    return 0;
  }
  if (monthsDisabled > 12)
  {
    return 0;
  }
  if (isPartTime)
  {
    return 0;
  }
  // Compute the disability amount.
  // ...
}
```

```
double DisabilityAmount()
{
  if (IsNotEligibleForDisability())
  {
    return 0;
  }
  // Compute the disability amount.
  // ...
}
```

Consolidate Duplicate Conditional Fragments

```
if (IsSpecialDeal())
{
  total = price * 0.95;
  Send();
}
else
{
  total = price * 0.98;
  Send();
}
```

```
if (IsSpecialDeal())
{
  total = price * 0.95;
}
else
{
  total = price * 0.98;
}
Send();
```

Remove Control Flag

Problem

You have a boolean variable that acts as a control flag for multiple boolean expressions.

Solution

Instead of the variable, use `break`, `continue` and `return`.

Replace Nested Conditional with Guard Clauses

```
public double GetPayAmount()
{
  double result;

  if (isDead)
  {
    result = DeadAmount();
  }
  else
  {
    if (isSeparated)
    {
      result = SeparatedAmount();
    }
    else
    {
      if (isRetired)
      {
        result = RetiredAmount();
      }
      else
      {
        result = NormalPayAmount();
      }
    }
  }

  return result;
}
```

```
public double GetPayAmount()
{
  if (isDead)
  {
    return DeadAmount();
  }
  if (isSeparated)
  {
    return SeparatedAmount();
  }
  if (isRetired)
  {
    return RetiredAmount();
  }
  return NormalPayAmount();
}
```

Introduce Assertion

```
double GetExpenseLimit()
{
  // Should have either expense limit or
  // a primary project.
  return (expenseLimit != NULL_EXPENSE) ?
    expenseLimit :
    primaryProject.GetMemberExpenseLimit();
}
```

If no assertion, it will trown error when expenseLimit == NULL_EXPENSE and primaryProject == null

```
double GetExpenseLimit()
{
  Assert.IsTrue(expenseLimit != NULL_EXPENSE || primaryProject != null);

  return (expenseLimit != NULL_EXPENSE) ?
    expenseLimit:
    primaryProject.GetMemberExpenseLimit();
}
```

Replace Conditional with Polymorphism

```
public class Bird
{
  // ...
  public double GetSpeed()
  {
    switch (type)
    {
      case EUROPEAN:
        return GetBaseSpeed();
      case AFRICAN:
        return GetBaseSpeed() - GetLoadFactor() * numberOfCocon
      case NORWEGIAN_BLUE:
        return isNailed ? 0 : GetBaseSpeed(voltage);
      default:
        throw new Exception("Should be unreachable");
    }
  }
}
```

```
public abstract class Bird
{
  // ...
  public abstract double GetSpeed();
}

class European: Bird
{
  public override double GetSpeed()
  {
    return GetBaseSpeed();
  }
}
class African: Bird
{
  public override double GetSpeed()
  {
    return GetBaseSpeed() - GetLoadFactor() * numberOfCoconuts;
  }
}
class NorwegianBlue: Bird
{
  public override double GetSpeed()
  {
    return isNailed ? 0 : GetBaseSpeed(voltage);
  }
}

// Somewhere in client code
speed = bird.GetSpeed();
```

Different kind of bird have different implementation of GetSpeed()

Introduce Null Object

```
if (customer == null)
{
  plan = BillingPlan.Basic();
}
else
{
  plan = customer.GetPlan();
}
```

```
public sealed class NullCustomer: Customer
{
  public override bool IsNull
  {
    get { return true; }
  }

  public override Plan GetPlan()
  {
    return new NullPlan();
  }
  // Some other NULL functionality.
}

// Replace null values with Null-object.
customer = order.customer ?? new NullCustomer();

// Use Null-object as if it's normal subclass.
plan = customer.GetPlan();
```

Simplifying Method Calls

Rename Method

Add Parameter

Remove Unused Parameter

Separate Query from Modifier

Parameterize Method

Preserve Whole Object

```
int low = daysTempRange.GetLow();
int high = daysTempRange.GetHigh();
bool withinPlan = plan.WithinRange(low, high);
```

```
bool withinPlan = plan.WithinRange(daysTempRange);
```

Replace Parameter with Method Call

```
int basePrice = quantity * itemPrice;
double seasonDiscount = this.GetSeasonalDiscount();
double fees = this.GetFees();
double finalPrice = DiscountedPrice(basePrice, seasonDiscount, fees);
```

```
int basePrice = quantity * itemPrice;
double finalPrice = DiscountedPrice(basePrice);
```

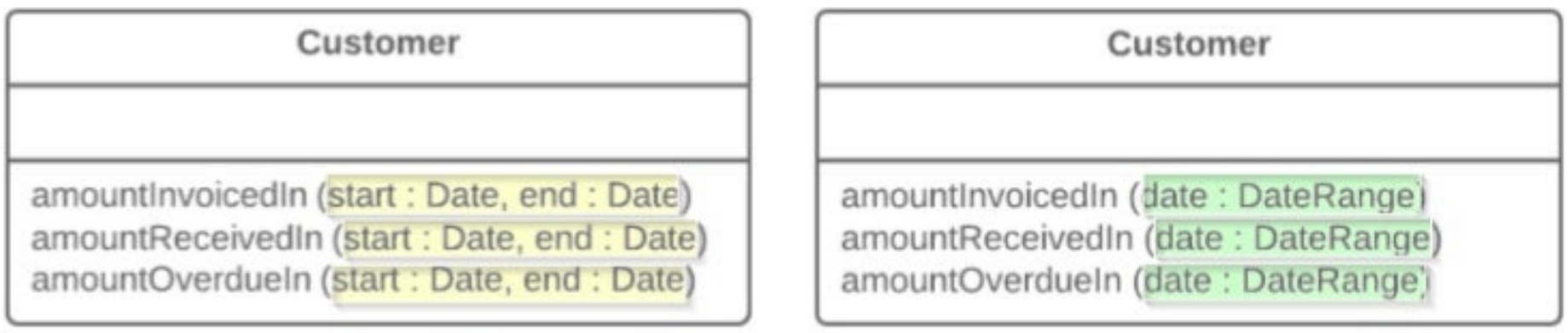
Get seasonDiscount and fees inside the DiscountedPrice()

Replace Parameter with Explicit Methods

```
void SetValue(string name, int value)
{
  if (name.Equals("height"))
  {
    height = value;
    return;
  }
  if (name.Equals("width"))
  {
    width = value;
    return;
  }
  Assert.Fail();
}
```

```
void SetHeight(int arg)
{
  height = arg;
}
void SetWidth(int arg)
{
  width = arg;
}
```

Introduce Parameter Object



Remove Setting Method for Immutable Fields

Hide Method that should not be Public

Replace Constructor with Factory Method

```
public class Employee
{
  public Employee(int type)
  {
    this.type = type;
  }
  // ...
}
```

```
public class Employee
{
  public static Employee Create(int type)
  {
    employee = new Employee(type);
    // Do some heavy lifting.
    return employee;
  }
  // ...
}
```

Replace Error Code with Exception

```
int Withdraw(int amount)
{
  if (amount > _balance)
  {
    return -1;
  }
  else
  {
    balance -= amount;
    return 0;
  }
}
```

```
///<exception cref="BalanceException">Thrown when amount > _
void Withdraw(int amount)
{
  if (amount > _balance)
  {
    throw new BalanceException();
  }
  balance -= amount;
}
```

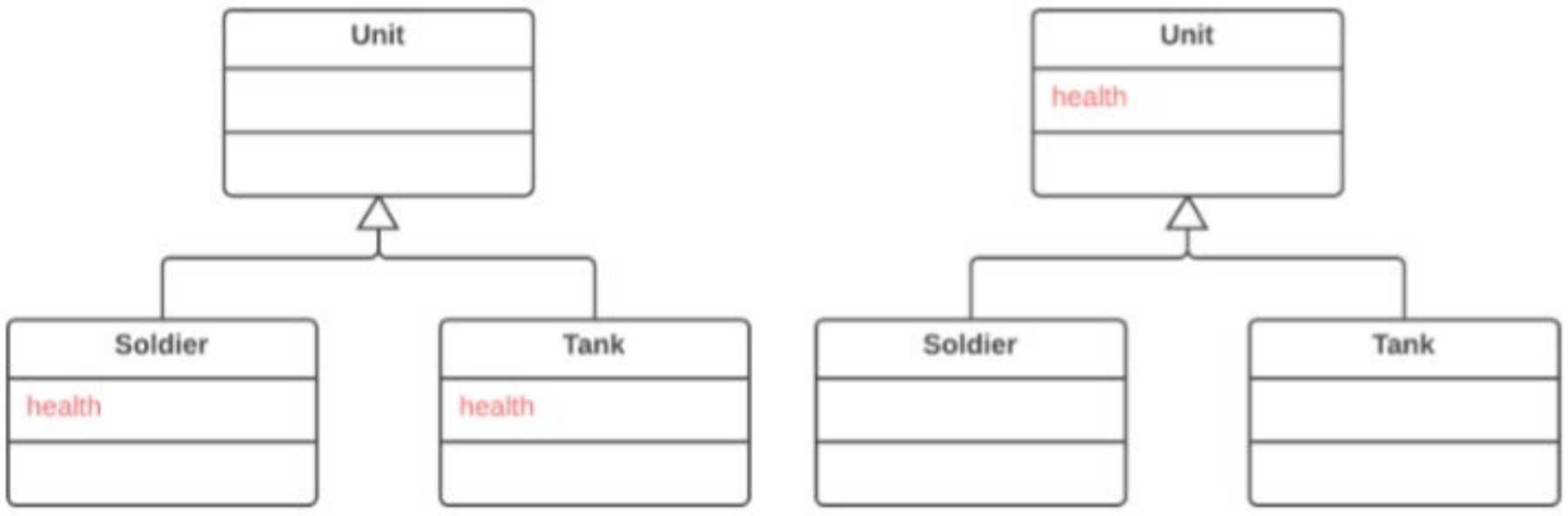
Replace Exception with Test

```
double GetValueForPeriod(int periodNumber)
{
  try
  {
    return values[periodNumber];
  }
  catch (IndexOutOfRangeException e)
  {
    return 0;
  }
}
```

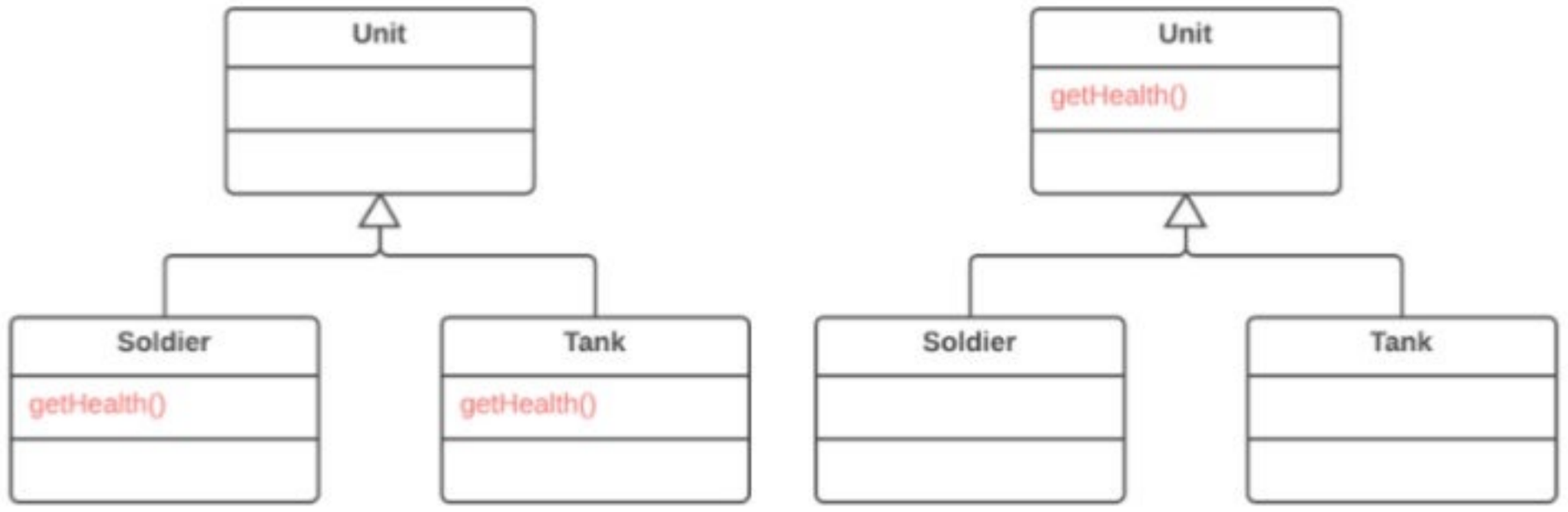
```
double GetValueForPeriod(int periodNumber)
{
  if (periodNumber >= values.Length)
  {
    return 0;
  }
  return values[periodNumber];
}
```

Dealing with Generalization

Pull Up Field



Pull Up Method

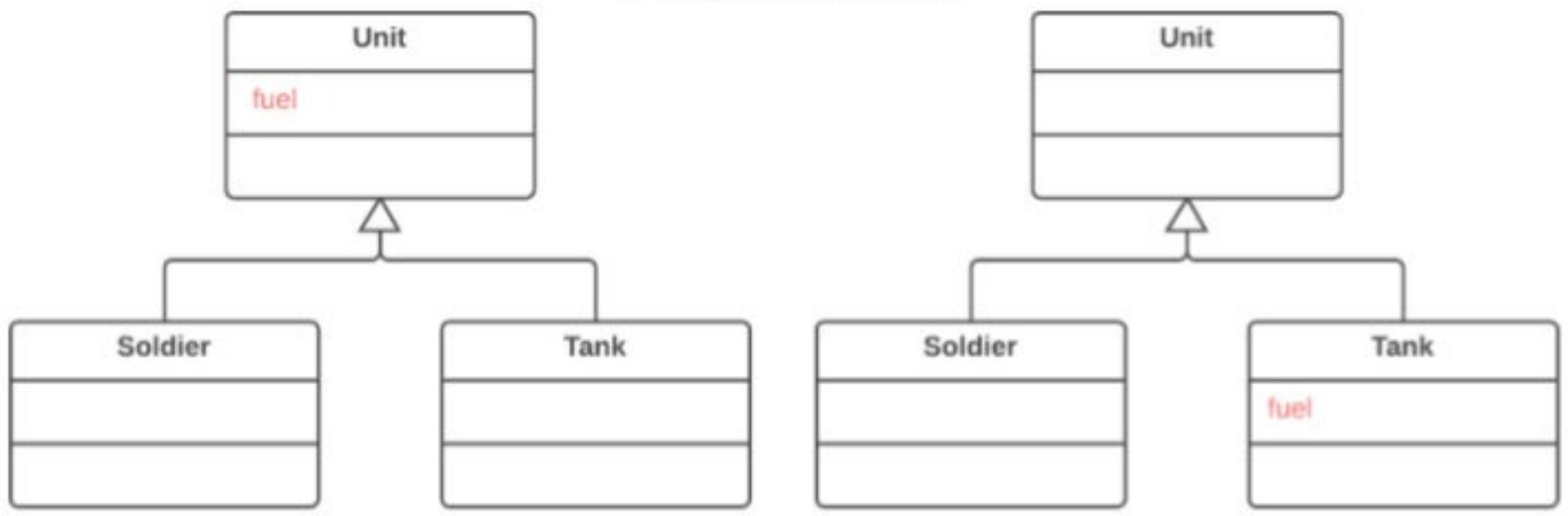


Pull Up Constructor Body

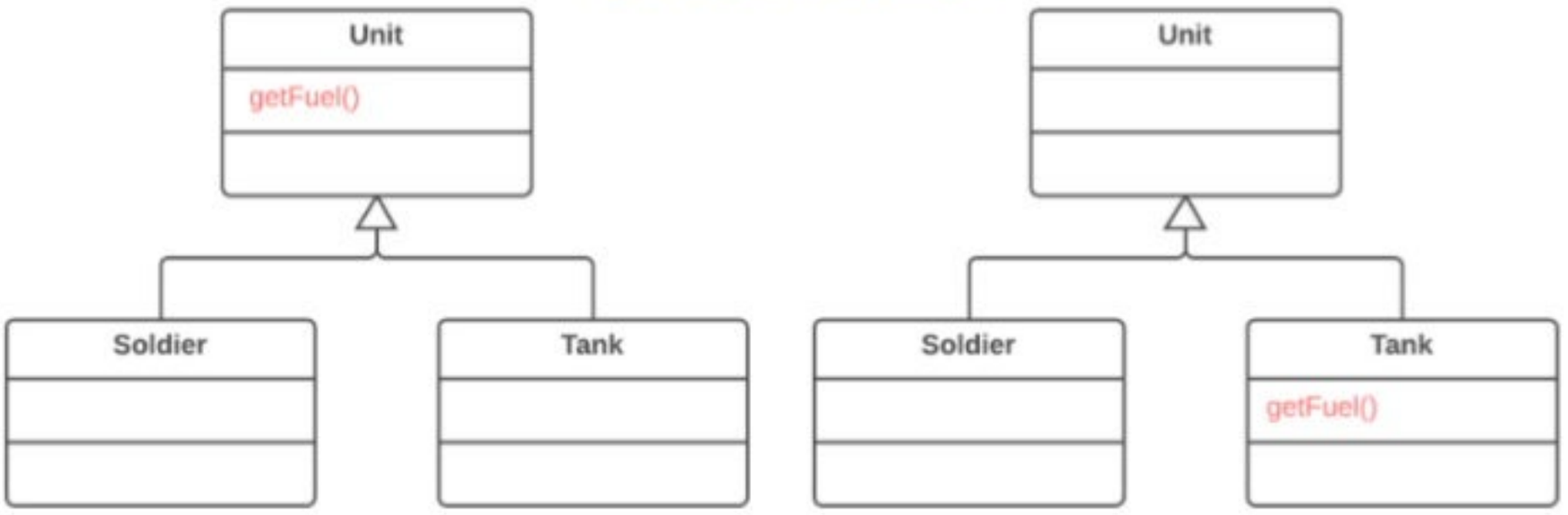
```
public class Manager: Employee
{
  public Manager(string name, string id, int grade)
  {
    this.name = name;
    this.id = id;
    this.grade = grade;
  }
  // ...
}
```

```
public class Manager: Employee
{
  public Manager(string name, string id, int grade): base(name, id)
  {
    this.grade = grade;
  }
  // ...
}
```

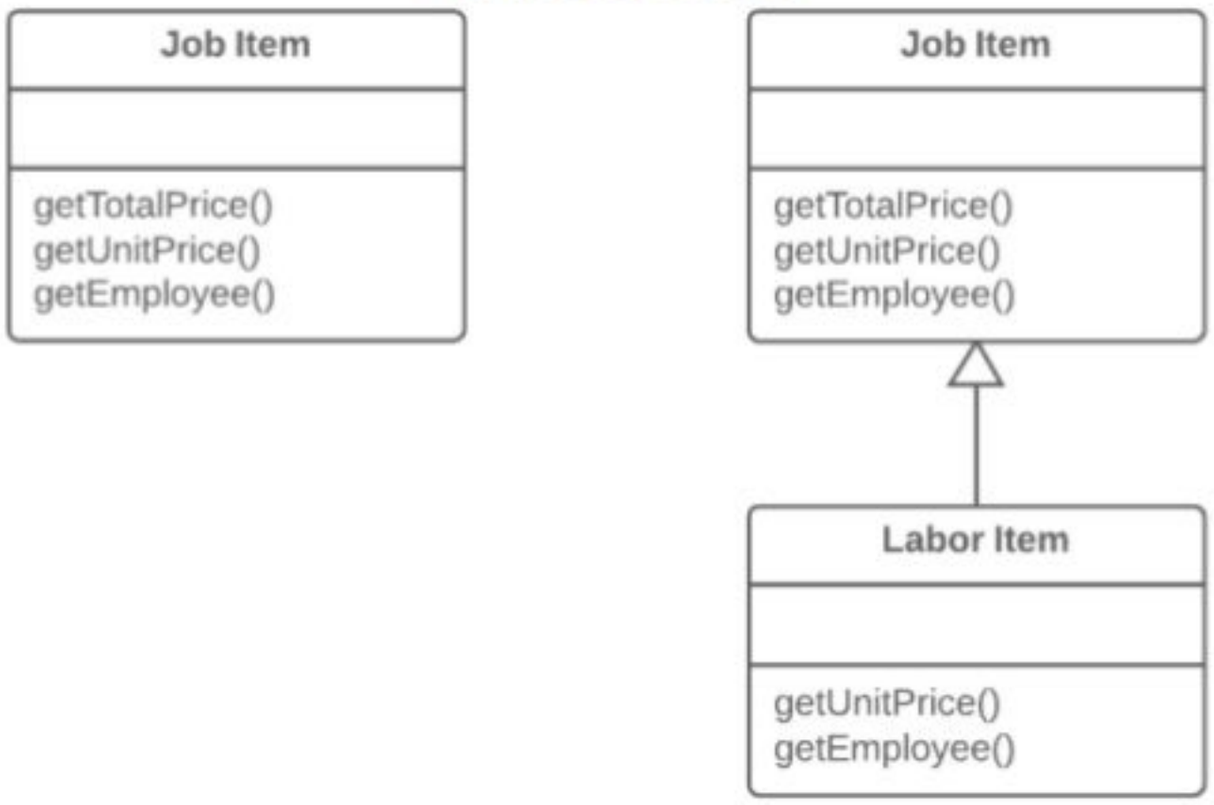
Push Down Field



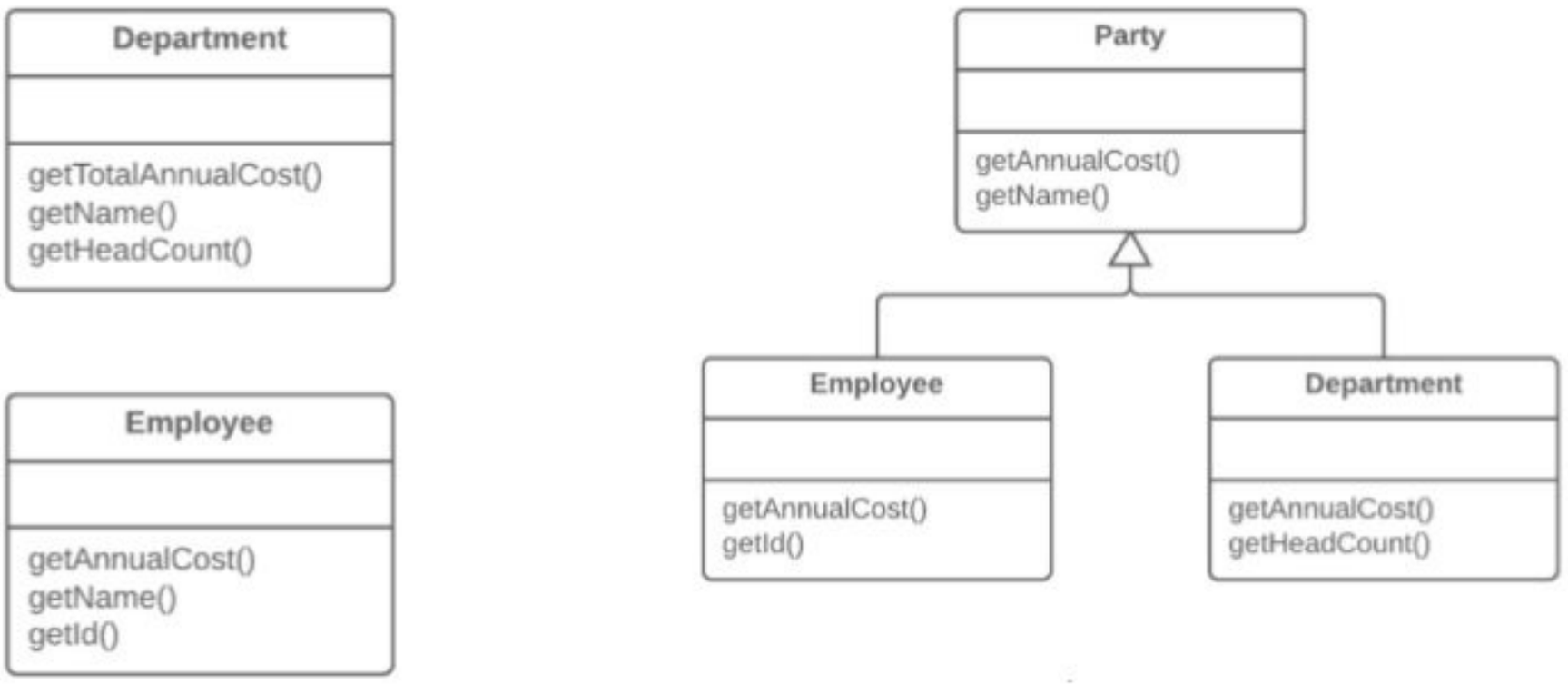
Push Down Method



Extract Subclass

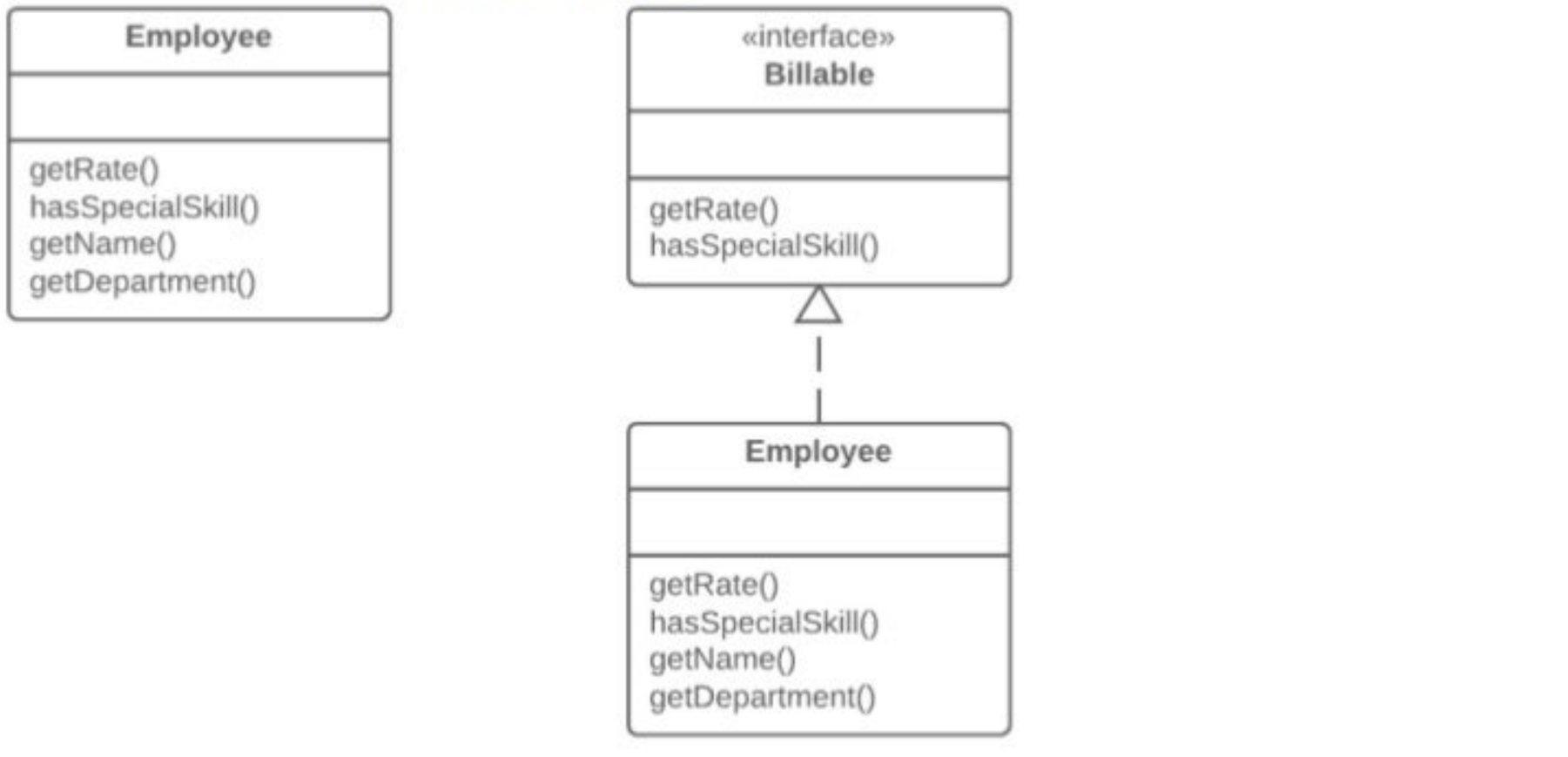


Extract Superclass

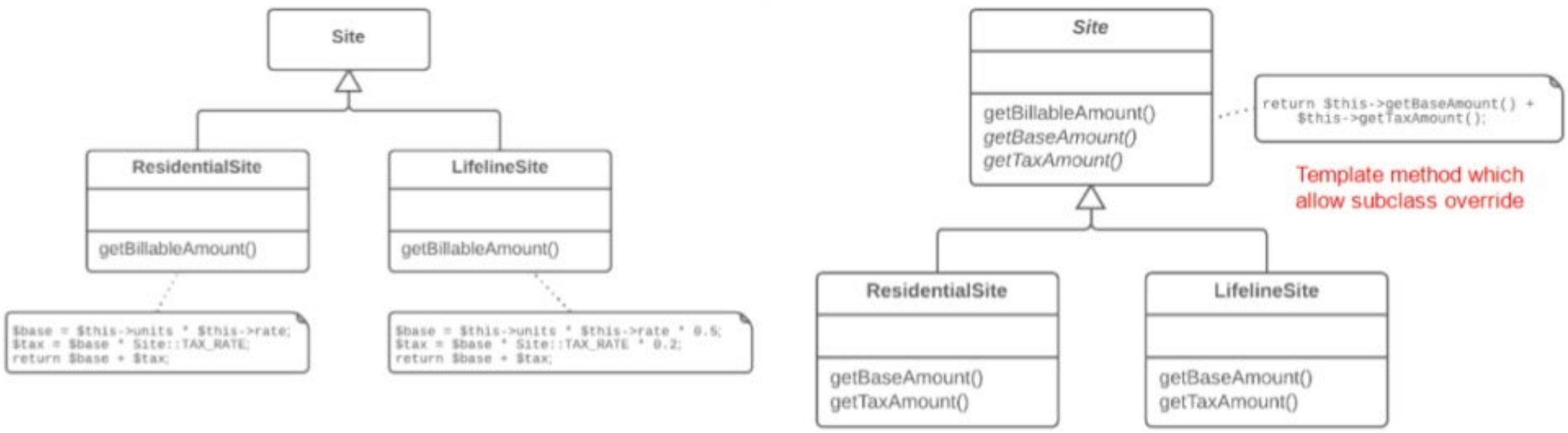


Collapse Hierarchy

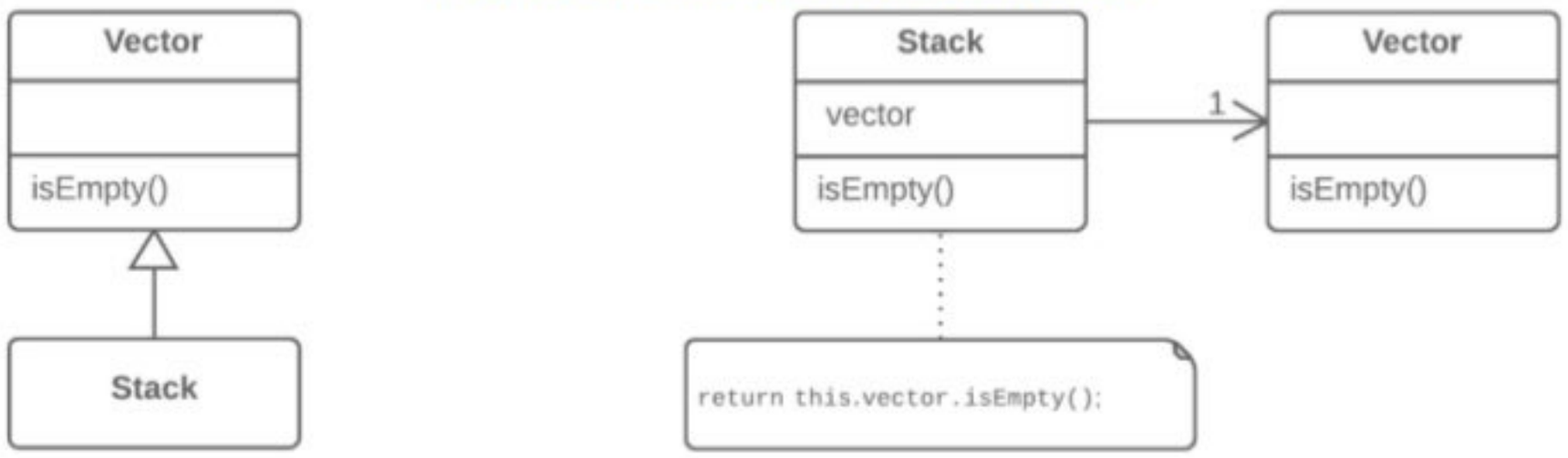
Extract Interface



Form Template Method



Replace Inheritance with Delegation



Replace Delegation with Inheritance