Técnicas de Refactorización

Semana 11

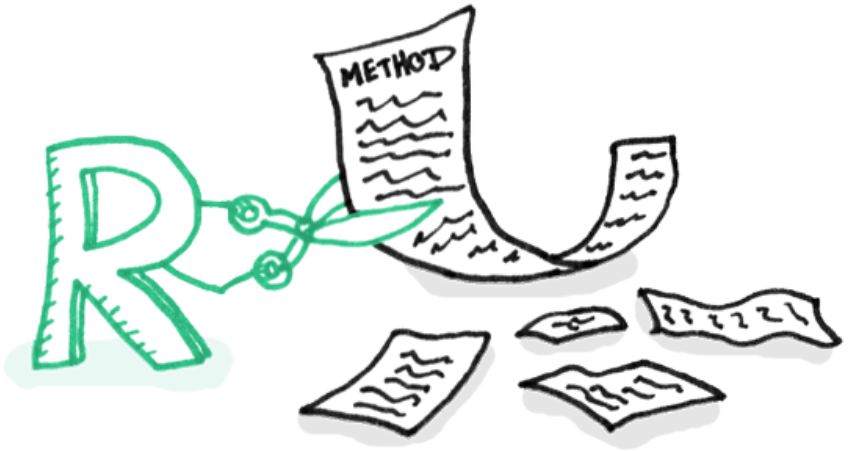
Agenda

- Composición de Métodos
- Simplificación de Sentencias Condicionales
- Manejo de Generalización
- Mover características entre Objetos

Composición de Métodos

Composición de Métodos

- Métodos extremadamente largos son la razón de muchos problemas de un programa.
  - Provocan que éstos sean difíciles de mantener y modificar.
- Las técnicas de refactorización de este grupo están enfocadas a remover código duplicado y preparar el código para modificaciones futuras.



Técnicas

- Extraer Método
- Métodos Inline
- Extraer Variables
- Inline Temp
- Redefinir Métodos como una nueva Clase
- Sustituir Algoritmo

Extraer Métodos

Problema

Existen fragmentos de código que podrían "Agruparse"

Solución

Mover dicho código a un nuevo método. Invocarlo cuando se lo necesite

```
void printOwing() {
  printBanner();

  //print details
  System.out.println("name: "
                     + name);
  System.out.println("amount: "
                     + getOutstanding());
}
```

```
void printOwing() {
  printBanner();
  printDetails(getOutstanding());
}

void printDetails(double outstanding) {
  System.out.println("name: "
                     + name);
  System.out.println("amount: "
                     + outstanding);
}
```

Métodos Inline

Problema

Cuando la implementación de un método es corta y resulta más obvia que su nombre.

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

Métodos Inline

Solución

Reemplazar la llamada al método con su implementación

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

Extraer Variables

Problema

Existe una expresión que es difícil de Entender

```
if ((platform.toUpperCase().indexOf("MAC") > -1) &&
    (browser.toUpperCase().indexOf("IE") > -1) &&
     wasInitialized() && resize > 0 )
  {
    // do something
  }
}
```

Extraer Variable

Solución

Colocar el resultado de una expresión o sus resultados parciales en variables , de tal forma que su evaluación sea más entendible.

```
void renderBanner() {
  final boolean isMacOs = platform.toUpperCase().indexOf("MAC") > -1;
  final boolean isIE = browser.toUpperCase().indexOf("IE") > -1;
  final boolean wasResized = resize > 0;

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

Inline TEMP

Problema

Se usa una variable temporal para asignar el valor de una expresión simple y no se la usa más.

```
double hasDiscount(Order order) {
  double basePrice = order.basePrice();
  return (basePrice > 1000);
}
```

Solución

Reemplazar la variable con la expresión.

```
double hasDiscount(Order order) {
  return (order.basePrice() > 1000);
}
```

Redefinir Métodos como una nueva Clase

Problema

Existe un método largo, en el cual las variables locales están tan entrelazadas que es difícil aplicar 'Extraer Método'

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

Redefinir Métodos como una nueva Clase

Solución

Transformar el método en una clase, de tal forma que las variables locales se convierten en los atributos. Ahora el método puede ser separado en varios métodos dentro de la misma clase *(Extraer Métodos)*

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

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

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

  public double compute() {
    // long computation.
    //...
  }
}
```

Sustituir Algoritmo

Problema

Se desea reemplazar un algoritmo existente con uno nuevo.

```
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 "";
}
```

Sustituir Algoritmo

¿Cómo proceder?

- Asegurarse de tener una versión simplificada.
- Mover código a otros métodos usando Extraer Método.
- Crear el nuevo algoritmo en un nuevo método. Reemplazar el viejo con el nuevo.
- **Hacer pruebas.**
- Cuando las pruebas hayan sido exitosas, borrar el algoritmo viejo, no mantenerlo en el código.

Sustituir Algoritmo

Solución

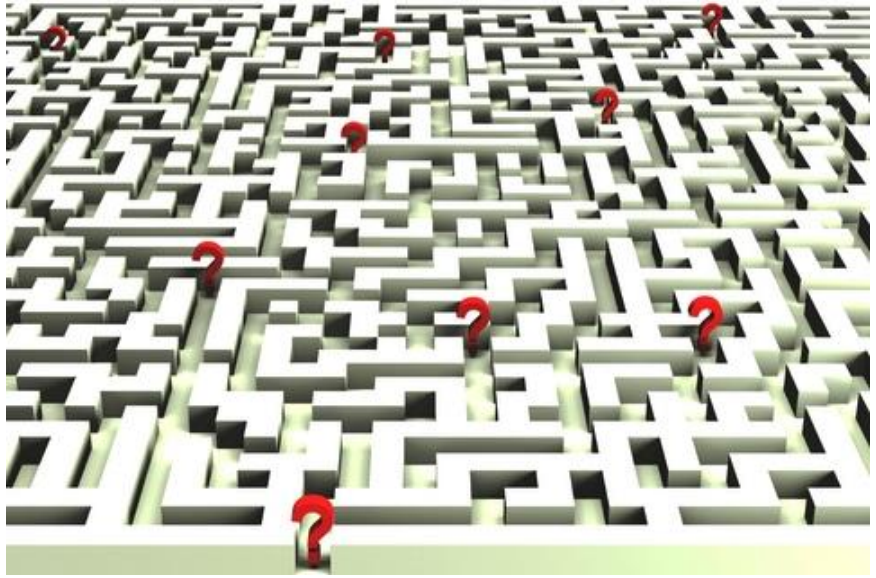
Reemplazar el cuerpo del método con el nuevo algoritmo

```
String foundPerson(String[] people){
  List candidates =
    Arrays.asList(new String[] {"Don", "John", "Kent"});
  for (int i=0; i < people.length; i++) {
    if (candidates.contains(people[i])) {
      return people[i];
    }
  }
  return "";
}
```

Simplificación de Sentencias Condicionales

Simplificación de sentencias Condicionales

- Las estructuras condicionales pueden crecer y volverse complejas con el tiempo.
- Existen varias técnicas de Refactoring para combatir este mal olor.



Simplificación de condicionales

- Consolidar fragmentos duplicados
- Descomponer Condicional
- Consolidar expresiones
- Reemplazar Condicional con Polimorfismo
- Reemplazar condicionales anidados con banderas
- Remove banderas
- Agregar Null Object
- Agregar Aserciones

Consolidar Fragmentos Duplicados

Problema

El mismo código puede encontrarse en varias ramas del condicional

```
if (isSpecialDeal()) {
  total = price * 0.95;
  send();
}
else {
  total = price * 0.98;
  send();
}
```

Solución

Mover el código afuera del condicional

```
if (isSpecialDeal()) {
  total = price * 0.95;
}
else {
  total = price * 0.98;
}
send();
```

Descomponer Condicional

Problema

Existen condicionales complejos.

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

Solución

Crear una función booleana.

```
if (notSummer(date)) {
  charge = winterCharge(quantity);
}
else {
  charge = summerCharge(quantity);
}
```

Consolidar Expresión

Problema

Existen varias condiciones que llevan a un mismo resultado o acción.

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

Solución

Consolidar todas esas condiciones en una sola expresión.

```
double disabilityAmount() {
  if (isNotEligableForDisability()) {
    return 0;
  }
  // compute the disability amount
  //...
}
```

Reemplazar Condicionales anidados con banderas

Problema

```
if () {
    if () {
        do {
            if () {
                if () {
                    if () {
                        ...
                    }
                }
                ...
            }
            ...
        }
        while ();
        ...
    }
    else {
        ...
    }
}
```

Reemplazar Condicionales anidados con banderas

Solución

Aislar los casos extremos y revisiones especiales , colocarlos antes de los chequeos principales.

Idealmente, deberíamos tener una secuencia “plana” de condicionales, una tras otra

```
public double getPayAmount() {
  if (isDead){
    return deadAmount();
  }
  if (isSeparated){
    return separatedAmount();
  }
  if (isRetired){
    return retiredAmount();
  }
  return normalPayAmount();
}
```

Reemplazar Switch con Polimorfismo

Problema

Existe una estructura condicional que realiza varias acciones, dependiendo del tipo de Objeto.

```
class Bird {
  //...
  double getSpeed() {
    switch (type) {
      case EUROPEAN:
        return getBaseSpeed();
      case AFRICAN:
        return getBaseSpeed() - getLoadFactor() * numberOfCoconuts;
      case NORWEGIAN_BLUE:
        return (isNailed) ? 0 : getBaseSpeed(voltage);
    }
    throw new RuntimeException("Should be unreachable");
  }
}
```

Reemplazar Switch con Polimorfismo

Solución

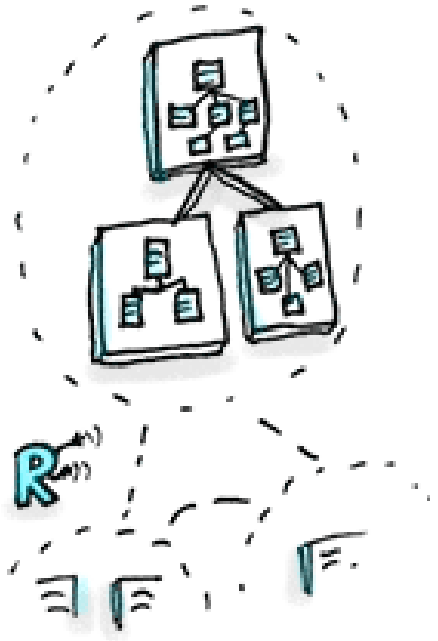
Hacer uso de interfaces y subclases

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

class European extends Bird {
  double getSpeed() {
    return getBaseSpeed();
  }
}
class African extends Bird {
  double getSpeed() {
    return getBaseSpeed() - getLoadFactor() * numberOfCoconuts;
  }
}
class NorwegianBlue extends Bird {
  double getSpeed() {
    return (isNailed) ? 0 : getBaseSpeed(voltage);
  }
}

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

Manejo de Generalización

Manejo de Generalización

- Las técnicas pertenecientes a este grupo están relacionadas a mover funcionalidades en una jerarquía, creando nuevas clases e interfaces y reemplazando delegación con herencia.



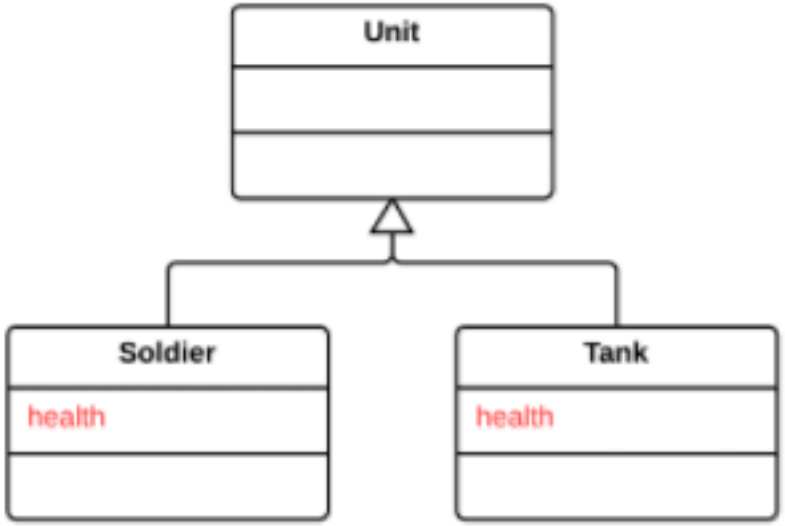
Técnicas

- Subir Campo / Método
- Bajar Campo / Método
- Extraer Subclase
- Extraer Superclase
- Colapsar Jerarquía
- Form Template Method
- Reemplazar Herencia con Delegación
- Reemplazar Delegación con Herencia

Subir campo / método

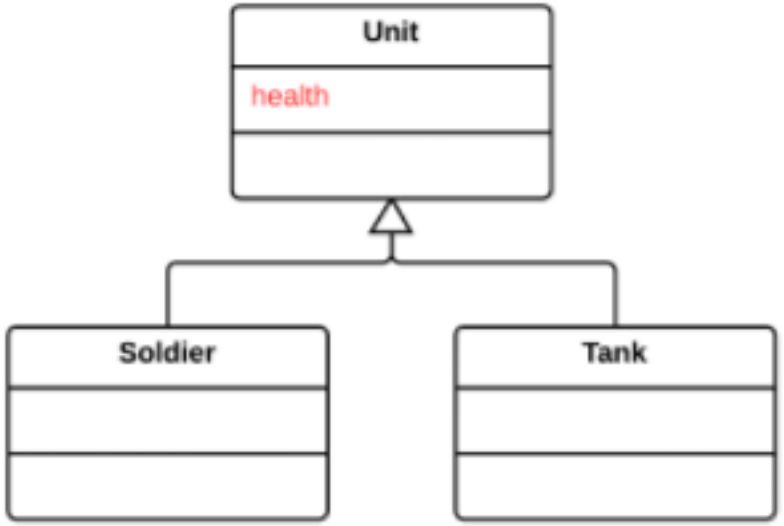
Problema

Dos clases tienen el mismo campo



Solución

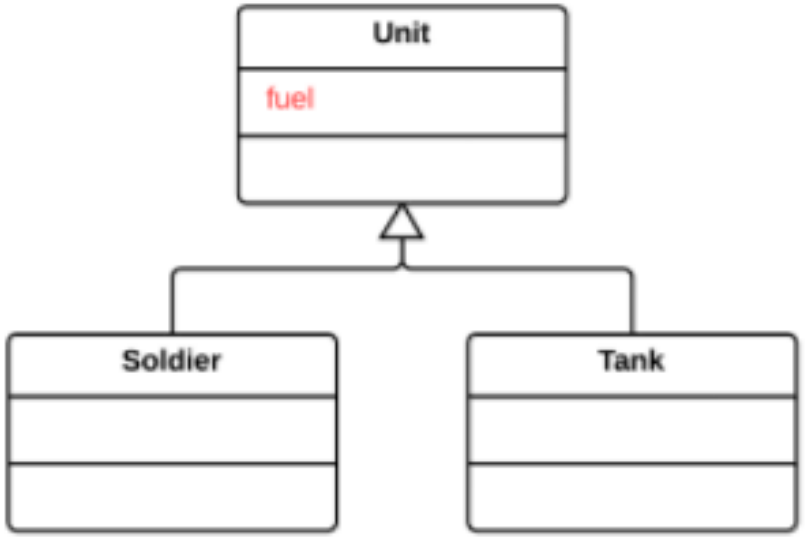
Remover el campo de la subclase y moverlo a la superclase.



Bajar Campo / Método

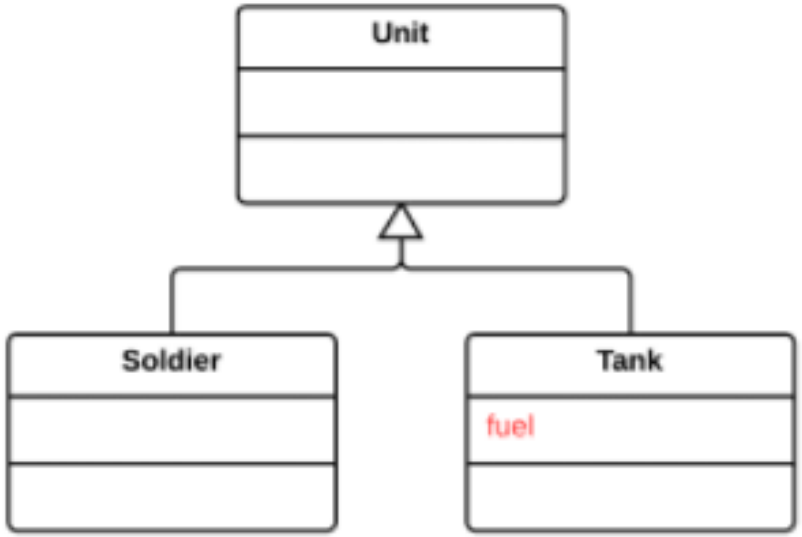
Problema

Existe un campo usado solo en una subclase.



Solución

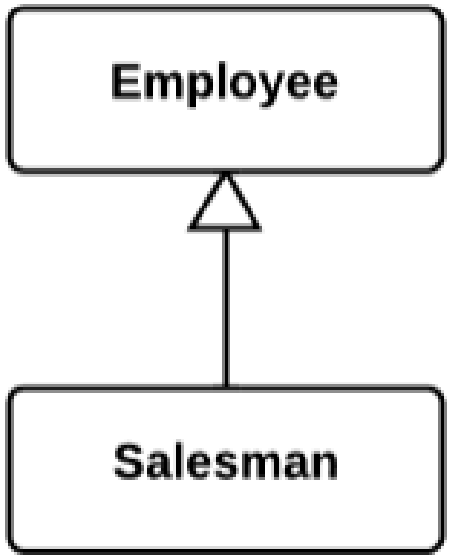
Mover el campo a la subclase correspondiente.



Colapsar Jerarquía

Problema

Existen una subclase que prácticamente es su superclase



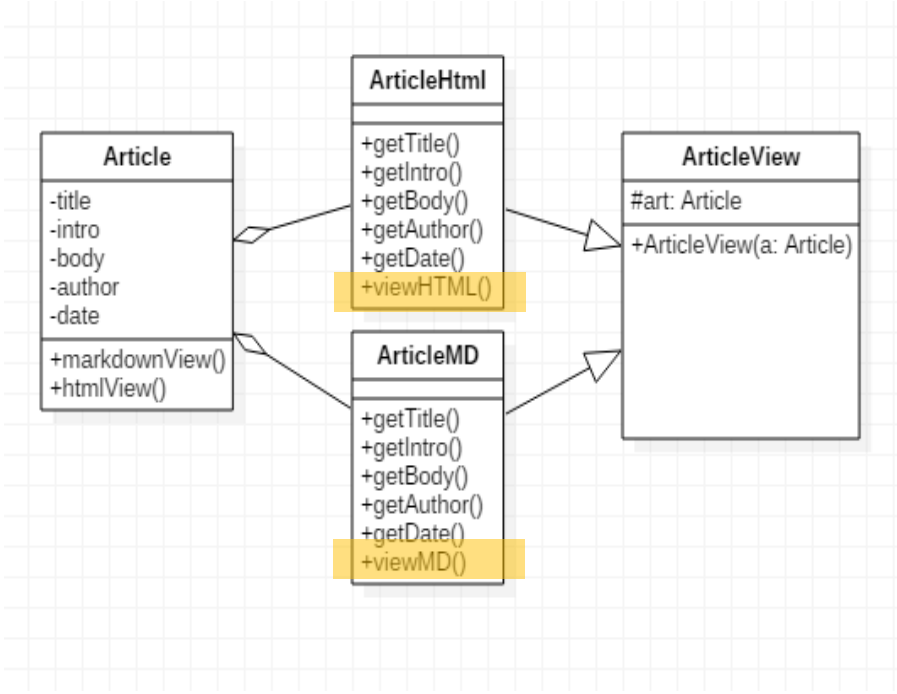
Solución

Unirlas y quedarse con una sola clase.

Form Template Method

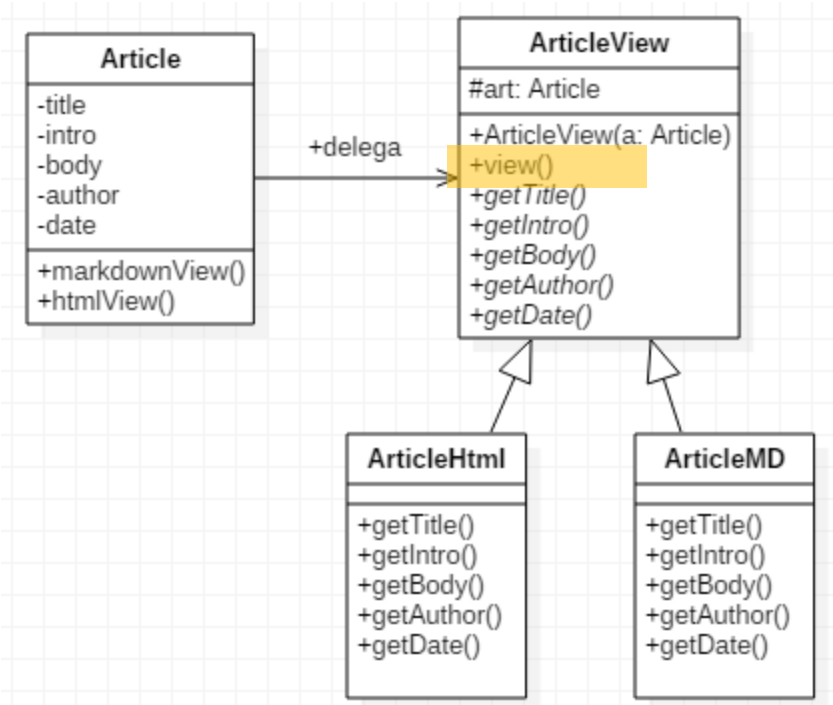
Problema

Existen subclases que implementan algoritmos que tienen estructura similar y se ejecutan en el mismo orden.
Un artículo puede mostrarse como HTML o en Markdown. viewHTML() y viewMD()



Solución

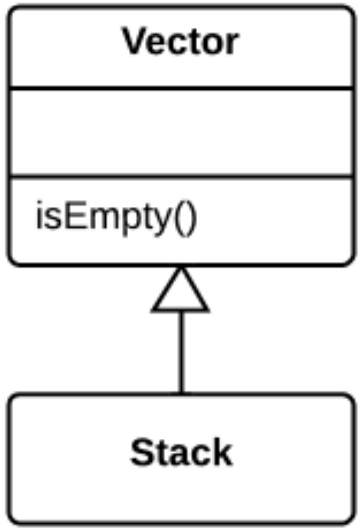
Mover el/los métodos a la superclase con los pasos a ejecutar y dejar la implementación a las subclases



Reemplazar Herencia con Delegación

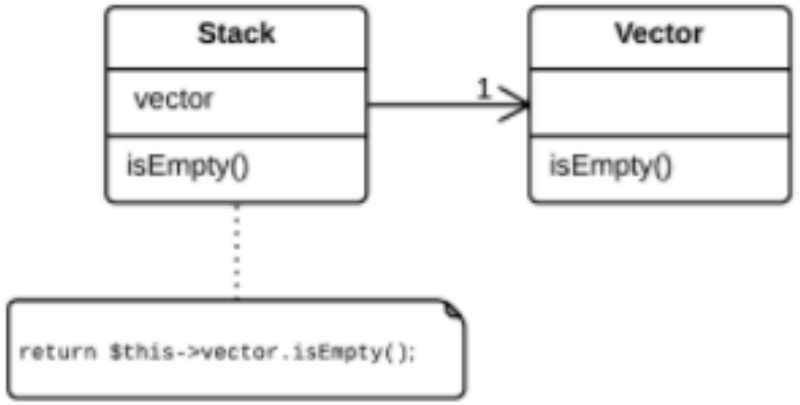
Problema

Existe una subclase que usa solo una porción de los métodos de su superclase.



Solución

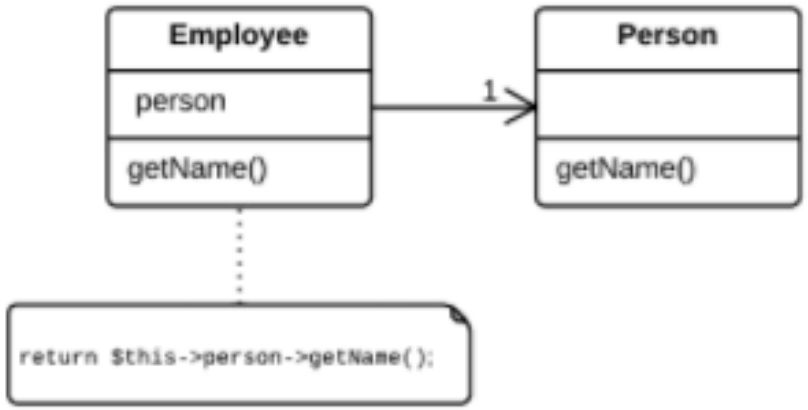
Establecer una relación de asociación entre las clases y delegar el trabajo.



Reemplazar Delegación con Herencia

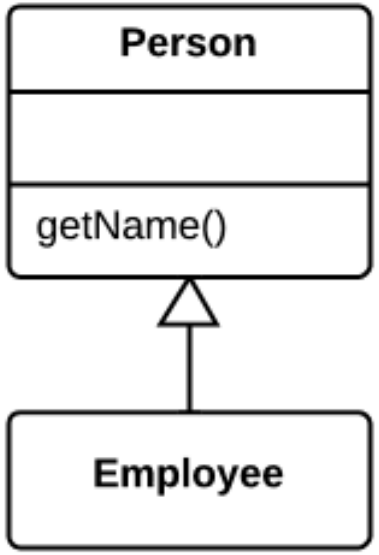
Problema

Existen una clase que contiene métodos que delegan su trabajo a métodos de otra clase.



Solución

Implementar una relación de herencia. Colocar en la superclase los métodos necesarios.



¿Diferencia entre las dos últimas?

- Reemplazar Herencia con Delegación.
- Reemplazar Delegación con Herencia.

Para reemplazar delegación con herencia se debe utilizar todos los métodos y atributos de la clase delegada.

Para reemplazar herencia con delegación es porque se está subutilizando los métodos y atributos de la clase padre.

Mover características entre Objetos

Mover características entre Objetos

- Las técnicas de refactorización de este grupo muestran como mover funcionalidades entre clases de una manera segura.

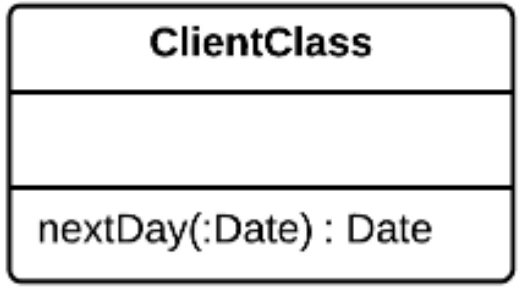
Técnicas

- Mover Método
- Mover Campo
- Extraer Clase
- Fusionar Clase
- Ocultar Delegado
- Introducir Extensión Local

Introducir Extensión Local

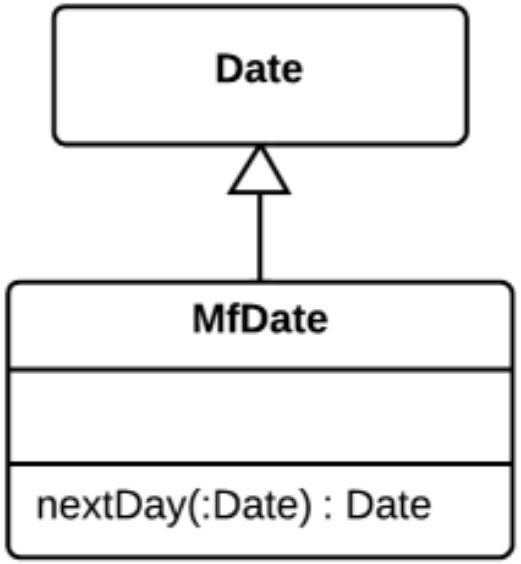
Problema

Una clase no contiene métodos que necesitas y no los puedo agregar a la clase.



Solución

Crear una nueva clase con los métodos que se necesitan y hacer un wrapper o subclase.



Antes de finalizar

Puntos para recordar

- Técnicas de Composición de Métodos
- Técnicas de Simplificación de Sentencias Condicionales
- Técnicas de Manejo de Generalización
- Técnicas de Mover características entre Objetos

Lectura adicional

- Martin Fowler, “Refactoring: Improving the Design of Existing Code”, second edition
- Source Making:
  - https://sourcemaking.com/refactoring
- Refactoring Guru:
  - https://refactoring.guru/

Próxima sesión

- Más técnicas de refactorización