Refactoring

Thierry Sans

with slides from Anya Tafliovich

Composing Methods

Extract Method

```
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);
}
```

Extract Variables

```
double price () {
    // price is base price - quantity discount + shipping
    return _quantity * _itemPrice -
                Math.max(0, _quantity - 500) * _itemPrice * 0.05 +
                Math.min(_quantity * _itemPrice * 0.1, 100.0);
}
```

```
double price () {
  final double basePrice = _quantity * _itemPrice;
  final double discount = Math.max(0,_quantity-500)*_itemPrice*0.05;
  final double shipping = Math.min(basePrice * 0.1, 100.0);
  return basePrice - discount + shipping;
}
```

Organizing Data

Replace magic number with symbolic constant

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

```
static final double GRAVITY = 9.81;

double potentialEnergy(double mass, double height) {
    return mass * GRAVITY * height;
}
```

Encapsulate field

```
private String _name;
public String getName() { return _name; }
public void setName(String name) { _name = name; }
```

Encapsulate collection

```
class Student {
   private Set<Course> _courses;

   Set<Course> getCourses() { return _courses; }

   void setCourses(Set<Course> courses) { _courses = courses; }
}
```

```
class Student {
   private Set<Course> _courses;

   Set<Course> getCourses() {
       return Collections.unmodifiableSet(_courses);
   }
   void setCourses(Set<Course> courses) {
       _courses = new HashSet<>();
       for (Course c : courses) _courses.add(c);
   }
}
```

Simplifying Conditional Expressions

Consolidate conditional expression

```
double  disabilityAmount() {
    if ( _seniority < 2 ) return 0;
    if ( _monthsDisabled > 12 ) return 0;
    if ( _isPartTime ) return 0;
    // now compute disability amount (lots of code here)
 }
```

```
double  disabilityAmount() {
  if  ( _seniority < 2  || _monthsDisabled > 12 || _isPartTime)
      return 0;
  // now compute disability amount (lots of code here)
}
```

Consolidate conditional expression + Extract method

```
double  disabilityAmount() {
  if ( isNotEligibleForDisability() ) return 0;
  // now compute disability amount (lots of code here)
}

boolean isNotEligibleForDisability() {
  return ( _seniority < 2  || _monthsDisabled > 12 || _isPartTime);
}
```

Consolidate duplicate conditional fragments

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

```
if ( isSpecialDeal() )
     total = price * 0.95;
else
     total = price + surcharge * 0.98;
send();
```

Remove control flags

```
boolean found false;
i = 0;
while (!found || i<l.length){
   if (l[i+1] == 'key') found = true;
}
```

```
for (i=0, i<l.length, i++){
   if (l[i+1] == 'key') break;
}
```

Replace nested conditionals

```
double getPayAmt() {
    double result;
    if ( _isSingle ) result = singleAmount();
    else {
        if ( _isSeparated ) result = separatedAmount();
        else {
            if ( _isRetired ) result = retiredAmount();
            else result = normalPayAmount();
        }
    }
    return result;
}
```

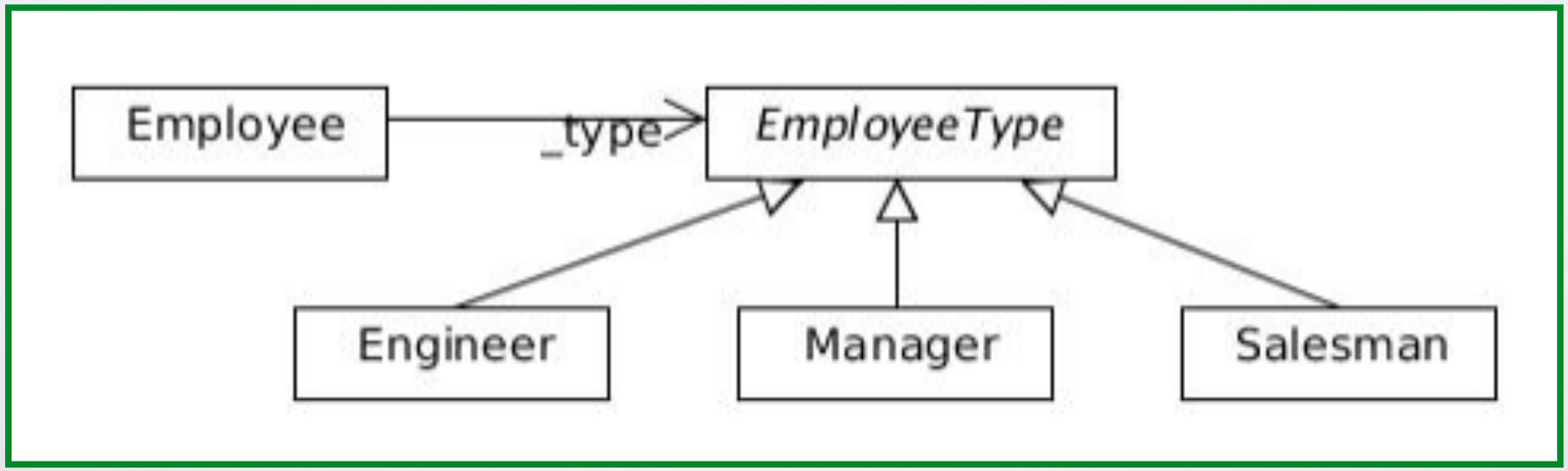
```
double getPayAmt() {
    if ( _isSingle ) return singleAmount();
    if ( _isSeparated ) return separatedAmount();
    if ( _isRetired ) return retiredAmount();
    return normalPayAmount();
}
```

Replace nested conditionals + Extract method

```
double getAdjustedCapital() {
  if ( _capital <= 0.0 || _intRate <= 0.0 || _duration <= 0.0 )
       return 0.0;
  return  (_income / _duration) * ADJ_FACTOR;
 }

double getAdjustedCapital() {
  if ( noAdjustment() ) return 0.0;
  return  (_income / _duration) * ADJ_FACTOR;
}

boolean noAdjustment() {
  return ( _capital <= 0.0 || _intRate <= 0.0 || _duration <= 0.0 );
}
```

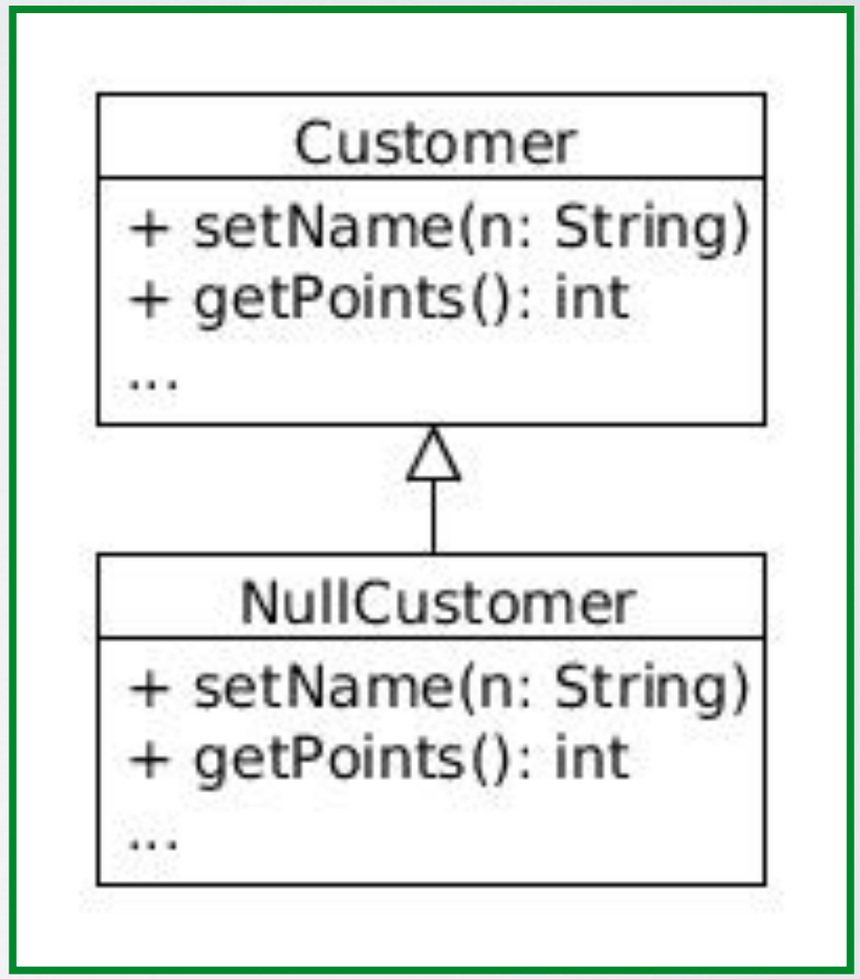
Replace conditional with polymorphism

```
double payAmount() {
    switch ( _type ) {
        case EmployeeType.ENGINEER:
            return _monthlySalary;
        case EmployeeType.SALESMAN:
            return _monthlySalary + _commission;
        case EmployeeType.MANAGER:
            return _monthlySalary + _bonus;
        default:
            throw new RuntimeException("Incorrect Employee");
    }
}
```



Introduce Null Object

```
Customer customer
 …
if ( _customer == null )
     plan = BillingPlan.basic();
else
     plan = _customer.getPlan();
 …
if ( _customer != null )
     _customer.setName(n);
 …
if ( _customer != null )
     points = _customer.getPoints();
else
     points = 0;
 ...
```



Simplifying Method Calls

Parametrize method

```
class Employee {

     void fivePercentRaise() {…}

     void tenPercentRaise() {…}

}
```

```
class Employee {

     void percentRaise(double p) {…}

     void fivePercentRaise() { percentRaise(5); }  // if still used
     void tenPercentRaise() { percentRaise(10); }  // if still used
}
```

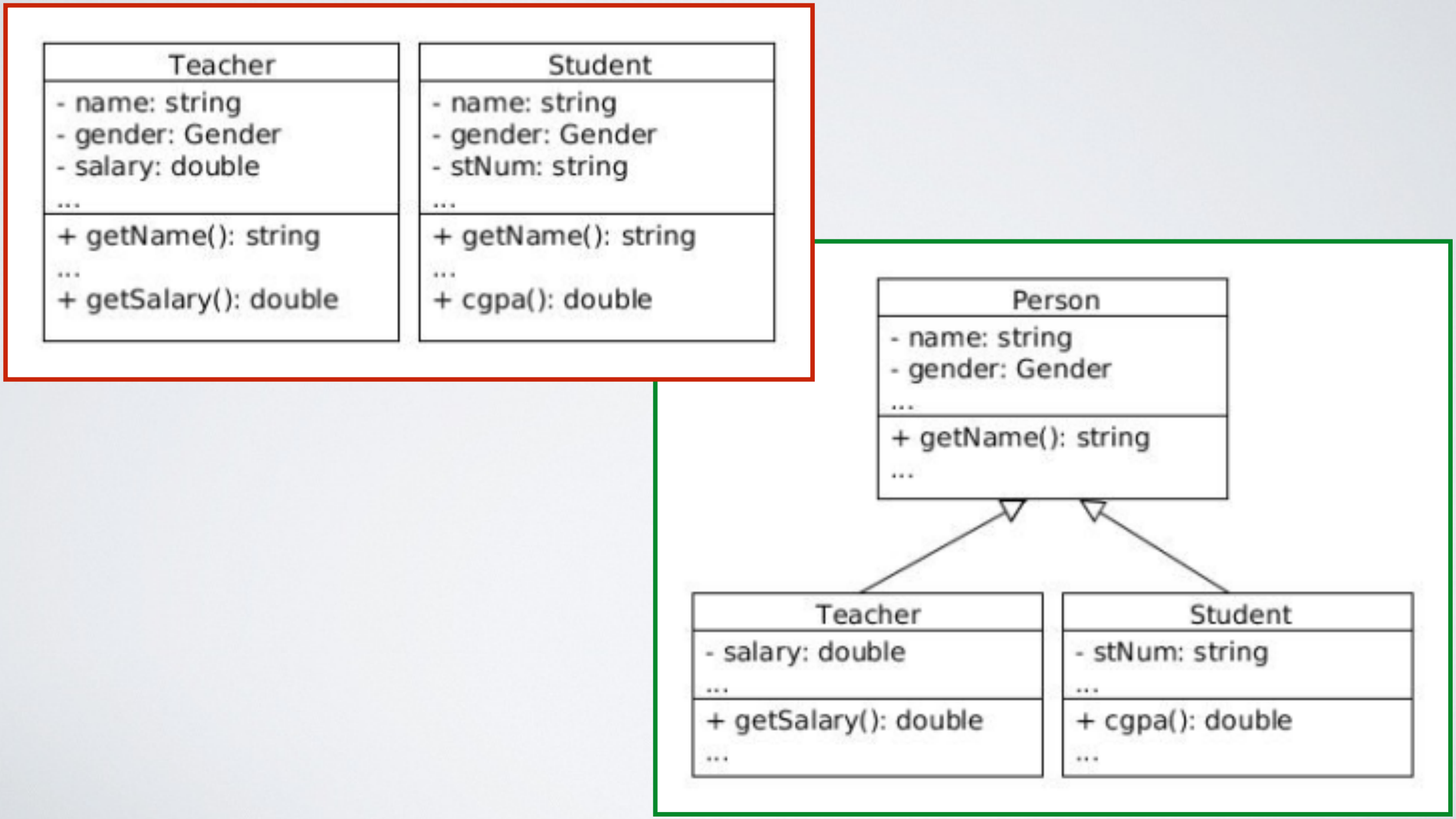
Replace parameter with explicit methods

```
void setValue(String name, int value) {
    if ( name.equals("height") ) {
        _height = value;
        return;
    }
    if ( name.equals("width") ) {
        _width = value;
        return;
    }
}
```

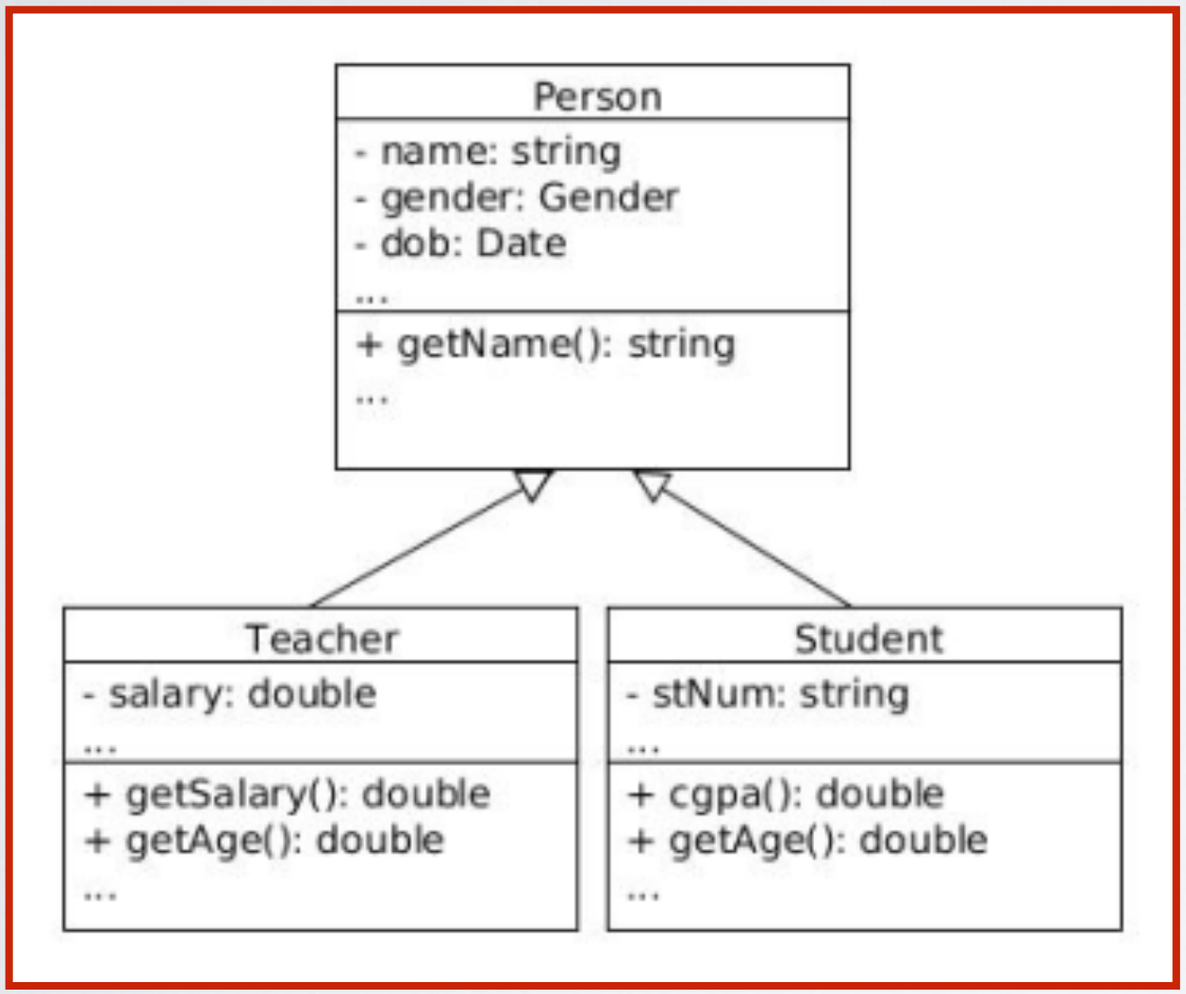
```
void setHeight(int value) {
    _height = value;
}

void setWidth(int value) {
        _width = value;
}
```

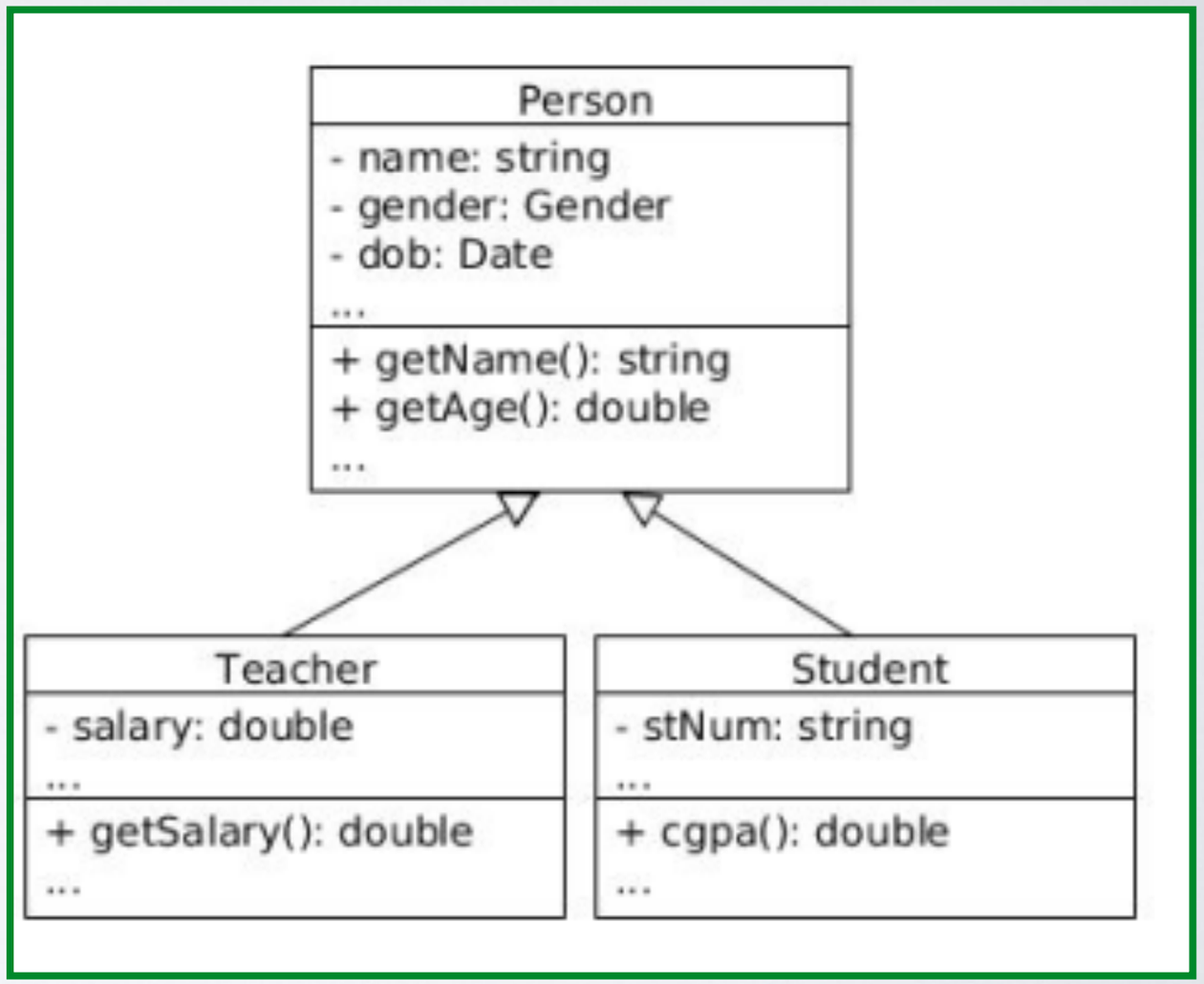
Dealing with Generalization

Extract Superclass

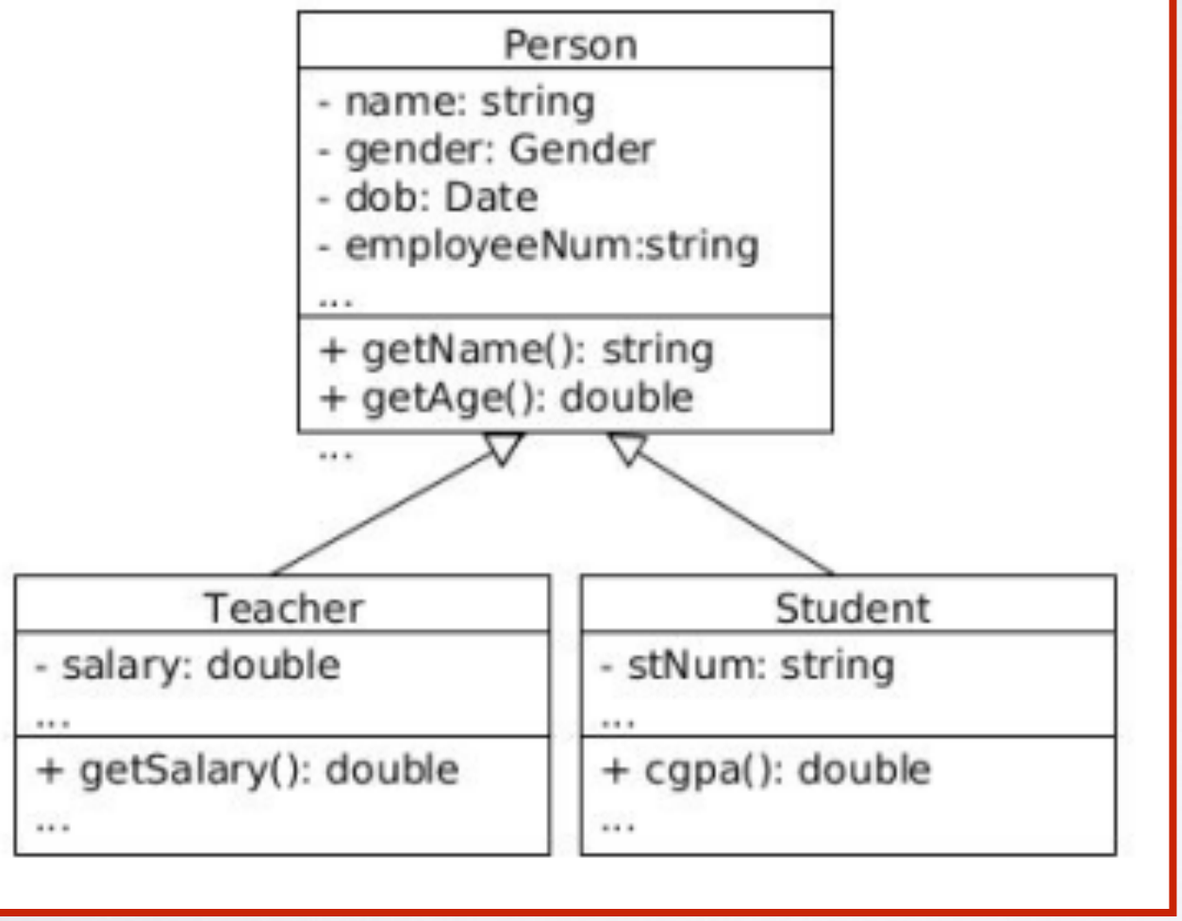


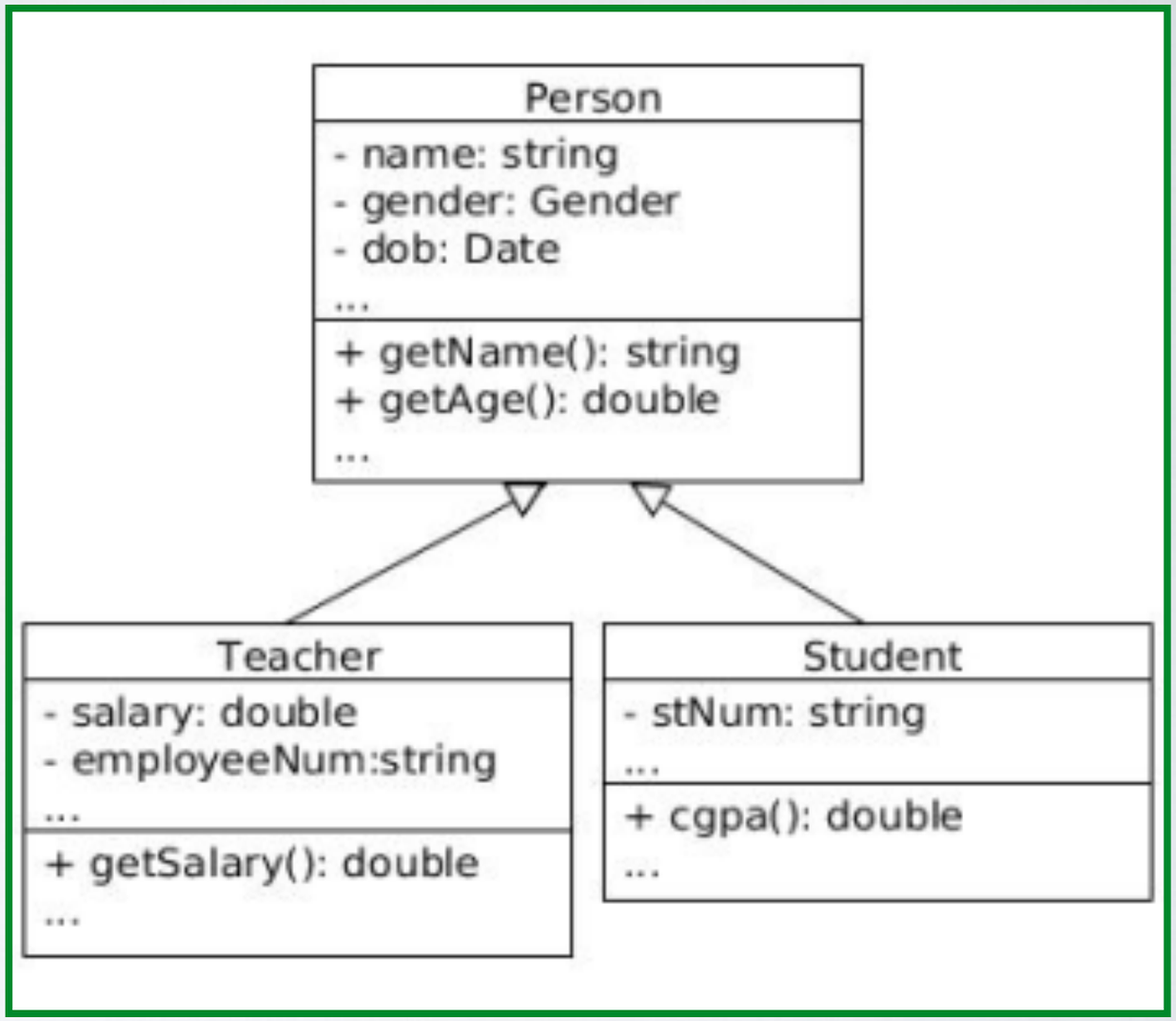
Pull up method/field



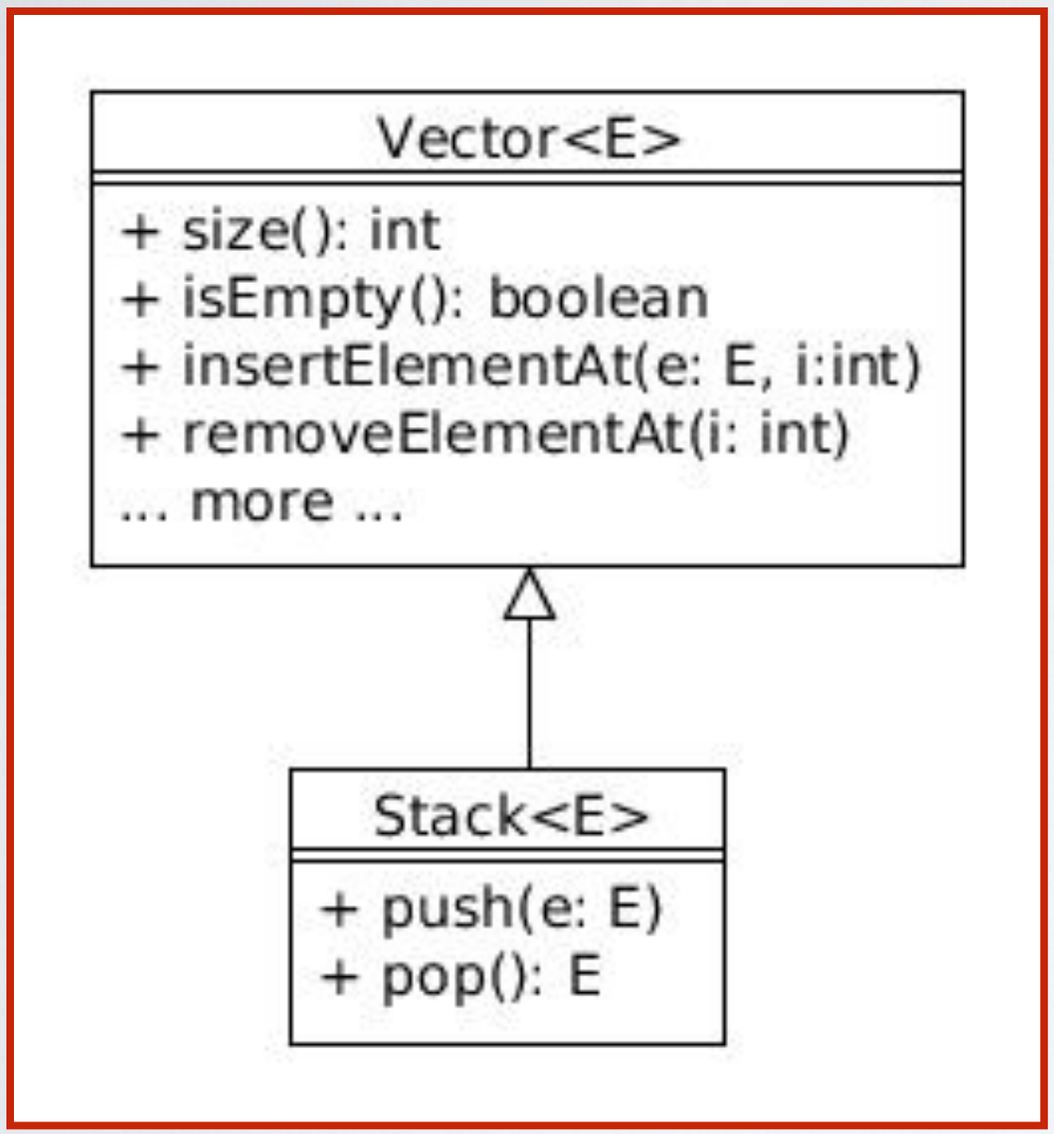


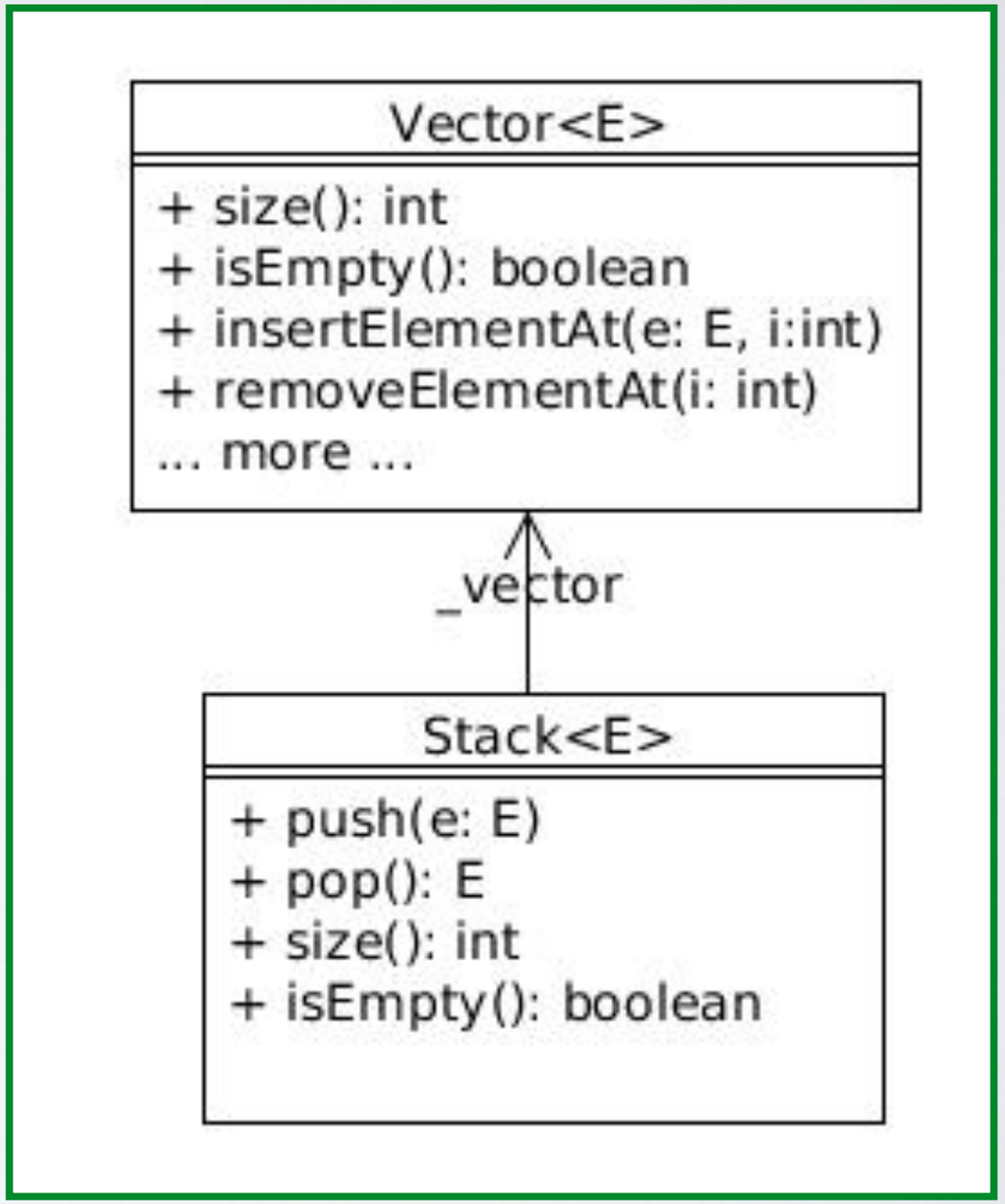
Push down method/field





Replace inheritance with delegation





Replace delegation with inheritance