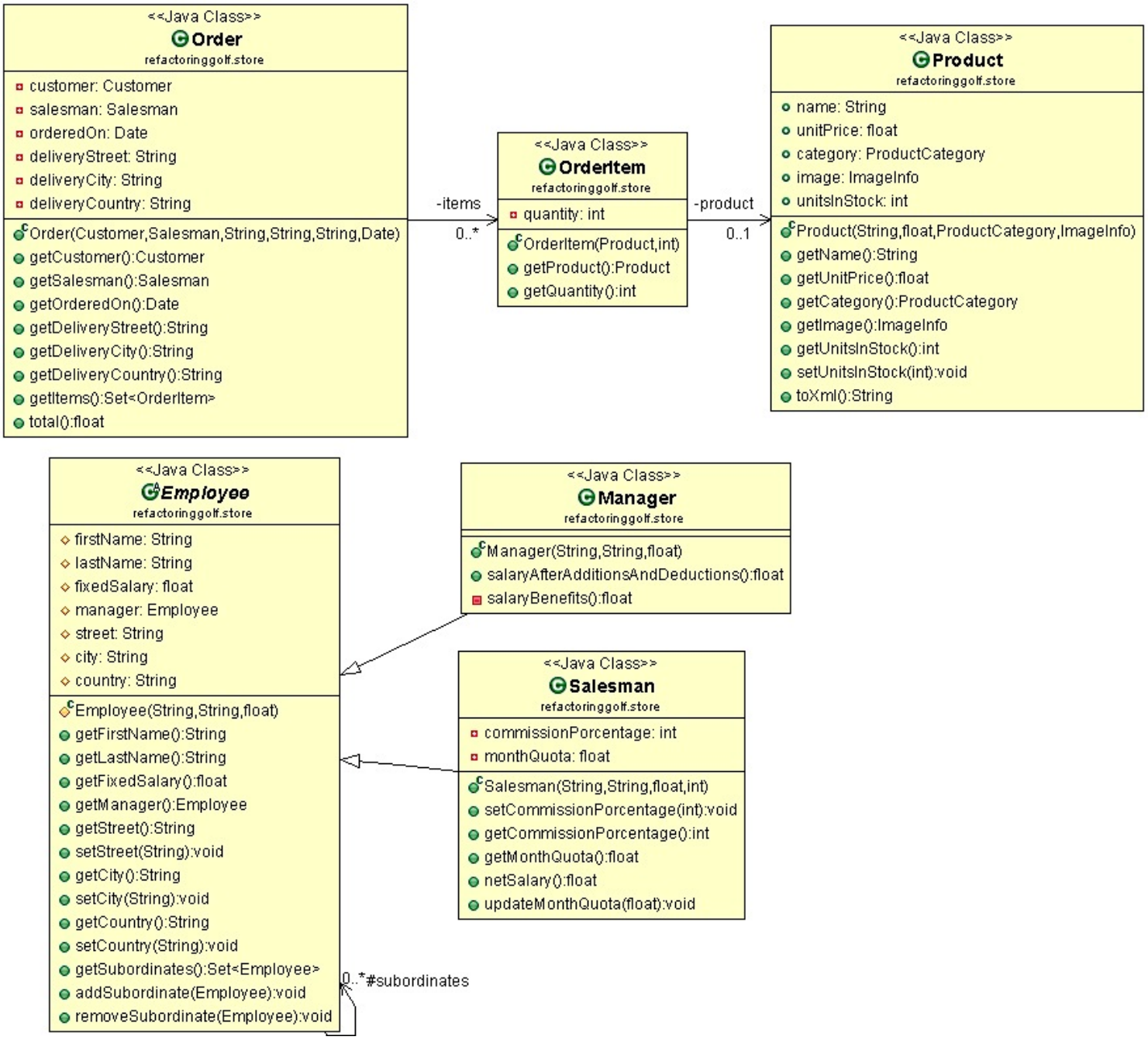
1. FIRST COURSE



INITIAL TEE


<<Java Class>>
Order
refactoringgolf.store
customer: Customer
salesman: Salesman
orderedOn: Date
deliveryStreet: String
deliveryCity: String
deliveryCountry: String
items: Set<OrderItem>
Order(Customer,Salesman,String,String,String,Date)
getCustomer():Customer
getSalesman():Salesman
getOrderedOn():Date
getDeliveryStreet():String
getDeliveryCity():String
getDeliveryCountry():String
getItems():Set<OrderItem>
total():float

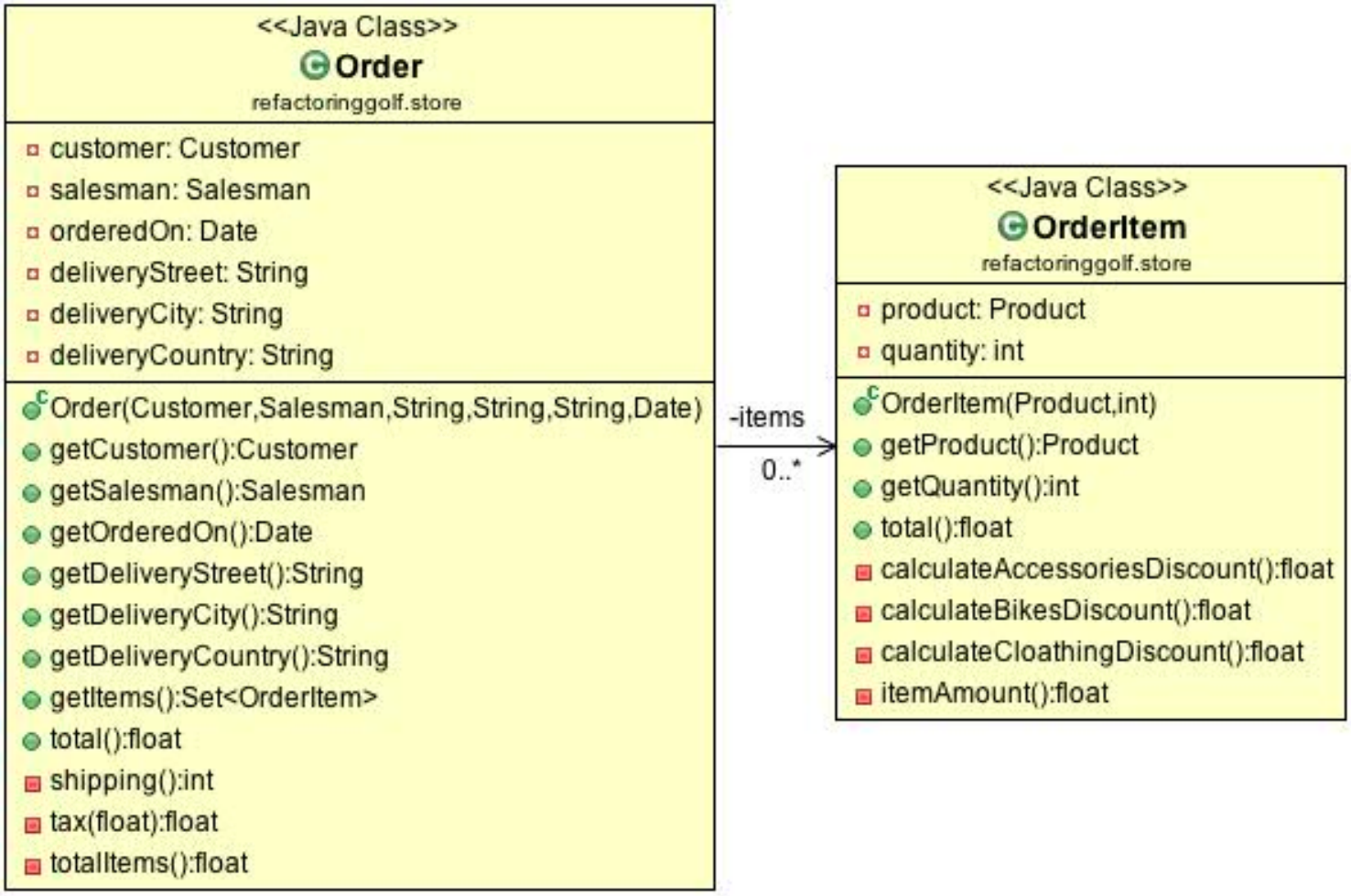
```
public class Order {

     public float total() {
          float totalItems = 0;
          for (OrderItem item : items) {
               float totalItem=0;
               float itemAmount = item.getProduct().getUnitPrice() *
                                  item.getQuantity();
               if (item.getProduct().getCategory() ==
                                             ProductCategory.Accessories) {
                    float booksDiscount = 0;
                    if (itemAmount >= 100) {
                         booksDiscount = itemAmount * 10 / 100;
                    }
                    totalItem = itemAmount - booksDiscount;
               }
               if (item.getProduct().getCategory()== ProductCategory.Bikes) {
                    // 20% discount for Bikes
                    totalItem = itemAmount - itemAmount * 20 / 100;
               }
               if (item.getProduct().getCategory() ==
                                                ProductCategory.Cloathing) {
                    float cloathingDiscount = 0;
                    if (item.getQuantity() > 2) {
                        cloathingDiscount = item.getProduct().getUnitPrice();
                    }
                    totalItem = itemAmount - cloathingDiscount;
               }
               totalItems += totalItem;
          }

          if (this.deliveryCountry == "USA"){
               // total=totalItems + tax + 0 shipping
               return totalItems + totalItems * 5 / 100;
          }

          // total=totalItemst + tax + 15 shipping
          return totalItems + totalItems * 5 / 100 + 15;
     }
}
```

FIRST HOLE


```
public class Order {

    public float total() {
        float totalItems = totalItems();
        float tax = tax(totalItems);
        int shipping = shipping();

        return totalItems + tax + shipping;
    }
```

```
    private int shipping() {
        int shipping = 15;
        if (this.deliveryCountry == "USA") {
            shipping = 0;
        }
        return shipping;
    }

    private float tax(float totalItems) {
        return totalItems * 5 / 100;
    }

    private float totalItems() {
        float totalItems = 0;
        for (OrderItem item : items) {
            totalItems += totalItem(item);
        }
        return totalItems;
    }

    private float totalItem(OrderItem item) {
        float totalItem=0;
        float itemAmount = item.getProduct().getUnitPrice() *
                                                item.getQuantity();
        if (item.getProduct().getCategory()==ProductCategory.Accessories) {
            float booksDiscount = 0;
            if (itemAmount >= 100) {
                booksDiscount = itemAmount * 10 / 100;
            }
            totalItem = itemAmount - booksDiscount;
        }
        if (item.getProduct().getCategory() == ProductCategory.Bikes) {
            // 20% discount for Bikes
            totalItem = itemAmount - itemAmount * 20 / 100;
        }
        if (item.getProduct().getCategory() == ProductCategory.Cloathing) {
            float cloathingDiscount = 0;
            if (item.getQuantity() > 2) {
                cloathingDiscount = item.getProduct().getUnitPrice();
            }
            totalItem = itemAmount - cloathingDiscount;
        }
        return totalItem;
    }
}
```

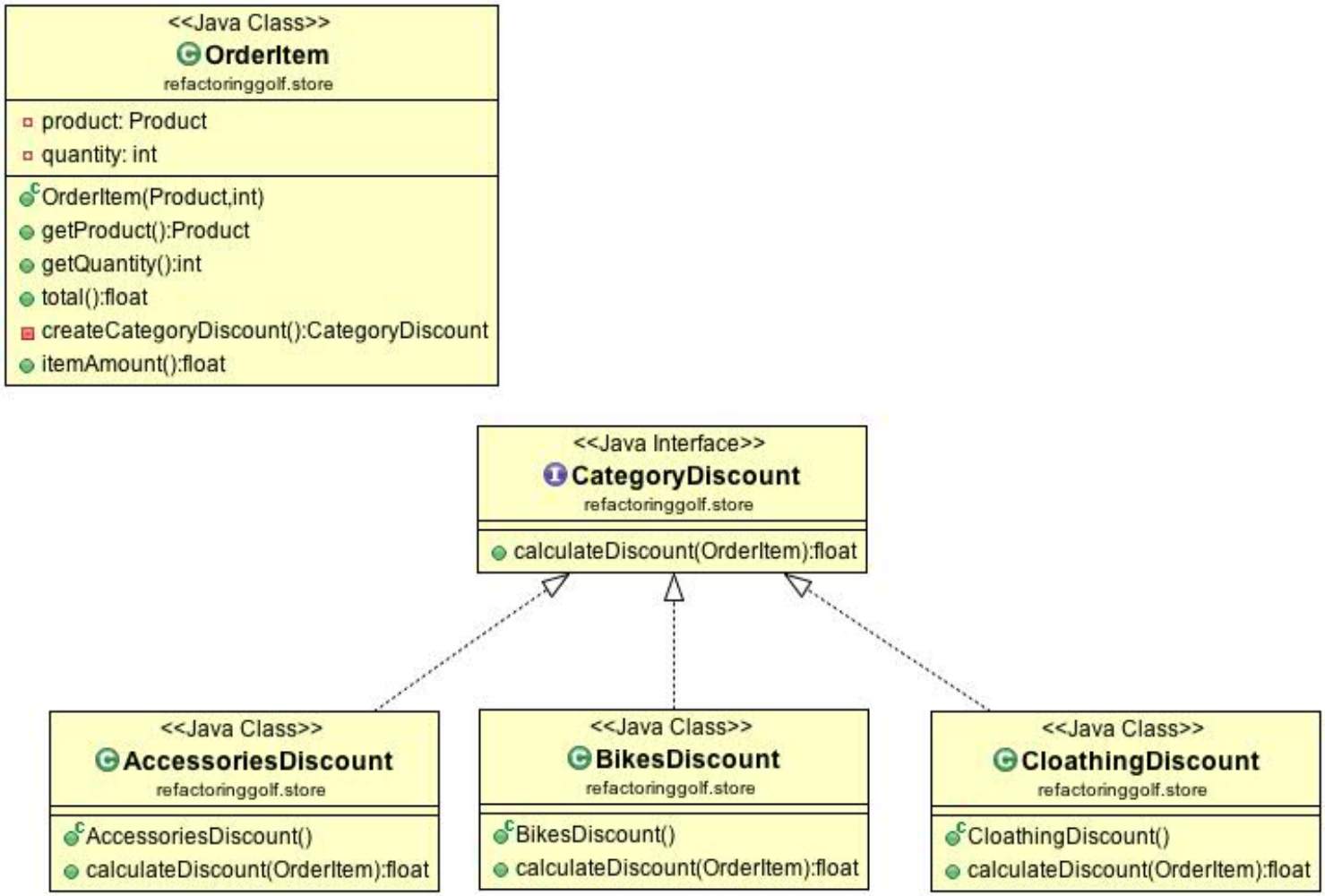
SECOND HOLE




```
public class Order {

     private float totalItems() {
          float totalAmount = 0;
          for (OrderItem item : items) {
               totalAmount += item.total();
          }
          return totalAmount;
     }
}
```

```
public class OrderItem {

     public float total() {
          float discount = 0;
          if (getProduct().getCategory() == ProductCategory.Accessories) {
               discount = calculateAccessoriesDiscount();
          }
          if (getProduct().getCategory() == ProductCategory.Bikes) {
               discount = calculateBikesDiscount();
          }
          if (getProduct().getCategory() == ProductCategory.Cloathing) {
               discount = calculateCloathingDiscount();
          }
          return itemAmount() - discount;
     }

     private float calculateAccessoriesDiscount() {
          float discount = 0;
          float unitPricePerQuantity = itemAmount();
          if (unitPricePerQuantity >= 100) {
               discount = unitPricePerQuantity * 10 / 100;
          }
          return discount;
     }

     private float calculateBikesDiscount() {
          return itemAmount() * 20 / 100;
     }

     private float calculateCloathingDiscount() {
          float discount = 0;
          if (getQuantity() > 2) {
               discount = getProduct().getUnitPrice();
          }
          return discount;
     }

     private float itemAmount() {
          return getProduct().getUnitPrice() * getQuantity();
     }
}
```

THIRD HOLE

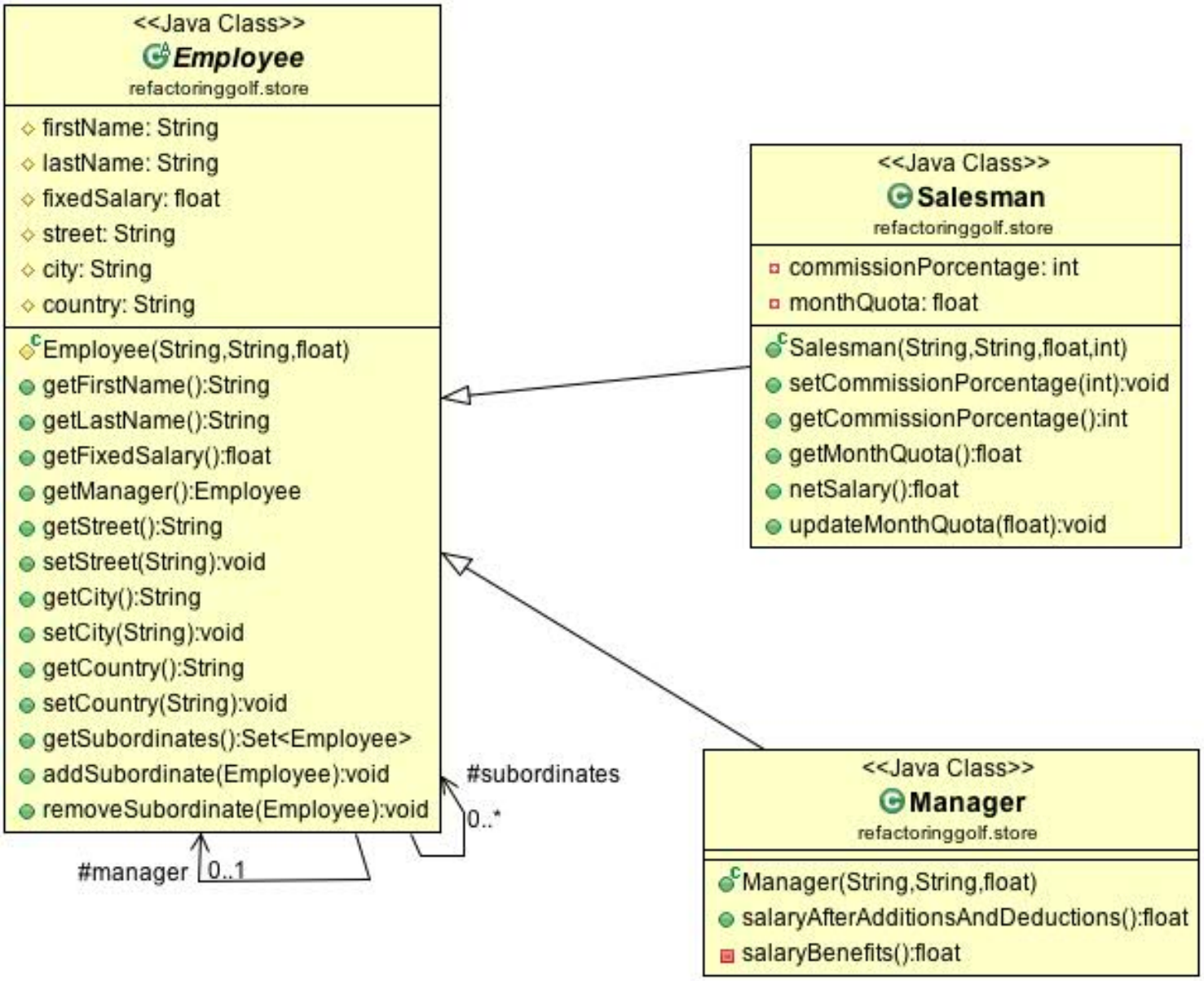



```
public class OrderItem {

     private CategoryDiscount createCategoryDiscount() {
          CategoryDiscount categoryDiscount=null;
          if (getProduct().getCategory() == ProductCategory.Accessories) {
               categoryDiscount = new AccessoriesDiscount();
          }
          if (getProduct().getCategory() == ProductCategory.Bikes) {
               categoryDiscount = new BikesDiscount();
          }
          if (getProduct().getCategory() == ProductCategory.Cloathing) {
               categoryDiscount = new CloathingDiscount();
          }
          return categoryDiscount;
     }
}
```

```
public interface CategoryDiscount {
     float calculateDiscount(OrderItem orderItem);
}

public class AccessoriesDiscount implements CategoryDiscount {

     public float calculateDiscount(OrderItem orderItem) {
          float discount = 0;
          float unitPricePerQuantity = orderItem.itemAmount();
          if (unitPricePerQuantity >= 100) {
               discount = unitPricePerQuantity * 10 / 100;
          }
          return discount;
     }
}

public class BikesDiscount implements CategoryDiscount{

     public float calculateDiscount(OrderItem orderItem) {
          return orderItem.itemAmount() * 20 / 100;
     }
}

public class CloathingDiscount implements CategoryDiscount {

     public float calculateDiscount(OrderItem orderItem) {
          float discount = 0;
          if (orderItem.getQuantity() > 2) {
               discount = orderItem.getProduct().getUnitPrice();
          }
          return discount;
     }
}
```

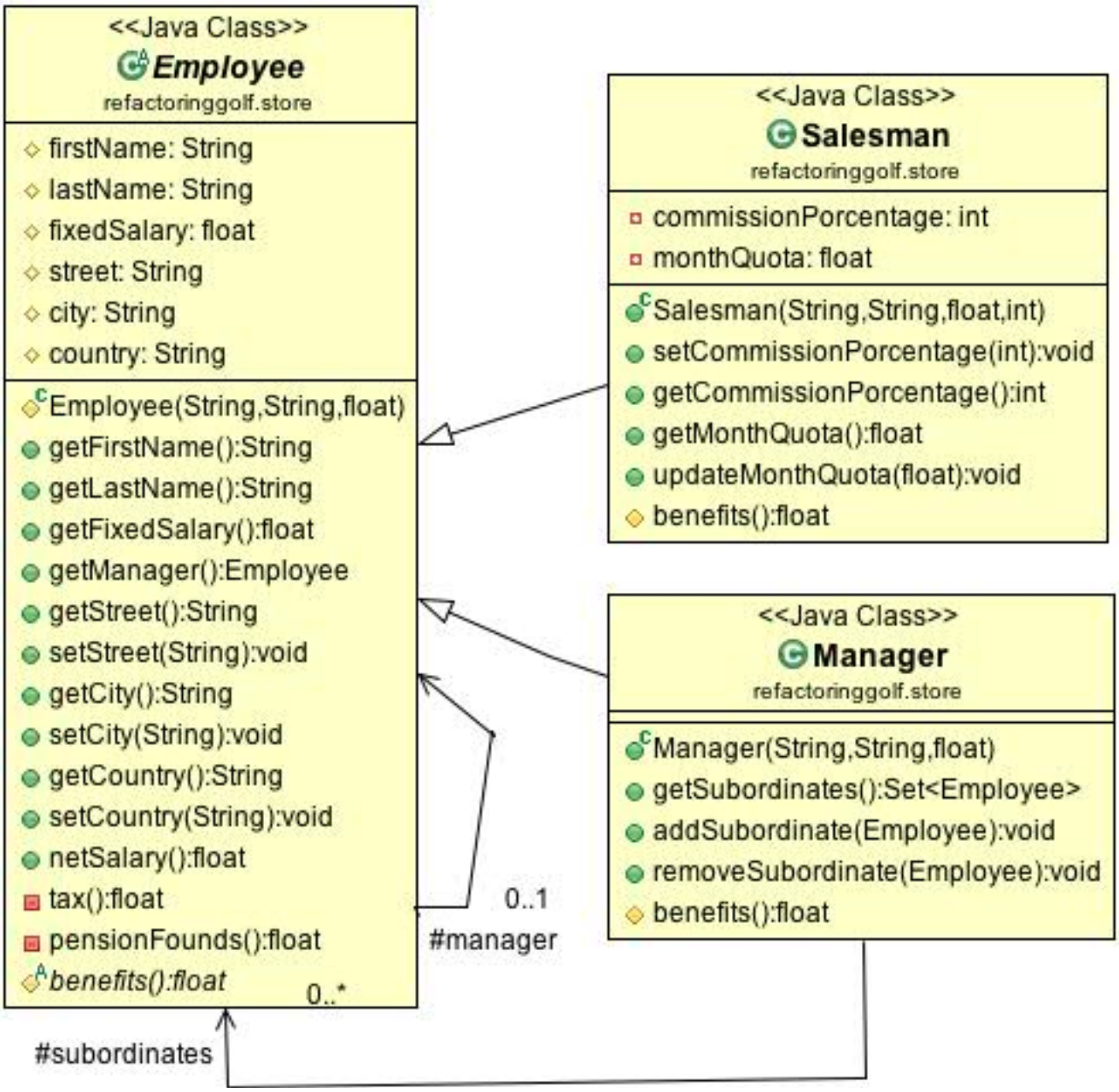
FOURTH HOLE

- **PREVIOUS HOLE (THIRD HOLE)**



```
public abstract class Employee {
     protected Set<Employee> subordinates = new HashSet<Employee>();
     public Set<Employee> getSubordinates() {
          return Collections.unmodifiableSet(subordinates);
     }

     public void addSubordinate(Employee subordinate) {
          subordinates.add(subordinate);
          subordinate.manager = this;
     }

     public void removeSubordinate(Employee subordinate) {
          subordinates.remove(subordinate);
          subordinate.manager = null;
     }
   }

public class Salesman extends Employee {
     public float netSalary() {
          float benefits = monthQuota * commissionPorcentage / 100;
          float pensionFounds = fixedSalary * 10 / 100;
          float tax = 0;
          if (fixedSalary > 3500)
               tax = fixedSalary * 5 / 100;
          return fixedSalary + benefits - pensionFounds - tax;
     }

     public void updateMonthQuota(float addQuota) {
          monthQuota = monthQuota + addQuota;
     }
}

public class Manager extends Employee {
     public float salaryAfterAdditionsAndDeductions() {
          float benefits = salaryBenefits();
          float pensionFounds = this.fixedSalary * 10 / 100;
          float tax = 0;
          if (fixedSalary > 3500)
               tax = fixedSalary * 5 / 100;
          return fixedSalary + benefits - pensionFounds - tax;
     }

     private float salaryBenefits() {
          return this.subordinates.size() * 20;
     }
```

- **CURRENT HOLE**



```
public abstract class Employee {
     public float netSalary() {
          return fixedSalary + benefits() - pensionFounds() - tax();
     }

     private float tax() {
          float tax = 0;
          if (fixedSalary > 3500)
               tax = fixedSalary * 5 / 100;
          return tax;
     }

     private float pensionFounds() {
          return this.fixedSalary * 10 / 100;
     }

     protected abstract float benefits();
}

public class Salesman extends Employee {
     protected float benefits() {
          return monthQuota * commissionPorcentage / 100;
     }
}

public class Manager extends Employee {
     protected Set<Employee> subordinates = new HashSet<Employee>();

     public Set<Employee> getSubordinates() {
          return Collections.unmodifiableSet(subordinates);
     }

     public void addSubordinate(Employee subordinate) {
          subordinates.add(subordinate);
          subordinate.manager = this;
     }

     public void removeSubordinate(Employee subordinate) {
          subordinates.remove(subordinate);
          subordinate.manager = null;
     }

     protected float benefits() {
          return this.subordinates.size() * 20;
     }
}
```