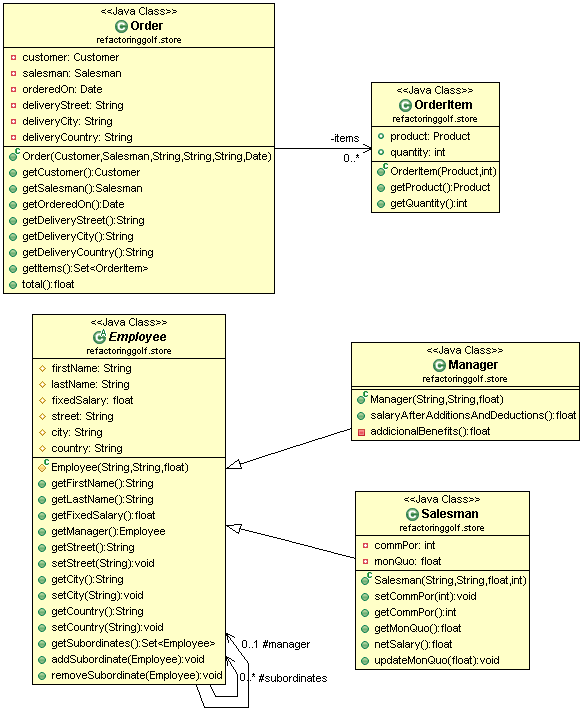
1. **FIRST COURSE**

**INITIAL TEE**

* **CLASS DIAGRAM**

****

* **CODE**

public class Order {

private Customer customer;

private Salesman salesman;

private Date orderedOn;

private String deliveryStreet;

private String deliveryCity;

private String deliveryCountry;

private Set<OrderItem> items;

public Order(Customer customer, Salesman salesman, String deliveryStreet, String deliveryCity, String deliveryCountry, Date orderedOn) {

this.customer = customer;

this.salesman = salesman;

this.deliveryStreet = deliveryStreet;

this.deliveryCity = deliveryCity;

this.deliveryCountry = deliveryCountry;

this.orderedOn = orderedOn;

this.items = new HashSet<OrderItem>();

}

public Customer getCustomer() {

return customer;

}

public Salesman getSalesman() {

return salesman;

}

public Date getOrderedOn() {

return orderedOn;

}

public String getDeliveryStreet() {

return deliveryStreet;

}

public String getDeliveryCity() {

return deliveryCity;

}

public String getDeliveryCountry() {

return deliveryCountry;

}

public Set<OrderItem> getItems() {

return items;

}

public float total() {

float totalItems = 0;

for (OrderItem item : items) {

float itemAmount = item.getProduct().getUnitPrice() \* item.getQuantity();

if (item.getProduct().getCategory() == ProductCategory.*Accessories*) {

float booksDiscount = 0;

if (itemAmount >= 100) {

booksDiscount = itemAmount \* 10 / 100;

}

itemAmount = itemAmount - booksDiscount;

}

if (item.getProduct().getCategory() == ProductCategory.*Bikes*) {

// itemAmount=itemAmount-discount

itemAmount = itemAmount - itemAmount \* 20 / 100;

}

if (item.getProduct().getCategory() == ProductCategory.*Cloathing*) {

float cloathingDiscount = 0;

if (item.getQuantity() > 2) {

cloathingDiscount = item.getProduct().getUnitPrice();

}

itemAmount = itemAmount - cloathingDiscount;

}

totalItems += itemAmount;

}

if (this.deliveryCountry == "USA")

// totalAmount=totalItemAmount + tax + 0 shipping

return totalItems + totalItems \* 5 / 100;

// totalAmount=totalItemAmount + tax + 15 shipping

return totalItems + totalItems \* 5 / 100 + 15;

}

}

public class OrderItem {

public Product product;

public int quantity;

public OrderItem(Product product, int quantity) {

this.product = product;

this.quantity = quantity;

}

public Product getProduct() {

return product;

}

public int getQuantity() {

return quantity;

}

}

public abstract class Employee {

protected String firstName;

protected String lastName;

protected float fixedSalary;

protected Employee manager;

protected String street;

protected String city;

protected String country;

/\*

\* Only the managers have subordinates but anyone could have a manager.

\*/

protected Set<Employee> subordinates = new HashSet<Employee>();

protected Employee(String firstName, String lastName, float fixedSalary) {

this.firstName = firstName;

this.lastName = lastName;

this.fixedSalary = fixedSalary;

}

public String getFirstName() {

return firstName;

}

public String getLastName() {

return lastName;

}

public float getFixedSalary() {

return fixedSalary;

}

public Employee getManager() {

return manager;

}

public String getStreet() {

return street;

}

public void setStreet(String street) {

this.street = street;

}

public String getCity() {

return city;

}

public void setCity(String city) {

this.city = city;

}

public String getCountry() {

return country;

}

public void setCountry(String country) {

this.country = country;

}

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 {

private int commPor; // CommissionPorcentage

private float monQuo; // MonthQuota

public Salesman(String firstName, String lastName, float fixedSalary, int commPor){

super(firstName, lastName, fixedSalary);

this.commPor = commPor;

}

public void setCommPor(int commPor) {

this.commPor = commPor;

}

public int getCommPor() {

return commPor;

}

public float getMonQuo() {

return monQuo;

}

public float netSalary() {

float addicionalBenefits = monQuo \* commPor / 100;

float pensionFounds = fixedSalary \* 10 / 100;

float tax = 0;

if (fixedSalary > 3500)

tax = fixedSalary \* 5 / 100;

return addicionalBenefits + fixedSalary - pensionFounds - tax;

}

public void updateMonQuo(float addQuo) {

monQuo = monQuo + addQuo;

}

}

public class Manager extends Employee {

public Manager(String firstName, String lastName, float fixedSalary) {

super(firstName, lastName, fixedSalary);

}

public float salaryAfterAdditionsAndDeductions() {

float addicionalBenefits = addicionalBenefits();

float pensionFounds = this.fixedSalary \* 10 / 100;

float tax = 0;

if (fixedSalary > 3500)

tax = fixedSalary \* 5 / 100;

return addicionalBenefits + fixedSalary - pensionFounds - tax;

}

private float addicionalBenefits() {

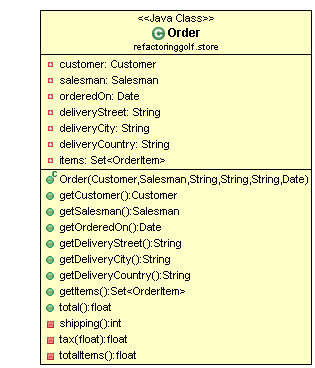
return this.subordinates.size() \* 20;

}

}

**FIRST HOLE**

* **CLASS DIAGRAM**

****

* **CODE**

public class Order {

private Customer customer;

private Salesman salesman;

private Date orderedOn;

private String deliveryStreet;

private String deliveryCity;

private String deliveryCountry;

private Set<OrderItem> items;

public Order(Customer customer, Salesman salesman, String deliveryStreet, String deliveryCity, String deliveryCountry, Date orderedOn) {

this.customer = customer;

this.salesman = salesman;

this.deliveryStreet = deliveryStreet;

this.deliveryCity = deliveryCity;

this.deliveryCountry = deliveryCountry;

this.orderedOn = orderedOn;

this.items = new HashSet<OrderItem>();

}

public Customer getCustomer() {

return customer;

}

public Salesman getSalesman() {

return salesman;

}

public Date getOrderedOn() {

return orderedOn;

}

public String getDeliveryStreet() {

return deliveryStreet;

}

public String getDeliveryCity() {

return deliveryCity;

}

public String getDeliveryCountry() {

return deliveryCountry;

}

public Set<OrderItem> getItems() {

return items;

}

**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 totalAmount) {**

**return totalAmount \* 5 / 100;**

**}**

**private float totalItems() {**

**float totalAmount = 0;**

**for (OrderItem item : items) {**

**float discount=0;**

**float itemAmount = item.getProduct().getUnitPrice() \* item.getQuantity();**

**if (item.getProduct().getCategory() == ProductCategory.*Accessories*) {**

**discount = 0;**

**if (itemAmount >= 100) {**

**discount = itemAmount \* 10 / 100;**

**}**

**}**

**if (item.getProduct().getCategory() == ProductCategory.*Bikes*) {**

**discount = itemAmount \* 20 / 100;**

**}**

**if (item.getProduct().getCategory() == ProductCategory.*Cloathing*) {**

**discount = 0;**

**if (item.getQuantity() > 2) {**

**discount = item.getProduct().getUnitPrice();**

**}**

**}**

**itemAmount = itemAmount - discount;**

**totalAmount += itemAmount;**

**}**

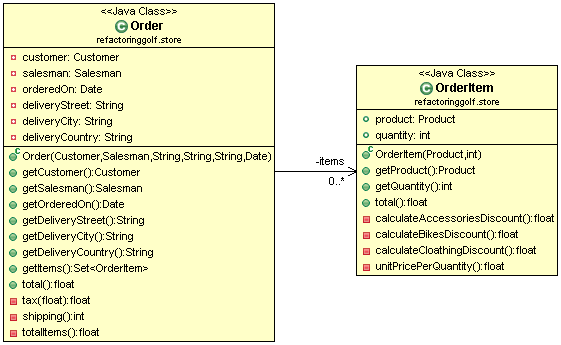
**return totalAmount;**

**}**

**}**

**SECOND HOLE**

* **CLASS DIAGRAM**



* **CODE**

public class Order {

private Customer customer;

private Salesman salesman;

private Date orderedOn;

private String deliveryStreet;

private String deliveryCity;

private String deliveryCountry;

private Set<OrderItem> items;

public Order(Customer customer, Salesman salesman, String deliveryStreet, String deliveryCity, String deliveryCountry, Date orderedOn) {

this.customer = customer;

this.salesman = salesman;

this.deliveryStreet = deliveryStreet;

this.deliveryCity = deliveryCity;

this.deliveryCountry = deliveryCountry;

this.orderedOn = orderedOn;

this.items = new HashSet<OrderItem>();

}

public Customer getCustomer() {

return customer;

}

public Salesman getSalesman() {

return salesman;

}

public Date getOrderedOn() {

return orderedOn;

}

public String getDeliveryStreet() {

return deliveryStreet;

}

public String getDeliveryCity() {

return deliveryCity;

}

public String getDeliveryCountry() {

return deliveryCountry;

}

public Set<OrderItem> getItems() {

return items;

}

public float total() {

float totalItems = totalItems();

float tax = tax(totalItems);

int shipping = shipping();

return totalItems + tax + shipping;

}

private float tax(float totalAmount) {

return totalAmount \* 5 / 100;

}

private int shipping() {

int shipping = 15;

if (this.deliveryCountry == "USA") {

shipping = 0;

}

return shipping;

}

**private float totalItems() {**

**float totalAmount = 0;**

**for (OrderItem item : items) {**

**totalAmount += item.total();**

**}**

**return totalAmount;**

**}**

}

public class OrderItem {

public Product product;

public int quantity;

public OrderItem(Product product, int quantity) {

this.product = product;

this.quantity = quantity;

}

public Product getProduct() {

return product;

}

public int getQuantity() {

return quantity;

}

**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 unitPricePerQuantity() - discount;**

**}**

**private float calculateAccessoriesDiscount() {**

**float discount = 0;**

**float unitPricePerQuantity = unitPricePerQuantity();**

**if (unitPricePerQuantity >= 100) {**

**discount = unitPricePerQuantity \* 10 / 100;**

**}**

**return discount;**

**}**

**private float calculateBikesDiscount() {**

**return unitPricePerQuantity() \* 20 / 100;**

**}**

**private float calculateCloathingDiscount() {**

**float discount = 0;**

**if (getQuantity() > 2) {**

**discount = getProduct().getUnitPrice();**

**}**

**return discount;**

**}**

**private float unitPricePerQuantity() {**

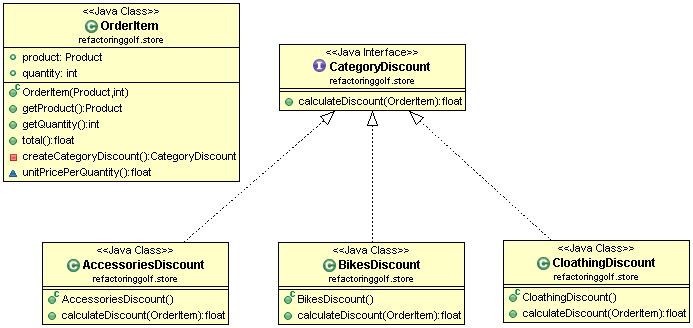
**return getProduct().getUnitPrice() \* getQuantity();**

**}**

}

**THIRD HOLE**

* **CLASS DIAGRAM**

****

* **CODE**

public class OrderItem {

public Product product;

public int quantity;

public OrderItem(Product product, int quantity) {

this.product = product;

this.quantity = quantity;

}

public Product getProduct() {

return product;

}

public int getQuantity() {

return quantity;

}

**public float total() {**

**return unitPricePerQuantity() - createCategoryDiscount().calculateDiscount(this);**

**}**

**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;**

**}**

float unitPricePerQuantity() {

return getProduct().getUnitPrice() \* getQuantity();

}

}

**public interface CategoryDiscount {**

**float calculateDiscount(OrderItem orderItem);**

**}**

**public class AccessoriesDiscount implements CategoryDiscount {**

**public float calculateDiscount(OrderItem orderItem) {**

**float discount = 0;**

**float unitPricePerQuantity = orderItem.unitPricePerQuantity();**

**if (unitPricePerQuantity >= 100) {**

**discount = unitPricePerQuantity \* 10 / 100;**

**}**

**return discount;**

**}**

**}**

**public class BikesDiscount implements CategoryDiscount{**

**public float calculateDiscount(OrderItem orderItem) {**

**return orderItem.unitPricePerQuantity() \* 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;**

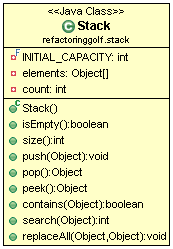
**}**

**}**

1. **SECOND COURSE**

**TEE**

* **CLASS DIAGRAM**

****

* **CODE**

public class Stack {

private final int INITIAL\_CAPACITY = 5;

private Object[] elements = new Object[INITIAL\_CAPACITY];

private int count;

public boolean isEmpty() {

return count == 0;

}

public int size() {

return count;

}

public void push(Object element) {

if (count + 1 > elements.length) {

Object[] temp = new Object[2 \* elements.length];

System.*arraycopy*(elements, 0, temp, 0, elements.length);

elements = temp;

}

elements[count] = element;

count++;

}

public Object pop() {

if (isEmpty())

throw new IllegalStateException();

Object element = elements[count - 1];

count--;

return element;

}

public Object peek() {

return elements[count - 1];

}

public boolean contains(Object elementToFind) {

for (int i = 0; i < count; i++) {

if (elementToFind == elements[i]) {

return true;

}

}

return false;

}

public int search(Object elementToFind) {

for (int i = 1; i <= count; i++) {

if (elementToFind == elements[count - i]) {

return i;

}

}

return -1;

}

public void replaceAll(Object elementToFind, Object newElement) {

for (int i = count - 1; i >= 0; i--) {

if (elementToFind == elements[i]) {

elements[i] = newElement;

}

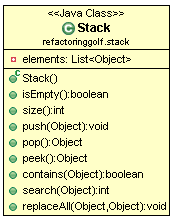
}

}

}

**HOLE**

* **CLASS DIAGRAM**

****

* **CODE**

public class Stack {

private List<Object> elements = new ArrayList<Object>();

public boolean isEmpty()

{

return size() == 0;

}

public int size()

{

return elements.size();

}

public void push(Object element)

{

elements.add(element);

}

public Object pop()

{

if (isEmpty())

throw new IllegalStateException();

Object element = elements.get(size() - 1);

elements.remove(size()-1);

return element;

}

public Object peek()

{

return elements.get(size() - 1);

}

public boolean contains(Object elementToFind)

{

int indexOf = search(elementToFind);

if (indexOf != -1)

{

return true;

}

return false;

}

public int search(Object elementToFind)

{

for (int i = 1; i <= size(); i++)

{

if (elementToFind == elements.get(size()-i))

{

return i;

}

}

return -1;

}

public void replaceAll(Object elementToFind, Object newElement)

{

for (int i = size() - 1; i >= 0; i--)

{

if (elementToFind == elements.get(i))

{

elements.set(i,newElement);

}

}

}

}