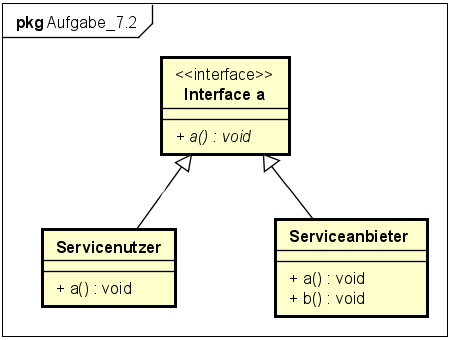
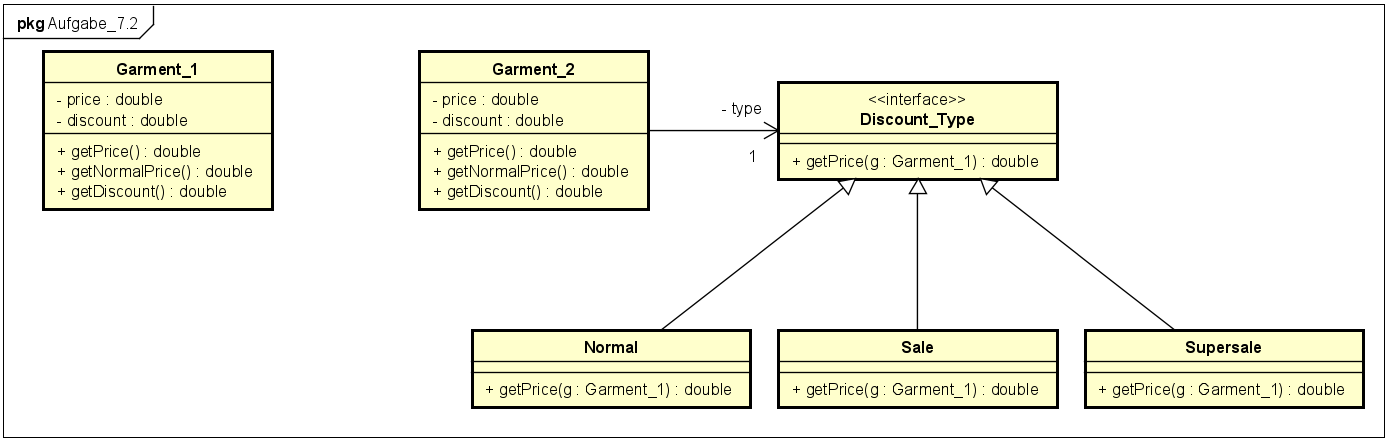
Aufgabe 7.1)

1. Servicenutzer ist von Serviceanbieter abhängig, durch das ein Attribut vom typ Serviceanbieter in der Servicenutzer Klasse und einen Methodenaufruf des Serviceanbieter Attribut in der Servicenutzer Klasse



Aufgabe 7.2)

public class Garment\_2 {  
 private double price;  
 private double discount;  
 private Discount\_type type;  
  
 public Garment\_2(double price, double discount) {  
 this.price = price;  
 this.discount = discount;  
 }  
  
 public double getPrice() {  
 return type.getPrice(this);  
 }  
  
 public double getDiscount() {  
 return discount;  
 }  
  
 public double getNormalPrice() {  
 return price;  
 }  
}

public interface Discount\_type {  
 double getPrice(Garment\_2 g);  
}

public class Normal implements Discount\_type {  
 @Override  
 public double getPrice(Garment\_2 g) {  
 return g.getNormalPrice();  
 }  
}

public class Sale implements Discount\_type {  
 @Override  
 public double getPrice(Garment\_2 g) {  
 return g.getNormalPrice() \* g.getDiscount();  
 }  
}

public class Supersale implements Discount\_type {  
 @Override  
 public double getPrice(Garment\_2 g) {  
 return g.getNormalPrice() \* 0.5;  
 }  
}

Aufgabe 7.3)

**Nachteile:**

* Häufige unnötige Checks
* Redundante Attribute
* Alles als Attribut zu speichern 🡪 schlechtes Design

**Verbesserungen**:

* Klasse Student mit Vor-/Nachnamen hat 1..n Beziehung zu Konto
* Klasse Konto mit Iban und Datum der Gültigkeit hat 1..1 Beziehung zu Geldinstitut
* Klasse Geldinstitut mit Namen etc.

Aufgabe 7.4)

* Extract Class auf Teilnehmer für die Kreditkarte + rename(Präfixe Karte entfernen)
* Move Method getAlleTeilnehmer() und getAlleProfessoren() in Seminar

Aufgabe 7.5)

* Extract Class auf die Punkte
* Extract Interface für flache()

public class Prog {  
 public static void main(String[] args) {  
 Linie l1 = new Linie(0.0,0.0,1.0,1.0);  
 System.out.println(l1.distanz());  
 System.out.println(l1.flaeche());  
 }  
}

public interface LinieInterface {  
 double flaeche();  
}

public class Linie implements LinieInterface {  
  
 private final Aufgabe7\_5.punktStart punktStart = new punktStart();  
 private final Aufgabe7\_5.punktEnde punktEnde = new punktEnde();  
  
 public Linie(double startX, double startY,  
 double endX, double endY) {  
 this.punktStart.setStartX(startX);  
 this.punktStart.setStartY(startY);  
 this.punktEnde.setEndX(endX);  
 this.punktEnde.setEndY(endY);  
 }  
  
 public double distanz() {  
 return Math.sqrt(Math.pow(punktEnde.getEndX() - punktStart.getStartX(), 2.0) + Math.pow(punktEnde.getEndY() - punktStart.getStartY(), 2.0));  
 }  
  
 @Override  
 public double flaeche() {  
 return (Math.abs(punktEnde.getEndX() - punktStart.getStartX()) \* Math.abs(punktEnde.getEndY() - punktStart.getStartY())) / 2;  
 }  
}

public class punktEnde {  
 private double endX;  
  
 public double getEndX() {  
 return endX;  
 }  
  
 public void setEndX(double endX) {  
 this.endX = endX;  
 }  
  
 private double endY;  
  
 public double getEndY() {  
 return endY;  
 }  
  
 public void setEndY(double endY) {  
 this.endY = endY;  
 }  
  
 public punktEnde() {  
 }  
}

public class punktStart {  
 private double startX;  
  
 public double getStartX() {  
 return startX;  
 }  
  
 public void setStartX(double startX) {  
 this.startX = startX;  
 }  
  
 private double startY;  
  
 public double getStartY() {  
 return startY;  
 }  
  
 public void setStartY(double startY) {  
 this.startY = startY;  
 }  
  
 public punktStart() {  
 }  
}