Prof. Dr. Christian Kohls
Alexander Dobrynin, M.Sc.

April 19, 2022
SoSe 2022

Algorithmen und Programmierung 2
Praktikum 2 - Vererbung und Polymorphie

In diesem Praktikum beschäftigen Sie sich mit Vererbung und Polymorphie. Dazu wird eine Klassenhierarchie aufgebaut, Eigenschaften und Methoden vererbt und überschrieben. Zudem wird ein Interface verwendet und unterschiedliche Implementierungen dafür bereitgestellt. Außerdem werden Sie die Bedeutung und Vorteile von Polymorphie kennenlernen.

Für die Aufgaben sind die folgenden Videos aus der Playlist auf YouTube relevant:

- Die Videos zur Vererbung und Polymorphie verwenden die Movie Maker App als Beispiel. Schauen Sie sich zunächst die Videos zur Entstehung der Movie Maker App an: "21. Movie Maker App - Konzept" und "22. Klassen Schauspieler und Regisseur festlegen".
- "23. Vererbung" bis "26. Typkompatibilität".
- "30. Abstrakte Klassen". Hinweis: Abstrakte Klassen sind für dieses Übungsblatt zwar nicht notwendig, aber grundsätzlich wichtig, um Vererbung zu verstehen.
- "31. Interfaces definieren" bis "33. Typen und Operationen".

Bzw. folgende Kapitel aus dem Buch Programmieren lernen mit Kotlin:

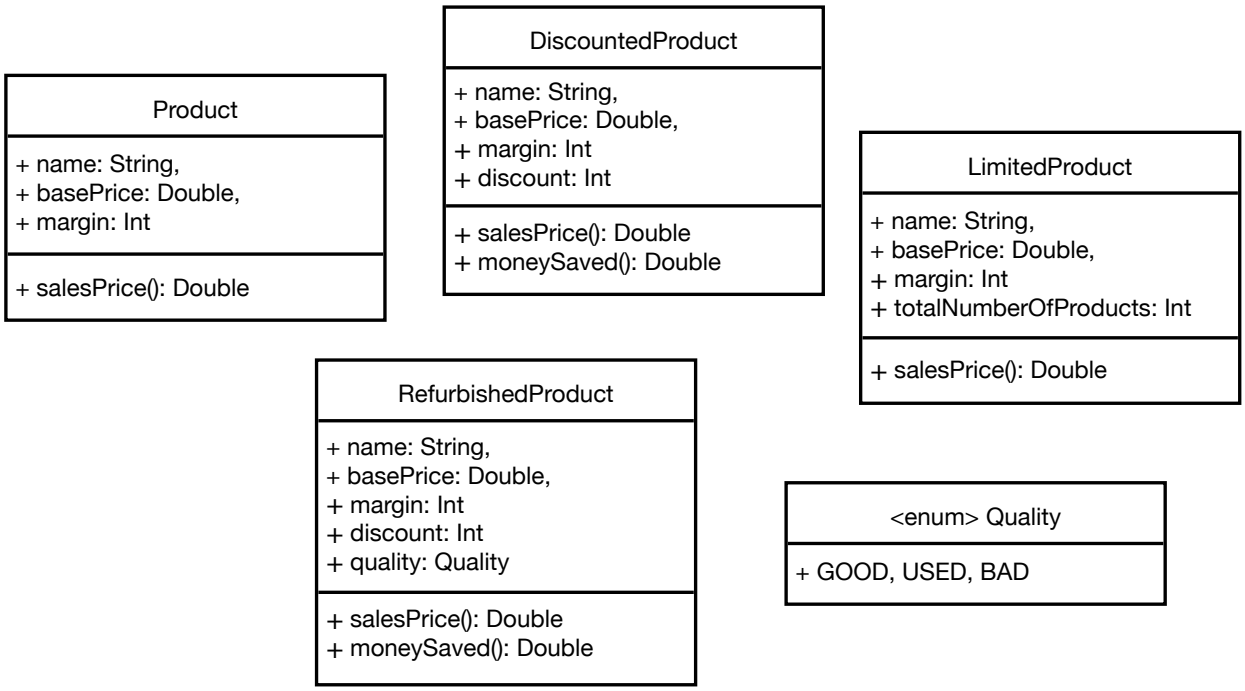
- Kapitel "16 Vererbung".
- Kapitel "17 Polymorphie".
- Kapitel "18 Abstrakte Klassen und Schnittstellen".

Contents

1 Vererbung

In dieser Aufgabe müssen Sie anhand einer Klassenstruktur Gemeinsamkeiten und Unterschiede erkennen. Auf dieser Grundlage bilden Sie daraus Ober- und Unterklassen.
Bei dem Fallbeispiel handelt es sich um ein kleines Shopsystem, in dem es unterschiedliche Arten von Produkten gibt.

Das ist die Klassenstruktur:



Das ist der dazugehörige Code:

```
class Product(
  val name: String,
  val basePrice: Double,
  val margin: Int
) {
  fun salesPrice(): Double {
      TODO("")
  }
}

class DiscountedProduct(
  val name: String,
  val basePrice: Double,
  val margin: Int,
  val discount: Int
) {
  fun salesPrice(): Double {
```

```
        TODO("")
    }

    fun moneySaved(): Double {
        TODO("")
    }
}

enum class Quality {
    GOOD, USED, BAD
}

class RefurbishedProduct(
    val name: String,
    val basePrice: Double,
    val margin: Int,
    val discount: Int,
    val quality: Quality
) {
    fun salesPrice(): Double {
        TODO("")
    }

    fun moneySaved(): Double {
        TODO("")
    }
}

class LimitedProduct(
    val name: String,
    val basePrice: Double,
    val margin: Int,
    val totalNumberOfProducts: Int
) {
    fun salesPrice(): Double {
        TODO("")
    }
}
```

Ein paar Hinweise zu den Eigenschaften:

- `name: String` => Name des Produktes.
- `basePrice: Double` => Einkaufspreis.
- `margin: Int` => Gibt die erwartete Gewinnmarge des Produktes in Prozent an. Bspw. 20 => 20%

- `discount: Int` => Gibt den Rabatt des Produktes in Prozent an.
- `quality: Quality` => Gibt die Qualität des Produktes an. Entweder Gut, Benutzt oder Schlecht.
- `totalNumberOfProducts: Int` => Gibt an, wie viele Exemplare es von diesem Produkt gibt.

1.1 Struktur konzeptionieren

Betrachten Sie die dargestellte Klassenstruktur und den dazughehörigen Code. Machen Sie sich zu den folgenden Punkten der Vererbung Gedanken:

- Was ist die Oberklasse?
- Was sind die Unterklassen?
- Welche Eigenschaften und Methoden gehören zur Oberklasse, weil diese allgemein sind?
- Welche Eigenschaften und Methoden gehören in die Unterklassen, weil diese spefizisch sind?
- Welche Eigenschaften und Methoden müssten von welcher Unterklasse überschrieben / erweitert werden?

1.2 Struktur implementieren

Setzen Sie die Vererbung um. Überlegen Sie sich, was als `open` deklariert werden muss und welche Parameter an die Oberklasse übergeben werden müssen.
Die Methoden `salesPrice` und `moneySaved` implementieren Sie erst in der nächsten Teilaufgabe.

1.3 Methoden implementieren

Implementieren Sie jetzt die folgenden Methoden. Überlegen Sie, welche Methoden überschrieben oder ergänzt werden sollen:

- `salesPrice` => Der Verkaufspreis berechnet sich aus dem Einkaufspreis und der erwarteten Gewinnmarge. Beispiel:
  Verkaufspreis: 5.0
  Gewinnmarge: 20%
  Der Verkaufspreis ist demnach: 6.0

  Bei `DiscountedProduct` und `RefurbishedProduct` muss der Rabatt (`discount`) zusätzlich abgezogen werden.

Bei `LimitedProduct` steigt der Verkaufswert mit der Seltenheit des Produktes.

Beispiel:

- Wenn `totalNumberOfProducts` auf 10 gesetzt ist, würde dies einem Aufschlag von 90% entsprechen.
- Wenn `totalNumberOfProducts` auf 30 gesetzt ist, würde dies einem Aufschlag von 70% entsprechen.
- Die Berechnung ist demnach: Aufschlag in Prozent = 100 - `totalNumberOfProducts`.

- `moneySaved` => Gibt den gesparten Betrag zurück.
- `toString` => Überschreiben Sie die `toString` Methode so, dass sie einer Produktbeschreibung ähnelt. Formatieren Sie die Preise auf 2 Nachkommastellen. Je nach Produkt sieht `toString` etwas anders aus:
  - Product: `Buch für 12,99 Euro`
  - DiscountedProduct: `Süßigkeiten für 2,07 Euro (0,52 Euro gespart)`
  - RefurbishedProduct: `Handy für 287,99 Euro (72,00 Euro gespart) (Gebraucht)`
  - LimitedProduct: `Bild für 237,45 Euro (nur 10 Examplare verfügbar)`

1.4 Implementierung testen

Schreiben Sie eine `main` Funktion, in der Sie unterschiedliche Arten von Produkten erzeugen und die jeweiligen Methoden aufrufen. Hier ein paar Beispiele:

```
fun main() {
  val book = Product("Buch", 9.99, 30)
  println(book.salesPrice()) // 12.987
  println(book.toString()) // Buch für 12,99 Euro

  val ball = Product("Ball", 4.99, 100)
  println(ball.salesPrice()) // 9.98
  println(ball.toString()) // Ball für 9,98 Euro

  val candy = DiscountedProduct("Süßigkeiten", 1.99, 30, 20)
  println(candy.salesPrice()) // 2.0696000000000003
  println(candy.moneySaved()) // 0.5173999999999999
  println(candy.toString()) // Süßigkeiten für 2,07 Euro (0,52 Euro
     gespart)

  val phone = RefurbishedProduct("Handy", 299.99, 20, 20, Quality.USED)
  println(phone.salesPrice()) // 287.9904
  println(phone.moneySaved()) // 71.99759999999999
```

```
  println(phone.toString()) // Handy für 287,99 Euro (72 ,00 Euro
     gespart) (Gebraucht)

  val picture = LimitedProduct("Bild", 49.99, 150, 10)
  println(picture.salesPrice()) // 237.45250000000001
  println(picture.toString()) // Bild für 237,45 Euro (nur 10
     Exemplare verfügbar)
}
```

In den Kommentaren steht jeweils das, was durch `println` auf der Konsole ausgegeben wird. Stellen Sie sicher, dass Sie ähnliche Ergebnisse haben.

2 Dynamische Bindung

2.1 Begriffsdefinition

Was ist Dynamische Bindung? Geben Sie ein Beispiel.

2.2 Typkompatibilität

Schauen Sie sich die folgenden Zuweisungen an. Beantworten Sie für jede Zuweisung, ob diese valide ist, ob es zu einem Compilezeitfehler oder zu einem Laufzeitfehler kommt. Begründen Sie ihre Aussage. Zur Wiederholung:

- Compilezeitfehler: Code kompiliert erst gar nicht.
- Laufzeitfehler: Code kompiliert zwar, stürzt aber zur Laufzeit ab.
- Valide: Code kompiliert und stürzt nicht zur Laufzeit ab.

Hier die Zuweisungen:

```
val p1: Product = Product("Buch", 9.99, 30)
val p2: Product = DiscountedProduct("Süßigkeiten", 1.99, 30, 20)
val p3: Product = RefurbishedProduct("Handy", 299.99, 20, 20,
   Quality.USED)
val p4: DiscountedProduct = p1
val p5: DiscountedProduct = p3 as RefurbishedProduct
val p6: DiscountedProduct = p2 as RefurbishedProduct
val p7: RefurbishedProduct = DiscountedProduct("Süßigkeiten", 1.99,
   30, 20)
val p8: Product = p1 as Any
val p9: Any = p1
```

3 Interface

In dieser Aufgabe soll eine Bewertung (Review) als Interface festgelegt werden.

3.1 Begriffsdefinition

Was ist ein Interface? Worin unterscheidet sich ein Interface zu einer Oberklasse?

3.2 Reviews implementieren

Definieren Sie folgendes Interface für ein `Review`:

```
interface Review {
  fun score(): Int
  fun infoText(): String
}
```

Implementieren Sie jetzt drei unterschiedlichen Arten von Reviews:

- Sternebewertungen (`StarReview`): Die Klasse `StarReview` nimmt eine Anzahl von Sternen zwischen 0 und 5 entgegen.
  - Der `score` entspricht den Sternen. Wenn die Sterne außerhalb von 0 bis 5 liegen, wird der `score` entsprechend auf 0 oder 5 korrigiert.
    Beispiel:
    3 Sterne -> Score ist 3
    -2 Sterne -> Score ist 0
    8 Sterne -> Score ist 5
  - Der `infoText` gibt je nach Anzahl der Sterne eine Beschreibung zurück.
    Beispiel:
    0 Sterne -> `"Schlechtes Produkt"`
    3 Sterne -> `"Brauchbares Produkt"`
    5 Sterne -> `"Super Produkt"`
    Für die Sterne 1, 2 und 4 können Sie sich einen entsprechenden Text ausdenken.
- Daumen-Hoch Bewertungen (`ThumbBasedReview`): Die Klasse `ThumbBasedReview` nimmt einen `Boolean` entgegen, welcher angibt, ob ein "Daumen hoch" vergeben wurde.
  - Ist der Daumen oben, ist der `score` 5, sonst 0.
  - Der `infoText` gibt entweder `"Daumen hoch"` oder `"Daumen runter"` zurück, je nach dem, welchen Wert der `Boolean` hat.

3.3 Produkte um Reviews erweitern

Jedes Produkt soll zusätzlich eine Liste von Reviews haben. Passen Sie den Code entsprechend an. Machen Sie sich hierfür die Vererbungshierarchie zunutze.

Fügen Sie auch eine Methode `addReview` hinzu, die ein `Review` akzeptiert und dieses Review der Liste hinzufügt.

3.4 Implementierung testen

Erzeugen Sie ein paar Reviews in der `main` Funktion. Fügen Sie die Reviews ein paar Produkten hinzu. Beispiel:

```
fun main() {
  val book = Product("Buch", 9.99, 30)
  book.addReview(StarReview(5))
  book.addReview(ThumbBasedReview(false))

  val picture = LimitedProduct("Bild", 49.99, 150, 10)
  picture.addReview(ThumbBasedReview(true))
}
```

Achtung: Je nachdem, wie Sie die Aufgabe 3.3 implementiert haben, sieht Ihr Code etwas anders aus. Das wichtige ist, dass jedes Produkt über die Methode `addReview` verfügt und dass Sie der `addReview` Methode jedes `Review`, also z.B. ein `StarReview` oder `ThumbBasedReview` übergeben können.

4 Polymorphie

In dieser Aufgabe soll ein Warenkorb simuliert werden.

4.1 Die Klasse "ShoppingCart"

Definieren Sie eine Klasse namens `ShoppingCart`, die eine veränderliche Liste von Produkten als Eigenschaft hat.

4.2 Begriffsdefinition

Was bedeutet es, wenn etwas polymorph ist? Beziehen Sie sich z.B. auf die `addReview` Methode aus der Aufgabe 3.3 oder auf die Liste von Produkten.

4.3 Methoden implementieren

Implementieren Sie die folgenden Methoden in der Klasse `ShoppingCart`. Überlegen Sie sich, was der jeweilige Rückgabetyp ist:

- `add(product: Product)` => Fügt das übergebene Produkt der Liste hinzu.
- `totalPrice()` => Gibt den Gesamtpreis des Warenkorbs zurück. Dieser basiert auf dem Verkaufspreis aller Produkte.
- `hasFreeShipping()` => Gibt zurück, ob der Versand kostenlos ist. Der Versand ist kostenlos, wenn der Gesamtpreis des Warenkorbs über 30.0 liegt.
- `getProductByName(productName: String)` => Gibt das erste Produkt zurück, welches dem übergebenen Namen entspricht. Wenn kein passendes Produkt gefunden wurde, wird `null` zurückgegeben.
- `reviewTextForProduct(productName: String)` => Gibt alle Infotexte des ersten Produkts zurück, welches dem übergebenen Namen entspricht. Die Infotexte kommen aus den Reviews und sollen zu einem String kombiniert werden. Wenn kein passendes Produkt gefunden wurde, wird ein leerer String zurückgegeben.
- `removeProductsBelowReviewScore(score: Int)` => Entfernt alle Produkte aus dem Warenkorb, dessen durchschnittliche Bewertung (`score`) unter dem übergebenen `score` liegt.
- `savedMoney()` => Gibt zurück, wie viel durch die Rabatte gespart wurde. Tipp: Hierfür benötigen Sie einen Typecast.
- `show()` => Gibt zuerst den Gesamtpreis des Warenkorbs auf der Konsole aus. Zudem wird ausgegeben, wie viel durch Rabatte gespart wurde. Danach werden alle Produkte nacheinander ausgegeben. Die gesamte Ausgabe kann z.B. so aussehen:

```
Gesamt: 550,48 Euro (72,51 Euro gespart)
Buch für 12,99 Euro
Ball für 9,98 Euro
Süßigkeiten für 2,07 Euro (0,52 Euro gespart)
Handy für 287,99 Euro (72,00 Euro gespart) (Gebraucht)
Bild für 237,45 Euro (nur 10 Examplare verfügbar)
```

Auch hier können Sie für die Ausgabe von Geldbeträgen alle Preise auf 2 Nachkommastellen formatiert ausgeben.

4.4 Implementierung testen

Erzeugen Sie in der `main` Funktion ein Objekt vom Typ `ShoppingCart` und fügen Sie diesem verschiedene Produkte hinzu. Rufen Sie die in der Aufgabe 4.1 implementierten Methoden auf. Hier ein Beispiel:

```
fun main() {
  val cart = ShoppingCart()
  cart.add(Product("Buch", 9.99, 30))
  cart.add(Product("Ball", 4.99, 100))
  cart.add(DiscountedProduct("Süßigkeiten", 1.99, 30, 20))
  cart.add(RefurbishedProduct("Handy", 299.99, 20, 20, Quality.USED))
  cart.add(LimitedProduct("Bild", 49.99, 150, 10))
  cart.show()
}
```

Der Aufruf von `cart.show` erzeugt die folgende Ausgabe auf der Konsole:

```
Gesamt: 550,48 Euro (72,51 Euro gespart)
Buch für 12,99 Euro
Ball für 9,98 Euro
Süßigkeiten für 2,07 Euro (0,52 Euro gespart)
Handy für 287,99 Euro (72,00 Euro gespart) (Gebraucht)
Bild für 237,45 Euro (nur 10 Examplare verfügbar)
```

Stellen Sie sicher, dass Sie in etwa die gleichen Ergebnisse bekommen.