

Dokumentation

Hausaufgabe 1 Anwendungssysteme

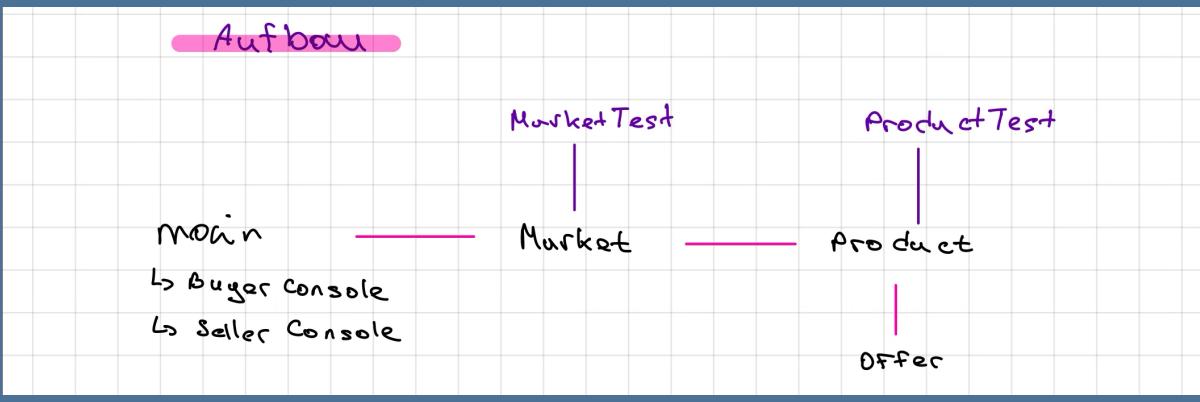
Taras Levankou

Matrikelnummer: 493936

B. Sc. Wirtschaftsinformatik

Mai 2025

Aufbau



ORDNERSTRUKTUR:

```

Assignment1Maven ~/soru/Study/A5
  > .idea
  > src
    > main
      > java
        > de.mcc
          Main
          Market
          Offer
          Product
      > test
        > java
          MarketTest
          ProductTest
    > target
    .gitignore
    Example Programm Output.md
    pom.xml
  > External Libraries
  > Scratches and Consoles
  
```

-Das Projekt ist ein konsolenbasiertes Handelssystem für Kauf und Verkauf von Produkten. Die Logik liegt in Market, Produkte in Product, Angebote in Offer, Main steuert die Benutzeroberfläche. Als Ergänzung sind zentrale Funktionen mit JUnit in ProductTest und MarketTest getestet.

MAIN-KLASSE:

```

1 package de.mcc;
2
3 import lombok.extern.java.Log;
4 import java.util.Scanner;
5
6 @Log
7 public class Main {
8
9     static {
10         for (var handler : java.util.logging.Logger.getLogger("").getHandlers()) {
11             handler.setFormatter(new java.util.logging.Formatter() {
12                 @Override
13                 public String format(java.util.logging.LogRecord record) {
14                     return record.getMessage() + "\n";
15                 }
16             });
17         }
18     }
19
20     private static final Scanner scanner = new Scanner(System.in); 19 usages
21     private static final Market market = new Market(); 10 usages
  
```

-Verwendet Lombok, um automatisch einen Logger zu erzeugen (log.info(), log.warning() usw.), ohne ihn manuell zu deklarieren.
-Setzt ein benutzerdefiniertes Log-Format. Entfernt Zeitstempel und Metadaten aus der Log-Ausgabe, um die Konsole übersichtlicher zu gestalten.
-Initialisiert den Scanner für Benutzereingaben und ein zentrales Market-Objekt für die Handelslogik.

```

public static void main(String[] args) {
    while (true) {
        log.info("Hauptmenü ===
        1. Als Käufer fortfahren
        2. Als Verkäufer fortfahren
        0. Beenden
        ");

        log.info("Auswahl: ");
        String input = scanner.nextLine();

        switch (input) {
            case "1" -> runBuyerConsole();
            case "2" -> runSellerConsole();
            case "0" -> {
                log.info("Programm beendet. Auf Wiedersehen!");
                return;
            }
            default -> log.warning("Ungültige Eingabe.");
        }
    }
}
  
```

Zeigt ein Hauptmenü mit drei Optionen:

1 = Käufer-Menü
2 = Verkäufer-Menü
0 = Beenden

Die Benutzerwahl wird eingelesen und an die passende Methode weitergeleitet.

```

private static void runBuyerConsole() { 1 usage
    while (true) {
        log.info( msg: "====\n"
        == Käufermenü ===
        1. Alle Produkte anzeigen
        2. Produkt kaufen
        3. Angebote zu einem Produkt anzeigen
        4. Kursverlauf anzeigen
        5. Nachschlagewörter ... "
        log.info( msg: "Auswahl: ");
        String input = scanner.nextLine();
        0. Zurück
    }
    switch (input) {
        case "1" -> market.showAllProducts();
        case "2" -> {
            log.info( msg: "Produktname: ");
            String name = scanner.nextLine();
            log.info( msg: "Menge: ");
            int qty;
            while (true) {
                try {
                    qty = Integer.parseInt(scanner.nextLine());
                    break;
                } catch (NumberFormatException e) {
                    log.warning( msg: "Bitte geben Sie eine gültige ganze Zahl ein.");
                }
            }
            market.buyProduct(name, qty);
        }
        case "3" -> {
            log.info( msg: "Produktname: ");
            String name = scanner.nextLine();
            market.showProductOffers(name);
        }
        case "4" -> {
            log.info( msg: "Produktname: ");
            String name = scanner.nextLine();
            market.showPriceHistory(name);
        }
        case "5" -> {
            log.info( msg: "Attribut eingeben: ");
            String attr = scanner.nextLine();
            market.searchByAttribute(attr);
        }
        case "0" -> {
            log.info( msg: "Zurück zum Hauptmenü.");
            return;
        }
        default -> log.warning( msg: "Ungültige Eingabe.");
    }
}

```

Menü für Käufer mit folgenden Funktionen:

- Alle Produkte anzeigen
 - Produkt kaufen (inkl. Eingabevervalidierung für Menge)
 - Angebote zu einem Produkt anzeigen
 - Kursverlauf anzeigen
 - Suche nach Attributen
- Fehleingaben werden abgefangen und mit Warnung erneut abgefragt.

```

private static void runSellerConsole() { 1 usage
    log.info( msg: "Ihr Verkäufername: ");
    String seller = scanner.nextLine();

```

```

    while (true) {
        log.info( msg: "====\n"
        == Verkäufermenü ===
        1. Neues Angebot hinzufügen
        2. Eigene Angebote anzeigen
        3. Preis/Menge ändern
        4. Attribute ändern
        5. Angebot löschen
        6. Zurück
    }

```

```

    log.info( msg: "Auswahl: ");
    String input = scanner.nextLine();

    switch (input) {
        case "1" -> {
            log.info( msg: "Produktname: ");
            String name = scanner.nextLine();
            log.info( msg: "Menge: ");
            int qty;
            while (true) {
                try {
                    qty = Integer.parseInt(scanner.nextLine());
                    break;
                } catch (NumberFormatException e) {
                    log.warning( msg: "Bitte geben Sie eine gültige ganze Zahl ein.");
                }
            }
            log.info( msg: "Preis: ");
            double price;
            while (true) {
                try {
                    price = Double.parseDouble(scanner.nextLine().replace( target: ",", replacement: "."));
                    break;
                } catch (NumberFormatException e) {
                    log.warning( msg: "Bitte geben Sie eine gültige Dezimalzahl ein.");
                }
            }
            log.info( msg: "Attribute (kommagetrennt): ");
            String[] attrs = scanner.nextLine().split( regex: "\\s" );
            market.addOfferWithAttributes(name, new Offer(seller, price, qty), attrs);
            log.info( msg: "Angebot hinzugefügt.");
        }
        case "2" -> {
            log.info( msg: "Eigene Angebote: ");
            market.showSellerOffers(seller);
        }
        case "3" -> {
            log.info( msg: "Produktname: ");
            String name = scanner.nextLine();
            log.info( msg: "Neuer Preis: ");
            double newPrice;
            while (true) {
                try {
                    newPrice = Double.parseDouble(scanner.nextLine().replace( target: ",", replacement: "."));
                    break;
                } catch (NumberFormatException e) {
                    log.warning( msg: "Bitte geben Sie eine gültige Dezimalzahl ein.");
                }
            }
            log.info( msg: "Neue Menge: ");
            int newQty;
            while (true) {
                try {
                    newQty = Integer.parseInt(scanner.nextLine());
                    break;
                } catch (NumberFormatException e) {
                    log.warning( msg: "Bitte geben Sie eine gültige ganze Zahl ein.");
                }
            }
            market.updateSellerOffer(seller, name, newPrice, newQty);
        }
        case "4" -> {
            log.info( msg: "Produktname: ");
            String name = scanner.nextLine();
            log.info( msg: "Neue Attribute (kommagetrennt): ");
            String[] newAttrs = scanner.nextLine().split( regex: "\\s" );
            market.updateProductAttributes(seller, name, newAttrs);
        }
        case "5" -> {
            log.info( msg: "Produktname: ");
            String name = scanner.nextLine();
            market.removeSellerOffer(seller, name);
        }
        case "6" -> {
            log.info( msg: "Zurück zum Hauptmenü.");
            return;
        }
        default -> log.warning( msg: "Ungültige Eingabe.");
    }
}

```

Menü für Verkäufer mit folgenden Optionen:

- Neues Angebot hinzufügen (inkl. Validierung von Menge und Preis)
- Eigene Angebote anzeigen
- Preis und Menge eines bestehenden Angebots ändern
- Attribute eines Produkts ändern
- Eigenes Angebot löschen

Benutzereingaben werden auf korrekte Formate geprüft und ggf. erneut angefordert.

MARKET-KLASSE:

```

1 package de.mod;
2
3 import lombok.extern.java.Log;
4 import lombok.Getter;
5 import java.util.*;
6
7 @Log 4 usages
8 @Getter
9 public class Market {
10     private Map<String, Product> products = new HashMap<>();
11
12     @
13     public void addOfferWithAttributes(String productName, Offer offer, String[] attrs) { 11 usages
14         String key = productName.toLowerCase();
15         if (!products.containsKey(key)) {
16             products.put(key, new Product(productName, attrs));
17         }
18         products.get(key).addOffer(offer);
19     }
20
21     public void searchByAttribute(String keyword) { 3 usages
22         boolean found = false;
23         for (Product p : products.values()) {
24             if (p.matchesAttribute(keyword)) {
25                 log.info( msg: p.getName() + " -> Attribute: " + String.join( delimiter: ", ", p.getAttributes()));
26                 p.showOffers();
27                 found = true;
28             }
29         }
30         if (!found) {
31             log.info( msg: "Keine Produkte mit Attribut: " + keyword);
32         }
33     }
34     public void showAllProducts() { 3 usages
35         if (products.isEmpty()) {
36             log.info( msg: "Keine Produkte vorhanden.");
37         } else {
38             for (Product p : products.values()) {
39                 String preis = (p.getCurrentPrice() > 0)
40                     ? p.getCurrentPrice() + "€"
41                     : "n/a";
42                 log.info( msg: p.getName() + " - Aktueller Kurs: " + preis);
43             }
44         }
45     }
46
47     public void buyProduct(String name, int qty) { 4 usages
48         Product p = products.get(name.toLowerCase());
49         if (p == null) {
50             log.warning( msg: "Produkt '" + name + "' nicht gefunden.");
51             return;
52         }
53         p.processPurchase(qty);
54     }
55
56     public void showProductOffers(String name) { 3 usages
57         Product p = products.get(name.toLowerCase());
58         if (p == null) {
59             log.warning( msg: "Produkt '" + name + "' nicht gefunden.");
60             return;
61         }
62         p.showOffers();
63     }
64     public void showPriceHistory(String name) { 3 usages
65         Product p = products.get(name.toLowerCase());
66         if (p == null) {
67             log.warning( msg: "Produkt '" + name + "' nicht gefunden.");
68             return;
69         }
70         p.showPriceHistory();
71     }
72
73     public void showSellerOffers(String seller) { 3 usages
74         boolean found = false;
75         for (Product p : products.values()) {
76             List<Offer> offers = p.getOffersBySeller(seller);
77             if (!offers.isEmpty()) {
78                 log.info( msg: "Produkt: " + p.getName());
79                 for (Offer o : offers) {
80                     log.info( msg: " " + o);
81                 }
82                 found = true;
83             }
84         }
85         if (!found) {
86             log.info( msg: "Keine Angebote von '" + seller + "' gefunden.");
87         }
88     }
89     public void updateSellerOffer(String seller, String productName, double newPrice, int newQty,
90     ) {
91         Product p = products.get(productName.toLowerCase());
92         if (p == null) {
93             log.warning( msg: "Produkt nicht gefunden.");
94             return;
95         }
96         boolean success = p.updateOfferFromSeller(seller, newPrice, newQty);
97         if (success) {
98             log.info( msg: "Angebot aktualisiert.");
99         } else {
100            log.warning( msg: "Kein passendes Angebot gefunden.");
101        }
102    }
103
104    public void updateProductAttributes(String seller, String productName, String[] newAttrs) {
105        Product p = products.get(productName.toLowerCase());
106        if (p == null) {
107            log.warning( msg: "Produkt nicht gefunden.");
108            return;
109        }
110        p.setAttributes(Arrays.asList(newAttrs));
111        log.info( msg: "Attribut aktualisiert: " + String.join( delimiter: ", ", newAttrs));
112    }
113
114    public void removeSellerOffer(String seller, String productName) { 3 usages
115        Product p = products.get(productName.toLowerCase());
116        if (p == null) {
117            log.warning( msg: "Produkt nicht gefunden.");
118            return;
119        }
120        boolean removed = p.removeOfferFromSeller(seller);
121        if (removed) {
122            log.info( msg: "Angebot von '" + seller + "' entfernt.");
123        } else {
124            log.warning( msg: "Kein passendes Angebot gefunden.");
125        }
126    }

```

-@Log: Ermöglicht Nutzung von log.info(), log.warning() ohne manuelle Logger-Initialisierung.

-@Getter: Generiert automatisch Getter für alle Felder, insbesondere für products.

-Speichert alle Produkte in einer Map, wobei der Schlüssel der Produktnamen in Kleinbuchstaben ist.

-addOfferWithAttributes: Fügt ein neues Produkt (falls noch nicht vorhanden) mit Attributen hinzu und ergänzt ein Angebot dazu.

-searchByAttribute: Durchsucht alle Produkte nach einem bestimmten Attribut. Gibt Namen und Angebote gefundener Produkte aus.

-showAllProducts: Zeigt alle registrierten Produkte mit aktuellem Kurs an. Wenn keine Produkte vorhanden sind, folgt ein Hinweis.

-buyProduct: Führt einen Kaufvorgang für ein bestimmtes Produkt durch. Gibt eine Warnung aus, wenn das Produkt nicht existiert.

-showProductOffers: Zeigt alle Angebote für ein bestimmtes Produkt. Bei fehlendem Produkt erfolgt eine Warnmeldung.

-showPriceHistory: Zeigt den Verlauf der letzten Transaktionspreise eines Produkts. Gibt Warnung bei nicht vorhandenem Produkt.

-showSellerOffers: Listet alle Angebote eines bestimmten Verkäufers auf. Gibt Hinweis, falls keine gefunden wurden.

-updateSellerOffer: Erlaubt einem Verkäufer, Preis und Menge eines vorhandenen Angebots zu ändern. Gibt Erfolg oder Warnung zurück.

-updateProductAttributes: Aktualisiert die Attributliste eines Produkts. Meldet Fehler, falls das Produkt nicht existiert.

-removeSellerOffer: Löscht das Angebot eines Verkäufers für ein bestimmtes Produkt. Rückmeldung über Erfolg oder Misserfolg.

PRODUCT-KLASSE:

```

package de.mcc;

import lombok.Getter;
import lombok.extern.java.Log;

import java.util.*;

@Getter 18 usages
@Log
public class Product {
    private String name;
    private List<Offer> offers = new ArrayList<>();
    private List<Double> priceHistory = new ArrayList<>();
    private List<String> attributes = new ArrayList<>();

    public Product(String name, String... attrs) { 3 usages
        this.name = name;
        this.attributes.addAll(Arrays.asList(attrs));
    }

    public boolean matchesAttribute(String keyword) {...}

    public void addOffer(Offer offer) { 11 usages
        offers.add(offer);
        offers.sort(Comparator.comparingDouble(Offer::getPrice));
    }

    public void processPurchase(int quantity) { 4 usages
        Iterator<Offer> iterator = offers.iterator();
        int remaining = quantity;

        while (iterator.hasNext() && remaining > 0) {
            Offer offer = iterator.next();
            int available = offer.getQuantity();

            if (available <= remaining) {
                remaining -= available;
                priceHistory.add(offer.getPrice());
                iterator.remove();
            } else {
                offer.setQuantity(available - remaining);
                priceHistory.add(offer.getPrice());
                remaining = 0;
            }
        }

        if (remaining > 0) {
            log.warning(msg: "Nicht genug Ware verfügbar.");
        } else {
            log.info(msg: "Kauf erfolgreich.");
        }
    }

    public double getCurrentPrice() { return offers.isEmpty() ? -1 : offers.get(0).getPrice(); }

    public void showOffers() { 2 usages
        if (offers.isEmpty()) {
            log.info(msg: "Keine Angebote verfügbar.");
        } else {
            for (Offer o : offers) {
                log.info(o.toString());
            }
        }
    }

    public void showPriceHistory() { 1 usage
        int size = priceHistory.size();
        if (size == 0) {
            log.info(msg: "Keine Transaktionen vorhanden.");
            return;
        }
        int from = Math.max(size - 3, 0);
        List<Double> recent = priceHistory.sublist(from, size);
        log.info(msg: "Letzte Kurse: " + recent);
    }

    public List<Offer> getOffersBySeller(String seller) { 2 usages
        return offers.stream()
            .filter(o -> o.getSellerName().equalsIgnoreCase(seller))
            .toList();
    }

    public boolean updateOfferFromSeller(String seller, double newPrice, int newQty) { 3 usages
        for (Offer o : offers) {
            if (o.getSellerName().equalsIgnoreCase(seller)) {
                o.setPrice(newPrice);
                o.setQuantity(newQty);
                return true;
            }
        }
        return false;
    }

    public boolean removeOfferFromSeller(String seller) { 3 usages
        return offers.removeIf(o -> o.getSellerName().equalsIgnoreCase(seller));
    }

    public void setAttributes(List<String> newAttrs) { 2 usages
        this.attributes = new ArrayList<>(newAttrs);
    }

    @Override
    public String toString() {
        return name + "[Attribute: " + String.join(delimiter: ", ", attributes) + "]";
    }
}

```

-**@Log**: Aktiviert Logging-Funktionen (log.info(), log.warning()).
-@Getter: Generiert automatisch Getter für alle Felder (z. B. getName(), getAttributes()).
-name: Name des Produkts.
-offers: Liste aller Verkaufsangebote für dieses Produkt.
-priceHistory: Liste der letzten Preise aus durchgeführten Käufen.
-attributes: Liste der Produktattribute (z. B. „bio“, „regional“).
-Konstruktor: Initialisiert das Produkt mit Namen und beliebig vielen Attributen.
-matchesAttribute: Prüft, ob das Produkt ein bestimmtes Attribut besitzt (nicht case-sensitiv).
-addOffer: Fügt ein neues Angebot hinzu und sortiert die Angebotsliste nach Preis aufsteigend.

-processPurchase: Verarbeitet einen Kaufvorgang: reduziert Mengen, speichert gekaufte Preise, gibt Feedback über Erfolg oder Misserfolg.
-getCurrentPrice: Gibt den günstigsten aktuellen Preis zurück, oder -1, wenn keine Angebote vorhanden sind.
-showOffers: Gibt alle aktuellen Angebote dieses Produkts aus. Wenn keine vorhanden, folgt eine Info-Meldung.

-showPriceHistory: Zeigt die letzten drei Verkaufspreise des Produkts an. Gibt Meldung bei fehlender Historie.
-getOffersBySeller: Filtert alle Angebote, die von einem bestimmten Verkäufer stammen.
-updateOfferFromSeller: Ändert Preis und Menge eines vorhandenen Angebots des Verkäufers. Gibt Erfolg/Misserfolg zurück.
-removeOfferFromSeller: Entfernt das Angebot eines bestimmten Verkäufers. Gibt true zurück, wenn etwas gelöscht wurde.

-setAttributes: Überschreibt die aktuellen Attribute mit einer neuen Liste.
-toString: Gibt den Produktnamen und die zugehörigen Attribute als Zeichenkette zurück.

```

package de.mcc;
OFFER-KLASSE:
import lombok.AllArgsConstructor;
import lombok.Getter;
import lombok.Setter;

@Getter 33 usages
@Setter
@AllArgsConstructor
public class Offer {
    private String sellerName;
    private double price;
    private int quantity;

    @Override
    public String toString() {
        return quantity + " Stück von " + sellerName + " Price: " + price + "€";
    }
}

```

-@Getter: Generiert Getter für alle Felder (getSellerName(), getPrice(), getQuantity()).
-@Setter: Generiert Setter für alle Felder.
-@AllArgsConstructor: Erzeugt automatisch einen Konstruktor mit allen Feldern.
-sellerName: Name des Verkäufers.
-price: Preis pro Stück des Angebots.
-quantity: Verfügbare Menge des Produkts.

ZUSAMMENFASSUNG:

Die Klasse Market verwaltet alle Produkte im System und ist der zentrale Zugangspunkt für Käufer und Verkäufer. Jedes Produkt wird durch die Klasse Product repräsentiert, die eine Liste von Offer-Objekten enthält – also konkreten Verkaufsangeboten eines Verkäufers mit Preis und Menge. Market ruft Methoden aus Product auf, um Angebote hinzuzufügen, Käufe zu verarbeiten oder nach Attributen zu suchen. Die Klasse Offer speichert dabei die Details eines einzelnen Verkaufs.

ANFORDERUNG 1:

```

public void showAllProducts() { 3 usages
    if (products.isEmpty()) {
        log.info(msg: "Keine Produkte vorhanden.");
    } else {
        for (Product p : products.values()) {
            String preis = (p.getCurrentPrice() > 0)
                ? p.getCurrentPrice() + "€"
                : "n/a";
            log.info(msg: p.getName() + " - Aktueller Kurs: " + preis);
        }
    }
}

```

```

==== Käufermenü ====
1. Alle Produkte anzeigen
2. Produkt kaufen
3. Angebote zu einem Produkt anzeigen
4. Kursverlauf anzeigen
5. Nach Attribut suchen
0. Zurück

Auswahl:
1
Pfirsche - Aktueller Kurs: 0.2€
Apfel - Aktueller Kurs: 0.99€

```

Der Käufer kann sich alle auf dem Marktplatz verfügbaren Produkte anzeigen lassen. Falls keine Produkte vorhanden sind, wird eine entsprechende Nachricht ausgegeben.

ANFORDERUNG 2:

```

public void searchByAttribute(String keyword) { 3 usages
    boolean found = false;
    for (Product p : products.values()) {
        if (p.matchesAttribute(keyword)) {
            log.info(msg: p.getName() + " → Attribute: " + String.join(delimiter: ", ", p.getAttributes()));
            p.showOffers();
            found = true;
        }
    }
    if (!found) {
        log.info(msg: "Keine Produkte mit Attribut: " + keyword);
    }
}

```

```

==== Käufermenü ====
1. Alle Produkte anzeigen
2. Produkt kaufen
3. Angebote zu einem Produkt anzeigen
4. Kursverlauf anzeigen
5. Nach Attribut suchen
0. Zurück

Auswahl:
5
Attribut eingeben:
bio
Apfel → Attribute: bio, frucht
20 Stück von Jo Price: 1.0€
Gurke → Attribute: gemuse, bio
10 Stück von Jo Price: 2.0€

```

Der Käufer hat die Möglichkeit, Produkte anhand eines bestimmten Attributs zu suchen. Wenn ein Produkt dieses Attribut besitzt, wird es zusammen mit seinen Angeboten angezeigt. Andernfalls erhält der Käufer eine Mitteilung, dass kein entsprechendes Produkt gefunden

ANFORDERUNG 3:

```

public void addOfferWithAttributes(String productName, Offer offer, String[] attrs) {
    String key = productName.toLowerCase();
    if (!products.containsKey(key)) {
        products.put(key, new Product(productName, attrs));
    }
    products.get(key).addOffer(offer);
}

```

```

==== Verkäufermenü ====
1. Neues Angebot hinzufügen
2. Eigene Angebote anzeigen
3. Preis/Menge ändern
4. Attribute ändern
5. Angebot löschen
6. Zurück

Auswahl:
1
Produktname:
Birne
Menge:
99      Birne - Aktueller Kurs: 0.5€
Preis:    Apfel - Aktueller Kurs: 1.0€
0.5      Gurke - Aktueller Kurs: 2.0€
Attribute (kommagetrennt):
frucht
Angebot hinzugefügt.

Produkt: Birne
80 Stück von Jo Price: 0.5€

Birne - Aktueller Kurs: n/a

```

Der Verkäufer hat die Möglichkeit, ein neues Produkt samt Angebot auf die Börse zu stellen. Wenn das Produkt noch nicht existiert, wird es mit den angegebenen Attributen angelegt und anschließend das Angebot hinzugefügt. Nach dem Kauf eines Produkts wird die verfügbare Menge automatisch entsprechend reduziert oder das Angebot vollständig entfernt, wenn die Menge aufgebraucht ist.

ANFORDERUNG 4:

```
public void updateSellerOffer(String seller, String productName, double newPrice, int newQty)
    Product p = products.get(productName.toLowerCase());
    if (p == null) {
        log.warning("Produkt nicht gefunden.");
        return;
    }
    boolean success = p.updateOfferFromSeller(seller, newPrice, newQty);
    if (success) {
        log.info("Angebot aktualisiert.");
    } else {
        log.warning("Kein passendes Angebot gefunden.");
    }
}
```

```
Produkt: Apfel
  999 Stück von Jo Price: 5.0€
Produkt: Gurke
  10 Stück von Jo Price: 2.0€
Produktname:
Apfel
Neuer Preis:
1
Neue Menge:
20
Angebot aktualisiert.
Produkt: Apfel
  20 Stück von Jo Price: 1.0€
Produkt: Gurke
  10 Stück von Jo Price: 2.0€
  
```

Der Verkäufer hat die Möglichkeit, sowohl den Preis als auch die Menge seiner angebotenen Waren anzupassen. Zusätzlich kann er auch die Attribute eines Produkts ändern.

ANFORDERUNG 5:

```
public void buyProduct(String name, int qty) { 4 usages
    Product p = products.get(name.toLowerCase());
    if (p == null) {
        log.warning("Produkt '" + name + "' nicht gefunden.");
        return;
    }
    p.processPurchase(qty);
}
```

```
public void processPurchase(int quantity) { 4 usages
    Iterator<Offer> iterator = offers.iterator();
    int remaining = quantity;

    while (iterator.hasNext() && remaining > 0) {
        Offer offer = iterator.next();
        int available = offer.getQuantity();

        if (available <= remaining) {
            remaining -= available;
            priceHistory.add(offer.getPrice());
            iterator.remove();
        } else {
            offer.setQuantity(available - remaining);
            priceHistory.add(offer.getPrice());
            remaining = 0;
        }
    }

    if (remaining > 0) {
        log.warning("Nicht genug Ware verfügbar.");
    } else {
        log.info("Kauf erfolgreich.");
    }
}
```

```
Birne - Aktueller Kurs: n/a
Apfel - Aktueller Kurs: 1.0€
Gurke - Aktueller Kurs: 2.0€
Produktname:
Gurke
Menge:
50
Kauf erfolgreich.
Birne - Aktueller Kurs: n/a
Apfel - Aktueller Kurs: 1.0€
Gurke - Aktueller Kurs: 3.0€
  
```

```
Produktname:
Gurke
Menge:
89
Preis:
3
Attribute (kommagetrennt):
bio,gold
Angebot hinzugefügt.
  
```

Transaktionen auf dem Marktplatz folgen einer einfachen Logik: Der Kauf eines Produkts erfolgt stets zum günstigsten verfügbaren Angebot. Die aktuelle Marktpreisangabe entspricht daher dem niedrigsten Preis unter allen aktiven Angeboten. Wenn alle Einheiten eines Angebots zu diesem Preis verkauft werden, verschwindet dieses Angebot aus dem System, und das nächsthöhere Angebot wird zur neuen aktuellen Preisbasis. So passt sich der Marktpreis dynamisch an das verfügbare Angebot an.

ANFORDERUNG 6:

```
public void showPriceHistory(String name) { 3 usages
    Product p = products.get(name.toLowerCase());
    if (p == null) {
        log.warning("Produkt '" + name + "' nicht gefunden.");
        return;
    }
    p.showPriceHistory();
}

public void showPriceHistory() { 1 usage
    int size = priceHistory.size();
    if (size == 0) {
        log.info("Keine Transaktionen vorhanden.");
        return;
    }
    int from = Math.max(size - 3, 0);
    List<Double> recent = priceHistory.subList(from, size);
    log.info("Letzte Kurse: " + recent);
}

public List<Offer> getOffersBySeller(String seller) { 2 usages
    return offers.stream()
        .filter(o -> o.getSellerName().equalsIgnoreCase(seller))
        .toList();
}
```

```
== Käufermenü ==
1. Alle Produkte anzeigen
2. Produkt kaufen
3. Angebote zu einem Produkt anzeigen
4. Kursverlauf anzeigen
5. Nach Attribut suchen
0. Zurück

Auswahl:
4
Produktname:
Gurke
Letzte Kurse: [2.0, 3.0, 2.3]
  
```

Der Käufer kann sich die letzten drei Preisänderungen eines Produkts anzeigen lassen. Dabei wird der Verlauf der letzten Transaktionen ausgegeben, sofern mindestens eine stattgefunden hat.

TESTS:

```

    ✓ MarketTest (de.mcc) 39 ms
      ✓ Tests passed: 19 of 19 tests – 39 ms
        INFO: Apfel - Aktueller Kurs: 1.0€
        мая 28, 2025 12:02:23 AM de.mcc.Market searchByAttribute
        INFO: Keine Produkte mit Attribut: fleisch
        мая 28, 2025 12:02:23 AM de.mcc.Market removeSellerOffer
        INFO: Angebot von 'Sophie' entfernt.
        мая 28, 2025 12:02:23 AM de.mcc.Market updateSellerOffer
        INFO: Angebot aktualisiert.
        мая 28, 2025 12:02:23 AM de.mcc.Market removeSellerOffer
        WARNING: Produkt nicht gefunden.
        мая 28, 2025 12:02:23 AM de.mcc.Market buyProduct
        WARNING: Produkt 'Zitrone' nicht gefunden.
        мая 28, 2025 12:02:23 AM de.mcc.Market updateSellerOffer
        WARNING: Produkt nicht gefunden.
        мая 28, 2025 12:02:23 AM de.mcc.Market updateProductAttributes
        INFO: Attribut aktualisiert: langkorn, bio
        мая 28, 2025 12:02:23 AM de.mcc.Product processPurchase
        INFO: Kauf erfolgreich.
        мая 28, 2025 12:02:23 AM de.mcc.Market showAllProducts
        INFO: Keine Produkte vorhanden.
        мая 28, 2025 12:02:23 AM de.mcc.Market showSellerOffers
        INFO: Keine Angebote von 'Unbekannt' gefunden.
        мая 28, 2025 12:02:23 AM de.mcc.Product showOffers
        INFO: 6 Stück von John Doe Price: 2.5€
        мая 28, 2025 12:02:23 AM de.mcc.Market searchByAttribute
        INFO: Birne → Attribute: bio
        мая 28, 2025 12:02:23 AM de.mcc.Product showOffers
        INFO: 5 Stück von John Doe Price: 1.5€
        мая 28, 2025 12:02:23 AM de.mcc.Market showPriceHistory
        WARNING: Produkt 'Ananas' nicht gefunden.

      Process finished with exit code 0
  
```

```

    ✓ ProductTest (de.mcc) 28 ms
      ✓ Tests passed: 11 of 11 tests – 28 ms
        /Users/soromori/Library/Java/JavaVirtualMachines/openjdk-23.0.1-fcs/jdk-23.0.1/bin/java -Dfile.encoding=UTF-8 -jar "/Users/soromori/Desktop/Marketplace/de.mcc/target/de.mcc-1.0-SNAPSHOT.jar"
        INFO: Kauf erfolgreich.
        мая 28, 2025 12:03:40 AM de.mcc.Product processPurchase
        INFO: Kauf erfolgreich.
        мая 28, 2025 12:03:40 AM de.mcc.Product processPurchase
        WARNING: Nicht genug Ware verfügbar.

      Process finished with exit code 0
  
```

Das Projekt ist vollständig durch Unit-Tests abgedeckt. Dabei werden zentrale Methoden wie das Hinzufügen von Angeboten, Käufe, Attributsuche sowie Preis- und Mengenänderungen getestet. Es wurden sowohl Standardfälle als auch Grenzfälle berücksichtigt, um die Funktionsweise unter verschiedenen Bedingungen sicherzustellen.

@BeforeEach

Diese Methode wird vor jedem einzelnen Test ausgeführt. Sie initialisiert die notwendigen Objekte (z. B. Product, Market, Offer) und stellt sicher, dass jeder Test mit einer frischen Ausgangssituation beginnt.

@Test

Jeder Test prüft gezielt eine Methode der Anwendung, z. B. das Hinzufügen von Angeboten, das Durchführen von Käufen oder die Attributsuche. Über assertEquals, assertTrue, assertFalse usw. werden Standard- und Randfälle getestet.

-Product:

matchesAttribute, addOffer, processPurchase, getCurrentPrice, getOffersBySeller, updateOfferFromSeller, removeOfferFromSeller, setAttributes, toString

-Market:

addOfferWithAttributes, searchByAttribute, showAllProducts, buyProduct, showProductOffers, showPriceHistory, showSellerOffers, updateSellerOffer, updateProductAttributes, removeSellerOffer

```

public class ProductTest {

    private Product product; 33 usages
    private Offer offer1; 6 usages
    private Offer offer2; 4 usages

    @BeforeEach
    void setUp() {
        product = new Product(name: "Apfel", ...attrs: "obst", "bio");
        offer1 = new Offer(sellerName: "Seller1", price: 1.0, quantity: 5);
        offer2 = new Offer(sellerName: "Seller2", price: 1.5, quantity: 3);
    }

    @Test
    void testMatchesAttribute() {
        assertTrue(product.matchesAttribute(keyword: "Bio"));
        assertFalse(product.matchesAttribute(keyword: "fleisch"));
    }

    @Test
    void testAddOffer() {
        product.addOffer(offer2); // 1.5
        product.addOffer(new Offer(sellerName: "Seller3", price: 3.1, quantity: 20));
        assertEquals(expected: 1.5, product.getCurrentPrice());
    }
}
  
```

```

@Test
void testProcessPurchaseCases() {
    product.addOffer(offer1);
    product.addOffer(offer2);

    // Standardfall
    product.processPurchase(quantity: 4);
    assertEquals(expected: 1, product.getOffers().get(0).getQuantity());
    assertEquals(expected: 1.0, product.getCurrentPrice());

    // Grenzfall
    product.processPurchase(quantity: 4);
    assertEquals(expected: -1, product.getCurrentPrice());

    // NichtGenugFall
    product.addOffer(new Offer(sellerName: "Seller1", price: 2.0, quantity: 2));
    product.processPurchase(quantity: 5);
    assertEquals(expected: -1, product.getCurrentPrice());
}

@Test
void testGetCurrentPriceEmpty() {
    assertEquals(expected: -1, product.getCurrentPrice());
}

@Test
void testGetOffersBySeller() {
    product.addOffer(offer1);
    product.addOffer(offer2);
    List<Offer> seller1Offers = product.getOffersBySeller("Seller1");
    assertEquals(expected: 1, seller1Offers.size());
    assertEquals(expected: 1.0, seller1Offers.get(0).getPrice());
}
  
```

```

    @Test
    void testUpdateOfferFromSeller() {
        product.addOffer(offer1);
        boolean updated = product.updateOfferFromSeller("Seller1", newPrice: 3.0, newQty: 10);
        assertTrue(updated);
        Offer updatedOffer = product.getOffers().get(0);
        assertEquals(expected: 3.0, updatedOffer.getPrice());
        assertEquals(expected: 10, updatedOffer.getQuantity());
    }

    @Test
    void testUpdateOfferFromSellerNotFound() {
        boolean updated = product.updateOfferFromSeller("Unbekannt", newPrice: 2.0, newQty: 5);
        assertFalse(updated);
    }

    @Test
    void testRemoveOfferFromSeller() {
        product.addOffer(offer1);
        boolean removed = product.removeOfferFromSeller("Seller1");
        assertTrue(removed);
        assertEquals(product.getOffers().isEmpty());
    }

    @Test
    void testRemoveOfferFromSellerNotFound() {
        product.addOffer(offer1);
        boolean removed = product.removeOfferFromSeller("Nobody");
        assertFalse(removed);
    }
}

```

```

void testShowAllProductsMitProdukt() {
    market.addOfferWithAttributes(productName: "Apfel", new Offer(sellerName: "John Doe", price: 1.0, quantity: 3), new String[]{"obst", "bio"});
    market.showAllProducts();
}

@Test
void testShowAllProductsOhneProdukte() {
    market.showAllProducts();
}

@Test
void testBuyProductErfolgreich() {
    market.addOfferWithAttributes(productName: "Banane", new Offer(sellerName: "John Doe", price: 1.0, quantity: 4), new String[]{"frucht", "obst", "natur"});
    market.buyProduct(name: "Banane", qty: 2);
    Product p = market.getProducts().get("banane");
    assertEquals(expected: 2, p.getOffers().get(0).getQuantity());
}

@Test
void testBuyProductNichtGefunden() {
    market.buyProduct(name: "Zitrone", qty: 1);
}

@Test
void testShowProductOffersGefunden() {
    market.addOfferWithAttributes(productName: "Mango", new Offer(sellerName: "John Doe", price: 2.5, quantity: 6), new String[]{"exotisch", "obst", "frucht"});
    market.showProductOffers(name: "Mango");
}

@Test
void testShowProductOffersNichtGefunden() {
    market.showProductOffers(name: "Papaya");
}

```

```

    @Test
    void testUpdateSellerOfferNichtGefunden() {
        market.updateSellerOffer(seller: "Karl", productName: "Brot", newPrice: 2.0, newQty: 10);
    }

    @Test
    void testUpdateProductAttributesErfolgreich() {
        market.addOfferWithAttributes(productName: "Reis", new Offer(sellerName: "Markus", price: 1.5, quantity: 3), new String[]{"korn", "langkorn", "bio"});
        market.updateProductAttributes(seller: "Markus", productName: "Reis", new String[]{"langkorn", "bio"});
        Product p = market.getProducts().get("reis");
        assertTrue(p.matchesAttribute(keyword: "bio"));
    }

    @Test
    void testUpdateProductAttributesNichtGefunden() {
        market.updateProductAttributes(seller: "Nina", productName: "Milch", new String[]{"weib"});
    }

    @Test
    void testRemoveSellerOfferErfolgreich() {
        market.addOfferWithAttributes(productName: "Wasser", new Offer(sellerName: "Sophie", price: 0.5, quantity: 8), new String[]{"getränk", "wasser"});
        market.removeSellerOffer(seller: "Sophie", productName: "Wasser");
        Product p = market.getProducts().get("wasser");
        assertTrue(p.getOffers().isEmpty());
    }

    @Test
    void testRemoveSellerOfferNichtGefunden() {
        market.removeSellerOffer(seller: "X", productName: "Cola");
    }
}

```

```

    @Test
    void testSetAttributes() {
        product.setAttributes(List.of("regional", "saison"));
        assertTrue(product.matchesAttribute(keyword: "regional"));
        assertFalse(product.matchesAttribute(keyword: "bio"));
    }

    @Test
    void testToStringContainsAttributes() {
        String result = product.toString();
        assertTrue(result.contains("obst"));
        assertTrue(result.contains("bio"));
        assertTrue(result.contains("Apfel"));
    }
}

```

```

public class MarketTest {
    private Market market; 35 usages

    @BeforeEach
    void setup() {
        market = new Market();
    }

    @Test
    void testAddOfferWithAttributes() {
        Offer offer = new Offer(sellerName: "John Doe", price: 2.0, quantity: 10);
        market.addOfferWithAttributes(productName: "Apfel", offer, new String[]{"obst", "bio"});
        Product p = market.getProducts().get("Apfel");
        assertNotNull(p);
        assertEquals(expected: 1, p.getOffers().size());
        assertTrue(p.matchesAttribute(keyword: "obst"));
    }

    @Test
    void testSearchByAttributeMitTreffer() {
        Offer offer = new Offer(sellerName: "John Doe", price: 1.5, quantity: 5);
        market.addOfferWithAttributes(productName: "Birne", offer, new String[]{"bio"});
        market.searchByAttribute(keyword: "bio");
    }

    @Test
    void testSearchByAttributeOhneTreffer() {
        market.searchByAttribute(keyword: "fleisch");
    }
}

```

```

void testShowPriceHistoryGefunden() {
    market.addOfferWithAttributes(productName: "Kiwi", new Offer(sellerName: "Jane Doe", price: 1.0, quantity: 2), new String[]{"gruen", "obst", "frucht"});
    market.buyProduct(name: "Kiwi", qty: 2);
    market.showPriceHistory(name: "Kiwi");
}

@Test
void testShowPriceHistoryNichtGefunden() {
    market.showPriceHistory(name: "Ananas");
}

@Test
void testShowSellerOffersMitTreffer() {
    market.addOfferWithAttributes(productName: "Pfirsich", new Offer(sellerName: "Jane Doe", price: 2.0, quantity: 7), new String[]{"obst", "frucht", "frucht"});
    market.showSellerOffers("Jane Doe");
}

@Test
void testShowSellerOffersOhneTreffer() {
    market.showSellerOffers("Unbekannt");
}

@Test
void testUpdateSellerOfferErfolgreich() {
    market.addOfferWithAttributes(productName: "Tomate", new Offer(sellerName: "Jo", price: 1.0, quantity: 5), new String[]{"gemuese", "obst", "frucht"});
    market.updateSellerOffer(seller: "Jo", productName: "Tomate", newPrice: 2.0, newQty: 10);
    Product p = market.getProducts().get("tomate");
    Offer o = p.getOffers().get(0);
    assertEquals(expected: 2.0, o.getPrice());
    assertEquals(expected: 10, o.getQuantity());
}

```

TESTABDECKUNG:

Coverage MarketTest				
Element ^	Clas...	Metho...	Line, %	Branc...
de.mcc	60% (... 83% (30... 55% (10... 60% (4...			
>Main	0% (0... 0% (0/5) 0% (0/80) 0% (0/17)			
Market	100%... 100% (1... 95% (5... 84% (3...			
Offer	100%... 100% (6... 100% (6... 100% (0...			
Product	100%... 92% (13... 89% (4... 65% (13...			

Coverage ProductTest				
Element ^	Clas...	Metho...	Line, %	Branc...
de.mcc	40% (... 44% (16... 21% (43... 17% (13...			
>Main	0% (0... 0% (0/5) 0% (0/80) 0% (0/17)			
Market	0% (0... 0% (0/11) 0% (0/61) 0% (0/38)			
Offer	100%... 83% (5/6) 83% (5/6) 100% (0...			
Product	100%... 78% (11/... 77% (38... 65% (13...			

Ich habe für alle Methoden, bei denen es sinnvoll möglich war, passende Testmethoden geschrieben – meist mit Abdeckung sowohl eines Standardfalls als auch eines Grenzfalls. Die Methoden in der Main-Klasse wurden dabei nicht getestet, da sie stark von der Benutzereingabe über die Konsole abhängig sind. Zudem erzeugen sie Konsolenausgaben, was automatisierte Tests zusätzlich erschwert. Daher wurde auf Unit-Tests für diese Methoden bewusst verzichtet.

ALLGEMEINE INFORMATION:

```
static {
    for (var handler : java.util.logging.Logger.getLogger(name: "").getHandlers()) {
        handler.setFormatter(new java.util.logging.Formatter() {
            @Override
            public String format(java.util.logging.LogRecord record) {
                return record.getMessage() + "\n";
            }
        });
    }
}
```

Mit Hilfe von Statischen Block wurde das Standard-Format der Java-Logger-Ausgabe angepasst, sodass keine Zeit-, Datums- oder Klassennameninformationen mehr angezeigt werden. Stattdessen wird nur die eigentliche Log-Nachricht ausgegeben. Das sorgt für eine übersichtlichere Darstellung in der Konsole.

- Flexible Dezimaltrennung: Beim Preiseingabefeld akzeptiert das System sowohl Punkt („1.5“) als auch Komma („1,5“) als Dezimaltrennzeichen.
- Fehlertolerante Eingabe: Bei ungültigen Eingaben (z. B. falsches Zahlenformat) wird der Nutzer solange zur Wiederholung aufgefordert, bis eine gültige Eingabe erfolgt.
- Mehrfachattributeingabe: Attribute für ein Produkt müssen als durch Kommas getrennte Liste eingegeben werden (z. B. bio, regional, frisch).