



Technische Universität Berlin

Software and Embedded Systems Engineering Group

Prof. Dr. Sabine Glesner

www.sese.tu-berlin.de    Sekr. TEL 12-4    Ernst-Reuter-Platz 7    10587 Berlin



# Softwaretechnik und Programmierparadigmen WiSe 2023/2024

Prof. Dr. Sabine Glesner  
Simon Schwan  
Julian Klein

## Übungsblatt 9

Beispiellösung

### Anforderungstext

Die Funktion `createOrder(int customerID, int productID, Instant created)` soll im System einen Auftrag erstellen. Als erstes wird geprüft, ob die Kunden- und Kundinnennummer (`customerID`) im System vorhanden ist. Ist dies nicht der Fall wird nur die Fehlermeldung „Customer nicht im System“ ausgegeben. Andernfalls wird die Warennummer (`productID`) geprüft. Existiert die Warennummer nicht im System, wird nur die Fehlermeldung „Es existiert keine Ware mit der gegebenen Warennummer“ ausgegeben. Ansonsten wird geprüft, ob ein Erstelldatum (`created`) angegeben wurde. Ist dies nicht der Fall wird nur die Meldung „Kein Erstellungsdatum angegeben“ ausgegeben. Sowohl `customerID` als auch `productID` werden als fortlaufende Nummern beginnend bei 0 vergeben und zur Vereinfachung gehen wir davon aus, dass einmal angelegte Kunden/Kundinnen und Waren niemals aus dem System gelöscht werden. Wurden alle Daten korrekt eingegeben, wird ein Auftrag im System erstellt.

### Aufgabe 1: Funktionsorientierter Test

Ermittelt mit den folgenden funktionsorientierten Verfahren Testfälle:

- a) Äquivalenzklassenbildung
- b) Grenzwertanalyse
- c) Entscheidungstabellentest
- d) optimierte Entscheidungstabelle



## Lösung:

a) Äquivalenzklassenbildung:

customerID	$] \infty, 0[$ , $[0, \#customers[$ , $[\#customers, \infty[$
productID	$] \infty, 0[$ , $[0, \#products[$ , $[\#products, \infty[$
created	<i>null, !null</i>

Wobei  $\#customer$  und  $\#products$  die Anzahlen von Kunden/Kundinnen und Waren im System sind.

b) Grenzwertanalyse:

Für die ersten beiden jeweils:  $(-1, 0)$  für beide;  $(\#prod-1, \#prod)$ ;  $(\#cust-1, \#cust)$ . Bei **created** gibt es eigentlich keine Grenzwerte, bzw. sind alle Werte Grenzwerte (**null** ist ein Nachbarwert für alle andere Werte) und die Analyse bringt uns nicht weiter als beliebige Representanten.

```
// #cust=6, #prod=6
createOrder(-1,1,nextWeek); // customerID
createOrder(0, 1, nextWeek); // customerID
createOrder(5, 1, nextWeek); // customerID
createOrder(6, 1, nextWeek); // customerID
createOrder(3, 0, nextWeek); // productID
createOrder(3, -1, nextWeek); // productID
createOrder(3, 5, nextWeek); // productID
createOrder(3, 6, nextWeek); // productID
```

c) Entscheidungstabelle:

	R1	R2	R3	R4	R5	R6	R7	R8
CustomerID ungültig?	N	J	N	J	N	J	N	J
Warennummer ungültig?	N	N	J	J	N	N	J	J
Erstelldatum ungültig?	N	N	N	N	J	J	J	J
„Kunde/Kundin nicht im System“	N	J	N	J	N	J	N	J
„Es existiert keine Ware...“	N	N	J	N	N	N	J	N
„Kein Erstelldatum angegeben“	N	N	N	N	J	N	N	N
Auftrag erstellt	J	N	N	N	N	N	N	N

d) optimierte Entscheidungstabelle:

Die Testfälle lassen sich zusammenfassen. Da im Fehlerfall nicht weiter ausgewertet wird, sind die folgenden Parameter egal.

	R1	R2468	R37	R5
CustomerID ungültig?	N	J	N	N
Warennummer ungültig?	N	-	J	N
Erstelldatum ungültig?	N	-	-	J
„CustomerID nicht im System“	N	J	N	N
„Es existiert keine Ware...“	N	N	J	N
„Kein Erstelldatum angegeben“	N	N	N	J
Auftrag erstellt	J	N	N	N

Beispielhafte Testfälle:

```
// #cust=6, #prod=6, now:Instant
R1:   createOrder(0, 0, now); // Eingaben gültig
R2468: createOrder(8, 0, now); // #cust
R37:  createOrder(0, -5, now); // #prod
R5:   createOrder(0, 0, null); // #date
```






## Implementierung

Ihr habt nun den Code der Funktion erhalten. Dabei stellt ihr fest, dass die Kunden und Kundinnen für bestimmte Waren Rabatte sammeln können. Handelt es sich bei den Bestellenden um einen Premium-Kunden oder eine Premium-Kundin und bei der bestellten Ware um rabattierfähige Ware, so wird der Preis der Ware um alle bisher gesammelten Rabatte des Kunden und der Kundin reduziert. Zu bestimmten Jubiläumsaktionen werden auch bei allen anderen Kunden und Kundinnen und rabattierfähiger Ware die bisher gesammelten Rabatte eingelöst.

```
1 public Order createOrder(int customerID, int productID,
2     Instant created) throws ShopInputException {
3
4     // handle invalid inputs
5     if(!customers.containsKey(customerID)){
6         throw new ShopInputException(
7             ShopInputException.INVALID_CUST_NR);
8     }
9     if(!products.containsKey(productID)){
10        throw new ShopInputException(
11            ShopInputException.INVALID_PROD_NR);
12    }
13    if(created == null){
14        throw new ShopInputException(
15            ShopInputException.INVALID_DATE);
16    }
17
18    // get costumer and product for given IDs
19    Customer c = customers.get(customerID);
20    Product p = products.get(productID);
21    // create new order
22    Order order = new Order(customerID, productID,
23        p.getPrice(), created);
24
25    if((c.isPriorityCustomer() & p.isDiscountable())
26        || isAnniversary(created)) {
27        order.setDiscount(true);
28        while(!c.getDiscount().isEmpty()) {
29            order.discount(c.getDiscount().removeFirst());
30        }
31    }
32    return order;
33 }
```

## Aufgabe 2: Strukturorientierter Test

Für diese Aufgabe kann die Code-Vorgabe um passende JUnit-Tests erweitert werden. Die Anweisung- und Zweigüberdeckung können mit dem EcEmma Java Code Coverage Plug-In überprüft werden.

- a) Installiert und testet die Eclipse Plugins EcEmma und Metrics mit der Code-Vorlage. 
- b) Ermittelt die mit den bisherigen Testfällen erreichte Anweisungsüberdeckung. Erstellt gegebenenfalls weitere Testfälle bis 100% Anweisungsüberdeckung erreicht ist. 
- c) Ermittelt die mit den bisherigen Testfällen erreichte Entscheidungsüberdeckung (Zweigüberdeckung). Falls diese unter 100% liegt, erarbeitet weitere Testfälle um vollständige Entscheidungsüberdeckung zu erreichen. 
- d) Ermittelt die mit den bisherigen Testfällen erreichte (einfache) Bedingungsüberdeckung. Falls diese unter 100% liegt, erarbeitet weitere Testfälle um vollständige Bedingungsüberdeckung zu erreichen. 
- e) Ermittelt die mit den bisherigen Testfällen erreichte Pfadüberdeckung. Falls diese unter 100% liegt, erarbeitet weitere Testfälle um vollständige Pfadüberdeckung zu erreichen. 

**Lösung:** Siehe JUnit-Tests (Zip-Datei auf ISIS verfügbar).

- b) Nicht vollständig, da keine Rabatte gewährt wurden. Knoten 28–30 fehlen. Weiterer Testfall R8: Prioritäts-Kunden/Kundinnen mit Discountmarken kauft nicht rabattierfähiges Produkt, es ist aber Jubiläum.
- c) Nicht vollständig: Kante von 26 nach 28 ist noch nicht abgedeckt. Weiterer Testfall R9: Prioritäts-Kunden/Kundinnen mit Discountmarken kauft rabattierfähiges Produkt.
- d) Mit den bestehenden Testfällen ist die einfache Bedingungsüberdeckung und die minimale Mehrfach-Bedingungsüberdeckung erfüllt. Schon wegen der Zweigüberdeckung. Das geht nur, weil die Tests R1–R8 mit *nicht* rabattierfähigen Produkten formuliert wurden, sonst würde `!c.isDiscountable()` fehlen.  
Eine vollständige Mehrfach-Bedingungsüberdeckung wird aber nicht erfüllt. Es fehlen z.B. Fälle mit `c.isPriorityCustomer()`, `c.isDiscountable()` und `isAnniversary(created)`. Ob die zusätzlichen Tests für die Mehrfach-Bedingungsüberdeckung was bringen würden ist fragwürdig.
- e) Pfadüberdeckung: nicht möglich, da unendlich viele Testfälle nötig um theoretisch unendlich viele discounts zu berücksichtigen.