Technische Universität Berlin
Software Engineering for Embedded Systems Group – Prof. Dr. Sabine Glesner
www.pes.tu-berlin.de Secr. TEL 12-4 Ernst-Reuter-Platz 7 10587 Berlin

Softwaretechnik und Programmierparadigmen WiSe 2014/2015

Prof. Dr. Sabine Glesner
Joachim Fellmuth joachim.fellmuth@tu-berlin.de
Dr. Thomas Göthel thomas.goethel@tu-berlin.de
Lydia Mattick lydia.mattick@tu-berlin.de
Tutoren

Übungsblatt 13 **Ausgabe: 22.01. (Besprechung: 26.01. und 27.01.)**

Einführung Java Exception

Exceptions sind in Java ein wichtiges Mittel der Fehlerbehandlung. Würde ein Programm in einen undefinierten Zustand geraten, kann eine Exception geworfen (throw) werden. Ein Beispiel dafür ist eine Division durch 0 oder der Zugriff auf eine nicht initialisierte (null) Variable. Funktionen, in denen eine solche Exception auftreten kann, können „markiert“ werden. Mit dem Begriff „throws“ kann man bei Funktionen darstellen, dass eventuell bei der Ausführung eine bestimmte Exception geworfen wird.

```
public int divide(int x, int y) throws IllegalArgumentException
```

Wenn man weiß, dass solche Fehler auftreten könnten, kann man sie entsprechend behandeln. Dazu existieren sogenannte „try/catch“ Blöcke. Funktionen, in denen ein Fehler auftreten kann, werden in einem try-Block ausgeführt, auf den ein catch-block für eine eventuelle Fehlerbehandlung folgt.

```
try{
  return divide(a,b);
}catch(IllegalArgumentException e){
  System.out.println("'Division durch null!"');
  return -1;
}
```

Wird nun beim Ausführen der Funktion die Exception geworfen, werden die im catch-block definierten Anweisungen ausgeführt. Das Programm kann dann normal weiterlaufen.

1. Black-Box-Verfahren

Ihr arbeitet für einen professionellen Dienstleister für Softwaretests und bekommt den Auftrag, Testfälle für eine Funktion zu erstellen. Die Implementierung dieser Funktion ist euch nicht bekannt, ihr habt nur die folgende Beschreibung:

Die Funktion `createOrder(int customerNr, int productNr, Date deadline)` soll im System einen Auftrag erstellen. Als erstes wird geprüft, ob die Kundennummer im System vorhanden ist. Ist dies nicht der Fall wird die Fehlermeldung „Kunden nicht im System“ ausgegeben. Anschließend wird die Warennummer geprüft. Existiert die Warennummer nicht im System, wird die Fehlermeldung „Es existiert keine Ware mit der gegebenen Warennummer“ ausgegeben. Außerdem wird geprüft, ob die Deadline in der Zukunft(mindestens 24 Stunden) liegt. Ist dies nicht der Fall, wird die Meldung „Deadline ist ungültig“ ausgegeben. Die Eingaben werden in der oben beschriebenen Reihenfolge geprüft.
Für Langzeitkunden werden für bestimmte Waren Rabatte berechnet. Ist ein Kunde bereits länger als zwei Jahre im System und gehört die bestellte Ware zu den „Rabattwaren“, so erhält der Kunde fünf Prozent Rabatt. Wurden alle Daten korrekt eingegeben, wird ein Auftrag im System erstellt.

Ermittelt mit den folgenden Black-Box-Verfahren Testfälle:

a) Äquivalenzklassenbildung

b) Grenzwertanalyse

c) Entscheidungstabellentest

2. White-Box-Verfahren

Ihr habt nun den Code der oben beschriebenen Funktion erhalten.
Die Anforderungen des Kunden sind 100% Pfadüberdeckung zu erreichen.

```
public Order createOrder(int customerNr, int productNr, Date deadline)
throws ShopInputException{
    Date today = new Date(System.currentTimeMillis());
    if(!customers.containsKey(customerNr)){
      throw new ShopInputException("Kunden nicht im System");
    }
    if(!products.containsKey(productNr)){
      throw new ShopInputException("Es existiert keine Ware mit
```

```
            der gegebenen Warennummer");
    }
    if(!deadline.after(today)){
      throw new ShopInputException("Deadline ist ungueltig");
    }
    Order order =
    new Order(customerNr, productNr,
    products.get(productNr).getPrice(), deadline);
    if(System.currentTimeMillis()
    - customers.get(customerNr).getDateOfRegistration().getTime()
    >= 63072000){
      order.setRabatt(true);
    }
    return order;
}
```

a) Ermittelt die mit den bisherigen Testfällen erreichte Anweisungsabdeckung. Falls diese unter 100% liegt, erarbeitet weitere Testfälle um eine komplette Anweisungsüberdeckung zu erreichen.

b) Ermittelt die mit den bisherigen Testfällen erreichte Entscheidungsüberdeckung (Zweigüberdeckun Falls diese unter 100% liegt, erarbeitet weitere Testfälle um eine komplette Entscheidungsüberdeckung zu erreichen.

c) Ermittelt die mit den bisherigen Testfällen erreichte Pfadüberdeckung. Falls diese unter 100% liegt, erarbeitet weitere Testfälle um eine komplette Pfadüberdeckung zu erreichen.

d) Diskutiert, wie die Überdeckungskriterien miteinander in Beziehung stehen.

e) Diskutiert, wann man prinzipiell 100% Pfadüberdeckung erreichen kann und wann nicht.

3. Unit Test

Prüft die Funktion jetzt mit allen ermittelten Testfällen. Erstellt dazu aus den erarbeiteten Testfällen jUnit-Tests.