Hochschule **Rosenheim**
University of Applied Sciences

Fakultät für Informatik **Übung 14: Objektorientierte Programmierung (INF)** Prüfer: Prof. Dr. Kai Höfig Datum: 04.07.2019 Dauer: 90 Minuten Material: Ein Buch mit ISBN-Nr./RZ Handbuch	*(Name, Vorname)* *(Matrikelnummer)*
Erreichte Punktzahl und Gesamtnote: 1 | 2 | | | | Σ | | | | | | **Note:**	*(Erstkorrektor)* *(Zweitkorrektor)*

Hinweise:

- Die Heftklammern dürfen nicht gelöst werden. Bitte überprüfen Sie: Die Angabe umfasst **6 Seiten incl. Deckblatt.** Nur die Vorderseite eines Blattes ist jeweils bedruckt.
- Bearbeiten Sie die Fragen direkt in der Angabe. Nutzen Sie ggfs. die Rückseite.
- Verwenden Sie keinen Bleistift und keinen roten oder grünen Stift.
- Sollten Ihrer Meinung nach ***Widersprüche*** in der Angabe existieren bzw. Angaben in der Aufgabenstellung fehlen, machen Sie ***sinnvolle Annahmen und dokumentieren*** Sie diese.
- Alle Fragen und Lösungen beziehen sich auf die ***Programmiersprache Java 8***.
- `Java import` ***Statements*** müssen in Ihrer Lösung nicht angegeben werden.
- Die Angabe der erreichbaren Punkte dient zur Orientierung und kann sich noch verändern.
- Bitte ***schreiben Sie leserlich***, möglichst in Druckschrift.

Aufgabe 1: Verständnisfragen (8 Punkte)

a) Was ist der Unterscheid zwischen checked und unchecked exceptions? (2P)

b) In welchen Fällen ist die Verwendung der Klasse **`Stringbuffer`** klassischen Strings zu bevorzugen? (2P)

c) Wieso sollten Klassen möglichst *immutable*, also unveränderbar sein? (2P)

d) Für welchen Zweck werden Interfaces benötigt? (2P)

Aufgabe 2: Eigene Ausnahmebehandlungen (60 Punkte)

Im Restaurant **`ZumGoldenenEimer`** werden Gästen Lebensmittel serviert. Gäste haben eine bestimmte Menge Hunger dargestellt als Zahlenwert und eine begrenzte Menge Geld in Euro. Einem Gast können Gerichte durch die Methode **`iss(Gericht g)`** serviert werden. Dadurch sinkt der Hunger des Gastes und zwar um genau die Menge die durch die Methode **`getMenge`** eines Gerichts zur Verfügung gestellt wird. Gleichzeitig steigt seine Rechnung um den Betrag der durch die Methode **`getPreis`** ausgegeben wird. Im Restaurant **`ZumGoldenenEimer`** ist es nicht üblich, dass Gäste mehr bestellen als sie essen können oder dass sie Gerichte bestellen die sie nicht zahlen können. Daher sind diese Fälle die absoluten Ausnahmen, um die Sie sich in den folgenden Aufgaben kümmern sollen.

a) Implementieren Sie zunächst die Ausnahmen **`GastPleiteAusnahme`** und **`GastSattAusnahme`**, die die Klasse **`Exception`** erweitern. Im Falle dass der Gast satt ist, soll von der Methode **`getMessage`** immer der String *Uaaa bin i satt!* Ausgegeben werden. Für die Ausnahme, dass der Gast pleite ist, soll der Ausnahme der Fehlbetrag übergeben werden und von der Methode **`getMessage`** zum Beispiel der String *Leider € 2.21* zu wenig ausgegeben werden. (5 Punkte)

b) Implementieren Sie nun die Methoden **`abrechnen()`** und **`iss(Gericht g)`**. Sorgen Sie dafür, dass der Aufrufer der Methoden auf alle Ausnahmen reagieren muss.

- die Methode **`abrechnen`** zieht dem Kunden von seinem Geld den Betrag seiner bisherigen Rechnung ab. Sollte der Geldbetrag nicht ausreichen, ist sein Geldbetrag 0 und eine **`GastPleiteAusnahme`** mit den offenen Schulden wird geworfen.
- Die Methode **`iss`** schlägt dem Kunden den Preis des Gerichts auf seine Rechnung auf. Übersteigt der Rechnungsbetrag das Geld des Gastes, wird sofort die abrechnen Methode aufgerufen. Ausnahmen werden an den Aufrufer von **`iss`** weitergegeben. Ist die Menge des Gerichts höher als der Hunger, ist der Hunger gestillt (also 0) und eine **`GastSattAusnahme`** wird geworfen. (15P)

```
public class Gast{
    double rechnung;
    double geld;
    double hunger;


    public Gast(double geld, double hunger){
        this.hunger=hunger;
        this.geld=geld;
        this.rechnung=0;
    }
```

c) Ergänzen Sie folgende JUnit Testklasse für die Klasse Gast. (20 Punkte)

- Überprüfen Sie, ob eine `GastPleiteAusnahme` geworfen wird für den Fall, dass der Gast pleite ist.
- Überprüfen Sie, ob korrekt abgerechnet wird für den Fall, dass der Gast nicht pleite ist
- Überprüfen Sie, ob eine `GastSattAusnahme` geworfen wird für den Fall, dass der Gast pleite ist.
- Überprüfen Sie, ob der Gast auch brav aufgegessen hat.

```
public class GastTest {
    Gast egon;
    Gericht napoli;

    @Before
    public void init(){
        Zutat nudeln = new Zutat("Nudeln", 1.99, 0.5, 0.2);
        Zutat tomatensoße = new Zutat("Tomatensoße", 0.99, 0.25, 0.15);
        Zutat[] zutaten = new Zutat[]{nudeln, tomatensoße};
        this.napoli = new Gericht("Pasta Napoli", , 2);

        this.egon=new Gast(10,2); // Gast hat €10 und Hunger 2
        egon.rechnung=0;
    }
```

d) Erstellen Sie für die Klassen Gast, Gericht, Zutat, `GastSattAusnahme`, `GastPleiteAusnahme` und `GastTest` ein Klassendiagramm. Verwenden Sie die gängigen Modellierungselemente und vergessen Sie nicht die Quantitäten und Rollenbezeichnungen an den Beziehungen.
Zusätzlich sind sowohl die Ausnahme `GastSattAusnahme` als auch die Ausnahme `GastPleiteAusnahme` eine Zumutung für ein Restaurant. Daher sollen sie die `toString` Methode überschreiben und bei Aufruf jeweils „`unverschämt!`“ ausgeben. Zutaten als auch Gerichte sind ja essbare Nahrungsmittel die über eine Methode `essen` auf unterschiedliche Weise verzehrt werden können. Erweitern Sie das Klassendiagramm um diese Eigenschaften. Verwenden Sie die Ihnen bekannten Mittel, um Coderedundanzen zu vermeiden und über Polymorphie Typsicherheit zu erzeugen. (20 Punkte)