Übungsbeispiele – PR1
OOP-Basics

Beispiel 1 – Erweiterung Bruch

Erweitern Sie das „Bruch"-Beispiel aus der Vorlesung, sodass Sie eine weitere Überladung der Methode multiplicate erstellen, welche einen Array von Brüchen übernimmt und diese miteinander multipliziert: *public Bruch multiplicate(Bruch[] arr)*

Schreiben Sie eine Methode *public void trim()*, welche den Zähler und Nenner bestmöglich kürzt. Hierzu sind Zahlen, durch jede Zahl solange zu dividieren, bis die Division einen Rest liefern würde. Beispiel:

Zähler = 100, Nenner = 32
Division durch 2, 0 Rest, Z=50, N=16
Division durch 2, 0 Rest, Z=25, N=8
Division durch 2, liefert bei Zähler Rest, nächster Version
Division durch 3, liefert Rest, nächster Version
Division durch 4, liefert Rest, nächster Version
… Division durch 8, liefert Rest, Ende
Ergebnis: Z=25, N=8

Beispiel 2 – Rezeptverwaltung

Schreiben Sie eine Klasse Zutat, mit der Sie eine Zutat eines Rezepts für vereinfachte Kochrezepte verwalten können.

Attribut-Name	**Beschreibung**	**Über Konstruktor-parameter setzen**	**Getter**	**Setter**
Name (String)	Gibt die Zutat an	JA	JA	NEIN
Menge (int)	Gibt die Menge der Zutat als Ganzzahl	JA	JA	NEIN

Die Klasse Zutat hat außer dem Konstruktor und den Getter-Methoden keine weiteren Methoden.

Schreiben Sie eine zweite Klasse Rezept, mit der Sie Rezepte verwalten und umrechnen können.

Attribut-Name	**Beschreibung**	**Über Konstruktor-parameter setzen**	**Getter**	**Setter**
Name (String)	Gibt den Namen des Rezepts an	JA	JA	NEIN
Personen (int)	Gibt an, für wie viele Personen das Rezept normiert ist.	JA	JA	NEIN

Zutaten (Array von Zutat ➔ Zutat[])	Übernimmt die Zutaten eines Rezepts als Array	JA	JA	NEIN

public void printRezept()
Gibt das Rezept mit dem Rezeptnamen und dann, Zutat für Zutat mit dem Namen und der Menge aus.

public Rezept umrechnen(int personen)
Rechnet das Rezept für eine neue Anzahl an Personen um. Sind beispielsweise ursprünglich 100 Gramm einer Zutat für zwei Personen notwendig, so sind für 4 Personen 200 Gramm notwendig. Die Methode liefert ein neues Objekt der Klasse Rezept als Ergebnis zurück.

Demo-Klasse
Implementieren Sie auch eine Demo-Klasse zum Test Ihrer Klassen.

Beispiel 3 – Bestellverwaltung

Schreiben Sie eine Klasse Bestellzeile, mit der Sie eine Zeile einer Bestellung abbilden.

Attribut-Name	**Beschreibung**	**Über Konstruktor-parameter setzen**	**Getter**	**Setter**
Name (String)	Gibt den Namen des Artikels an	JA	JA	NEIN
Menge (int)	Gibt die Anzahl der Artikel an	JA	JA	NEIN
Preis (int)	Gibt den Preis für einen Artikel an	JA	JA	NEIN

public double getKosten()
Berechnet die Kosten für die Bestellzeile, in dem Menge und Preis multipliziert werden.

Schreiben Sie eine zweite Klasse Bestellung, mit der Sie die Bestellzeilen verwalten können.

Attribut-Name	**Beschreibung**	**Über Konstruktor-parameter setzen**	**Getter**	**Setter**
Nummer (String)	Gibt die Nummer der Bestellung an	JA	JA	NEIN
Zeilen (Array von Bestellzeilen)	Übernimmt alle Bestellzeilen in Form eines Arrays	JA	JA	NEIN

public void printBestellung()

Gibt die Bestellung mit all ihren Bestellzeilen aus.

public double getKosten()
Berechnet die Gesamtkosten für die Bestellung, indem für jede Bestellzeile die Methode getKosten() aufgerufen wird, diese summiert und zurückgegeben werden.

Demo-Klasse
Implementieren Sie auch eine Demo-Klasse zum Test Ihrer Klassen.

Beispiel 4 – Abschreibung

Schreiben Sie eine neue Klasse *Anlage*, welche ein Anlagengut aus betriebswirtschaftlicher Sicht (z. B. einen Schreibtisch oder einen Laptop) repräsentiert. Jede Anlage soll folgende Felder haben, welche tlw. über Konstruktor-Parameter gesetzt und über Getter ausgelesen werden können. Details dazu finden sich in der nachfolgenden Tabelle.

Attribut-Name	**Beschreibung**	**Über Konstruktor-parameter setzen**	**Getter**	**Setter**
Bezeichnung (String)	Bezeichnung des Anlagengutes	JA	JA	NEIN
initialWert (double)	Ursprünglicher Preis	JA	JA	NEIN
nutzungsdauer (int)	Nutzungsdauer in Jahren	JA	JA	NEIN
restWert	Gibt an, wie viel das Anlagegut noch Wert ist. Erhält den Wert von *initialWert* im Konstruktor.	NEIN	JA	NEIN
alter (int)	Gibt an, wie viele Jahre das Produkt schon im Einsatz ist. Wird im Konstruktor auf 0 gesetzt.	NEIN	JA	NEIN

public void abschreiben()
Wird am Ende jedes Jahres aufgerufen. Erhöht das Alter des Anlagenguts um 1. Wenn das Alter kleiner oder gleich der Nutzungsdauer ist, wird der Rest-Wert wie folgt neu berechnet:

restWert = Math.floor(initialWert / nutzungsdauer * (nutzungsdauer - alter))

Anmerkung: Math.floor rundet ab.

public void simulate(int maxJahre, int minWert)

Simuliert mehrere Jahre und ruft für jedes Jahr einmal die zuvor beschriebene Methode *abschreiben* auf. Davor und danach wird der aktuelle Rest-Wert gemeinsam mit dem Jahr (z. B. 1 für 1. Jahre, 2 für 2. Jahr etc.) ausgegeben. Die Methode stoppt, wenn min. eines der folgenden Kriterien erfüllt ist:

- Rest-Wert ist 0

- Es wurden *maxJahre* simuliert
- Der Restwert hat *minWert* unterschritten

public Anlage renew(int zusatzWert, int zusatzJahre)

Liefert basierend auf den Daten der betroffenen Anlage eine neue Anlage retour. Diese soll denselben Namen aufweisen. Der Initial-Wert soll der Summe aus dem aktuellen Rest-Wert und dem übergebenen Zusatz-Wert entsprechen; die Nutzungsdauer soll der Summe aus (*nutzungsdauer - alter*) und dem für *zusatzJahre* übergebenen Wert entsprechen.

Anmerkung: Diese Methode wird aufgerufen, wenn das jeweilige Anlagengut erneuert wird. Beispiele dafür sind das Aufrüsten eines Laptops oder das Renovieren einer Wohnung.

Demo-Klasse
Implementieren Sie auch eine Demo-Klasse, welche zeigt, dass die Klasse *Anlage* funktioniert.

Beispiel 5 – Bonusberechnung für Mitarbeiter

Schreiben Sie eine Klasse „Mitarbeiter“, mit der eine Lohnabrechnung und Bonusberechnung für einen Mitarbeiter durchgeführt werden kann.

Attribut-Name	**Beschreibung**	**Über Konstruktor-parameter setzen**	**Getter?**	**Setter**
Vorname (String)	Vorname eines Mitarbeiters	JA	JA	NEIN
Nachname (String)	Nachname eines Mitarbeiters	JA	JA	JA
Mitarbeiternummer (int)	Fortlaufende Nummer eines Mitarbeiters. Stellen Sie dies mittels Klassenattributen sicher.	NEIN	JA	NEIN
Gehalt (double)	Gibt das Gehalt pro Monat eines Mitarbeiters an	JA	JA	JA
Alter (int)	Gibt das Alter eines Mitarbeiters an.	NEIN	JA	NEIN

public double monatsAbrechnung()
Schreiben Sie die Methode monatsAbrechnung, welche für einen Monat eine Gehaltsabrechnung durchführt und das Ergebnis zurückliefert. Vom Gehalt sind 20 % Sozialversicherung abzuziehen. Vom Restbetrag wird noch die Einkommensteuer, welche sich wie folgt gliedert, abgezogen.

bis 10.000 Euro	10 %
10.000 Euro bis 20.000 Euro	20 %
20.000 Euro bis 30.000 Euro	32 %
30.000 Euro bis 50.000 Euro	45 %
ab 50.001 Euro	60 %

Beispiel:

Gehalt (Monatsgehalt * 12)	**28.000**
- 20 % SV	22.400
10 % von 10.000	- 1.000
20 % von 10.000 bis 20.000 (10.000)	- 2.000
32 % von 20.000 bis 30.000 (2.400)	- 768
Ergebnis	**18.632**
Ergebnis pro Monat (/ 12)	**1.552**

public double jahresAbrechnung()
Schreiben Sie die Methode jahresAbrechnung, welche das gesamte Gehalt für ein Jahr berechnet und das Ergebnis zurückliefert.

public double jahresAbrechnung(int monate)
Nachdem nicht alle Mitarbeiter am 1.1. eines Kalenderjahres beginnen, ist es erforderlich, dass eine Überschreibung der Methode jahresAbrechnung erstellt wird, welche die Anzahl der Monate, die der Mitarbeiter bei der Firma gearbeitet hat, übernimmt und dafür das Jahresgehalt berechnet.

Demo-Klasse
Implementieren Sie auch eine Demo-Klasse, welche zeigt, dass Ihre Klassen funktionieren.