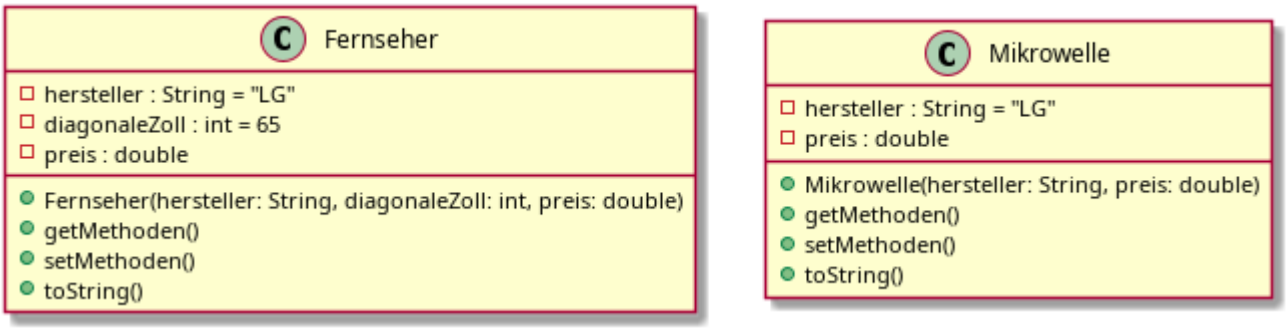
1 PLF	Name: _____
	Klasse: _____
OFFLINE	Datum: _____

VERERBUNG ___/5

Geben Sie das folgende UML Diagramm:



Implementieren Sie die Vererbung:

- Klassendefinition (`public class …`)
- Eigenschaften
- Konstruktor
- `setPreis` Methode
- `toString()` Methode

Der Name der Vaterklasse ist `Produkt`.

Der Preis soll geprüft werden:

- Fernseher zwischen 100 und 1200 Euro
- Mikrowelle zwischen 50 und 400 Euro

Im Fehlerfall geben Sie eine entsprechende Meldung aus. Der Defaultwert ist 150 Euro.

Die `toString()` Methode gibt die Produkte wie folgt aus:

```
Fernseher: LG - 199.0 Euro - 65 Zoll
Mikrowelle: Sony - 89.9 Euro
Fernseher: Sony - 249.9 Euro - 70 Zoll
Fernseher: LG - 349.9 Euro - 70 Zoll
Mikrowelle: Miele - 149.9 Euro
```

```
public _____ Produkt {
```

```
public _____ Fernseher {
```

public _____ Mikrowelle {

JUnit Test	___/5

Gegeben ist der folgende Code:

```
public class Fernseher {
   ...
   public void setDiagonaleZoll(diagonaleZoll) throws ProduktException {
      if (diagonaleZoll < 30)
         throw new ProduktException("Diagonale mind. 30 Zoll sein!");
      this.diagonaleZoll =  diagonaleZoll;
   }
   ...
}
```

Erstellen Sie die JUnitTests für die `setDiagonaleZoll()` Methode!

@Override ___/4

Warum wird bei einer Klasse ***OHNE Vererbung*** beim Generieren der `toString()` Methode automatisch das Kewort `@Override` hinzugefügt?

```
@Override
public String toString() {
```

Good luck!

1 PLF	Name: _____
	Klasse: _____
ONLINE	Datum: _____

Im weiteren Beispiel ist KEINE Vererbung zu implementieren!!!

EXCEPTION ___/ 4

Der `preis` eines Produkts muss über 100 Euro liegen.
Im Fehlerfall werfen Sie eine entsprechende `ProduktException`.
Ändern Sie die bestehende Parameterprüfung!

```
new Produkt("HP", 99.0);
   ==> ProduktException
new Produkt("HP", 199.9);
   ==> OK
```

AUFNEHMEN ___/ 4

Die Methode `aufnehmen()` fügt eine Objektreferenz der `ArrayList` hinzu.
Fügen Sie den fehlenden Code in die Klasse `Hander` ein.
Es dürfen maximal 8 Produkte aufgenommen werden.
Jedes `Produkt` (Objektreferenz) darf nur einmal vorkommen.
Im Fehlerfall werfen Sie eine entsprechende `ProduktException`.

```
public void aufnehmen(Produkt produkt) {
```

LOOP 1 ___/ 4

Die Methode `billigstesProdukt(hersteller)` liefert das günstigste Produkt des entsprechenden Herstellers.
Fügen Sie den fehlenden Code in die Klasse `Hander` ein.
Wenn der Hersteller kein Produkt anbietet, dann soll `null` zurückgegeben werden.

```
public Produkt billigstesProdukt(String hersteller) {
```

```
a = new Produkt("Lenovo", 999.9);
b = new Produkt("HP",     849.5);
c = new Produkt("Dell",   1099.0);
d = new Produkt("Dell",   999.9);
e = new Produkt("Asus",   799.0);
f = new Produkt("Acer",   699.5);
g = new Produkt("Lenovo", 1299.0);

billigstesProdukt("Lenovo")
   ==> a
billigstesProdukt("Apple")
   ==> null
```

LOOP 2 ___/ 4

Die Methode `loscheProdukt(uberPreis)` löscht alle Produkte über dem entsprechenden Preis.

Fügen Sie den fehlenden Code in die Klasse `Hander` ein.

Als Rückgabewert soll die Anzahl an gelöschten Elementen zurückgegeben werden.

```
public int loscheProdukt(double uberPreis) {
```

Ausgangslage:

```
a = new Produkt("Lenovo", 999.9);
b = new Produkt("HP",     849.5);
c = new Produkt("Dell",   1099.0);
d = new Produkt("Dell",   999.9);
e = new Produkt("Asus",   799.0);
f = new Produkt("Acer",   699.5);
g = new Produkt("Lenovo", 1299.0);
```

Beispiel 1:

```
loscheProdukt(2000)
```

==> 0

```
a = new Produkt("Lenovo", 999.9);
b = new Produkt("HP",     849.5);
c = new Produkt("Dell",   1099.0);
d = new Produkt("Dell",   999.9);
e = new Produkt("Asus",   799.0);
f = new Produkt("Acer",   699.5);
g = new Produkt("Lenovo", 1299.0);
```

Beispiel 1:

```
loscheProdukt(900)
```

==> 0

~~`a = new Produkt("Lenovo", 999.9);`~~
`b = new Produkt("HP",     849.5);`
~~`c = new Produkt("Dell",   1099.0);`~~
~~`d = new Produkt("Dell",   999.9);`~~
`e = new Produkt("Asus",   799.0);`
`f = new Produkt("Acer",   699.5);`
~~`g = new Produkt("Lenovo", 1299.0);`~~

Good luck!