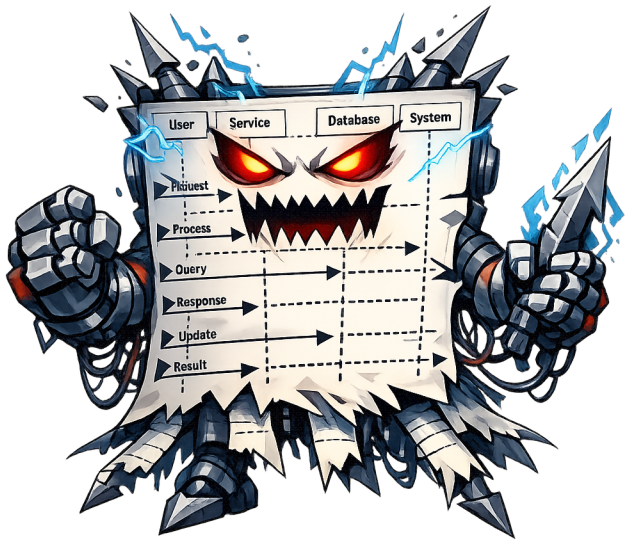
UML - Sequenzdiagramm

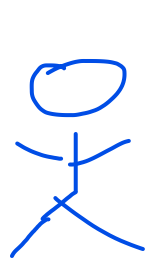


Den Austausch oder verlauf von Nachrichten ..
... Aufruf einer Methode

In welcher Klasse steht die Methode die aufgerufen werden soll ?
-> da muss der Pfeil hingehen.
-> jeder Pfeil muss in einem Balken enden - entsteht neu.

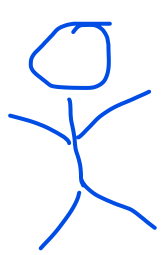
Es gibt noch Schleifen und If/Else ...

- Der Kunde geht zur Theke und bestellt einen Kaffee bei der Kellnerin.
- Die Kellnerin notiert die Bestellung und gibt sie an den Barista weiter.
- Der Barista bereitet den Kaffee zu.
- Sobald der Kaffee fertig ist, übergibt der Barista ihn an die Kellnerin.
- Die Kellnerin bringt den Kaffee zum Kunden.
- Der Kunde bezahlt bei der Kellnerin.
- Die Kellnerin bestätigt den Erhalt des Geldes und verabschiedet den Kunden.

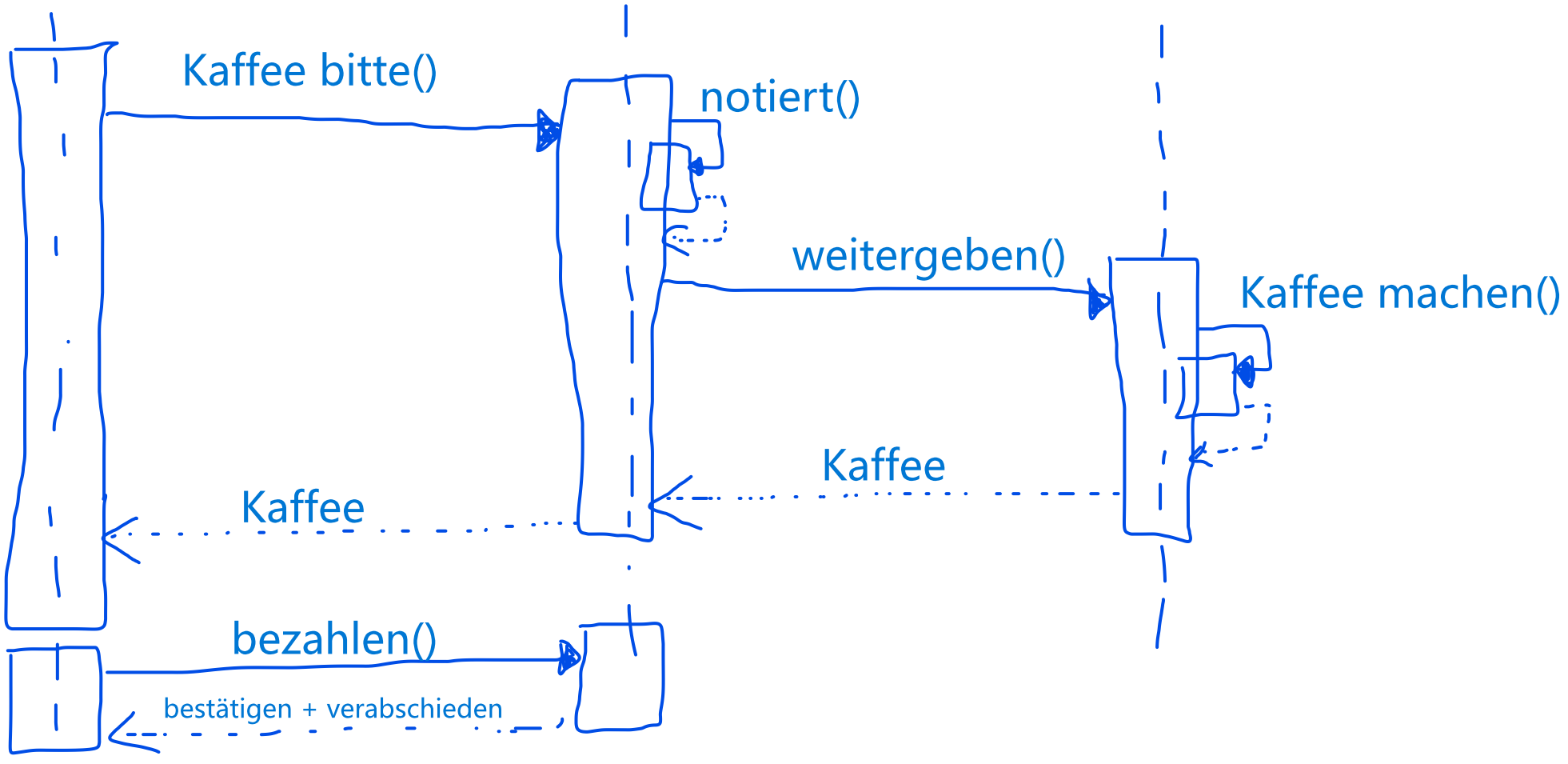


Kunde

Kellnerin



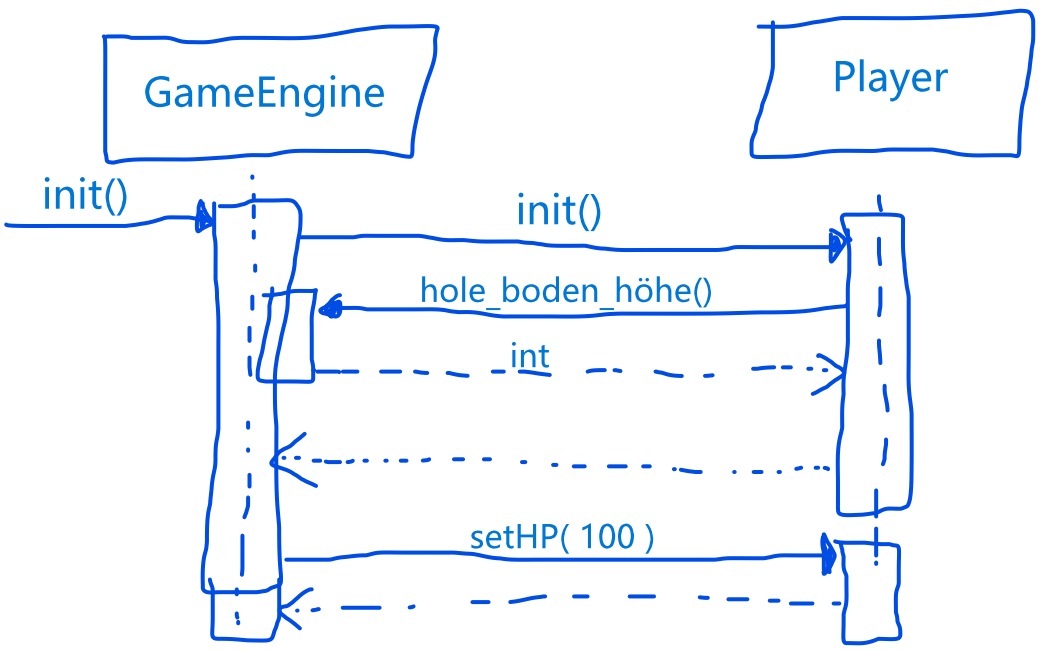
Barista



```
class GameEngine
{

    void init()
    {
        player.init();
        player.setHP( 100 );
    }

    int hole_boden_höhe()
    {
        ...
    }

}
```

Die *init()* Methode der GameEngine Klasse wird aufgerufen. Aus dieser wird die *init()* Methode eines Player Objektes aufgerufen. Der Player holt sich durch die *hole_boden_höhe()*Methode die Höhe des Players im aktuellen Level. Danach werden aus der GameEngine die HP des Players mit der Methode *setHP()* auf 100 gesetzt.

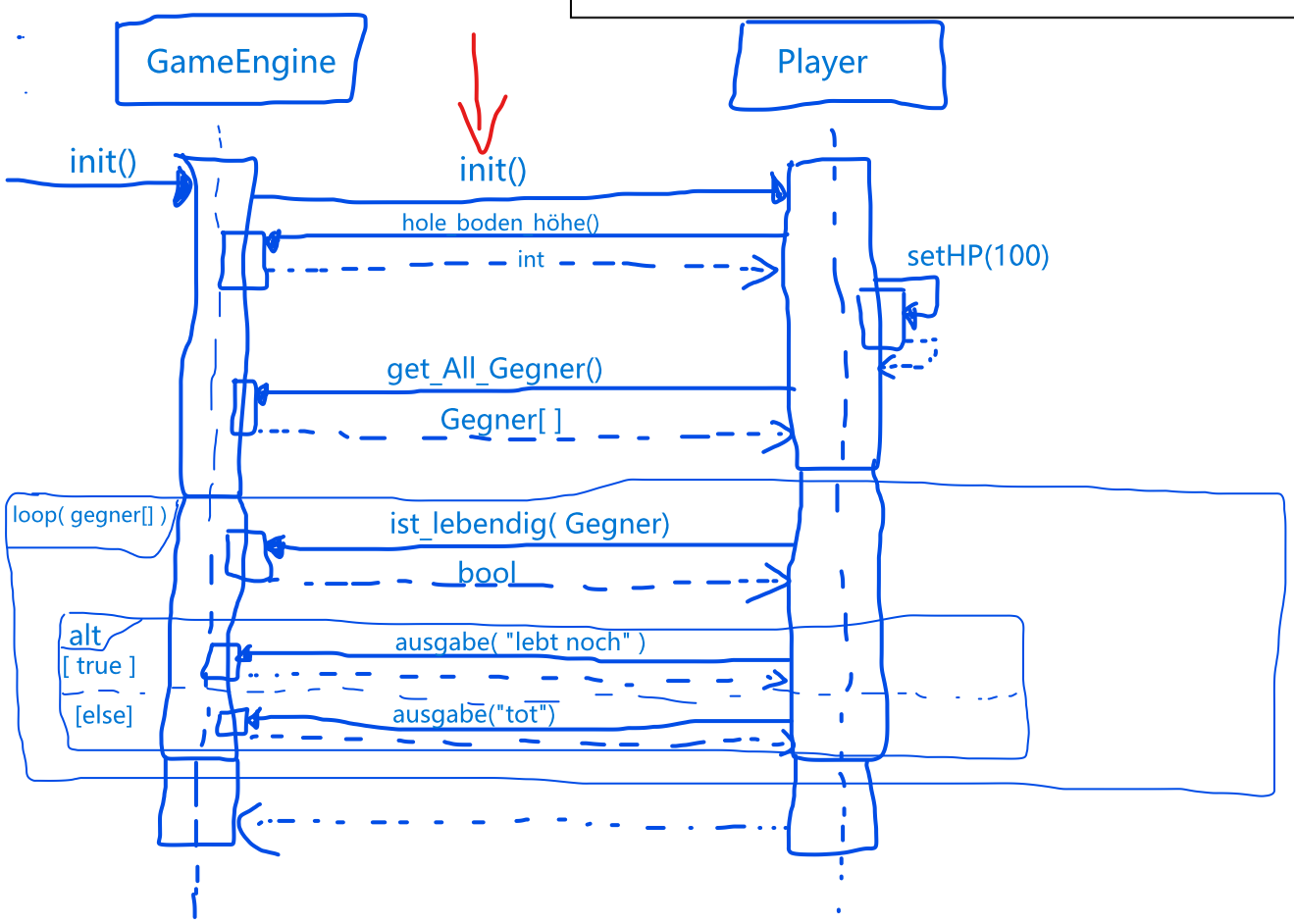
```
class Player
{

    void init()
    {
        this.höhe = game.hole_boden_höhe();
    }

    void setHP( int )
    {
        ...
    }

}
```



```
class GameEngine
{
    void init()
    {
        player.init();
    }

    int hole_boden_höhe()
    { ... }

    Gegner[] get_All_Gegner()
    { ... }

    bool ist_lebendig( Gegner )
    { ... }

    void ausgabe( string )
    { ... }
}
```

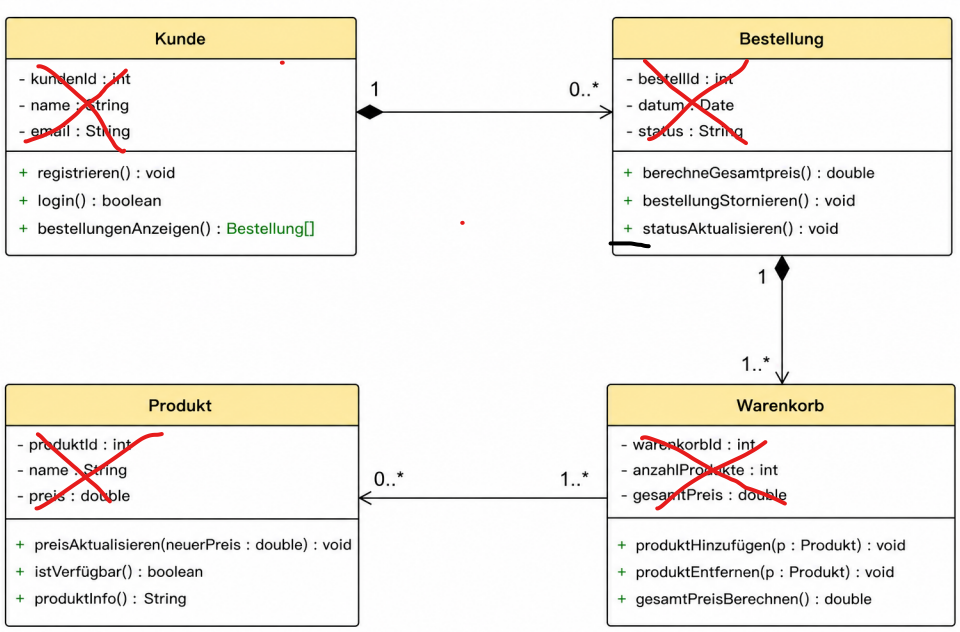
Die *init()* Methode der GameEngine Klasse wird aufgerufen. Aus dieser wird die *init()* Methode eines Player Objektes aufgerufen. Der Player holt sich durch die *hole_boden_höhe()* Methode die Höhe des Players im aktuellen Level. Danach werden die HP des Players mit der Methode *setHP()* auf 100 gesetzt. Dann holt sich der Player alle Gegner mit *get_All_Gegner()*. Der Player geht alle Gegner durch und prüft mit *ist_lebendig()* ob ein Gegner noch lebt. Wenn ja, wird mit *ausgabe()* „lebt noch“ angezeigt. Wenn nein wird mit *ausgabe()* „tot“ angezeigt.

```
class Player
{
    void init()
    {
        this.höhe = game.hole_boden_höhe();
        this.setHP( 100 );

        Gegner[] all_gegner = game.get_All_Gegner();

        foreach( Gegner tmp_gegener in all_gegner )
        {
            if(  game.ist_lebendig( tmp_gegener ) == true )
            {
                game.ausgabe( " lebt noch " );
            }
            else
            {
                game.ausgabe( " tot " );
            }
        }
    }

    void setHP( int )
    {
        ...
    }
}
```



Sync. Nachricht ()

Antwort



UML-Sequenzdiagramm

Ein Online-Shop-System wird gestartet.
Zu Beginn wird ein Kunde im System angelegt und führt anschließend mehrere Aktionen im Shop aus.

Zuerst meldet sich der Kunde im System an, indem die Methode login() ausgeführt wird. Danach wird eine neue Bestellung erzeugt. Nachdem die Bestellung erstellt wurde, wird ein Warenkorb erzeugt.

Anschließend wird ein Produkt in den Warenkorb gelegt, indem produktHinzufügen() aufgerufen wird. Um Informationen über das Produkt zu erhalten, wird anschließend produktInfo() ausgeführt. Darin wird überprüft, ob das Produkt verfügbar ist, indem istVerfügbar() aufgerufen wird.

Nachdem das Produkt bestätigt wurde, wird der Gesamtpreis berechnet, indem gesamtPreisBerechnen() ausgeführt wird. Danach wird zusätzlich der Gesamtpreis der Bestellung mit berechneGesamtpreis() berechnet.

Zum Abschluss wird der Bestellstatus aktualisiert, indem statusAktualisieren() ausgeführt wird und Dann lässt sich der Kunde alle seine Bestellungen anzeigen, indem bestellungenAnzeigen() aufgerufen wird.