Nachdenkzettel Clean Code

1. Klassenexplosion (Schwierig..)

```
class Formularfeld;
class Textfeld extends Formularfeld;
class Zahlfeld extends Formularfeld;
class TextUndZahlFeld extends Formularfeld;
class TextfeldOCR extends Textfeld;
class ZahlfeldOCR extends Zahlfeld;
class TextUndZahlFeldOCR extends TextUndZahlFeld;
class TextfeldSonderZ extends TextUndZahlFeld;
class TextfeldOCRSonderZ extends TextUndZahlFeldOCR;
class .....
```

> Jedeweitere Eigenschaft oder Spezialisierung führt zu vielen neuen Klassen durch Kombination. Die Folge ist explosives Anwachsen der Zahl der Klassen mit identischem Code. (Lösung?)

Zum einen sollte ein gutes Interface oder Abstrakte Klasse grundlegend sein (z.B. Feld, Eingabefeld).
Zum anderen könnte man statt für jede spezielle From eines Formularfeldes die einzelnen Eigenschaften im Konstruktor als Variablen übergeben. Dies könnte ähnlich wie z.B. bei HTML beim input-Element funktionieren mit type = ...
Aber: dies stoßt auch bei vielen Eigenschaften an ihre Grenzen, da die Basisklasse mit Funktionen zu gemüllt wird, deswegen sollte man eine geschickte Gruppierung in Obertypen erstellen, welche dann von der Basisklasse erben.

2. Der verwirrte und der nicht-verwirrte Indexer
was genau unterscheidet die beiden Indexer? Wieso ist der eine „verwirrt“?

Der verwirrte Indexer hat 2 Member-Variablen langID und langISOCode die eigentlich einem Constraint unterworfen sein sollten, da die ID immer zum ISO-Code passen muss. Dieser Constraint existiert aber nicht in diesem Code. Die Übereinstimmung beider Argumente muss vom User manuell überprüft werden, da dies nicht im Code passiert. Das ist nicht user friendly! Es sollte also nur entweder ID oder ISO Code als Parameter im Konstruktor definiert werden und die Setter entfernt werden, wie bei dem nicht-verwirrten Indexer der Fall ist.

3. Korrekte Initialisierung und Updates von Objekten

```
public class Address {

        private String City;
        private String Zipcode;
        private String Streetname;
        private String Number;

        public void set City (String c) {
                City = c;
        }
        public void set Zipcode(String z) {
                Zipcode = z;
        }
}
```

Wie initialisieren Sie Address richtig? Wie machen Sie einen korrekten Update der Werte?

Um zu vermeiden ein ungültiges Objekt zu erzeugen, indem durch die set-Methoden das Objekt an Gültigkeit verliert, sollte man nur den Konstruktor benutzen um die Werte einmalig zu setzen oder direkt ein neues Objekt erstellen. Die Wahrscheinlichkeit ist nämlich sehr unwahrscheinlich dach sich nur ein Wert ändert und die Adresse immernoch korrekt ist.

```
public class Address {

        private String City;
        private String Zipcode;
        private String Streetname;
        private String Number;

        public Adress(String city, String zipcode, String street,
        String number){
                this.City = city;
                this.Zipcode = zipcode;
                this.Streetname = street;
                this.Number = number;
        }
        [... getter je nach Bedarf]
}
```

4. Kapselung und Seiteneffekte

```
public class Person {

        public Wallet wallet = new Wallet();
        int balance = 0;

        public Wallet getWallet(void) {
                return wallet;
        }

        public addMoney(int money) {
                wallet.add(money);
                balance = wallet.size();

        public int getBalance(){
                return balance;
        }
}
```

```
public class Person {

        private Wallet wallet = new Wallet();
        private int balance = 0;

        private Wallet getWallet() {
                return wallet;
        }

        public addMoney(int money) {
                wallet.add(money);
                balance = wallet.size();

        protected int getBalance(){
                return balance;
        }
}
```

Reparieren Sie die Klasse und sorgen Sie dafür, dass die Gültigkeit der Objekte erhalten bleibt und keine Seiteneffekte auftreten.