Nachdenkzettel: Interfaces

1. `class B implements X`. Jetzt fügen Sie eine neue Methode in Interface `X` ein. Was passiert? Was bedeutet dies für die "Robustheit" der Interface-Definition?

2. Ihr Code enthält folgendes Statement

   ```
   X myX = new X();
   myX.doSomething();
   ```

   Was ist daran problematisch, wenn Sie eine Applikation für verschiedene Branchen/Kunden/Fälle bauen? Wie schaffen Sie Abhilfe? Welche Rolle spielen dabei Factories?

3. Was gehört zum "Interface" einer Java-Klasse? Heißt: Was ist für außenstehende Objekte ersichtlich?

4. *Optionale Aufgabe*: Zwei Interfaces sind nicht voneinander abgeleitet, haben aber zufällig die gleiche Methode. Können Sie Implementationen dieser Interfaces polymorph behandeln?

   ```
   interface X{
       void foo();
   }

   interface Y{
       void foo();
   }

   class B implements Y {...}

   X x = new B(); // möglich?
   x.foo(); // möglich?
   ```

Nachdenkzettel: Interfaces

1. Wenn Sie eine neue Methode zu Interface X hinzufügen, führt dies zu einem Kompilierungsfehler in der Klasse B, da diese das Interface implementiert und daher alle seine Methoden implementieren muss. Wenn Sie eine neue Methode hinzufügen, wird die Klasse B nicht mehr als gültig betrachtet, bis Sie die neue Methode implementiert. Das Hinzufügen von Methoden zu einem bestehenden Interface kann Auswirkungen auf die Rückwärtskompatibilität haben, da alle Klassen, die das Interface implementieren, geändert werden müssen, um die neuen Methoden zu implementieren. Dies kann die "Robustheit" der Interface-Definition beeinträchtigen, da es Änderungen erfordert, die sich auf bestehenden Code auswirken können.
2. Ein Interface kann nicht mit „new“ selbst initialisiert werden.
   Dies kann mit einer Art von Factory behoben werden, in dem man eine Methode schreibt, die das Interface neu erstellt. Diese muss dann immer aufgerufen werden, wenn man das interface aufrufen will.

3. Das Interface einer Java-Klasse definiert die öffentlichen Methoden, die von

```
interface X {
    void doSomething();
    static X createX() {
        return new XImpl();
}}
class XImpl implements X {
    @Override
    public void doSomething() {
        // Implementierung der Methode
}}
X myX = X.createX();
myX.doSomething();
```

   außenstehenden Objekten aufgerufen werden können. Es umfasst alle öffentlichen Methoden und Konstanten (public static final Variablen), die die Klasse anbietet. Das Interface beschreibt die Schnittstelle zur Verwendung der Klasse und sagt, welche Funktionalitäten für externe Objekte verfügbar sind.

4. Ja dies funktioniert, wenn die Methoden denselben Namen besitzen sowie selbe Rückgabewert!