Nachdenkzettel: Exceptions

1. Was ist falsch mit folgendem Code? Benennen Sie die Fehler und beheben Sie diese.

```
public void doSomething() {
    try {
        lock(); // lock some resource
        // open some resource
        // try to change some things
        // fool around a bit
    } catch (Exception e) {

    } catch (ConcurrentModificationException e) {
        System.out.println("bad stuff going on today!") }
    finally {
        return;
    }
}
```

2. Was ist die Ausgabe des folgenden Programms?

```
public class TestException1 {

    public static void main(String[] args) {
        try {
            badMethod();
            System.out.print("A"); }
        catch (Exception ex) {
            System.out.print("B"); }
        finally {
            System.out.print("C");
        }

        System.out.print("D");

        }

        public static void badMethod() {
            throw new Error();
        }
}
```

3. Wann sollten Sie einen re-throw einer `Exception` implementieren?

4. In einem Web-Shop wird zur Laufzeit festgestellt, dass eine Kunden-ID nicht in der Datenbank vorhanden ist. Ist dies

    - eine System-Exception?
    - eine Custom-Exception?
    - keine Exception?

  Warum?

5. Was ist der Vorteil von Exceptions gegenüber dem Auswerten von Fehlerwerten im Return?

1.

a. Man sollte niemals leere catch Blöcke verwenden

b. Die ConcurrentModificationException wurde schon gefangen, weil der catch Block davor jede Exception fängt. Man sollte immer zuerst die spezifischsten Exceptions fangen und dann immer unspezifischer werden.

c. Das return im Finally Block kann zu Fehlern führen. Wenn schon vorher etwas returned wird, dann wird es sicher durch das return im Finally Block überschrieben, und so auch Exceptions die möglicherweise vorher auftreten.

```
public void doSomething() {
    try {
        lock();
        throw new ConcurrentModificationException();
    } catch (ConcurrentModificationException e) {
        System.out.println("bad stuff going on today!");
    } catch (Exception e) {
        System.err.println(e.getMessage());
    } finally {
        // Cleanup
    }
}
```

2. C, weil der Error nicht gefangen wird und dadurch wird nur der Finally Block ausgeführt und im Anschluss bricht das Programm sofort ab.

3. Man sollte eine Exception rethrowen, wenn man entweder die Exception woanders behandeln will (und evtl. beim ersten Auftreten nur loggen), oder man die Exception nur teilweise behandeln will/kann.

4. In diesem Fall sollte keine Exception verwendet werden, weil schon vorher durch andere Methoden geprüft werden sollte ob die ID existiert oder nicht. Möglich wäre eine Custom-Exception, aber eine System-Exception sollte bei so etwas nicht auftreten.

5. Die Exceptions können besser behandelt werden und ggf. kann noch Code im Finally Block ausgeführt werden. Außerdem kann klarer zwischen Fehlermeldungen und wirklichen Rückmeldungen unterschieden werden.