

Clean Code

What is Clean Code?

- Can be read and enhanced by a developer other than its original author
- Has unit and acceptance tests
- Does one thing well
- Looks like it was written by someone who cares
- Never obscures the designer's intent
- Each routine you read turns out to be pretty much what you expected
- Reads like well written prose
- Provides a clear and minimal API
- Elegant & efficient

Names

What might these functions do? (🦇)

```
// ...  
player1.setmenOCom(1);  
player2.setmenOCom(0);  
// ...
```

The "Batman Mode" metaphor (🦇)

When there is a mystery or crime to be solved, Batman will utilize his brain and all kinds of fancy gadgets to get it done! He will analyze, investigate and deduce until he has the answer. For him as a *costumed Super Hero Detective* it's part of the job! *Software engineers* should **never** have to go into Batman Mode to investigate about names used in the code!

Clarifying (!?) declaration

```
public void setmenOCom(int a) {  
    this.menOCom = a;  
}
```

Comment () to the rescue

```
/**
 * setzt menOCom, 0 = Mensch, 1 = Computer
 *
 * @param int fuer menOCom
 */
public void setmenOCom(int a) {
    this.menOCom = a;
}
```

Reveal your intent

```
import static PlayerType.*;
// ...
player1.setType(HUMAN);
player2.setType(COMPUTER);
// ...
```

```
public void setType(PlayerType type) {
    this.type = type;
}

public enum PlayerType {
    HUMAN, COMPUTER
}
```


Reveal your intent (💡)

- Any Variable/Method/Class name should tell
 - Why it exists
 - What it does
 - How it is used
- If you need a comment to explain the name, the name is probably ill chosen

Disinformation

What might `ssd`, `sd` and `cd` mean?

```
private final XXXXXXXXXXXXXXXXXXXX ssd;  
private final XXXXXXXXXXXXXXX sd;  
private final XXXXXXXXXXXXXXX cd;
```

Entrenched vs. Intended Meaning

```
private final XXXXXXXXXXXXXXXXXXXX ssd;  
private final XXXXXXXXXXXXXXXX sd;  
private final XXXXXXXXXXXXXXXX cd;
```



Entrenched vs. *Intended* Meaning

```
private final IShipmentSearchDao ssd;  
private final IShipmentDao sd;  
private final IContainerDao cd;
```

Abbreviations easily get out of control (:trollface:)

```
private final IShipmentSearchDao ssd;  
private final IShipmentDao sd;  
private final IContainerSearchDao csd;  
private final IContainerDao cd;  
private final IShipmentStatusSearchDao sstsd;  
private final IShipmentStatusDao sstd;  
private final ISpecialShipmentSearchDao spssd;  
private final ISpecialShipmentDao spsd;  
private final IStatusSearchDao stsd;  
private final IStatusDao std;
```

Avoid indistinguishable characters

```
int a = 1;  
if (0 == 1)  
    a = 01;  
else  
    I = 01;
```

No Disinformation (💡)

- Do not leave false clues
- Do not obscure the meaning of code
- Consider entrenched vs. intended meaning
- Avoid inconsistent spelling

No language mashups

```
private void maxFourEqualValues(int[] werte) {  
    int testValue = werte[0];  
    int equalValues = 1;  
  
    for (int i = 1; i < 7; i++) {  
        if (testValue == werte[i]) {  
            equalValues++;  
        } else {  
            equalValues = 1;  
            testValue = werte[i];  
        }  
        if (equalValues == 5) {  
            throw new IllegalArgumentException(  
                "Ein Wert wurde häufiger als 4x übergeben.  
                + Betroffener Wert: " + testValue);  
        }  
    }  
}
```

No unpronouncable names

```
class BaseDxoProcessMilestone7600LstBo  
class Dx2FltrShipmentCustPartyXDto  
class GyqfaChBppResDao  
class SegmentG041Data  
class KnlobiLocation  
class SwotService
```

Avoid prefix **I** for interfaces

- Preceding **I** is a distraction at best...
- ...and too much information at worst

 **Leave interfaces unadorned!**

Encoding & Mappings (💡)

- Stick to problem or solution domain names
- Follow commonly accepted naming conventions...
- ...and change any home-grown bad ones

🎯 *Don't force the reader to translate your names into ones they use and understand.*

Functions

Comments

Formatting