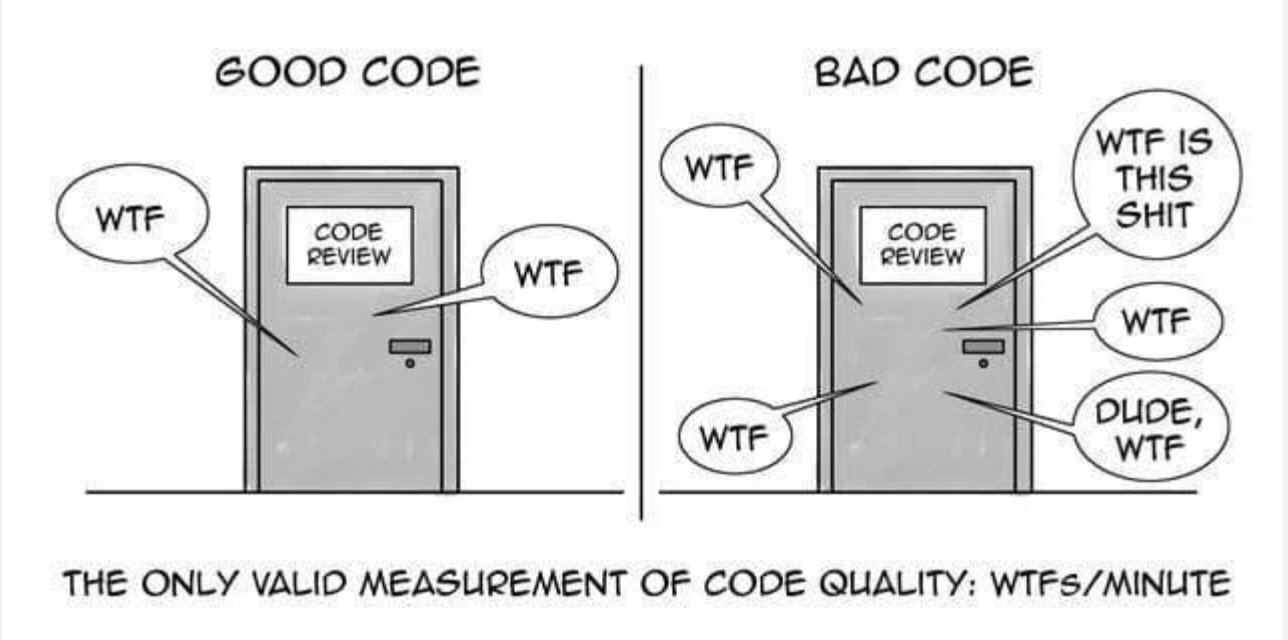
OBJECT CALISTHENICS

MOTIVATION


GOOD CODE
WTF
CODE REVIEW
WTF
BAD CODE
WTF
CODE REVIEW
WTF IS THIS SHIT
WTF
WTF
DUDE, WTF
THE ONLY VALID MEASUREMENT OF CODE QUALITY: WTFS/MINUTE

1. ONE LEVEL OF INDENTATION PER METHOD

2. DON'T USE THE ELSE KEYWORD

Definitely the latter. The former doesn't look bad right now, but when you get more complex code, I can't imagine anyone would think this:

```
public int SomeFunction(bool cond1, string name, int value, AuthInfo perms)
{
    int retval = SUCCESS;
    if (someCondition)
    {
        if (name != null && name != "")
        {
            if (value != 0)
            {
                if (perms.allow(name)
                {
                    // Do Something
                }
                else
                {
                    reval = PERM_DENY;
                }
            }
            else
            {
                retval = BAD_VALUE;
            }
        }
        else
        {
            retval = BAD_NAME;
        }
    }
    else
    {
        retval = BAD_COND;
    }
    return retval;
}
```

is more readable than

```
public int SomeFunction(bool cond1, string name, int value, AuthInfo perms)
{
    if (!someCondition)
        return BAD_COND;

    if (name == null || name == "")
        return BAD_NAME;

    if (value == 0)
        return BAD_VALUE;

    if (!perms.allow(name))
        return PERM_DENY;

    // Do something
    return SUCCESS;
}
```

I fully admit I never understood the advantage of single exit points.

Share

edited Jul 7, 2017 at 16:44

community wiki
6 revs, 5 users 81%
Jason Viers

3. WRAP ALL PRIMITIVES AND STRINGS

```
@TinyType
@EqualsAndHashCode
@ToString
@AllArgsConstructor(access = AccessLevel.PRIVATE)
class Foo {

    @Getter
    @NonNull
    private final String value;

    no usages
    public static Foo foo(@NonNull String value) {
        return new Foo(value);
    }
}
```

4. FIRST CLASS COLLECTIONS

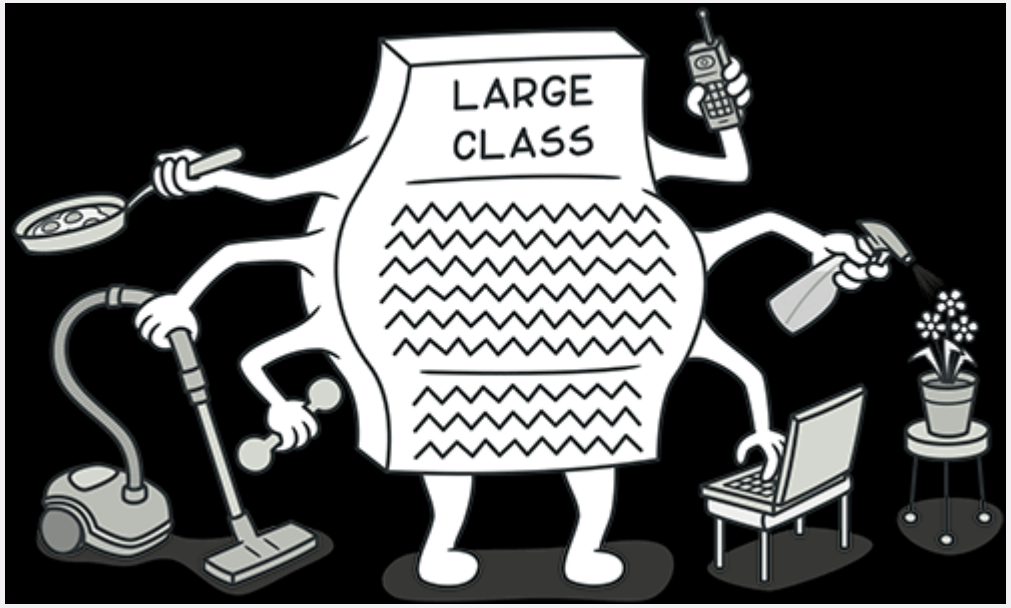
```java
import java.util.ArrayList;
import java.util.List;

public class WinningConditions {
    private final List<WinningCondition> winningConditions;

    public WinningConditions() {
        winningConditions = new ArrayList<>();
        winningConditions.add(new WinningCondition(Position.TOP_LEFT, Position.TOP_CENTER, Position.TOP_RIGHT));
        winningConditions.add(new WinningCondition(Position.MIDDLE_LEFT, Position.MIDDLE_CENTER, Position.MIDDLE_RIGHT));
        winningConditions.add(new WinningCondition(Position.BOTTOM_LEFT, Position.BOTTOM_CENTER, Position.BOTTOM_RIGHT));
    }

    public List<WinningCondition> getConditions() {
        return winningConditions;
    }
}
```

5. ONLY ONE DOT PER LINE

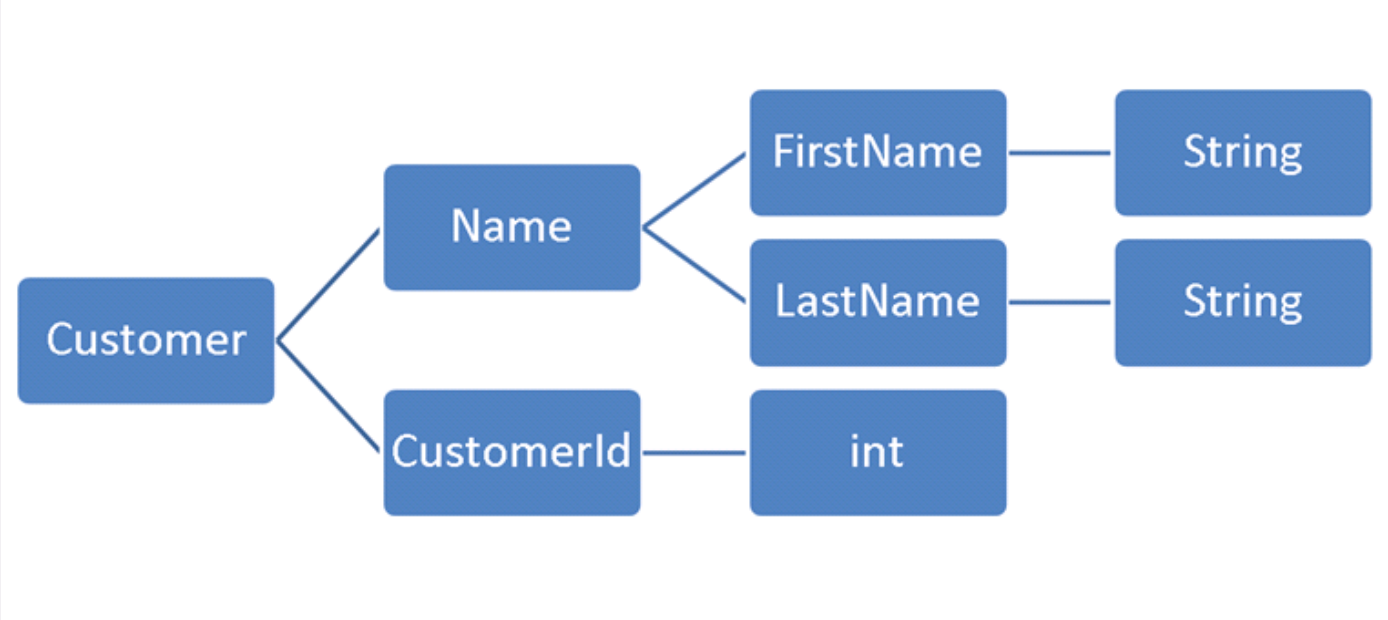
```
// bad
a.getFoo()
.getBar()
.getFoobar()
.doAll();

// okay
fooBuilder.bar("value")
.foobar("value")
.barFoo("value")
.build();
```

6. NO ABBREVIATIONS

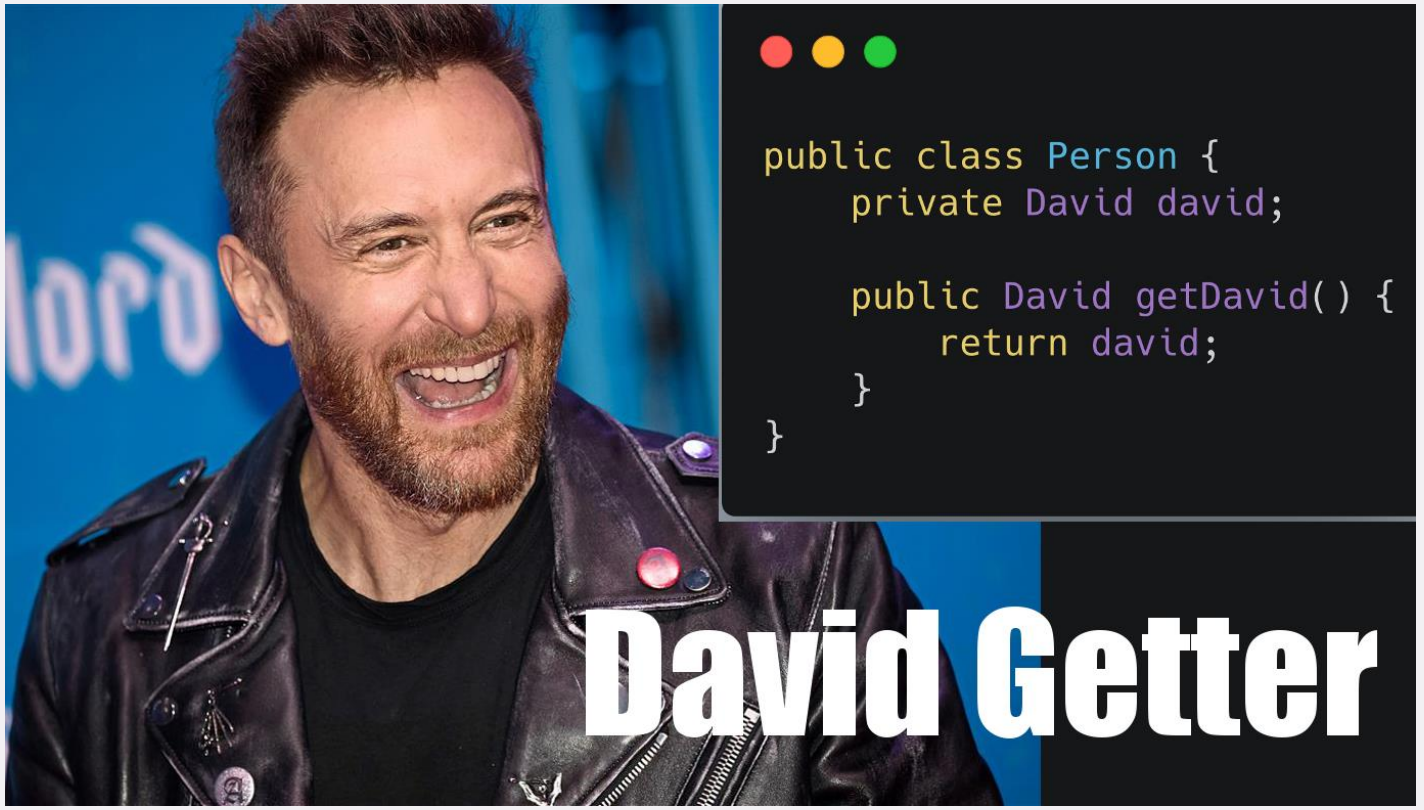
7. KEEP ALL ENTITIES SMALL



8. NO CLASSES WITH MORE THAN TWO INSTANCE VARIABLES



9. NO PUBLIC GETTERS SETTERS PROPERTIES



THANK YOU!

QUESTIONS?