EMBRACING OBJECT CALISTHENICS

BEYOND GETTERS AND THE POWER OF "TELL, DON'T ASK"

Patrick Ronecker

OVERVIEW OF OBJECT CALISTHENICS

- Programming exercise
- Aimed at improving object-oriented design skills
- Focuses on writing clean & maintainable code
- At first follow the rules strictly to later break them
- Additional rule: Use TDD in combination for feedback

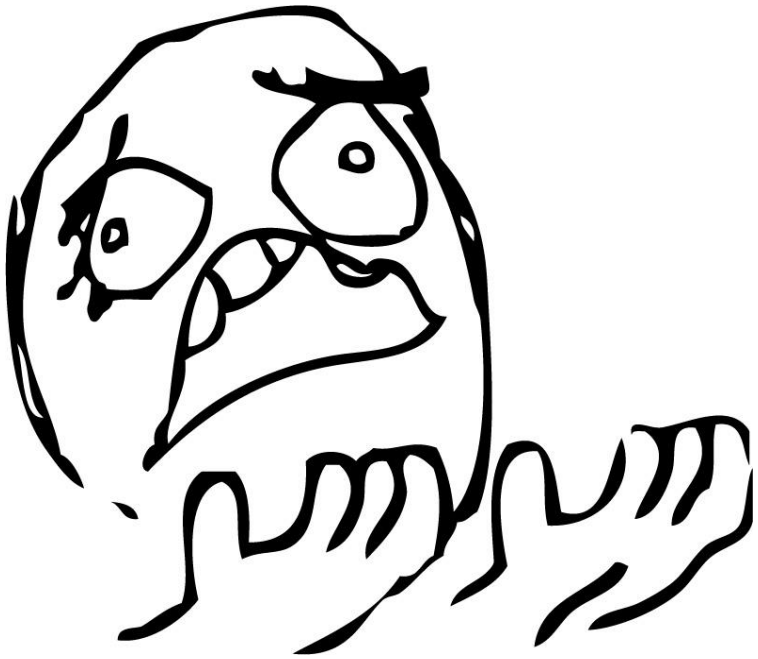
Rules:

1. Only one level of indentation per method
2. Don't use the ELSE keyword
3. Wrap all primitives and strings (wrap primitiv types in classes)
4. First class collections (wrap collections in classes)
5. One dot per line
6. Don't abbreviate
7. Keep all entities small
8. No classes with more than two instance variables
9. No getters/setters/properties
10. All classes must have state

EMBRACING RULE 9: NO GETTERS/SETTERS/PROPERTIES.

- Avoidance of Direct Access of Internals
- Encourage Behavior Methods
- Shift from Data-Centric to Behavior-Centric
- Forces you to really think object-oriented

  (Network of entities that collaborate by passing messages)



BASIC EXAMPLE – AVOIDING GETTERS FOR TESTS

Instead of using getters:

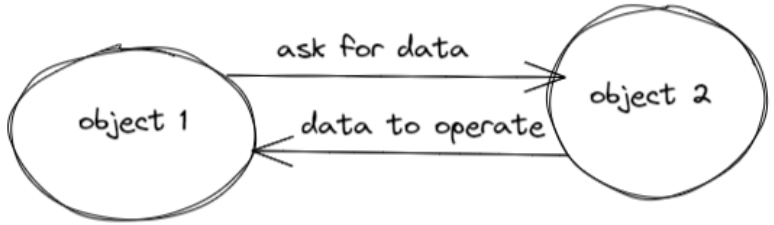
```
public class Person
{
    string getFirstName()
    string getLastName()
}

public void A_Test_That_Verifies_We_Get_The_Expected_Person()
{
  // we get a person instance from somewhere
  assert.equal("Bart", person.getFirstName)
  assert.equal("Simpson", person.getLastName)
}
```

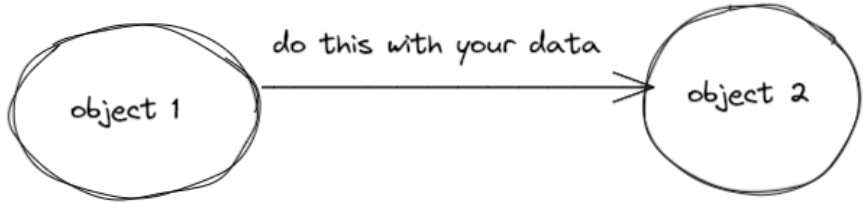
Use object equality:

```
public class Person
{
    firstName
    lastName

    // Override equality
    boolean equals(Object o) {
      // implement equality for person
    }
}

public void A_Test_That_Verifies_We_Get_The_Expected_Person()
{
  var bart = new Person("Bart", "Simpson")
  // we get a person instance from somewhere
  assert.equal(bart, person)
}
```

GOING FURTHER – THE "TELL, DON'T ASK" PRINCIPLE

- Telling objects what to do, rather than asking them for dat acting upon that data externally
- shifts focus
  - from procedural style (asking for data and then processing it)
  - to object-oriented style (directing the object to perform an a
- Aligns perfectly with the "No Getters/Setters" rule





"TELL, DON'T ASK": EXAMPLE

class AskMonitor...

```
private int value;
private int limit;
private boolean isTooHigh;
private String name;
private Alarm alarm;

public AskMonitor (String name, int limit, Alarm alarm) {
  this.name = name;
  this.limit = limit;
  this.alarm = alarm;
}
public int getValue() {return value;}
public void setValue(int arg) {value = arg;}
public int getLimit() {return limit;}
public String getName()  {return name;}
public Alarm getAlarm() {return alarm;}
```

```
AskMonitor am = new AskMonitor("Time Vortex Hocus", 2, alarm);
am.setValue(3);
if (am.getValue() > am.getLimit())
  am.getAlarm().warn(am.getName() + " too high");
```

class TellMonitor...

```
public void setValue(int arg) {
  value = arg;
  if (value > limit) alarm.warn(name + " too high");
}
```

```
TellMonitor tm = new TellMonitor("Time Vortex Hocus", 2, alarm);
tm.setValue(3);
```

"TELL, DON'T ASK": BENEFITS & CHALLANGES

Benefits

- Enhances Encapsulation
- Reduces Coupling
- Improves Code Maintainability
- Promotes Clearer Intentions in Code
- Supports Better Abstraction
- Fosters More Robust Object Model
- Facilitates Testing

Challenges

- Learning Curve
- Conceptual Shift
- Increased Design Effort and potential Over-Engineering
- Difficult to fit in an existing codebase

BEST PRACTICES & FINAL THOUGHTS

- There is nothing wrong with writing a getter method
- Use both approaches thoughtfully
- Avoid poor rationale
- Make a deliberate choice and document it
- There is No 'One-Size-Fits-All'-Solution
- Stay Open to New Styles or Alternatives

THANK YOU FOR YOUR ATTENTION

PATRICK.RONECKER@CSS.CH

SOURCES

- Agile Technical Practices Distilled by Pedro Moreira Santos, Marco Consolaro, Alessandro Di Gioia
- https://martinfowler.com/bliki/TellDontAsk.html
- https://martinfowler.com/bliki/GetterEradicator.html