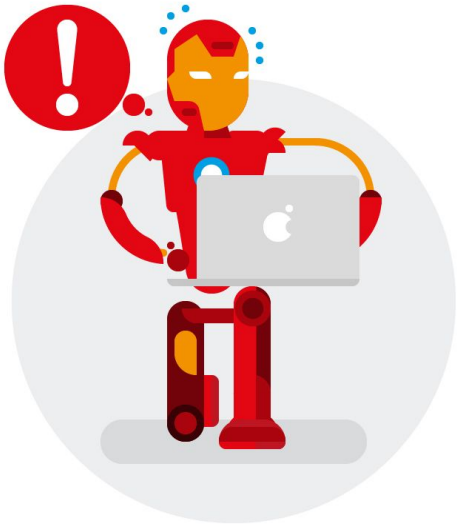
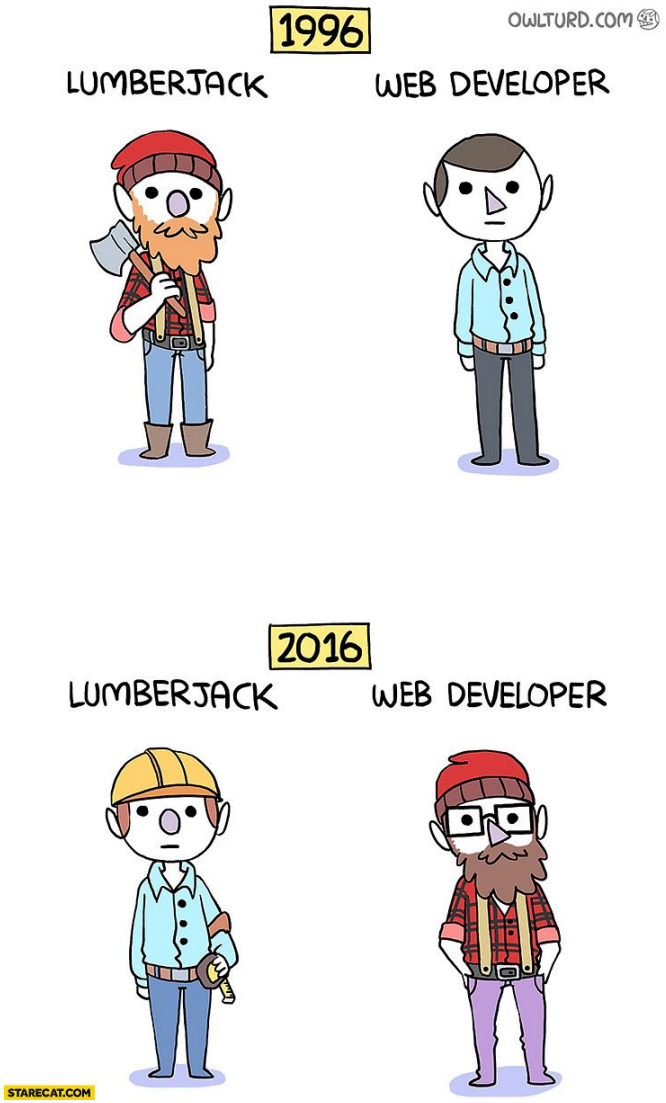
SOLID + CleanCode

Design objektorientierter Systeme

https://blog.codinghorror.com/why-cant-programmers-program/

Office Space (1999)



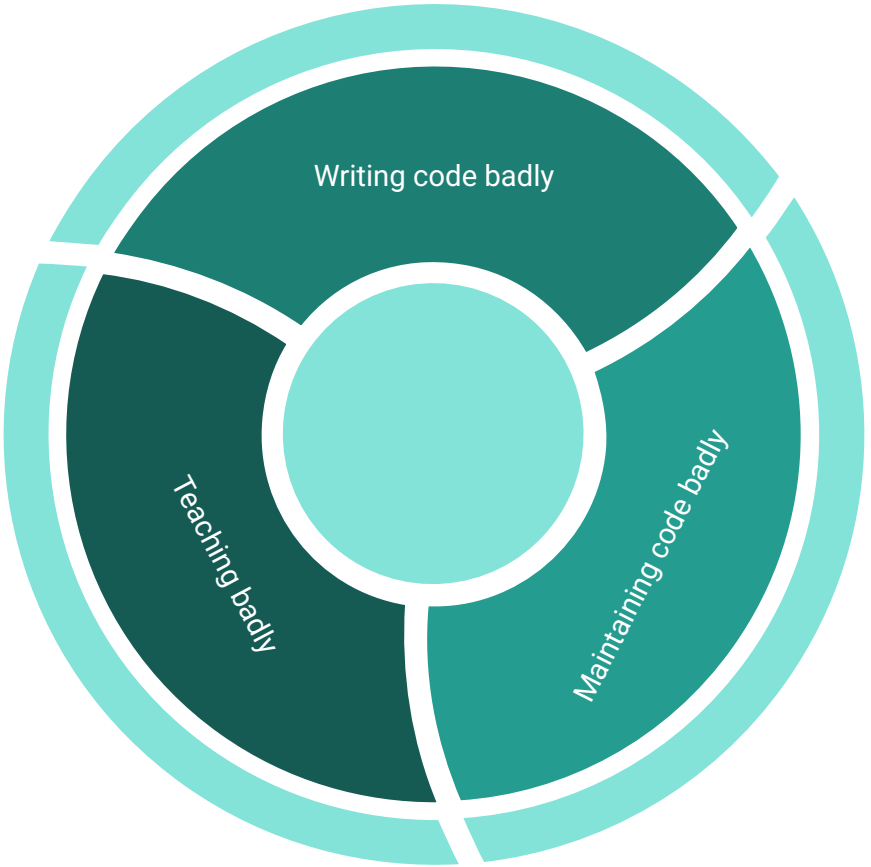


[redacted]

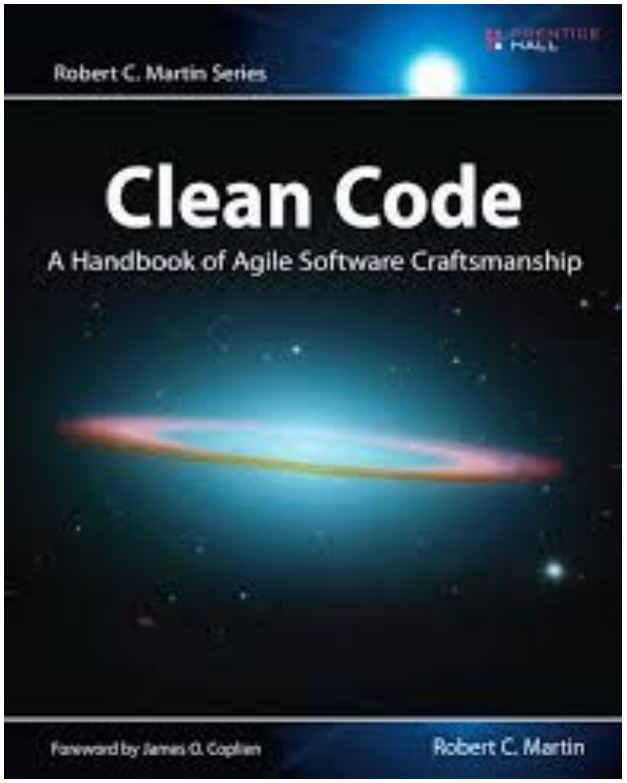
🚀 Senior React Frontend Developer • Contractor • Freelancer | Implementing clean, maintainable and scalable web apps

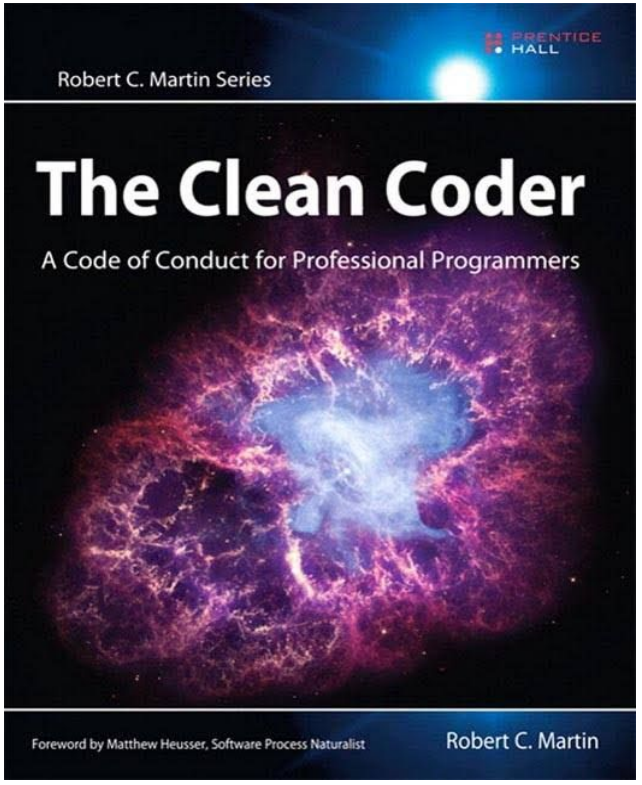
[redacted] Romania · Contact info

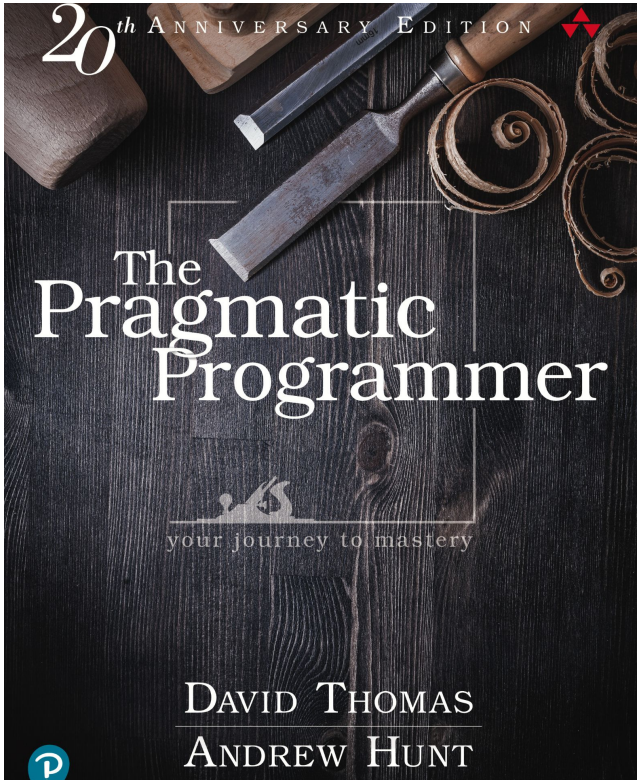
500+ connections

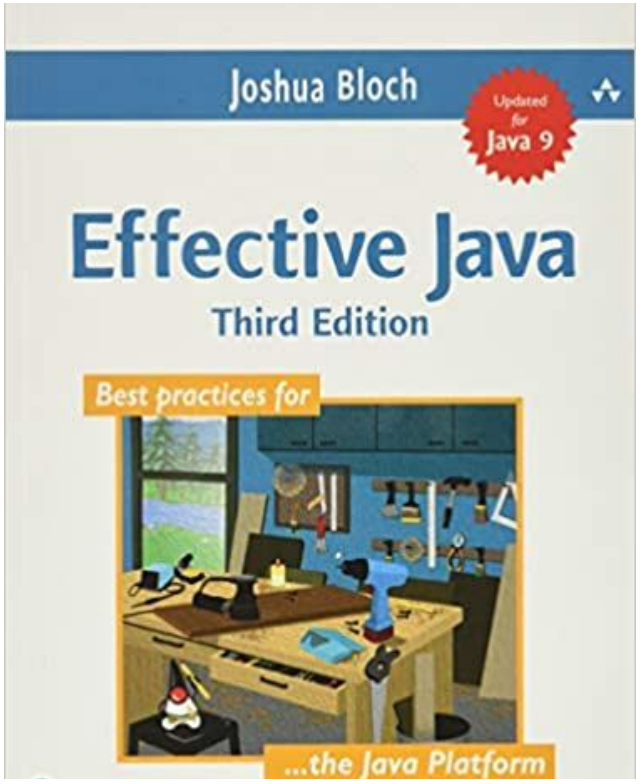

Writing code badly
Maintaining code badly
Teaching badly


BORN
TO
CODE


Robert C. Martin Series
Clean Code
A Handbook of Agile Software Craftsmanship
Foreword by James O. Coplien
Robert C. Martin


Robert C. Martin Series
The Clean Coder
A Code of Conduct for Professional Programmers
Foreword by Matthew Heusser, Software Process Naturalist
Robert C. Martin


20th Anniversary Edition
The Pragmatic Programmer
your journey to mastery
David Thomas
Andrew Hunt

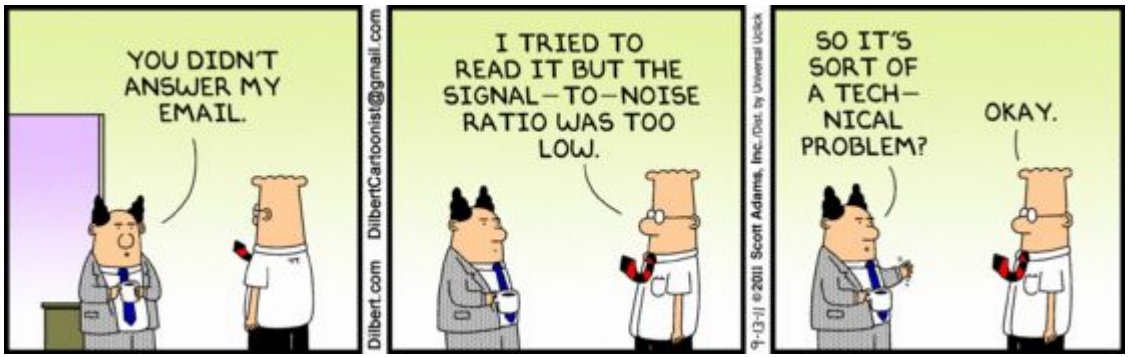

Joshua Bloch
Updated for Java 9
Effective Java
Third Edition
Best practices for
...the Java Platform

„Style matters and there's beauty in simplicity."

Was sollte wichtig sein

- Clarity
- Brevity
- Simplicity
- Humanity*

code-to-noise ratio

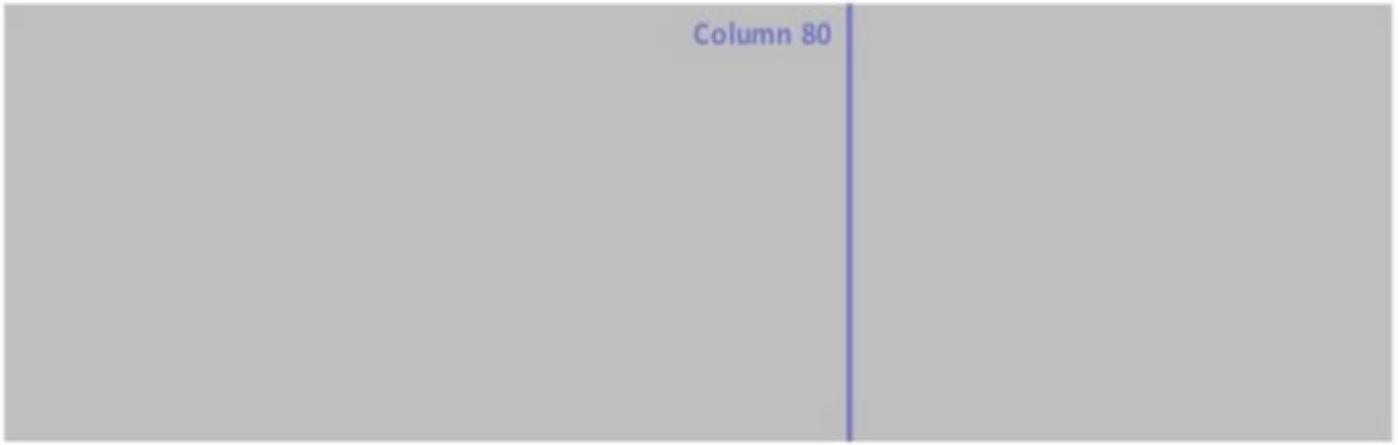


```
public class RecentlyUsedList
{
    private List<string> items;
    public RecentlyUsedList()
    {
        items = new List<string>();
    }
    public void Add(string newItem)
    {
        if (items.Contains(newItem))
        {
            int position = items.IndexOf(newItem);
            string existingItem = items[position];
            items.RemoveAt(position);
            items.Insert(0, existingItem);
        }
        else
        {
            items.Insert(0, newItem);
        }
    }
    public int Count
    {
        get
        {
            int size = items.Count;
            return size;
        }
    }
    public string this[int index]
    {
        get
        {
            int position = 0;
            foreach (string item in items)
            {
                if (position == index)
                    return item;
                ++position;
            }
            throw new ArgumentOutOfRangeException();
        }
    }
}
```

```
public class RecentlyUsedList
{
    private List<string> items = new List<string>();
    public void Add(string newItem)
    {
        items.Remove(newItem);
        items.Add(newItem);
    }
    public int Count
    {
        get
        {
            return items.Count;
        }
    }
    public string this[int index]
    {
        get
        {
            return items[Count - index - 1];
        }
    }
}
```

```
true ->
  case User.find(%{email: email}) do
    {:ok, [user | _]} ->
      # get a list of all unused tokens
      {:ok, tokens} = Token.find(%{user: user.id, type: token_t
      # some of them might be expired
      # => remove them (should be done on the DB!)
      # this seems rather pointless
      active_tokens = tokens
      |> Enum.filter(fn token ->
          token.expires_at > Timex.to_unix(Timex.now)
      end)

      # one token should be active at most
      token = active_tokens |> List.first

      cond do
        is_nil(token) ->
          #token is either expired or doesn`t exist
          # => generate new token
```


LE GRILL WHAT THE
HELL IS THAT

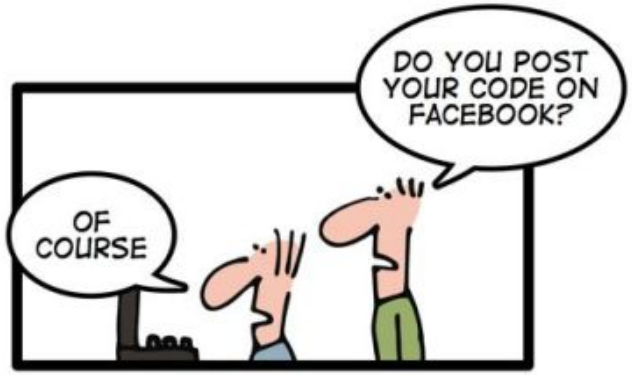


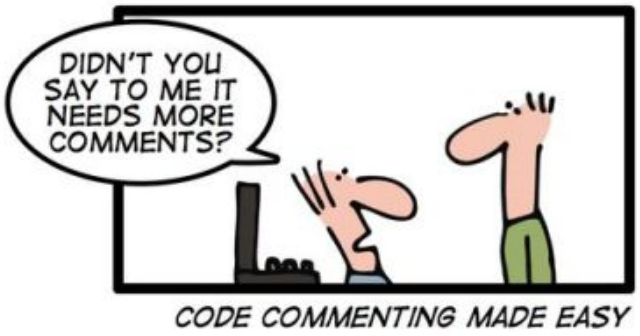
How many programmers lay out their code

How people read

Kommentare

- Erkläre nie, was der Code tut
  - Erkläre, warum es genau so umgesetzt ist
- Auskommentieren von Code ist nicht gewünscht
  - Versionskontrolle existiert
- Kommentiere seltsamen Code - auch wenn es nicht dein ist
- Keine redundanten Kommentare
  - Getter/Setter etc

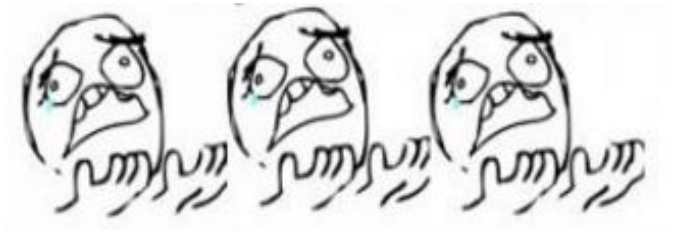




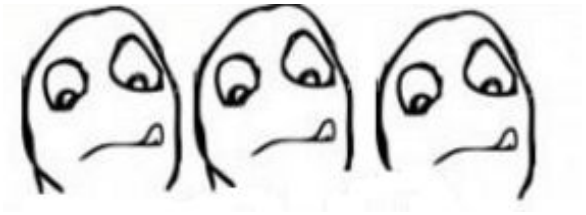

Clean Code

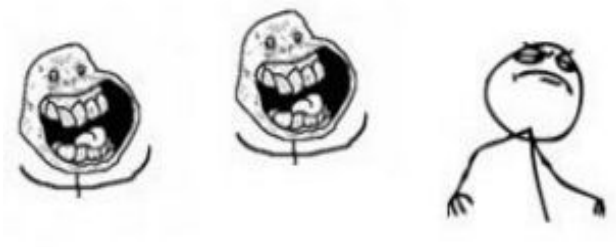
"Clean code can be read, and enhanced by a developer other than its original author."

SOLL



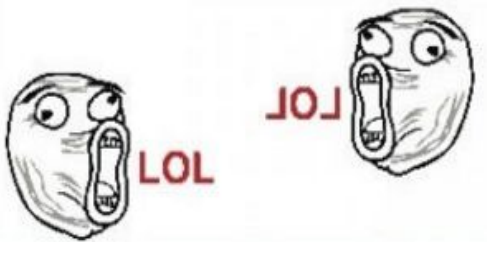

Okay
Okay
Okay



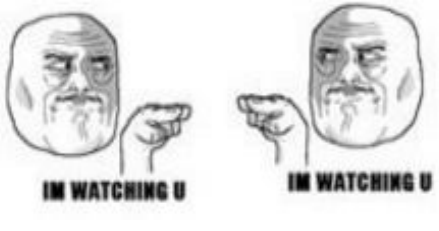


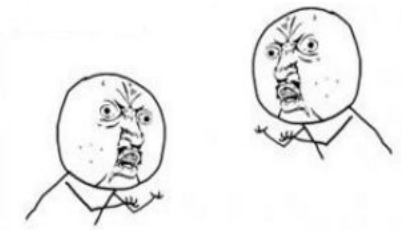
IST


CHALLENGE ACCEPTED
CHALLENGE ACCEPTED

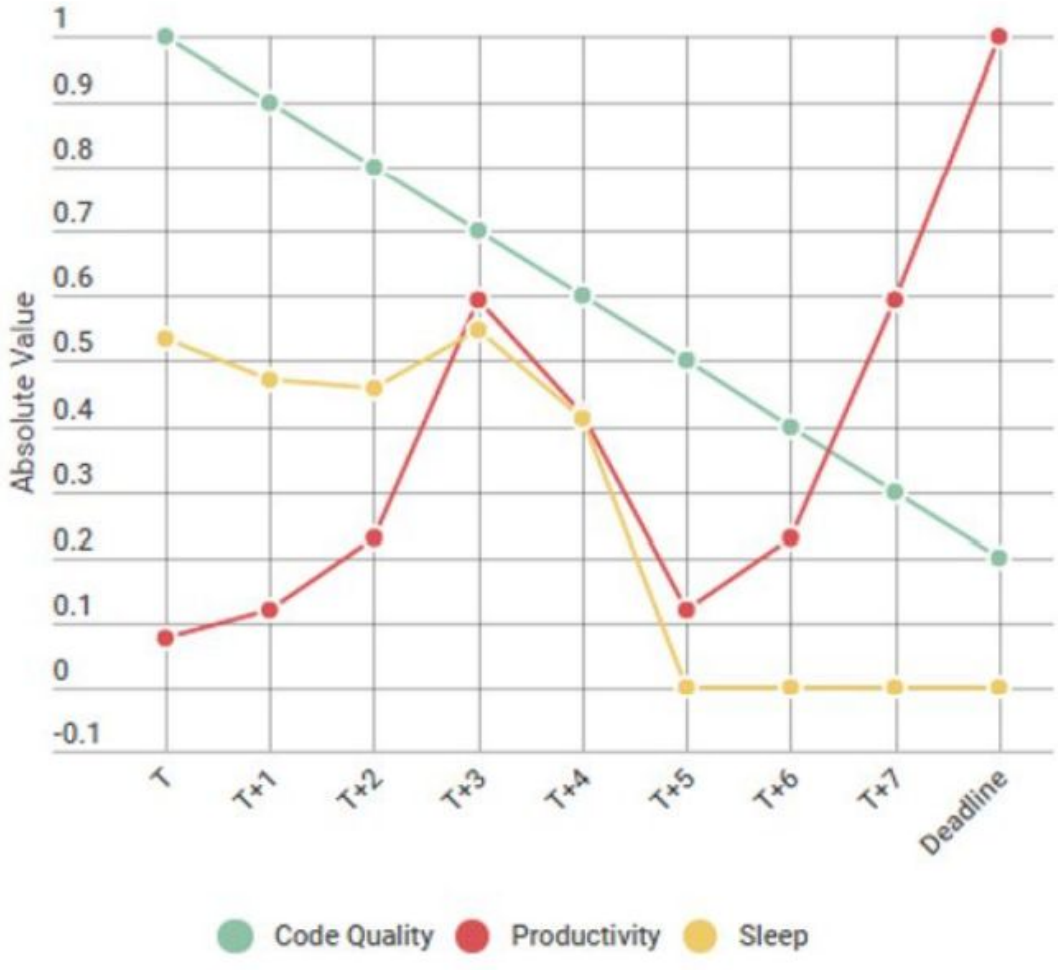

LOL
LOL


PFF
PFF

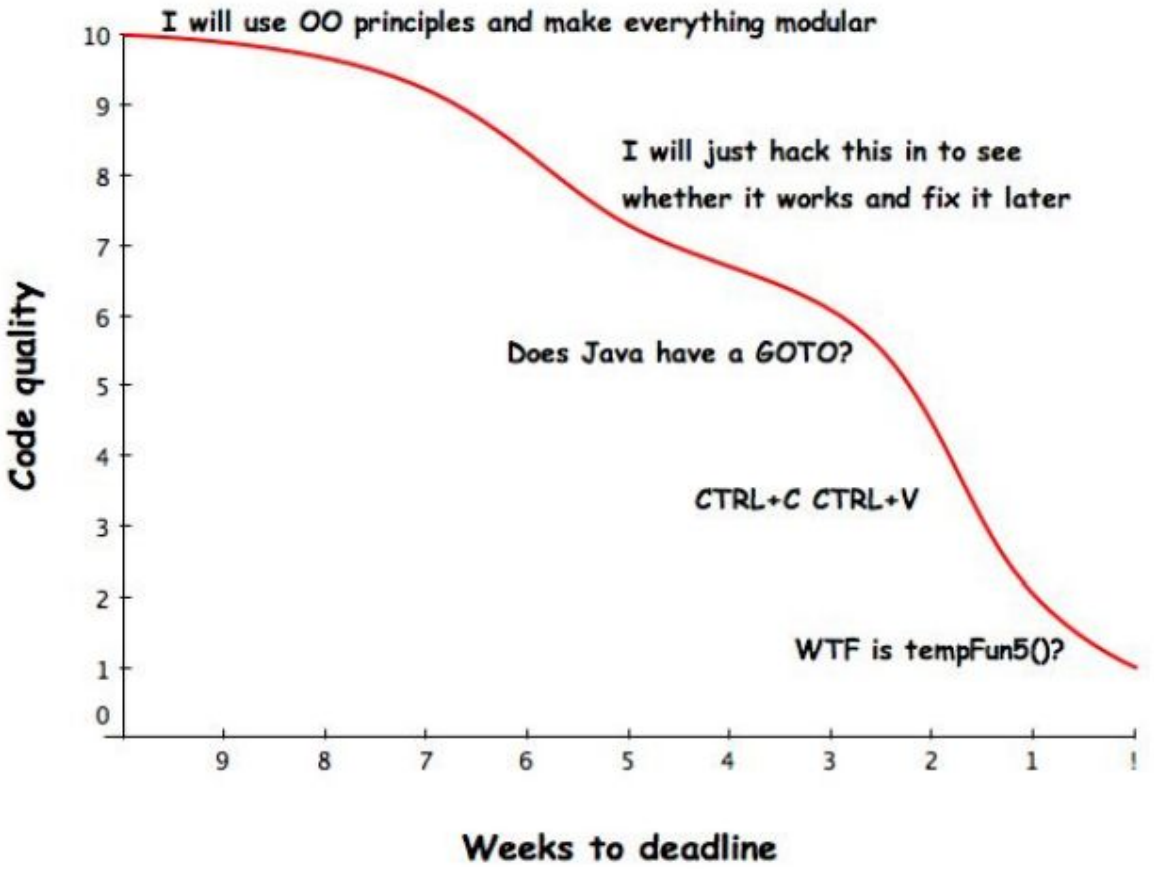

IM WATCHING U
IM WATCHING U



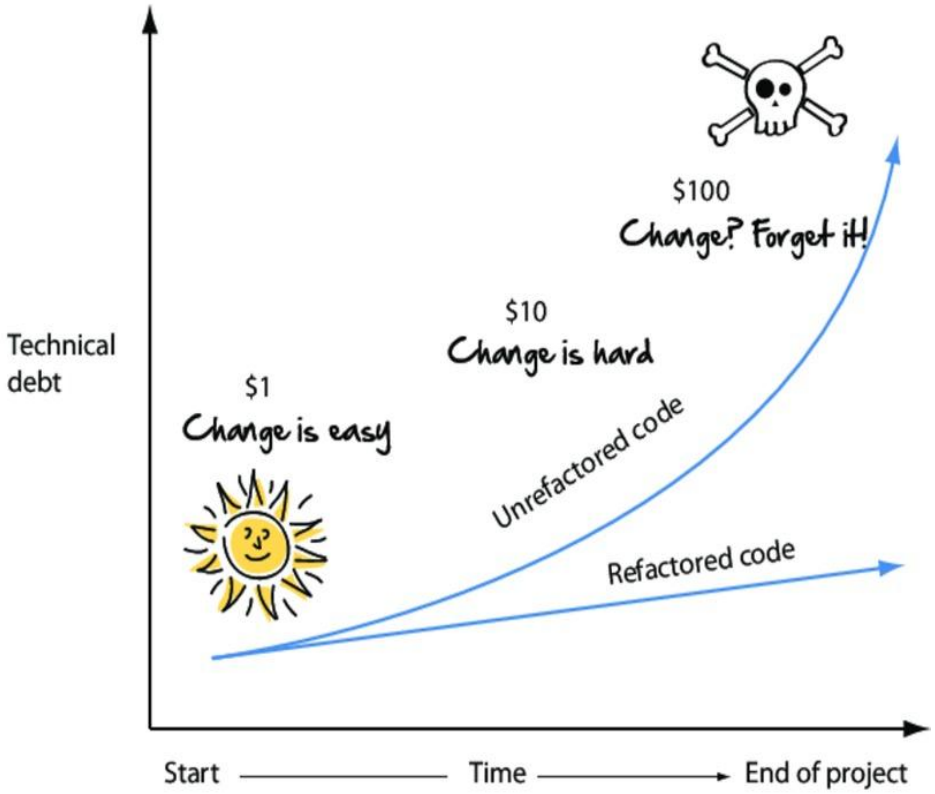
Technical Debt

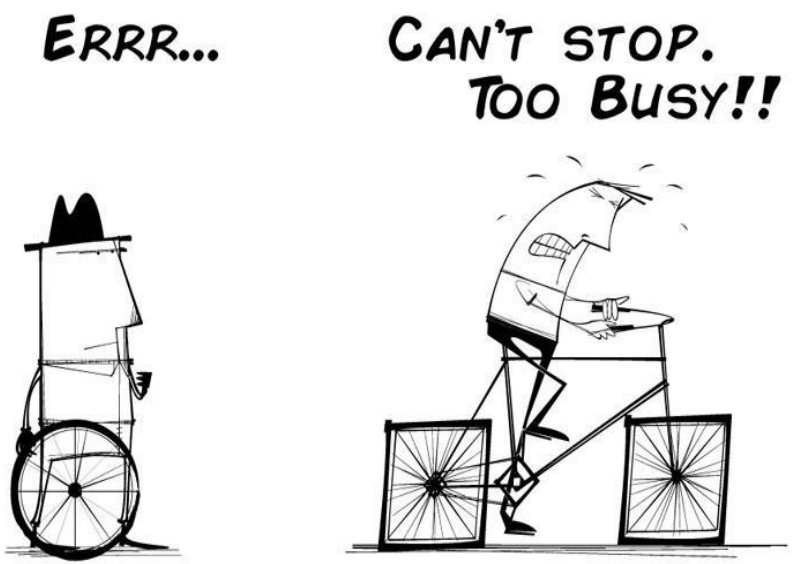


Technical Debt



Technical Debt





Technical debt

is...well...debt and it may or may not be evil

Keep the code at top quality and stop it from “rotting*”

*as if code is alive, or is it?

„All sucessful software gets changed"

Fred Brooks, The mythical man month

S.O.L.I.D.-Prinzipien

- Fünf Prinzipien für wartbaren Code
- Nicht an eine Technologie gebunden

S.O.L.I.D.-Prinzipien

?

The Single Responsibility Principle

Single Responsability Principle

```
public void erstelleProjekt(String projektName, String projektManager) {
     if (projektName == null || "".equals(projektName)) {
          throw new IllegalArgumentException("argument projektName darf nicht null oder leer sein");
     }

     if (projektManager == null || "".equals(projektManager)) {
          throw new IllegalArgumentException("argument projektManager darf nicht null oder leer sein");
     }

     TypedQuery<User> query = entityManager.createQuery("select u from User u where u.name=:name", User.class);
     query.setParameter("name", projektManager);
     try {
          User user = query.getSingleResult();

          Projekt projekt = new Projekt(projektName);
          projekt.setProjektManager(user);

          entityManager.persist(projekt);
     } catch (EntityNotFoundException e) {
          throw new ProAdmException("projekt konnte nicht erstellt werden.");
     }
}
```

Single Responsability Principle

```
public void erstelleProjekt(String projektName, String projektManager) {
    if (projektName == null || "".equals(projektName)) {
        throw new IllegalArgumentException("argument projektName darf nicht null oder leer sein");
    }

    if (projektManager == null || "".equals(projektManager)) {
        throw new IllegalArgumentException("argument projektManager darf nicht null oder leer sein");
    }

    TypedQuery<User> query = entityManager.createQuery("select u from User u where u.name=:name", User.class);
    query.setParameter("name", projektManager);
    try {
        User user = query.getSingleResult();

        Projekt projekt = new Projekt(projektName);
        projekt.setProjektManager(user);

        entityManager.persist(projekt);
    } catch (EntityNotFoundException e) {
        throw new ProAdmException("projekt konnte nicht erstellt werden.");
    }
}
```

Single Responsability Principle

```
@Inject
private SolidValidator validator;

public void erstelleProjekt(String projektName, String projektManager) {
     validator.validateProjektName(projektName);
     validator.validateProjektManagerName(projektManager);

     TypedQuery<User> query = entityManager.createQuery("select u from User u where u.name=:name", User.class);
     query.setParameter("name", projektManager);
     try {
          User user = query.getSingleResult();

          Projekt projekt = new Projekt(projektName);
          projekt.setProjektManager(user);

          entityManager.persist(projekt);
     } catch (EntityNotFoundException e) {
          throw new ProAdmException("projekt konnte nicht erstellt werden.");
     }
}
```

Single Responsability Principle

```
@Inject
private SolidValidator validator;

public void erstelleProjekt(String projektName, String projektManager) {
     validator.validateProjektName(projektName);
     validator.validateProjektManagerName(projektManager);

     TypedQuery<User> query = entityManager.createQuery("select u from User u where u.name=:name", User.class);
     query.setParameter("name", projektManager);
     try {
          User user = query.getSingleResult();

          Projekt projekt = new Projekt(projektName);
          projekt.setProjektManager(user);

          entityManager.persist(projekt);
     } catch (EntityNotFoundException e) {
          throw new ProAdmException("projekt konnte nicht erstellt werden.");
     }
}
```

Single Responsability Principle

```java
@Inject
private SolidValidator validator;

@Inject
private BenutzerRepository users;

public void erstelleProjekt(String projektName, String projektManager) {
    validator.validateProjektName(projektName);
    validator.validateProjektManagerName(projektManager);

    Benutzer user = users.getProjektManager(projektManager);
    Projekt projekt = new Projekt(projektName);
    projekt.setProjektManager(user);

    entityManager.persist(projekt);
}
```

Single Responsability Principle

```java
@Inject
private SolidValidator validator;

@Inject
private BenutzerRepository users;

@Inject
private ProjektRepository projects;

public void erstelleProjekt(String projektName, String projektManager) {
    validator.validateProjektName(projektName);
    validator.validateProjektManagerName(projektManager);

    Benutzer user = users.getProjektManager(projektManager);
    Projekt projekt = new Projekt(projektName);
    projekt.setProjektManager(user);

    projects.saveProjekt(projekt);
}
```

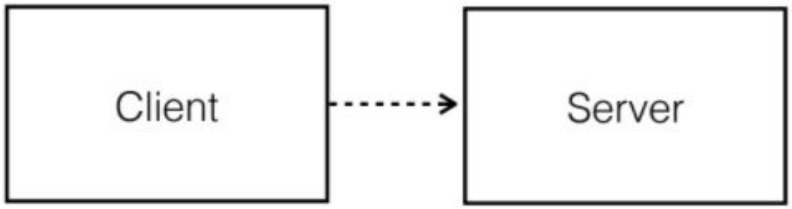
Single Responsability Principle

- Unterschiedliche Aspekten fachlich (Prüfungen,Aktivitäten, …) und technisch (Performance, Sicherheit, UI, Persistenz, …) in der selben Klasse
- Wie viele Methoden verwenden dasselbe Feld?
- Mehrere Verantwortungen
- Verantwortung ist in diesem Kontext analog zu „Grund für eine Änderung“ sehen

The Open/Closed Principle

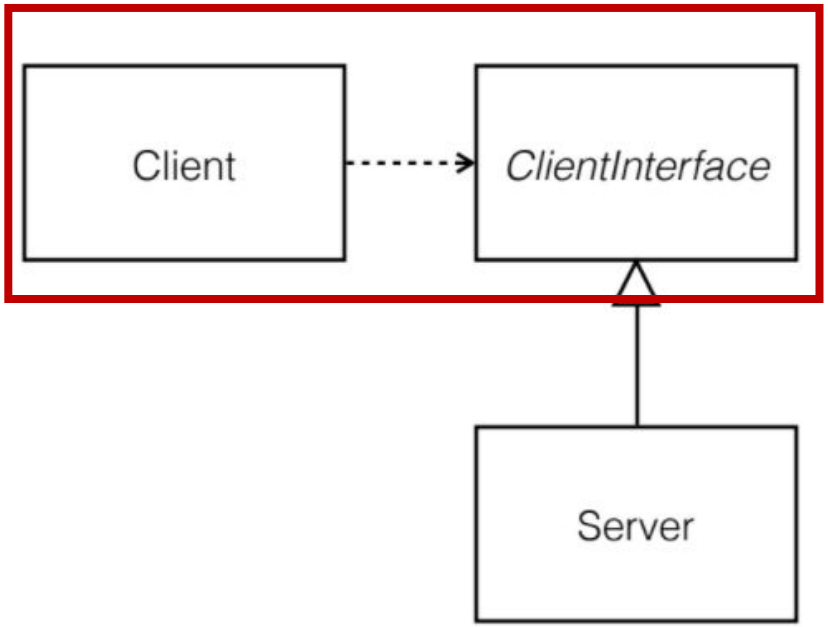
The Open/Closed Principle

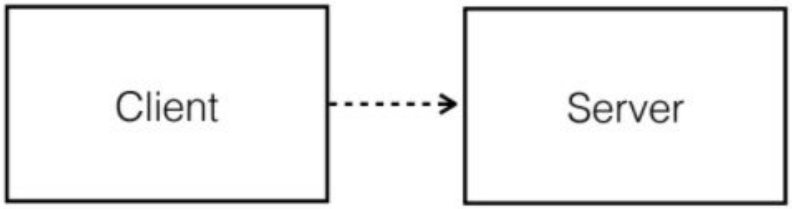
- Welche Methoden darf ich verwenden?
- Welche Methoden werden eigentlich benötigt?



The Open/Closed Principle

- Welche Methoden darf ich verwenden?
- Welche Methoden werden eigentlich benötigt?

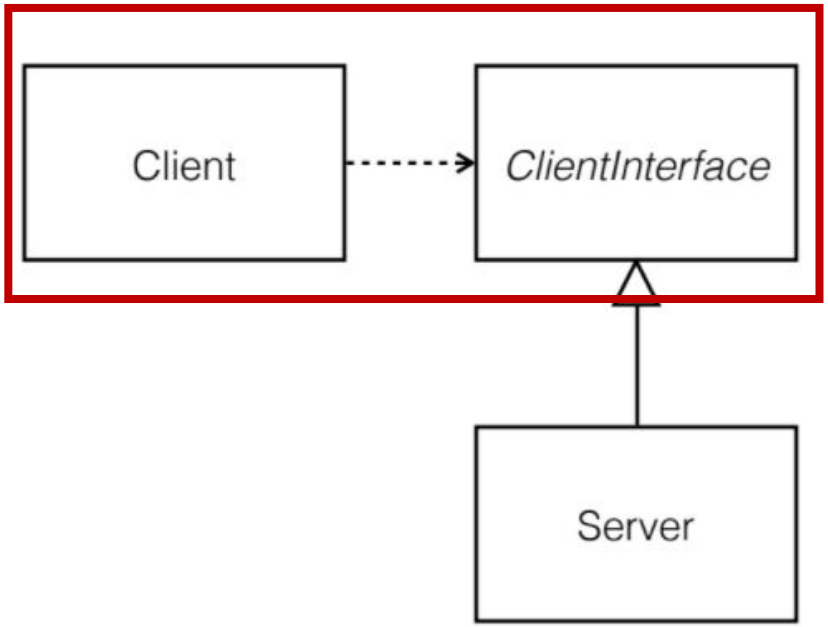


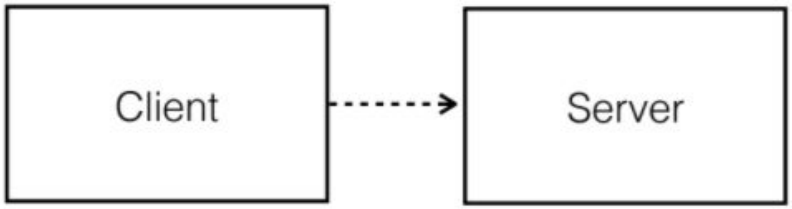


The Open/Closed Principle

- Welche Methoden darf ich verwenden?
- Welche Methoden werden eigentlich benötigt?

- closed for modification
- open for extension





Beispiel

- Neue Anforderung?
- Projektstart/Projektende (beide Meilensteine)

```
public enum Typ {
    ARBEITSPAKET, MEILENSTEIN
}

…


public Arbeitspaket findeErstenMeilenstein(List<Arbeitspaket> arbeitspakete){
    for (Arbeitspaket arbeitspaket : arbeitspakete) {
        if(Typ.MEILENSTEIN == arbeitspaket.getTyp()){
                return arbeitspaket;
        }
    }
    return null;
}
```

Beispiel

- Neue Anforderung?
- Projektstart/Projektende

```
public enum Typ {
    ARBEITSPAKET, MEILENSTEIN
}

…


public Arbeitspaket findeErstenMeilenstein(List<Arbeitspaket> arbeitspakete){
    for (Arbeitspaket arbeitspaket : arbeitspakete) {
        if (arbeitspaket.getTyp().isMeilenstein()) {
            return arbeitspaket;
        }
    }
    return null;
}
```

Beispiel

- Neue Anforderung?
- Projektstart/Projektende

```
public enum Typ {
    ARBEITSPAKET, MEILENSTEIN
}

…
```

```
public Arbeitspaket findeErstenMeilenstein(List<Arbeitspaket> {
    for (Arbeitspaket arbeitspaket : arbeitspakete) {
        if (arbeitspaket.getTyp().isMeilenstein()) {
            return arbeitspaket;
        }
    }
    return null;
}
```

```
public interface ArbeitspaketTyp {
    boolean isMeilenstein();
    boolean isArbeitspaket();
}
```

```
public enum Typ implements ArbeitspaketTyp{
    ARBEITSPAKET {
        @Override
        public boolean isArbeitspaket() {
            return true;
        }
    },
    PROJEKTSTART {
        @Override
        public boolean isMeilenstein() {
            return true;
        }
    },
    PROJEKTENDE {
        @Override
        public boolean isMeilenstein() {
            return true;
        }
    },
    MEILENSTEIN {
        @Override
        public boolean isMeilenstein() {
            return true;
        }
    };

    public boolean isMeilenstein() {
        return false;
    }

    public boolean isArbeitspaket() {
        return false;
    }
}
```

The Open/Closed Principle

- closed for modification
- open for extension

- Änderungen einer Klasse (zB das Hinzufügen/Entfernen einer Methode) werden in vielen anderen Klassen gespiegelt

- Schnittstellen entsprechen den Anforderungen des Anbieters und nicht des Clients

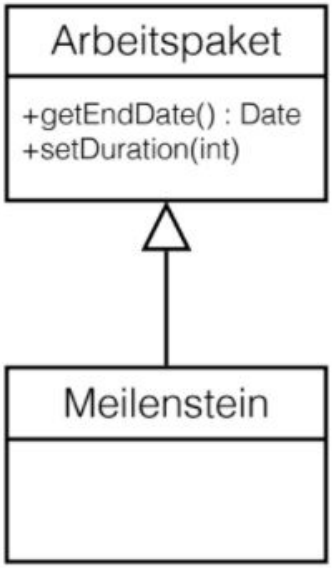
The Liskov Substitution Principle

The Liskov Substitution Principle

- Eine neue Klasse ist erforderlich: Meilensteine

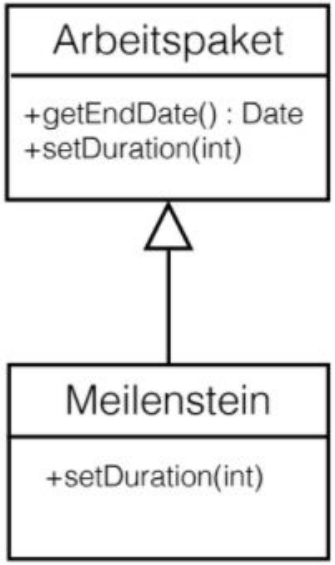
The Liskov Substitution Principle

- Eine neue Klasse ist erforderlich: Meilensteine



The Liskov Substitution Principle

- Eine neue Klasse ist erforderlich: Meilensteine
- Darf Beginn- und Enddatum eines Meilensteins voneinander abweichen?

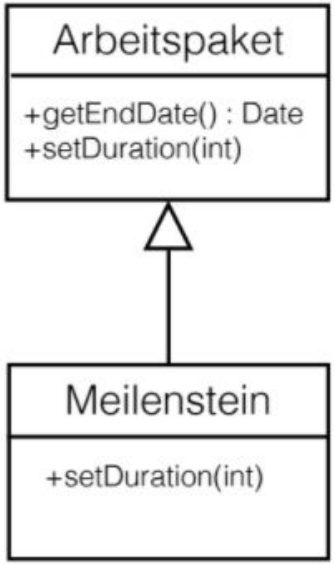


The Liskov Substitution Principle

- Eine neue Klasse ist erforderlich: Meilensteine
- Dürfen Beginn- und Enddatum eines Meilensteins voneinander abweichen?

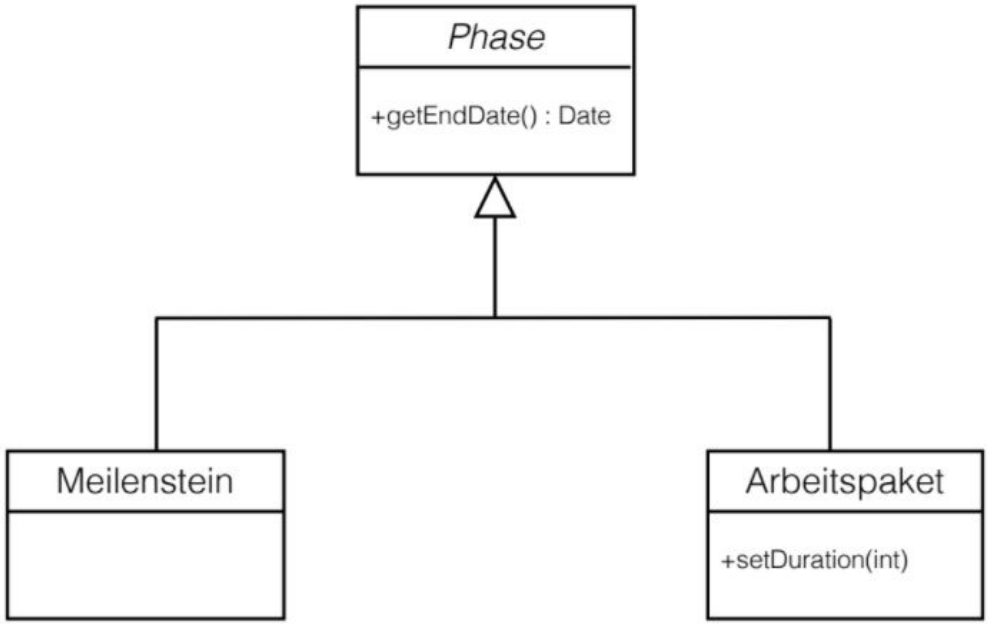
```
public class Meilenstein extends Arbeitspaket{
    public void setDuration(int days){
        throw new ProAdmException("unsupported operation.");
    }
}
```

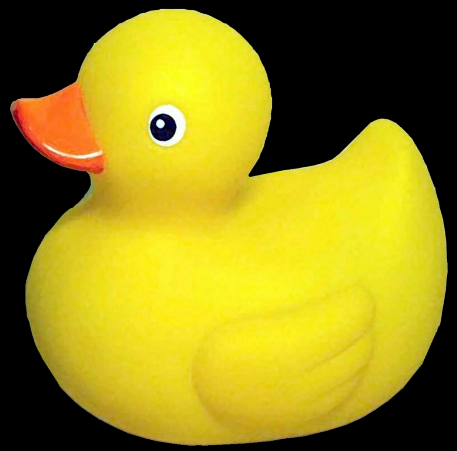
```
public class Meilenstein extends Arbeitspaket{
    public void setDuration(int days){
        // ignore
    }
}
```



The Liskov Substitution Principle

- Eine neue Klasse ist erforderlich: Meilensteine

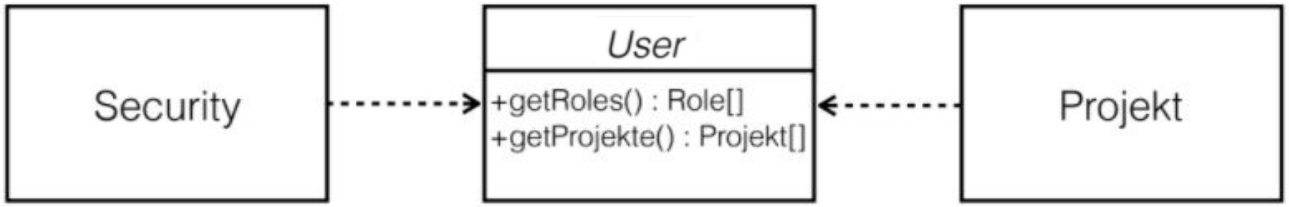




The Interface Segreggation Principle

The Interface Segregation Principle

- High Cohesion (Kohäsion)?
- wie gut ist die Klasse entworfen?
  - mit einem einzigen Ziel
- Schnittstellen sind klein, spezialisiert -> High Cohesion



Cohesion

- ist wichtig für einzelne Klassen
- beschreibt die Vielfalt und Anzahl von Aufgaben, für die eine Klasse zuständig ist
- wenn eine Klasse genau für eine Aufgabe verantwortlich ist, dann spricht man von hoher Kohäsion
- Ziel: hohe Kohäsion

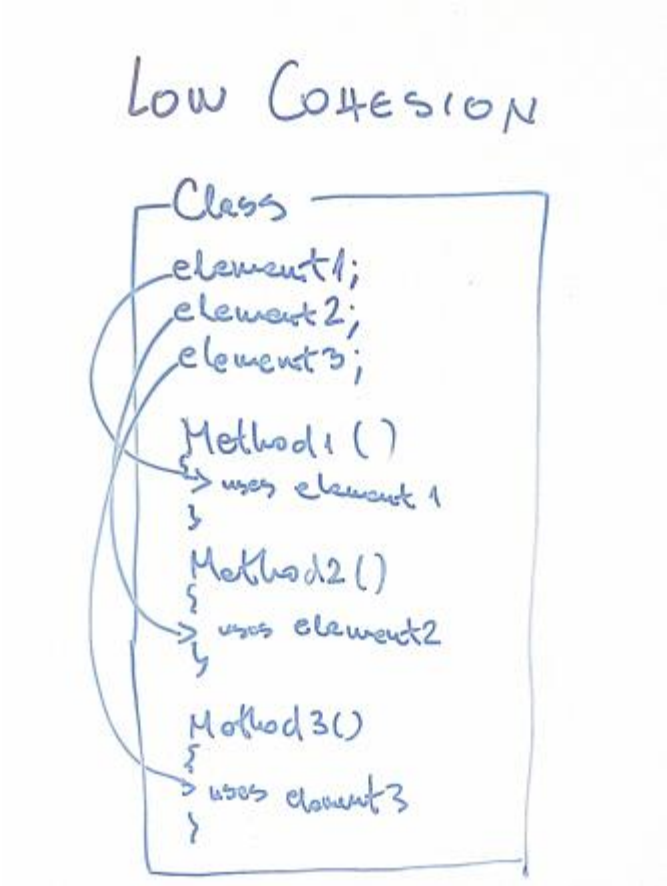
Cohesion

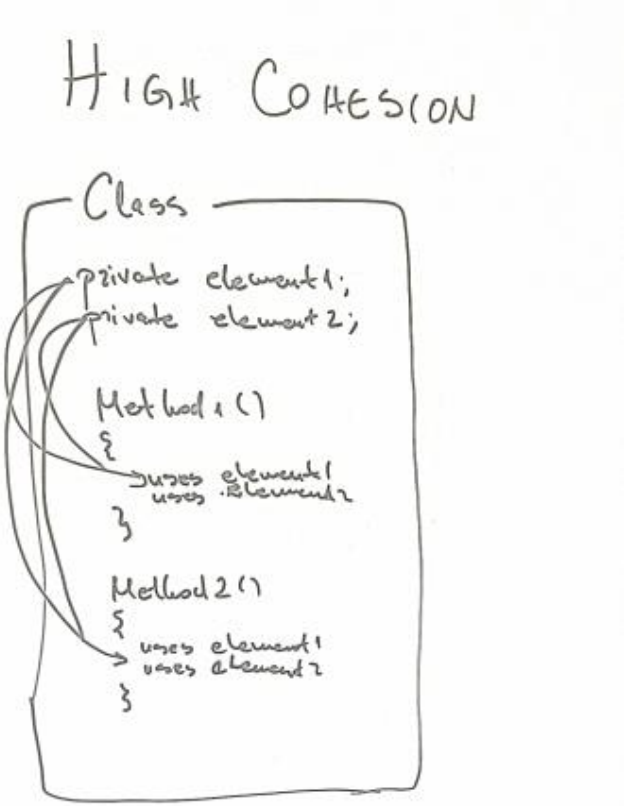
Hohe Cohesion unterstützt:

- zu verstehen was eine Klasse macht
- Klassen wieder zu verwenden
- reduziert Komplexität
- erhöht Wartbarkeit
- treffende Klassennamen zu finden

Cohesion

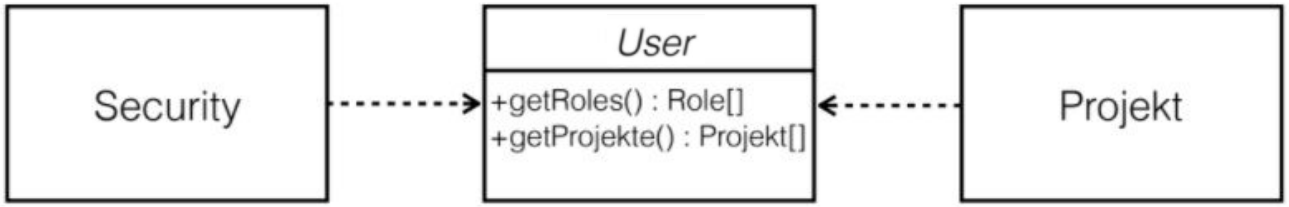
eine Klasse sollte eine klar definierte Einheit modellieren





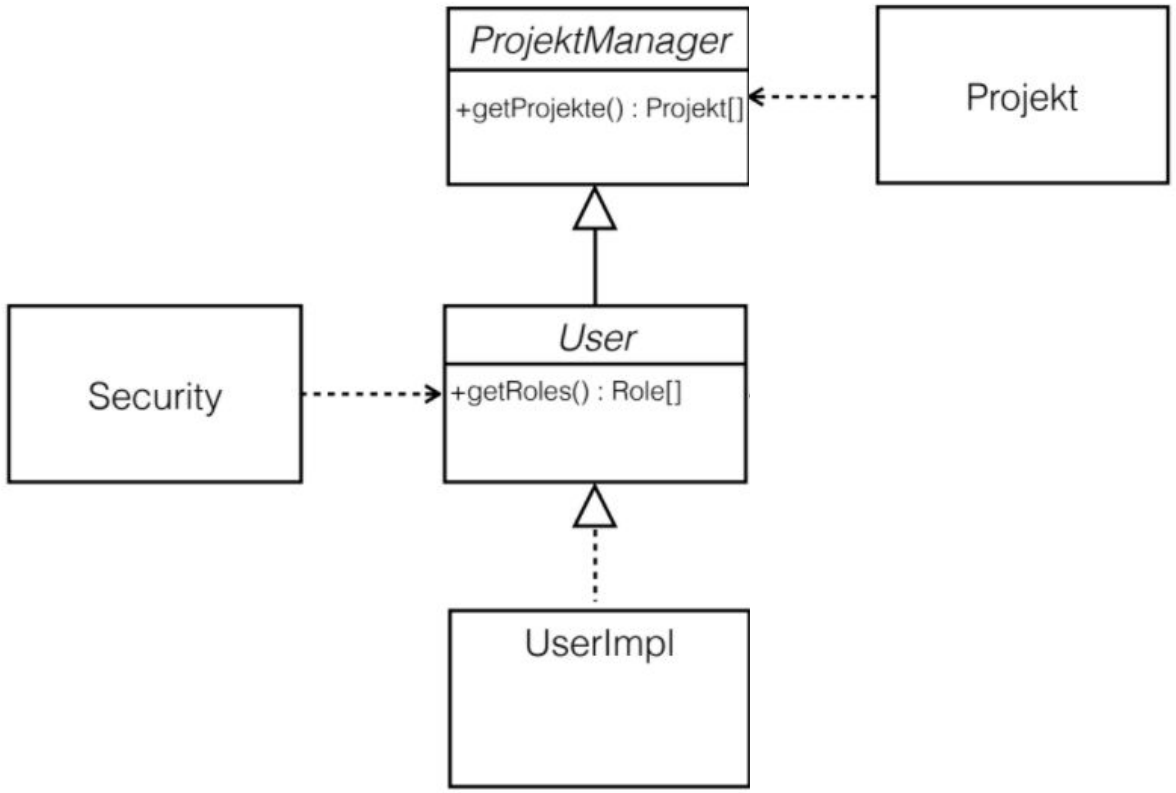
The Interface Segregation Principle

- High Cohesion
- Schnittstellen sind klein, spezialisiert -> High Cohesion

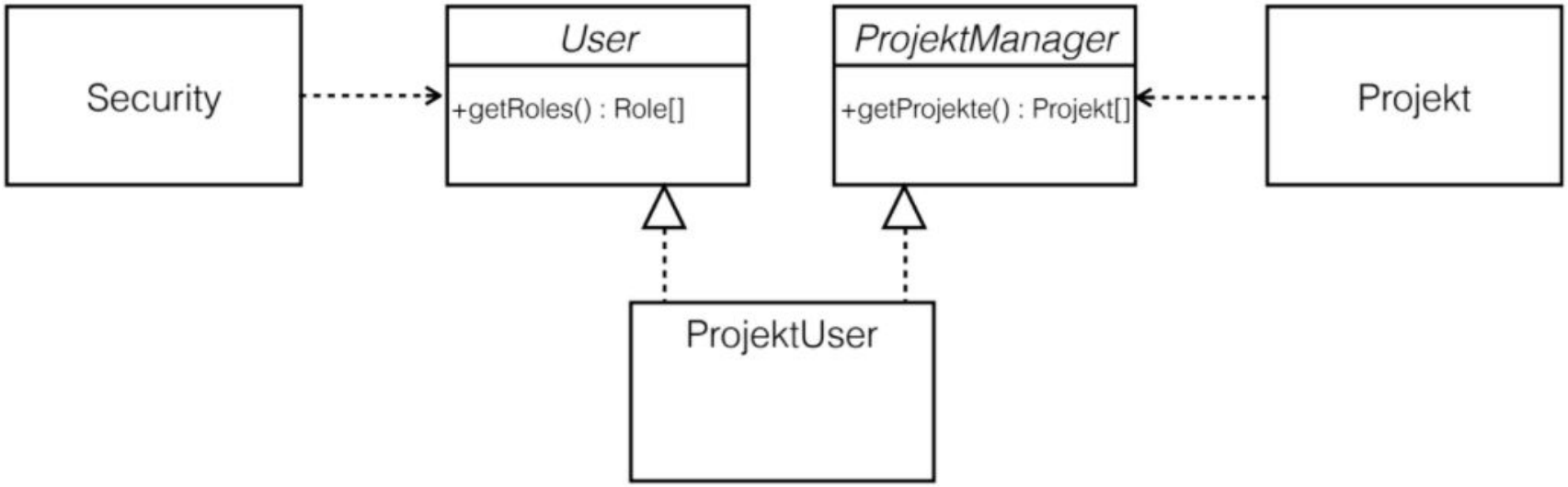


The Interface Segregation Principle

man muss Methoden implementieren die in der konkreten Klasse gar keine Funktion erfüllen



The Interface Segregation Principle

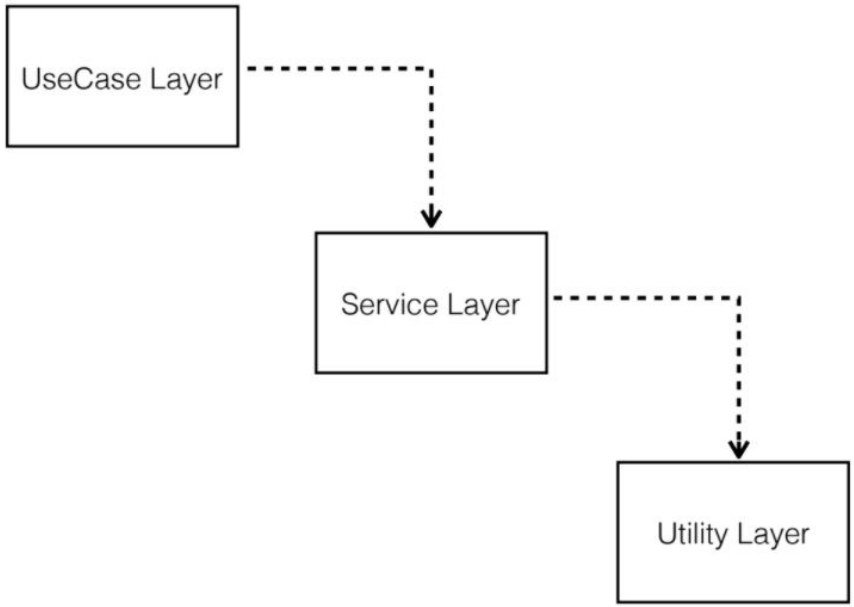


The Dependency Inversion Principle

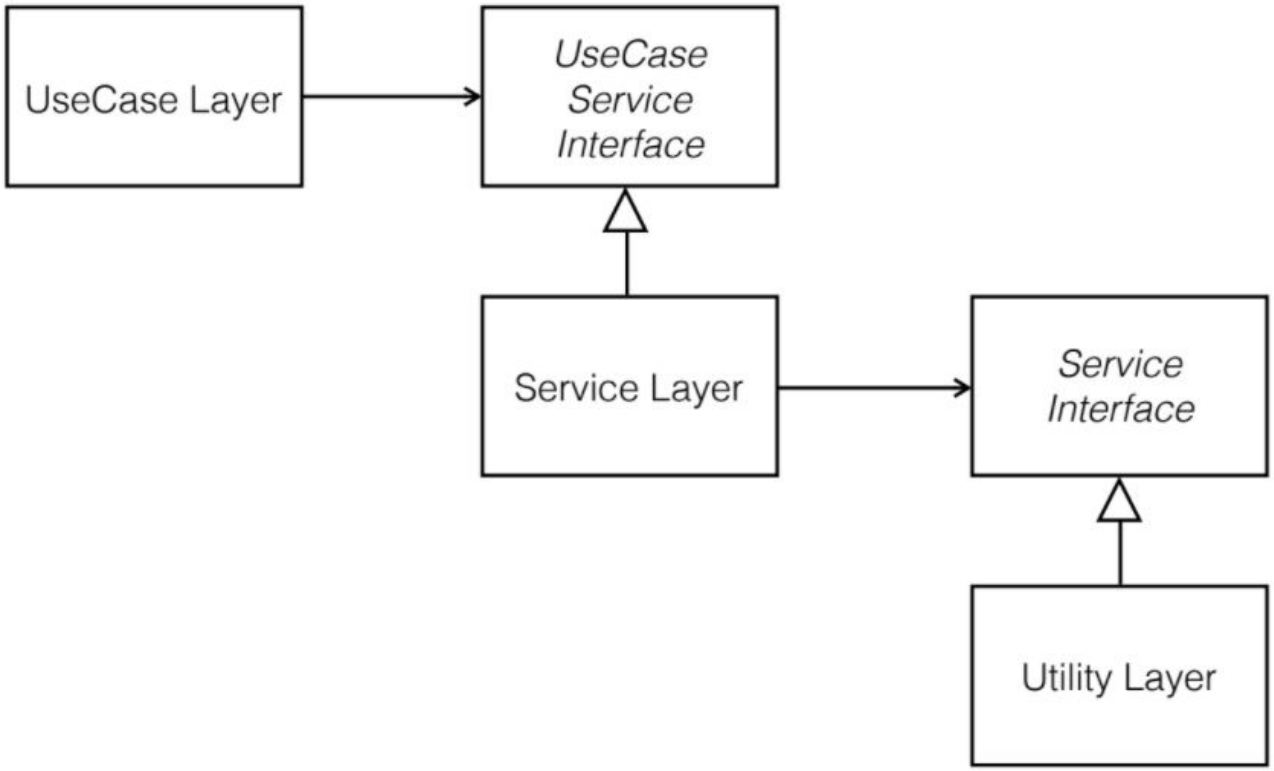
The Dependency-Inversion Principle

- High-Level-Module sollten nicht von Low-Level-Modulen abhängen
  - Beides sollte von Abstraktionen abhängen

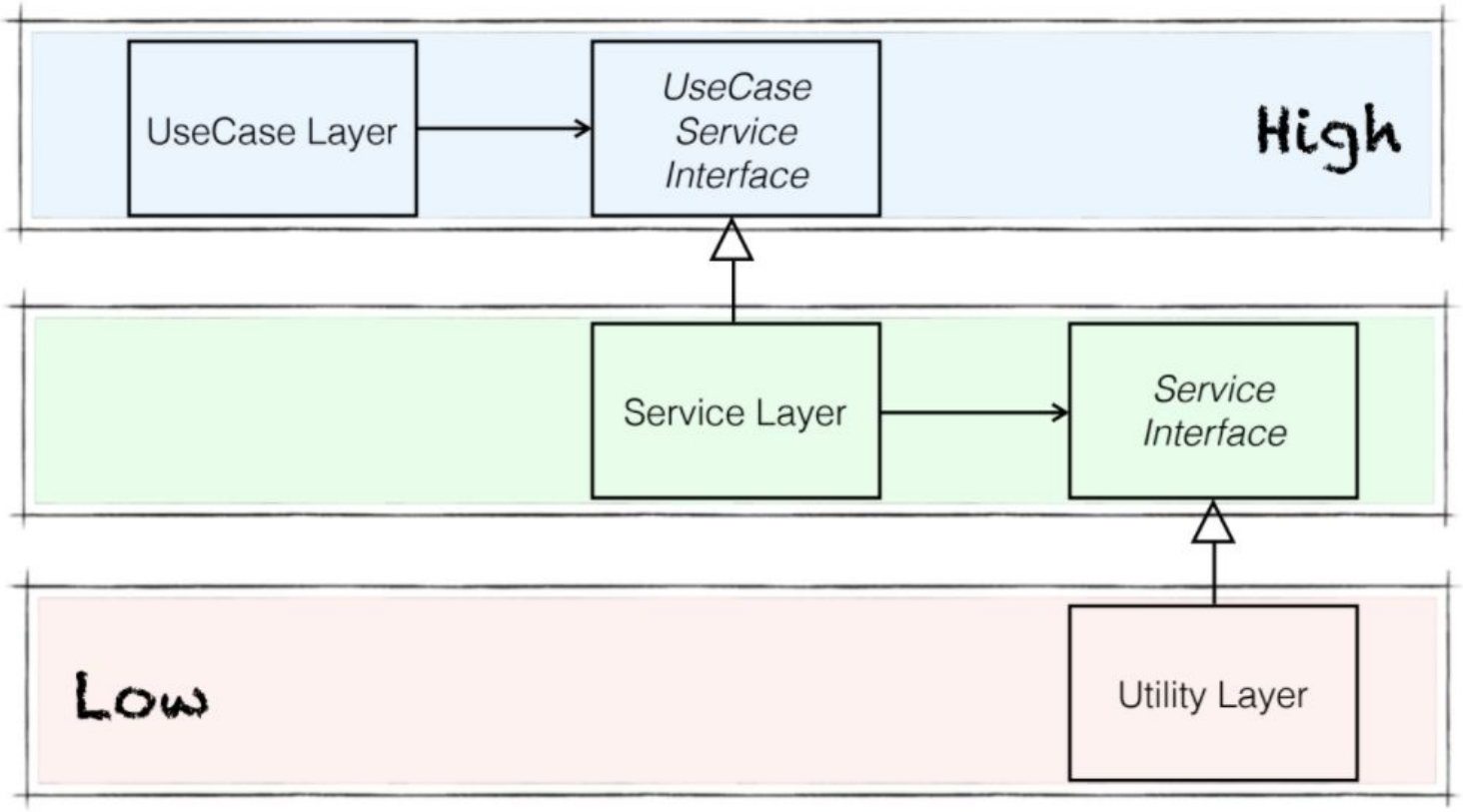
- Abstraktionen sollten nicht von Details abhängen
  - Details sollten von Abstraktionen abhängen



The Dependency-Inversion Principle



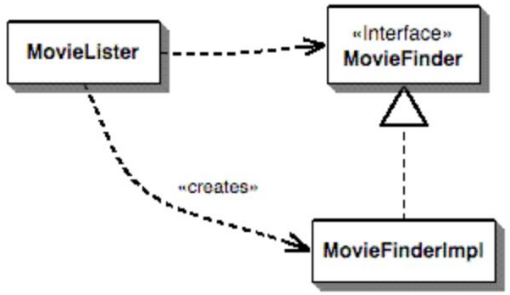
The Dependency-Inversion Principle



The Dependency-Inversion Principle (refactoring.com)

class MovieLister...

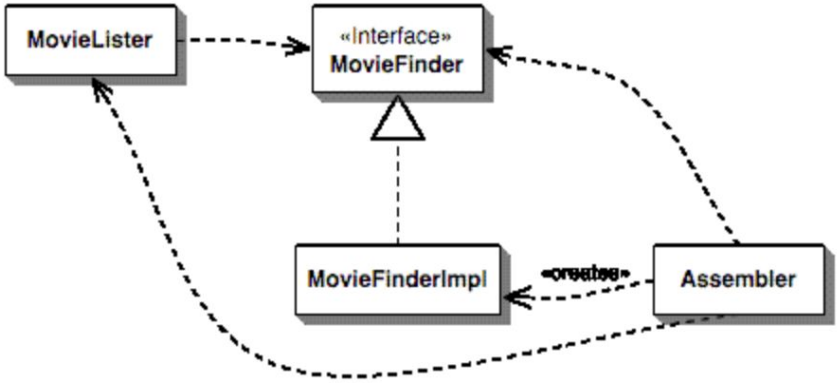
```
public Movie[] moviesDirectedBy(String arg) {
    List allMovies = finder.findAll();
    for (Iterator it = allMovies.iterator(); it.hasNext();) {
        Movie movie = (Movie) it.next();
        if (!movie.getDirector().equals(arg)) it.remove();
    }
    return (Movie[]) allMovies.toArray(new Movie[allMovies.size()]);
}
```




```
public interface MovieFinder {
    List findAll();
}
```

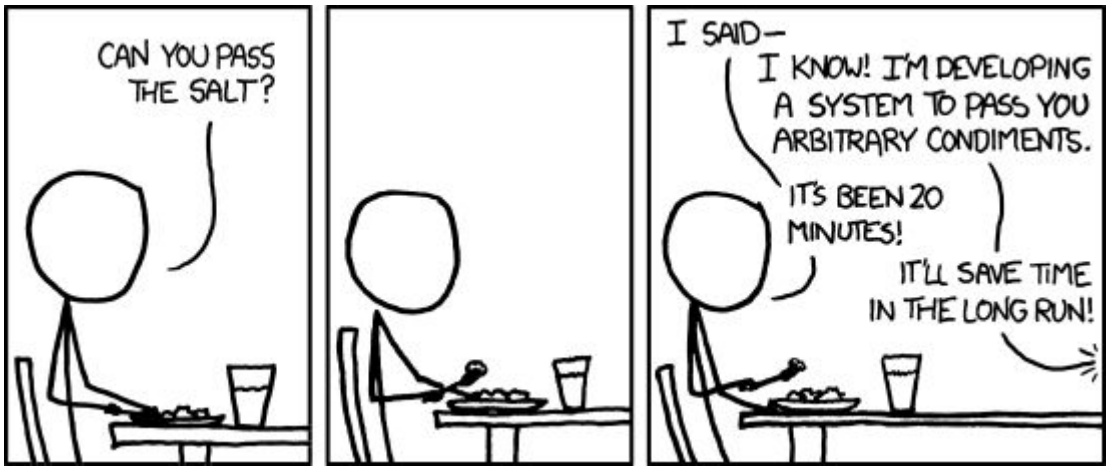
class MovieLister...

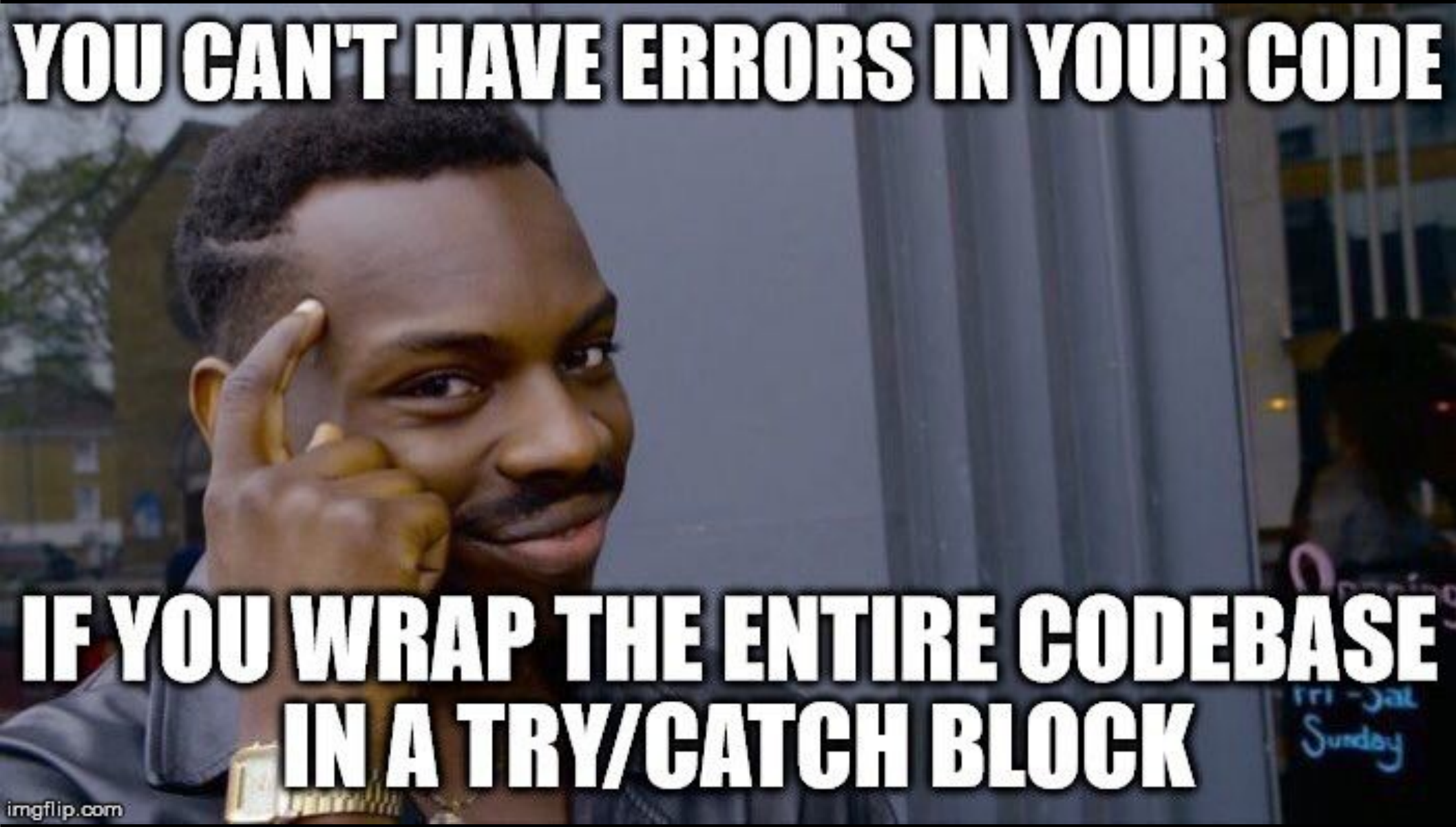
```
public MovieLister(MovieFinder finder) {
    this.finder = finder;
}
```



class MovieLister...

```
private MovieFinder finder;
public MovieLister() {
  finder = new ColonDelimitedMovieFinder("movies1.txt");
}
```


CAN YOU PASS THE SALT?
I SAID—
I KNOW! I'M DEVELOPING A SYSTEM TO PASS YOU ARBITRARY CONDIMENTS.
IT'S BEEN 20 MINUTES!
IT'LL SAVE TIME IN THE LONG RUN!


YOU CAN'T HAVE ERRORS IN YOUR CODE
IF YOU WRAP THE ENTIRE CODEBASE IN A TRY/CATCH BLOCK
Sunday
imgflip.com

NullPointerException

NullPointerException

```
private Boolean isFinished(String status) {
    if (status.equalsIgnoreCase("Finished") {
        return true;
    }

    return false;
}


private Boolean isFinished(String status) {
    if ("Finished".equalsIgnoreCase(status) {
        return true;
    }

    return false;
}
```

NullPointerException

```
BigDecimal price = getPrice();

//if price is null?
System.Out.println(String.valueOf(price));
System.Out.println(price.toString());
//throws java.lang.NullPointer.Exception


StringUtil.isEmpty (= String.isEmpty + null check)
StringUtil.isBlank
StringUtil.isNumeric
//^ are null safe
```

NullPointerException

```
String capitalize(@NonNull String in) {
    return in.toUpperCase(); //static code analysis → precondition
}

/*
* @param in should not be null
*/
String capitalize(String in) {
    return in.toUpperCase();
}


void caller(String s) {
    if (s!=null)
        System.Out.println(capitalize(s)) //nullcheck
}
```

Null Pattern

- als Alternative zur einer Nullprüfung im Clientcode kann man ein Nullobjekt zurückgegeben

```
public interface Bird {
    void fly() ;
}

public class EagleFang implements Bird {
    public void Fly() {
        System.out.println("flying...");
    }
}

public class NullObject implements Bird {
    public void fly() {
                // really?...do nothing
    }
}
```

Null Pattern

- als Alternative zur einer Nullprüfung im Clientcode kann man ein Nullobjekt zurückgegeben

```
public interface Bird {
    void fly() ;
}

public class EagleFang implements Bird {
    public void Fly() {
        System.out.println("flying...");
    }
}

public class NullObject implements Bird {
    public void fly() {
                // really?...do nothing
    }
}
```

Null Pattern

```
public class BirdHandler {
    public void handle(List<String> types) {
        for (String type : types) {
            Bird bird = BirdFactory.getBird(type);
            bird.fly(msg);
        }
    }
}
```

NullPointerException

```
public Optional<Customer> findCustomerByName(String name) {
    //hard work
}
```

```
Optional<Customer> optionalCustomer = findCustomerByName ("Nelson
Diamantu");
```

```
if (optionalCustomer.isPresent()) {
    Customer customer = optional.get();
    //do stuff
}
else {
    //not there; deal with it!
}
```

Exception Handling

- verstecke es einfach nicht

```
catch (NoSuchMethodException e) {
    return null;  //the cause of the exception is now forever lost
}

Throw specific exceptions in your methods

public void foo throws Exception {
    //avoid this
}

public void foo throws SpecificException, EvenMoreSpecificException
{
    //atta boy
}
```

Exception Handling

- vermeide `Exception`

```
try {
     someMethod();
} catch (Exception e) {
    Logger.log(“call failed”, e);
}
```

- `Wrapping`

```
catch (NoSuchMethodException e) {
    throw new MyServiceException (“Info: “ + e.getMessage());
//incorrect, stack trace lost
}

catch (NoSuchMethodException e) {
    throw new MyServiceException (“Info: “, e); //correct
}
```

Exception Handling

- Entweder `log` oder `throw`. Nie beides

```
catch (NoSuchMethodException e) {
    Logger.log(“call failed”, e);
    throw e;
}
```

- Exceptions und `finally`

```
try {
     someMethod(); //throws exception
} finally (Exception e) {
    cleanUp();
    throw e; //exception will be lost if we throw here
}
```

Exception Handling

```
void f() throws CustomeException2 {
   try {
       int x = 1/0;
   }
   catch (Exception y) {
       System.out.print("catch...");
       throw new CustomeException1();
   }
   finally {
       System.out.print("finally");
       throw new CustomeException2();
   }
}
```

```
try {
   f();
} catch (Exception ex) {
   ex.printStackTrace();
}


//catch…finallyCustomeException
2
```

Exception Handling

- catch nur wenn die Ausnahme behandeln werden kann

```
catch (NoSuchMethodException e) {
    throw e; //throwing this doesn’t really help
}
```

- finally wenn die Ausnahme nicht behandeln wird

```
try {
     someMethod(); //throws exception
} finally {
    cleanUp();
}
```

Throw early, catch late!

Exception Handling

- man sollte immer nach dem Umgang mit Ausnahmen aufräumen
- nur relevante Ausnahmen sollten ausgelöst werden
- Ausnahmen sollten den Fluss im Code nicht bestimmen
- einzelner Protokolleintrag pro Ausnahme

„Ommit needless words”

"Klassen sollten unveränderlich sein, außer wenn einen sehr guten Grund existiert, sie veränderlich zu machen"

unveränderliche Objekte

Immutable = ein schon erstelltes Objekt darf nicht mehr geändert werden

Implementation

- alle Attribute sollten `final` sein
- Objektreferenz darf nicht ausgestellt werden
- jede Änderung führt zu einem neuen (unveränderlichen) Objekt
- die Klasse sollte mit `final` markiert werden

Vorteile und Beispiele

- Thread safe
- Steigert die Leistung und vereinfacht die Entwicklung
- Verbessert die Wiederverwendbarkeit
- in Java
  - java.lang.String
  - java.math.BigInteger
  - java.math.BigDecimal
  - java.io.File

java.lang.String

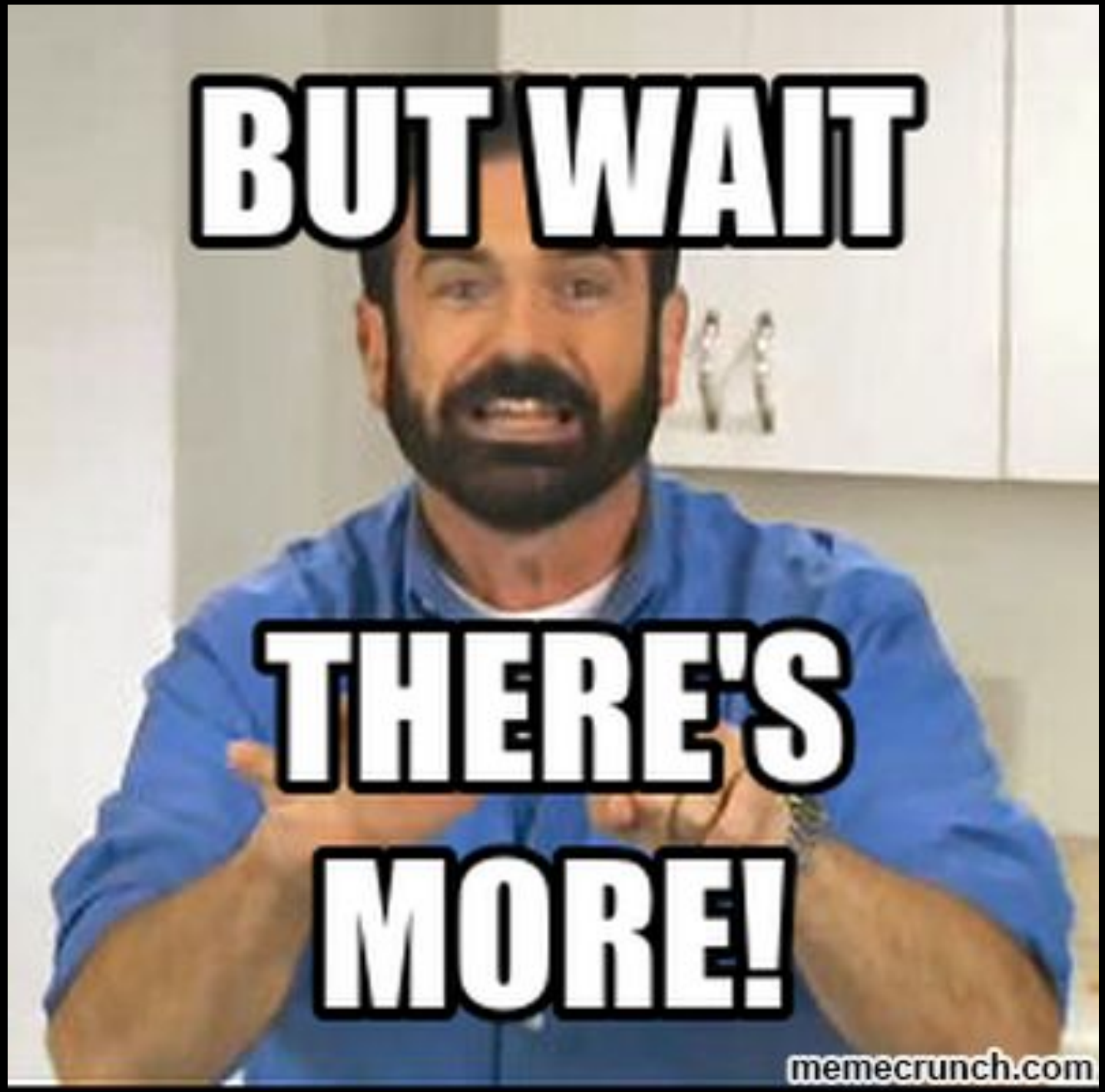
```
String x = "abc"; //1. use a literal
String y = new String("abc"); //2. use constructor

String a = "abcd";
String b = "abcd";
System.out.println(a == b); // True
System.out.println(a.equals(b)); // True

String c = new String("abcd");
String d = new String("abcd");
String e = a;
System.out.println(c == d); //False
System.out.println(c.equals(d)); // True

System.out.println(e == a); //True
System.out.println(e == b); //True
```

java.lang.String

```
class S{
   public static void main(String [] args){
    String a = new String("abc");

    if (a == "abc") {
       System.out.println("egal");
    }

    else {
        System.out.println("nu e");
      }
    }
}
```

java.lang.String

```
class S{
  public static void main(String [] args){
    String s = "Java";

    s.concat(" SE 6");
    s.replace('6','7');

    System.out.print(s);
   }
}
```


BUT WAIT
THERE'S
MORE!
memecrunch.com

java.lang.String

```
String s1 = "string"
String s2 = "string"
String s3 = new String("newstring")

String result="";

for(String s: arr){
     result = result + s;
```

StringBuffer, StringBuilder

- **StringBuffer, StringBuilder** sind mutable
- Alle Methoden sind synchronisiert, daher ist es thread-safe, aber auch sehr langsam
- **StringBuilder** ist eine Kopie von **StringBuffer**, jedoch ohne Synchronisation - man sollte es verwenden, wann immer dies möglich ist

String, StringBuffer, StringBuilder

- String ist immutable
- Objekt anlegen mit “” → String Pool
- Objekt anlegen mit new → Heap
- +-Operator ist für String überschrieben
  - mit StringBuffer implementiert
- String ist final

String, StringBuffer, StringBuilder


```
String str = "testing";
str = str + "abc";

String str = "testing";
StringBuffer tmp = new StringBuffer(str);
tmp.append("abc");
str = tmp.toString();
```

String, StringBuffer, StringBuilder


```
String string = "String is slow";
for (int i = 0; i < 100000; ++i)
   string = string + "slow";

return string;
// 3146ms
```

```
StringBuffer sb = new StringBuffer("String is slow");
for (int i = 0; i < 100000; ++i)
   sb.append("slow");

return sb.toString();
// 5ms
```

IntelliJ IDEA Keyboard Shortcut Cheat Sheet

JRebel | XRebel

Action	Windows	OS X
SEARCH		
Find usages	Ctrl + Alt + F7	⌘ + F7
Find usages (results)	Ctrl + Alt + Shift + F7	⌘ + Alt + F7
Find / Replace in file	Ctrl + F	⌘ + F / ⌘ + R
Find / Replace in projects	Ctrl + Shift + F	⌘ + Shift + F ⌘ + Shift + R
Find next	F3	F3
FILE NAVIGATION		
Open resource / Navigate to file	Ctrl + Shift + N	⌘ + Shift + O
Open type	Ctrl + N	⌘ + O
Go to symbol	Ctrl + Alt + Shift + N	⌘ + Alt + G
Go to line	Ctrl + G	⌘ + L
Recent files	Ctrl + E	⌘ + E
Tab / File switcher	Ctrl + Tab	⌘ + Shift + [/]
WINDOWS ACTIONS		
Maximize active window	Ctrl + Shift + F12	⌘ + Shift + F12
Next view (editor)	Alt + Left / Right	Ctrl + Left / Right
Quick switch editor	Ctrl + E	⌘ + E
Back	Ctrl + [	⌘ + [
Forward	Ctrl +]	⌘ +]
Show UML popup	Ctrl + Alt + U	⌘ + Alt + U
Activate editor	Ctrl + Tab	Ctrl + Tab
CODE COMPLETION		
Quick fix	Alt + Enter	Alt + Enter
Code completion	Ctrl + Space	Ctrl + Space
Smart code completion	Ctrl + Shift + Space	Ctrl + Shift + Space
Live templates	Ctrl + J	⌘ + J

Action	Windows	OS X
TEXT EDITING ACTIONS		
Move lines	Alt + Shift + Up/Down	Alt + Shift + Up/Down
Delete lines	Ctrl + Y	⌘ + Y
Copy / Duplicate lines	Ctrl + D	⌘ + D
Select identifier	Ctrl + W	Alt + Up
Format code	Ctrl + Alt + L	⌘ + Alt + L
Correct indentation	Ctrl + Alt + I	Ctrl + Alt + I
Structured selection	Ctrl + W	Alt + Up
CODE NAVIGATION		
Find usages / References in workspace	Alt + F7	Alt + F7
Find usages results	Ctrl + Alt + Shift + F7	⌘ + Alt + Shift + F7
Quick outline / File structure	Ctrl + F12	⌘ + F12
Inspect code hierachy	Ctrl + Alt + H	Ctrl + Alt + H
Open / Navigate to declaration	Ctrl + Alt + B	⌘ + Alt + B
Open / Navigate to type hierarchy	Ctrl + H	Ctrl + H
Open / Navigate to member hierarchy	Ctrl + Shift + H	⌘ + Shift + H
REFACTORING		
Refactor this	Ctrl + Alt + Shift + T	⌘ + Alt + Shift + T
Show quick refactoring menu		⌘ + Shift + T
Rename	Ctrl + Alt + R	Shift + F6
Surround with	Ctrl + Alt + T	⌘ + Alt + T
Extract local variable	Ctrl + Alt + V	⌘ + Alt + V
Extract / Assign to field	Ctrl + Alt + F	⌘ + Alt + F
Inline	Ctrl + Alt + N	⌘ + Alt + N
Extract method	Ctrl + Alt + M	⌘ + Alt + M
UNIVERSAL ACCESS		
Quick access / search everywhere	Ctrl + Shift + A	⌘ + Shift + A