Code Smell	Beschreibung
Alternative Classes With Different Interfaces	Zwei unterschiedliche Interfaces in zwei Klassen, die aber eigentlich gleich sind. Man muss die Gemeinsamkeiten finden um gleiches Interface zu nutzen. z.B. XMLBuilder und DOMBuilder funktionieren ähnlich habe aber unterschiedliche Interfaces, man macht daraus ein Interface. Unify Interface with Adapter, Rename Method, Move Method
Black Sheep	Von einem Black Sheep spricht man wenn eine Subklasse nicht zur Familie passt (d.h. andere Subklassen und Superklasse). Ist Inter-Class Smell. Move Method
Combinatorial Explosion	Wenn zusätzliche Daten oder zusätzliches Verhalten ein bereits aufgeblähtes Design noch mehr aufblähen spricht man von einer “Combinatorial Explosion”. Diesen Smell kann man auflösen wenn man das “Interpreter-Pattern” oder das “Protection Proxy Pattern” anwendet. Replace Implicit Language with Interpreter
Comment Smell	Wenn immer möglich soll der Code die Absicht des Kommentars ausdrücken —> Kein Kommentar. Gute Kommentare —> z.B. warum man einen bestimmten Algorithmus gewählt hat oder Hinweise zu Copyrights etc. Rename Method, Extract Method, Introduce Assertion
Conditional Complexity	Dieser Code Smell tritt auf wenn mehrere **verschachtelte** if-else-statements verwendet werden. Je älter der Code wird desto schwerer wird es ihn zu warten. Smell ist **auch wenn viele AND oder OR** in conditional expression. Im Vergleich zum “Switch-Statement-Smell” geht es hier wirklich um die Verschachtlung und **nicht darum ob es eine Liste von “if-else”** hat. **Typisches Refactoring:** “Decompose Conditional” oder “Introduce Explaining Variable” werden verwendet um die komplexen Teile einer Expression zu vereinfachen. Introduce Null Object, Move Embellishment to Decorator, Replace Conditional Logic with Strategy, Replace State-Altering Conditionals with State
Data Class	Eine Klasse die nur Data beinhaltet aber keine oder fast keine Methoden. —>DDH = Dumb Data Holder. DDH’s sind in Ordnung wenn man eine lose Kopplung will z.B. zwischen Business Logic und View oder zwischen verteilten Systemen zur Datenübertragung. Generell sollte man aber eher Klassen haben die Daten mit nützlichen Methoden paaren. **Typisches Refactoring:** “Move Method” wird verwendet um eine Klasse vom Schicksal als “Dumb Data Holder” zu retten in dem man Methoden in die Klasse rein-verschiebt. Move Method, Encapsulate Field, Encapsulate Collection
Data Clumps	Ein Haufen Daten die zusammen vorkommen und eigentl. besser ein eigenes Objekt wären. Test: ob die Variablen noch Sinn machen wenn man eine löschen würde. Wenn sie dann keinen Sinn machen —> besser eig. Objekt. Extract Class, Preserve Whole Object, Introduce Parameter Obj.
Dead Code	Code der nicht mehr verwendet wird aber immer noch in der Codebasis ist wird “toter Code” genannt. Man sollte ihn entsorgen weil er sonst “die Komplexität erhöht”, “man daran versehentlich Maintenance Änderungen vornimmt weil man ihn für echten Code hält” und ausserdem auch ein Präzedenzfall für weiteren Code schafft. Man keine Angst haben dass man Code verliert denn man noch braucht so lange man Source Control und Unit Tests verwendet.
Divergent Change	Kommt vor wenn eine Klasse geändert wird um noch zusätzliche/andere Verantwortlichkeiten zu erfüllen. Es wäre besser diese anderen Verantwortlichkeiten in eine separate Klasse zu extrahieren. Extract Class
Duplicate Code	Es gibt zwei Typen: “Blatant” und “Subtil”. Blatant ist es wenn der exakt gleiche Code an mehr als einem Ort vorkommt. Subtil ist es wenn der Code unterschiedlich ist aber das gleiche Verhalten hat bzw. das gleiche Problem löst. —> DRY = Don’t Repeat Yourself **Typisches Refactoring:** Mit “Extract Method” kann man den duplizierten Code in eine eigne Methode refactoren die dann an allen Stellen verwendet werden kann. Chain Constructors, Extract Composite, Extract Method, Extract Class, Form Template Method, Introduce Null Object, Pull Up Method, Pull Up Field
Feature Envy	Tritt auf wenn eine Methode sich mehr für eine andere Klasse interessiert als für seine eigene Klasse. Extract Method, Move Method, Move Field
Inappropriate Intimacy	Gutes Klassendesign schreibt vor dass eine Klasse nicht zu viel von seinen internen Details bekannt geben soll. Wenn zwei Klassen zu stark gekoppelt sind und auf ihre Gegenseitigen internen Details zugreifen dann spricht man von “Inappropriate Intimacy”. Dies führt zu sprödem, schwer-zu-änderndem Code. Move Method, Move Field, Extract Class, Hide Delegate, Replace Inheritance with Delegation, Change Bidirectional Association to Unidirectional Association
Incomplete Library Class	Kommt vor wenn es in unserem Code Verantwortlichkeiten gibt die klar in eine Library Klasse gehören würden aber wir nicht in der Lage sind bzw. die Library Klasse nicht verändern wollen. Introduce Foreign Method, Introduce Local Extension
Indecent Exposure	Von “Indecent Exposure” spricht mann wenn **Klassen public sind die private** sein sollten. Dieser Smell ist ähnlich wie “Inappropriate Intimacy”. z.B. Mock-Klassen die nicht einfach so verwendet werden dürfen. Encapsulate Classes with Factory

Large Class	Haben zu viele Methoden, zu viele Felder, zu viele Linen von Code und zu viele Verantwortungen. 10 Felder sind schon sehr viele. Wenn man Felder gruppieren möchte dann ist dass ein Zeichen das eine “Klasse” ausbrechen möchte. z.B. “server ip, login, pw, port” könnte eine eigene “ServerAccessData” Klasse sein. - **Typisches Refactoring:** Mit "Extract Class” kann man die Klasse in mehre Klassen aufbrechen. Alternativ kann man mit “Move Method” einige Methoden in andere Klassen verschieben, so dass unsere Klasse etwas leichter wird. Extract Class, Extract Subclass, Extract Interface, Replace Data Val with Obj, Replace Cond. Dispatcher with Command, Replace Implicit Language with Interpreter
Lazy Class (Freeloader)	Bietet nur wenig Funktionalität die auch von einer anderen Klasse übernommen werden könnte. Wir sollten die Lazy Class löschen. Collapse Hierarchy, Inline Class, Inline Singleton
Long Method	Die Vorteile von **kurzen** Methoden sind: “lesbarer Code”, “teilbarer Code”, “einfacher testbar”. Extract Method, Compose Method, Introduce Parameter Obj., Decompose Conditional, Preserve Whole Object
Long Parameter List	Tritt auf wenn eine Methode zu viele Parameter hat. Lange Parameter-Listen sind noch tolerierbar wenn sie automatisch default Werte annehmen (in Java jedoch nicht möglich). Replace Para. with Method, Introduce Parameter Obj., Preserve Whole Object
Oddball Solution	Wenn ein Problem einmal auf eine Art gelöst wird und dass selbe Problem an einer anderen Stelle auf einer andere Art gelöst wird dann sprechen wir von einer “Oddball Solution”. Wir sollten am besten über all im Code die selbe Art von Lösung verwenden. (Manchmal hat man auch duplicate code smell) **Typisches Refactoring:** “Substitute Algorithm” kann verwendet werden um inkonsistente Algorithmen durch konsistente zu ersetzen. Unify Interfaces with Adapter, Substitute Algorithm
Primitive Obsession	Man verwendet primitive Tools um etwas zu programmieren obwohl es “highlevel/entwickelte” Tools gäbe die besser geeignet währen. z.B. if (someString.indexOf("substring") != -1) anstatt if (someString.contains(“substring")) zu verwenden ist Primitive Obsession. Replace Data Val. with Obj., Introduce Param. Object, Extract Class, Encapsulate Composite with Builder, Move Embellishment to Decorator
Refused Bequest	Subklassen erben normalerweise Methoden und Data von ihren Eltern-Klassen. Wenn Sie dies jedoch nicht wollen und brauchen sprint man von einem “Refused Bequest”. Dies passiert wenn eine Subklasse das Interface der Superklasse nicht unterstützt. Push Down Field, Push Down Method, Replace Inheritance with Delegation
Side Effect	Methoden sollen nur ihre Haupt-Absicht durchführen und keine Seiteneffekte haben. Seiteneffekte sind wie Luftverschmutzung.
Solution Sprawl	Wenn Code oder Daten über viele Klassen hinweg verteilt sind. Es wäre besser alle zusammengehörigen Sachen einer Klasse zusammenzuziehen. Eine Lösungsmöglichkeit ist es eine “NodeFactory” einzuführen, die die Produktion übernimmt. Move Creation Knowledge to Factory
Speculative Generality	Man erhält diesen Smell wenn man Code schreibt der “noch” nicht gebraucht wird. Dieser unnötige Code verkompliziert das System nur unnötig und bildet mögliche, zusätzliche Fehlerquellen. Man sollte ihn besser erst in der Zukunft schreiben wenn er tatsächlich benötigt wird. **Typisches Refactoring:** “Collapse Hierarchy” und “Inline Class” können verwendet werden um Klassen zu entfernen die nicht nötig sind (ähnlich Lazy Class). Hier eifach weil der Code für “die Zukunft” gebaut wurde. Collapse Hierarchy, Rename Method, Remove Parameter, Inline Class
Switch Statement	Anstelle eines Switch-Statements könnte man auch Polymorphie verwenden. Muss aber nicht immer so sein. z.B. die Filme-Typen bei einem Film-Verleih kann man in polymorphe Klassen auslagern anstatt ein Switch-Statement zu verwenden. Machmal würde es jedoch nichts bringe das Switch-Statement zu ersetzen. z.B. wenn die Subklassen die entstehen würden sehr wenig Code enthielten und/oder keine Duplikaten eliminieren würde. **Erkennt man auch an instanceOf() in switches**. Move Accumulation to Visitor, Replace Conditional with Polymorphism, Replace Type Code with Subclasses, Replace Type Code with State/Strategy
Temporary Field	Ein Feld das nur kurzfristig verwendet wird, dessen Lebenszeit viel kürzer ist als die der Klasse oder Applikation. Man würde besser eine lokale Variable verwenden. Extract Class, Introduce Null Object
Message Chains	Kommt vor wenn man eine Lange Sequence von Methoden Aufrufen oder temporären Variablen hat um Daten zu erhalten. Die Kette macht den Code abhängig von den Verhältnissen von unabhängigen Objekten. —> Bad, Hide Delegate, Extract Method, Replace Delegate with Inheritance
Middle Man	Delegation ist gut, aber man soll es nicht übertreiben. D.h. man soll keine Objekte einführen die selbst nicht bringen ausser Nachrichten zu einem anderen Objekt zu bringen. Remove Middle Man, Inline Methode, Replace Delegate with Inheritance
Parallel Inheritance Hierarchies	Ein Spezialfall von Shotgun Surgery. Jedes Mal wenn man eine Subklasse einer Klasse macht, muss man ebenfalls eine Subklasse einer anderen Klasse machen. Move Method, Move Field
Shotgun Surgery	Man muss viele Codeteile ändern, an unterschiedlichen Orten nur um neues Verhalten hinzuzufügen oder Verhalten zu erweitern. Move Method, Move Field, Inline Class