Add Parameter	Eine Methode braucht mehr Informationen von ihrem Aufrufer.	Remove Parameter
Change Bidirectional Association to Unidirectional	Man hat eine zwei-seitige Beziehung zwischen Klassen. Aber eine Klasse braucht die Funktionen/Eigenschaften der anderen Klasse nicht mehr. Man ändert die zwei-seitige Beziehung in einen ein-seitige Beziehung, so dass die eine Klasse nichts mehr von der anderen Klasse weiss.	Change Unidirectional Assocication to Bidirectional
Change Reference to Value	Man hat ein Referenz-Objekt das klein, immutable und awkward zum verwalten ist. Man kann es in ein Werte-Objekt umwandeln (value object).	Change Value to Reference
Change Unidirectional Association to Bidirectional	Man hat zwei Klassen die sich gegenseitig brauchen aber es gibt nur eine Beziehung in eine Richtung. Man fügt back pointers ein und ändert die modifiers so dass beide sets updated werden.	Change Bidirectional Association to Unidirectional
Change Value to Reference	Man hat eine Klasse mit vielen, gleichen Instanzen die man mit einem einzigen Objekt ersetzen will. Man ändert das Objekt in ein Referenz-Objekt.	Change Reference to Value
Collapse Hierarchy	Eine Superklasse und eine Subklasse unterscheiden sich nicht sehr stark. Man kann daraus eine einzige Klasse machen.	Extract Superclass, Extract Subclass
Consolidate Conditional Expression	Man hat eine Sequenz von Conditional Tests mit dem selben Ergebnis. Man kann diese in eine einzelne Conditional Expression kombinieren und diese dann extrahieren. E.g. if xy, if yz, if za, —> if (isXYZA())	
Consolidate Duplicate Conditional Fragments	Das gleiche Code-Fragment wird in allen Ästen einer Conditional Expression verwendet. Man kann es aus der Expression heraus bewegen. if(){…; send();} else if(){…;send();} // 2x send() —> if(){..;} else if(){..;}send(); // 1x send() am Schluss	
Decompose Conditional	Man hat ein kompliziertes (if-then-else) Statement. Man kann die Methoden von der Condition extrahieren und einzelne Teile machen. if(date.before(SUMMER_START) ll date.after(S_E) —> if(notSummer(date)){}	
Duplicate Observed Data	Domain-Daten befinden sich in einer GUI-Control aber eine neue Domain-Methode braucht Zugang zu diesen Daten. Man kopiert die Daten ins Domain Objekt. Das installiert man einen observer der die Daten im DO und im GUI synchronisiert.	
Encapsulate Collection	Eine Methode gibt eine Collection zurück. Anstelle der Collection gibt man eine read-only View zurück und stellt add/remove Methoden zur Verfügung. So kann man z.B. nicht die ganze Collection null setzen.	
Encapsulate Downcast	Eine Methode gibt ein Objekt zurück dass vom Caller ge-downcasted werden muss. Man beweget den downcast in die Methode selbst. return readings.lastElement(); —> return (Reading) readings.lastElement();	
Encapsulate Field	Man hat ein public field. Man macht es private und stellt getter/setter zur Verfügung.	
Extract Class	Man hat eine Klasse die Arbeitet verrichtet welche eigentlich von zwei Klassen erledigt werden sollte. (Zuviel Funktionalität in einer Klasse.) Man erstellt eine neue Klasse und beweget die entsprechenden Felder und Methoden der alten Klasse in die neue Klasse. Doh! :P	Inline Class
Extract Interface	Mehrere Clients nutzen dasselbe Subset von den Interfaces einer Klasse. Oder: Zwei Klassen haben teilweise das gleiche Interface. Man kann das Subset in ein neues Interface extrahieren.	Inline Class
Extract Method	Man hat ein Code-Fragement das gruppiert werden könnte. D.h. man wandelt das Fragement in eine eigene Methode um. void blah(){doBlah; doBläh; //print stuff \n sysout(ddd); sysout(ddd3);} —> void printStuff(){sysout(ddd); sysout(ddd3);}	Inline Method
Extract Subclass	Eine Klasse hat Funktionen/Eigenschaften die nur in bestimmten Instanzen gebraucht werden. Man erstellt eine Subclass für dieses Subset von Funktionen/Eigenschaften.	Collapse Hierarchy
Extract Superclass	Zwei Klassen haben ähnliche Funktionen/Eigenschaften. Man erstellt eine Superklasse und bewegt die gemeinsamen Eigenschaften in diese neue Klasse. Die alten Klassen erben von der neuen Klasse. Vorteil: Highly Cohesive	Collapse Hierarchy
Extract Variable (auch: Introduce Explaining Variable)	Man hat eine komplizierte Expression. Man kann Teile der Expression oder das Resultat in eine temporäre Variable tun. Diese sollte einen Namen haben der den Sinn der Variable bzw. der Daten erklärt. z.B. man braucht in einer Methode “platform.toUpperCase().indexOf(“MAC”) > -1)” zum rausfinden ob das System ein mac ist. Anstatt dass man dies in der Methode selbst abfragt erstellt man eine neue Variable gem. —> final boolean isMacOs = platform.toBlahBlah	Inline Temp
Form Template Method	Man hat zwei Methoden in Subklassen die ähnliche Schritte in der gleichen Reihenfolge ausführen. Die Schritte selbst sind unterschiedlich. Man ändert die Schritte zu Methoden mit der gleichen Signatur so dass die originalen Methoden gleich sind. Dann kann man diese hochziehen (pull up). -	
Hide Delegate	Ein Client ruft eine delegierte Klasse eines Objekts auf. Man erstellt eine Methode auf dem Server die, die delegierte Klasse versteckt.	Remove Middle Man
Hide Method	Eine Methode wird von keiner anderen Klasse verwendet. Man kann die Methode privat machen. (Nachteil.. microtesting.)	
Inline Class	Eine Klasse tut nicht sehr viel nützliches. Man beweget alle Funktionalität von dieser Klasse zur einer anderen Klasse und löscht die alte Klasse.	Extract Class, Extract Interface
Inline Method	Der Körper einer Methode ist genau so klar wie ihr Name. Man kann den Körper der Klasse in den Aufrufer verschieben und die Methode selbst löschen.	Extract Method
Inline Temp	Man hat ein temporäre Variable die eine einfache Expression hat. Wir ersetzen alle Referenzen zur temporären Variable mit dem Ausdruck selbst und löschen dann die Variable.	Extract Variable
Introduce Assertion	Eine Sektion von Code nimmt etwas an über den Status des Programms. Man sollte diese assumption explizit machen mit einer assertion um sicher zu gehen dass das Program tatsächlich in diesem Status ist. Assert.isTrue(blah);	
Introduce Foreign Method	Eine Server-Klasse braucht eine zusätzliche Methode aber wir dürfen die Server-Klasse nicht verändern. Wir erstellen eine Methode in der Client-Klasse mit einer Instanz der Server-Klasse als erstes Argument.	
Introduce Local Extension	Eine Server-Klasse braucht eine zusätzliche Methode aber wir dürfen die Server-Klasse nicht verändern. Wir erstellen eine neue Klasse die die zusätzlichen Methoden enthält. Wir machen die Extension-Klasse zu einer Subklasse oder zu einem Wrapper des Originals.	
Introduce Null Object	Wir haben wiederholt Checks für ein null value. Wir ersetzen das null value durch ein null object. e.g. NullCustomer Klasse hat keine oder default Daten aber trotzdem Methoden	
Introduce Parameter Object	Wir haben eine Gruppe von Parameter die natürlich zusammen passen. Wir ersetzen diese durch ein Objekt.	
Move Field	Ein Feld wird von einer anderen Klasse verwendet als von der Klass in der es definiert wird. Wir erstellen ein neues Feld in der Ziel-Klasse und ändern alle Verwendungen des alten Felds.	
Move Methode	Eine Methode wird von einer anderen Klasse mehr verwendet als von der Klasse in der sie definiert wird. Wir erstellen eine neue Methode in der Zielklasse und ändern die Verwendungen der alten Methode zur neuen Methode. Die alte Methode wird entweder gelöscht oder sie delegiert zur neuen Methode.	

Parametrize Method	Mehrere Methoden tun ähnliche Dinge aber mit unterschiedlichen Werten. Man erstellt eine Methode mit vielen Parametern die alle Arbeit erledigt. Die anderen “spezifischen” Methoden mit wenigen Parametern delegieren auf die Arbeits-Methode.	Replace Parameter with Explicit Method
Preserve Whole Object	Man erhält mehrere Werte von einem Objekt und gibt diese Werte in einem Methoden-Aufruf weiter. Anstelle der Werte geben wir das ganze Objekt weiter, die Methode kann sich die Werte die sie braucht selbst aus dem Objekt nehmen.	
Pull Up Constructor Body	Man hat Konstruktoren in Subklassen die mehrheitlich identische Körper haben. Man erstellt einen Superklassen-Konstruktor, dieser wird von der Subklasse her aufgerufen.	
Pull Up Field	Zwei Subklassen beinhalten das gleiche Feld. Man bewegt das Feld zur Superklasse.	Push Down Field
Pull Up Method	Man hat zwei Methoden in Subklassen die identische Resultate liefern. Man verschiebt die Methoden in die Superklasse.	Push Down Method
Push Down Field	Ein Feld wird nicht von allen Subklassen verwendet. Man verschiebt das Feld in die Subklassen die es verwenden.	Pull Up Field
Push Down Method	Das Verhalten einer Superklasse ist nur relevant für einzelne Subklassen. Man verschiebt die Methode in die Subklassen für die das Verhalten interessant ist.	Pull Up Method
Remove Assignment to Parameters	Das Resultat einer Operation wird in eine Parameter-Variable gespeichert. Man soll besser eine separate, temporäre Variable verwenden.	-
Remove Control Flag	Man hat eine Variable die als Kontroll-Flag für eine Serie von boolean-expressions agiert. Man soll besser break oder return verwenden.	-
Remove the Middle Man	Eine Klasse delegiert zu viel an andere Klassen. Wir entfernen den Middle-Man und bringen die Client-Klasse dazu direkt dene delegate aufzurufen. **(Erhöht Kopplung!)**	Hide Delegate
Remove Setting Method	Ein Feld wird nur beim Erstellen der Klasse gefüllt und nachher nie mehr. Wir entfernen die Setter-Methode.	
Replace Array with Object	Wenn man ein Array hat in dem einzelne Elemente unterschiedliche Dinge bedeuten. z.B. Ein Person-Array mit Feld1 = Namen, Feld2 = Vornamen, Feld3 = Adresse etc. dann sollte man dies durch ein Objekt mit Feldern ersetzen.	
Replace Conditional with Polymorphism	Wenn man ein Conditional hat dass über das Verhalten einer Methode entscheidet sollte man besser Subklassen nehmen und die Methode jeweils überschreiben anstatt in einer Methode unterschiedliches Verhalten zu haben. z.B. if(){do blah} else if(){do blöh} —> besser: Klasse Bird, Subklasse Europäisch und Afrikanisch, darin Methode überschreiben.	
Replace Constructor with Factory Method	Wenn man mehr als eine simple Konstruktion eines Objekts machen will sollte man den Konstruktor durch eine Factory Methode ersetzen.	
Replace Data Value with Obj.	Man hat ein Daten Item das zusätzliche Daten oder Verhalten übernehmen soll. Dann macht man aus dem Daten Item ein Objekt.	
Replace Delegation with Inheritance	Wenn man oft simple Delegationen für das ganze Interface verwendet könnte man auch die delegierende Klasse zu einer Subklasse des Delegators machen.	Replace Inheritance with Delegation
Replace Error Code with Exception	Eine Methode gibt einen speziellen Code zurück um einen Error anzuzeigen. Man sollte besser eine Exception werfen.	
Replace Exception with Test	Es wird eine Exception geworfen für etwas dass der Aufrufer problemlos überprüfen könnte. z.B. IndexOutOfBounds. Hier ist es besser wenn der Aufrufer dies prüft anstatt dass wir die Exception werfen.	
Replace Inheritance with Delegation	Eine Subklasse verwendet nur einen Teil des Superklassen Interface oder will nicht alle Daten der Superklasse erben. Man führt eine Delegation anstelle der Vererbung ein.	
Replace Magic Number with Symbolic Constant	Wenn man spezielle Zahlen, z.B. pi, Gravitation etc. hat sollte man führ diese eine Konstante definieren anstatt die Zahlen hardcoded zu haben.	
Replace Method with Method Object	Man hat eine lange Methode die lokale Variablen verwendet und man kann das Refactoring “Extract Method” nicht anwenden. Wird verwenden die Methode in ihr eigenes Objekt so dass die lokalen Variablen die Felder des Objekts werden. Die lange Methode wird in kleiner Methoden des neuen Objekts aufgeteilt.	
Replace Nested Conditional with Guard Clauses	Eine Methode hat ein Konditionales Verhalten dass den normalen Pfad der Ausführung unklar macht. Wir verwenden neu eine Guard Klausel für die Spezialfälle, dadurch wird der normale Pfad wieder sichtbar. D.h. if(){} else if(){if(){if(){if(){}} —> if(){return x}; if(){return x}; if(){…}; (Wirklich **if** und nicht if else!)	
Replace Parameter with Explicit Methods	Man hat eine Methode die je nach dem Wert einer enumerated Parameters unterschiedlichen Code ausführt. Wir erstellen eine separate Methode für jeden Wert des Parameters. setValue(String name, int value) —> setHeight(int value), setWidth(int value)	
Replace Parameter with Method	Ein Objekt aktiviert eine Methode und übergibt dann deren Resultat als Parameter einer anderen Methode. Der Empfänger des Resultats kann ebenfalls diese Methode aufrufen. Wir entfernen den Parameter und lassen den Empfänger die Methode von Anfang an selbst aufrufen. Effekt—> weniger Code	
Replace Record with Data Class	Man braucht ein Interface mit einer Record (Eintrag) Struktur in einer traditionellen Programmierumgebung. Wir machen ein “dumb data object” für den Record (Eintrag). anm. wie DB wahrscheinlich.	
Replace Subclass with Fields	Wir haben Subklassen die sich nur in Methoden unterscheiden die konstante Werte zurückliefen. Wir geben die Felder der Superklasse und eliminieren die Subklassen. z.B. Superklasse Person, Subklassen Male, Female mit Methoden “getCode()” die M und F zurückgeben —> nur die Superklasse Person überlebt.	
Replace Temp with Query	Wir verwenden eine temporäre Variable um das Resultat einer Expression zu speichern. Wir extrahieren die Expression in eine Methode und ersetzen alle Referenzen zur Variable durch die Expression. Die neue Methode kann dann auch in anderen Methoden genutzt werden. Lesbarkeit erhöht sich.	
Replace Type Code with Class	Eine Klasse hat einen numerischen Typ-Code der das Verhalten nicht verändert. Wir ersetzen die Nummer mit einer neuen Klasse. Vorher: Person Nacher: Person, Blutgruppe	
Replace Type Code with State / Strategy	Wir haben einen Type Code der das Verhalten der Klasse beeinflusst aber wir dürfen kein Subclassing verwenden. Wir ersetzen den Type Code durch ein State (Status) Objekt.	
Replace Type Code with Subclasses	Wir haben einen immutable Type Code der das Verhalten unserer Klasse beeinflusst. Wir ersetzen den Type Code durch Subklassen.	
Self Encapsulate Field	Auf ein Feld wird direkt zugegriffen, die Kupplung zu diesem Feld wird awkward. Wir erstellen getters/setters für das Feld und greifen nur noch mit diesen zu.	
Separate Query from Modifier	Wir haben eine Methode die nicht nur einen Wert zurückgibt sondern auch den Status eines Objekts ändert. Wir machen aus der einen Methode zwei separate Methoden. Eine für die Abfrage (return) und eine für die Modifikation des Status.	
Split Temporary Variable	Wir haben eine temporäre Variable die mehr als einmal zugewiesen wird. Sie ist aber keine loop-Variable und auch keine Collection. Wir machen einen separate temporäre Variable für jede Zuweisung so dass eine klare Trennung vorhanden ist.	