**Replace Multiple Constructors with Creation Methods**

**Motivation**

In some languages, such as C++ and Java, each constructor must be named after the class name. This can lead to confusion in a class with multiple constructors as they will all be very similar; the only difference is the parameters each constructor takes. Therefore, it will be hard to determine exactly what each constructor does. This can be simplified using the Factory pattern. Creation methods can be created instead of constructors to differentiate each object with a suitable name. This makes the code easier to read and work with as programmers will spend less time on choosing the correct constructor to use.

**Methods**

1. Identify a class with many constructors.
2. Create one catch-all constructor that all other constructors are derived from.
3. For every object created by a constructor, create a Creation Method with an intention revealing name. Ensure it takes the same parameters as the constructor method before it.
4. Replace all calls to constructors with calls to the new Creation Methods.

**Sample code to refactor**

A bank offers different kinds of loans to its customers. The type of loan to create is handled by the Loan class.

public class Loan {

private static String TERM\_LOAN = “TL”;

private static String REVOLVER = “RC”;

private static String RCTL = “RCTL”;

private String type;

private CapitalStrategy strategy;

private float notional;

private float outstanding;

private int customerRating;

private Date maturity;

private Date expiry;

public Loan(float notional, float outstanding, int customerRating, Date expiry) {

. . .

}

public Loan(float notional, float outstanding, int customerRating, Date expiry,

Date maturity) {

. . .

}

public Loan(CapitalStrategy strategy, float notional, float outstanding,

int customerRating, Date expiry, Date maturity) {

. . .

}

public Loan(String type, CapitalStrategy strategy, float notional,

float outstanding, int customerRating, Date expiry) {

. . .

}

public Loan(String type, CapitalStrategy strategy, float notional,

float outstanding, int customerRating, Date expiry, Date maturity) {

this.type = type;

this.strategy = strategy;

this.notional = notional;

this.outstanding = outstanding;

this.customerRating = customerRating;

this.expiry = expiry;

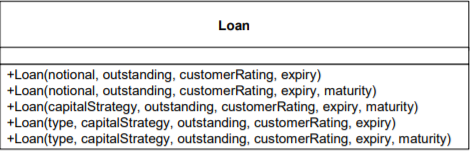
if (RCTL.equals(type))

this.maturity = maturity;

}

}

**UML Diagram**



**Risks**

* A constructor may be transformed into the wrong creation method
* It is difficult to identify which constructor should be transformed to which creation method without looking through the parameters.