**Replacing constructors with creation methods**

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;

protected 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;

}

static Loan newTermLoan(float notional, float outstanding, int customerRating,

Date expiry) {

. . .

}

static Loan newTermWithStrategy(CapitalStrategy strategy, float notional,

float outstanding, int customerRating, Date expiry) {

. . .

}

static Loan newRevolver(float notional, float outstanding, int customerRating,

Date expiry) {

. . .

}

static Loan newRevolverWithStrategy(CapitalStrategy strategy, float notional,

float outstanding, int customerRating, Date expiry) {

. . .

}

static Loan newRCTL(float notional, float outstanding, int customerRating,

Date expiry, Date maturity) {

. . .

}

static Loan newRCTLWithStrategy(CapitalStrategy strategy, float notional,

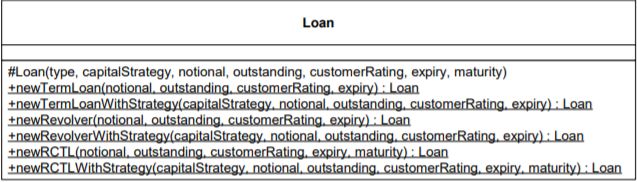
float outstanding, int customerRating, Date expiry, Date maturity) {

. . .

}

}

**UML Diagram**



Each constructor has now been transformed into the appropriate creation methods derived from the catch-all constructor. The new creation methods have been named to quickly reveal their intended uses, reducing the time programmers need to sort through the code.

**Testing**

A unit test can be created for each creation method to ensure that they return the correct object.