# REFACTORING

https://www.lynda.com/Developer-Programming-Foundations-tutorials/Welcome/122457/135613-4.html
https://www.youtube.com/watch?v=vqEg37e4Mkw
https://refactoring.com/catalog/
https://en.wikipedia.org/wiki/Code_smell

The process of changing a software system in such a way it does not alter the external behavior of the code, yet improves the internal structure

Refactoring code is a "...change made to the internal structure of software to make it easier to understand and cheaper to modify without changing its observable behavior… It is a disciplined way to clean up code that minimizes the chances of introducing bugs" *Martin Fowler*

Improvements to code that already exist and work
Changes to code / Cleanup of code
Well-structured code has many different benefits
Risk exist when refactoring

## WHAT IT IS AND IS NOT

### REFACTORING IS TO

- Fill in short-cuts
- Eliminate duplication and dead code
- Make the design and logic clear
- Simplify the code and to make it easier to understand
- Make it easier and safer to change in the future

### REFACTORING IS NOT

- Debugging
- Adding features
- Performance optimization

## WHY

Reality

- Extremely hard to get the design right the first time
- Hard to understand the business domain and requirements
- Hard to predict how the system will evolve over time
- The original design is often inadequate
- The system get harder to maintain over time

## EFFECT

- Easier to understand
- Clean code
- Better code

## WHEN

When

- After new features has been implemented

- Errors are corrected
- During code review

When not to refactor

- While solving a code problem.
- Sometimes the better approach is to ditch it all and start over

## CODE SMELLS

Many different code smells
Improve ability to detect code smells

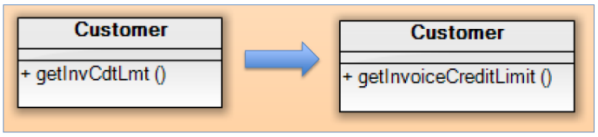
- Code duplication / Dead code
- Long methods / Large class / Many parameters
- Too many comments / Inconsistent naming

## CATALOG OF REFACTORINGS

Techniques for refactoring
100+ different individual refactoring techniques
Each deal with a single common issue
From very general purpose to very specific
Many opposing techniques for different situations

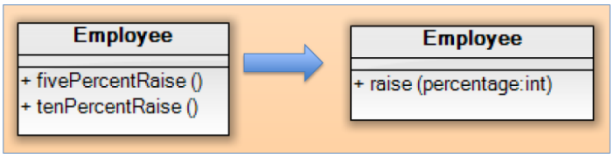
*Rename method*

- If the name of a method does not reveal its purpose
- Change the name of the method




*Parameterize method*

- If several methods do similar things but with different values contained in the method body
- Create one method that uses a parameter for the different values




*Preserve whole object*

- If you are getting several values from an object and passing these values as parameters in a method call
- Send the whole object instead

```
int low = daysTempRange().getLow();
int high = daysTempRange().getHigh();
withinPlan = plan.withinRange(low, high);
```

```
withinPlan = plan.withinRange(daysTempRange());
```

*Inline temp*

- If you have a temp that is assigned to once with a simple expression, and the temp is getting in the way of other refactoring
- Replace all references to that temp with the expression

```
double basePrice = someOrder.basePrice();
return (basePrice > 1000)
```

```
return (someOrder.basePrice() > 1000)
```

*Decompose conditional*

- If you have a complicated conditional (if-then-else) statement
- Extract methods from the condition, then part and else part

```
if (date.before(summer_start) || (summer_end))
  charge = quantity * winterRate + winterServiceCharge;
else
  charge = quantity * summerRate;
```

```
if (isSummer(date))
  charge = summerCharge(quantity);
else
  charge = winterCharge(quantity);
```

*Extract method*

- If you have a code fragment that can be grouped together
- Turn it into a method whose name explains the purpose of the method

```
void printOwing(double amount) {
   printBanner();
   //print details
   System.out.println (name: + _name);
   System.out.println (amount + amount);
```

```
void printOwing(double amount) {
   printBanner();
   printDetails(amount);
}

void printDetails (double amount) {
   System.out.println (name: + _name);
   System.out.println (amount + amount);
}
```