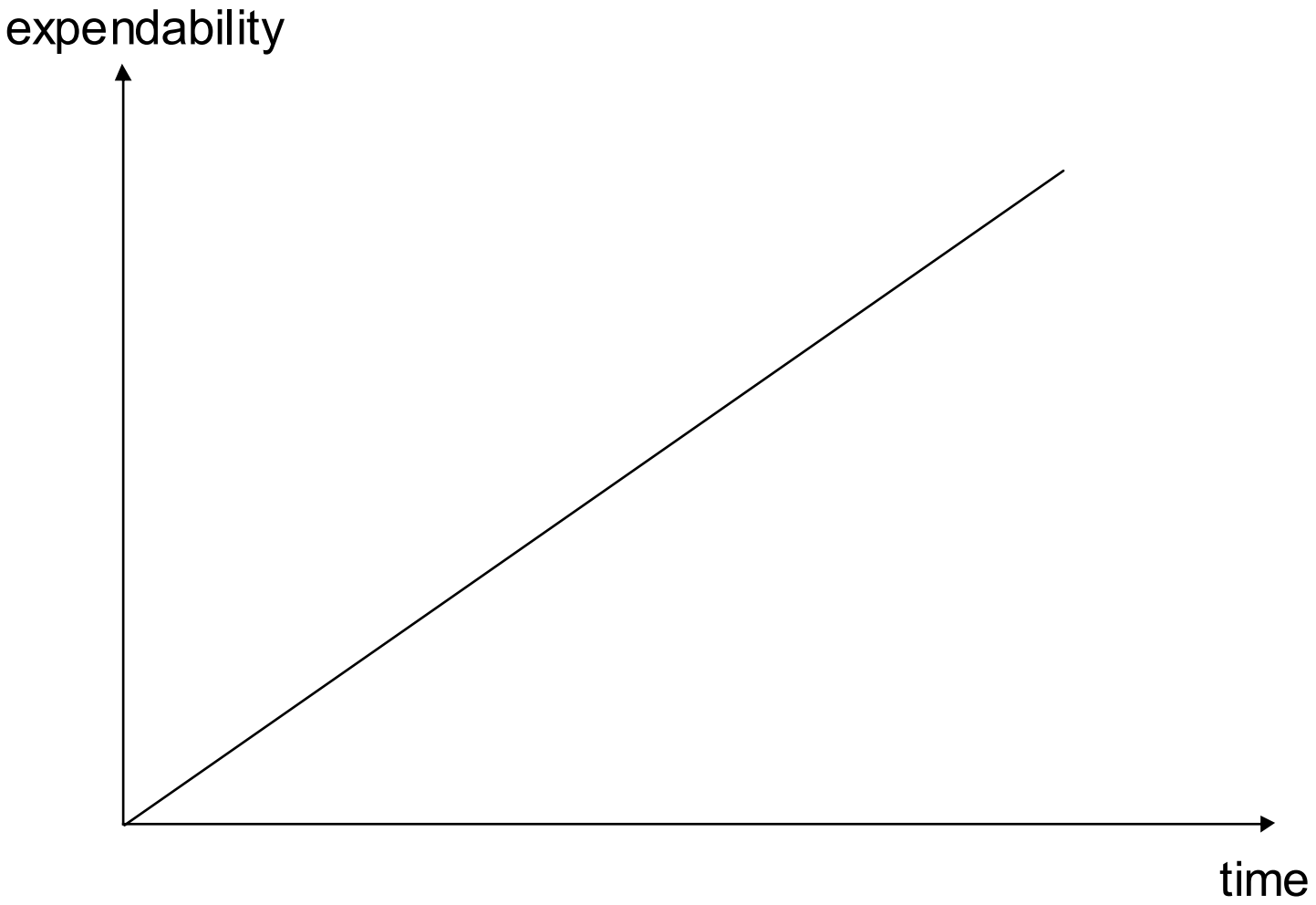
Refuctoring

Jason Gorman

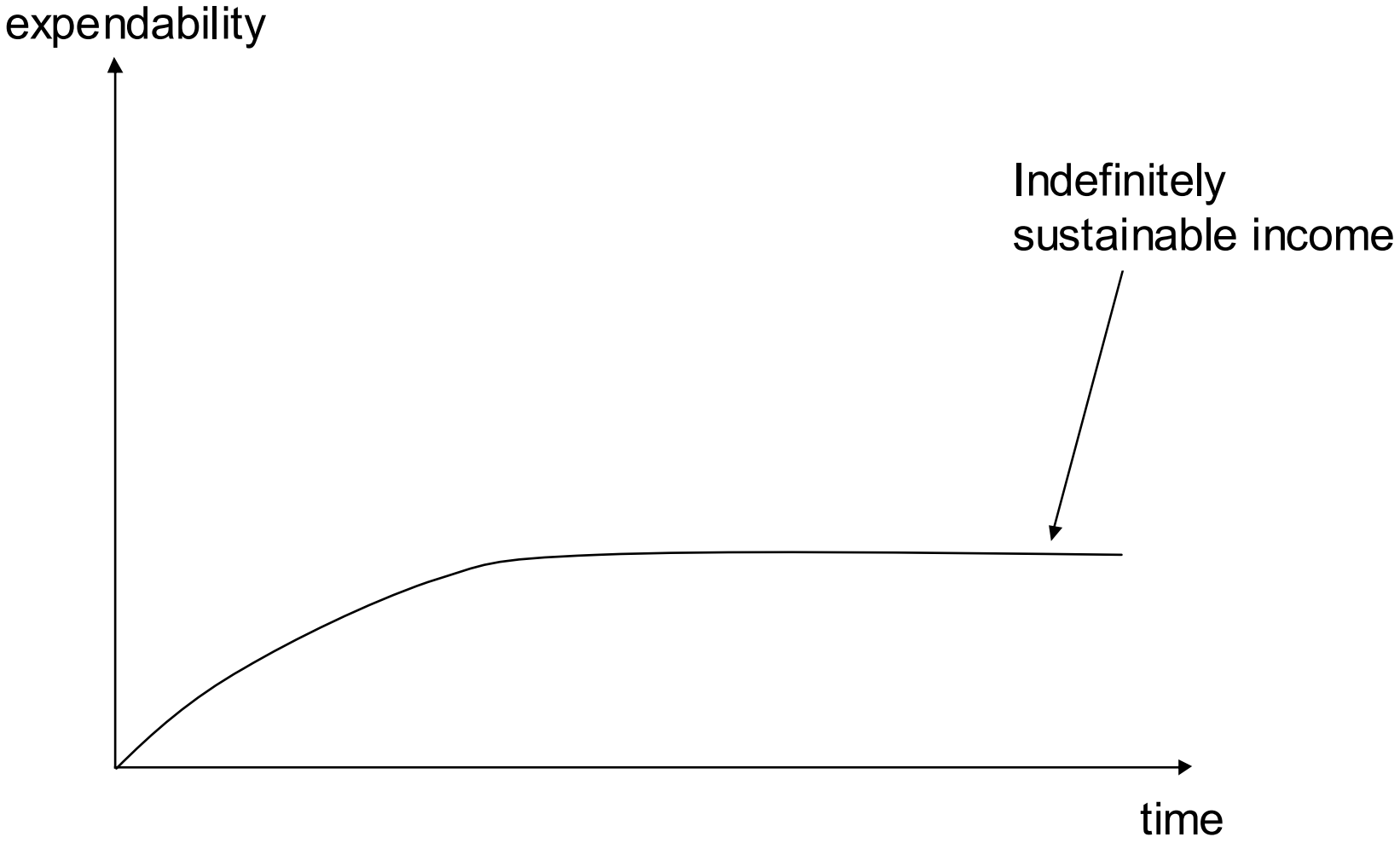
The Professional's Dilemma



What Is Refuctoring?

- Refuctoring is the process of taking a well-designed piece of code and, through a series of small, reversible changes, making it completely unmaintainable by anybody except yourself.
- Comprehensive regression testing guarantees that nobody will be any the wiser.

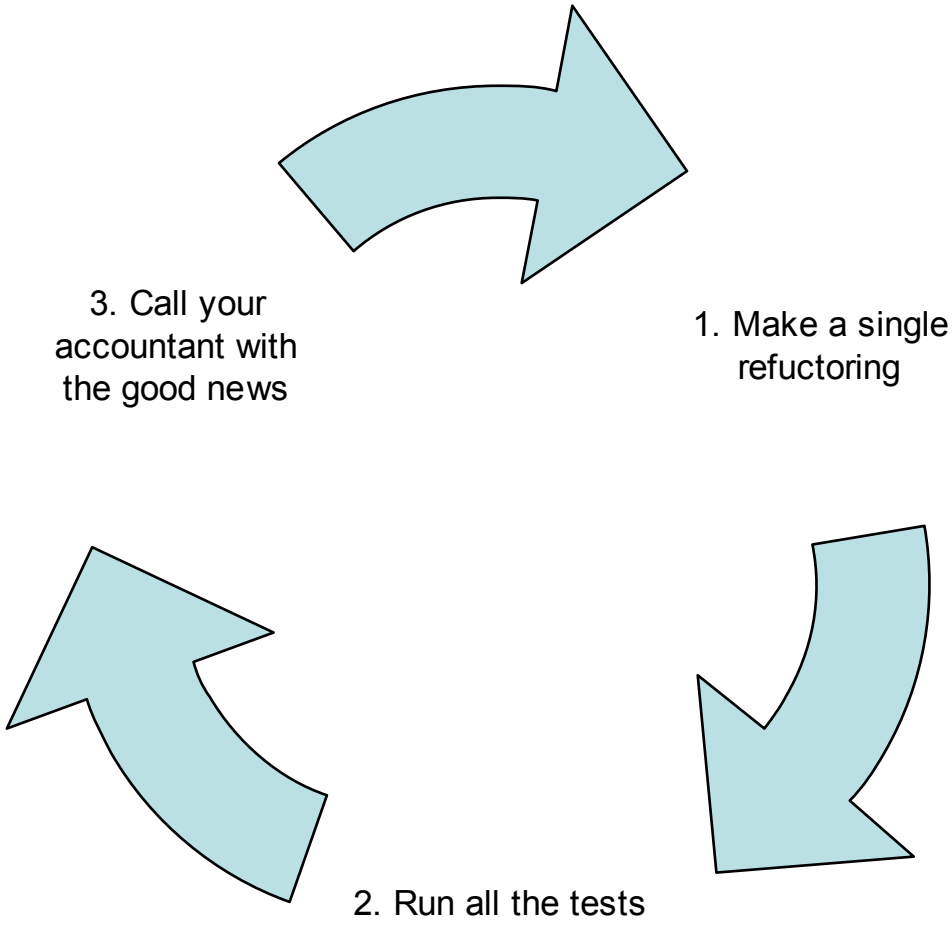
Dilemma Solved – With Refuctoring!



Code “Smiles”

- Common sense naming conventions
- Cohesive and loosely coupled modules
- Elegant abstractions
- Lack of duplication
- A close resemblance to the application domain

The Refuctoring Process



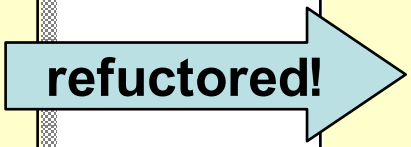
Common Refuctorings

Code Smile – Common Sense Naming Conventions

```
class Account {

        private float balance = 0;

        public void deposit(float amount) {
                balance += amount;
        }

        public void withdraw(float amount) {
                balance -= amount;
        }

        public float getBalance() {
                return balance;
        }
}
```

Refuctoring #1 – Pig Latin

```
class Account {

        private float balance = 0;

        public void deposit(float amount) {
                balance += amount;
        }

        public void withdraw(float amount) {
                balance -= amount;
        }

        public float getBalance() {
                return balance;
        }
}cv
```




```
class Accountway {

        private oatflay alancebay = 0;

        public void epositday(oatflay amountway) {
                alancebay += amountway;
        }

        public void ithdrawway(oatflay amountway) {
                alancebay -= amountway;
        }

        public oatfflay etBalancegay() {
                return alancebay;
        }
}
```


parlez|uml

Code Smile – Simple, Easy-to-follow Code

```
public void executeTransaction() {

        payer.w ithdraw(amount);
        payee.deposit(amount);

}
```

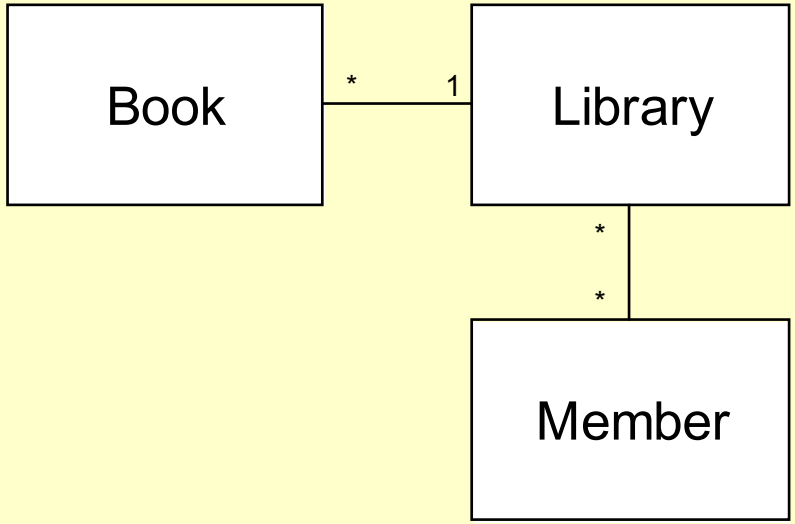
Refuctoring #2 – Treasure Hunt

```
public void executeTransaction() {

        payer.withdraw(amount);
        payee.deposit(amount);

}
```

refuctored!

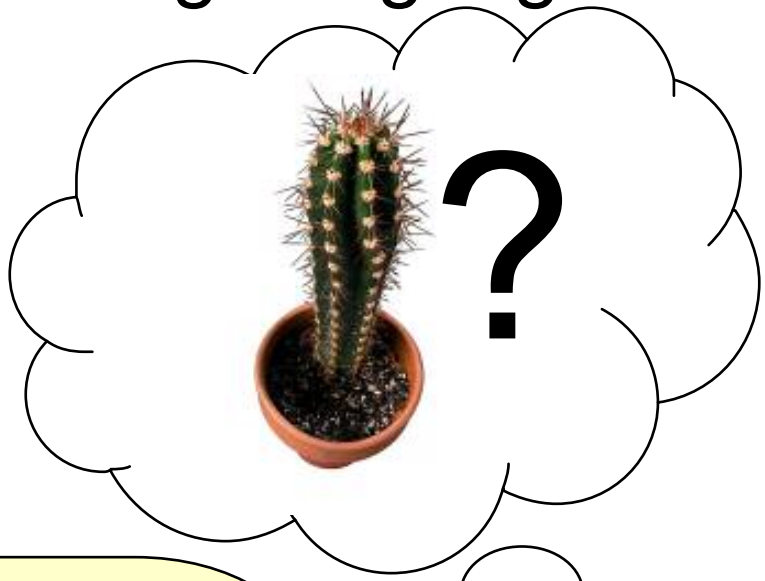
```
public void executeTransaction() {

        this.callExecuteTransaction();
}

private void callExecuteTransaction() {
        helper.doExecute();
}

……..

class TransactionExecutionHelper extends ExecutionHelper {

        public void doExecute() {
                base.doExecute(this);
        }

        public void execute()  {
                ExecuteTxCommand command =
                        CommandFactory.createCommand();
                command.execute();
                // and so on…
        }

}

……..

abstract class ExecutionHelper {

        protected void doExecute(ExecutionHelper helper) {
                helper.execute();
        }

        public void execute();
}
```

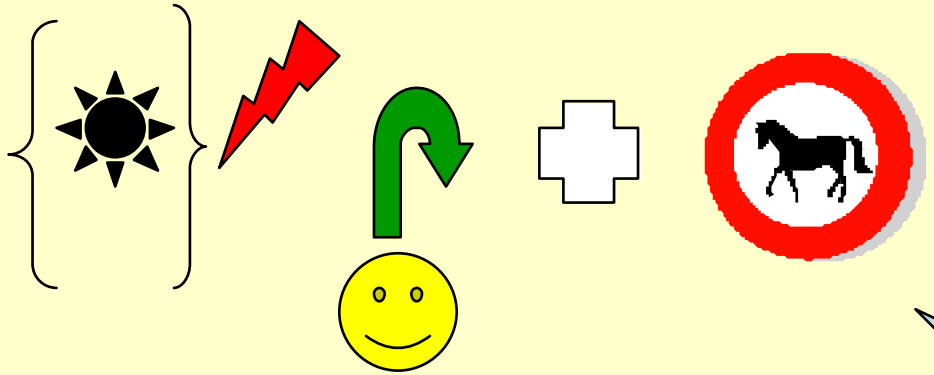
Code Smile – Shared Understanding



parlez|uml

Refuctoring #3 – Unique Modeling Language





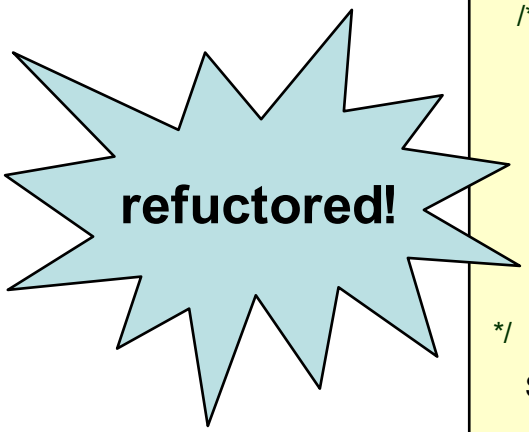
refuctored!

parlez|uml

Code Smile – Clutter-free Code

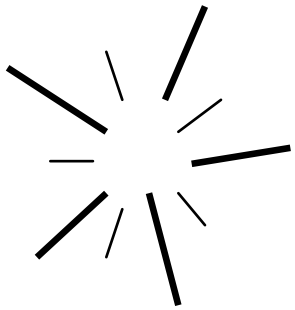
```
for (int number = 1; number <= 12; number++) {
    System.out.println(number + " squared is " + (number * number));
}
```

Refuctoring #4 – Stating The Bleeding Obvious




```
/*

                Author: Jason Gorman
                Date & Time: 13/9/05 12:32:06

                Revision History:

                13/9/05         12:33:14        Accidentally deleted a line then used "undo" to
                                                bring it back again

                Comment Body:

                Declare an integer called number with an initial value of 1, and
                then perform the same block of code 12 times, incrementing the
                value of number by 1 each time for the purposes of computation

*/
for (int number = 1; number <= 12; number++) {
  /*
                get the System's output stream to print the following text:

                number + " squared is " + (number * number)

                for the purposes of seeing what the square of number is
*/
   System.out.println(number + " squared is " + (number * number));

// use the { character to let the compiler know where the for loop ends
}
```

Code Smile – No Redundant Code

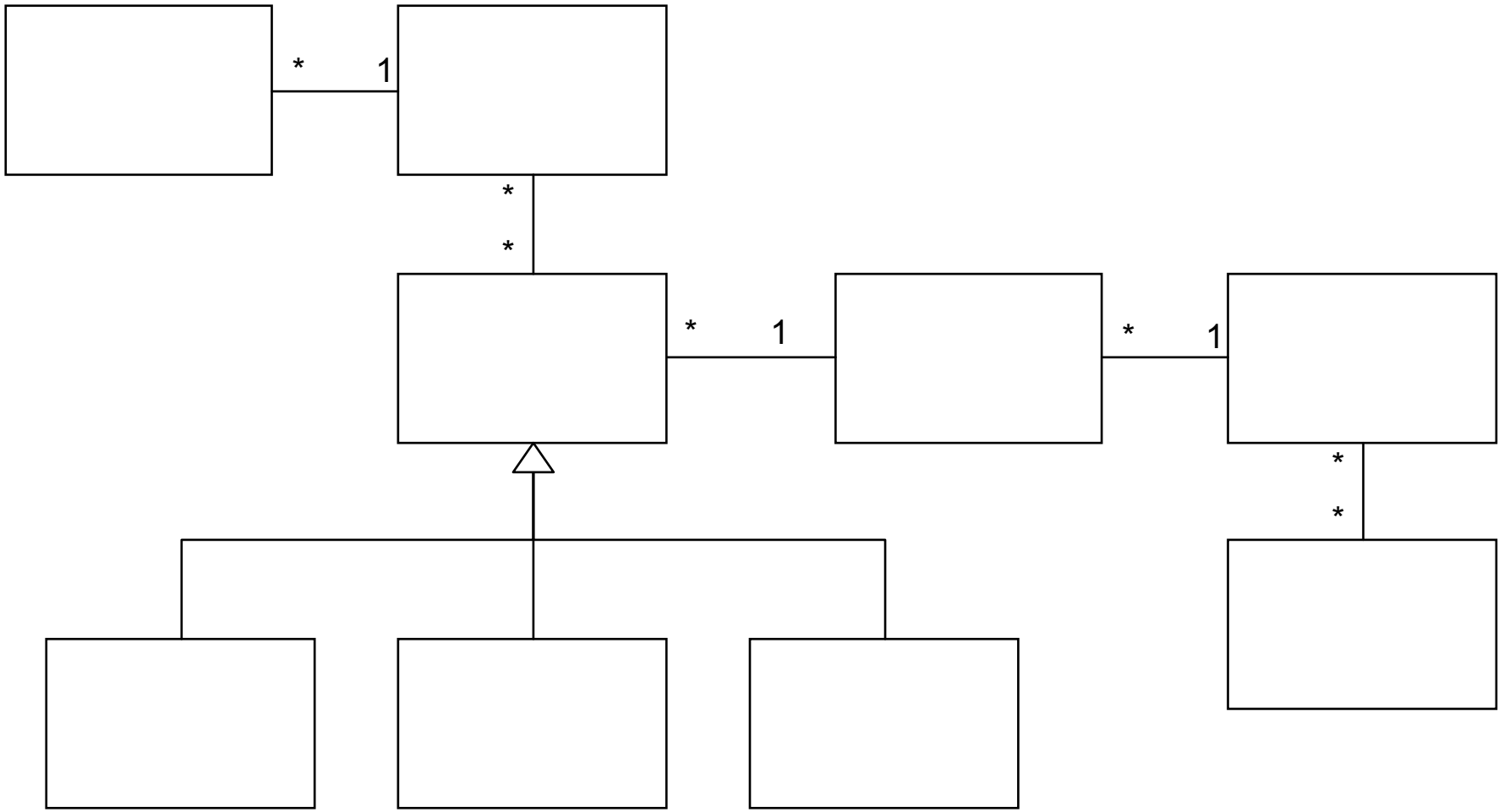


Refuctoring # 5 – Rainy Day Module

```
class SpareCode {

        private int spareInteger;
        private String luckyString;
        private bool youNeverKnow ;

        public void spareLogic() {

                spareInteger = 1;

                if( youNeverKnow ) {

                        spareInteger++;
                }
                System.out.println(luckyString);
        }
}
```

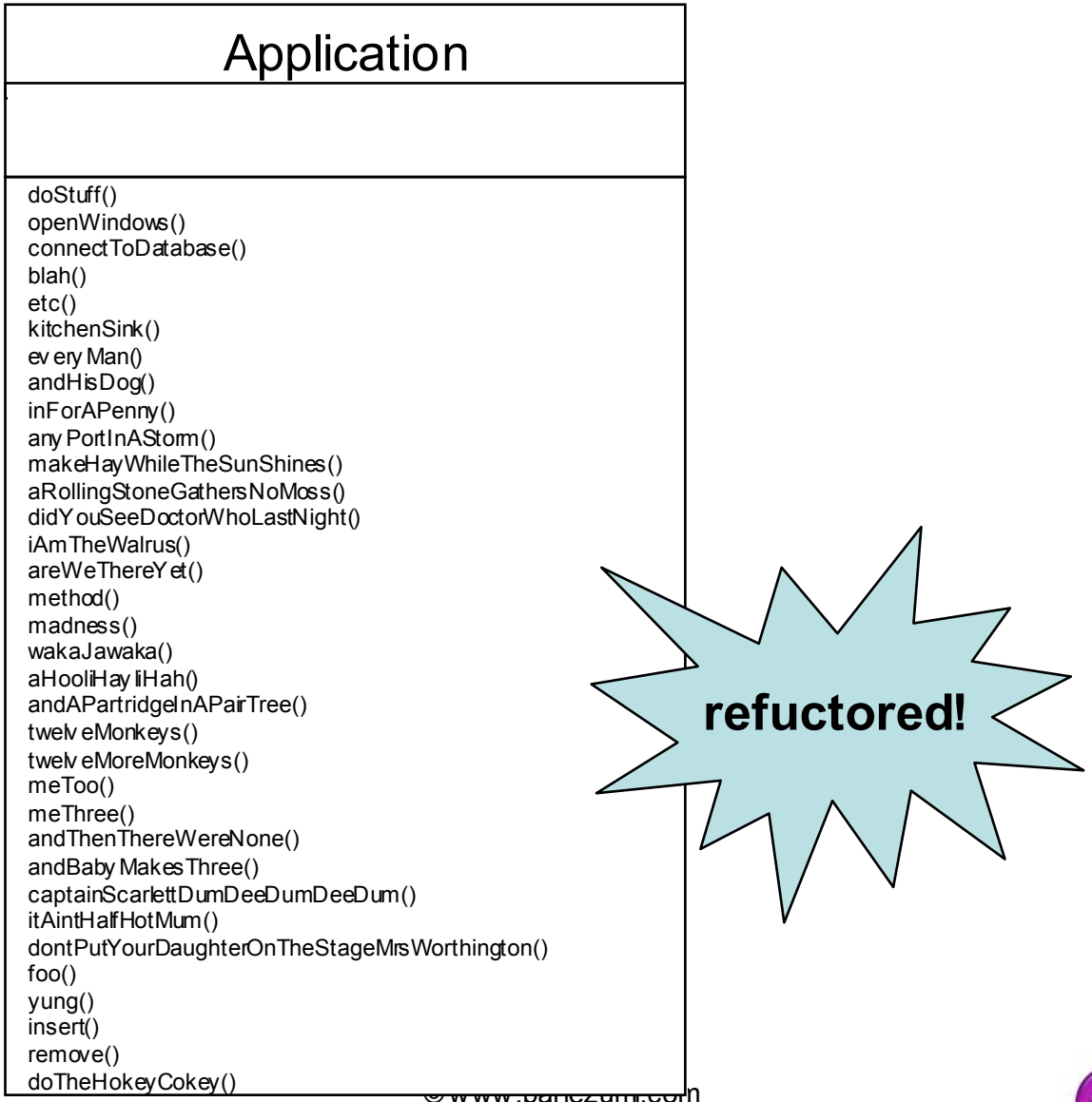
parlez|uml

Code Smile – Manageable Modules



parlez|uml

Refuctoring # 6 – Module Gravity Well



parlez|uml

Refuctoring Metrics

Job Security Index = 1 / Maintainability

Further Reading

- **Refuctoring To Anti-Patterns.** John Q. Clockwatcher
- **Mortgage-driven Development.** Keith Bank-Account
- **Timesheet-oriented Architecture.** Jill Tax-Haven

www.parlezuml.com