

# Code Smells

What is that smell? Did you write that code?



- » Skan.ai - chief Architect
- » Ai.robots - chief Architect
- » Genpact - solution Architect
- » Welldoc - chief Architect
- » Microsoft
- » Mercedes
- » Siemens
- » Honeywell



Mubarak



- » day 1
- » clean comments
- » Errors
- » Day 2
- » clean Functions
- » clean tests
- » Day 3
- » clean class
- » oo programming
- » Day 4
- » functional programming
- » Concurrency
- » Domain Driven Design

## Inheritance

```
Class CA extends CB
{
}
}
```

```
fun(Base base){
    Logic
}
```

## Agg /composition

```
Class CA
{
    CB ref;
}
```

```
Eager
Class CA
{
    CB ref;

    CA(CB cb){
        ref =cb;
    }
}
```

```
Lazy
Class CA
{
    CB ref;

    Set(CB cb){
        ref =cb;
    }
}
```

# Comments

- » Avoid unwanted comments
- » dont comment source code
- » dont use comments to manage history/  
author info
- » Use source control, specification pattern,  
Assert, readable fun/variable/class names,  
... to remove comments

# Errors

- » Avoid bool, int, null for errors
- » Use exception to manage exception flow
- » dont use exception for alternate flow
- » Avoid nested exception handling
- » Avoid duplication in catch blocks

# Functions

- » Return (avoid null)
- » Param (bool, optional, null, out), max 4
- » Size (< 10) steps and flow
- » Complexity (depth, cyclomatic complexity)
- » Dry
- » Dead code

# Review

- » Sizing (\*\*\*)
- » SOC (\*\*\*)
- » Coupling (\*\*\*)
- » Duplicate code (\*\*\*)

what do  
YOU?  
expect?

- > years of experience
- > languages
- > domain
- > expectation

# SOC

- 1.Domain logic & Domain Rules
- 2.Domain logic & Boundary Logic
- 3.Domain logic & Error handling logic
4. Flow & steps
- 5.

- » Clean Code
- » Implementation Patterns
- » Refactoring
- » Refactoring to Patterns
- » Beautiful Code
- » Xunit Patterns
- » 97 things very programmer should know
- » Solid Code
- » 50 rules for High Scalability
- » Inside C++ object model
- » Modern C++ design
- » inside JVM

# Code Smell

CODE SMELLS ARE  
SYMPTOMS OF POOR  
DESIGN OR  
IMPLEMENTATION CHOISES

[Martin Fowler]



# Solution to the spaghetti code problem

Tip:

Any fool can write code that a computer can understand.

Good programmers write code that humans can understand.

- Refactoring



```
public class SetScorer
{
    private int[] games = {0, 0};

    public void gameWon(int i) {
        games[i-1]++;
    }

    public String getSetScore() {
        if (games[0] < 6 && games[1] < 6) {
            if (games[0] > games[1]) {
                return "Player1 leads " + games[0] + " - " + games[1];
            } else if (games[1] > games[0]) {
                return "Player2 leads " + games[1] + " - " + games[0];
            } else {
                return "Set is tied at " + games[1];
            }
        }
        if (games[0] == 6 && games[1] < 5) {
            return "Player1 wins the set " + games[0] + " - " +
                games[1];
        }
        if (games[1] == 6 && games[0] < 5) {
            return "Player2 wins the set " + games[1] + " - " +
                games[0];
        }
        if (games[0] == 6 && games[1] == 5) {
            return "Player1 leads 6 - 5";
        }
        if (games[1] == 6 && games[0] == 5) {
            return "Player2 leads 6 - 5";
        }
        if (games[0] == 6 && games[1] == 6) {
            return "Set is tied at 6 games";
        }
        if (games[0] == 7) {
            return "Player1 wins the set 7 - " + games[1];
        }
        return "Player2 wins the set 7 - " + games[0];
    }
}
```

- » if, nested if
- » Magic numbers and strings
- » Duplicate (\*\*\*)
- » Function names
- » Comments ?
- »



# BOY SCOUT RULE

Leave your code better than you found it.

```
public class SetScorer {
    private int[] gamesWon = {0, 0};

    public void gameWon(int player) {
        gamesWon[player-1]++;
    }

    public String getSetScore() {
        int leader = gamesWon[0] > gamesWon[1] ? 1 : 2;
        int leadersGames = gamesWon[leader - 1];
        int opponentsGames = gamesWon[leader == 1 ? 1 : 0];
        String setScoreMessage = null;
        if ((gamesWon[0] < 6 && gamesWon[1] < 6)
            || (leadersGames == 6 && opponentsGames == 5)) {
            setScoreMessage = "Player" + leader + " leads " +
                leadersGames + " - " + opponentsGames;
        } else if (gamesWon[0] == gamesWon[1]) {
            setScoreMessage = "Set is tied at " +
                leadersGames;
        } else if ((leadersGames - opponentsGames >= 2)
            || (leadersGames == 7)) {
            setScoreMessage = "Player" + leader +
                " wins the set " + leadersGames + " - " +
                opponentsGames;
        }
        return setScoreMessage;
    }
}
```

# Problem Accumulation



Area was pristine clean for many days, one fine day a lazy guy threw garbage at the place just one bag, next one week place was filled with garbage, reason other people saw garbage and instead of picking it up they threw their own.

# Comments



```
//Add Books associated with the Category
public void add(Category category)
{
    bookMap.put(category.getCategoryId(),
                category.getBooks());
}
```

# Explain Yourself in Code

```
public void addBooksFromCategory(Category category)
{
    bookMap.put(category.getCategoryId(), category.getBooks());
}
```



i++; // increment I

## Comments are Failures

*they compensate for our inability to express in code*



```
InputStreamResponse response = new InputStreamResponse();
response.setBody(formatter.getResultStream(), formatter.getByteCount());
// InputStream resultsStream = formatter.getResultStream();
// StreamReader reader = new StreamReader(resultsStream);
// response.setContent(reader.read(formatter.getByteCount()));
```

- » Commented Code (\*\*\*)
- » Dead Code (\*\*\*)
- » Duplicate Code (\*\*\*) —> DRY



```
/*
public void registerItem(Item item) {
    if (item != null) {
        ItemRegistry registry = persistentStore.getItemRegistry();
        if (registry != null) {
            Item existing = registry.getItem(item.getID());
            if (existing.getBillingPeriod().hasRetailOwner()) {
                existing.register(item);
            }
        }
    }
}
...
*/

```



It is inappropriate for a comment to hold information better held in a source code control system, issue tracking system, or any other record-keeping system.

- Made changes
- Fixed bugs
- Small change
- Minor change
- asdf
- Updated
-



```
/**  
 * @dateModified 12/12/1947  
 * @modifiedBy RamgopalVerma  
 * @modifiedReason because sky is so high  
 * /  
public class Patient  
{  
}
```

# *Avoid Obsolete Comment*

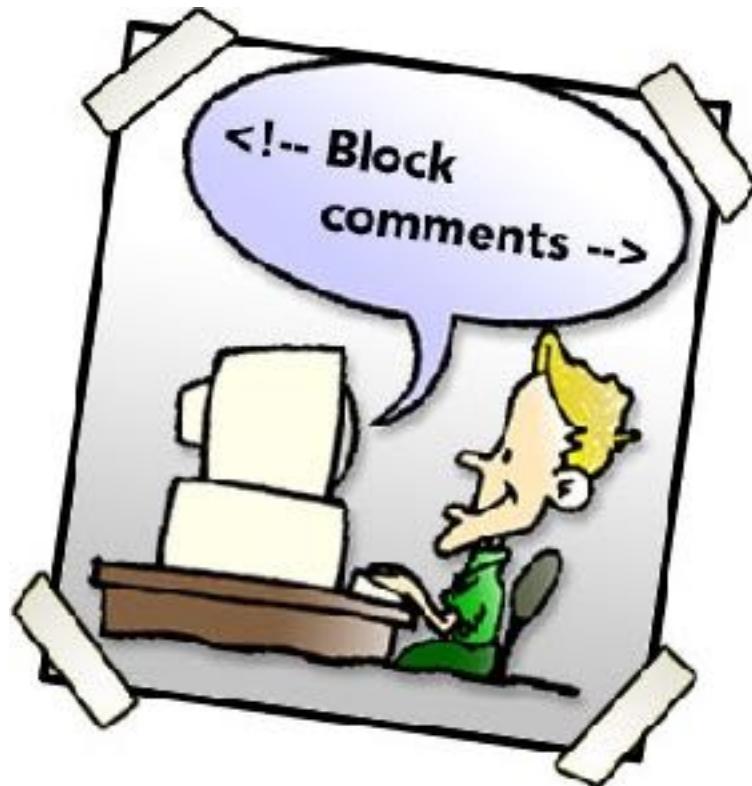


It is best not to write a comment that will become obsolete.

They become floating islands of irrelevance and misdirection in the code.



```
/**  
 * Returns the day of the month.  
 *  
 * @return the day of the month.  
 */  
public int getDayOfMonth() {  
    return dayOfMonth;  
}
```



Comments should say things that the code cannot say for itself.

# Todo



```
// Check to see if the employee is eligible for full benefits  
if ((employee.flags && HOURLY_FLAG) && (employee.age > 65))
```

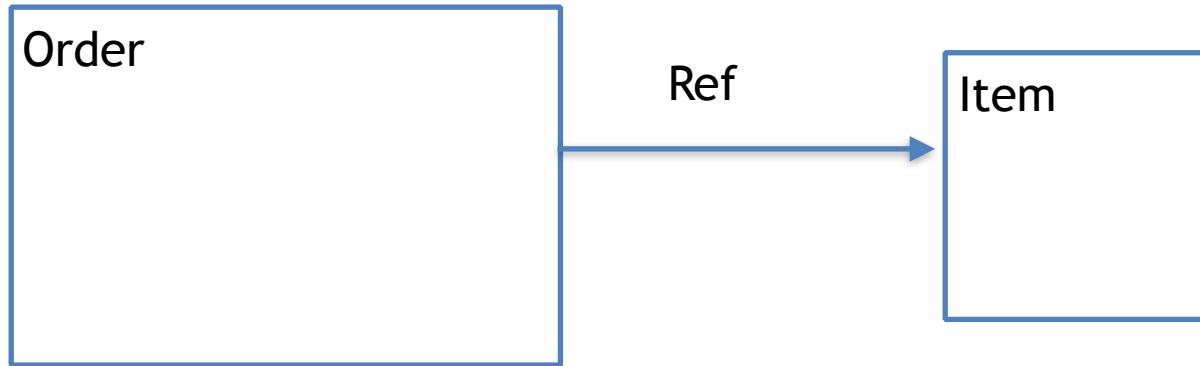
# Explain Yourself in Code

```
if (employee.isEligibleForFullBenefits())
```

# Specification pattern

```
Class Invoice
{
    InvoiceSpec spec;
    ...
    if(spec.IsExpried(data))
        ...
}
```

```
Class InvoiceSpec
{
    bool IsSufficientStock(){
        ...
    }
    bool IsExpired(){
        ...
    }
    bool IsROLevel(){
        ...
    }
}
```



```

Class Order{
    Item *item;
}
  
```

```

Class Order{
    Item *item;
}
  
```

```

Set(Item *item) {
    this->item = item;
}
~Order(){
    delete item;
}
  
```

```

Class Order{
    Item *item;
}
  
```

```

Order(Item *item) {
    this->item = item;
}
~Order(){
    delete item;
}
  
```

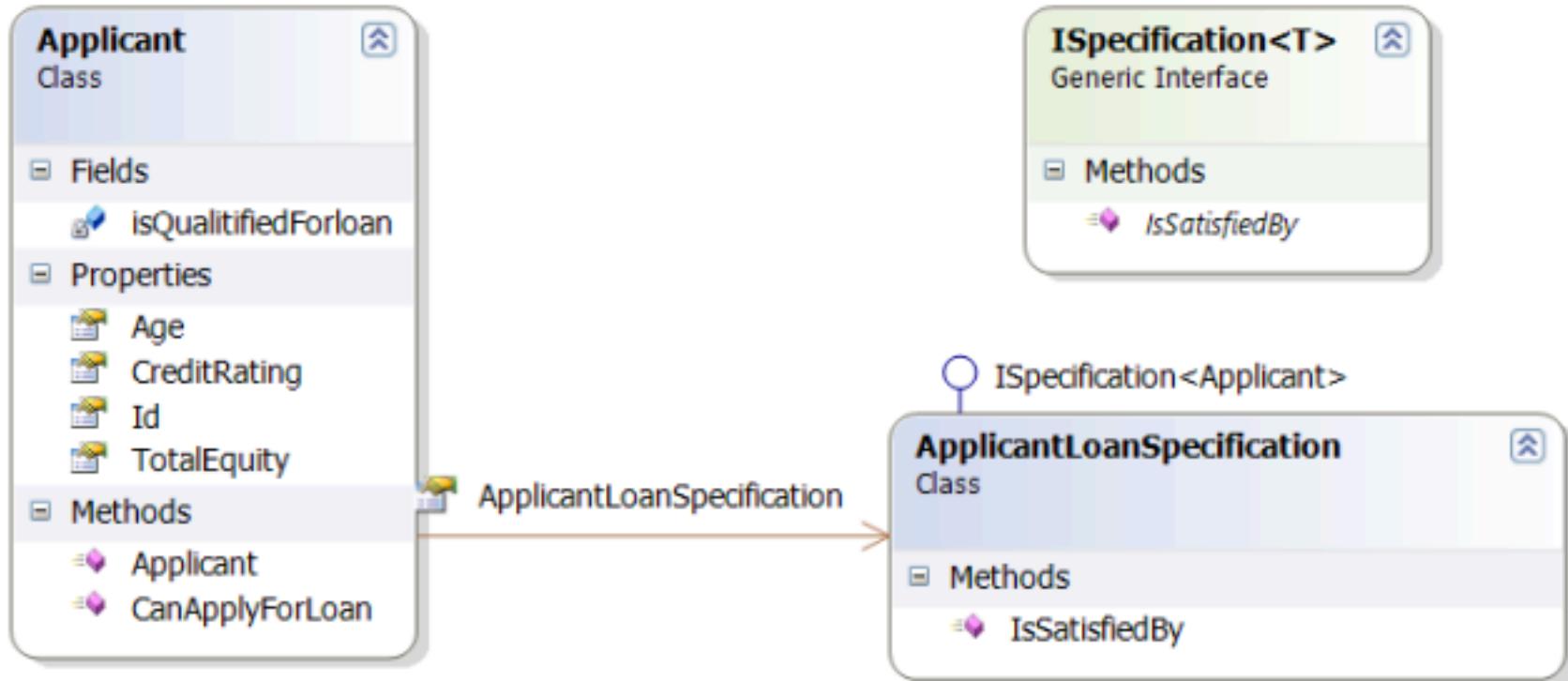
Ref ->

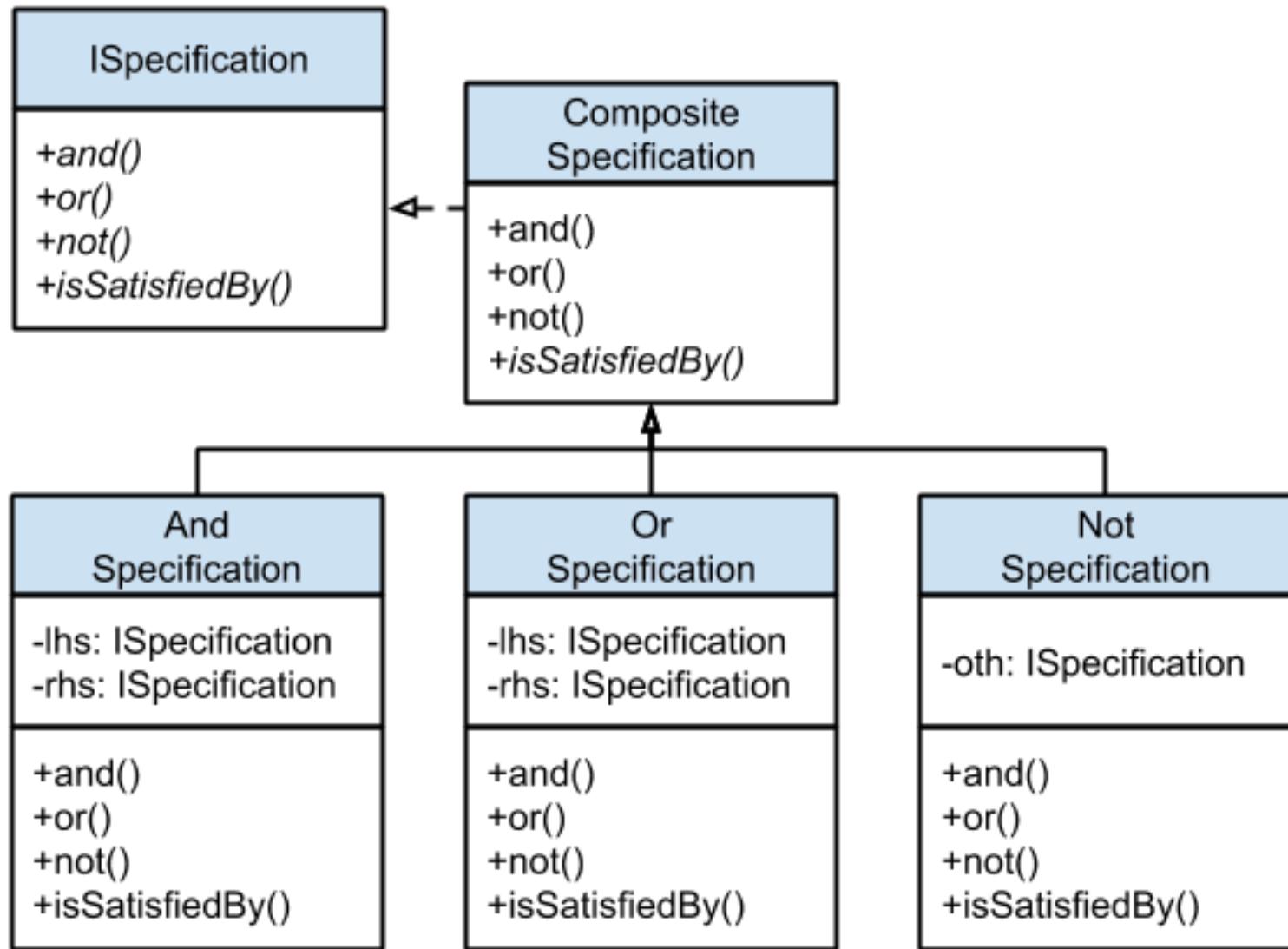
Association

->ref + owns

# Aggregation (special Association) : control life (owns)

# Composition (special Aggregation) : strong aggregation





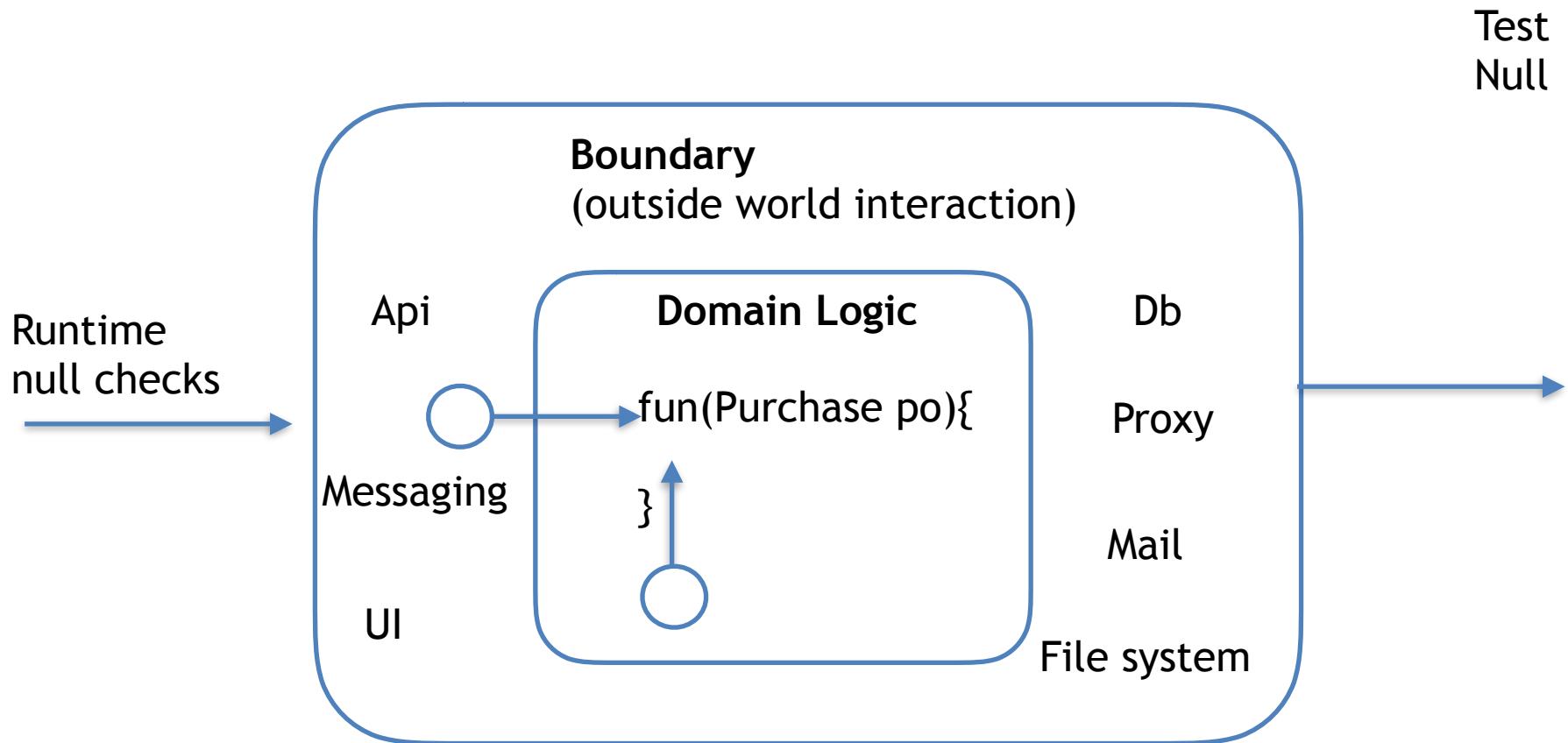


```
double getExpenseLimit() {  
    // should have either expense limit or a primary project  
    return (_expenseLimit != NULL_EXPENSE) ?  
        _expenseLimit:  
        _primaryProject.getMemberExpenseLimit();  
}
```

# Introduce Assertion

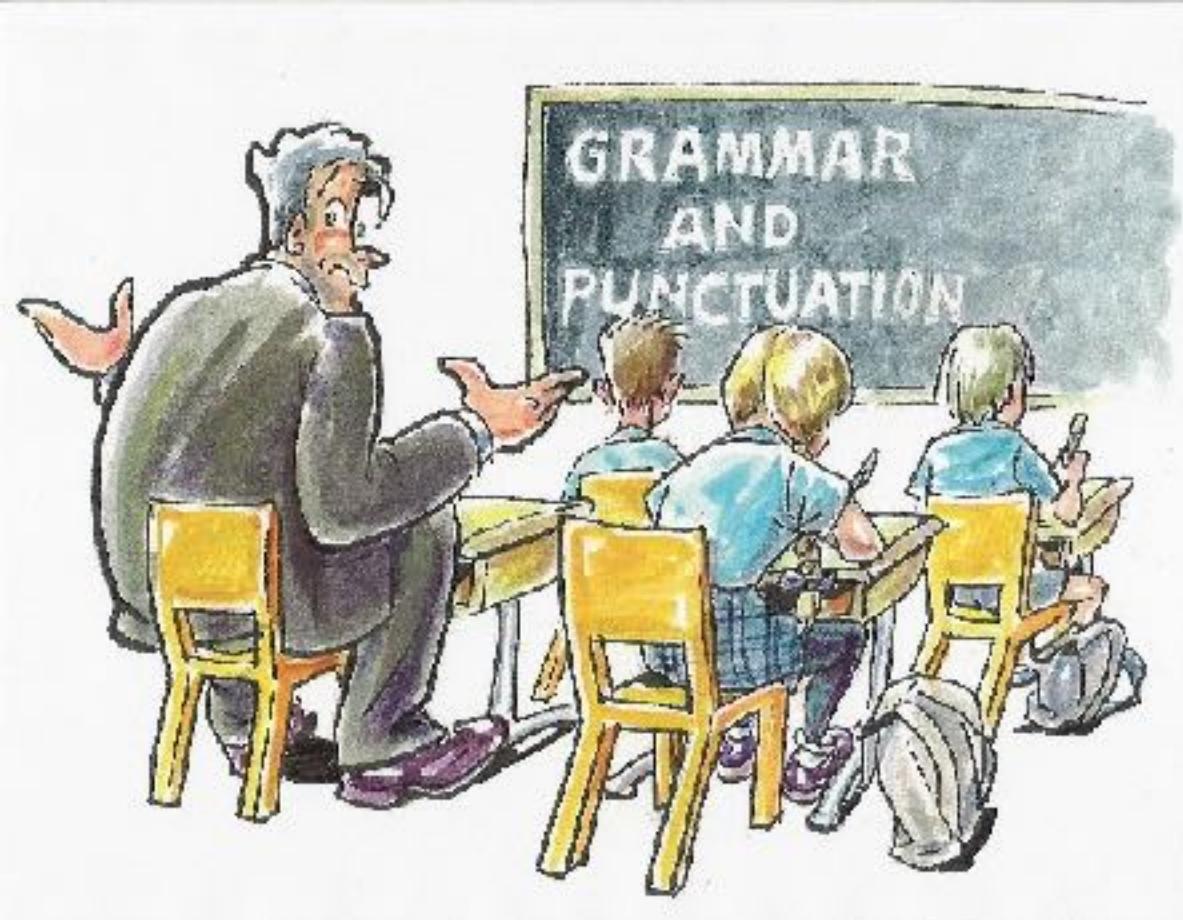
```
double getExpenseLimit() {  
    Assert.isTrue (_expenseLimit != NULL_EXPENSE || _primaryProject != null);  
  
    return (_expenseLimit != NULL_EXPENSE) ?  
        _expenseLimit:  
        _primaryProject.getMemberExpenseLimit();  
}
```

# Boundary Control Entity





```
patientCount++; // add patient hospital visiting
```



A comment worth writing is worth writing well.

# Summary

- » Renaming fun/class/ ..
- » Using Source control efficiently
- » Specification Pattern
- » Doc comments ?
- » Assert
- » Use comments only to give context
- » Boundary control Entity

# Error



```
public class DeviceController {  
    ...  
    public void sendShutdown() {  
        DeviceHandle handle = getHandle(DEV1);  
        // Check the state of the device  
        if (handle != DeviceHandle.INVALID) {  
            // Save the device status to the record field  
            retrieveDeviceRecord(handle);  
            // If not suspended, shut down  
            if (record.getStatus() != DEVICE_SUSPENDED) {  
                pauseDevice(handle);  
                clearDeviceWorkQueue(handle);  
                closeDevice(handle);  
            } else {  
                logger.log("Device suspended. Unable to shut down");  
            }  
        } else {  
            logger.log("Invalid handle for: " + DEV1.toString());  
        }  
    }  
    ...  
}
```



```
public class DeviceController {  
    ...  
    public void sendShutDown() {  
        DeviceHandle handle = getHandle(DEV1);  
        if (handle != DeviceHandle.INVALID) {  
            retrieveDeviceRecord(handle);  
            if (record.getStatus() != DEVICE_SUSPENDED) {  
                pauseDevice(handle);  
                clearDeviceWorkQueue(handle);  
                closeDevice(handle);  
            } else {  
                logger.log("Device suspended. Unable to shut down");  
            }  
        } else {  
            logger.log("Invalid handle for: " + DEV1.toString());  
        }  
    }  
    ...  
}
```

```
public bool ValidatePasswordStrength(string? password) {
    bool hasValue = !string.IsNullOrEmpty(password);

    if (hasValue) {
        bool hasSpecialCharacter = password.Any(character => !char.IsLetterOrDigit(character));

        if (hasSpecialCharacter) {
            bool longEnough = password.Length >= 10;

            if (longEnough) {
                return true;
            } else {
                throw new ArgumentException("Must be at least 10 characters");
            }
        } else {
            throw new ArgumentException("Must contain at least one special character");
        }
    } else {
        throw new ArgumentException("Password must have a value", nameof(password));
    }

    return true;
}
```



```
public class DeviceController {  
    ...  
    public void sendShutDown() {  
        DeviceHandle handle = getHandle(DEV1);  
        if (handle == DeviceHandle.INVALID) {  
            logger.log("Invalid handle for: " + DEV1.toString());  
            return;  
        }  
        retrieveDeviceRecord(handle);  
        if (record.getStatus() == DEVICE_SUSPENDED) {  
            logger.log("Device suspended. Unable to shut down");  
            return;  
        }  
        pauseDevice(handle);  
        clearDeviceWorkQueue(handle);  
        closeDevice(handle);  
    }  
    ...  
}
```

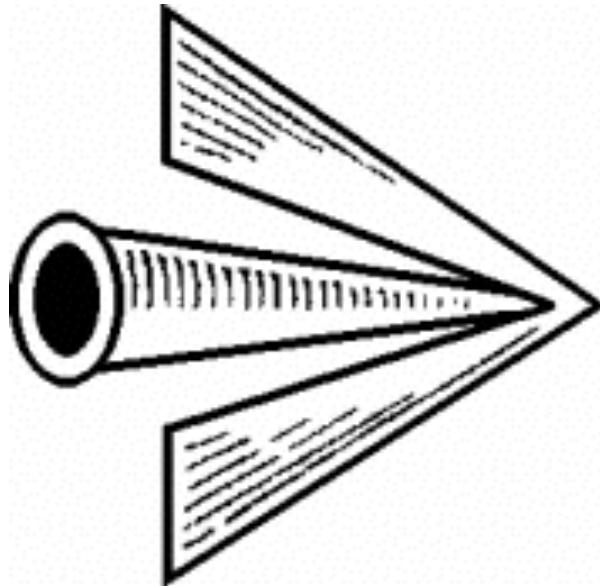
# Performance

- »  $2 + 3 \rightarrow 3$  cpu cycle
- » If
- » fun call  $\rightarrow 10$  cpu cycles (+ -)
- » Exception  $\rightarrow 3000$
- » Create Thread  $\rightarrow 200,000$
- » Destroy thread  $\rightarrow 100,000$
- » Write File  $\rightarrow 10,00,000$
- » Network db cmd execute  $\rightarrow 40,00,000$



```
void DoJob()
{
    Domain domain = new Domain();
    bool res = domain.fun();
    if(res == true){
        res = domain.fun2(100);
        if(res == true){
            Repository rep = new Repository();
            Emp emp = rep.get(1);
            if(emp != null){
                ...
            }else{
                ...
            }
        }else{
            ...
        }
    }else{
        ...
    }
}
```

# Arrow Code



```
if  
  if  
    if  
      if  
        do something  
      endif  
    endif  
  endif  
endif
```

```
function register()
{
    if (!empty($_POST)) {
        $msg = '';
        if ($_POST['user_name']) {
            if ($_POST['user_password_new']) {
                if ($_POST['user_password_new'] === $_POST['user_password_repeat']) {
                    if (strlen($_POST['user_password_new']) > 5) {
                        if (strlen($_POST['user_name']) < 65 && strlen($_POST['user_name']) > 1) {
                            if (preg_match('/^a-z\d{2,64}$/i', $_POST['user_name'])) {
                                $user = read_user($_POST['user_name']);
                                if (!isset($user['user_name'])) {
                                    if ($_POST['user_email']) {
                                        if (strlen($_POST['user_email']) < 65) {
                                            if (filter_var($_POST['user_email'], FILTER_VALIDATE_EMAIL)) {
                                                create_user();
                                                $_SESSION['msg'] = 'You are now registered so please login';
                                                header('Location: ' . $_SERVER['PHP_SELF']);
                                                exit();
                                            } else $msg = 'You must provide a valid email address';
                                        } else $msg = 'Email must be less than 64 characters';
                                    } else $msg = 'Email cannot be empty';
                                } else $msg = 'Username already exists';
                            } else $msg = 'Username must be only a-z, A-Z, 0-9';
                        } else $msg = 'Username must be between 2 and 64 characters';
                    } else $msg = 'Password must be at least 6 characters';
                } else $msg = 'Passwords do not match';
            } else $msg = 'Empty Password';
        } else $msg = 'Empty Username';
        $_SESSION['msg'] = $msg;
    }
    return register_form();
}
```



# Bad

- » bool for error (\*\*\*)
- » Null for error (\*\*\*)
- » Int for error (\*\*\*)

# Use Exceptions Rather Than Return Codes

```
public class DeviceController {  
    ...  
    public void sendShutDown() {  
        try {  
            tryToShutDown();  
        } catch (DeviceShutDownError e) {  
            logger.log(e);  
        }  
    }  
    private void tryToShutDown() throws DeviceShutDownError {  
        DeviceHandle handle = getHandle(DEV1);  
        DeviceRecord record = retrieveDeviceRecord(handle);  
        pauseDevice(handle);  
        clearDeviceWorkQueue(handle);  
        closeDevice(handle);  
    }  
    private DeviceHandle getHandle(DeviceID id) {  
        ...  
        throw new DeviceShutDownError("Invalid handle for: " + id.toString());  
        ...  
    }  
    ...  
}
```

```
public bool ValidatePasswordStrength(string? password) {
    if (string.IsNullOrEmpty(password))
        throw new ArgumentException("Cannot be null or empty");

    if (password.Any(character => !char.IsLetterOrDigit(character)))
        throw new ArgumentException("Must contain at least one special character");

    if (password.Length >= 10)
        throw new ArgumentException("Must be at least 10 characters");

    return true;
}
```



```
void DoJob()
{
    Domain domain = new Domain();
    domain.fun();
    domain.fun2(100);
    Repostory rep = new Repository();
    Emp emp = rep.get(1);
    ...
}

try{
    DoJob();
}catch(...){
    ....
}
```



```
int withdraw(int amount) {  
    if (amount > _balance)  
        return -1;  
    else {  
        _balance -= amount;  
        return 0;  
    }  
}
```



```
public void create(Map<String, Object> results) throws Exception
{
    //validate condition
    if(condition) {
        results.set("ERROR_CODE", "CONDITION VIOLATED");
    }
}

Map<String, Object> results = new HashMap<>();
create(results);
if(results.containsKey("ERROR_CODE")) {
    ...
}
```

# Replace Error Code with Exception

```
void withdraw(int amount) throws BalanceException
{
    if (amount > _balance)
        throw new BalanceException();

    _balance -= amount;
}
```



```
double getValueForPeriod (int periodNumber) {  
    try {  
        return _values[periodNumber];  
    } catch (ArrayIndexOutOfBoundsException e) {  
        return 0;  
    }  
}
```

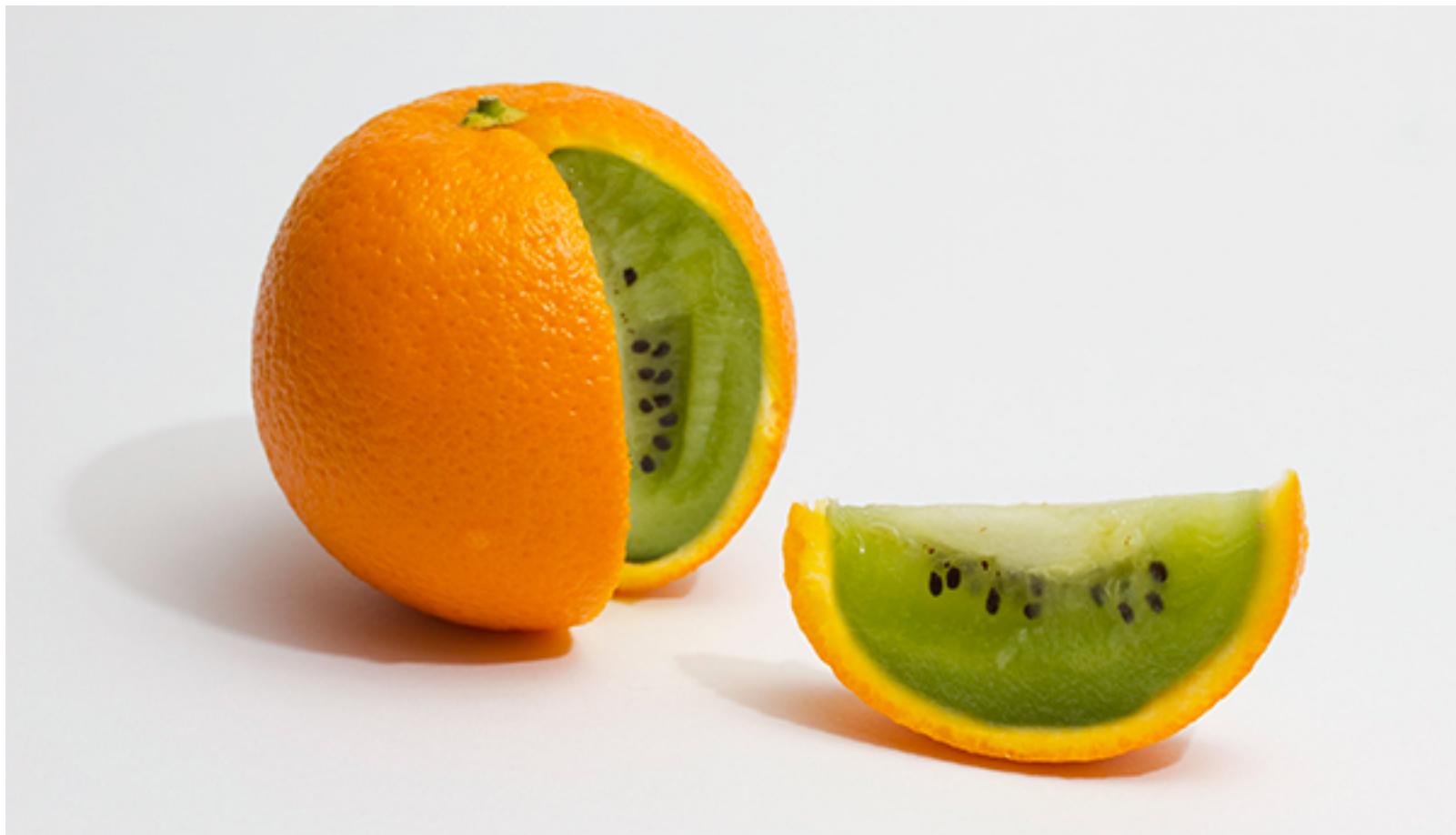
# Replace Exception with Test

```
double getValueForPeriod (int periodNumber) {  
    if (periodNumber >= _values.length)  
        return 0;  
  
    return _values[periodNumber];  
}
```



```
try
{
    while (AccountManager.HasMoreAccounts())
    {
        account = AccountManager.GetNextAccount();
        if (account.Name == userName)
        {
            //We have it
            throw new AccountFoundException(account);
        }
    }
}
catch (AccountFoundException found)
{
    Console.WriteLine("account details: " + found.Account.Details.ToString());
}
```

# Principle of Least Astonishment



# Flow

- **Main Flow:** The most common steps that achieves the use case's goal.
- **Exception Flow:** is anything that leads to NOT achieving the use case's goal. (**Termination**)
- **Alternate Flow :** achieves the use case's goal following different steps than described in the main flow. But the goal is achieved finally. (**Resumption**)

# Replace Exception with Test

```
Account found = null;  
while (AccountManager.HasMoreAccounts() && (found==null))  
{  
    account = AccountManager.GetNextAccount();  
    if (account.Name == userName)  
    {  
        //We found it  
        found = account;  
        break;  
    }  
}  
Console.WriteLine("account details: " + found.Details.ToString());
```



```
void fun(){
    try{
        ...logic 1

        if(cond)
            throw new Exception(...);

        ...payFullTimeEmployee()

    }catch(Exception e)
    {
        ...payPartTimeEmployee(...);
    }
}
```



**Farmers in India use Pepsi as pesticides, because it's cheaper than pesticides and gets the job done just as well.**



```
try
{
    var f = File.Open(myfile);
    // do something
}
catch (Exception x)
{
    // Assume file not found
    Console.WriteLine("Could not open file");
}
```

# Always catch specific Exception

```
try
{
    var f = File.Open(myfile);
    // do something which should normally not throw exceptions
}
catch (IOException)
{
    Console.WriteLine("File not found");
}
catch (UnauthorizedAccessException)
{
    Console.WriteLine("Insufficient rights");
}
```



```
private static List readLines(String fileName)
{
    String line;
    ArrayList list= new ArrayList();

    try
    {
        BufferedReader in = new BufferedReader(new FileReader(fileName));
        while ((line = in.readLine()) != null)
            list.add(line);
        in.close();
    }
    catch (Exception e)
    {
        System.out.println(e);
        return null;
    }
    return list;
}
```

# Always know why you are catching an exception



If you don't catch the exception, you get a traceback.



```
String line;
ArrayList file = new ArrayList();

try
{
    BufferedReader in = new BufferedReader(new FileReader(fileName));
    while ((line = in.readLine()) != null)
    {
        file.add(line);
    }
    in.close();
}
catch (Exception e)
{
}
```



```
//method 1
catch(SQLException exp) {
    log.fatal("Exception occurred:", exp);
    throw exp;
}

//method 2
try {
    //call method 1
}
catch(Exception exp) {
    log.fatal("Exception occurred:", exp);
}
```

# Log exception stack trace only once



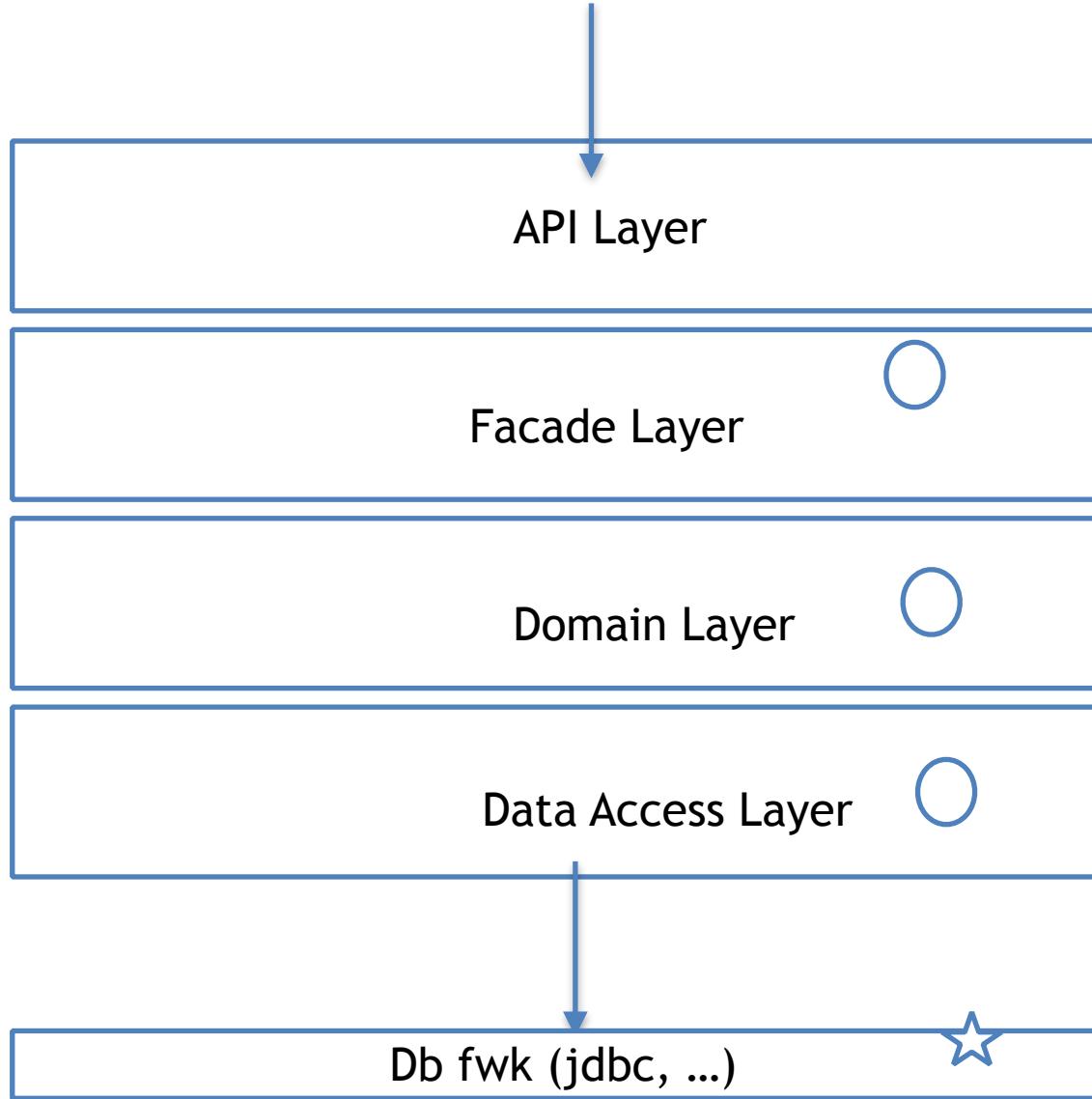
Opportunity knocks only once. You never know if you'll get another opportunity.

(Leon Spinks)

[laptopmag.com](http://laptopmag.com)



```
catch(InterruptedException e) {
    AppException exp = new AppException();
    exp.addError(Constants.ERROR_CODE, "EXP005");
    exp.addError(Constants.ERROR_MSG, e.getMessage());
    throw exp;
}
catch(SQLException e) {
    AppException exp = new AppException();
    exp.addError(Constants.ERROR_CODE, "EXP011");
    exp.addError(Constants.ERROR_MSG, e.getMessage());
    throw exp;
}
catch(Exception e) {
    AppException exp = new AppException();
    exp.addError(Constants.ERROR_CODE, "EXP003");
    exp.addError(Constants.ERROR_MSG, e.getMessage());
    throw exp;
}
```



I will not repeat myself  
I will not repeat myself

DON'T REPEAT YOURSELF

Repetition is the root of all software evil

“Perfection (in enterprise development) is achieved not when there is nothing more to add, but rather when there is nothing more to take away.”

--Antoine de Saint-Exupéry

# remove duplicate code

```
//code  
catch (Exception exp) {  
    AppException exp = new AppException();  
    exp.addError(Constants.ERROR_MESG, e.getMessage());  
    if(exp instanceof InterruptedException)  
        exp.addError(Constants.ERROR_CODE, "EXP05");  
    if(exp instanceof SQLException)  
        exp.addError(Constants.ERROR_CODE, "EXP011");  
    if(exp instanceof Exception)  
        exp.addError(Constants.ERROR_CODE, "EXP03");  
  
    throw exp;  
}
```

# remove duplicate code

```
private AppException createAppException(String errorCode, e) {  
    AppException exp = new AppException();  
    exp.addError(Constants.ERROR_CODE, errorCode);  
    exp.addError(Constants.ERROR_MESG, e.getMessage());  
    return exp;  
}  
  
//code  
catch(InterruptedException exp) {  
    throw createAppException("EXP005",exp);  
}  
catch(SQLException exp) {  
    throw createAppException("EXP011",exp);  
}  
catch(Exception exp) {  
    throw createAppException("EXP003",exp);  
}
```



```
logger.error("Exception occurred:" , e.getMessage());
```

# the exception stacktrace is lost





```
try {  
    ...  
  
} catch (Exception e) {  
    LOGGER.error("Exception :", e);  
    LOGGER.debug("Exception :" + e.getMessage());  
    try {  
        con.rollback();  
    } catch (SQLException e1) {  
        LOGGER.error("Exception :", e1);  
        LOGGER.debug("Exception :" + e1.getMessage());  
    }  
} finally {  
    try {  
        con.close();  
    } catch (SQLException e1) {  
        LOGGER.error("Exception :", e1);  
        LOGGER.debug("Exception :" + e1.getMessage());  
    }  
}
```

## ■ Unchecked Exceptions

- No more **try/throws**/empty **catch()**; no return status codes
- Runtime Exceptions don't annoy the caller
  - Can be handled uniformly in a transparent layer at the entry points
  - Can hold enum error codes for (i18n) user messages
  - Define custom exception sub-types for recoverable errors (selective catch)

# Summary

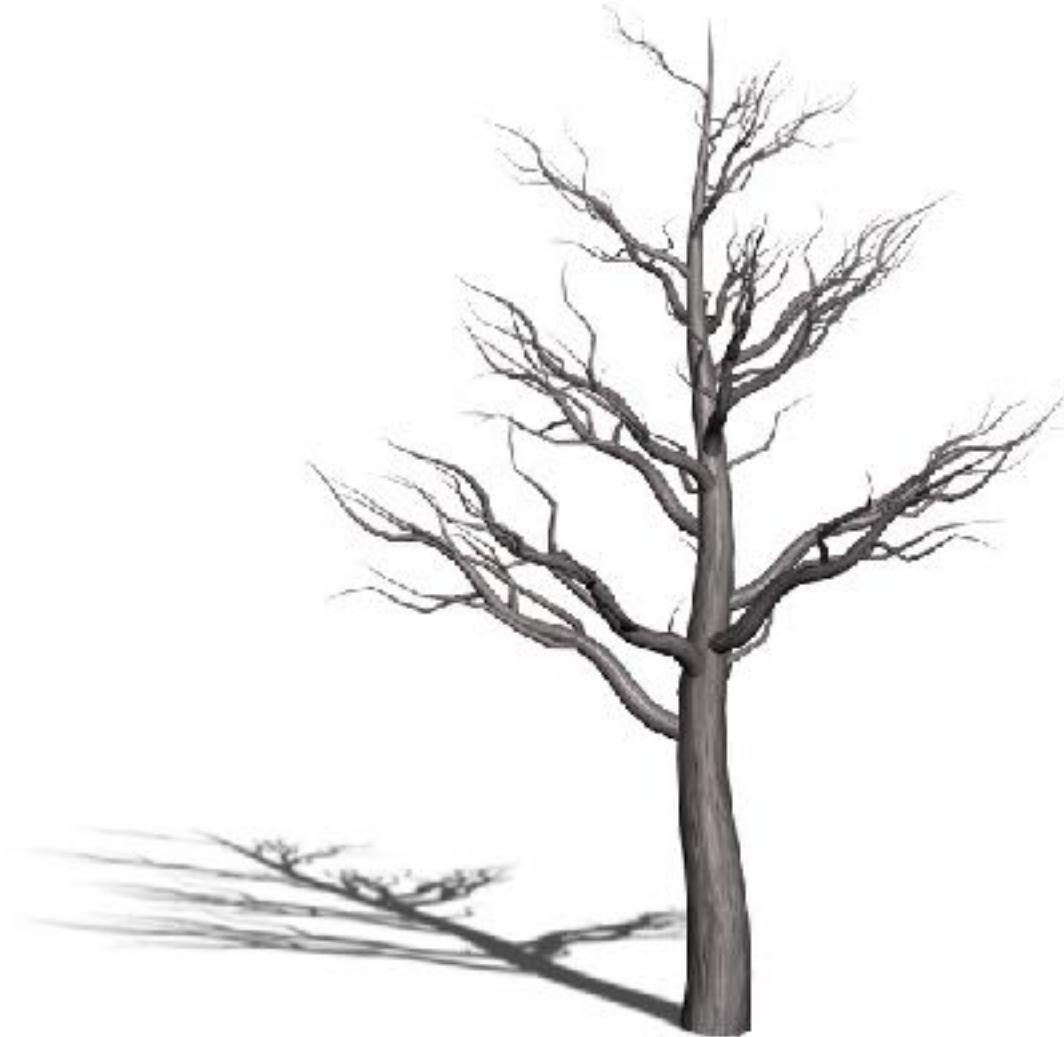
- » Avoid bool, int, null for error
- » Avoid Arrow code
- » POLA
- » Throw Exceptions for error flow
- » dont throw exceptions for alternate flow
- » Avoid Empty catch blocks, log stack trace,
- » Avoid rethrow, avoid nested try

# **functions**

# *Dead Function*



Methods that are never called.



Ruthlessly delete code that isn't being used.



```
double potentialEnergy(double mass, double height) {  
    return mass * height * 9.81;  
}
```



# Magic Numbers and Strings

*Explicit is better than implicit. - Tim Peters, The Zen of Python*

# Replace Magic Number with Symbolic Constant

```
double potentialEnergy(double mass, double height) {  
    return mass * GRAVITATIONAL_CONSTANT * height;  
}  
static final double GRAVITATIONAL_CONSTANT = 9.81;
```



```
class Constants  
{
```

```
    public final static String TOTO = "toto";  
    public final static Integer TEN = 10;  
    public final static String NAME=“name1”;  
    public final static String MAX_VAL=25;  
    //..  
}
```

- » Developer Config (build time) constants
  - » max\_size
  - » max\_length
  - » ...
- » Operational Config (deployment time) environment <— key value store
  - » Port
  - » Machine
  - » Disk
  - » Network
- » User Config (runtime) db / Domain config which can be modified by business users
  - » Profile
  - » Theme
  - » Settings
  - » Domain Rule

Inventory

InventoryConst

C3

Invoice

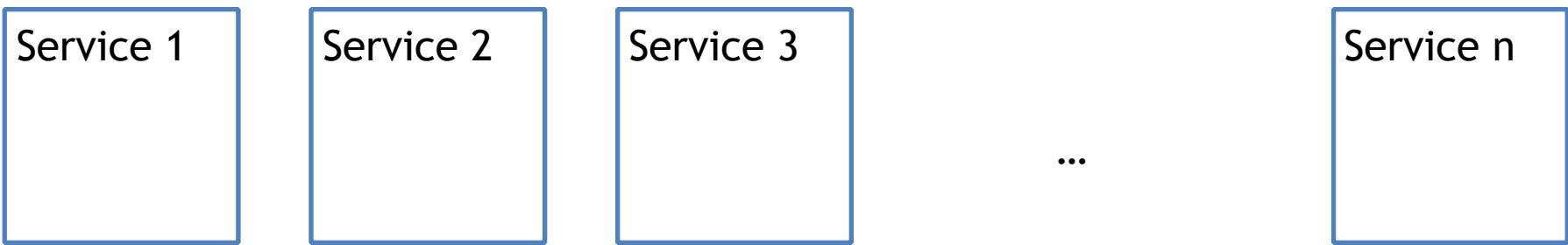
C1

Account

Order

C2

Localize domain, centralise technical concerns



# Authentication  
# Authorization  
# Audit  
# Logging  
# Caching  
# Config <-- redis, consul, simple db, ...  
# Monitoring  
# ...



- DO NOT create a dreadful 4000-lines Constants class !

- » Handler
  - » Util
  - » Const
  - » Helper
  - » Service
  - » Manager
  - » Controller
  - »
- » God Class
  - » Swiss Knife



0 references

```
public void CompareString_Dirty()
{
    Employee employee = new Employee();
    if (employee.EmployeeType == "Contractor")
    {
        // do some action
    }
}
```

```
0 references
public void CompareString_CodeReview()
{
    Employee employee = new Employee();
    if (employee.TypeOfEmployee == EmployeeType.Contractor)
    {
        // do some action
    }
}
```

leverage an Enum to make a  
strongly typed check



```
while ((!found) && (pos < (fileContent.Length - 6)))
{
    byteData = new byte[6];
    Array.Copy(fileContent, pos, byteData, 0, 6);
    pos = pos + 6;
    str_byteData = enc.GetString(byteData);
    if (str_byteData.Contains("s"))
    {
        posE_byteData = str_byteData.IndexOf("s");
        pos = pos + (posE_byteData - 6);
        Array.Copy(fileContent, pos, byteData, 0, 6);
        pos = pos + 6;
        if (byteData[0] == 0x73) // 's'
        {
            if (byteData[1] == 0x74) // 't'
            {
                if (byteData[2] == 0x72) // 'r'
                {
                    if (byteData[3] == 0x65) // 'e'
                    {
                        if (byteData[4] == 0x61) // 'a'
                        {
                            if (byteData[5] == 0x6D) // 'd'
                            {
                                found = true;
                                break;
                            }
                        else
                        {
                            if (byteData[5] == 0x73)
                            {
                                pos = pos - 1;
                            }
                        }
                    }
                }
            }
        }
    }
}
```

# Nesting Levels



```
{if < 4 ,  
switch,while,do, for <3}
```

```
FormView1.FindControl("noteRow").Visible = true;
FormView1.FindControl("RequiredFieldValidator4").Visible = false;
// TODO: Fix this - What if Action is 2nd parameter in URL
if (Request.Path.Contains(@"clientProfile.aspx?Action=Add"))
{
    FormView1.FindControl("thLoginID").Visible = true;
    FormView1.FindControl("tdLoginID").Visible = true;
    FormView1.FindControl("txtLoginID").Visible = true;
}

}
else
{
    FormView1.FindControl("noteRow").Visible = false;
}

if (HttpContext.Current.User.IsInRole("Administrator"))
{
    if (!request.Path.Contains(@"/administratorProfile.aspx"))
    {
        if (!request.Path.Contains(@"/profile.aspx"))
        {
            FormView1.FindControl("txtLoginID").Visible = false;
            FormView1.FindControl("lblLoginID").Visible = false;
            -- Skip skip skip - more controls are hidden or shown -- --
            if (Request.Path.Contains(@"/UserProfile.aspx") && HttpContext.Current.User.IsInRole("User") && (Request.QueryString["Action"] == null || 
            {
                FormView1.FindControl("thLoginID").Visible = true;
                FormView1.FindControl("lblLoginID").Visible = true;
                -- Skip skip skip - more controls are hidden or shown -- --
            }
            else
            {
                FormView1.FindControl("thLoginID").Visible = false;
                FormView1.FindControl("lblLoginID").Visible = false;
                -- Skip skip skip - more controls are hidden or shown -- --
            }
        }
    }
}
else
{
    FormView1.FindControl("thDateRegistered").Visible = false;
    FormView1.FindControl("tdDateRegistered").Visible = false;
    -- Skip skip skip - more controls are hidden or shown -- --
}

}
else_if (request.Path.Contains(@"/readerProfile.aspx"))
{
    FormView1.FindControl("lblAdviser").Visible = false;
    FormView1.FindControl("thAdviser").Visible = false;
    -- Skip skip skip - more controls are hidden or shown -- --
}

}
else_if (HttpContext.Current.User.IsInRole("Administrator"))
{
```

# Long Method





# Single Responsibility Principle

Yes, it's all there. No, it's not a good idea.

# Functions Should be Small !!

## ■ How small ?

- 4 *lines* is best; 5 is ok; 6 is already too much
- By any means, keep your functions smaller than a page of your IDE!

## ■ Why so small ?!

→ To be sure that they do only 1 THING

- In a few lines, you can't do much
- Its name can then tell you everything

1.10 functions with 50 lines each  
2.50 functions with 10 lines each

<i>500 lines of code</i>	<i>(1) - dense 10 fun 50 lines each</i>	<i>(2) - sparse 50 fun 10 lines each</i>
<i>Perf</i>	~	~
<i>Easy to Name</i>		+
<i>Unit test</i>		+
<i>Readability</i>		+
<i>Agility to change</i>		+
<i>Reusability</i>		+

# Performance

- » a + b -> 3 cycle
- » Fun call -> ~10 cycle
- » Throw ex -> 2000 cycles
- » Create thread -> 200,000 cycles
- » Destroy thread -> 100,000 cycles
- » WriteFile -> 10,00,0000 cycles
- » Db cmd execute -> 40,00,000 cycles

```
public void transferFunds(Account source, Account target, BigDecimal amount, boolean allowDuplicateTxn){  
    Account sourceAccount = null;  
    if(rs.next()) {  
        sourceAccount = new Account();  
        //populate account  
    }  
    Account targetAccount = null;  
    if(!sourceAccount.isOverdraftAllowed()) {  
        if((sourceAccount.getBalance() - amount) < 0) {  
            throw new RuntimeException("Insufficient Balance");  
        }  
    }  
    else {  
        if(((sourceAccount.getBalance()+sourceAccount.getOverdraftLimit()) - amount) < 0) {  
            throw new RuntimeException("Insufficient Balance, Exceeding Overdraft Limit");  
        }  
    }  
    AccountTransaction lastTxn = .. ; //JDBC code to obtain last transaction of sourceAccount  
    if(lastTxn != null) {  
        if(lastTxn.getTargetAcno().equals(targetAccount.getAcno()) && lastTxn.getAmount() == amount && !  
        allowDuplicateTxn) {  
            throw new RuntimeException("Duplicate transaction exception");  
        }  
    }  
    sourceAccount.debit(amount);  
    targetAccount.credit(amount);  
}
```



```
Public void doTheDomesticThings()
{
    takeOutTheTrash();
    walkTheDog();
    for(Dish dish : dirtyDishStack)
    {
        sink.washDish(dish);
        teaTowel.dryDish(dish);
    }
}
```

# Don't mix levels of abstraction

```
Public void doTheDomesticThings()
{
    takeOutTheTrash();
    walkTheDog();
    doTheDishes();
}
```



```
public List<ResultDto> buildResult(Set<ResultEntity> resultSet) {  
    List<ResultDto> result = new ArrayList<>();  
    for (ResultEntity entity : resultSet) {  
        ResultDto dto = new ResultDto();  
        dto.setShoeSize(entity.getShoeSize());  
        dto.setNumberOfEarthWorms(entity.getNumberOfEarthWorms());  
        dto.setAge(computeAge(entity.getBirthday()));  
        result.add(dto);  
    }  
    return result;  
}
```

# Don't mix levels of abstraction

```
public List<ResultDto> buildResult(Set<ResultEntity> resultSet) {
    List<ResultDto> result = new ArrayList<>();
    for (ResultEntity entity : resultSet) {
        result.add(toDto(entity));
    }
    return result;
}

private ResultDto toDto(ResultEntity entity) {
    ResultDto dto = new ResultDto();
    dto.setShoeSize(entity.getShoeSize());
    dto.setNumberOfEarthWorms(entity.getNumberOfEarthWorms());
    dto.setAge(computeAge(entity.getBirthday()));
    return dto;
}
```

```
private Account validateAndGetAccount(String acno){  
    Account account = AccountDAO.getAccount(acno);  
    if(account == null){  
        throw new InvalidAccountException("Invalid ACNO :" +acno);  
    }  
}  
  
private void checkForOverdraft(Account account, BigDecimal amount){  
    if(!account.isOverdraftAllowed()){  
        if((account.getBalance() - amount) < 0){  
            throw new InsufficientBalanceException("Insufficient Balance");  
        }  
    }  
    else{  
        if(((account.getBalance() + account.getOverdraftLimit()) - amount) < 0){  
            throw new ExceedingOverdraftLimitException("Insufficient Balance, Exceeding Overdraft Limit");  
        }  
    }  
}  
  
private void checkForDuplicateTransaction(FundTransferTxn txn){  
    AccountTransaction lastTxn = TransactionDAO.getLastTransaction(txn.getSourceAccount().getAcno());  
    if(lastTxn != null){  
        if(lastTxn.getTargetAcno().equals(txn.getTargetAccount().getAcno())  
        && lastTxn.getAmount() == txn.getAmount()  
        && !txn.isAllowDuplicateTxn()){  
            throw new DuplicateTransactionException("Duplicate transaction exception");  
        }  
    }  
}  
  
private void makeTransfer(Account source, Account target, BigDecimal amount){  
    sourceAccount.debit(amount);  
    targetAccount.credit(amount);  
    TransactionService.saveTransaction(source, target, amount);  
}
```

```
public void transferFunds(FundTransferTxn txn) {  
  
    Account sourceAccount = validateAndGetAccount(txn.getSourceAccount().getAcno());  
    Account targetAccount = validateAndGetAccount(txn.getTargetAccount().getAcno());  
    checkForOverdraft(sourceAccount, txn.getAmount());  
    checkForDuplicateTransaction(txn);  
    makeTransfer(sourceAccount, targetAccount, txn.getAmount());  
}
```



```
double getPrice()
{
    int basePrice = _quantity * _itemPrice;

    double discountFactor;
    if(basePrice > 1000)
        discountFactor = 0.95;
    else
        discountFactor = 0.98;

    return basePrice * discountFactor;
}
```

# Replace Temp with Query

```
double getPrice()
{
    return basePrice() * discountFactor();
}
private double discountFactor()
{
    if(basePrice > 1000)
        return 0.95;
    else
        return 0.98;
}
private int basePrice()
{
    return _quantity * _itemPrice;
}
```



```
File CreateFile(string name, boolean isTemp){
```

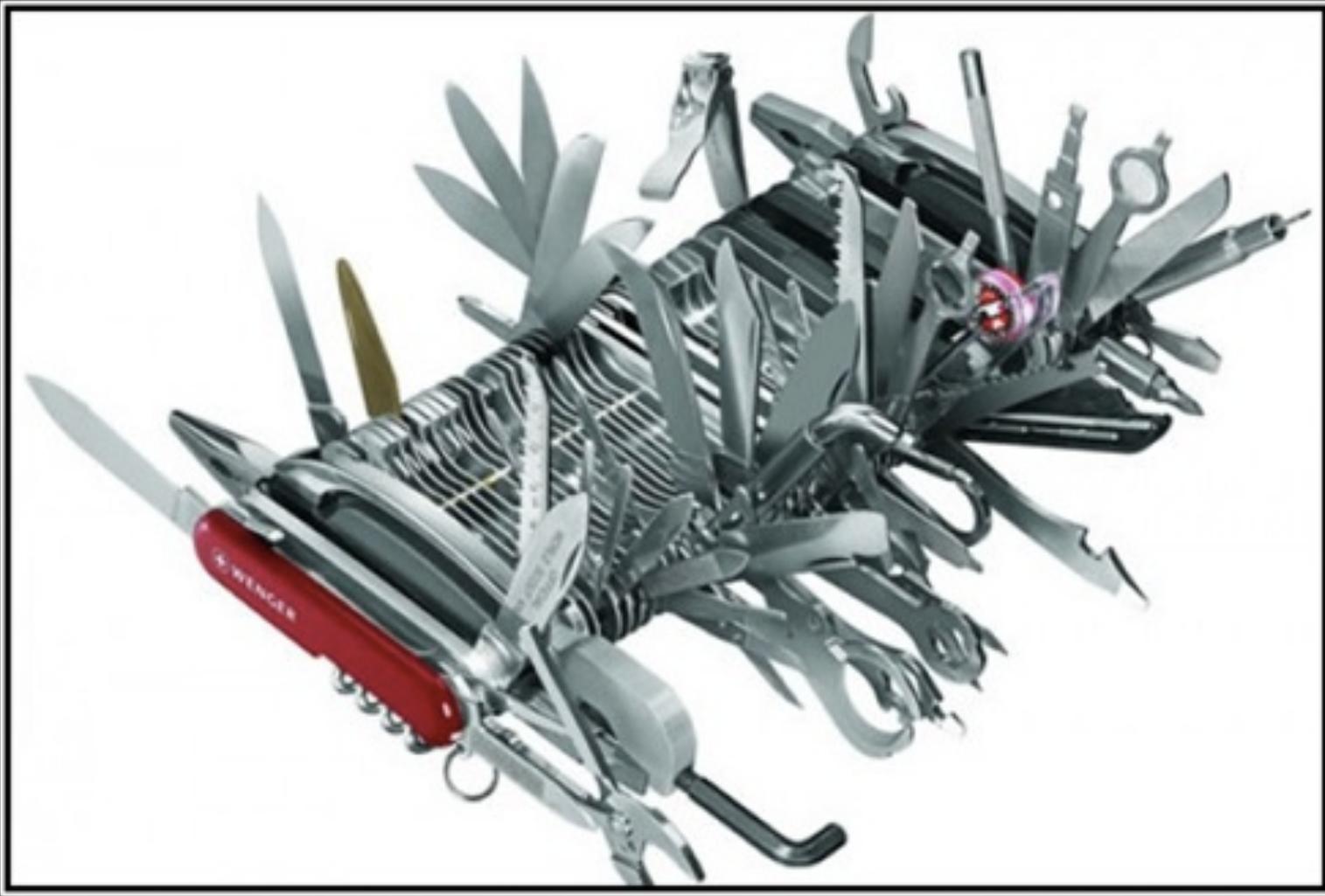
```
    ...
```

```
}
```

```
myfile = CreateFile("foo.txt", true);
```



```
void PrintReceipt(bool printDetails)
{
    foreach(Item item in this.order)
    {
        PrintLine(item.Name);
        if(printDetails)
        {
            PrintLine(item.Description);
        }
    }
}
```



## SINGLE RESPONSIBILITY PRINCIPLE

Every object should have a single responsibility, and all its services should be narrowly aligned with that responsibility.

# *Flag Arguments are Ugly*

```
customer.setLoggedIn(true);
```

```
removeOrders(customer, false, true);
```

- It shouts that **the function does more than one thing !**

```
bool fun(){  
    ...  
}
```

```
void fun(bool param){  
    ...  
}
```

# Reveal Intent

```
File CreatePermanentFile(String name);  
File CreateTempFile(String name);
```

# Reveal Intent

```
myFile = CreateFile("foo.txt", FileType.Temp);
```

# Reveal Intent

```
isTemp = true;  
CreateFile("foo.bar", isTemp);
```

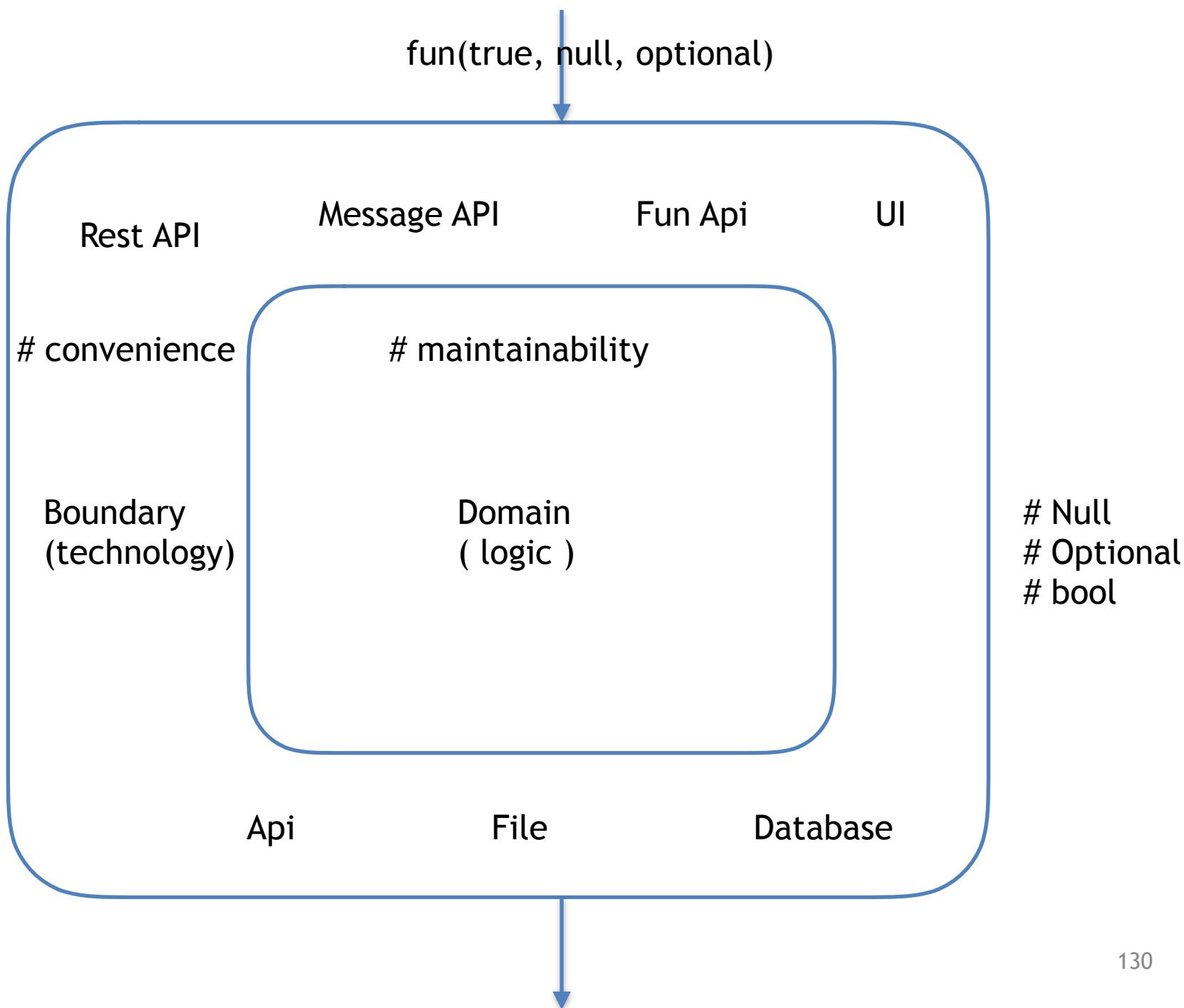
If you have to call an existing API that uses this style,  
you should introduce a local variable for clarity.



```
void myMethod(Optional<String> p)
{
    ...
}
```

# It Causes Conditional Logic Inside Methods

```
public void myMethod(Optional<String> p) {  
    // some code  
    if (p.isPresent()) {  
        doSomething(p.get());  
    } else {  
        doSomethingElse();  
    }  
    // some code  
}
```



#1

```
fun(Emp obj){  
    obj.do();  
}
```

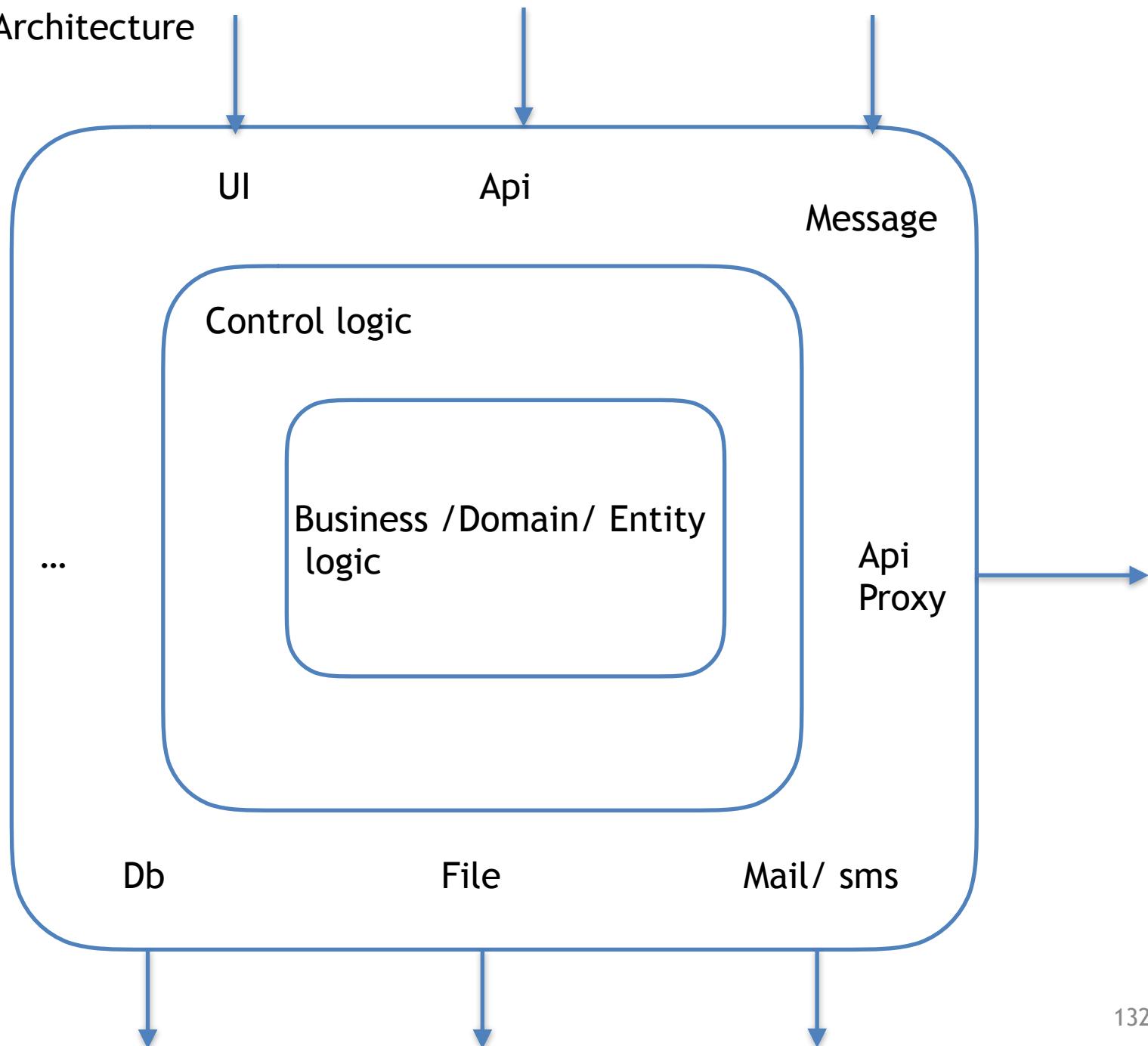
#2

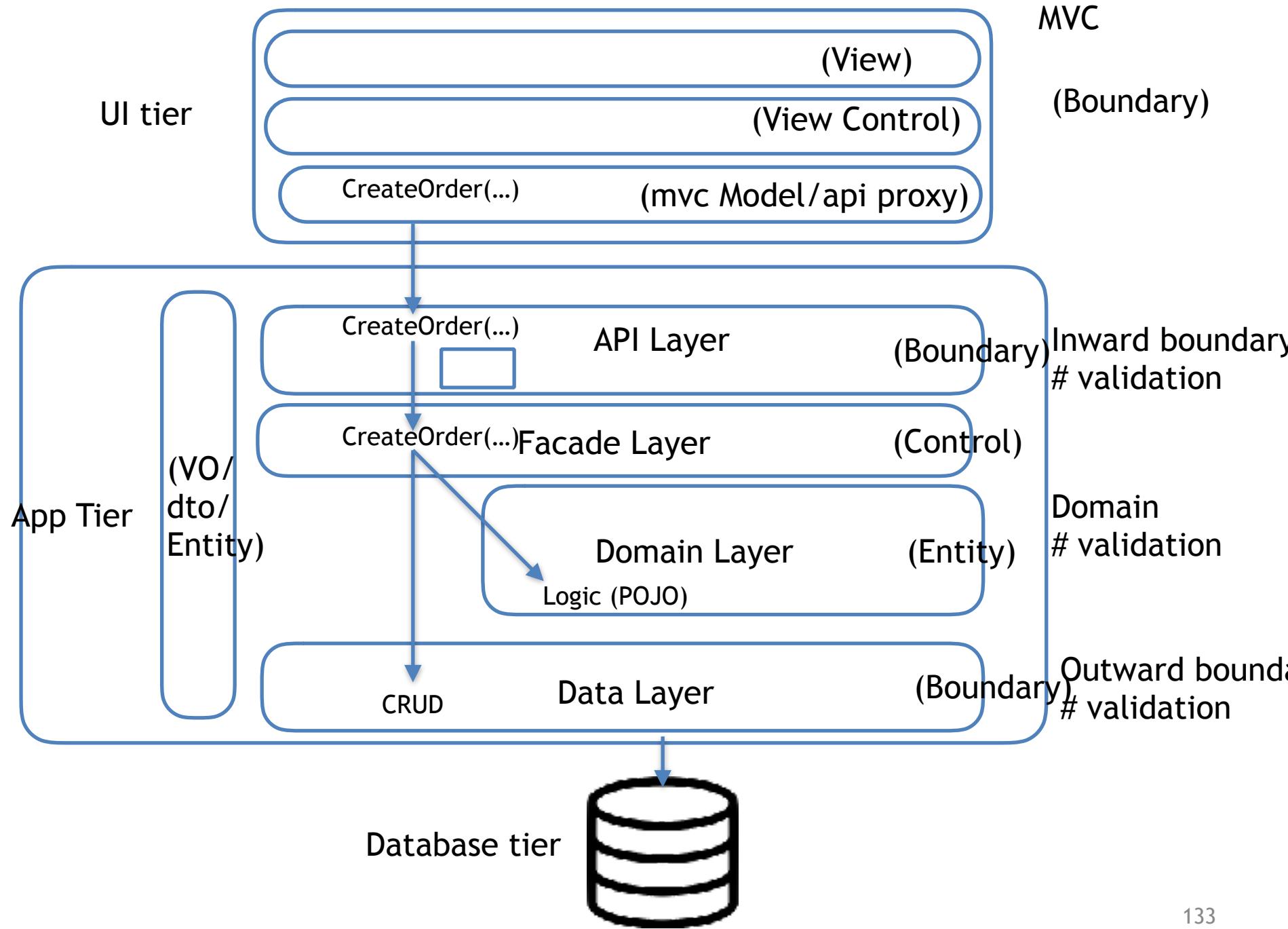
```
fun(Emp obj){  
    if(obj ==null)  
        return;  
  
    obj.do();  
}
```

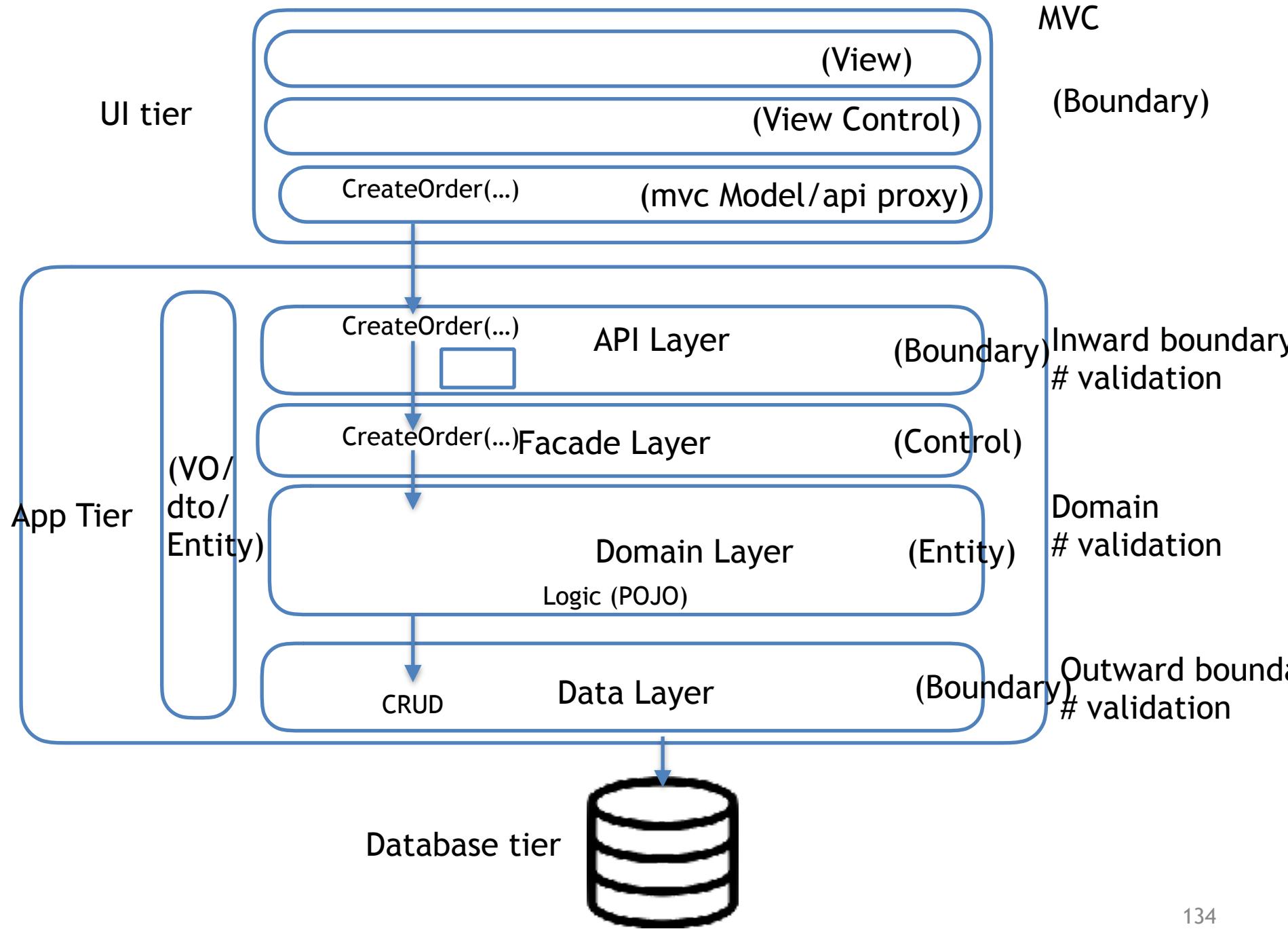
Domain  
( logic )

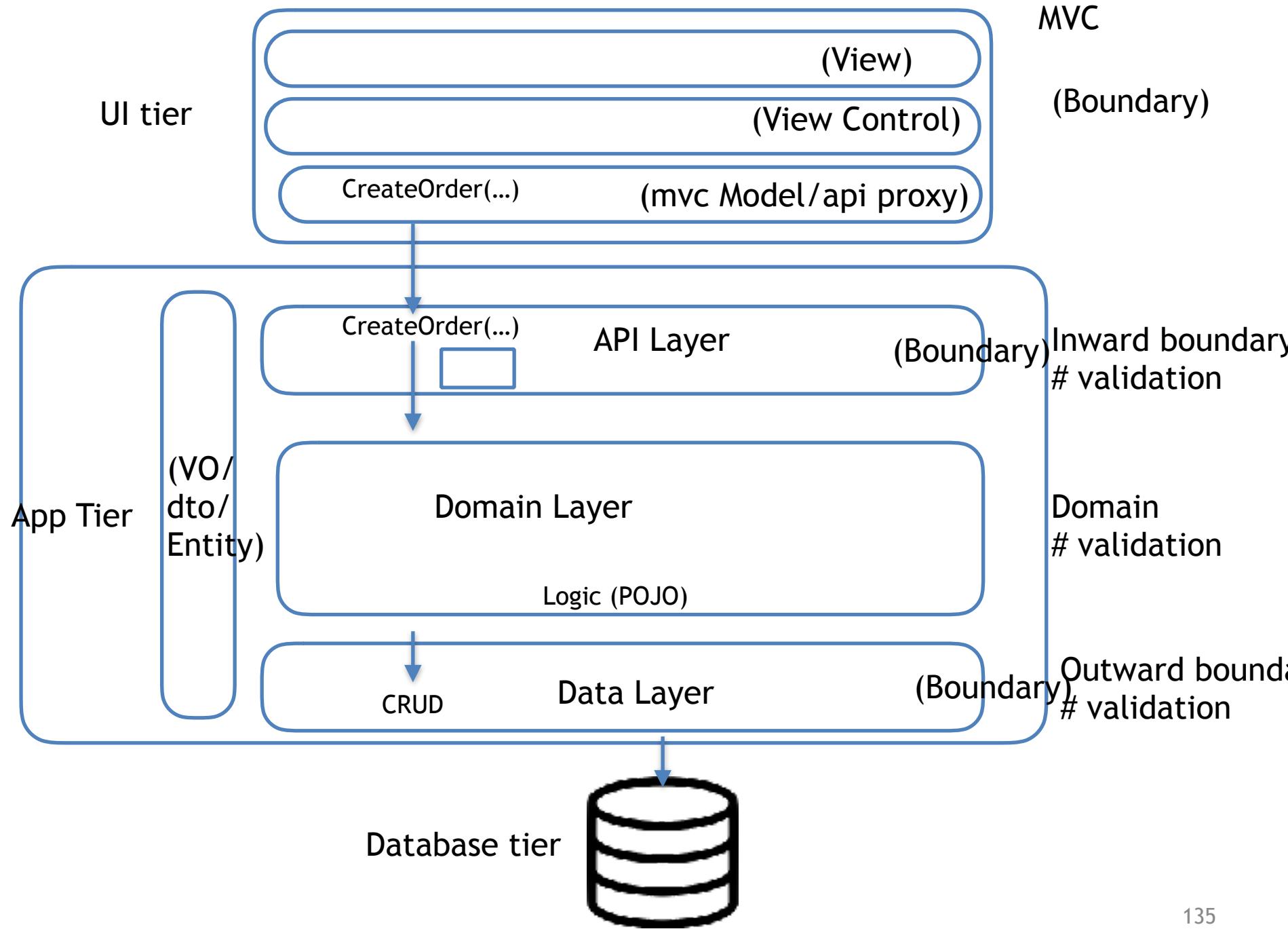
Boundary  
( technology )

## Hexagonal Architecture











```
void groupNumbers(List<Integer> numbers, List<Integer>
positiveList, List<Integer> negativeList) {

    for (Integer item: numbers)
        if (item >= 0)
            positiveList.add(item);
        else
            negativeList.add(item);
}
```

# *Avoid output parameters*

```
Tuple<List,List> sortNumbers(List<Integer> l1) {  
  
    List<Integer> l2 = new List<Integer>();  
    List<Integer> l3 = new List<Integer>();  
  
    for (Integer item: l1)  
        if (item > 0)  
            l2.add(item);  
        else  
            l3.add(item);  
  
    return new Tuple<List,List>(l2,l3);  
}
```

- The caller may be surprised that an argument changed.  
Did you expect that the second parameter will be changed  
when you call that method?



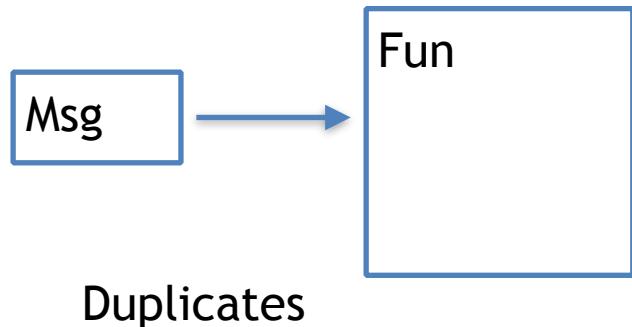
## principle of least astonishment

the name of a function should reflect what it does

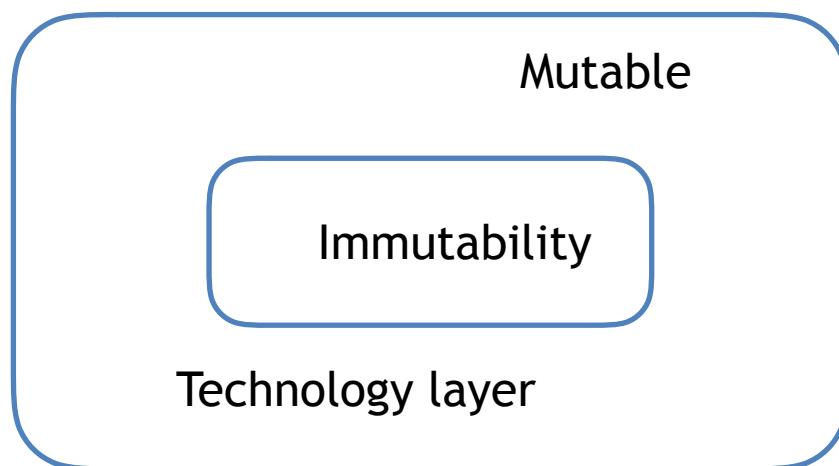
result of performing some operation should be obvious, consistent, and predictable, based upon the name of the operation

# Embrace Immutability

```
const int fun(const int i, const int j){  
    return i +j;  
}
```



# Unit test  
# Concurrency  
# Idempotency (Messaging)



No logic



```
myBadMethod( "John" , "Michael" ,  
             "Paris" , "St. Albergue" );
```

# Max 2 parameters

```
params.setStreet("Paris");  
...  
myBadMethod(params);
```

- For  $\geq 3$  parameters, it's hard to read: their order becomes a problem  
→ group them in a **Parameter Object/DTO**



```
void bad_handle_data(char *data, size_t length)
{
    if (check_CRC(data, length) == OK) {
        main_flow(data, length);
    }
    else {
        alternate_flow(data, length);
    }
}
```

# Test Exception instead of normal code

```
void good_handle_data(char *data, size_t length)
{
    if (check_CRC(data, length) != OK)
    {
        AlternateFlow();
        return;
    }

    MainFlow();
}
```



```
List<Employee> employees = getEmployees();  
if (employees != null) {  
    for(Employee e : employees) {  
        totalPay += e.getPay();  
    }  
}  
...  
...
```

```
Man findABrotherInLawOf(Man user) {  
    Man anyBrotherInLaw = null;  
    Woman userWife = user.getWife();  
  
    if (userWife != null) {  
        List<Man> brothers = userWife.getBrothers();  
  
        if (brothers != null) {  
            Battery firstBrother = brothers.get(0);  
  
            if (firstBrother != null) {  
                anyBrotherInLaw = firstBrother;  
            }  
        }  
    }  
    return anyBrotherInLaw;  
}
```



```
public void registerItem(Item item) {  
    if (item != null) {  
        ItemRegistry registry = persistentStore.getItemRegistry();  
        if (registry != null && registry.Count > 0) {  
            Item existing = registry.getItem(item.getID());  
            if (existing != null && existing.getBillingPeriod().hasRetailOwner()) {  
                existing.register(item);  
            }  
            if (existing != null && !existing.getBillingPeriod().hasRetailOwner()) {  
                existing.unregister(item);  
            }  
            if (existing != null) {  
                existing.remove(item);  
            }  
        }  
    }  
}
```

# Don't Return Null

If we change getEmployee so that it returns an empty list

```
List<Employee> employees = getEmployees();
for(Employee e : employees) {
    totalPay += e.getPay();
}
```

Fortunately, Java has Collections.emptyList(), and it returns a predefined immutable

```
public List<Employee> getEmployees() {
if( .. there are no employees .. )
    return Collections.emptyList();
}
```



```
public void registerItem(Item item) {  
    if (item != null) {  
        ItemRegistry registry =  
            persistentStore.getItemRegistry();  
        Item existing = registry.getItem(item.getID());  
        if (existing.getBillingPeriod().hasRetailOwner()) {  
            existing.register(item);  
        }  
    }  
}
```

# No nullable parameters

```
public void registerItem(Item item) {  
    ItemRegistry registry = persistentStore.getItemRegistry();  
    Item existing = registry.getItem(item.getID());  
    if (existing.getBillingPeriod().hasRetailOwner()) {  
        existing.register(item);  
    }  
}
```

- It's like a **boolean**  $\Leftrightarrow$  split that function in 2: one for null, one for not-null
- Thoroughly check parameters at boundaries (=defensive programming)



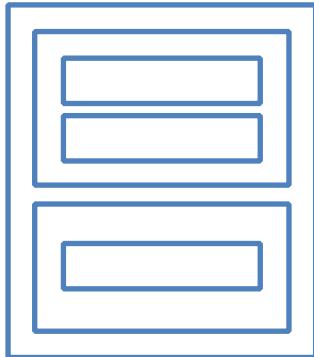
```
if (thisString == null || thisString == 0)  
{  
// code to execute  
}
```

# Avoid null

```
if (string.IsNullOrEmpty(thisString))  
{  
    //code to execute  
}
```



```
Outer outer = getObj();  
  
if (outer != null && outer.nested != null && outer.nested.inner != null) {  
    System.out.println(outer.nested.inner.foo);  
}
```



# Removing null check

```
int? length = customers?.Length;  
Customer first = customers?[0];  
int? count = customers?[0]?.Orders?.Count();
```

```
Integer value1 = null;  
Optional<Integer> a = Optional.ofNullable(value1);  
Integer value2 = a.orElse(new Integer(0));
```

# Removing null check

```
// let's assume you will get this from your model in the future; in the meantime...
Optional<Project> optionalProject = Optional.ofNullable(project);

// safe, java 8, but still ugly and omission-prone
if (optionalProject.isPresent()) {

    ApplicationType applicationType = optionalProject.get().getApplicationType();
    Optional<ApplicationType> optionalApplicationType = Optional.ofNullable(app licationType);
    if (optionalApplicationType.isPresent()) {
        String typeDirName = optionalApplicationType.get().getTypeDirName();
        Optional<String> optionalTypeDirName = Optional.ofNullable(typeDirName);
        if (optionalTypeDirName.isPresent()) {
            System.out.println(optionalTypeDirName);
        }
    }
}
```

# Removing null check

```
// let's assume you will get this from your model in the future; in the meantime...
Optional<Project> optionalProject = Optional.ofNullable(project);

// safe, java 8, but still ugly and omission-prone
if (optionalProject.isPresent()) {

    ApplicationType applicationType = optionalProject.get().getApplicationType();
    Optional<ApplicationType> optionalApplicationType = Optional.ofNullable(app licationType);
    if (optionalApplicationType.isPresent()) {
        String typeDirName = optionalApplicationType.get().getTypeDirName();
        Optional<String> optionalTypeDirName = Optional.ofNullable(typeDirName);
        if (optionalTypeDirName.isPresent()) {
            System.out.println(optionalTypeDirName);
        }
    }
}
```

# Removing null check

```
// safe, yet prettier

optionalProject

.flatMap(Project::getApplicationTypeOptional)

.flatMap(ApplicationType::getTypeDirNameOptional)

.ifPresent(System.out::println);
```



```
_currentUser

public WebUser getCurrentUser(){
    if (_currentUser == null) {
        Object obj = HttpContext.Current.Session["__currentUser"];
        if (obj != null) {
            _currentUser = (WebUser)obj;
            return _currentUser;
        }
        SecurityHelper secHelper = new SecurityHelper();
        WebUserRepository rep = new WebUserRepository();
        if (secHelper.TrackingGuid != Guid.Empty){
            _currentUser = rep.GetWebUserByTrackingGuid(secHelper.TrackingGuid);
            if (_currentUser != null)
                return _currentUser;
        }
        WebUserFactory factory = new WebUserFactory();
        _currentUser = factory.CreateWebUser();
    }
    return _currentUser;
}
```

# Avoid multiple returns

```
private WebUser _currentUser;
public WebUser getCurrentUser()
{
    if (_currentUser == null) _currentUser = GetWebUserFromSession();
    if (_currentUser == null) _currentUser = GetWebUserFromTrackingCookie();
    if (_currentUser == null) _currentUser = CreateNewWebUser();
    return _currentUser;
}
private WebUser GetWebUserFromSession()
{
    Object obj = HttpContext.Current.Session["__currentUser"];
    return obj == null ? null : (WebUser)obj;
}
private WebUser GetWebUserFromTrackingCookie()
{
    SecurityHelper secHelper = new SecurityHelper();
    WebUserRepository rep = new WebUserRepository();
    if (secHelper.TrackingGuid == Guid.Empty)
        return null;
    else
        return rep.GetWebUserByTrackingGuid(secHelper.TrackingGuid);
}
private WebUser CreateNewWebUser()
{
    WebUserFactory factory = new WebUserFactory();
    return factory.CreateWebUser();
}
```



```
double getPayAmount() {  
    double result;  
    if (_isDead) result = deadAmount();  
    else {  
        if (_isSeparated) result = separatedAmount();  
        else {  
            if (_isRetired) result = retiredAmount();  
            else  
                result = normalPayAmount();  
        }  
    }  
    return result;  
};
```



```
double getPayAmount() {  
    double result;  
    if (_isDead) result = deadAmount();  
    else if (_isRetired) result = retiredAmount();  
    else if (_isSeparated) result = separatedAmount();  
    else result = normalPayAmount();  
    return result;  
};
```



```
double getPayAmount() {  
    double result;  
    if (_isDead) { result = deadAmount(); }  
    else if (_isRetired) { result = retiredAmount(); }  
    else if (_isSeparated) { result = separatedAmount(); }  
    else { result = normalPayAmount(); }  
    return result;  
};
```

# Replace Nested Conditional with Guard Clauses

```
double getPayAmount() {  
    if (_isDead) return deadAmount();  
    if (_isSeparated) return separatedAmount();  
    if (_isRetired) return retiredAmount();  
    return normalPayAmount();  
};
```



```
List<Integer> stringsToInts(List<String> strings)
{
    if (strings != null) {
        List<Integer> integers = new ArrayList<>();
        for (String s : strings) {
            integers.add(Integer.parseInt(s));
        }
        return integers;
    } else {
        return null;
    }
}
```

# use Early returns

```
public List<Integer> stringsToInts2(
    List<String> strings) {
    if (strings == null) {
        return null;
    }
    List<Integer> integers = new ArrayList<>();
    for (String s : strings) {
        integers.add(Integer.parseInt(s));
    }
    return integers;
}
```

- Reduces the number of indentations

# Remove nulls

```
List<Integer> stringsToInts2(List<String> strings) {  
  
    List<Integer> integers = new ArrayList<>();  
    for (String s : strings) {  
        integers.add(Integer.parseInt(s));  
    }  
    return integers;  
}
```

# Remove nulls

```
List<Integer> stringsToInts2(List<String> strings) {  
  
    List<Integer> integers = new ArrayList<>();  
    for (String s : strings) {  
        integers.add(Parse(s));  
    }  
    return integers;  
}  
  
int Parse(string s){  
    Try...{ Integer.parseInt ... }  
}
```

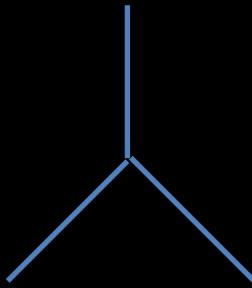
```
if (rowCount > rowIdx) {
    if(drc[rowIdx].Table.Columns.Contains("avalld")) {
        do {
            if (Attributes[attrVal.AttributeClassId] == null) {
                // do stuff
            }
        } else {
            if (!(Attributes[attrVal.AttributeClassId] is ArrayList)) {
                // do stuff
            }
            else {
                if (!isChecking) {
                    // do stuff
                } else
                {
                    // do stuff
                }
            }
        }
        rowIdx++;
    } while(rowIdx < rowCount && GetIdAsInt32(drc[rowIdx]) == Id);
}
else rowIdx++;
}
return rowIdx;
```

# Cyclomatic Complexity

```
public void ProcessPages()
{
    while(nextPage !=true)
    {
        if((lineCount<=linesPerPage) && (status != Status.Cancelled)
&& (morePages == true))
        {
            //....
        }
    }
}
```

# Cyclomatic Complexity

```
public int getValue(int param1)
{
    int value = 0;
    if (param1 == 0)
    {
        value = 4;
    }
    else
    {
        value = 0;
    }
    return value;
}
```



# Cyclometric Complexity

<b>CC Value</b>	<b>Risk</b>
1-10	Low risk program
11-20	Moderate risk
21-50	High risk
>50	Most complex and highly unstable method

# Rules for a good fun

- » < 10 lines
- » Nesting depth <3
- » CC < 10
- » < 3 param
- » Should not return null, bool, int for error
- » No null, bool, optional, out param
- » SOC

Code  
Code

Code

Code  
Code

Code  
Code

Code

Code  
Code  
Code

Code  
Code

Code

Code  
Code  
Code  
Code  
Code  
Code  
Code  
Code  
Code  
Code  
Code  
Code  
Code  
Code  
Code  
Code  
Code  
Code



```
private int x;  
public int increment_and_return_x()  
{  
    x = x + 1;  
    return x;  
}
```

# Command / Query Separation

1. A **Query** doesn't have side effects  
and returns a result

e.g. `getState():State`, `isLoggedIn():boolean`,  
`findCustomer():Customer`



2. A **Command** changes the system state  
and **returns void**



e.g. `processOrder(Order)`, `consumeMessage(Message)`, `setName(String)`

▪ Avoid doing both in the same method

e.g. `processOrder(Order):boolean` (that returns `false` if it failed)  
(It forces the caller to check the result right away. Prefer **exceptions** instead!)

```
private int x;
public int value()
{
    return x;
}
void increment()
{
    x = x + 1;
}
```

```
string result = _storeService.PurchaseItem(buyer, item);
```

# Summary

- » Avoid magic numbers/strings
- » Use constants / enum
- » Avoid God Class
- » Control of depth of nesting
- » Control fun size
- » Null Pattern
- » Avoid null, optional, boolean, out parameters
- »

# SRP / SOC (\*\*\*)

- » Domain logic and Domain Rules
- » Boundary and Domain
- » Domain logic and error handling logic
- » Flow and action(step)
- »

- » Max 4 parameters
- » Immutable parameter
- » No bool param
- » No optional param
- » No Out param (return)
- » No nullable param
- » No null return (exception, null pattern)
- » No bool return for error (exception)
- » No int error code return (exception)

# Data Structure



```
Map sensors = new HashMap();
```

```
...
```

```
Sensor s = (Sensor)sensors.get(sensorId );
```



```
Map<int, Sensor> sensors = new HashMap<int, Sensor>();  
...  
Sensor s = sensors.get(sensorId );
```

# Hide boundary (Map)

```
public class Sensors
{
    private Map sensors = new HashMap();

    public Sensor getById(String id)
    {
        return (Sensor) sensors.get(id);
    }
    //snip
}
```

Code at the boundaries needs clear separation

# Encapsulate Collection





```
private ArrayList fieldDetails = new ArrayList();  
private HashMap linkedFields = new HashMap();  
  
private void setTransactionParams(Hashtable params) {  
    ...  
}
```

# Use collection framework interfaces, use generics

```
private List<Field> fieldDetails = new ArrayList<>();  
private Map<String,FieldProperties> linkedFields = new  
    HashMap<>();  
  
private void setTransactionParams(Map<String, Object> params) {  
    ...  
}
```



```
private void setParameters(HashMap<String, String> parameters) {  
}
```

# Use interfaces instead of classes

```
private void setParameters(Map<String, String> parameters) {  
}
```

# basics



```
private List getTableColumns(Map tableFieldListMap) {  
    Set tables = tableFieldListMap.keySet();  
    Iterator iterator = tables.iterator();  
    while(iterator.hasNext()) {  
        String tableName = (String)iterator.next();  
        ...  
    }  
}
```

# Use generics, for..each loop

```
private List<String> getTableColumns (Map<String,Field>
    tableFieldListMap) {
    for (String tableName : tableFieldListMap.keySet ()) {
        ...
    }
    ...
}
```



```
private static final String _ALPHABETS = "ALPHABETS";
private static final int _ALPHABETS_NO = 1;

private int getSwitchId(String value) {
    if(value.equals(_ALPHABETS)) {
        return _ALPHABETS_NO;
    }
    ...
}

int switchId = getSwitchId(data);
switch(switchId) {
    case _ALPHABETS_NO:
        ...
}
```

# Use enum

```
private enum DataType {  
    ALPHABETS, ALPHA_NUMERIC, NUMERIC, SWIFT, ALL, DATE, TIME  
}  
  
switch(DataType.valueOf(data)) {  
    case ALPHABETS:  
        ...  
}
```



```
public String getUpdateQuery() {  
    StringBuffer updateQuery = new StringBuffer();  
    ...  
    return updateQuery.toString();  
}
```

# Use StringBuilder for local variables

```
public String getUpdateQuery() {  
    StringBuilder updateQuery = new StringBuilder();  
    ...  
    return updateQuery.toString();  
}
```



```
strName = TextBox1.Text;  
iAge = int.parse(TextBox2.Text);
```

# Avoid Type Embedded in Name

```
name = TextBox1.Text;  
age = int.parse(TextBox2.Text);
```

Avoid placing types in method names; it's not only redundant, but it forces you to change the name if the type changes.



```
void Clean(int id)
{
    Person object = myCollection.get(id);
    boolean present = myCollection.remove(object);
}
```

# Don't preserve return values you don't use

```
void Clean(int id)
{
    Person object = myCollection.get(id);
    myCollection.remove(object);
}
```





```
public class ServiceLocator {  
    Map<String, Object> cache;  
  
    public ServiceLocator() {  
        cache = Collections.synchronizedMap(  
            new HashMap<String, Object>());  
    }  
}
```

# ConcurrentHashMap has better performance than Collections.synchronizedMap

```
public class ServiceLocator {  
    ConcurrentHashMap<String, Object> cache;  
  
    public ServiceLocator() {  
        cache = new ConcurrentHashMap<String, Object>();  
    }  
}
```



```
Map<String, String> parameters = new Hashtable<>();
```

# ConcurrentHashMap has better performance than Hashtable

```
Map<String, String> parameters = new ConcurrentHashMap<>();
```



```
if (condition)
    do something
else
    do something else
```

# Use curly braces for if..else blocks

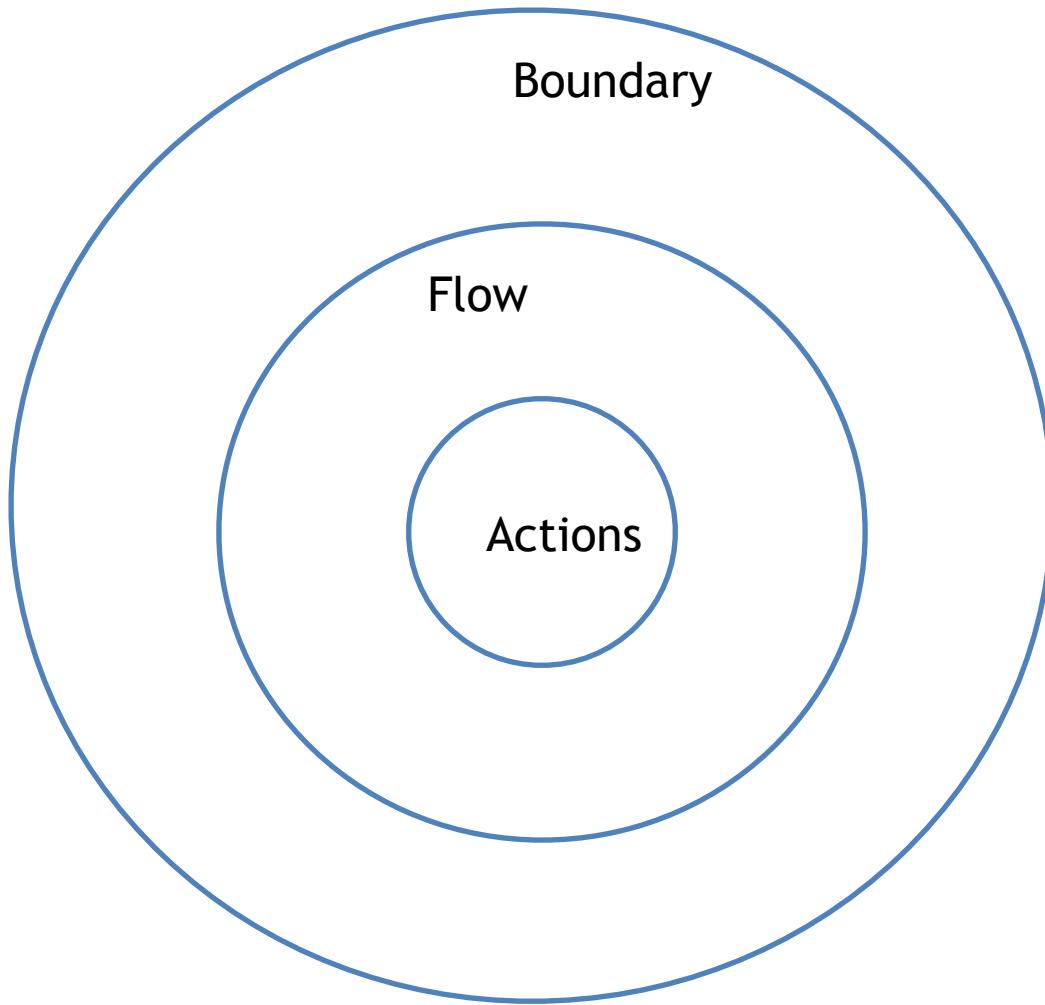
```
if (condition) {  
    //do something  
}  
  
else {  
    //do something else  
}
```



```
public String getColumns(List<Field> fields) {  
    String column = "";  
    StringBuilder columns = new StringBuilder();  
    for(Field field : fields) {  
        column = field.getColumn();  
        columns.append(column);  
        columns.append(",");  
    }  
    return columns.toString();  
}
```

# Declare variable only where required

```
public String getColumns(List<Field> fields) {  
    StringBuilder columns = new StringBuilder();  
    for(Field field : fields) {  
        String column = field.getColumn();  
        columns.append(column);  
        columns.append(",");  
    }  
    return columns.toString();  
}
```



API Layer

Application Feature

Domain Layer

Repository

DB

Rule

Business Logic

- » [GitHub.com/getmubarak/cleancode](https://GitHub.com/getmubarak/cleancode)
- » elegant object

```
size_t increment(size_t idx) const
{
    return (idx + 1) % Capacity;
}
```

## % modulus operator

The `%` modulus operator can be confusing if not encountered before. The `% operator` gives the remainder from integer division. It can be used in a range of ways and here it is used to calculate when the index should be wrapped back to the beginning in a *circular* way.

[Hide](#) [Copy Code](#)

```
// Example: A CircularFifo with Size of 99
// Capacity is Size+1. The padding of +1 is used to simplify the logic
increment(98) ---> will yield 99
increment(99) ---> will yield 0
}
```