Write Clean Functions

Functions

- Reduce complexity
- Avoid code-duplication
- Improve maintainability
- Hide implementation details

Functions

- Size matters ! Small functions!
  - The first rule of functions is that they should be small. The second rule of functions is that they should be smaller than that.
  - Functions should hardly ever be 20 lines long.
    - Why ?
  - Blocks within if statements, else statements, while statements, and so on should be one line long.
    - How ?

Functions

- Do one thing !
  - Functions should do one thing – what their name says and nothing more or less.
  - They should do it well.
  - They should do it only.
  - What is ONE THING?

Functions

- Have no side effects

Functions

- Examples:
  - Sum elements
  - Triangle area by the three sides
  - Invoke something on a hash table value

Functions

- What if a function name is incorrect?
  - The function can do too many things
  - Can have bad cohesion, doing too many things
  - Can have side effects and that leads to spaghetti
  - Can return strange result instead of error

Functions

- Switch statements are always long

```
public Money calculatePay(Employee e)
throws InvalidEmployeeType {
    switch (e.type) {
      case COMMISSIONED:
        return calculateCommissionedPay(e);
      case HOURLY:
        return calculateHourlyPay(e);
      case SALARIED:
        return calculateSalariedPay(e);
      default:
        throw new InvalidEmployeeType(e.type);
    }
  }
```

Functions

- switch statements are tolerated if:
  - they appear only once
  - are used to create polymorphic objects,
  - are hidden behind an inheritance

  - **DEMO**

Functions

- Arguments
  - the ideal number of arguments for a function is zero
  - next comes one
  - followed closely by two
  - three arguments should be avoided where possible
  - more than three requires very special justification

Functions

- Arguments
  - hard to understand
  - hard to test

Functions

- Arguments Rules
  - Put the most important arguments first
    - registerUser
  - Non-important or optional – last
  - Do not modify arguments

Functions

- Cohesion
  - Functional – return a result entirely on input based calculations - sqrt
  - Sequential – performs a set of steps = algorithm – sendMail
  - Communicational – process some data and return a result – generateExpenseReport(employeeId)
  - Temporal – not related operations that need to happen together at a given time - initializeApplication
  - Logical – performs different operation depending on a given parameter
  - Coincidental – not related operations grouped without strong reason

Functions

- Coupling
  - What tight coupling means???
  - What is ideal coupling?? – only arguments are your dependencies

Functions

- Coupling

```
class Manager {
    void updateState();
    void initializeState();
    void sortAccounts();
}
```

Functions

- Coupling
  - How you will implement sum(a, b) and make it tightly coupled?

Functions

- Coupling
  - Parameters coupling
  - Class fields coupling
  - Static methods or constant coupling
  - Static variables coupling
  - Defining public method
  - Member functions taking member variables as arguments

Functions

- One level of abstraction per function
  - In order to make sure our functions are doing “one thing,” we need to make sure that the statements within our function are all at the same level of abstraction.
  - getHtml();
  - String pagePathName = PathParser.render(pagePath);
  - … .append("\n").
    - Are these the same level of abstraction or not?

Functions

- Cohesion and coupling
  - Layers in some system should not use functions in both directions – Data Layer to Service Layer and vice-versa

Functions

- Reduce coupling – with OOP – abstraction and encapsulation

Functions

- Command query separation
  - Every function should be a command (change state) or query (return state, transformed)

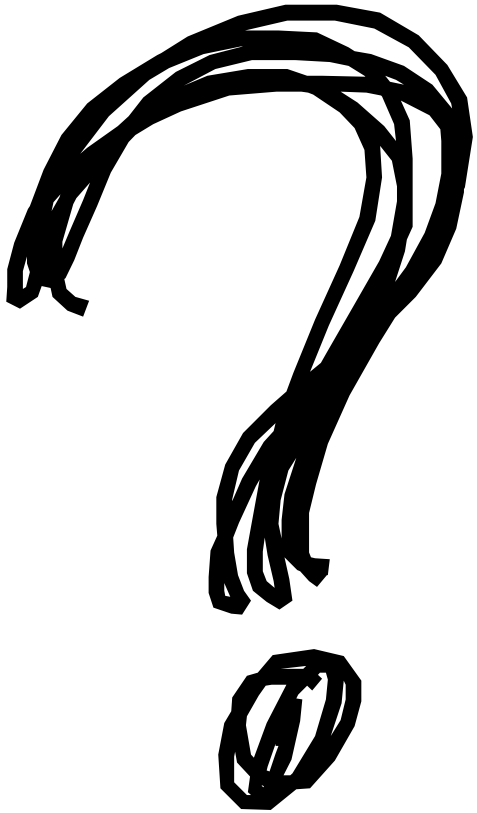
Functions

- Prefer exceptions to returning error codes

Functions

- Don’t repeat yourself


СВ. КЛИМЕНТ ОХРИДСКИ



THANK YOU