03 - Functions

Como fazer uma função ser fácil de entender?

- Funções devem ser **bem pequenas.**
- 20 linha já é bem alto.
    - Ideal é até umas 4 linhas
- If statemente devem ser simples
    - Sem ifs aninhados.
    - No máximo duas indentações no código
- Funções devem fazer **uma coisa apenas**
    - Funções servem pra quebrar operações maiores, afinal
- Um nível de abstração por função
    - e.g. não juntar algo abstrato como "getHtml" com funções básicas como ".append('\n')"
- Leitura de funções deve ser fluida e um passo levar ao outro intuitivamente.
  SWITCH
- Switch statements requerem por definição várias linhas e fazem várias coisas ao mesmo tempo.
- Ideal é que fique escondido dentro de uma abstract factory.
    - Deve aparecer só uma vez
    - Deve criar objetos polimórficos
    - Escondidos atrás de inheritance
    - Exemplo:

Employee and Factory

```
public abstract class Employee {
  public abstract boolean isPayday();
  public abstract Money calculatePay();
  public abstract void deliverPay(Money pay);
}
----------------
public interface EmployeeFactory {
  public Employee makeEmployee(EmployeeRecord r) throws InvalidEmployeeType;
}
----------------
public class EmployeeFactoryImpl implements EmployeeFactory {
  public Employee makeEmployee(EmployeeRecord r) throws InvalidEmployeeType {
    switch (r.type) {
      case COMMISSIONED:
        return new CommissionedEmployee(r) ;
      case HOURLY:
        return new HourlyEmployee(r);
      case SALARIED:
        return new SalariedEmploye(r);
      default:
        throw new InvalidEmployeeType(r.type);
    }
  }
}
```

NOMES

- Usar nomes bem descritivos para as funções.
  - Mesmo que o nome seja longo.
  - Escolher bons nomes pode requerer tempo, esforço e retentativas.
  - Ser consistente com nomes, não misturar termos equivalentes.

ARGUMENTOS

- Quantos menos argumentos, melhor.
  - Zero é o ideal (Niládico).
    - Seguido de Monádico (1) , Diádico (2)
  - Três é o limite, não é ideal. (Triádico)
  - Evitar funções poliádicas.
- Mais argumentos também dificulta testar as funções.
  - Mais combinações de input possíveis.
- É melhor evitar modificação do input in place (output argument).
  - A ação da função deve ser apenas pelo return value, não pela modificação do input.

MONÁDICOS

- Formas monádicas em relação ao argumento podem ser:
  - Extração
  - Transformação
  - Geração de eventos -- esse aqui deixar bem claro quando é o caso.
  - Melhor evitar outras formas.
- Argumentos flag (boolean) são uma péssima ideia.
  - Indica que uma função faz mais de uma coisa.
  - Melhor dividir em duas funções, uma para cada caso.

DIÁDICOS

- Dois argumentos são mais difíceis de entender do que um.
- Mas em alguns casos são naturais.
  - e.g. definir um ponto num plano cartesiano, x e y.

        - São duas informações de um mesmo dado.
    - Os dois argumentos devem ser coerentes entre si.

TRIÁDICO

- Bem mais difíceis de entender.
- Só devem ser criados quando são realmente necessários.
- Pode ser útil agrupar argumentos em um único objeto específico e usar esse objeto como argumento.
- Funções que recebem número variável de argumentos (e.g. print) são equivalentes a monádico, diádico ou triádico, é como se fosse uma lista só. Não são um problema em si.

- Funções monádicas costumam ter nome da forma <verbo>_<substantivo>.

SIDE EFFECTS

- Função não deve ter side-effects
    - i.e. ação fora do return value.
    - Gera ofuscação
- Também gera acoplamento temporal entre duas ações.
    - Se houver acoplamento temporal, o nome deve descrever isso
- Output arguments são desnecessários hoje.
    - Se uma função for mudar o estado de algo, que seja o do objeto que detém o método (self).

COMMAND QUERY SEPARATION

- Função deve fazer algo ou responder algo, nunca ambos.
- Por exemplo, função que faz algo e retorna um boolean de sucesso é ruim:
- Melhor usar a alterantiva mais clara e separada:
- É melhor usar exceptions do que códigos de erro por causa disso.
    - Error codes geram problema de dependências.
    - Corpo do Try / except devem ser funções mínimas por si só, para separar ações de lidar com erro e ação de fazer a ação esperada.
        - Para não violar o princípio de uma coisa por função.
        - Função não deve ter nada além do try /except / finally

|
|

```
public boolean set(String attribute, String value)
```

```
<br>if (attributeExists("username")) { <br>setAttribute("username",
"unclebob"); <br>... <br>}<br>
```

- Evitar duplicação
- Diminui lugares que requerem reescrita em caso de alteração.
- Análogo a normalização de banco de dados
- Muito importante!
- Toda função deve ter apenas uma entrada e uma saída.

    - Apenas um único RETURN
    - Sem BREAK, CONTINUE em loops
    - Sem GOTO

- Mas como as funções já são pequenas, tudo bem quebrar um pouco essas regras de vez em quando. (Exceto o GOTO)

- Não é trivial escrever código assim e requer treino.

    - Melhora deve ser um processo iterativo de refino.
    - Dificilmente alguém consegue escrever com essas regras desde o início.

DON'T REPEAT YOURSELF

ESTRUTURAÇÃO