SmartSampa

Estudo de caso sobre o
desenvolvimento de código limpo

Sumário

Introdução

- Aplicação de princípios, padrões e boas práticas de desenvolvimento de software num projeto real.

- Construção de um serviço web que integre as diferentes fontes de dados sobre o transporte público.

Dados

API Olho Vivo

- Tempo real
- Linhas (Route)
- Viagens (Trip)
- Pontos de ônibus (Stop)
- Predições
- Corredores

GTFS

- Arquivo .zip
- Linhas (Route)
- Viagens (Trip)
- Pontos de ônibus (Stop)
- Desenho do traçado (Shape)
- Frequências de saída
- Tabela de horários

Boas práticas

Nome do domínio problema/solução

Procure utilizar nomes do espaço problema/solução em variáveis, classes, métodos, etc.

- Termos do espaço problema:
  - nomes do problema real a ser resolvido.
  - Ex: Trip, Stop, Shape
  - Ex. de uso: classes Trip, Stop, OlhovivoBus . . .
- Termos do espaço solução:
  - nomes de algoritmos, padrões e outros termos familiares a programadores.
  - Ex: DepthFirstSearch, Adapter, NullObject
  - Ex. de uso: classes GtfsTripAdapter, NullStop . . .

Evite o mapeamento mental

O leitor é obrigado a mapear mentalmente o nome de uma variável/método a sua função dentro do código.

- Ocorre devido a nomes pouco explicativos.
- Nomes longos e explicativos são preferidos a nomes curtos e misteriosos.
- Variáveis pouco explicativas podem ser usadas em escopos pequenos.
    - Ex: o contador “i” dentro de um laço curto.

Função do código é explicar o que ele faz!

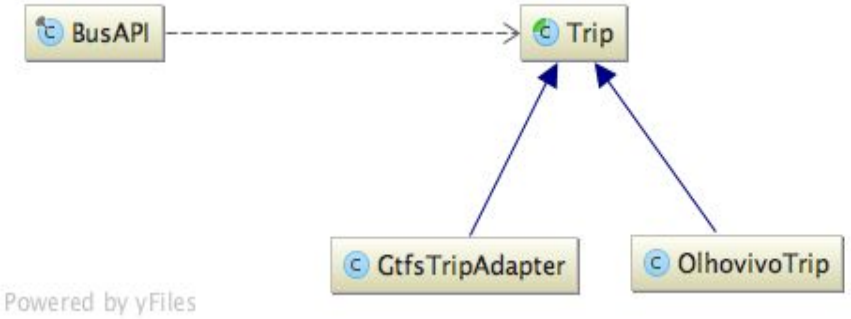
Princípios

Princípio da responsabilidade única

- Classes devem ter apenas uma responsabilidade.
- Responsabilidade implica em possíveis mudanças de implementação.
- Classes com mais de uma responsabilidade são mais frágeis pois mudanças em uma de suas atribuições podem comprometer o funcionamento de outras.

Princípio da inversão de dependência

- Módulos de alto nível não devem depender de módulos de baixo nível.
- Abstrações não devem depender da implementação, mas o inverso deve ocorrer.
- Comportamento definido de forma abstrata.
- Implementação pode variar.

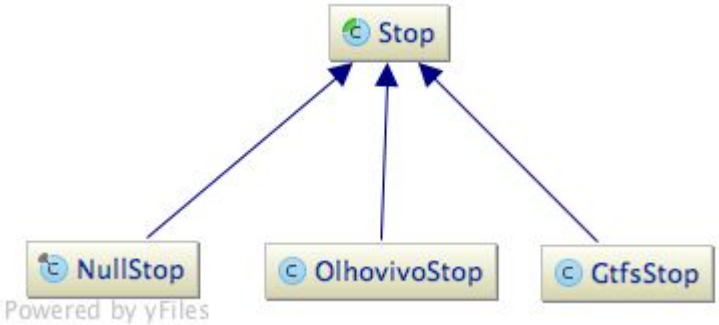


Padrões

Null Object

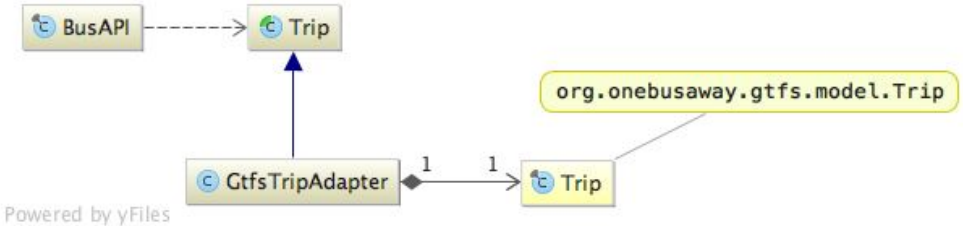
- Cria um substituto do *null* para um determinado tipo.
- Evita problemas com *NullPointerException*.
- Evita blocos *try/catch*.

```
public Stop getStop(Stop equivalentStop) {
        return getAllStops()
                .stream()
                .filter(equivalentStop::equals)
                .findAny()
                .orElse(Stop.emptyStop());
}
```



Adapter

- Traduz uma interface antiga para uma interface nova.
- Permite a reutilização de componentes antigos ou de terceiros.



Referências

1. Kent Beck. *Implementation patterns*. Pearson Education, 2007.
2. Robert C. Martin. *Clean code: a handbook of agile software craftsmanship.* Pearson Education, 2009.
3. Robert C. Martin. *Agile software development: principles, patterns, and practices*. Prentice Hall PTR, 2003.