

A decorative graphic in the top-left corner consisting of a network of interconnected nodes and lines. Some nodes are highlighted with blue circles or dots.

# Strategy

A decorative graphic in the bottom-right corner, similar to the one in the top-left, featuring a network of nodes and lines with some blue highlights.

A decorative network diagram at the top of the slide, featuring a series of interconnected nodes and lines. A central node is highlighted with a dashed circle and a blue double quote symbol.

“

*Zapouzdřuje rodinu algoritmů,  
aby byly*  
***vzájemně zaměnitelné***



## Problém

- ◎ navigační systém nad mapovým podkladem
  - nejrychlejší cesta na zadanou adresu
- ◎ 1. cesty podél silnic
- ◎ 2. pěší zóny
- ◎ 3. veřejná doprava
- ◎ ... další možnosti a algoritmy

Hlavní třída se bude zvětšovat ~→ neudržitelná



## Řešení - naivní

- ⦿ algoritmy uzavřu do metod
- ⦿ if-else hell


```
public Output DisplayPath(string path, string address) {  
    if (path == "roadPath") {  
        return SearchPathOnRoads(address);  
    }  
    else if (path == "walkPath") {  
        return SearchPathOnWalkPaths(address);  
    }  
    else if (path == "walkPath" && withPublicTransport) {  
        if (useMetro) return SearchPathOnWPAndPT(address, true);  
        else return SearchPathOnWPAndPT(address, false);  
    }  
    throw new Exception("feature is not supported");  
}
```



## ☹️ **Řešení - naivní**

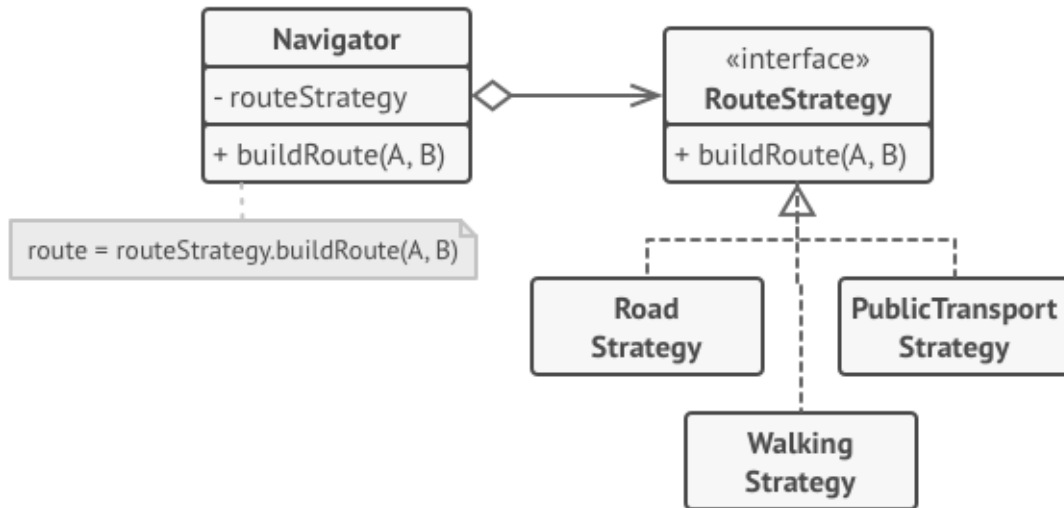
- ⦿ kód komplexní, špatně čitelný
- ⦿ těžko spravovatelný a složitě rozšiřitelný

## 😊 **Řešení - Strategy**

- ⦿ encapsulace algoritmů do tříd = strategie
  - ⦿ jednotné rozhraní pro algoritmy
  - ⦿ hlavní třída referencuje aktuální strategii
- 

## 😊 Řešení - Strategy

- 🎯 Hlavní třída: **Navigator**
- 🎯 Interface: **RouteStrategy** s metodou na spuštění strategie **buildRoute**



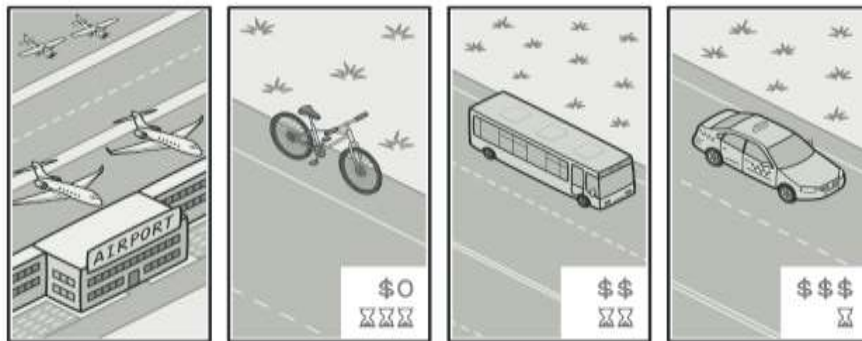
Strategie lze  
snadno přidat  
nebo upravit

# **Přirovnání**

**Problém:** Potřebuji se dostat na letiště

**Přepravní strategie:**

- Autobus
- Auto
- Kolo



Výběr strategie dle parametrů - čas a cena

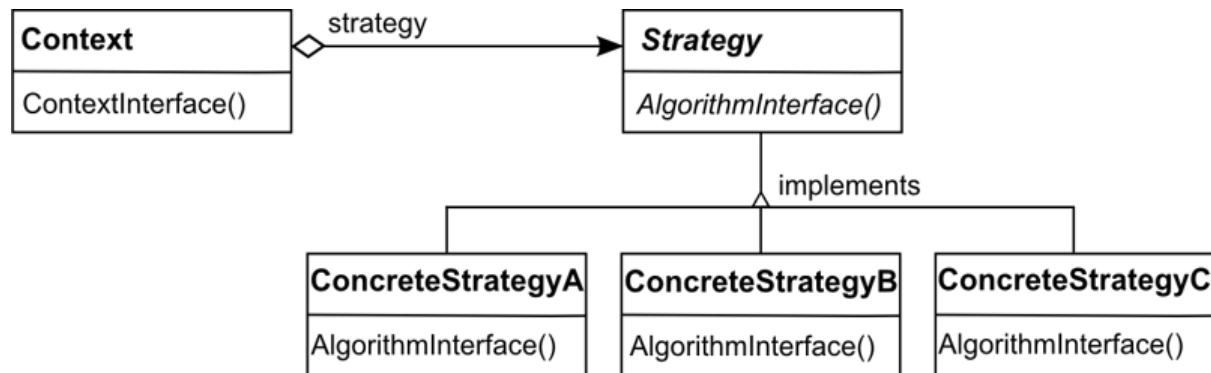
# Struktura

## Context

- reference na konkrétní algoritmus
- vykonává strategii, ale neví kterou

## Strategy

- společné rozhraní



## ConcreteStrategy

- implementace konkrétního algoritmu





# Implementace

```
interface Strategy {  
    Result method execute(Input input);  
}
```

```
class ConcreteStrategy implements Strategy {  
    Result method execute(Input input) {  
        var result = processInput(input);  
        return result;  
    }  
}
```

```
class Context {  
    private Strategy strategy;  
  
    public Context(Strategy strategy) => setStrategy(strategy);  
    method setStrategy(Strategy strategy) => this.strategy = strategy;  
  
    Result method executeStrategy(Input input)  
        => return strategy.execute(input);  
}
```



# Implementace - default strategy, templates

```
class Context {  
    private Strategy strategy;  
  
    public Context(Strategy strategy = defaultStrategy)  
        => setStrategy(strategy);  
  
}
```

compile  
time !

```
template <class AStrategy>  
class Context {  
    void Operation {  
        theStrategy.DoAlgoritihm();  
    }  
    // ...  
private:  
    AStrategy theStrategy;  
};
```



## Kdy volit/nevolit Strategy?



- ⦿ volba algoritmu za běhu programu
- ⦿ izolace logiky od implementace
- ⦿ různé druhy stejného algoritmu
- ⦿ if-else hell



- ⦿ málo algoritmů
- ⦿ není nutné je za běhu měnit
- ⦿ overhead mezi Strategy a Context



## Výhody

- ⦿ izolace implementačních detailů od logiky
- ⦿ volba algoritmy za běhu programu
- ⦿ nahrazení dědičnosti kompozicí
  - více ortogonálních strategií
- ⦿ *Open/Closed Principle*:
  - otevřená pro rozšíření
  - uzavřená pro modifikaci



## Nevýhody

- ⦿ vzor může zkomplikovat kód
- ⦿ klient musí znát strategie a jejich rozdíly
- ⦿ moderní programovací jazyky obsahují podporu pro funkcionální typy
  - implementace algoritmů v anonymních funkcích nahrazujících Strategy  
~> overhead



# Možné aplikace

## ⦿ GUI

- validace formulářových prvků
- layout manager – různé algoritmy pro layout prvků

## ⦿ třídění

- třídící algoritmus dle velikosti a struktury dat

## ⦿ komprese dat

- různé metody

## ⦿ obecně

- nahrávání, ukládání a generování výsledků v různých formátech
- I/O (konzole, soubor, síť...)



# Vztahy s ostatními vzory

## **Command**

- ⊙ konvertuje operace do objektů

## **State**

- ⊙ chování určeno stavem
- ⊙ stav se může měnit implicitně, Strategii mění klient

## **Template method**

- ⊙ částečné kroky vs. celý algoritmus
- ⊙ funguje na základě dědičnosti, Strategie využívá kompozici