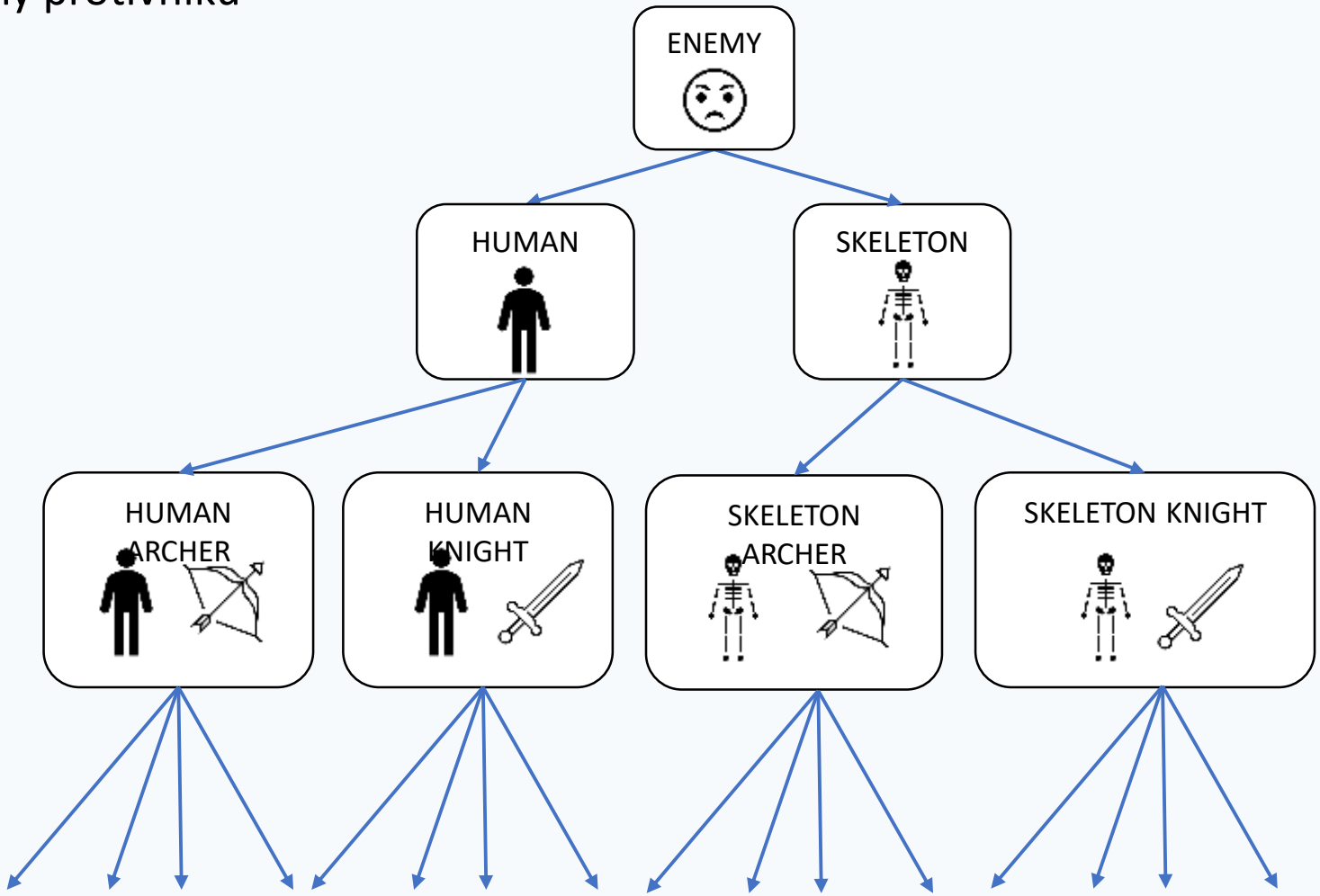
Bridge

Jan Šimek

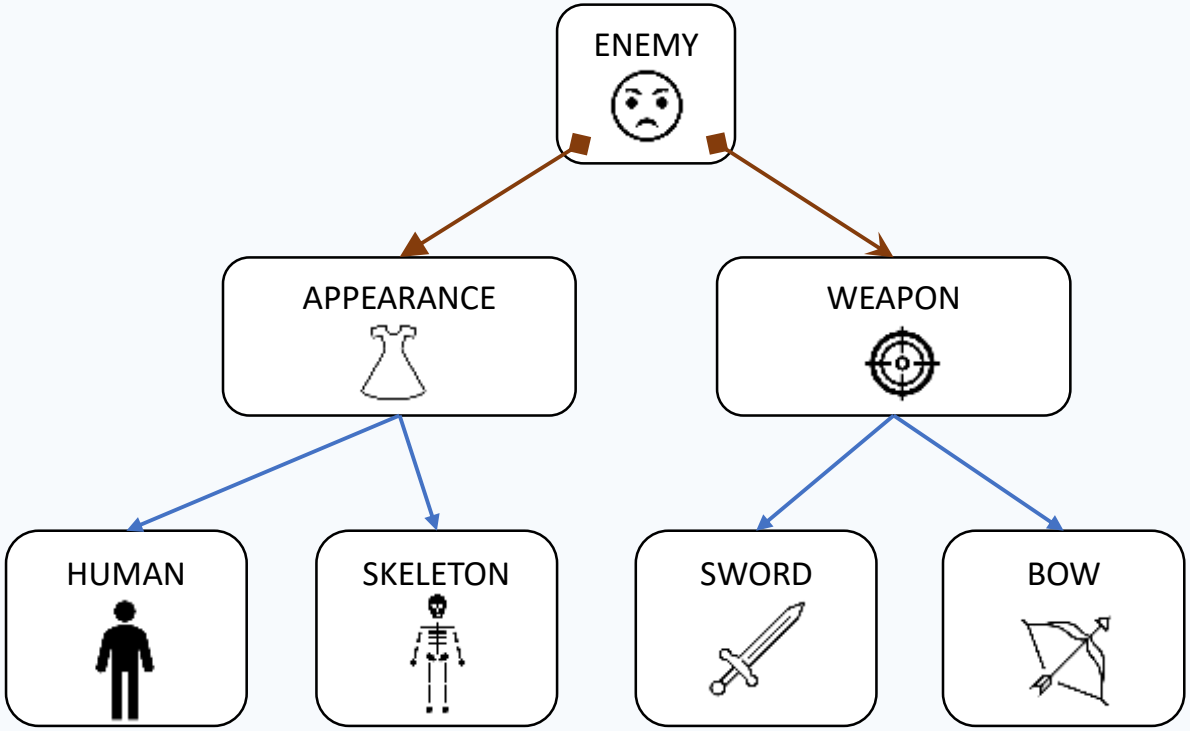
Motivace

- bojová hra
- různé druhy protivníků
- zbraně
- obtížnost
- brnění
- …



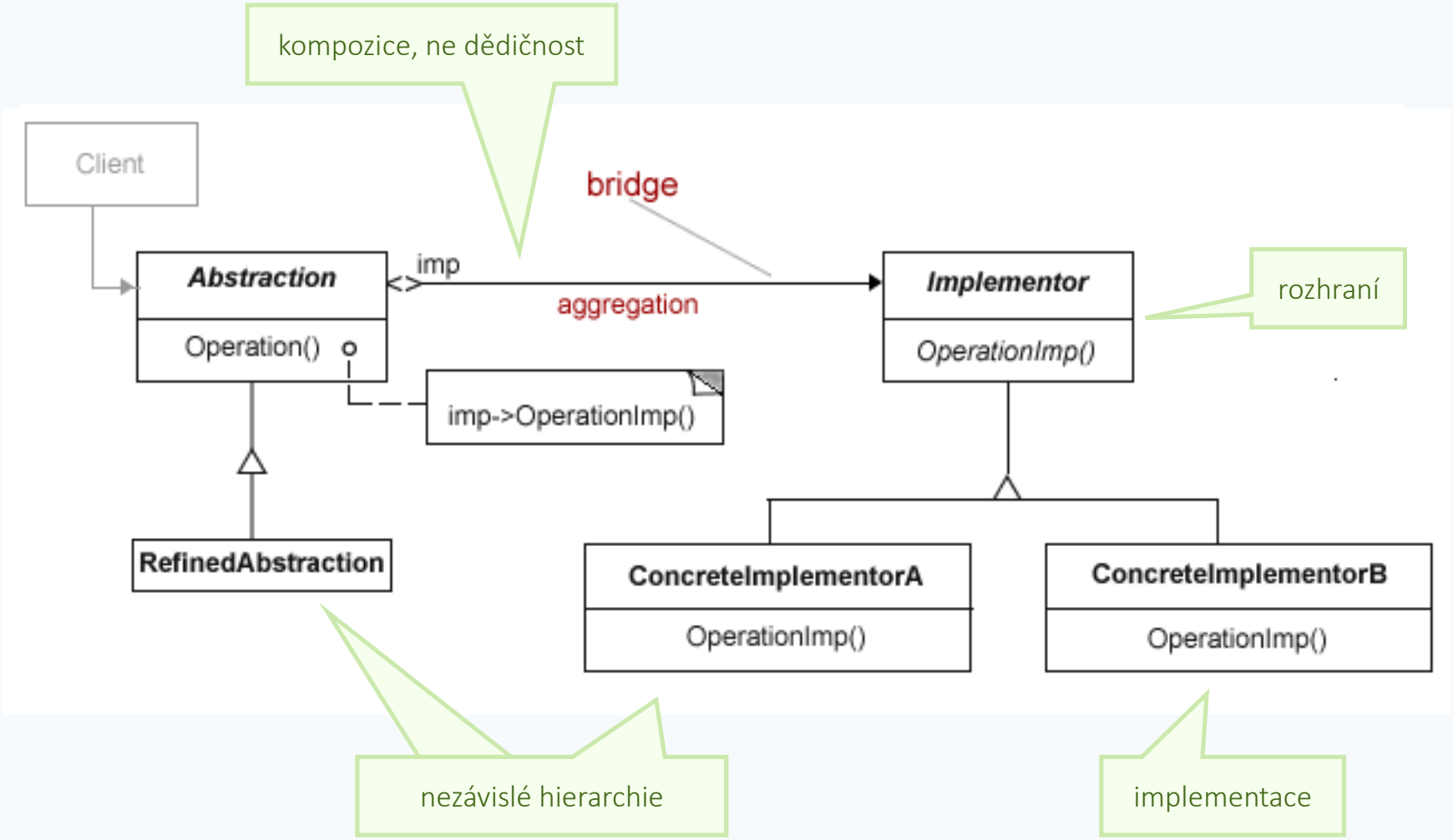
Motivace

- Lepší varianta
  - Oddělení konceptů
  - Možné pozdější změny, přidání dalších variant

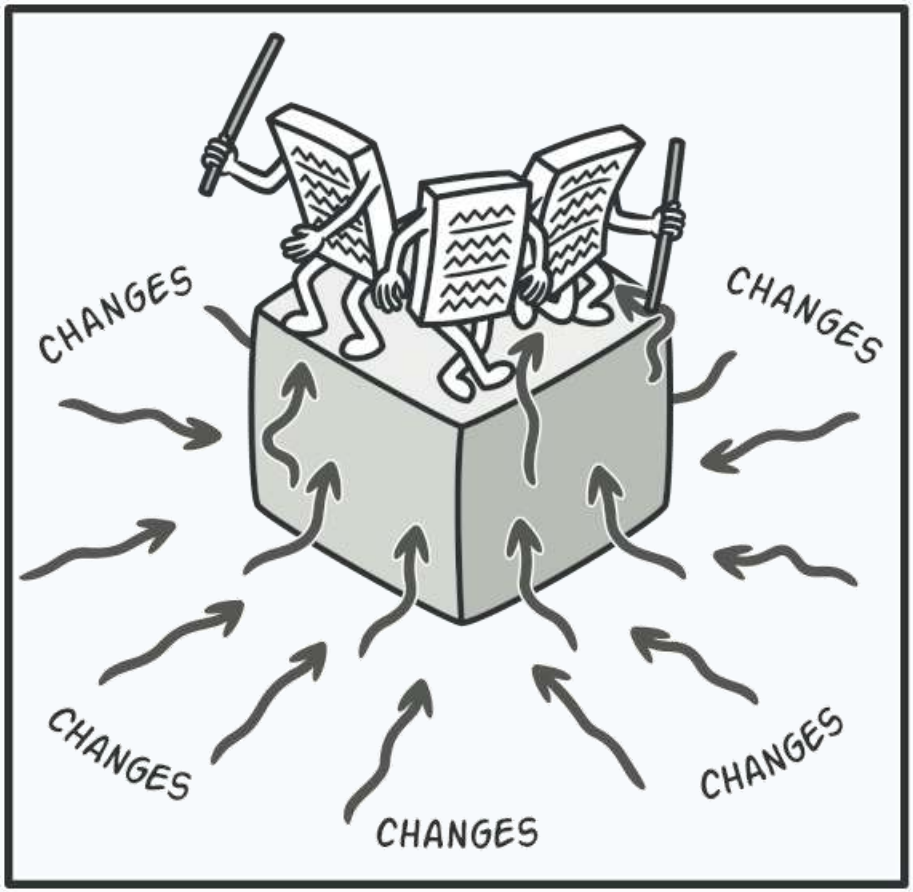


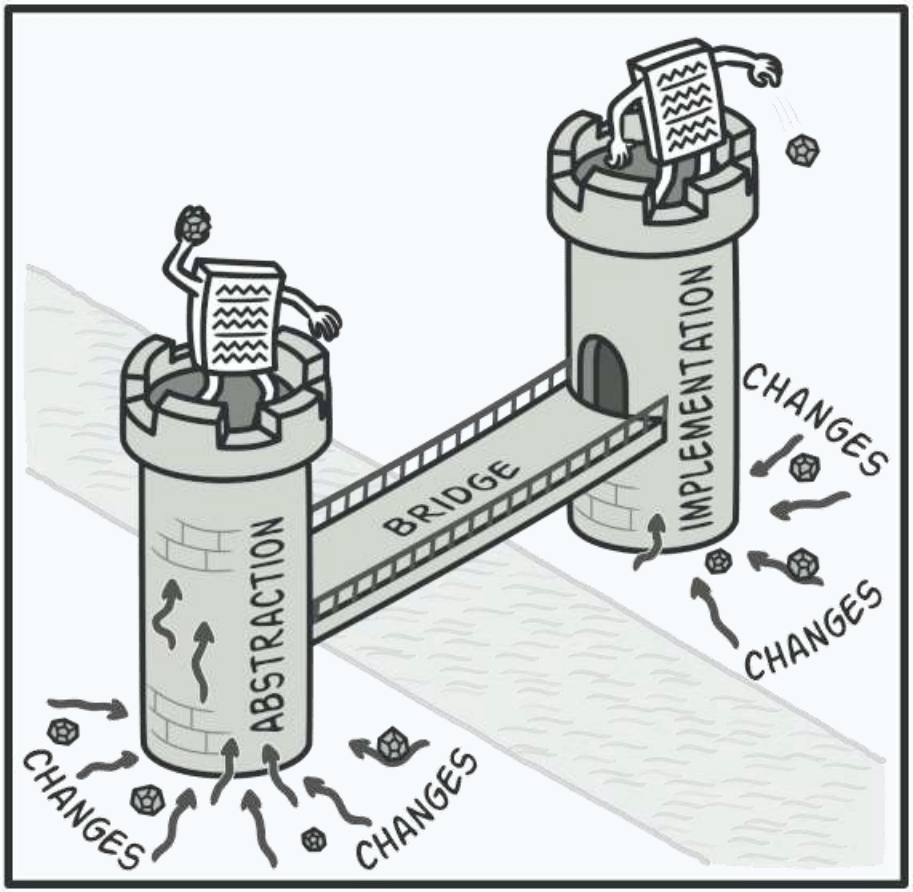
Struktura

oddělení polymorfní abstrakce od polymorfní implementace

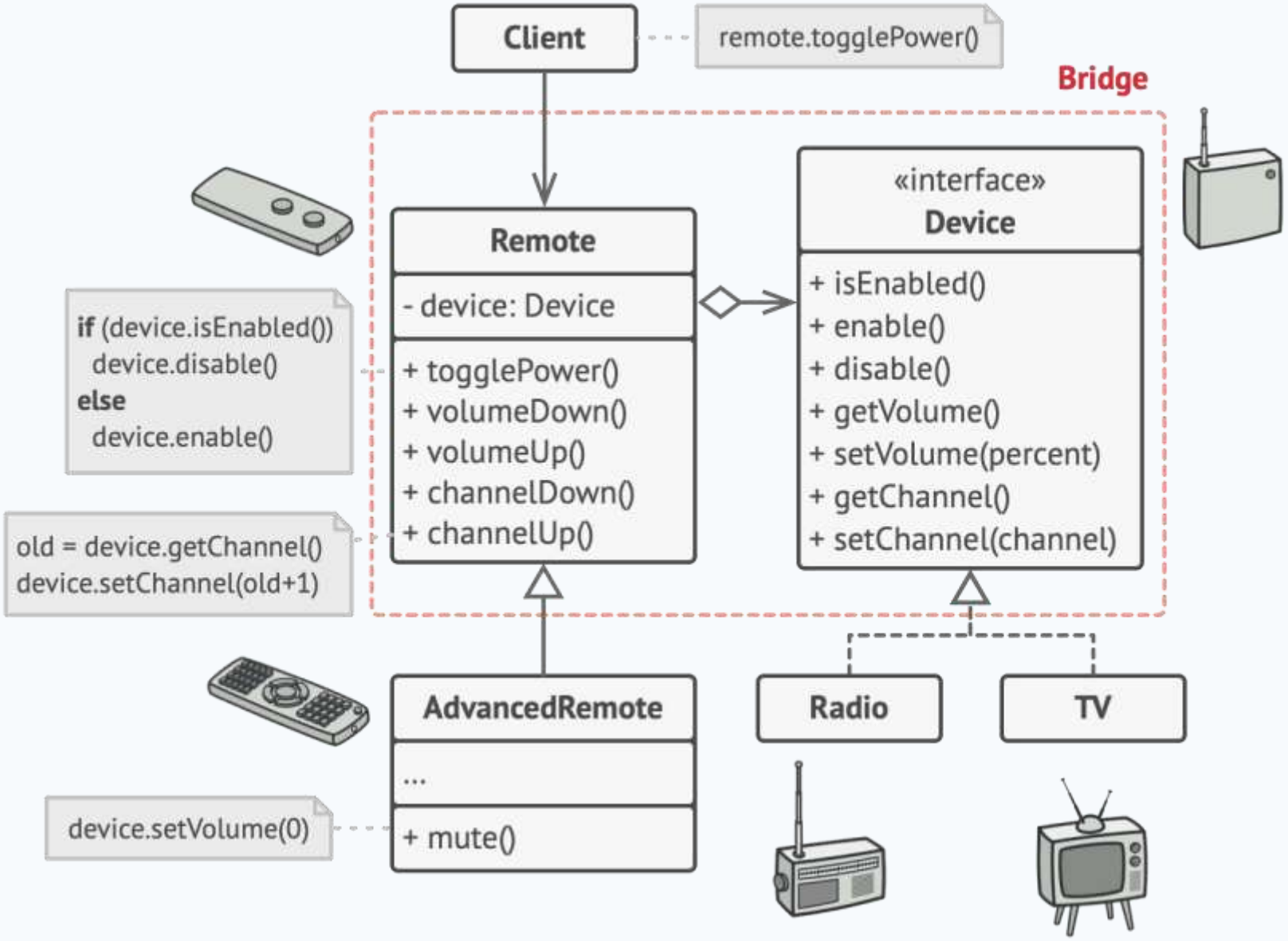


Motivace


CHANGES
CHANGES
CHANGES
CHANGES
CHANGES
CHANGES


ABSTRACTION
BRIDGE
IMPLEMENTATION
CHANGES
CHANGES
CHANGES
CHANGES
CHANGES

Příklad


Client
remote.togglePower()
Bridge
Remote
- device: Device
+ togglePower()
+ volumeDown()
+ volumeUp()
+ channelDown()
+ channelUp()
«interface»
Device
+ isEnabled()
+ enable()
+ disable()
+ getVolume()
+ setVolume(percent)
+ getChannel()
+ setChannel(channel)
if (device.isEnabled())
device.disable()
else
device.enable()
old = device.getChannel()
device.setChannel(old+1)
AdvancedRemote
...
+ mute()
Radio
TV
device.setVolume(0)

Příklad

```
class Device {                              // Implementor
public:
    virtual bool isEnabled() = 0;
    virtual void enable() = 0;
    virtual void disable() = 0;
    virtual int getVolume() = 0;
    virtual void setVolume(int percent) = 0;
    virtual int getChannel() = 0;
    virtual void setChannel(int channel) = 0;
};

class Tv : public Device {                  // Concrete Implementor
private:
    bool power; int volume; int channel;
public:
     Tv() : power(false), volume(50), channel(1) {}
     bool isEnabled() override {
         return power;
     }
     void enable() override {
         power = true;
     }
     // etc...
}

class Radio : public Device {               // Concrete Implementor
    // Implementations for Radio class
};
```

Příklad

```
class RemoteControl {                                   // Abstraction
protected:
    Device* device;
public:
    RemoteControl(Device* device) : device(device) {}
    void togglePower() {
        if (device->isEnabled()) device->disable();
        else device->enable();
    }
    void volumeDown() { device->setVolume(device->getVolume() - 10); }
    void volumeUp() { device->setVolume(device->getVolume() + 10); }
    void channelDown() { device->setChannel(device->getChannel() - 1); }
    void channelUp() { device->setChannel(device->getChannel() + 1); }
};

class AdvancedRemoteControl : public RemoteControl {         // Refined Abstraction
public:
    AdvancedRemoteControl(Device* device) : RemoteControl(device) {}
    void mute() { device->setVolume(0); }
};
```

Příklad

```
int main() {
    Tv tv;
    RemoteControl remote(&tv);
    remote.togglePower();

    Radio radio;
    AdvancedRemoteControl advRemote(&radio);
    advRemote.mute();

    return 0;
}
```

Varianty implementace

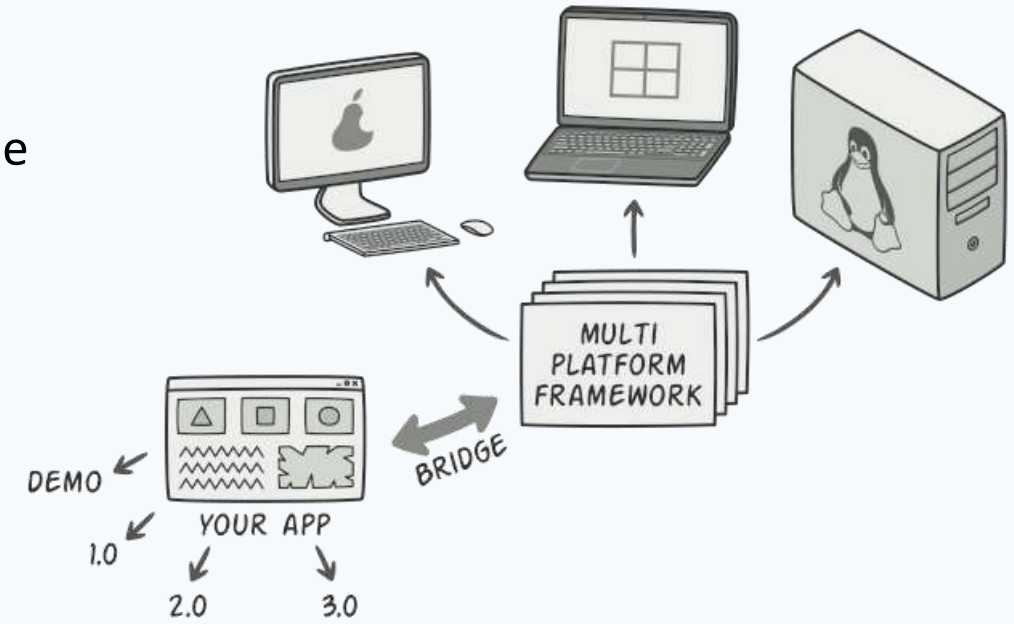
- jediný implementor
  - není nutné vytvářet abstraktní rozhraní
  - separace abstrakce a implementace

- více implementorů
  - typicky parametr konstruktoru
  - konstantní ⊗ vyměnitelný
    - Ukazatele, Reference
  - výběr konkrétního implementoru dle parametrů
    - např. druh kontejneru podle velikosti
    - možná pozdější změna
  - delegace na jiný návrhový vzor - Abstract Factory

- sdílení implementorů
  - shared_ptr

Použití

- různé varianty konkrétní funkčnosti
  - různé DB servery, API, ...
  - multiplatformní frameworky
  - nazávislý vývoj separátních nezávislých hierarchií
- potřeba rozšíření třídy ve více nezávislých rozměrech
  - separátní rozhraní/implementace pro každý rozměr
  - delegace funkčnosti přes rozhraní
- abstrakce a implementace vyžadují vlastní hierarchii
- změna implementace během runtime
  - jednoduchá změna reference
  - nezaměňovat se Strategy
- sdílení implementace více objekty



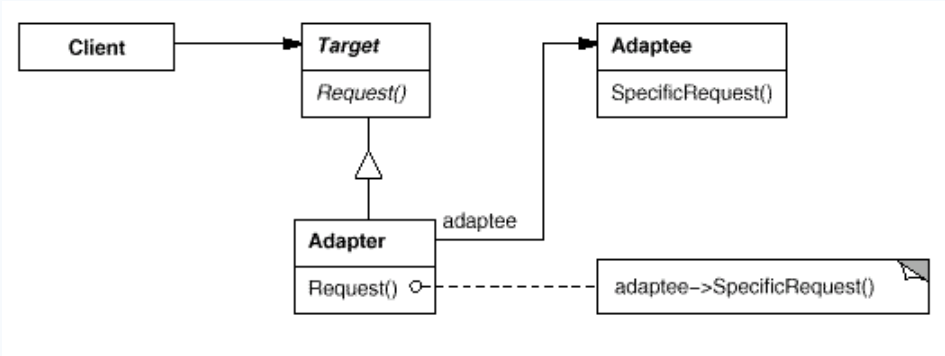
Výhody a nevýhody

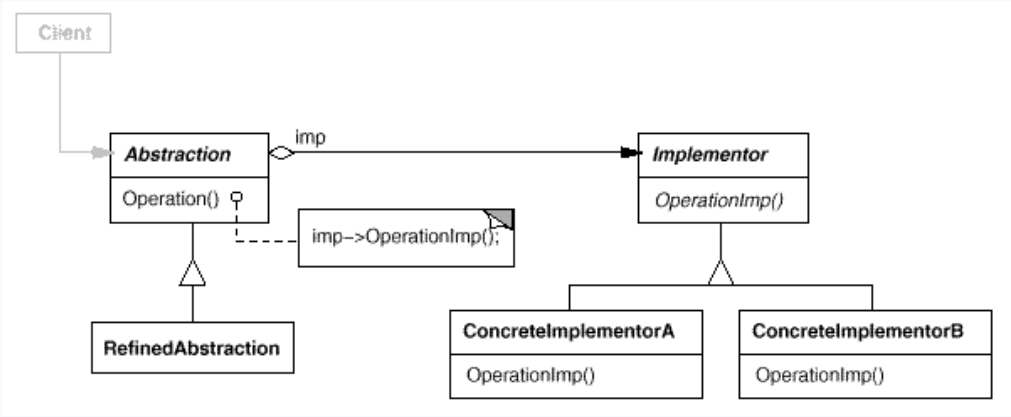
- oddělení abstrakce a implementace
  - abstrakce a implementace jsou nezávislé
  - méně tříd a duplikace kódu
    - inheritancy explosion
  - nezávislý vývoj
    - není nutná rekompilace při změně implementora
- schované implementační detaily
- změna implementace za běhu
- single-responsibility
- open-closed principle
  - otevřený pro rozšíření, uzavřený pro změny *(kódu)*

- zbytečné třídy pro jednoduché případy
  - obecná *nevýhoda* návrhových vzorů
  - není to nevýhoda principu ale použití

Související vzory

- State, Strategy, Adapter
  - založené na kompozici
- Adapter
  - podobná struktura a funčnost
  - jiný záměr
  - jiná fáze vývoje
    - design ⊗ implementace
- Abstract Factory
  - výroba a konfigurace implementorů
- Builder
  - director = abstraction
  - builders = implementations





Díky za pozornost