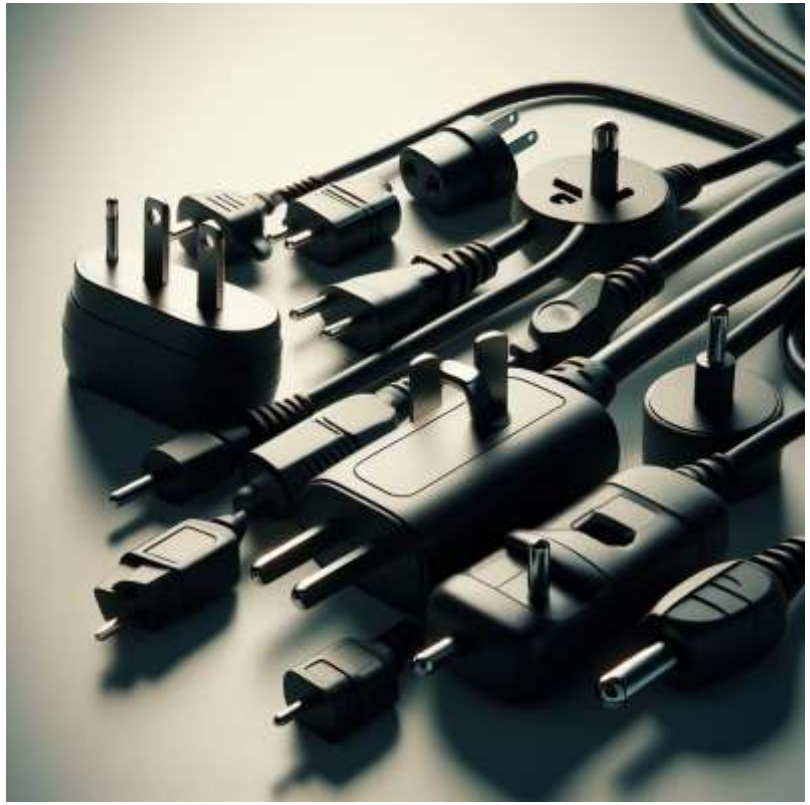


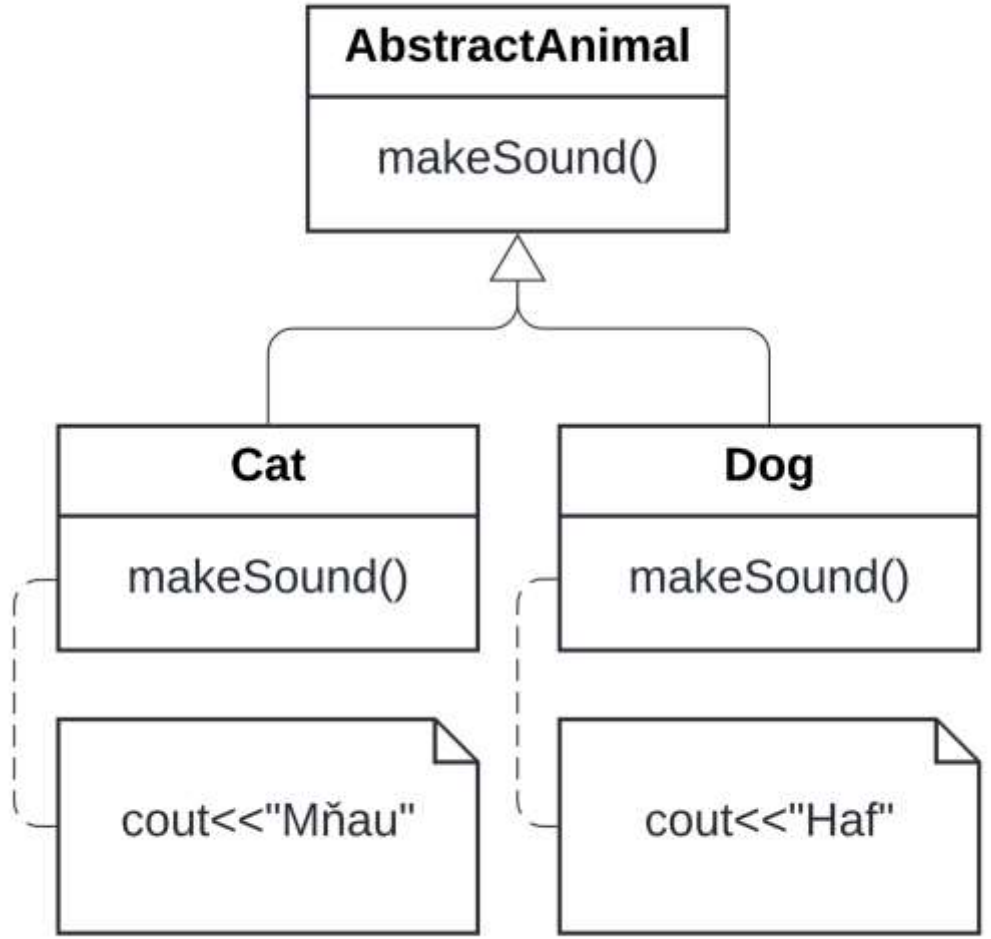
Adapter

Klára Pernicová

Convert the interface of a class into another interface clients expect. Adapter lets classes work together that couldn’t otherwise because of incompatible interfaces.

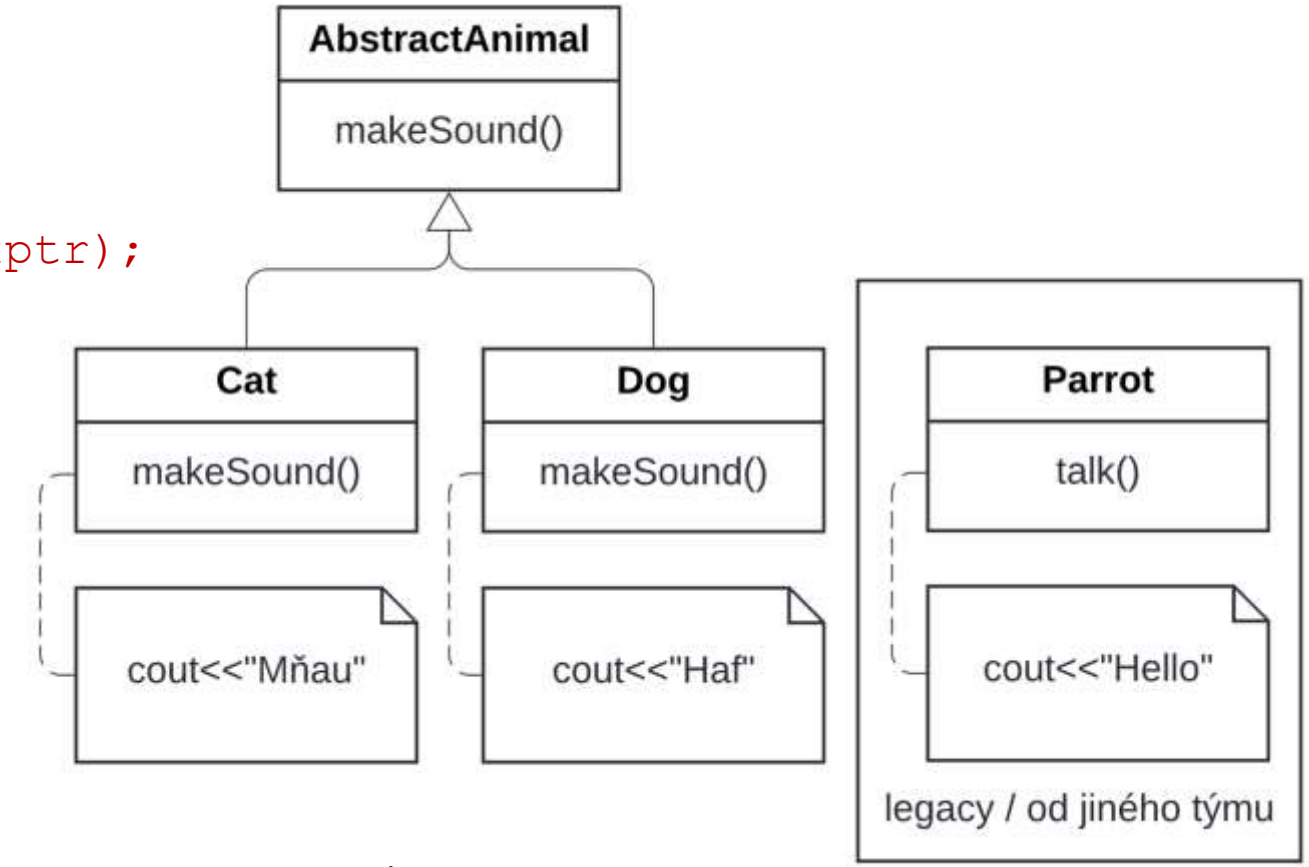
Růžový svět bez problémů

```
vector<AbstractAnimal*> myAnimals;

Cat c; myAnimals.push_back(&c);
Dog d; myAnimals.push_back(&d);

auto a = myAnimals.begin();
while(a != myAnimals.end()) {

  (*a)->makeSound();

  a++;
}

// náš kód používá iface AbstractAnimal
```



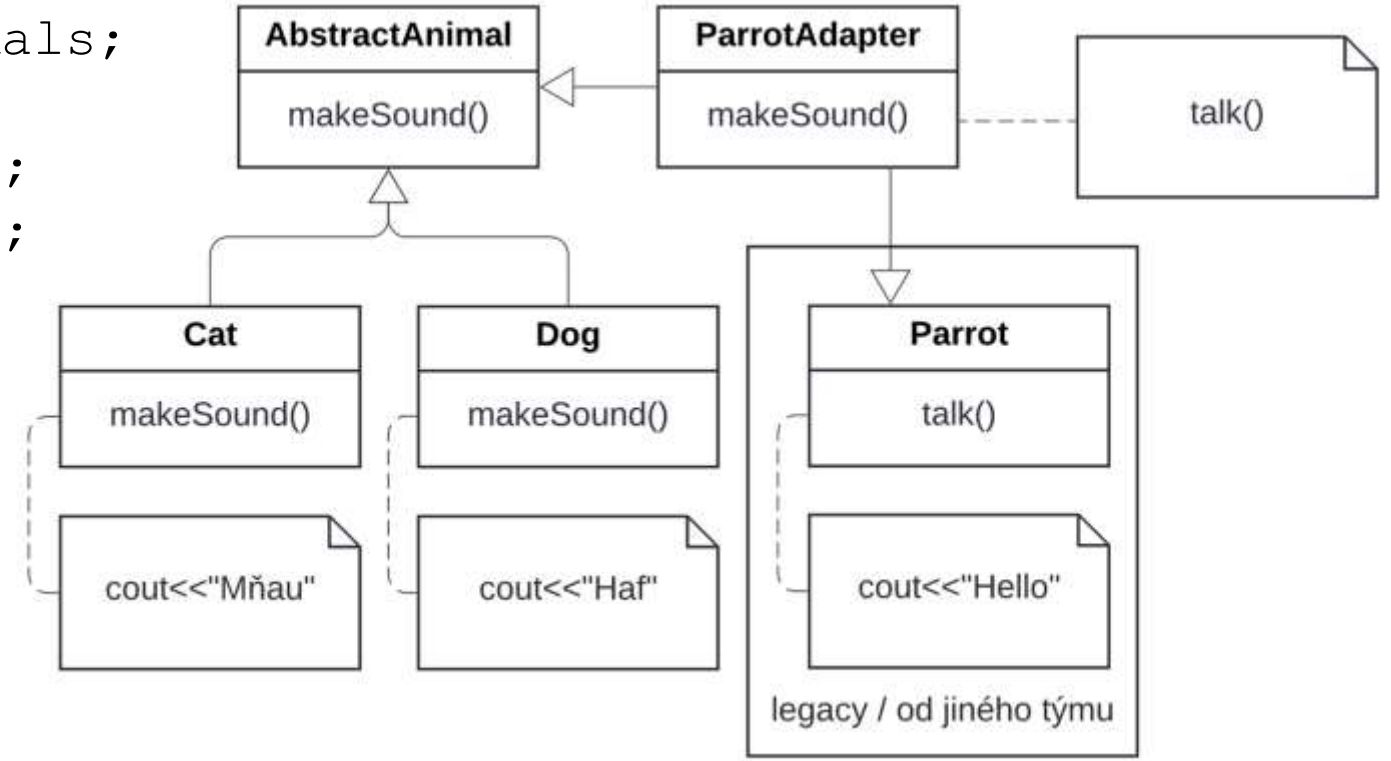
Problém vyřešen bez Adapteru

```
vector<AbstractAnimal*> myAnimals;
vector<Parrot*> myParrots;

Cat c;     myAnimals.push_back(&c);
Parrot p; myParrots.push_back(&p);
          myAnimals.push_back(nullptr);
Dog d;     myAnimals.push_back(&d);

auto a = myAnimals.begin();
auto b = myParrots.begin();
while(a != myAnimals.end()) {
  if (*a != nullptr) {
    (*a)->makeSound(); a++;
  } else {
    (*b)->talk(); b++;
  }
} // náš kód by chtěl používat iface AbstractAnimal
```



Vyřešeno Class Adapterem

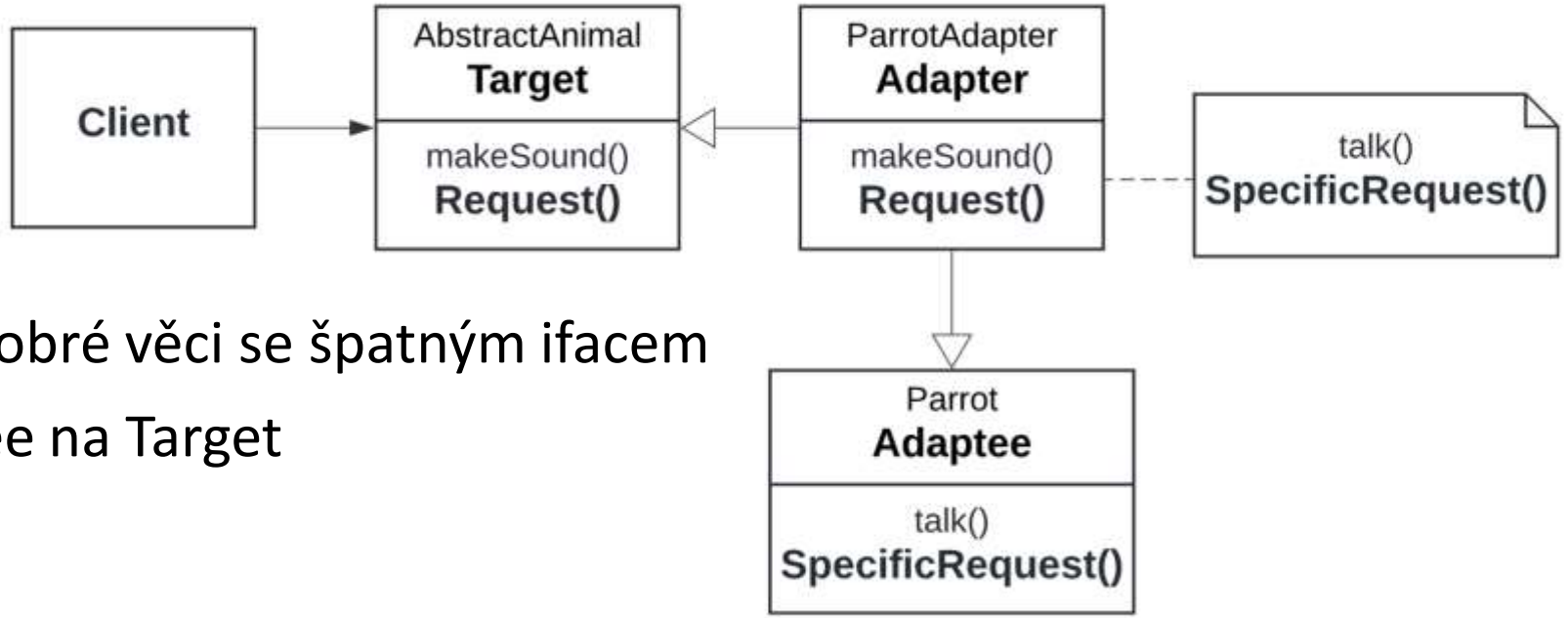
```
vector<AbstractAnimal*> myAnimals;

Cat c; myAnimals.push_back(&c);
Dog d; myAnimals.push_back(&d);

ParrotAdapter pa;
myAnimals.push_back(&pa);

auto a = myAnimals.begin();
while(a != myAnimals.end()) {

  (*a)->makeSound();
  a++;
}
```



- ParrotAdapter zajišťuje iface AbstractAnimal
- Parrot zůstává beze změny

Struktura Class Adapteru

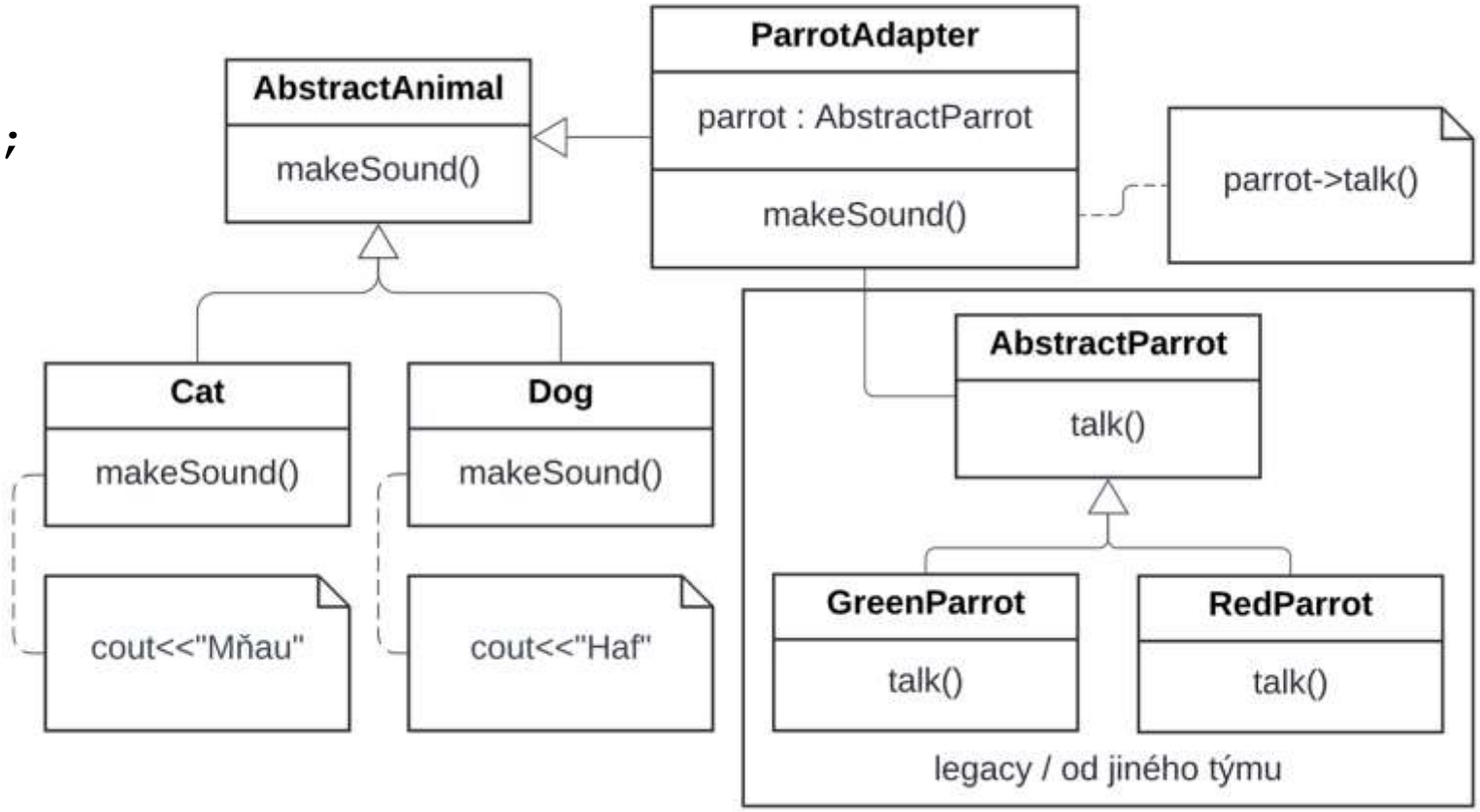
- Target: iface
- Client: využívá objekty splňující Target iface
- Adaptee: implementuje dobré věci se špatným ifacem
- Adapter: adaptuje Adaptee na Target
- Adapter dědí:
  - Veřejně Target - "is a"
  - Privátně Adaptee - "is implemented in terms of"



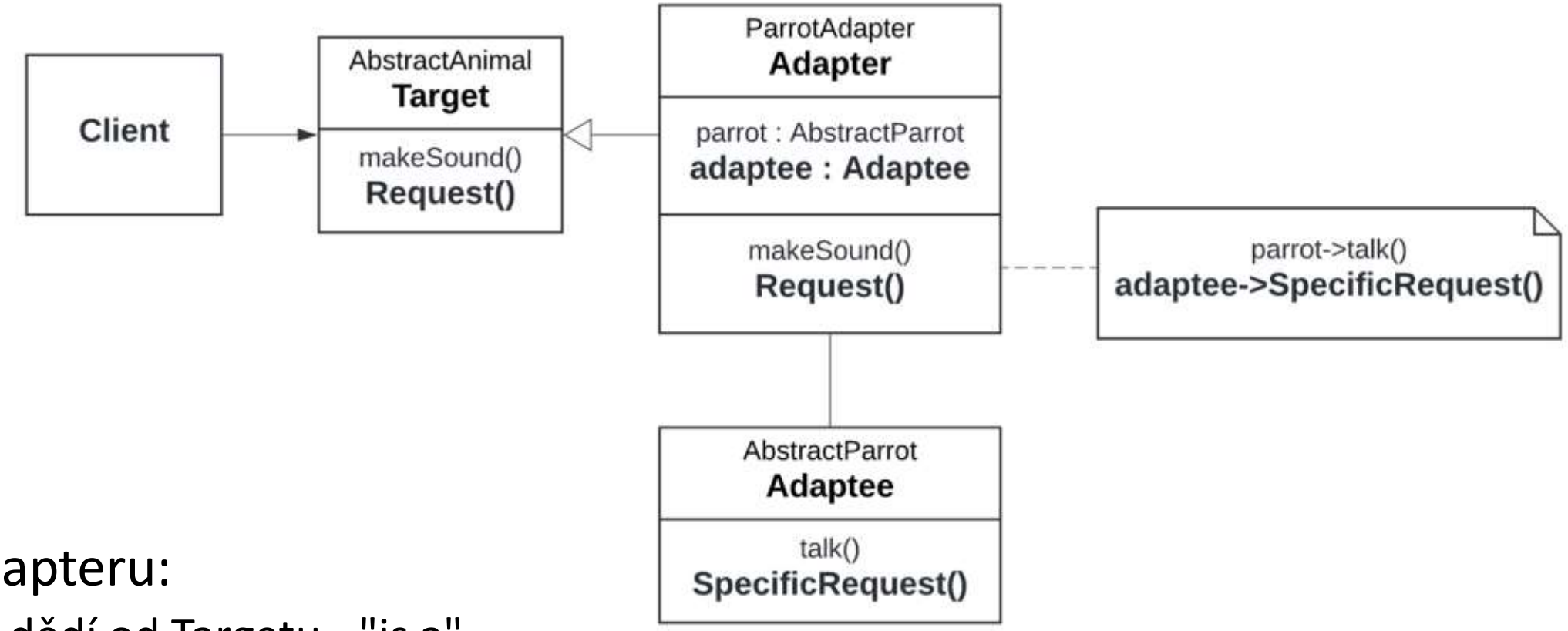
Vyřešeno Object Adapterem

```
RedParrot rp;
ParrotAdapter pa(&rp);

myAnimals.push_back(&pa);
```

- Konstruktor Adapteru bere instanci Papouška



Struktura Object Adapteru



- Vztahy Adapteru:
  - Veřejně dědí od Targetu - "is a"
  - Privátně obsahuje instanci Adaptee - "has a"

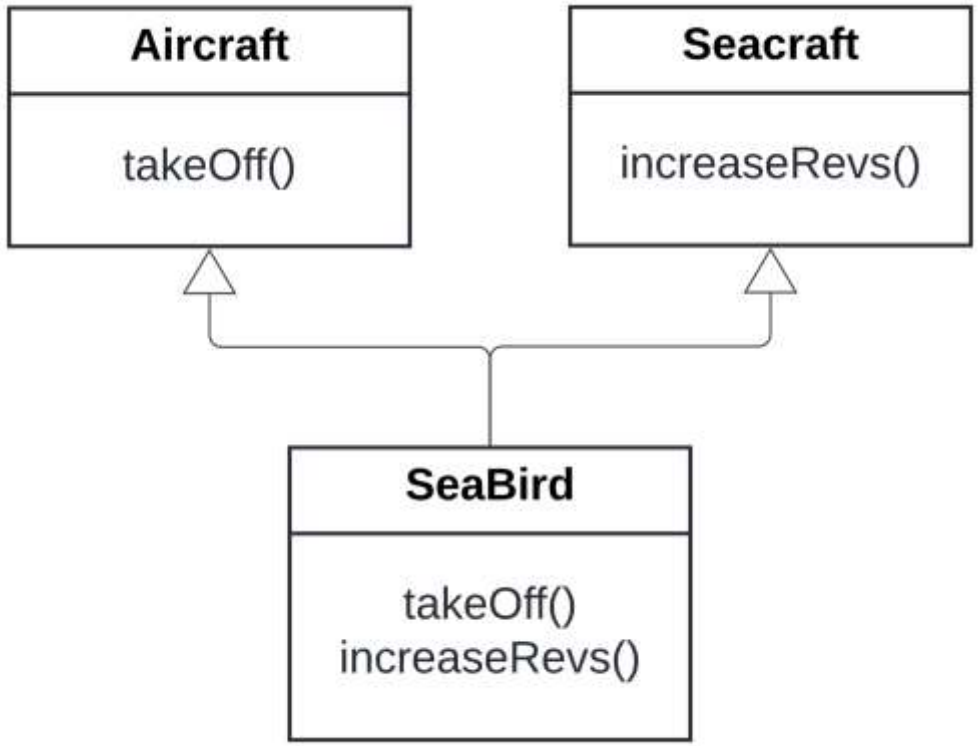
Class vs Object Adapter

- Class Adapter
- Dědí od Adaptee
- Volá metodu od Adapee přímo
- Adaptuje jednu třídu
- Možno overridovat
- Často nutná podpora vícenásobné dědičnosti

- Object Adapter
- Bere pointer na Adaptee
- Pointerová indirekce
- Adaptuje všechny třídy ifacu
- Lepší enkapsulace
- Bez vícenásobné dědičnosti

Two-Way Adapter

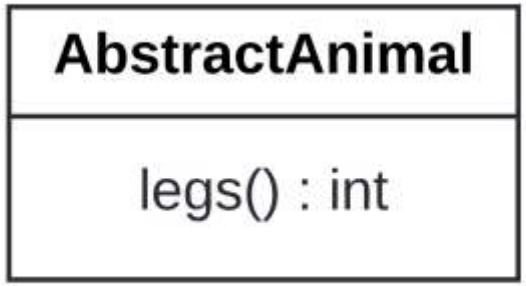
- Letadloď (Seabird) dědí veřejně od obou
- Letadloď je letadlo a zároveň loď

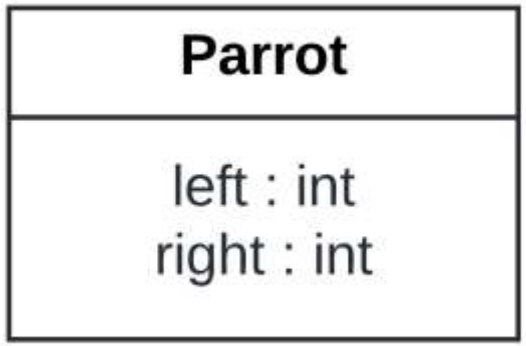


- Obtížná implementace

Pluggable Adapter

- Adapter není hard-coded na konkrétního Adapteeho






```
class AnimalAdapter : public AbstractAnimal {
  private: std::function<int()> func;
  public:
    AnimalAdapter(std::function<int()> legsFunc)
      : func(legsFunc) {}
    int legs() override { return func(); }
};

Parrot p(1, 1);
AnimalAdapter a([&p]() { return p.left + p.right; });
cout << a.legs();
```

- Velmi záleží na jazyku
- Nové features -> buďte kreativní

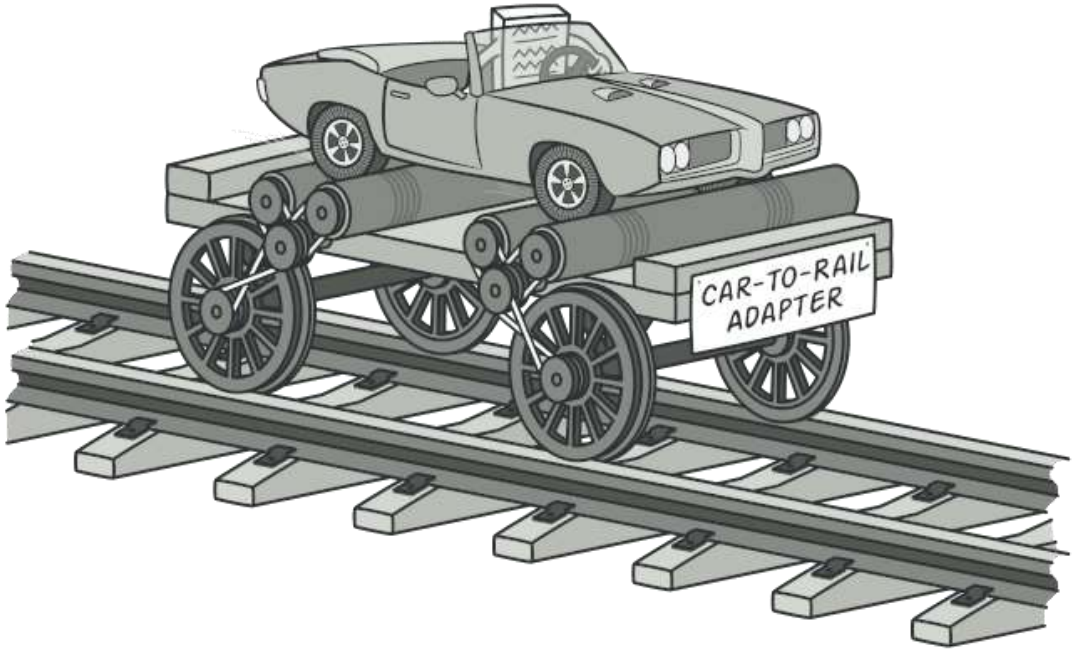
Plusy a mínusy

+ Nekompatibilní nezměnitelný iface -> dobrý iface
+ Jednoduché
- Nutno updatovat dle Adaptee iface
- Overhead
+ Cachování nagenerovaných výsledků adaptací

Podobné vzory

- Bridge
  - Design time
  - Oddělení iface od implementace
- Decorator
  - Přidává funkcionalitu
- Façade
  - Jednoduchý iface ke složitému systému
- Proxy
  - Nemění iface

- Adapter
  - Po design timu: Problem time
  - Změna špatného iface na dobrý



https://refactoring.guru/images/patterns/content/adapter/adapter-en.png