11. Cvičení

Polymorfismus, návrhové vzory.

Dědění a polymorfismus

- Situace, kde různé objekty:
  - rozumí stejné zprávě (mají metodu se stejnou signaturou),
  - ale na zprávu reagují různě (vyvoláním jiného kódu).
- Aby měl polymorfismus praktický význam, musí mít provedený kód stejný význam (v kontextu daného objektu).
- V C++, Javě omezen - polymorfismus existuje jen mezi objekty, jejichž třídy jsou ve vztahu předek/potomek.

Dědění

- Vztah mezi třídami:
  - členské proměnné a metody definované v předku mohou být použity i potomkem,
  - potomek může přidat nové členské proměnné a metody,
  - potomek může změnit definice metod předka.
- Má význam tehdy, pokud se struktura předka a potomka liší jen málo.
- V C++ lze dědit vícenásobně (z více předků).
- V C++ neexistují rozhraní (interfaces) z Javy.

Dědění

- Vždy musí platit vztah is-a.
  - Jedná se o specializaci nadtypu.
  - V našem příkladu je Cboss specializací typu CEmployee.
  - Liší se například výpočtem mzdy.
- Dědění je třeba vždy promyslet do všech důsledků (pozor na porušení Liskov's Substitution Principle (LSP)).
- Výsledek by měl vždy co nejlépe odrážet modelovanou skutečnost.

Dědění

- Rozdíl mezi vztahem is-a a has-a.
- Skládání je realizováno:
  - pomocí členské proměnné, kterou objekt využívá,
  - členské proměnné proměnných (pozor na Demeterovo pravidlo) ....
- CEmployee má (has-a) name.
- CBoss je typem Cemployee.
- Podrobněji viz OMO

Statická vazba

- Statická vazba:
  - volaná metoda je určená v době kompilace,
  - rozhoduje datový typ proměnné, kterou je instance zpřístupněna,
  - volání metody je trochu rychlejší,
  - v C++ je implicitní.

Dynamická vazba

- Dynamická vazba:
  - volaná metoda se určí v době běhu,
  - rozhoduje datový typ instance, se kterou se pracuje,
  - volání je trochu pomalejší,
  - vynutíme si ji klíčovým slovem **virtual** před deklarací metody.

Abstraktní třídy

- Abstraktní třída je třída, která obsahuje alespoň jednu abstraktní metodu.
- Abstraktní třída deklaruje metodu:
  - je dáno rozhraní metody (jméno, parametry, ...),
  - není definované tělo metody,
  - v deklaraci označena **=0,**
- existuje v předkovi, aby se vyhradil prostor v tabulce virtuálních metod (VMT).
- Těla metod definují potomci.
- Nelze vytvořit instanci abstraktní třídy.

Abstraktní třídy

- Abstraktní předek:
  - jednotný pohled na více heterogenních objektů,
  - využití rozhraní vyšší úrovně, netřeba rozlišovat detaily implementace podtříd,
- Uplatnění zejména v kolekcích.

Abstraktní třídy

- Abstraktní třídy:
  - nelze vytvořit instanci abstraktní třídy,
  - v programu existují pouze instance neabstraktních tříd - potomků,
  - Lze ale pracovat s ukazateli a referencemi typu abstraktní třída.
- Abstraktní metoda musí být **virtual. Proč?**

Abstraktní třídy

- Abstraktní metody:
  - lze vytvořit abstraktní instanční metodu,
  - nelze vytvořit abstraktní konstruktor a třídní metodu,
- Abstraktní destruktor vždy povede k chybě. Proč?

Návrhové vzory

- Návrhový vzor je obecné znovupoužitelné řešení často se vyskytujícího problému v návrhu software.
- Rozlišujeme různé druhy návrhových vzorů:
  - Creational patterns (Abstract Factory, Builder, Singleton, ...)
  - Structural patterns (Adapter, Decorator, Proxy, ...)
  - Behavioral patterns (Command, Iterator, Strategy, ...)
  - Concurrency patterns (Thread pool, ...)

Příklad

```
class DocumentManager
{
...
public:
Document* NewDocument();
private:
virtual Document* CreateDocument() = 0;
std::list<Document*> listOfDocs_;
};

Document* DocumentManager::NewDocument()
{
    Document* pDoc = CreateDocument();
    listOfDocs_.push_back(pDoc);
    ...
    return pDoc;
}
```

Problém

- Mějme obecnou knihovnu pro manipulaci s objekty. Pokud chceme, aby knihovna s objekty nejen manipulovala, ale zároveň je i vytvářela, pak je čas na použití Abstract Factory.

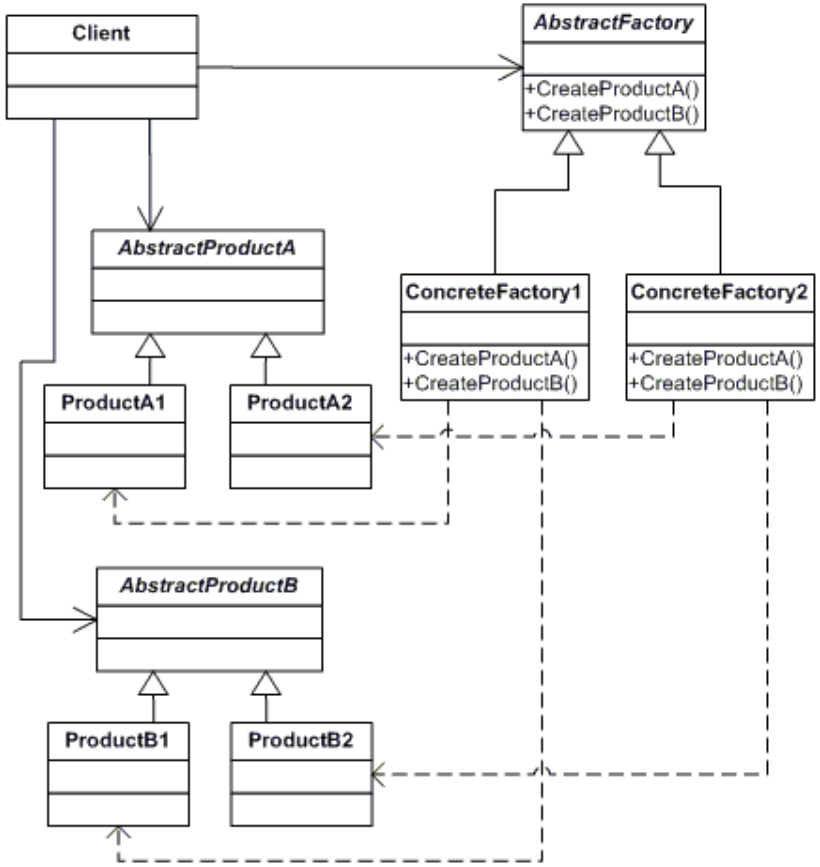
```
class Shape
{
public:
virtual void Draw() const = 0;
virtual void Rotate(double angle) = 0;
virtual void Zoom(double zoomFactor) = 0;
...
};
```

```
class Drawing
{
public:
void Save(std::ofstream& outFile);
void Load(std::ifstream& inFile);
...
};
void Drawing::Save(std::ofstream& outFile)
{
write drawing header
for (each element in the drawing)
{
(current element)->Save(outFile);
}
}
```

```
void Drawing::Load(std::ifstream& inFile)
{
// error handling omitted for simplicity
while (inFile)
{
// read object type
    int drawingType;
    inFile >> drawingType;
    // create a new empty object
    Shape* pCurrentObject;
    switch (drawingType)
    {
      using namespace DrawingType;
      case LINE:
                pCurrentObject = new Line;
      break;
      case POLYGON:
      pCurrentObject = new Polygon;
```

```
    break;
    case CIRCLE:
      pCurrentObject = new Circle;
      break;
    default:
    handle error—unknown object type
    }
// read the object's contents by invoking a virtual fn
pCurrentObject->Read(inFile);
add the object to the container
}
}
```

Abstract Factory



Zadání cvičení

- Vytvořte datový model aplikace vektorového editoru. Datový model by měl být snadno rozšiřitelný. Vyjděte z předchozího příkladu.
- Zaměřte se především na rozhraní pro vytváření jednotlivých instancí.