Dziedziczenie, polimorfizm

Marcin Michalski

ZMP 2024

08.05.2024 WMAT PWr

Dziedziczenie

```
class Derived_class: access_spec Base_class {
   ...
};
```

- access_spec to specyfikator dostępu: public, protected lub private. Pola i metody o dostępie mniej restrykcyjnym, niż access_spec są dziedziczone z dostępem access_spec, a pozostałe bez zmian (prywatne nie są dziedziczone wcale).
- Nie są dziedziczone również: konstruktory, destruktory "przyjaciele" i przeciążone operatory (a przynajmniej nie za darmo).
- Chociaż konstruktory nie są dziedziczone, to inicjalizacja obiektu Derived_class wiąże się z inicjalizacją obiektu Base_class przez wezwanie jej konstruktora.

Dziedziczenie

- Jeśli nie korzystamy z konstruktorów domyślnych, to można jawnie wezwać konstruktor klasy bazowej podobnie jak inicjalizujemy pola klasy

```
Derived_class(args) : Base_class(args) {...};
```

Przykłady

Kody: `constructor_call.cpp`, `inheritance.cpp`

Dziedziczenie z wielu klas

```
class Derived_class: access_spec_1 Base_class_1,
   ..., access_spec_n Base_class_n {
   ...
};
```

- Pola i metody wymienionych klas są dziedziczone zgodnie ze specyfikatorami dostępu na tych samych zasadach, co z jednej klasy.
- Najlepiej, gdy pola i metody w klasach dziedziczonych mają różne identyfikatory. Konflikty uniemożliwiają odwołanie się do pola/metody bezpośrednio przez klasę dziedziczącą.

Przykłady

Kod: `mult_classes_conflict.cpp`, `shapes.cpp`

Pozbywamy się redundancji

Q: Jak uniknąć powtarzania osobno niemal tej samej funkcji (tu: `area`) w każdej dziedziczącej klasie?

Będziemy eksploatować fakt, że wskaźniki na klasę bazową i na klasy pochodne są kompatybilne.

Pozbywamy się redundancji

Q: Jak uniknąć powtarzania osobno niemal tej samej funkcji (tu: `area`) w każdej dziedziczącej klasie?

Będziemy eksploatować fakt, że wskaźniki na klasę bazową i na klasy pochodne są kompatybilne.

Demonstracja

Kod: `shapes_ptrs.cpp`

Metody wirtualne

```
class Base_class {
  public:
    virtual type fun(...){...}
};
```

- Deklaracja metody wirtualnej jest poprzedzona słowem kluczowym `virtual`.
- Metody wirtualne mogą być redefiniowane w klasach pochodnych i być przez nie poprawnie wzywane, nawet przez wskaźnik na klasę bazową.

Metody wirtualne

```
class Base_class {
  public:
    virtual type fun(...){...}
};
```

- Deklaracja metody wirtualnej jest poprzedzona słowem kluczowym `virtual`.
- Metody wirtualne mogą być redefiniowane w klasach pochodnych i być przez nie poprawnie wzywane, nawet przez wskaźnik na klasę bazową.

Demonstracja

Kod: `shapes_virtual.cpp`

Metody czysto wirtualne i klasy abstrakcyjne

```
class Base_class {
  public:
    virtual type fun(...)=0;
};
```

- Deklaracja metody czysto wirtualnej jest poprzedzona słowem kluczowym `virtual`, a definicję ma zastąpioną wyrażeniem =0.
- Klasa z czyto wirtualną metodą jest nazywana klasą abstrakcyjną. Może być tylko dziedziczona, nie da się zainicjować obiektów takiej klasy.
- Można za to deklarować na nią wskaźniki!

Metody czysto wirtualne i klasy abstrakcyjne

```
class Base_class {
  public:
    virtual type fun(...)=0;
};
```

- Deklaracja metody czysto wirtualnej jest poprzedzona słowem kluczowym `virtual`, a definicję ma zastąpioną wyrażeniem =0.
- Klasa z czyto wirtualną metodą jest nazywana klasą abstrakcyjną. Może być tylko dziedziczona, nie da się zainicjować obiektów takiej klasy.
- Można za to deklarować na nią wskaźniki!

Demonstracja

Kody: `pure_virtual.cpp`, `pure_virtual2.cpp`

Przeciążanie standardowych operatorów

```
type operator symbol (...) {...}
```

- `type`, to zwracany typ (może być `void`),
- `operator` to słowo kluczowe,
- `symbol` to symbol przeciążanego operatora,
- można przeciążyć każdy operator poza `::` `.` `.*` `:?`

Przeciążanie standardowych operatorów

```
type operator symbol (...) {...}
```

- `type`, to zwracany typ (może być `void`),
- `operator` to słowo kluczowe,
- `symbol` to symbol przeciążanego operatora,
- można przeciążyć każdy operator poza `::` `.` `.*` `:?`

Demonstracja

Kody: `overload_arithmetic.cpp`, `overload_io.cpp`, `overload_[].cpp`