Programowanie obiektowe

Ćwiczenie nr. 5

Dziedziczenie. Polimorfizm.

dr inż. Adam Wolniakowski

1. Cel ćwiczenia

Celem ćwiczenia jest zapoznanie się z podstawowymi pojęciami z zakresu programowania obiektowego: mechanizmem dziedziczenia i polimorfizmu. W trakcie ćwiczenia studenci modyfikują opracowany wcześniej projekt poprzez utworzenie wspólnego interfejsu dla różnego rodzaju sygnałów. Zaimplementowany zostaną generator sygnału sinusoidalnego oraz wprowadzona interpolacja dla sygnału spróbkowanego.

2. Informacje

2.1. Funkcje wirtualne

W normalnej sytuacji, kiedy zdefiniujemy hierarchię klas w następujący sposób:

```
class A {
public:
  void foo() { cout << "Jestem A" << endl; }
};

class B : public A {
public:
  void foo() { cout << "Jestem B" << endl; }
};
```

oraz utworzymy obiekty i wywołamy ich metody foo():

```
A* a = new A;
A* b = new B;
a->foo();
b->foo();
```

otrzymamy wynik:

```
Jestem A
Jestem A
```

Kompilator nie przechowuje informacji, że wskaźnik b tak naprawdę wskazuje na obiekt klasy B. Aby to zmienić (aby kompilator pamiętał na co pokazuje wskaźnik), metodę foo() deklarujemy jako wirtualną:

```
class A {
public:
  virtual void foo() { cout << "Jestem A" << endl; }
};

class B : public A {
public:
  virtual void foo() { cout << "Jestem B" << endl; }
};
```

Teraz przy wywołaniu:

```
A* a = new A;
A* b = new B;
a->foo();
b->foo();
```

Program "pamięta", że wskaźnik b wskazuje na obiekt klasy B:

```
Jestem A
Jestem B
```

2.2. Abstrakcyjna klasa bazowa (interfejs)

Abstrakcyjna klasa bazowa (ang. *abstract base class*; inaczej: interfejs) definiuje jakie metody powinny zawierać klasy, które po niej dziedziczą. Klasa taka nie zawiera konstruktora – obiektów tej klasy nie można utworzyć. Klasa zawiera wyłącznie definicje stałych, czystych metod wirtualnych, oraz ewentualnie metod zaprzyjaźnionych. Przykład:

```
class Sygnal {
public:
  virtual ~Sygnal() {}
  virtual double x(double t) = 0;
  virtual void wypisz(std::ostream& s) = 0;
   friend std::ostream& operator<<(std::ostream& s, const Sygnal&
sygnal) {
    sygnal.wypisz(s);
    return s;
  }
};
```

Dla klasy ABC (*Abstract Base Class*) nie definiujemy metod w pliku cpp. Klasa taka powinna zawierać wirtualny destruktor.

W klasach dziedziczących po klasie ABC metody czysto wirtualne **powinny być zaimplementowane.**

2.3. Sygnał sinusoidalny

Sygnał sinusoidalny opisany jest następującym równaniem:

$$x = A\sin(\omega t + \psi)$$

gdzie: x oznacza wartość sygnału w określonej chwili t, ω – częstotliwość kołową sygnału (w rad/s), natomiast ψ – przesunięcie fazowe (w radianach).

Przykładowa implementacja (z pominięciem getterów/setterów itp.):

```
class SygnalSinusoidalny : public Sygnal {
public:
  SygnalSinusoidalny(double a=1.0, double w=1.0, double psi=0.0) :
    _a(a), _w(w), _psi(psi) {}
  virtual ~SygnalSinusoidalny() {}
  virtual double x(double t) {
    return _A * sin(_w * t + _psi);
  }
  virtual void wypisz(std::ostream& s) {
    s << "Sygnal sinusoidalny" << endl;
    s << " - A= " << _A << endl;
    s << " - w= " << _w << endl;
    s << " - psi= " << _psi << endl;
  }
private:
  double _A;
  double _w;
  double _psi;
};
```

2.4. Interpolacja sygnału

Wartości dla sygnału próbkowanego są znane tylko w dyskretnych chwilach czasu. Jak znaleźć wartość sygnału dla chwili pomiędzy próbkami?

Sygnał można *interpolować*.

Metoda 1. Interpolacja zerowego rzędu (ZOH – *Zero Order Hold*).

Dla zadanej chwili t znajdujemy najbliższą próbkę (xp, tp), która wystąpiła przed chwilą t. Zwracamy wartość sygnału dla tej próbki:

$$x(t)=x_p$$

Metoda 2. Interpolacja liniowa (pierwszego rzędu).

Dla zadanej chwili t znajdujemy poprzednią próbkę (tp, xp) oraz następną (tn, xn). Znajdujemy wartość sygnału w danej chwili:

$$x(t)=x_p+\frac{x_n-x_p}{t_n-t_p}(t-t_p)$$

2.5. Implementacja operatora << dla hierarchii klas

Funkcje zaprzyjaźnione nie są dziedziczone. Jak zdefiniować operator wyjścia dla klasy bazowej tak aby jego działanie można było przedefiniować dla klas potomnych?

Można zdefiniować operator << jako funkcję zaprzyjaźnioną w klasie bazowej, która wywołuje pewną wirtualną funkcję klasy bazowej. Tę funkcję można następnie przedefiniować w klasach potomnych. Przykład:

```
class A {
public:
  virtual ~A();
  virtual wypisz(std::ostream& strumien) = 0;
  friend std::ostream& operator<<(std::ostream& strumien, const A&
a) {
    a.wypisz(strumien);
    return strumien;
  }
};

class B : public A {
public:
  B();
  virtual ~B();
  virtual wypisz(std::ostream& strumien) {
    strumien << _zmienna << endl;
```

```
  }
private:
  string _zmienna;
};
```

3. Zadanie

Należy zmodyfikować projekt programu operującego na sygnałach utworzony na poprzednich zajęciach. Zaleca się pracę na tym samym repozytorium (np. poprzez utworzenie nowej gałęzi *branch* projektu). W realizacji zadania należy oprzeć się na załączonym na końcu diagramie klas.

3.1. Należy zmienić nazwę klasy Sygnal na SygnalProbkowany. (Uwaga: należy też zmienić wszystkie odniesienia do klasy Sygnal na SygnalProbkowany).

3.2. Należy utworzyć abstrakcyjną klasę bazową (interfejs) Sygnal. W interfejsie należy zadeklarować metody *virtual double x(double t) = 0* oraz v*irtual void wypisz(std::ostream& strumien) = 0*. Klasa powinna też posiadać zaprzyjaźniony operator<< wywołujący funkcję *wypisz()*.

3.3. Klasa SygnalProbkowany powinna dziedziczyć po klasie Sygnal. W klasie potomnej należy zaimplementować czyto wirtualne metody interfejsu Sygnal. Metoda *x()* powinna realizować wybraną metodę interpolacji (ZOH / FOH) sygnału.

3.4. Należy utworzyć nową klasę SygnalSinusoidalny implementującą interfejs Sygnal. Klasa powinna w metodzie *x()* zwracać wartość sygnału sinusoidalnego opisanego parametrami: amplituda, częstotliwość, przesunięcie fazowe. Należy samodzielnie zaprojektować interfejs klasy (konstruktor, gettery/settery). Metoda *wypisz()* nadpisana w klasie powinna wypisać parametry sygnału (amplituda itd.).

3.5. W głównej funkcji programu (plik main.cpp) należy zrealizować następujące zadania:

(1) Wczytaj sygnał zapisany w pliku *sygnal.csv*, np.:

```
SygnalLoader loader;
Sygnal* syg_wczytany = loader.wczytajSygnal(sygnal.csv");
```

(2) Utwórz nowy sygnał sinusoidalny o parametrach A=1, $\omega=2\pi$, $\psi=0$, np.:

```
Sygnal* syg_sinus = new SygnalSinusoidalny(1.0, 6.28, 0.0);
```

(3) Utwórz nowy sygnał próbkowany, np. o nazwie syg_roznica. W sygnale zapisz w postaci próbek różnice pomiędzy sygnałem wczytanym a sinusoidalnym

zrealizowane w przedziale t = [0; 1s] z częstotliwością próbkowania 100 Hz. Zapisz otrzymany sygnał do pliku o wybranej nazwie.

(4) (*) Oblicz całkę i średnią wartość różnicy sygnałów.

3.6. (*) Zaprojektuj i zaimplementuj klasę (np. SygnalProstokatny) reprezentującą sygnał prostokątny o zadanej amplitudzie, częstotliwości i wypełnieniu).

3.7. (**) Zaproponuj modyfikację klasy AnalizatorSygnalu tak, aby mogła obsługiwać też sygnały klas SygnalSinusoidalny i SygnalProstokatny.