Metody wirtualne i abstrakcyjne

Bogdan Kreczmer

bogdan.kreczmer@pwr.edu.pl

Katedra Cybernetyki i Robotyki
Politechnika Wrocławska

Kurs: Programowanie obiektowe

Niniejsza prezentacja została wykonana przy użyciu systemu składu LaTeX *oraz stylu* `beamer`, *którego autorem jest Till Tantau.*

Strona domowa projektu `Beamer`:
`http://latex-beamer.sourceforge.net`

Plan prezentacji

Plan prezentacji

Rozmiar obiektów

```
struct KlasaBezWMetody { //.....
        int _PoleInt;
        int Wartosc( ) const { return 1; }
}; //.....

 struct KlasaZWirMetoda1 { //.....
        int _PoleInt;
        virtual int Wartosc( ) const { return 1; }
}; //.....

 struct KlasaZWirMetoda2 { //.....
        int _PoleInt;
        virtual int Wartosc1( ) const { return 1; }
        virtual int Wartosc2( ) const { return 2; }
}; //.....

int main( )
{
      cout << sizeof(int ) << endl;
      cout << sizeof(KlasaBezWMetody) << endl;
      cout << sizeof(KlasaZWirMetoda1) << endl;
      cout << sizeof(KlasaZWirMetoda2) << endl;
}
```

Rozmiar obiektów

```
struct KlasaBezWMetody { //.....
        int _PoleInt;
        int Wartosc( ) const { return 1; }
}; //.....

 struct KlasaZWirMetoda1 { //.....
        int _PoleInt;
        virtual int Wartosc( ) const { return 1; }
}; //.....

 struct KlasaZWirMetoda2 { //.....
        int _PoleInt;
        virtual int Wartosc1( ) const { return 1; }
        virtual int Wartosc2( ) const { return 2; }
}; //.....

int main( )
{
      cout << sizeof(int ) << endl;
      cout << sizeof(KlasaBezWMetody) << endl;
      cout << sizeof(KlasaZWirMetoda1) << endl;
      cout << sizeof(KlasaZWirMetoda2) << endl;
}
```

Wynik działania:

```
4
4
8
8
```

Rozmiar obiektów

```
struct KlasaBezWMetody { //.....
        int _PoleInt;
        int Wartosc( ) const { return 1; }
}; //.....

 struct KlasaZWirMetoda1 { //.....
        int _PoleInt;
        virtual int Wartosc( ) const { return 1; }
}; //.....

 struct KlasaZWirMetoda2 { //.....
        int _PoleInt;
        virtual int Wartosc1( ) const { return 1; }
        virtual int Wartosc2( ) const { return 2; }
}; //.....

int main( )
{
      cout << sizeof(int ) << endl;
      cout << sizeof(KlasaBezWMetody) << endl;
      cout << sizeof(KlasaZWirMetoda1) << endl;
      cout << sizeof(KlasaZWirMetoda2) << endl;
}
```

Wynik działania:

```
4
4
8
8
```

Rozmiar obiektów

```
struct KlasaBezWMetody { //.....
        int _PoleInt;
        int Wartosc( ) const { return 1; }
}; //.....

 struct KlasaZWirMetoda1 { //.....
        int _PoleInt;
        virtual int Wartosc( ) const { return 1; }
}; //.....

 struct KlasaZWirMetoda2 { //.....
        int _PoleInt;
        virtual int Wartosc1( ) const { return 1; }
        virtual int Wartosc2( ) const { return 2; }
}; //.....

int main( )
{
      cout << sizeof(int ) << endl;
      cout << sizeof(KlasaBezWMetody) << endl;
      cout << sizeof(KlasaZWirMetoda1) << endl;
      cout << sizeof(KlasaZWirMetoda2) << endl;
}
```

Wynik działania:

```
4
4
8
8
```

Rozmiar obiektów

```
struct KlasaBezWMetody { //.....
        int _PoleInt;
        int Wartosc( ) const { return 1; }
}; //.....

 struct KlasaZWirMetoda1 { //.....
        int _PoleInt;
        virtual int Wartosc( ) const { return 1; }
}; //.....

 struct KlasaZWirMetoda2 { //.....
        int _PoleInt;
        virtual int Wartosc1( ) const { return 1; }
        virtual int Wartosc2( ) const { return 2; }
}; //.....

int main( )
{
      cout << sizeof(int ) << endl;
      cout << sizeof(KlasaBezWMetody) << endl;
      cout << sizeof(KlasaZWirMetoda1) << endl;
      cout << sizeof(KlasaZWirMetoda2) << endl;
}
```

Wynik działania:

```
4
4
8
8
```

Rozmiar obiektów

```
struct KlasaBezWMetody { //.....
        int _PoleInt;
        int Wartosc( ) const { return 1; }
}; //.....

 struct KlasaZWirMetoda1 { //.....
        int _PoleInt;
        virtual int Wartosc( ) const { return 1; }
}; //.....

struct KlasaZWirMetoda2 { //.....
        int _PoleInt;
        virtual int Wartosc1( ) const { return 1; }
        virtual int Wartosc2( ) const { return 2; }
}; //.....

int main( )
{
      cout << sizeof(int ) << endl;
      cout << sizeof(KlasaBezWMetody) << endl;
      cout << sizeof(KlasaZWirMetoda1) << endl;
      cout << sizeof(KlasaZWirMetoda2) << endl;
}
```

Wynik działania:

```
4
4
8
8
```

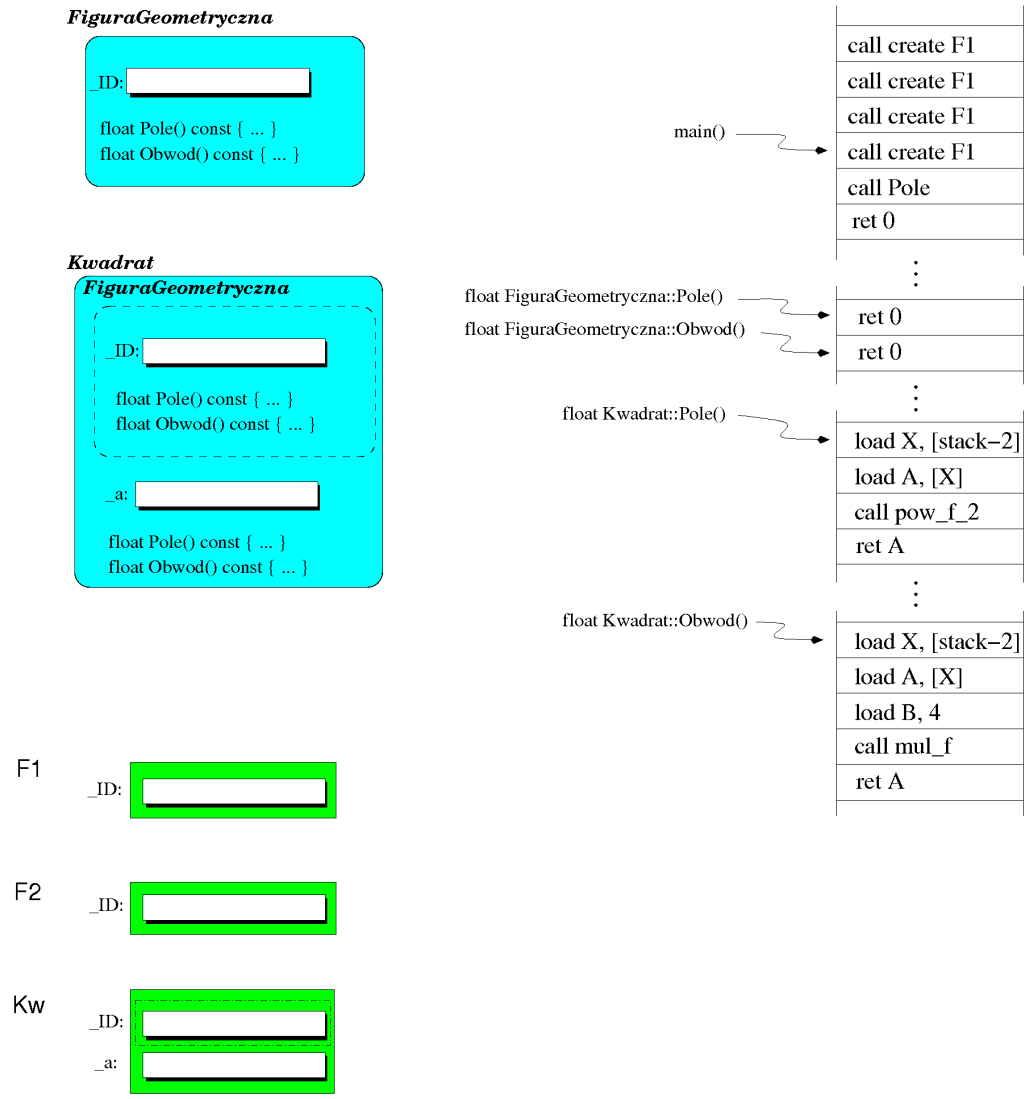
Rozmiar obiektów

```
class FiguraGeometryczna {
  public:
    int   _ID;
    float Pole() const { return 0; }
    float Obwod() const { return 0; }
};


class Kwadrat: public FiguraGeometryczna {
  public:
    float   _a;
    float Pole() const { return _a*_a; }
    float Obwod() const { return 4*_a; }
};


int main()
{
  FiguraGeometryczna  F1, F2;
  Kwadrat   Kw;

  F1.Pole();
}
```



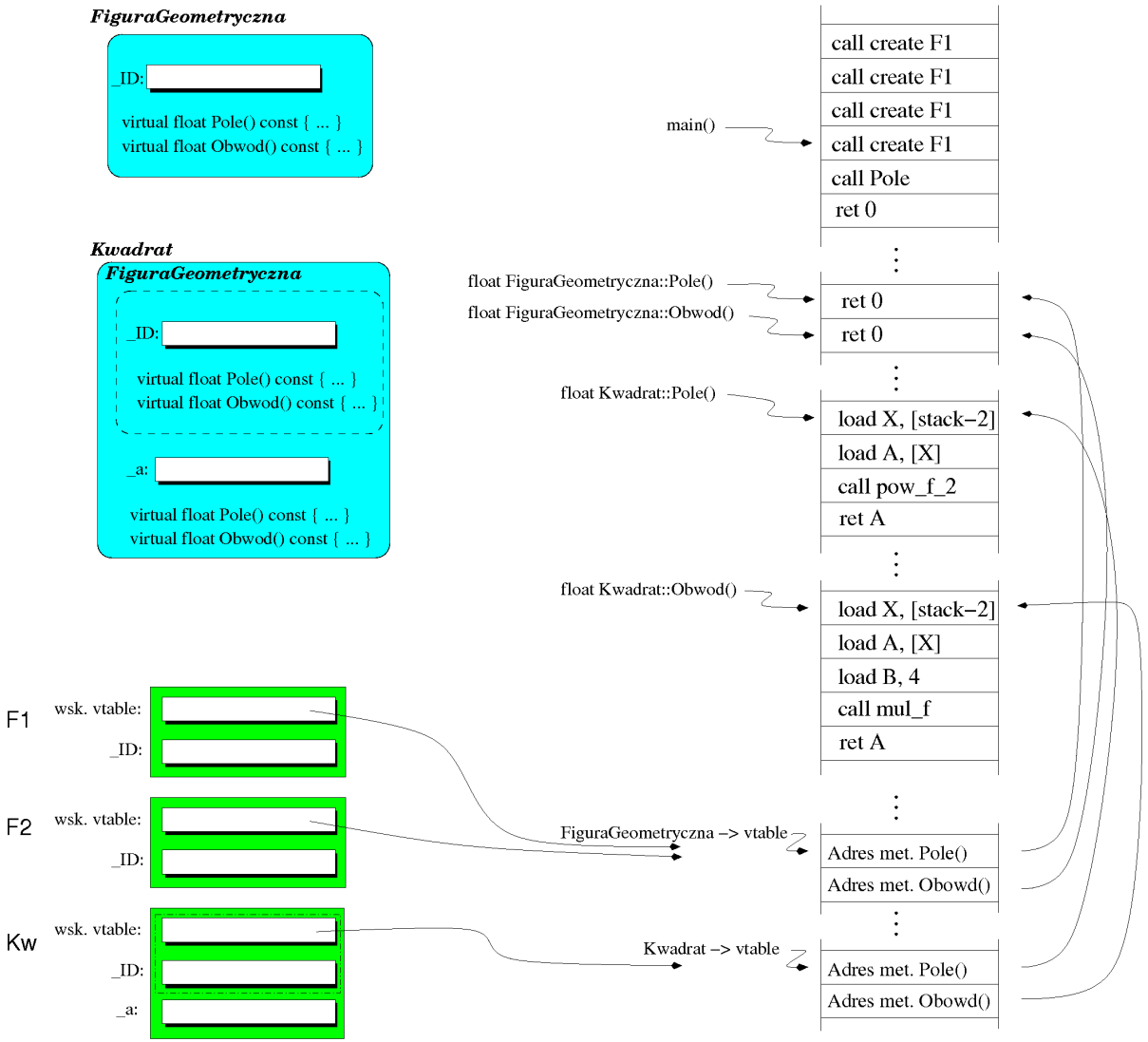
Rozmiar obiektów

```
class FiguraGeometryczna {
  public:
    int   _ID;
    virtual float Pole() const { return 0; }
    virtual float Obwod() const { return 0; }
};


class Kwadrat: public FiguraGeometryczna {
  public:
    float   _a;
    virtual float Pole() const { return _a*_a; }
    virtual float Obwod() const { return 4*_a; }
};


int main()
{
  FiguraGeometryczna  F1, F2;
  Kwadrat   Kw;

  F1.Pole();
}
```



Plan prezentacji

Definiowanie metod abstrakcyjnych

```
struct FiguraGeometryczna { //.....
    ...
    virtual   float  Pole( ) const { return   0; }
}; //.....

struct Kwadrat: public  FiguraGeometryczna{ //.....
    ...
    float  _DlugoscBoku;
    virtual   float  Pole( ) const  { return   _DlugoscBoku*_DlugoscBoku; }
    ...
}; //.....

int  main( )
{
    FiguraGeometryczna   ObFig;
    Kwadrat              ObKwa;



}
```

Definiowanie metod abstrakcyjnych

```
struct FiguraGeometryczna { //.....
    ...
    virtual   float  Pole( ) const = 0;
}; //.....

struct Kwadrat: public  FiguraGeometryczna{ //.....
    ...
    float  _DlugoscBoku;
    virtual   float  Pole( ) const  { return   _DlugoscBoku*_DlugoscBoku; }
    ...
}; //.....

int  main( )
{
    FiguraGeometryczna   ObFig;
    Kwadrat              ObKwa;



}
```

Definiowanie metod abstrakcyjnych

```
struct FiguraGeometryczna { //.....
    ...
    virtual   float  Pole( ) const = 0;
}; //.....

struct Kwadrat: public  FiguraGeometryczna{ //.....
    ...
    float  _DlugoscBoku;
    virtual   float  Pole( ) const  { return   _DlugoscBoku*_DlugoscBoku; }
    ...
}; //.....

int  main( )
{
    FiguraGeometryczna   ObFig;
    Kwadrat              ObKwa;



}
```

Definiowanie metod abstrakcyjnych

```
struct FiguraGeometryczna { //.....
    ...
    virtual   float  Pole( ) const = 0;
}; //.....

struct Kwadrat: public  FiguraGeometryczna{ //.....
    ...
    float  _DlugoscBoku;
    virtual   float  Pole( ) const  { return   _DlugoscBoku*_DlugoscBoku; }
    ...
}; //.....

int  main( )
{
    FiguraGeometryczna   ObFig;
    Kwadrat              ObKwa;



}
```

Definiowanie metod abstrakcyjnych

```
struct FiguraGeometryczna { //.....
    ...
    virtual   float  Pole( ) const = 0;
}; //.....

struct Kwadrat: public  FiguraGeometryczna{ //.....
    ...
    float  _DlugoscBoku;
    virtual   float  Pole( ) const  { return   _DlugoscBoku*_DlugoscBoku; }
    ...
}; //.....

int  main( )
{
    FiguraGeometryczna  ObFig;
    Kwadrat             ObKwa;



}
```

Zdefiniowanie metody abstrakcyjnej w danej klasie uniemożliwia samodzielne istnienie obiektu tej klasy, gdyż jedna z metod nie ma kodu. Klasa nie jest więc w pełni zdefiniowana.

Definiowanie metod abstrakcyjnych

```
struct FiguraGeometryczna { //.....
    ...
    virtual   float  Pole( ) const = 0;
}; //.....

struct Kwadrat: public  FiguraGeometryczna{ //.....
    ...
    float  _DlugoscBoku;
    virtual   float  Pole( ) const  { return   _DlugoscBoku*_DlugoscBoku; }
    ...
}; //.....

int  main( )
{
    FiguraGeometryczna  ObFig;
    Kwadrat             ObKwa;

    FiguraGeometryczna  *wsk_ObFig = &ObKwa;

}
```

Zdefiniowanie metody abstrakcyjnej w danej klasie uniemożliwia samodzielne istnienie obiektu tej klasy, gdyż jedna z metod nie ma kodu. Klasa nie jest więc w pełni zdefiniowana.
Możliwie jest jednak posługiwanie się wskaźnikami lub referencjami na obiekty tej klasy, które są składnikami innych obiektów. W klasach tych obiektów musi istnieć kod dla wszystkich metod abstrakcyjnych klasy bazowej.

Definiowanie metod abstrakcyjnych

```
struct FiguraGeometryczna { //.....
    ...
    virtual   float  Pole( ) const = 0;
}; //.....

struct Kwadrat: public  FiguraGeometryczna{ //.....
    ...
    float  _DlugoscBoku;
    virtual   float  Pole( ) const  { return   _DlugoscBoku*_DlugoscBoku; }
    ...
}; //.....

int  main( )
{
    FiguraGeometryczna  ObFig;
    Kwadrat             ObKwa;

    FiguraGeometryczna  *wsk_ObFig = &ObKwa;
    FiguraGeometryczna  &ref_ObFig = ObKwa;
}
```

Zdefiniowanie metody abstrakcyjnej w danej klasie uniemożliwia samodzielne istnienie obiektu tej klasy, gdyż jedna z metod nie ma kodu. Klasa nie jest więc w pełni zdefiniowana.
Możliwie jest jednak posługiwanie się wskaźnikami lub referencjami na obiekty tej klasy, które są składnikami innych obiektów. W klasach tych obiektów musi istnieć kod dla wszystkich metod abstrakcyjnych klasy bazowej.

Problem pełnej zgodność nazw metod i ich parametrów

```
struct FiguraGeometryczna { //.....
    virtual double Pole( ) const = 0;
}; //.....

struct Wielobok: FiguraGeometryczna { //.....
    double Pole( ) { ... }
}; //.....

struct Kwadrat: Wielobok { //.....
    double Pole( ) const { ... }
}; //.....

int main( )
{
    FiguraGeometryczna  ObFig;
    Wielobok            ObWielobok;
    Kwadrat             ObKwadrat;
}
```

Problem pełnej zgodność nazw metod i ich parametrów

```
struct FiguraGeometryczna { //.....
    virtual double Pole( ) const = 0;
}; //.....

struct Wielobok: FiguraGeometryczna { //.....
    double Pole( ) { ... }
}; //.....

struct Kwadrat: Wielobok { //.....
    double Pole( ) const { ... }
}; //.....

int main( )
{
    FiguraGeometryczna  ObFig;
    Wielobok            ObWielobok;
    Kwadrat             ObKwadrat;
}
```

Problem pełnej zgodność nazw metod i ich parametrów

```
struct FiguraGeometryczna { //.....
    virtual double Pole( ) const = 0;
}; //.....

struct Wielobok: FiguraGeometryczna { //.....
    double Pole( ) { ... }
}; //.....

struct Kwadrat: Wielobok { //.....
    double Pole( ) const { ... }
}; //.....

int main( )
{
    FiguraGeometryczna ObFig;
    Wielobok           ObWielobok;
    Kwadrat            ObKwadrat;
}
```

Problem pełnej zgodność nazw metod i ich parametrów

```
struct FiguraGeometryczna { //.....
    virtual double Pole( ) const = 0;
}; //.....

struct Wielobok: FiguraGeometryczna { //.....
    double Pole( ) { ... }
}; //.....

struct Kwadrat: Wielobok { //.....
    double Pole( ) const { ... }
}; //.....

int main( )
{
    FiguraGeometryczna ObFig;
    Wielobok           ObWielobok;
    Kwadrat            ObKwadrat;
}
```

Problem pełnej zgodność nazw metod i ich parametrów

```
struct FiguraGeometryczna { //.....
    virtual double Pole( ) const = 0;
}; //.....

struct Wielobok: FiguraGeometryczna { //.....
    double Pole( ) { ... }
}; //.....

struct Kwadrat: Wielobok { //.....
    double Pole( ) const { ... }
}; //.....

int main( )
{
    FiguraGeometryczna ObFig;
    Wielobok           ObWielobok;
    Kwadrat            ObKwadrat;
}
```

Problem pełnej zgodność nazw metod i ich parametrów

```
struct FiguraGeometryczna { //.....
    virtual double Pole( ) const = 0;
}; //.....

struct Wielobok: FiguraGeometryczna { //.....
    double Pole( ) { ... }
}; //.....

struct Kwadrat: Wielobok { //.....
    double Pole( ) const { ... }
}; //.....

int main( )
{
    FiguraGeometryczna ObFig;
    Wielobok           ObWielobok;
    Kwadrat            ObKwadrat;
}
```

Problem pełnej zgodność nazw metod i ich parametrów

```
struct FiguraGeometryczna { //.....
    virtual double Pole( ) const = 0;
}; //.....

struct Wielobok: FiguraGeometryczna { //....
    double Pole( ) { ... }
}; //.....

struct Kwadrat: Wielobok { //.....
    double Pole( ) const { ... }
}; //.....

int main( )
{
    FiguraGeometryczna ObFig;
    Wielobok           ObWielobok;
    Kwadrat            ObKwadrat;
}
```

Jeżeli w klasie pochodnej nie zostanie zdefiniowany kod dla dziedziczonej metody abstrakcyjnej, to klasa ta staje się również klasą abstrakcyjną

Plan prezentacji

Sposób na korzystanie z funkcjonalności klasy bazowej

```
struct FiguraGeometryczna { //.....
    virtual float Pole( ) const = 0;
}; //.....

struct Kwadrat: public FiguraGeometryczna{ //.....
    float _DlugoscBoku;
    virtual float Pole( ) const { return _DlugoscBoku*_DlugoscBoku; }
}; //.....

void WyswietlPole( const FiguraGeometryczna & Fig )
{
    cout << "Pole: " << Fig.Pole( ) << endl;
}

int main( )
{
    Kwadrat Kw;

    Kw._DlugoscBoku = 2;
    WyswietlPole( Kw );
}
```

Sposób na korzystanie z funkcjonalności klasy bazowej

```
struct FiguraGeometryczna { //.....
    virtual float Pole( ) const = 0;
}; //.....

struct Kwadrat: public FiguraGeometryczna{ //.....
    float _DlugoscBoku;
    virtual float Pole( ) const { return _DlugoscBoku*_DlugoscBoku; }
}; //.....

void WyswietlPole( const FiguraGeometryczna & Fig )
{
    cout << "Pole: " << Fig.Pole( ) << endl;
}

int main( )
{
    Kwadrat Kw;

    Kw._DlugoscBoku = 2;
    WyswietlPole( Kw );
}
```

Sposób na korzystanie z funkcjonalności klasy bazowej

```
struct FiguraGeometryczna { //.....
    virtual float Pole( ) const = 0;
}; //.....

struct Kwadrat: public FiguraGeometryczna{ //.....
    float _DlugoscBoku;
    virtual float Pole( ) const { return _DlugoscBoku*_DlugoscBoku; }
}; //.....

void WyswietlPole( const FiguraGeometryczna & Fig )
{
    cout << "Pole: " << Fig.Pole( ) << endl;
}

int main( )
{
    Kwadrat Kw;

    Kw._DlugoscBoku = 2;
    WyswietlPole( Kw );
}
```

Sposób na korzystanie z funkcjonalności klasy bazowej

```
struct FiguraGeometryczna { //.....
    virtual float Pole( ) const = 0;
}; //.....

struct Kwadrat: public FiguraGeometryczna{ //.....
    float _DlugoscBoku;
    virtual float Pole( ) const { return _DlugoscBoku*_DlugoscBoku; }
}; //.....

void WyswietlPole( const FiguraGeometryczna & Fig )
{
    cout << "Pole: " << Fig.Pole( ) << endl;
}

int main( )
{
    Kwadrat Kw;

    Kw._DlugoscBoku = 2;
    WyswietlPole( Kw );
}
```

Sposób na korzystanie z funkcjonalności klasy bazowej

```
struct FiguraGeometryczna { //.....
    virtual float Pole( ) const = 0;
}; //.....

struct Kwadrat: public FiguraGeometryczna{ //.....
    float _DlugoscBoku;
    virtual float Pole( ) const { return _DlugoscBoku*_DlugoscBoku; }
}; //.....

void WyswietlPole( const FiguraGeometryczna & Fig )
{
    cout << "Pole: " << Fig.Pole( ) << endl;
}

int main( )
{
    Kwadrat Kw;

    Kw._DlugoscBoku = 2;
    WyswietlPole( Kw );
}
```

Wynik działania:

```
Pole: 4
```

Sposób na korzystanie z funkcjonalności klasy bazowej

```
struct FiguraGeometryczna { //.....
    virtual float Pole( ) const = 0;
}; //.....

struct Kwadrat: public FiguraGeometryczna{ //.....
    float _DlugoscBoku;
    virtual float Pole( ) const { return _DlugoscBoku*_DlugoscBoku; }
}; //.....

void WyswietlPole( const FiguraGeometryczna & Fig )
{
    cout << "Pole: " << Fig.Pole( ) << endl;
}

int main( )
{
    Kwadrat Kw;

    Kw._DlugoscBoku = 2;
    WyswietlPole( Kw );
}
```

Wynik działania:

```
Pole: 4
```

Możliwość korzystania ze wskaźników i referencji do klasy abstrakcyjnej pozwala na definiowanie dla niej funkcji lub metod, które będą wykorzystywane dla obiektów klasy pochodnej.

Własny identyfikator typu

```
struct FiguraGeometryczna { //.....
        ...
    virtual int ID( ) const = 0;
}; //.....

struct Kwadrat: FiguraGeometryczna { //.....
        ...
    int ID( ) const { return 1; }
}; //.....

struct Okrag: FiguraGeometryczna { //.....
        ...
    int ID( ) const { return 2; }
}; //.....

int main( )
{
    FiguraGeometryczna *wFigA = new Kwadrat;
    FiguraGeometryczna *wFigB = new Okrag;

    cout << wFigA->ID( ) << endl;
    cout << wFigB->ID( ) << endl;
}
```

Własny identyfikator typu

```
struct FiguraGeometryczna { //.....
        ...
    virtual int ID( ) const = 0;
}; //.....

struct Kwadrat: FiguraGeometryczna { //.....
        ...
    int ID( ) const { return 1; }
}; //.....

struct Okrag: FiguraGeometryczna { //.....
        ...
    int ID( ) const { return 2; }
}; //.....

int main( )
{
    FiguraGeometryczna *wFigA = new Kwadrat;
    FiguraGeometryczna *wFigB = new Okrag;

    cout << wFigA->ID( ) << endl;
    cout << wFigB->ID( ) << endl;
}
```

Własny identyfikator typu

```
struct FiguraGeometryczna { //.....
        ...
    virtual int ID( ) const = 0;
}; //.....

struct Kwadrat: FiguraGeometryczna { //.....
        ...
    int ID( ) const { return 1; }
}; //.....

struct Okrag: FiguraGeometryczna { //.....
        ...
    int ID( ) const { return 2; }
}; //.....

int main( )
{
    FiguraGeometryczna *wFigA = new Kwadrat;
    FiguraGeometryczna *wFigB = new Okrag;

    cout << wFigA->ID( ) << endl;
    cout << wFigB->ID( ) << endl;
}
```

Własny identyfikator typu

```
struct FiguraGeometryczna { //.....
        ...
    virtual int ID( ) const = 0;
}; //.....

struct Kwadrat: FiguraGeometryczna { //.....
        ...
    int ID( ) const { return 1; }
}; //.....

struct Okrag: FiguraGeometryczna { //.....
        ...
    int ID( ) const { return 2; }
}; //.....

int main( )
{
    FiguraGeometryczna *wFigA = new Kwadrat;
    FiguraGeometryczna *wFigB = new Okrag;

    cout << wFigA->ID( ) << endl;
    cout << wFigB->ID( ) << endl;
}
```

Własny identyfikator typu

```
struct FiguraGeometryczna { //.....
        ...
    virtual int ID( ) const = 0;
}; //.....

struct Kwadrat: FiguraGeometryczna { //.....
        ...
    int ID( ) const { return 1; }
}; //.....

struct Okrag: FiguraGeometryczna { //.....
        ...
    int ID( ) const { return 2; }
}; //.....

int main( )
{
    FiguraGeometryczna *wFigA = new Kwadrat;
    FiguraGeometryczna *wFigB = new Okrag;

    cout << wFigA->ID( ) << endl;
    cout << wFigB->ID( ) << endl;
}
```

Własny identyfikator typu

```
struct FiguraGeometryczna { //.....
        ...
    virtual int ID( ) const = 0;
}; //.....

struct Kwadrat: FiguraGeometryczna { //.....
        ...
    int ID( ) const { return 1; }
}; //.....

struct Okrag: FiguraGeometryczna { //.....
        ...
    int ID( ) const { return 2; }
}; //.....

int main( )
{
    FiguraGeometryczna *wFigA = new Kwadrat;
    FiguraGeometryczna *wFigB = new Okrag;

    cout << wFigA->ID( ) << endl;
    cout << wFigB->ID( ) << endl;
}
```

Wynik działania:

```
1
2
```

Własny identyfikator typu

```
struct FiguraGeometryczna { //.....
        ...
    virtual int ID( ) const = 0;
}; //.....

struct Kwadrat: FiguraGeometryczna { //.....
        ...
    int ID( ) const { return 1; }
}; //.....

struct Okrag: FiguraGeometryczna { //.....
        ...
    int ID( ) const { return 2; }
}; //.....

int main( )
{
    FiguraGeometryczna *wFigA = new Kwadrat;
    FiguraGeometryczna *wFigB = new Okrag;

    cout << wFigA->ID( ) << endl;
    cout << wFigB->ID( ) << endl;
}
```

Metody wirtualne można wykorzystać do konstrukcji własnego identyfikatora typu. Wymaga to wykorzystanie jednej wspólnej klasy bazowej dla danego zbioru klas pochodnych.

Wynik działania:

```
1
2
```

Podsumowanie

- Metody abstrakcyjne pozwalają na tworzenie prototypu interfejsu klasy bazowej, który przejmowany jest przez klasę pochodną i w niej jest definiowany.
  Pozwala to stworzyć prototypy funkcjonalności klasy bazowej.

Koniec prezentacji
Dziękuję za uwagę