Wyjątki

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

Tradycyjny sposób obsługi błędów

```
int Funkcja1()
{
  ...
  if (Gdy_cos_jest_zle) { errno = KOD_BLEDU; return -1; }
  ...
  return 0;
}
...

int Funkcja()
{
  int err;
  if (Funkcja1() < 0) { /* Obsługa błędu */
    ...
    if (Jezeli_nie_mozemy_sobie_poradzic) return -1;
    ...
  }
  if ((err = Funkcja2()) < 0) { /* Obsługa błędu */ ... }
  ...
  return 0;
}
```

Czym jest obsługa wyjątków

- Wyjątki w C++ są sposobem na oddzielenie wykrywania błędów od ich obsługi.
- Mechanizm obsługi wyjątków jest nielokalną strukturą sterującą, która korzysta ze *zwijania stosu*.

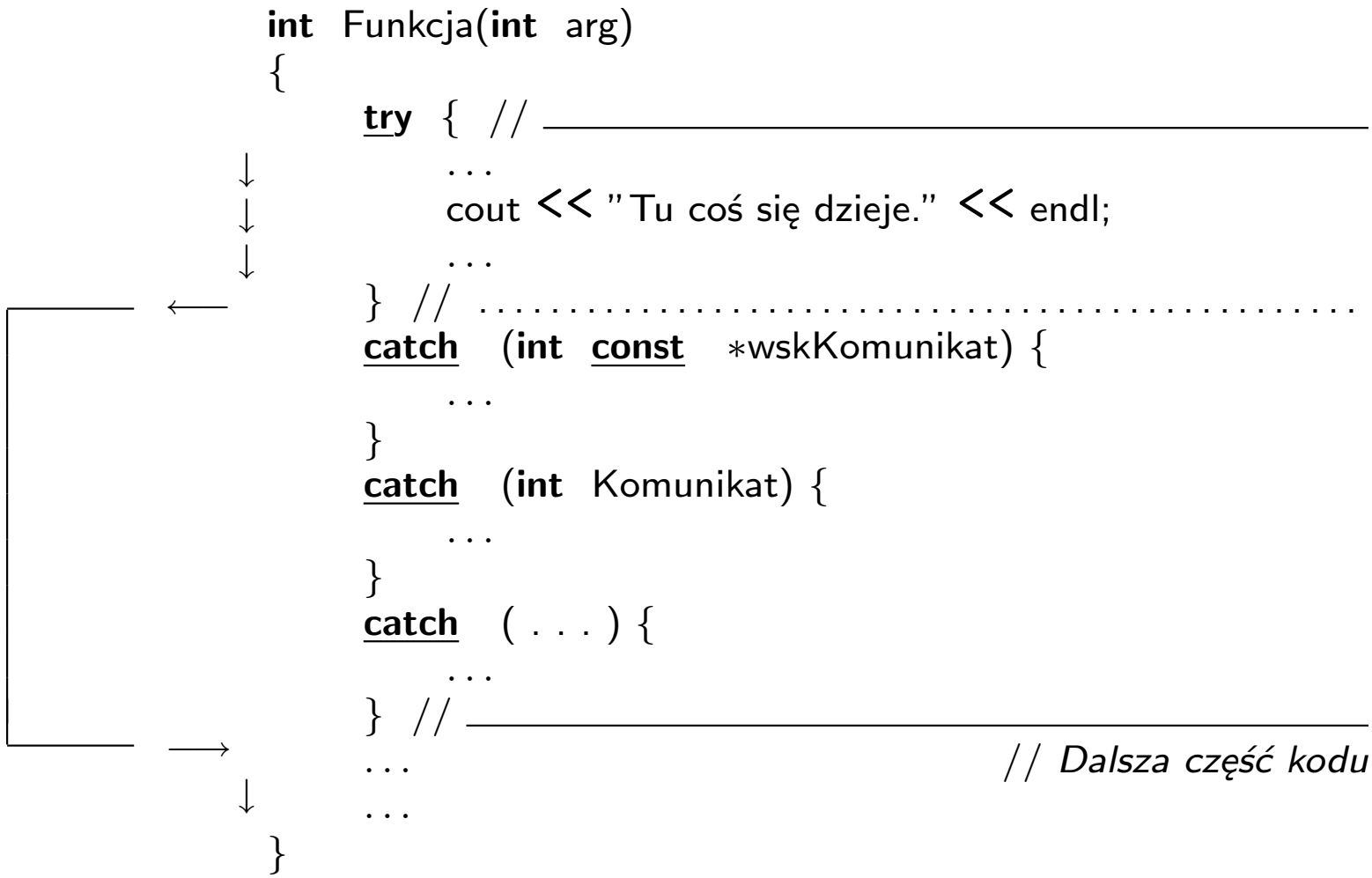
  Zwijaniem stosu nazywamy proces przeszukiwania stosu w celu znalezienia procedury obsługi wyjątku.
- Mechanizmy obsługi wyjątków służą do sygnalizowania i obsługi zdarzeń wyjątkowych. Jednak to programista musi zdecydować co to znaczy *wyjątkowy* w danym programie.
- Wyjątki często w sposób naturalny tworzą rodziny. Oznacza to, że dziedziczenie może być użyteczne w strukturalizacji wyjątków i ich obsłudze.

Plan prezentacji

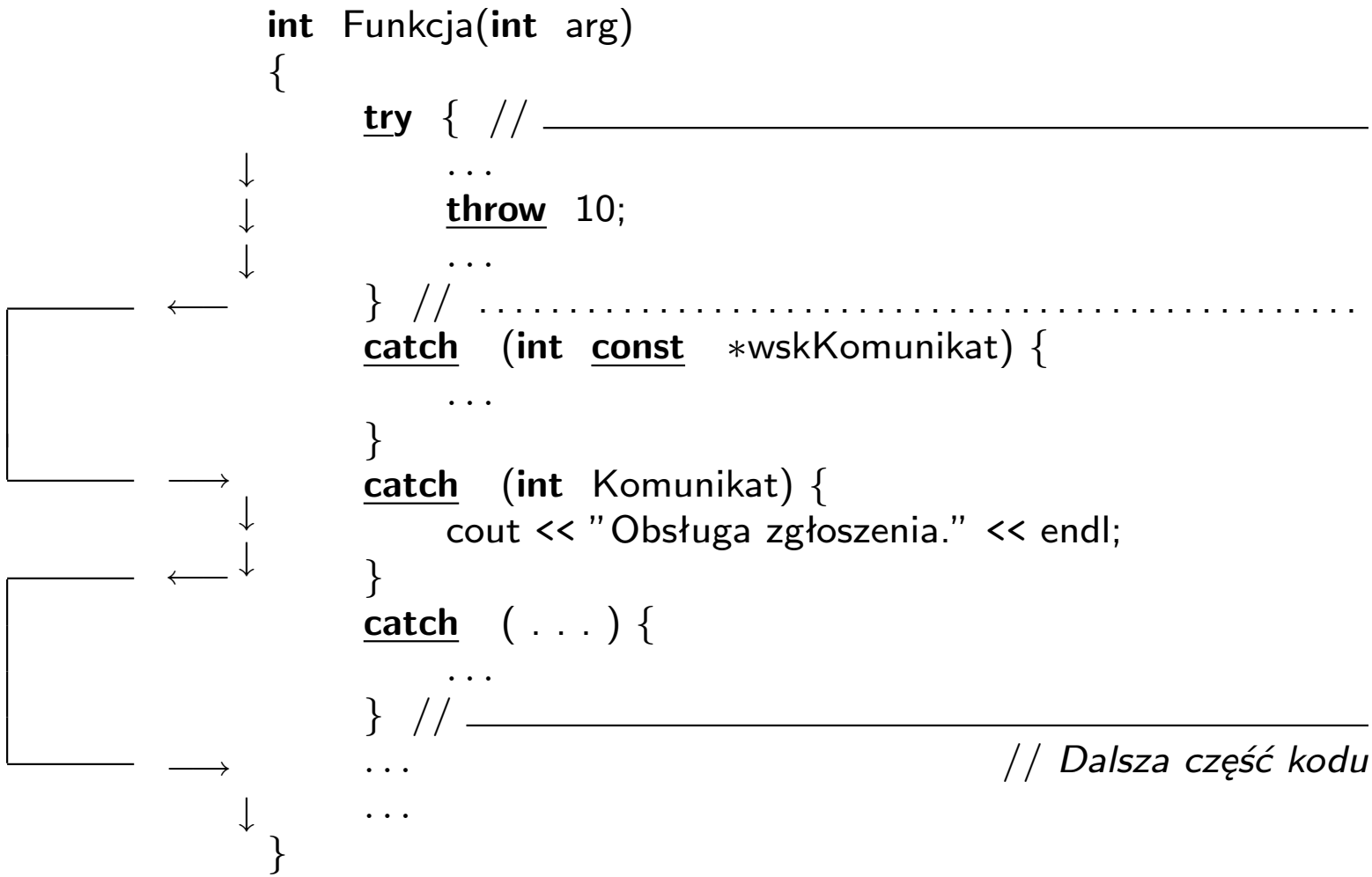
Obsługa wyjątków – trochę tradycyjnie

```
int Funkcja1( )
{
  . . .
  if (Gdy_cos_jest_zle) throw KOD_BLEDU;   // ..... Tutaj zgłaszamy wyjątek ←
  . . .
}
...
int Funkcja( )
{
  try { // ..... W tej sekcji zakładamy możliwość zgłoszeń wyjątki
    Funkcja1( );
    ...
  } // .....
  catch (int err) { // .. Tutaj rozpoczyna się przechwytywanie i obsługa wyjątków
    switch (err) {
      case KOD_BLEDU:     ...
                  if (Jezeli_nie_mozemy_sobie_poradzic) throw KOD_BLEDU;
                   ...
                  break ;
        ...
    }
  } // .....
   ...
```

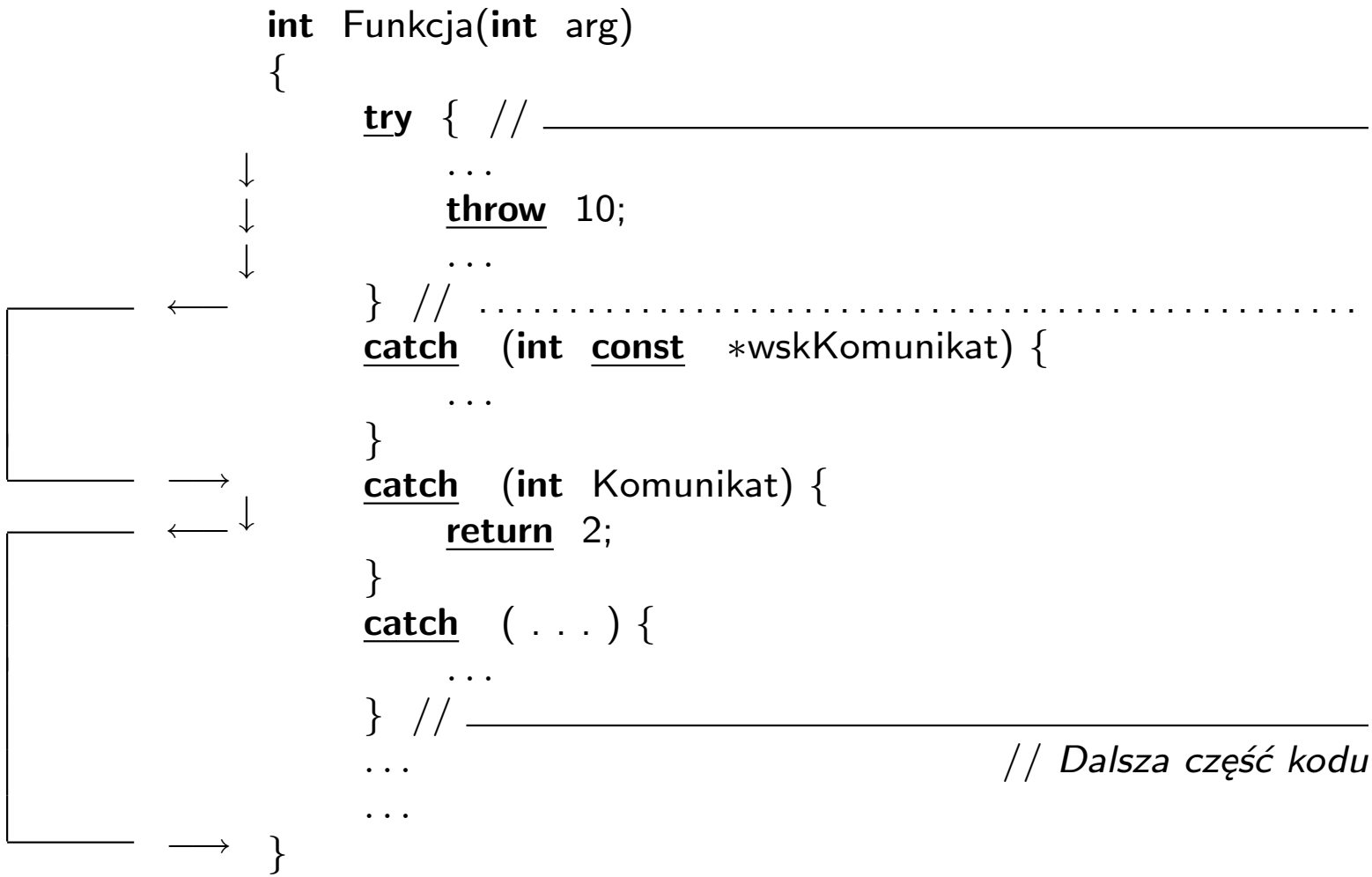
Przepływ sterowania – przypadek braku zgłoszenia wyjątku



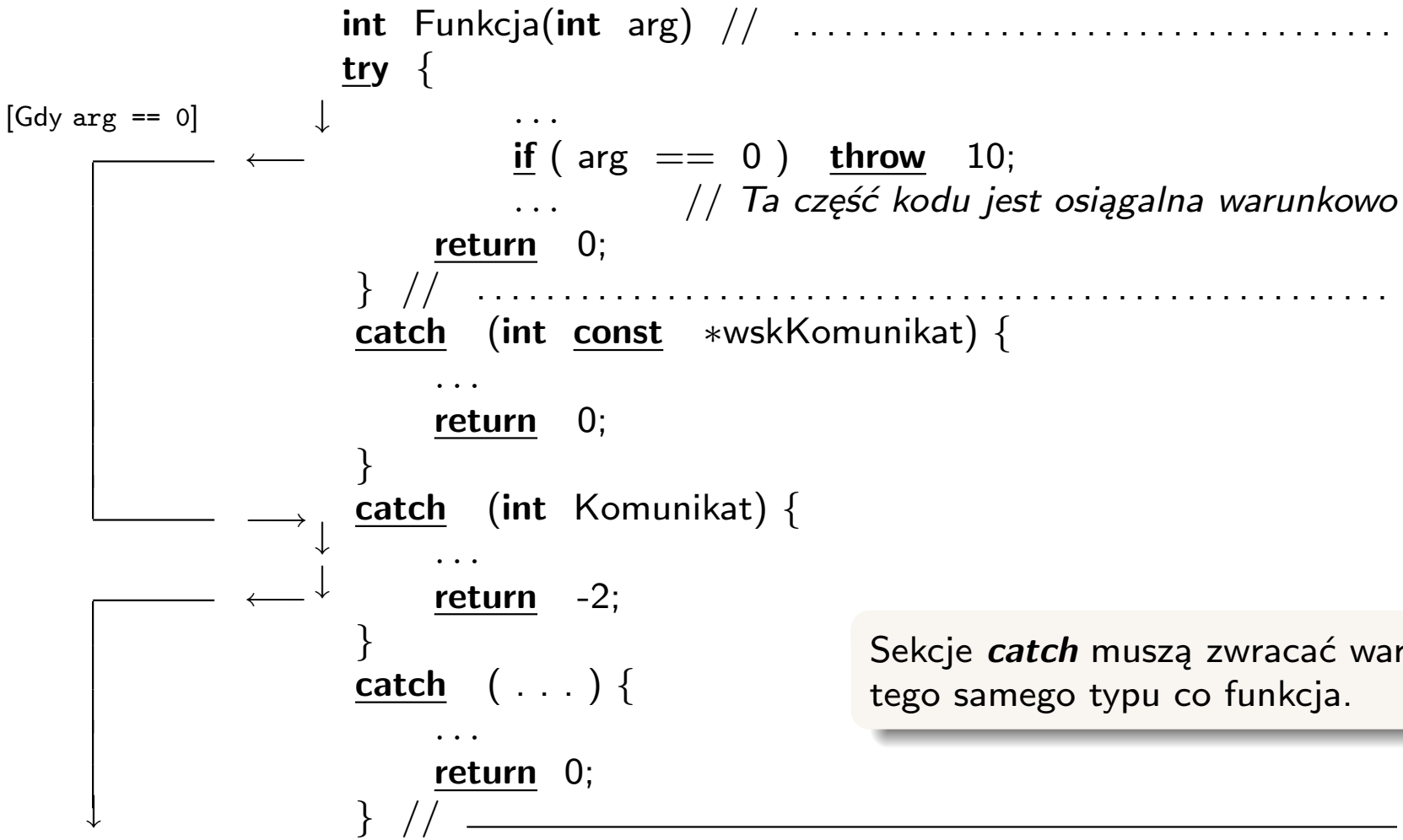
Przepływ sterowania – przypadek zgłoszenia wyjątku



Zgłoszenia wyjątku – wcześniejszy powrót z funkcji



Funkcja w sekcji **try**



Sekcje ***catch*** muszą zwracać wartość tego samego typu co funkcja.

Problem kopiowania zgłaszanej wartości

```
class Blad {
        Blad( Blad const & ) { }
    public :
        int _nr_bledu;
        Blad(int nr = 0) { _nr_bledu = nr; };
};

void Funkcja(int arg)
{
    try {
        ...
        throw Blad(3);
        ...
    }
    catch ( Blad &Bl ) {
        cout <<"Nr Bledu: " << Bl._nr_bledu << endl;
    }
    ...
}
```

Kopiowanie obiektu jest niemożliwe (poza tą klasą)

Problem kopiowania zgłaszanej wartości

```
class Blad {
        Blad( Blad const & ) { }
    public :
        int _nr_bledu;
        Blad(int nr = 0) { _nr_bledu = nr; };
};

void Funkcja(int arg)
{
    try {
        ...
        throw Blad(3);
        ...
    }
    catch ( Blad &Bl ) {
        cout <<"Nr Bledu: " << Bl._nr_bledu << endl;
    }
    ...
}
```

Kopiowanie obiektu jest niemożliwe (poza tą klasą)

Zgłoszenie wyjątku powoduje zawsze przekopiowanie parametru polecenia *throw*.

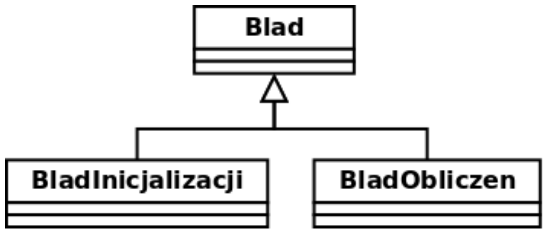
Ominięcie problemu kopiowania obiektu

```
class Blad {
        Blad( Blad const & ) { }
    public :
        int _nr_bledu;
        Blad(int nr = 0) { _nr_bledu = nr; };
};

void Funkcja(int arg)
{
    try {
        ...
        throw new Blad(3);
        ...
    }
    catch ( const Blad* wB ) {
        cout <<"Nr Bledu: " << B->_nr_bledu << endl;
        delete wB;
    }
    ...
}
```

Kopiowanie obiektu jest niemożliwe (poza tą klasą)

W tym przypadku kopiowany jest jedynie wskaźnik.

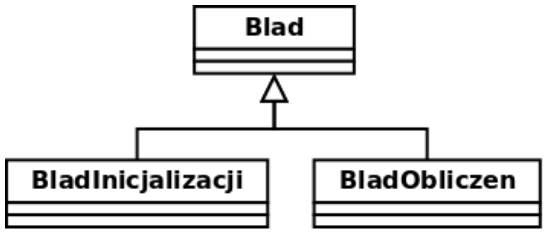
Hierarchia klas na potrzeby wyjątków




```
class Blad {
    public :
      virtual  const  char * Info() const  { return  "Blad ogolny"; }
      virtual  ~Blad() {}
};

class BladObliczen: public  Blad {
    public :
      virtual  const  char * Info() const  { return  "Blad obliczen"; }
};

class BladInicjalizacji: public  Blad {
    public :
      virtual  const  char * Info() const  { return  "Blad inicjalizacji"; }
};
```

Dziedziczenie pozwala na uszczegóławianie modeli pojęć (klas). Ten mechanizm wraz z niejawnym rzutowaniem *w górę* wykorzystywany w tworzeniu hierarchii wyjątków.

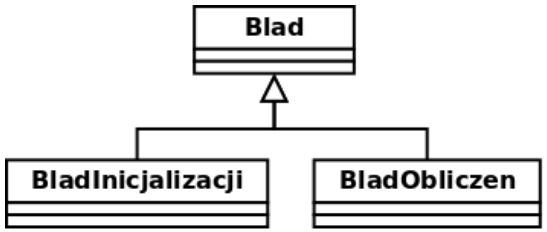
Niejawne rzutowaine *w górę*




```
void Test()
{
    void *Wsk;

    try {
        if (2.2+2.2+2.2 != 6.6) throw BladObliczen();
        cout << "OK" << endl;
    }
    catch (const BladObliczen &Ob) {
        cout << "P: " << Ob.Info() << endl;
    }
    catch (const Blad &Ob) {
        cout << "B: " << Ob.Info() << endl;
    }
}
```

W Sekcje **catch** przeglądane są od pierwszej do ostatniej. Dlatego też najpierw muszą pojawić się sekcje, których parametrami są obiekty klasy pochodnej.

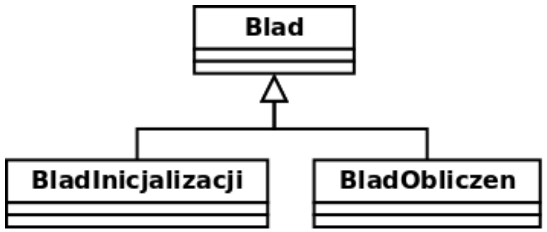
Niejawne rzutowaine *w górę*




```
void Test()
{
    void *Wsk;

    try {
        if (2.2+2.2+2.2 != 6.6) throw BladObliczen();
        cout << "OK" << endl;
    }
    catch (const BladObliczen &Ob) {
        cout << "P: " << Ob.Info() << endl;
    }
    catch (const Blad &Ob) {
        cout << "B: " << Ob.Info() << endl;
    }
}
```

A jaki będzie wynik działania tej funkcji?

Niejawne rzutowaine *w górę*



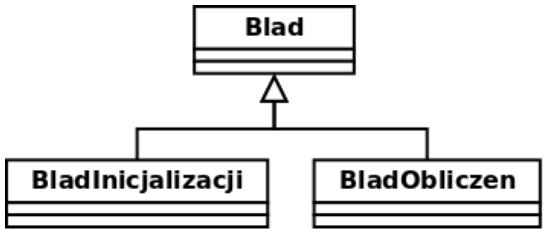

```
void Test()
{
    void *Wsk;

    try {
        if (2.2+2.2+2.2 != 6.6) throw BladObliczen();
        cout << "OK" << endl;
    }
    catch (const BladObliczen &Ob) {
        cout << "P: " << Ob.Info() << endl;
    }
    catch (const Blad &Ob) {
        cout << "B: " << Ob.Info() << endl;
    }
}
```

Wynik działania

P: Blad obliczen

Ze względu na skończoną reprezencję liczb proste działanie nie da, zdawałoby się, oczywistego wyniku. Zgłoszony zostanie wyjątek ***BladObliczen***.

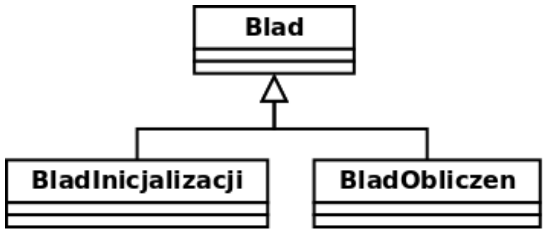
Niejawne rzutowaine *w górę*




```
void Test()
{
    void *Wsk;

    try {
        if (Wsk != nullptr) throw BladInicjalizacji();
        cout << "OK" << endl;
    }
    catch (const BladObliczen &Ob) {
        cout << "P: " << Ob.Info() << endl;
    }
    catch (const Blad &Ob) {
        cout << "B: " << Ob.Info() << endl;
    }
}
```

A dla tego przypadku jaki będzie wynik działania tej funkcji?

Niejawne rzutowaine *w górę*



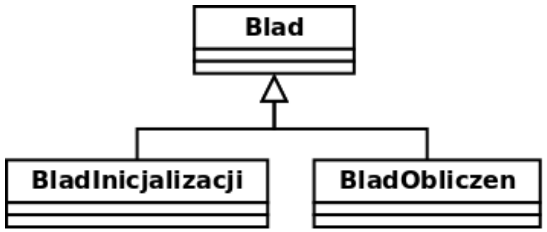

```
void Test()
{
    void *Wsk;

    try {
        if (Wsk != nullptr) throw BladInicjalizacji();
        cout << "OK" << endl;
    }
    catch (const BladObliczen &Ob) {
        cout << "P: " << Ob.Info() << endl;
    }
    catch (const Blad &Ob) {
        cout << "B: " << Ob.Info() << endl;
    }
}
```

Wynik działania

P: Blad inicjalizacji

Prawie na pewno zostanie zgłoszony wyjątek ***BladObliczen***, gdyż zmienne lokalne (automatyczne) nie są domyślnie inicjalizowane. Zgłoszony wyjątek zostanie przechwycony przez drugą sekcję dzięki niejawnemu rzutowaniu *w górę*.

Niejawne rzutowaine *w górę*




```
void Test()
{
    void *Wsk;

    try {
        if (Wsk != nullptr) throw BladInicjalizacji();
        cout << "OK" << endl;
    }
    catch (const BladObliczen &Ob) {
        cout << "P: " << Ob.Info() << endl;
    }
    catch (const Blad &Ob) {
        cout << "B: " << Ob.Info() << endl;
    }
}
```

Wynik działania

P: Blad inicjalizacji

Wynik działania funkcji zależy do tego jakie *śmieci* znajdą się w zmiennej `Wsk`. Może się zdarzyć, że będą same zera. Dlatego też nie można powiedzieć *na pewno* jaki będzie wynik działania funkcji dla tego przypadku.

Zwalnianie zasobów

```
void Funkcja(int arg)
{
    FILE *wPolecenie = popen("last -n 100","r");

    try {
        ...
        ...
    }
    catch ( ... ) {
        pclose(wProc);
        throw ;
    }
    pclose(wProc);
}
```

W przypadku zasobów, które nie mają charakteru lokalnego (np. dynamicznie przydzielana pamięć), należy pamiętać, aby zwalniać je w trakcie obsługi zgłoszonego wyjątku.

Wyjątki inaczej

```
#include <iostream>
#include <vector>
#include <stdexcept>      //Tu znajduje się definicje wyjątku std::out_of_range

int PrzegladajTablice( const std::vector<int> &Tablica )
{
    try {
        for (int ind = 0; ; ++ind) {
          if (Tablica.at(ind) / 2) std::cout << Tablica.at(++ind) << std::endl;
        }
    }
    catch (std::out_of_range) {
        return 0;
    }
    return 0;
}
```

Zgłoszenie wyjątku może być sposobem na przerwanie pętli w sytuacji, gdy bezpośrednie sprawdzanie warunku jest kłopotliwe w zapisie i sprawdzenie to musiałoby występować w wielu miejscach.
Zgłoszenie wyjątku może być pomocne również w przypadkach konstrukcji niestrukturalnych, w których wykorzystywano instrukcję **goto**.

Konwersje typów w obsłudze wyjątków

```
class Wektor2f { //.....
    public :
        ...
      Wektor2f( );
}; //.....

class LZespolona { //.....
    public :
        ...
    LZespolona( );
    LZespolona(const Wektor2f &W);
}; //.....

int main()
{
    try { throw Wektor2f( ); }
    catch (LZespolona Z) {          // Czy wyjątek zgłoszony powyżej zostanie tu obsłużony?
        ...
    }
}
```

Przy obsłudze wyjątków domyślne konwersje nie są realizowane.

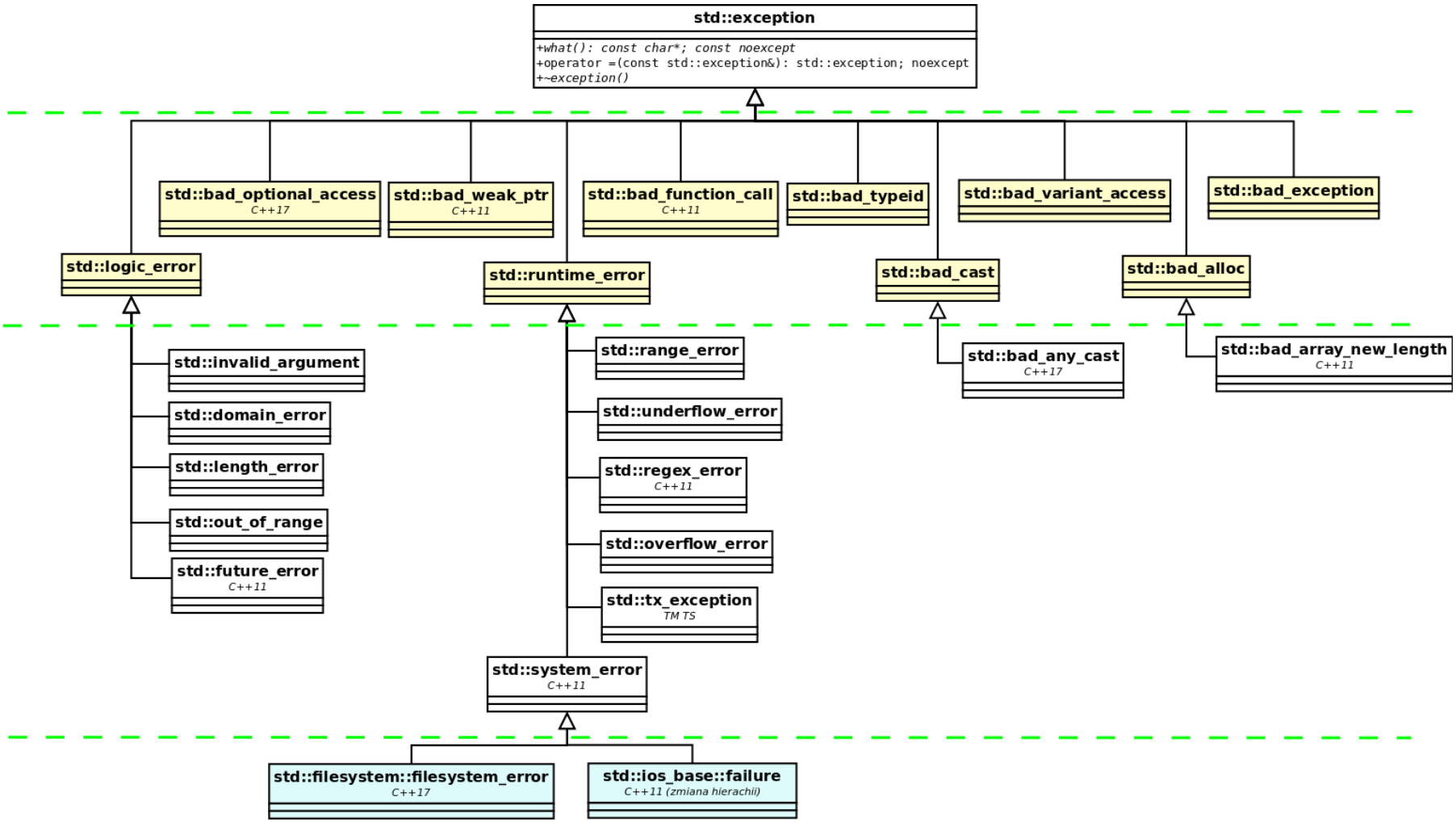
Plan prezentacji

Przykładowe wyjątki standardowe

Standardowe wyjątki zgłaszane przez język		
Nazwa	Zgłaszane przez	Nagłówek
bad_alloc	`new`	<new>
bad_cast	`dynamic_cast`	<typeinfo>
bad_typeid	`typeid`	<typeinfo>
bad_exception	`specyfikacja wyjątku`	<exception>

Przykłady standardowych wyjątków zgłaszanych przez bibliotekę STL		
Nazwa	Zgłaszane przez	Nagłówek
out_of_range	`at()`	<stdexcept>
	`bitset< >::operator []()`	<stdexcept>
invalid_argument	`konstruktor bitset`	<stdexcept>
overflow_error	`bitset< >::to_ulong()`	<stdexcept>
ios_base::failure	`ios base::clear()`	<ios>

Hierarchia wyjątków



Ustawianie handlera dla obsługi wyjątków

C++

unexpected_handler **set_unexpected**(unexpected_handler fun) throw();

C++11

unexpected_handler **set_unexpected**(unexpected_handler fun) noexcept;

Wyjątki dla operacji wejścia/wyjścia

```
int main ( ) {
    ifstream Plik;
    int Liczba;

    Plik.exceptions( ifstream ::failbit | ifstream ::badbit );
    try {
        Plik.open("zbior_liczba.txt");

        while (!Plik.eof()) Plik >> Liczba;
    }
    catch (ifstream ::failure) {
        cout << "Wyjątek operacji otwarcia lub czytania";
    }

    return 0;
}
```

Koniec prezentacji
Dziękuję za uwagę