

Stefan Sokołowski

PROGRAMOWANIE OBIEKTOWE II

wykład 7

Inst. Informatyki Stosowanej, ANS

Elbląg, 2024/2025

Wyjątki

Przerwanie normalnego wykonania programu:

```
throw ...;
```



Wyjątki

Przerwanie normalnego wykonania programu:

```
throw ...;
```

Najczęściej reakcja na błąd (i to jest **nie**ciekawe).



Wyjątki

Przerwanie normalnego wykonania programu:

```
throw ...;
```

Najczęściej reakcja na błąd (i to jest **nie**ciekawe).

„Rzucanie” wyjątków:

```
throw new Exception();
```



```
throw ...;
```

A detailed illustration of a grey and white wolf sitting on a dark, jagged rock. The wolf is shown in profile, facing left, with its head tilted back and mouth open in a howl. Behind the wolf is a large, bright, full moon with visible craters. The background is a dark, starry night sky. In the distance, there are dark, silhouetted mountains and a line of evergreen trees. The overall mood is mysterious and wild.

```
throw new Exception();
```

```
class WlasnyWyjatek extends Exception {
    public WlasnyWyjatek() {...}
}
```

• • • • •

```
throw new WlasnyWyjatek();
```

• • • • •

Wyjątki

Przerwanie normalnego wykonania programu:

```
throw ...;
```

Najczęściej reakcja na błąd (i to jest **nie**ciekawe).



„Rzucanie” wyjątków:

```
throw new Exception();
```

Własne klasy wyjątków:

```
class WlasnyWyjatek extends Exception {  
    public WlasnyWyjatek(...) {...}  
}
```

.

```
throw new WlasnyWyjatek(...);
```

.

parametr
wyjątku

Wyjątki

Przerwanie normalnego wykonania programu:

```
throw ...;
```

Najczęściej reakcja na błąd (i to jest **nie**ciekawe).



„Rzucanie” wyjątków:

```
throw new Exception();
```

Własne klasy wyjątków:

```
class WlasnyWyjatek extends Exception {  
    public WlasnyWyjatek(...) {...}  
}
```

```
...  
throw new WlasnyWyjatek(...);  
...
```

parametr
wyjątku

konstrukcja
obiektu

Wyjątki

Trochę jak break . . .

Wyjątki

Trochę jak break ...

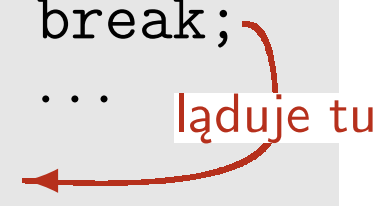
```
{  
    ...  
    break;  
    ...  
}
```

wyprowadza z pętli lub switch-a

Wyjątki

Trochę jak break ...

```
{  
  ...  
  break;  
  ...  
}
```



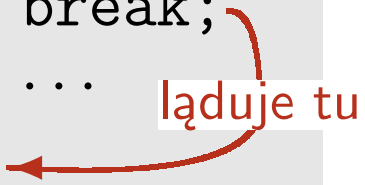
A red curved arrow originates from the `break;` statement and points to the closing curly brace `}` of the block. The text *ląduje tu* (lands here) is written in red next to the arrow.

wyprowadza z pętli lub switch-a

Wyjątki

Trochę jak break ...

```
{  
  ...  
  break;  
  ...  
}
```



lądaje tu

wyprowadza z pętli lub switch-a

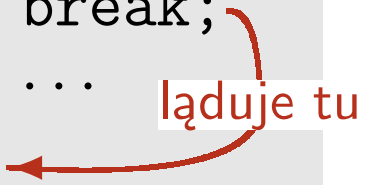
```
{  
  ...  
  throw ...;  
  ...  
}
```

wyprowadza z dowolnej komendy

Wyjątki

Trochę jak break ...

```
{  
  ...  
  break;  
  ...  
}
```



ląduje tu

wyprowadza z pętli lub switch-a

```
{  
  ...  
  throw ...;  
  ...  
}
```



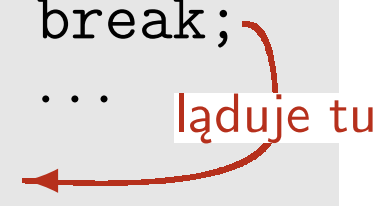
ucieka w dal

wyprowadza z dowolnej komendy

Wyjątki

Trochę jak break ...

```
{  
  ...  
  break;  
  ...  
}
```



A red arrow originates from the `break;` statement and points to the closing curly brace of the block, with the text "lądaje tu" (landing here) written in red next to the arrow.

wyprowadza z pętli lub switch-a

```
{  
  ...  
  throw ...;  
  ...  
}
```



A red arrow originates from the `throw ...;` statement and points downwards, away from the block, with the text "ucieka w dal" (escapes to the far) written in red next to the arrow.

wyprowadza z dowolnej komendy
(i daje się przechwycić)

Wyjątki

Przechwytywanie wyjątków:

```
{  
  ...  
  throw new MojWyjatek();  
  ...  
}
```

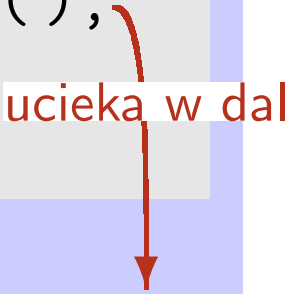
ucieka w dal



Wyjątki

Przechwytywanie wyjątków:

```
try
{
    ...
    throw new MojWyjatek();
    ...
}
catch (MojWyjatek ex) {
    ...
}
```



ucieka w dal

The diagram illustrates the flow of an exception. A red arrow originates from the `throw new MojWyjatek();` statement within the `try` block and points downwards to the `catch (MojWyjatek ex) {` block. The text "ucieka w dal" (escapes to the right) is written in red next to the arrow, indicating the direction of the exception's propagation.

Wyjątki

Przechwytywanie wyjątków:

```
try
{
    ...
    throw new MojWyjatek();
    ...
}
catch (MojWyjatek ex) {
    ...
}
```

ucieka w dal

łapiemy i mówimy, co robić

Gdzie stosuje się break-i?

Gdzie stosuje się break-i?

Zadanie:

Mamy przeczytać ciąg liczb i wydrukować ich iloczyn.

Gdzie stosuje się break-i?

Zadanie:

Mamy przeczytać ciąg liczb i wydrukować ich iloczyn.

```
. . . . .
int ile = czyt.nextInt(); ← ile będzie liczb
double ilocz=1.0;
for (int i=0; i<ile; i++) {
    double x = czyt.nextDouble();
    ilocz *= x;
}
System.out.println("iloczyn == " + ilocz);
```

Gdzie stosuje się break-i?

Zadanie:

Mamy przeczytać ciąg liczb i wydrukować ich iloczyn.

Ale przecież po przeczytaniu zera nie warto czytać dalej!

```
. . . . .
int ile = czyt.nextInt(); ← ile będzie liczb
double ilocz=1.0;
for (int i=0; i<ile; i++) {
    double x = czyt.nextDouble();
    ilocz *= x;
}
System.out.println("iloczyn == " + ilocz);
```

Gdzie stosuje się break-i?

Zadanie:

Mamy przeczytać ciąg liczb i wydrukować ich iloczyn.

Ale przecież po przeczytaniu zera nie warto czytać dalej!

```
. . . . .
int ile = czyt.nextInt(); ← ile będzie liczb
double ilocz=1.0;
for (int i=0; i<ile; i++) {
    double x = czyt.nextDouble();
    ilocz *= x;

}
System.out.println("iloczyn == " + ilocz);
```

Gdzie stosuje się break-i?

Zadanie:

Mamy przeczytać ciąg liczb i wydrukować ich iloczyn.

Ale przecież po przeczytaniu zera, nie warto czytać dalej!

```
. . . . .
int ile = czyt.nextInt(); ← ile będzie liczb
double ilocz=1.0;
for (int i=0; i<ile; i++) {
    double x = czyt.nextDouble();
    ilocz *= x;
    if (x == 0.0) break;
} ←
System.out.println("iloczyn == " + ilocz);
```

To samo wyjątkowo...

Zadanie:

Mamy przeczytać ciąg liczb i wydrukować ich iloczyn.

Ale przecież po przeczytaniu zera, nie warto czytać dalej!

```
. . . . .
int ile = czyt.nextInt(); ← ile będzie liczb
double ilocz=1.0;
for (int i=0; i<ile; i++) {
    double x = czyt.nextDouble();
    ilocz *= x;
    if (x == 0.0) throw new Zero();
}
System.out.println("iloczyn == " + ilocz);
```

To samo wyjątkowo...

Zadanie:

Mamy przeczytać ciąg liczb i wydrukować ich iloczyn.

Ale przecież po przeczytaniu zera, nie warto czytać dalej!

```
. . . . .
int ile = czyt.nextInt(); ← ile będzie liczb
double ilocz=1.0;
for (int i=0; i<ile; i++) {
    double x = czyt.nextDouble();
    ilocz *= x;
    if (x == 0.0) throw new Zero();
}
System.out.println("iloczyn == " + ilocz);
```

To samo wyjątkowo...

Zadanie:

Mamy przeczytać ciąg liczb i wydrukować ich iloczyn.

Ale przecież po przeczytaniu zera, nie warto czytać dalej!

```
. . . . .
int ile = czyt.nextInt(); ← ile będzie liczb
double ilocz=1.0;
    for (int i=0; i<ile; i++) {
        double x = czyt.nextDouble();
        ilocz *= x;
        if (x == 0.0) throw new Zero();
    }

System.out.println("iloczyn == " + ilocz);
```

To samo wyjątkowo...

Zadanie:

Mamy przeczytać ciąg liczb i wydrukować ich iloczyn.

Ale przecież po przeczytaniu zera, nie warto czytać dalej!

```
. . . . .
int ile = czyt.nextInt(); ← ile będzie liczb
double ilocz=1.0;
try { for (int i=0; i<ile; i++) {
    double x = czyt.nextDouble();
    ilocz *= x;
    if (x == 0.0) throw new Zero();
}
} catch (Zero ex) {} ←
System.out.println("iloczyn == " + ilocz);
```

To samo wyjątkowo...

Zadanie:

Mamy przeczytać ciąg liczb i wydrukować ich iloczyn.

Ale przecież po przeczytaniu zera, nie warto czytać dalej!

```
class Zero extends Exception { } ← własna klasa wyjątków
. . . . .
int ile = czyt.nextInt(); ← ile będzie liczb
double ilocz=1.0;
try { for (int i=0; i<ile; i++) {
    double x = czyt.nextDouble();
    ilocz *= x;
    if (x == 0.0) throw new Zero();
}
} catch (Zero ex) { } ←
System.out.println("iloczyn == " + ilocz);
```

Wyjście z pętli zagnieżdżonej

prosta komenda

`break`

Wyjście z pętli zagnieżdżonej

prosta komenda

`break`

skomplikowana komenda

```
class Nazwa extends Exception { }  
.  
.  
.  
try  
.  
.  
.  
throw new Nazwa( )  
.  
.  
.  
catch ( Nazwa ex ) { }
```

Wyjście z pętli zagnieżdżonej

prosta komenda
`break`

skomplikowana komenda

```
class Nazwa extends Exception { }  
.  
.  
.  
try  
.  
.  
throw new Nazwa(  
.  
.  
catch (Nazwa ex) { }
```

parametry?

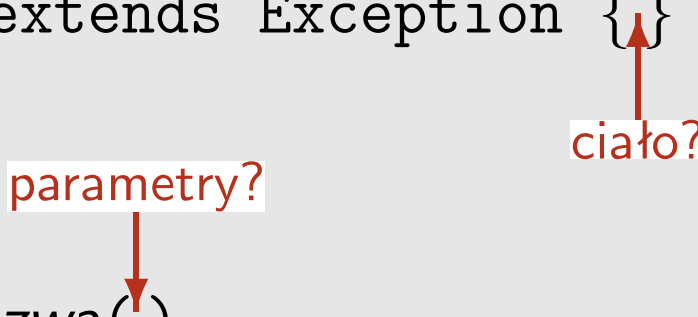


Wyjście z pętli zagnieżdżonej

prosta komenda
break

skomplikowana komenda

```
class Nazwa extends Exception {  
    . . . . .  
    try  
        . . . . .  
        throw new Nazwa()  
        . . . . .  
    catch (Nazwa ex) { }
```



parametry?

ciało?

Wyjście z pętli zagnieżdżonej

prosta komenda
break

skomplikowana komenda

```
class Nazwa extends Exception {  
    . . . . .  
    try  
        . . . . .  
        throw new Nazwa()  
        . . . . .  
    catch (Nazwa ex) {}
```

parametry?

ciało?

ciało?

Wyjście z pętli zagnieżdżonej

prosta komenda
break

skomplikowana komenda

```
class Nazwa extends Exception {  
    . . . . .  
    try  
        . . . . .  
        throw new Nazwa()  
        . . . . .  
    catch (Nazwa ex) {}  
}
```

parametry?

ciało?

ciało?

???

Wyjście z pętli zagnieżdżonej

prosta komenda
break

skomplikowana komenda

```
class Nazwa extends Exception {  
    .....  
    try  
        .....  
        throw new Nazwa()  
        .....  
    catch (Nazwa ex) {}  
}
```

parametry?

ciało?

ciało?

???

Na co tyle komplikacji?

Wyjście z pętli zagnieżdżonej

Na razie break jest prostszy od wyjątku.

Wyjście z pętli zagnieżdżonej

Na razie break jest prostszy od wyjątku. Ale:

Zadanie:

Mamy sprawdzić, czy 2-wymiarowa tablica liczb spełnia warunki

Wyjście z pętli zagnieżdżonej

Na razie break jest prostszy od wyjątku. Ale:

Zadanie:

Mamy sprawdzić, czy 2-wymiarowa tablica liczb spełnia warunki:

- (1) w każdym wierszu jest przynajmniej jedno zero

Wyjście z pętli zagnieżdżonej

Na razie break jest prostszy od wyjątku. Ale:

Zadanie:

Mamy sprawdzić, czy 2-wymiarowa tablica liczb spełnia warunki:

- (1) w każdym wierszu jest przynajmniej jedno zero;
- (2) w żadnym wierszu liczba ujemna nie może wystąpić przed tym zerem.

Wyjście z pętli zagnieżdżonej

Na razie break jest prostszy od wyjątku. Ale:

Zadanie:

Mamy sprawdzić, czy 2-wymiarowa tablica liczb spełnia warunki:

- (1) w każdym wierszu jest przynajmniej jedno zero;
 - (2) w żadnym wierszu liczba ujemna nie może wystąpić przed tym zerem.
-

Jeśli w wierszu znajdziemy zero, to **nie trzeba tego wiersza dalej przeglądać** — przejść do następnego.

Wyjście z pętli zagnieżdżonej

Na razie break jest prostszy od wyjątku. Ale:

Zadanie:

Mamy sprawdzić, czy 2-wymiarowa tablica liczb spełnia warunki:

- (1) w każdym wierszu jest przynajmniej jedno zero;
 - (2) w żadnym wierszu liczba ujemna nie może wystąpić przed tym zerem.
-

Jeśli w wierszu znajdziemy zero, to **nie trzeba tego wiersza** dalej przeglądać — przejść do następnego.

Jeśli natrafimy na liczbę ujemną, to **nie trzeba już niczego** przeglądać — warunek (2) niespełniony.

Wyjście z pętli zagnieżdżonej

Na razie break jest prostszy od wyjątku. Ale:

Zadanie:

Mamy sprawdzić, czy 2-wymiarowa tablica liczb spełnia warunki:

- (1) w każdym wierszu jest przynajmniej jedno zero;
 - (2) w żadnym wierszu liczba ujemna nie może wystąpić przed tym zerem.
-

Jeśli w wierszu znajdziemy zero, to **nie trzeba tego wiersza** dalej przeglądać — przejść do następnego.

Jeśli natrafimy na liczbę ujemną, to **nie trzeba już niczego** przeglądać — warunek (2) niespełniony.

Jeśli dojdziemy do końca wiersza, to **nie trzeba już niczego** przeglądać — warunek (1) niespełniony.

Wyjście z pętli zagnieżdżonej

Zadanie:

Mamy sprawdzić, czy 2-wymiarowa tablica liczb spełnia warunki:

- (1) w każdym wierszu jest przynajmniej jedno zero;
- (2) w żadnym wierszu liczba ujemna nie może wystąpić przed tym zerem.

Wyjście z pętli zagnieżdżonej

Zadanie:

Mamy sprawdzić, czy 2-wymiarowa tablica liczb spełnia warunki:

- (1) w każdym wierszu jest przynajmniej jedno zero;
- (2) w żadnym wierszu liczba ujemna nie może wystąpić przed tym zerem.

- pętla **po wierszach** i wewnątrz pętla **po kolumnach**

Wyjście z pętli zagnieżdżonej

Zadanie:

Mamy sprawdzić, czy 2-wymiarowa tablica liczb spełnia warunki:

- (1) w każdym wierszu jest przynajmniej jedno zero;
- (2) w żadnym wierszu liczba ujemna nie może wystąpić przed tym zerem.

- pętla **po wierszach** i wewnątrz pętla **po kolumnach**
- po napotkaniu zera opuścić **wewnętrzną pętlę** i przejść do następnego wiersza

Wyjście z pętli zagnieżdżonej

Zadanie:

Mamy sprawdzić, czy 2-wymiarowa tablica liczb spełnia warunki:

- (1) w każdym wierszu jest przynajmniej jedno zero;
- (2) w żadnym wierszu liczba ujemna nie może wystąpić przed tym zerem.

- pętla **po wierszach** i wewnątrz pętla **po kolumnach**
- po napotkaniu zera opuścić **wewnętrzną pętlę** i przejść do następnego wiersza
- po napotkaniu ujemnej opuścić **zewnętrzną pętlę** i zasygnalizować błąd

Wyjście z pętli zagnieżdżonej

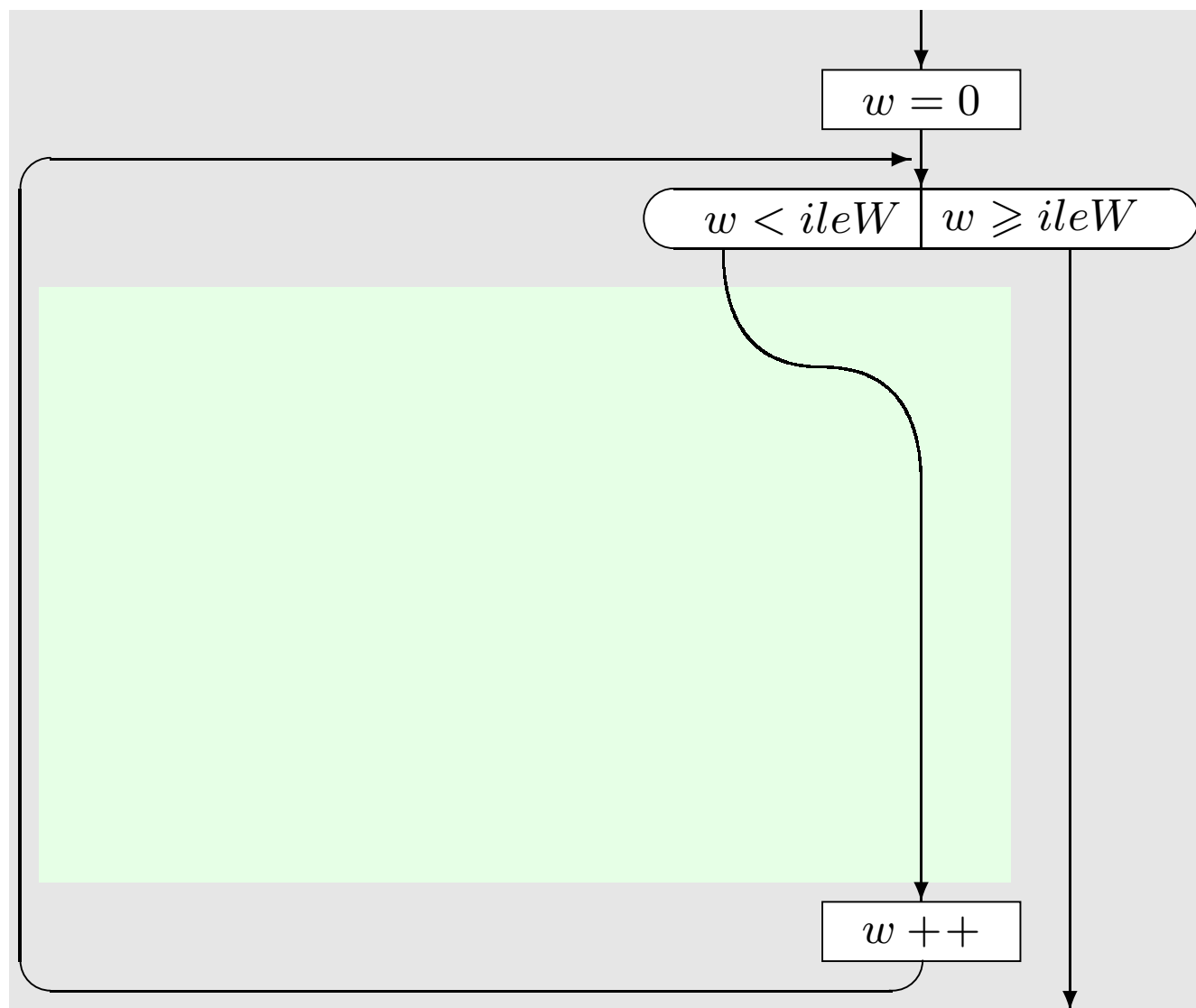
Zadanie:

Mamy sprawdzić, czy 2-wymiarowa tablica liczb spełnia warunki:

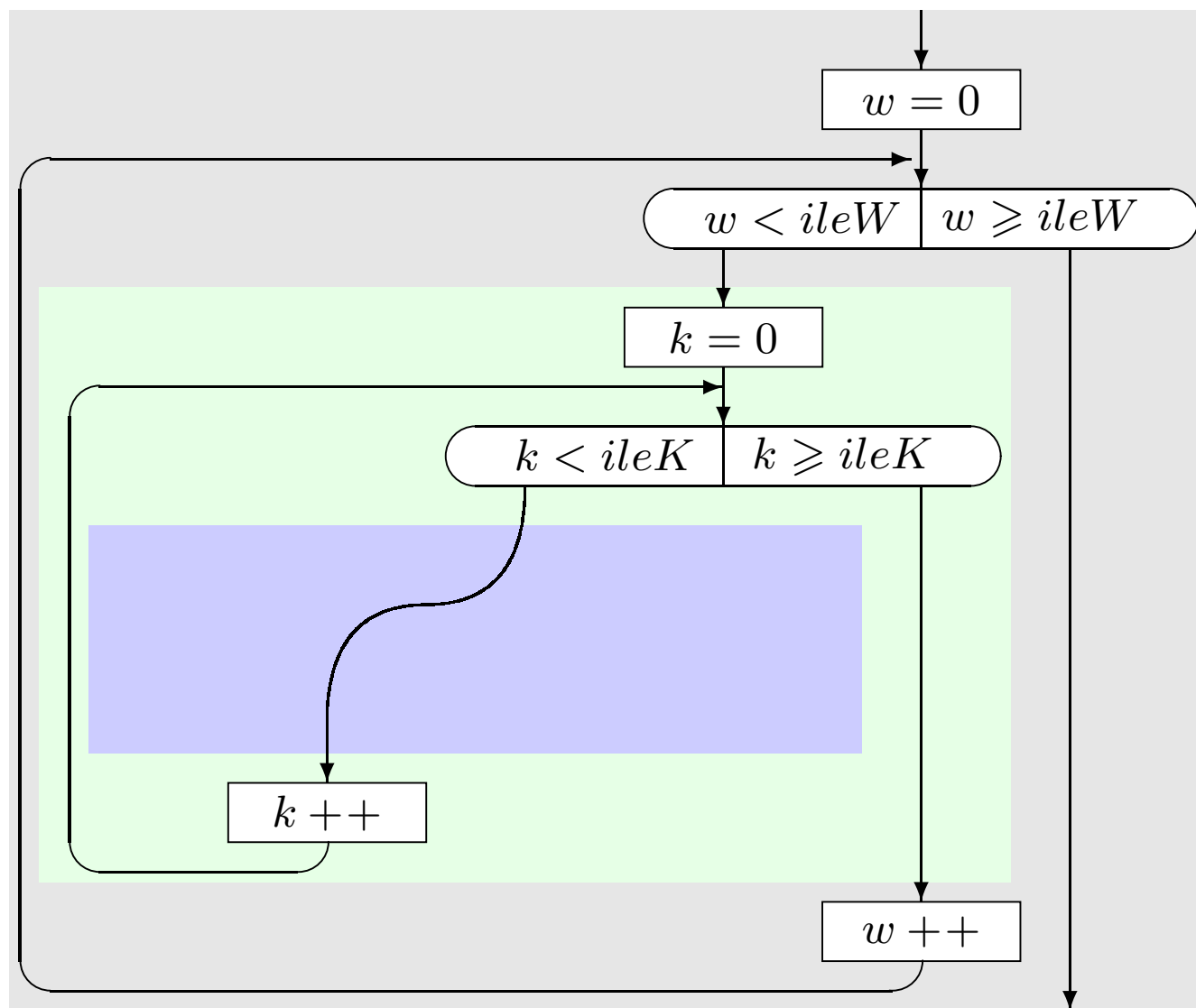
- (1) w każdym wierszu jest przynajmniej jedno zero;
- (2) w żadnym wierszu liczba ujemna nie może wystąpić przed tym zerem.

- pętla **po wierszach** i wewnątrz pętla **po kolumnach**
- po napotkaniu zera opuścić **wewnętrzną pętlę** i przejść do następnego wiersza
- po napotkaniu ujemnej opuścić **zewnętrzną pętlę** i zasygnalizować błąd
- po dojściu do końca wiersza opuścić **zewnętrzną pętlę** i zasygnalizować błąd

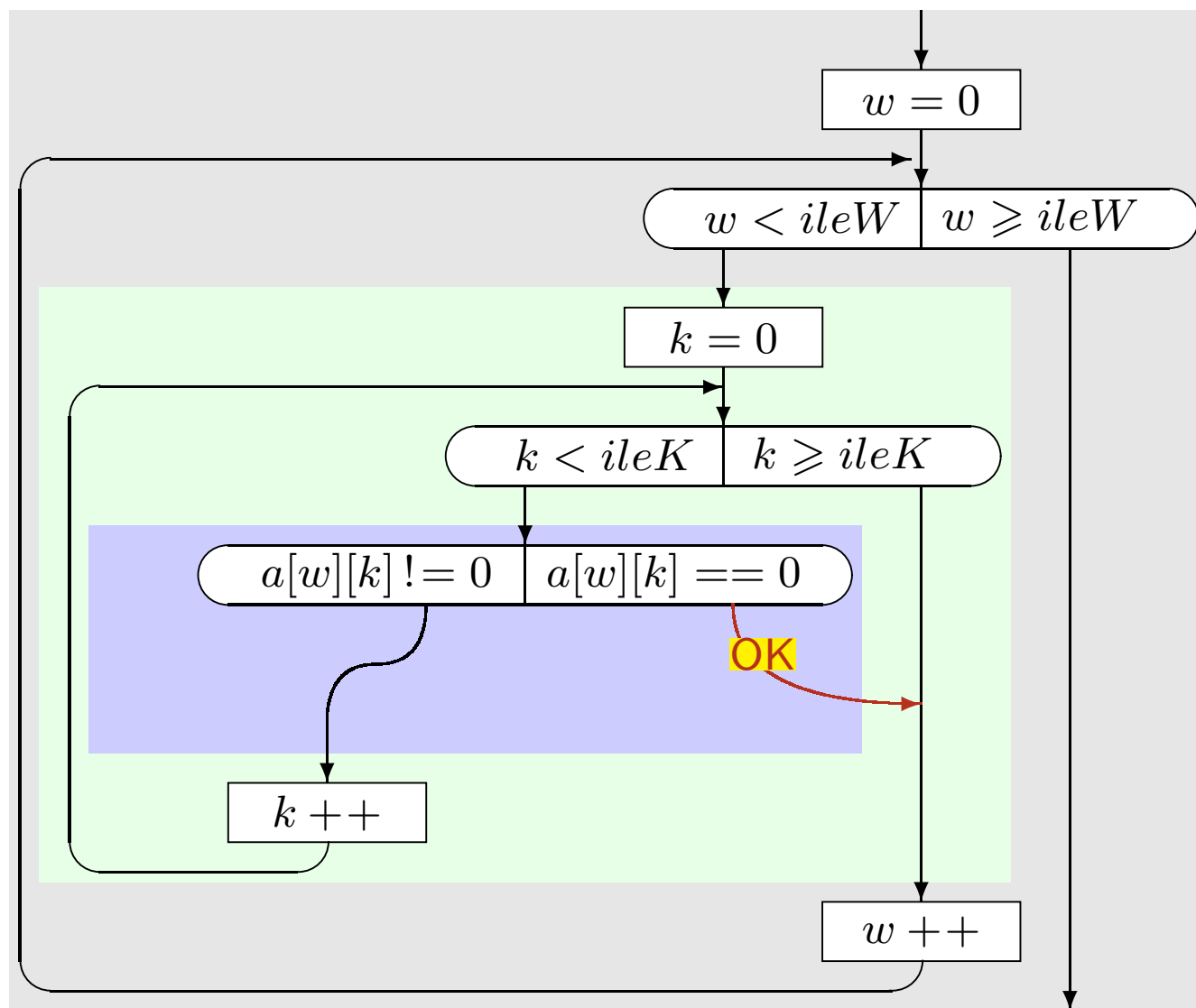
Wyjście z pętli zagnieżdżonej



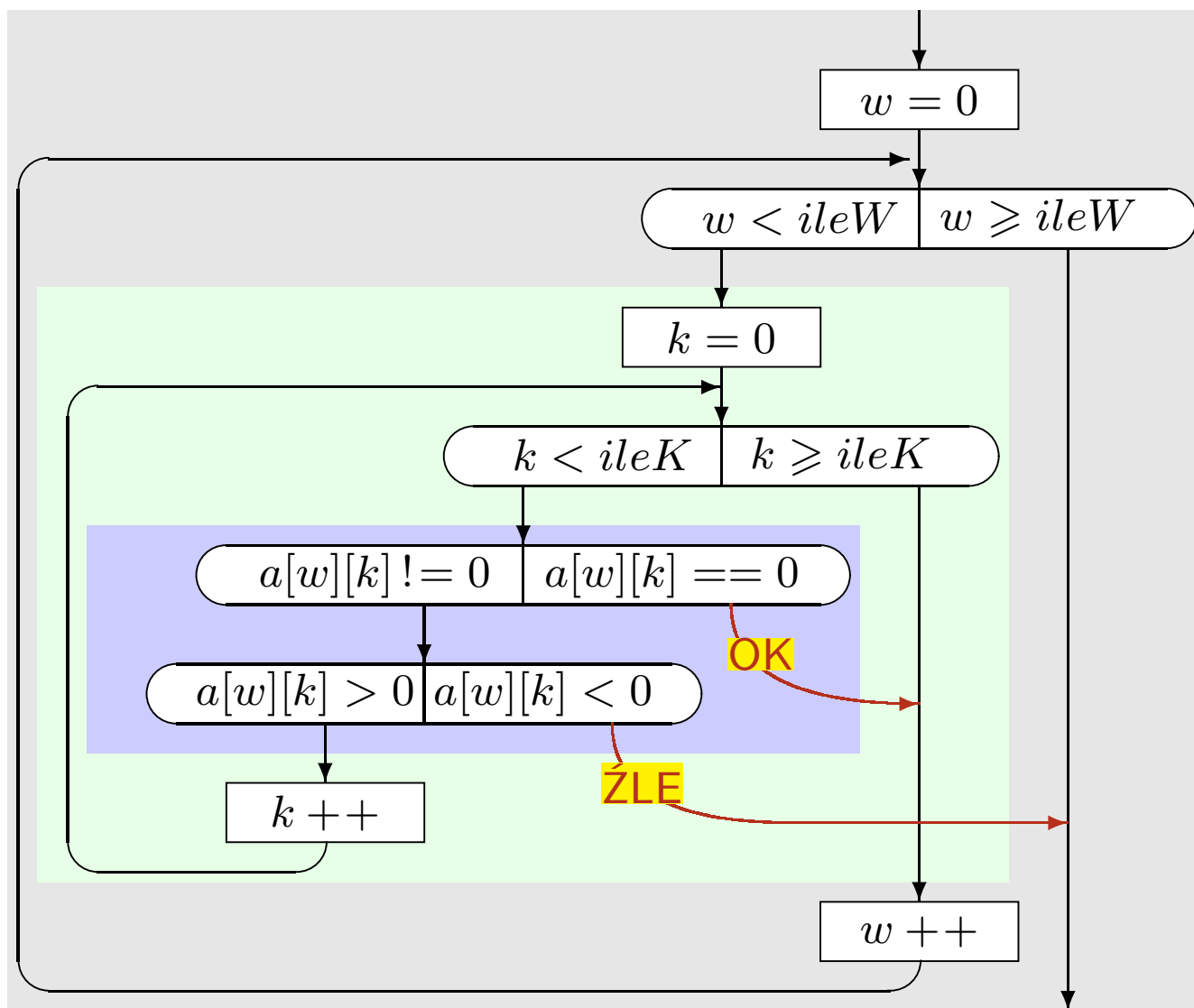
Wyjście z pętli zagnieżdżonej



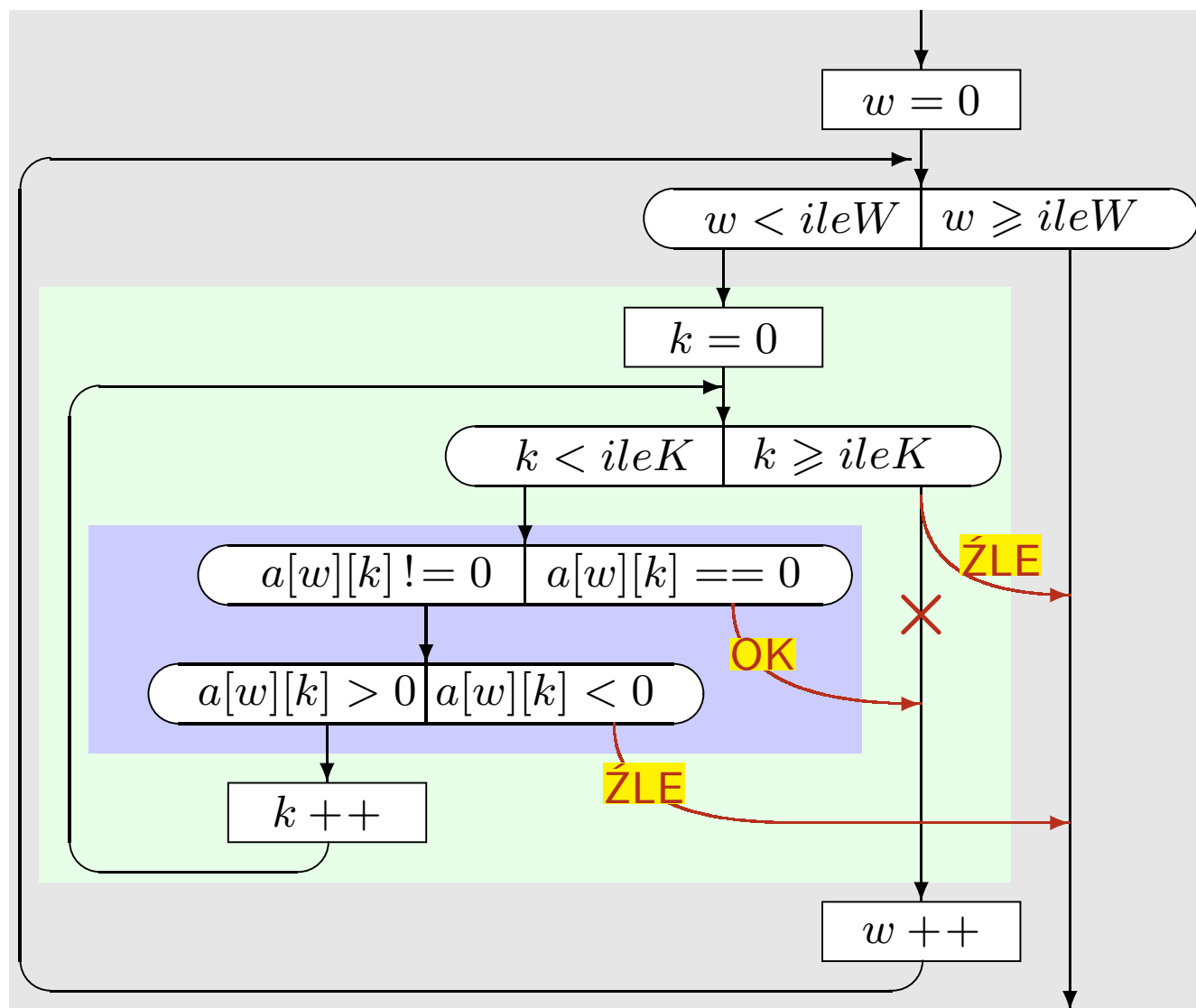
Wyjście z pętli zagnieżdżonej



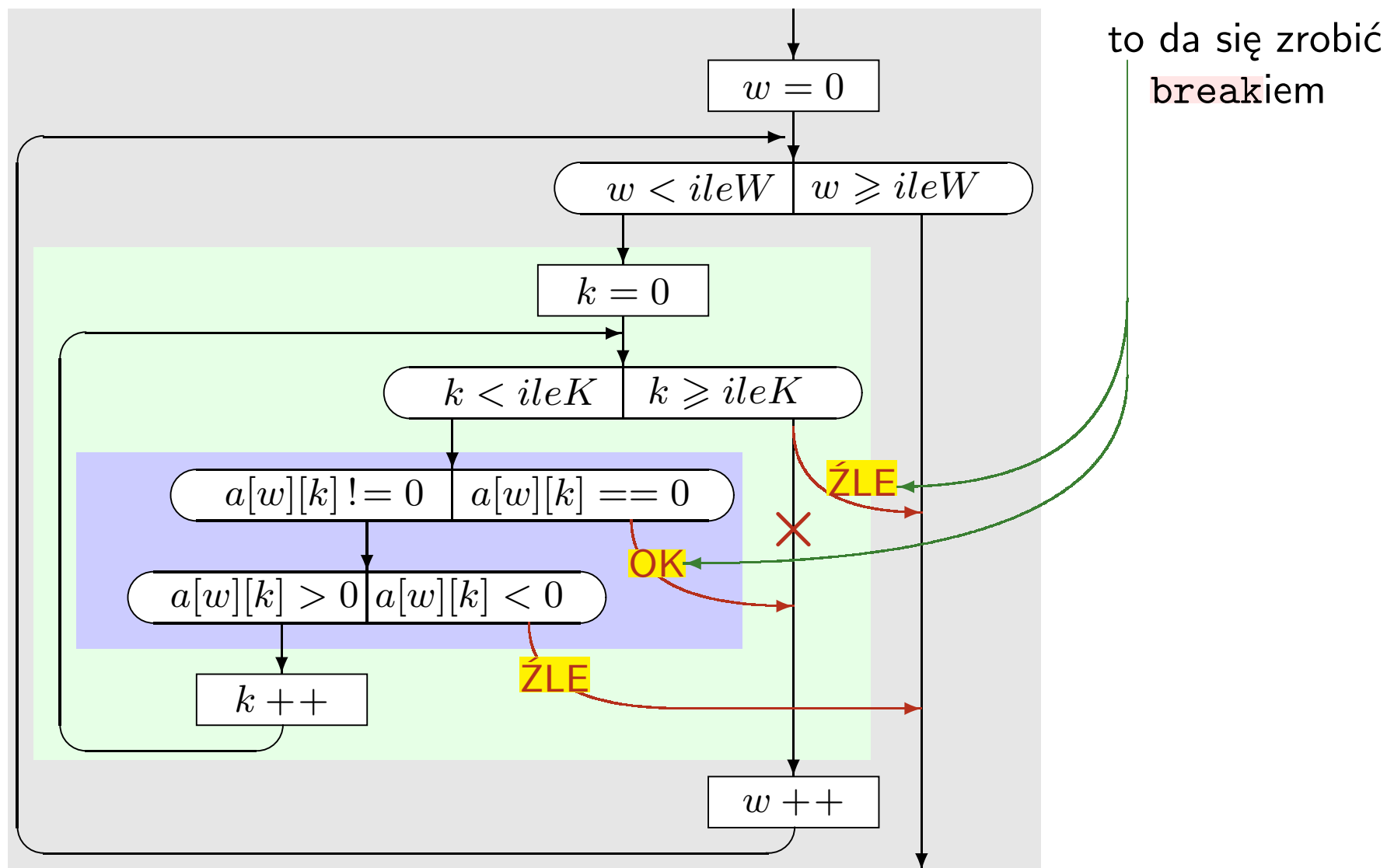
Wyjście z pętli zagnieżdżonej



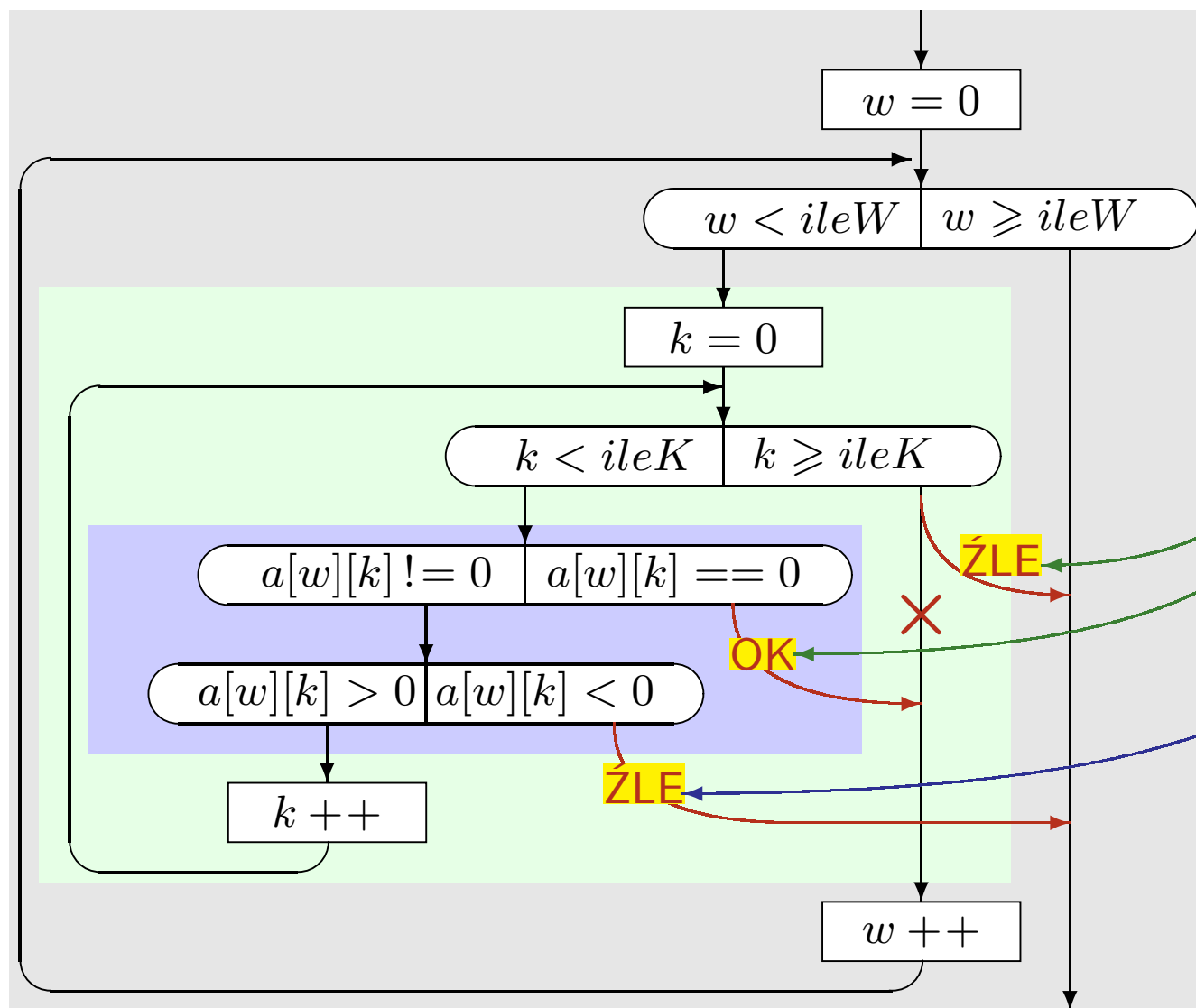
Wyjście z pętli zagnieżdżonej



Wyjście z pętli zagnieżdżonej



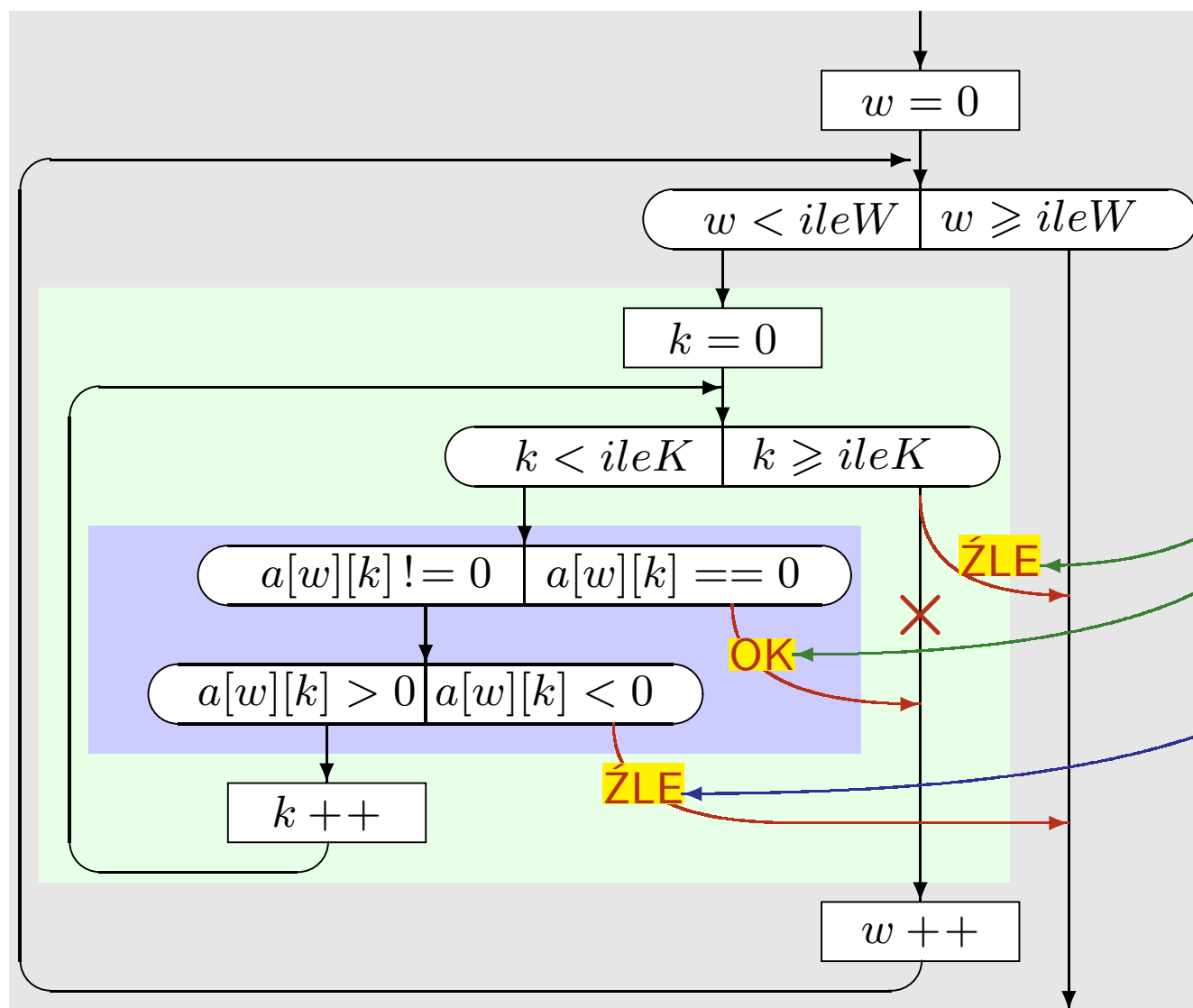
Wyjście z pętli zagnieżdżonej



to da się zrobić
breakiem

tęgo nie da się zrobić
breakiem

Wyjście z pętli zagnieżdżonej

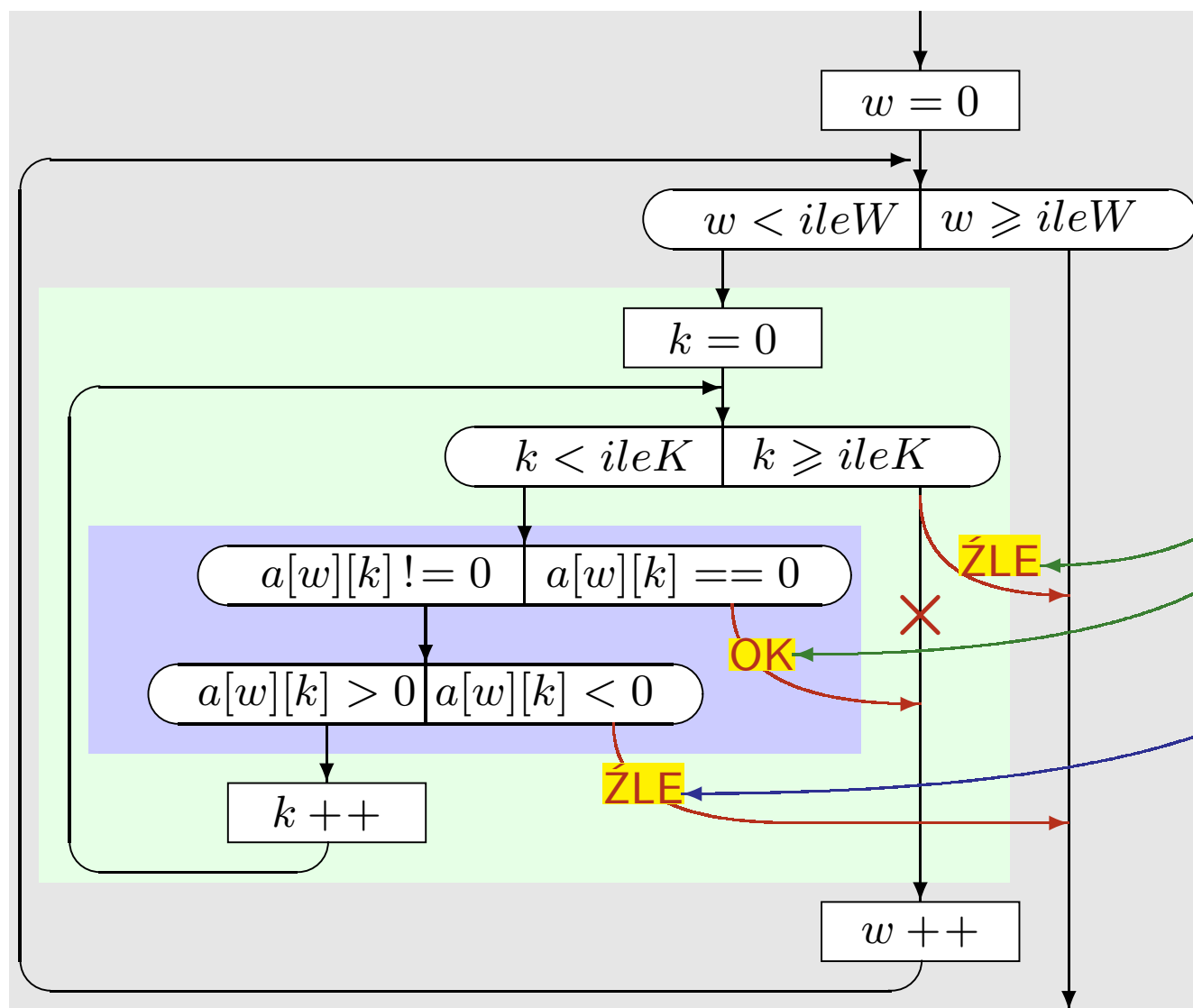


to da się zrobić
breakiem

tęgo nie da się zrobić
breakiem

Ale i w tym, co da się zrobić breakiem,
trzeba jeszcze po
wyjściu jakoś
zdecydować, czy
OK, czy ZŁE

Wyjście z pętli zagnieżdżonej



to da się zrobić
breakiem

tego nie da się zrobić
breakiem

Ale i w tym, co da
się zrobić breakiem,
trzeba jeszcze po
wyjściu jakoś
zdecydować, czy
OK, czy ŹLE —

zmienne logiczne?

Pola do wypełniania w wyjątkach

iis.ans-elblag.pl/~stefan/Dydaktyka/ProgObiII/Prog07/P0_Zera/W1.java

Pola do wypełniania w wyjątkach

iis.ans-elblag.pl/~stefan/Dydaktyka/ProgObiIII/Prog07/P0_Zera/W1.java

```
catch (BezZera ex) {  
    System.out.println("\n WIERSZ BEZ ZER\n");  
    System.exit(0);  
}
```

— przy przechwytywaniu wyjątku można określić reakcję na ten wyjątek.

Pola do wypełniania w wyjątkach

iis.ans-elblag.pl/~stefan/Dydaktyka/ProgObiII/Prog07/P0_Zera/W1.java

```
catch (BezZera ex) {  
    System.out.println("\n WIERSZ BEZ ZER\n");  
    System.exit(0);  
}
```

— przy przechwytywaniu wyjątku można określić reakcję na ten wyjątek.

Wykonanie komendy `throw ...` powoduje wyjście ze wszystkich bloków, przy których nie ma odpowiedniego `catch ...`.

Pola do wypełniania w wyjątkach

iis.ans-elblag.pl/~stefan/Dydaktyka/ProgObiIII/Prog07/P0_Zera/W1.java

```
catch (BezZera ex) {  
    System.out.println("\n WIERSZ BEZ ZER\n");  
    System.exit(0);  
}
```

— przy przechwytywaniu wyjątku można określić reakcję na ten wyjątek.

Wykonanie komendy `throw ...` powoduje wyjście ze wszystkich bloków, przy których nie ma odpowiedniego `catch ...`.

Jeśli nie trafi się odpowiedni `catch ...`, to program kończy działanie z sygnałem błędu wykonania.

Pola do wypełniania w wyjątkach

iis.ans-elblag.pl/~stefan/Dydaktyka/ProgObiIII/Prog07/P0_Zera/W1.java

```
catch (BezZera ex) {  
    System.out.println("\n WIERSZ BEZ ZER\n");  
    System.exit(0);  
}
```

— przy przechwytywaniu wyjątku można określić reakcję na ten wyjątek.

Wykonanie komendy `throw ...` powoduje wyjście ze wszystkich bloków, przy których nie ma odpowiedniego `catch ...`.

Jeśli nie trafi się odpowiedni `catch ...`, to program kończy działanie z sygnałem błędu wykonania.

Jeśli znajdzie się odpowiedni `catch ...`, to wykonuje się wskazana przy nim **komenda**, a potem komendy po tym `catch-u`.

Pola do wypełniania w wyjątkach

iis.ans-elblag.pl/~stefan/Dydaktyka/ProgObiII/Prog07/P0_Zera/W2.java

Pola do wypełniania w wyjątkach

iis.ans-elblag.pl/~stefan/Dydaktyka/ProgObiII/Prog07/P0_Zera/W2.java

```
class BezZera extends Exception {
    public int w;
    public BezZera(int w) { this.w=w; }
}
. . . . .
. . . throw new BezZera(w);
. . . . .
catch (BezZera ex) {
    System.out.println(
        "\n WIERSZ BEZ ZER: tab["+ex.w+"]\n"
    );
    System.exit(0);
}
```

```
class BezZera extends Exception {
    public int w;
    public BezZera(int w) { this.w=w; }
}

. . . . .
. . . throw new BezZera(w);
. . . . .

catch (BezZera ex) {
    System.out.println(
        "\n WIERSZ BEZ ZER: tab["+ex.w+"]\n"
    );
    System.exit(0);
}
```

powstaje obiekt
klasy BezZera

```
class BezZera extends Exception {
    public int w;
    public BezZera(int w) { this.w=w; }
}
```

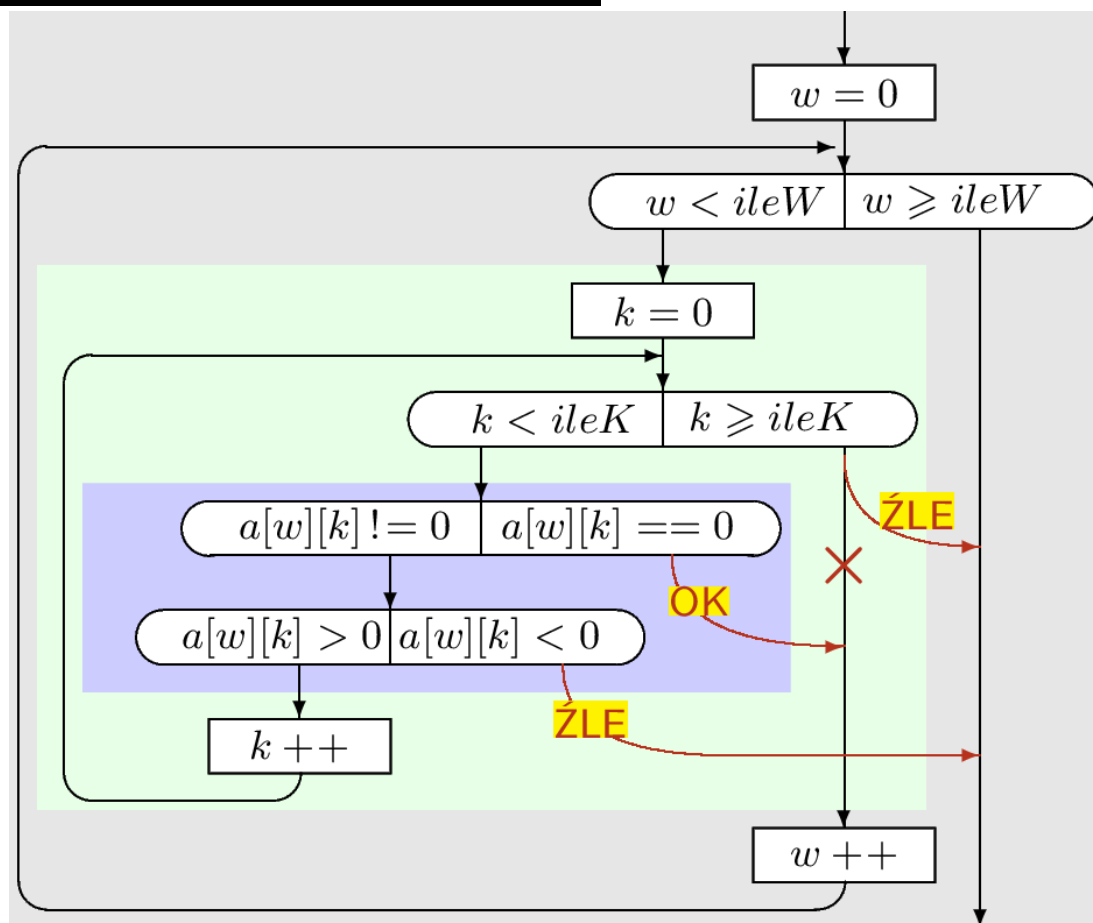
```

. . . . .
. . . throw new BezZera(w);
. . . . .
catch (BezZera ex) {

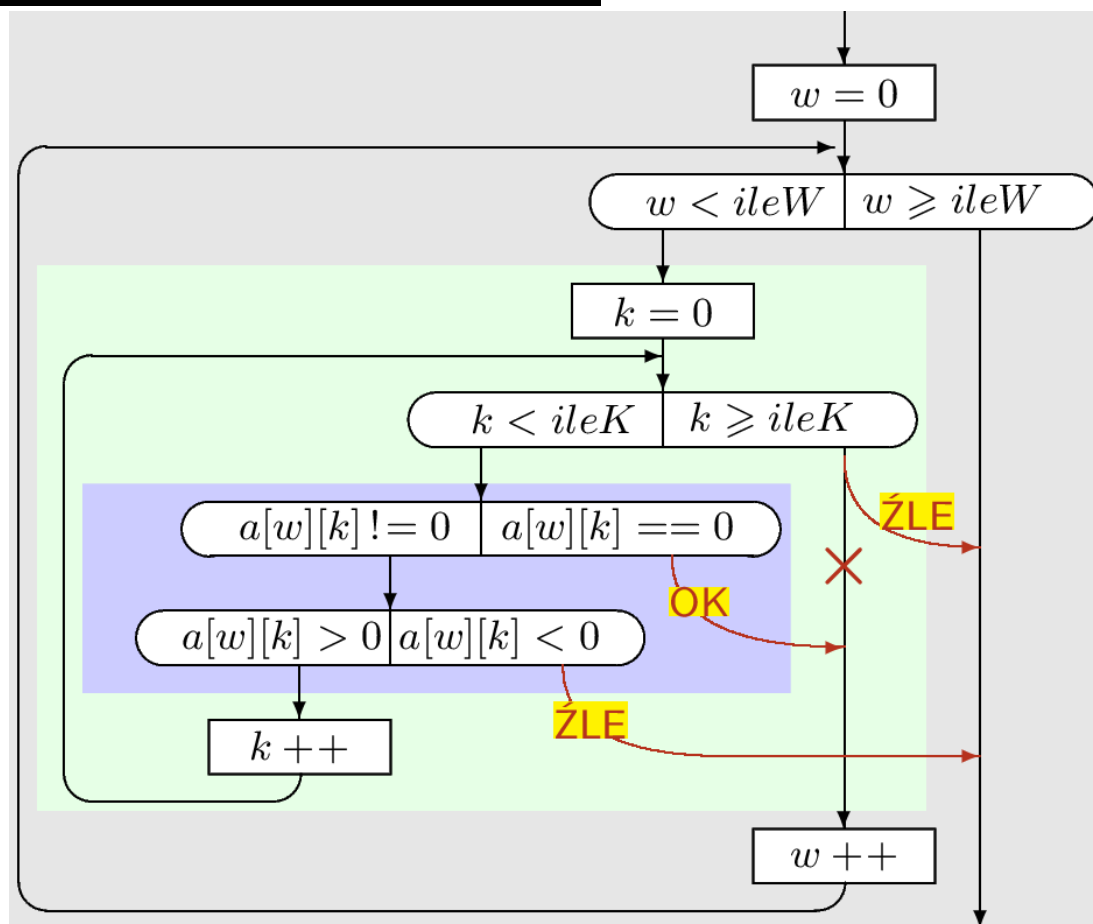
```

callback wywoływany na tym obiekcie
po „rzuceniu” wyjątku

Modularyzacja z wyjątkami

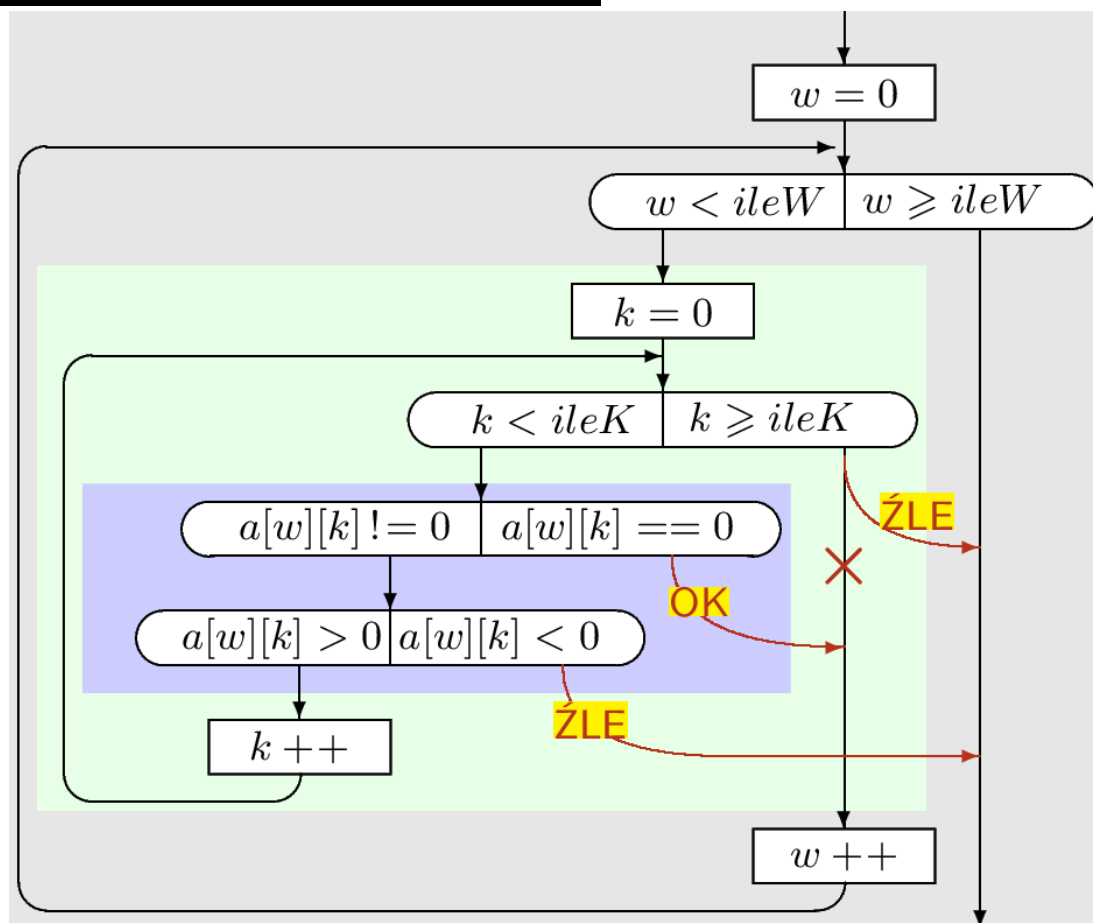


Modularyzacja z wyjątkami



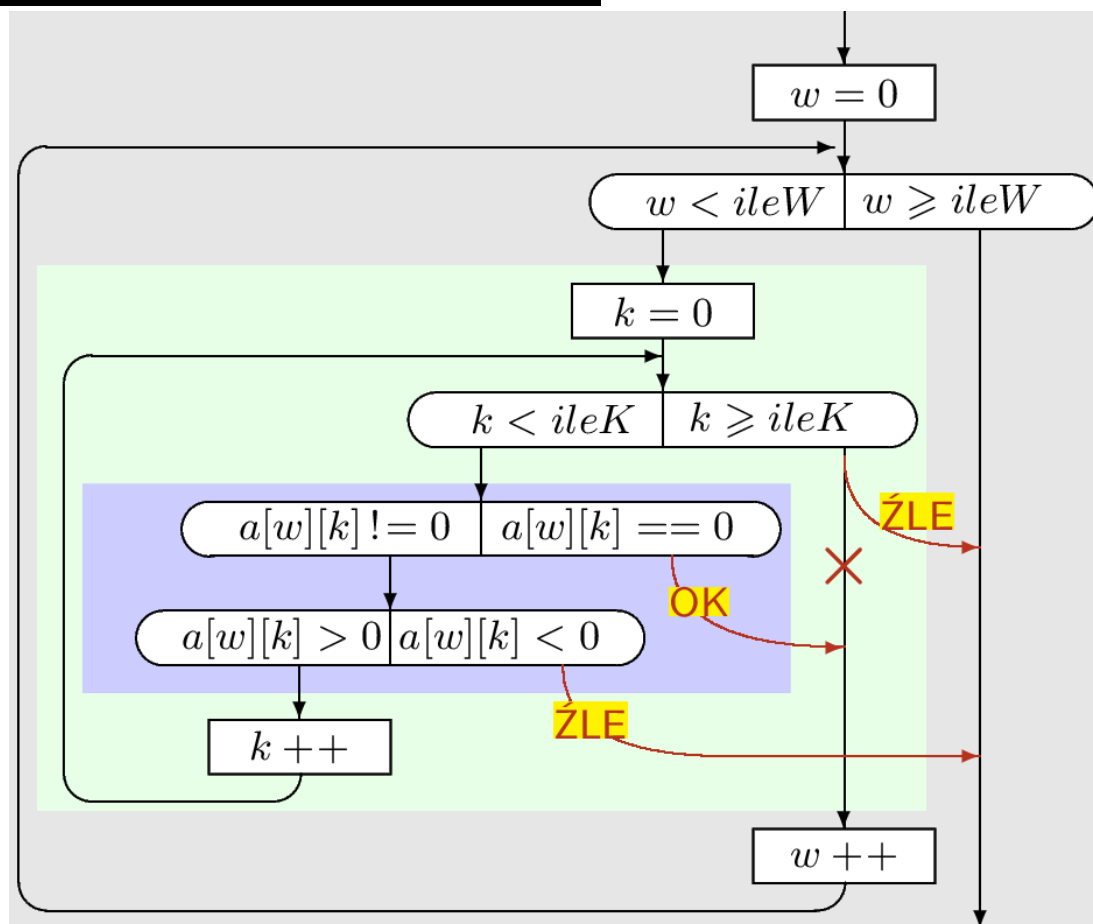
Skomplikowane schematy są trudne do analizy.

Modularyzacja z wyjątkami



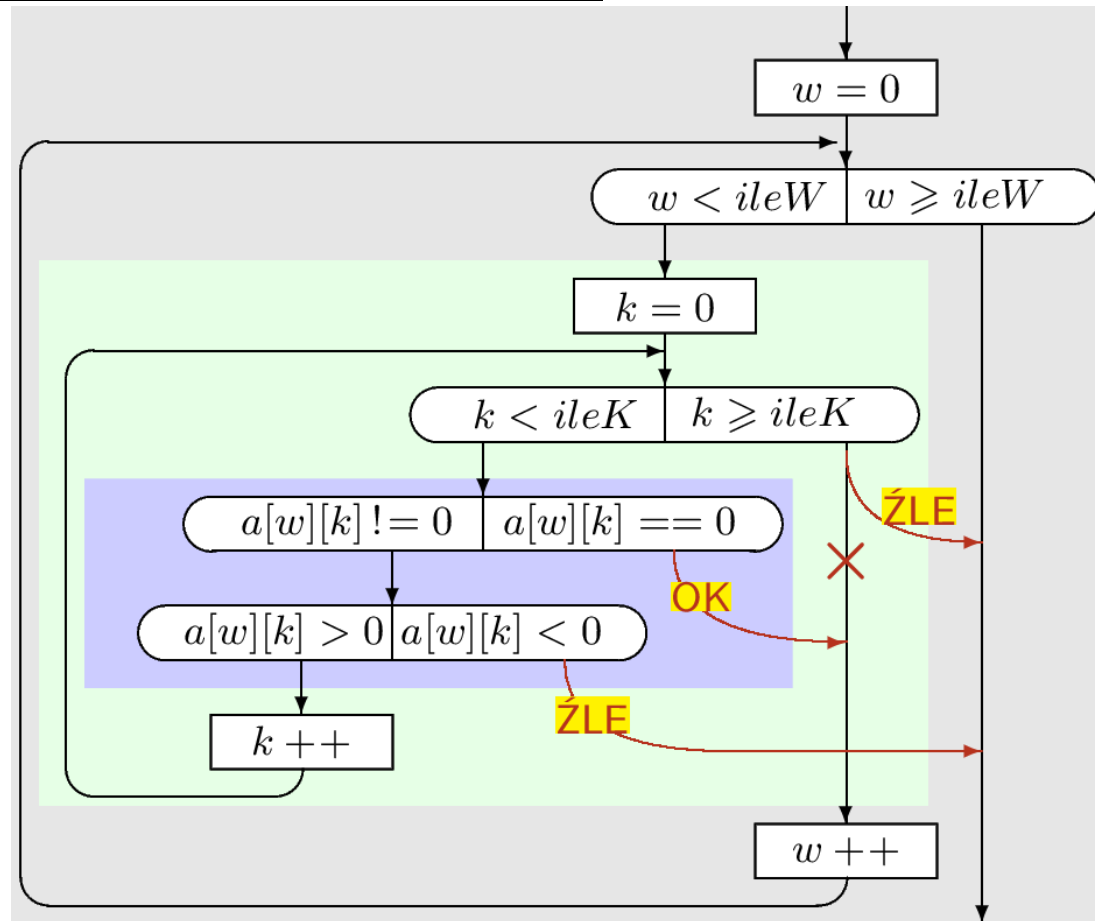
Skomplikowane schematy są trudne do analizy. Napisane wg nich programy są słabo czytelne.

Modularyzacja z wyjątkami



Skomplikowane schematy są trudne do analizy. Napisane wg nich programy są słabo czytelne. To zwiększa niebezpieczeństwo błędów.

Modularyzacja z wyjątkami



Skomplikowane schematy są trudne do analizy. Napisane wg nich programy są słabo czytelne. To zwiększa niebezpieczeństwo błędów.

Jak podzielić program z wyjątkami na względnie niezależne **niewielkie** kawałki, poprawnie współpracujące?

Modularyzacja z wyjątkami

Osobna metoda obsługująca pojedynczy wiersz

Modularyzacja z wyjątkami

Osobna metoda obsługująca pojedynczy wiersz:

```
public static void analizaWiersza(double[][] tab, int w)
{
    try {
        for (int k=0; k<tab[w].length; k++) {
            if (tab[w][k] < 0) throw new Ujemna(w,k);
            if (tab[w][k] == 0) throw new Zero();
        }
        throw new BezZera(w);
    }
    catch (Zero ex) {}
}
```

Modularyzacja z wyjątkami

Osobna metoda obsługująca pojedynczy wiersz:

```
public static void analizaWiersza(double[][] tab, int w)
{
    try {
        for (int k=0; k<tab[w].length; k++) {
            if (tab[w][k] < 0) throw new Ujemna(w,k);
            if (tab[w][k] == 0) throw new Zero();
        }
        throw new BezZera(w);
    }
    catch (Zero ex) {}
}
```

Kto przechwyci wyjątki generowane przez tą metodę?

Modularyzacja z wyjątkami

Osobna metoda obsługująca pojedynczy wiersz:

```
public static void analizaWiersza(double[][] tab, int w)
    throws Ujemna, BezZera {
    try {
        for (int k=0; k<tab[w].length; k++) {
            if (tab[w][k] < 0) throw new Ujemna(w,k);
            if (tab[w][k] == 0) throw new Zero();
        }
        throw new BezZera(w);
    }
    catch (Zero ex) {}
}
```



Kto przechwyci wyjątki generowane przez tą metodę?

Jeśli nikt, to trzeba ostrzec program, w którym metoda będzie wywoływana.

Modularyzacja z wyjątkami

Użycie tej metody w programie głównym

Modularyzacja z wyjątkami

Użycie tej metody w programie głównym:

```
try {  
    for (int w=0; w<ileW; w++) analizaWiersza(tab, w);  
}  
catch (Ujemna ex) {  
    System.out.println("\n UJEMNA ...");  
    System.exit(0);  
}  
catch (BezZera ex) {  
    System.out.println("\n WIERSZ BEZ ZER: ...");  
    System.exit(0);  
}  
System.out.println("\n OK\n");
```

Modularyzacja z wyjątkami

Użycie tej metody w programie głównym:

```
try {  
    for (int w=0; w<ileW; w++) analizaWiersza(tab, w);  
}  
catch (Ujemna ex) {  
    System.out.println("\n UJEMNA ...");  
    System.exit(0);  
}  
catch (BezZera ex) {  
    System.out.println("\n WIERSZ BEZ ZER: ...");  
    System.exit(0);  
}  
System.out.println("\n OK\n");
```



metoda
rzucająca
wyjątki

Klasy wyjątków — tych od błędów i innych

Klasy wyjątków — tych od błędów i innych

Wyjątki *checked* wymagają

- albo przechwycenia `catch (...) { ... }`
- albo deklaracji `throws ...`.

W braku jednego i drugiego program nie zostanie skompilowany

Klasy wyjątków — tych od błędów i innych

Wyjątki *checked* wymagają

- albo przechwycenia `catch (...) { ... }`
- albo deklaracji `throws ...`.

W braku jednego i drugiego program nie zostanie skompilowany — błąd:

```
unreported exception ...;  
must be caught or declared to be thrown
```

Klasy wyjątków — tych od błędów i innych

Wyjątki *checked* wymagają

- albo przechwycenia `catch (...) { ... }`
- albo deklaracji `throws ...`.

W braku jednego i drugiego program nie zostanie skompilowany — błąd:

```
unreported exception ...;  
must be caught or declared to be thrown
```

Wyjątki *unchecked* nie wymagają

- ani przechwycenia `catch (...) { ... }`
- ani deklaracji `throws ...`.

ale obie te rzeczy **można** wpisać.

Klasy wyjątków — tych od błędów i innych

Wyjątki *checked* wymagają

- albo przechwycenia `catch (...) { ... }`
- albo deklaracji `throws ...`.

W braku jednego i drugiego program nie zostanie skompilowany — błąd:

```
unreported exception ...;  
must be caught or declared to be thrown
```

Wyjątki *unchecked* nie wymagają

- ani przechwycenia `catch (...) { ... }`
- ani deklaracji `throws ...`.

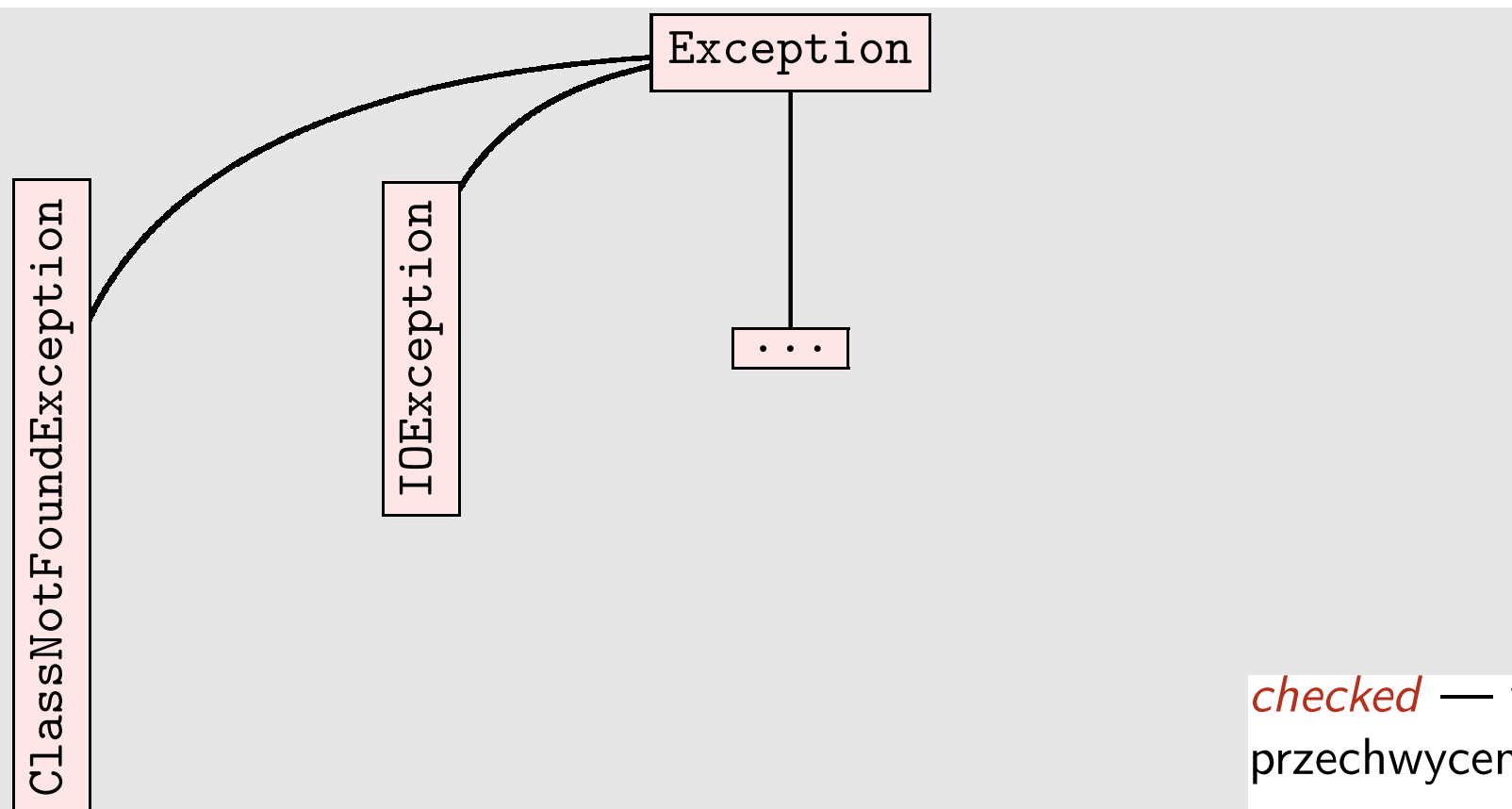
ale obie te rzeczy **można** wpisać. W braku jednego i drugiego program zostanie skompilowany, a tylko w przypadku zajścia sytuacji wyjątkowej jego wykonywanie zostanie przerwane.

Klasy wyjątków — tych od błędów i innych

Exception

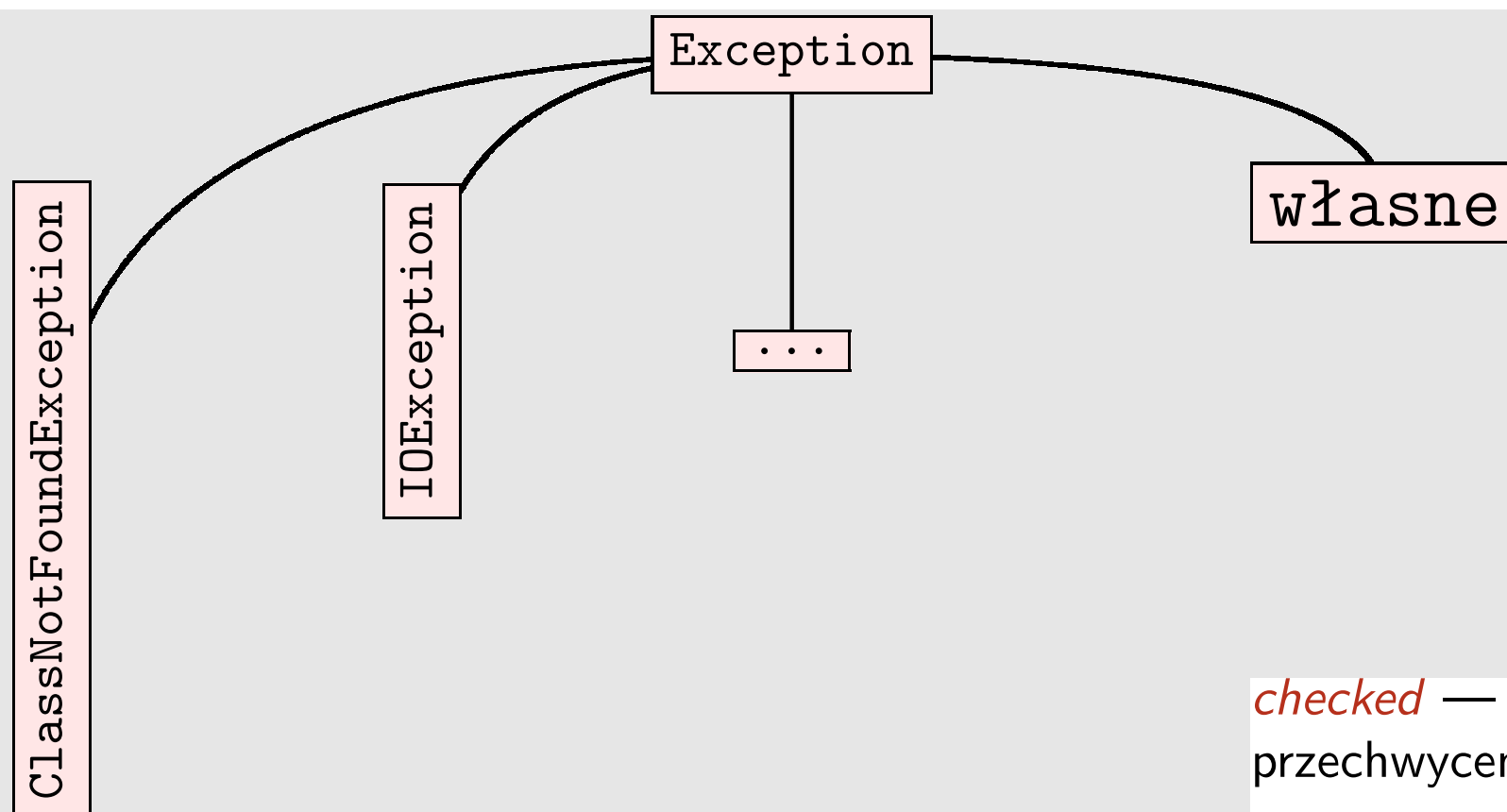
checked — wymagające
przechwycenia lub ostrzeżenia

Klasy wyjątków — tych od błędów i innych



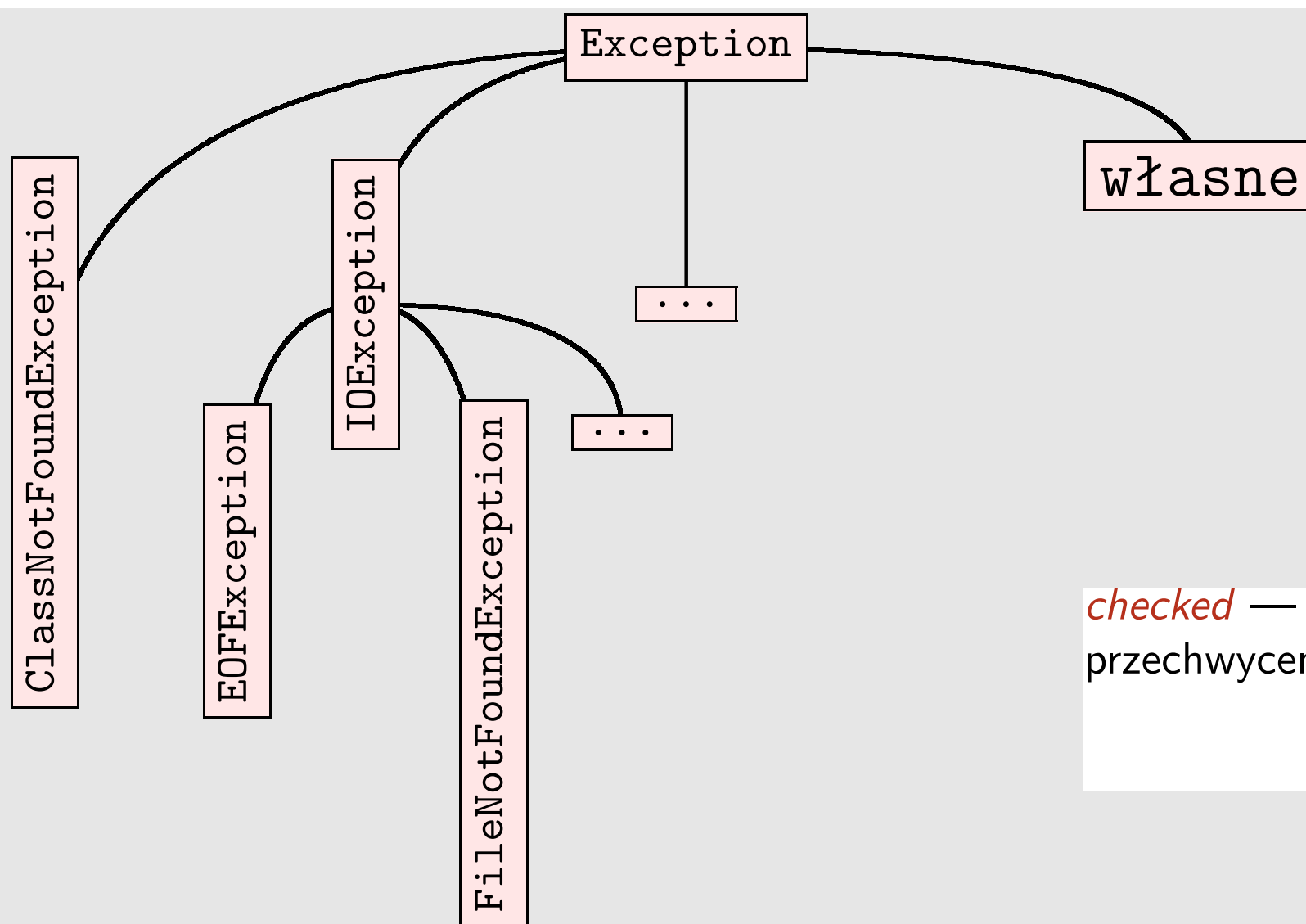
checked — wymagające
przechwycenia lub ostrzeżenia

Klasy wyjątków — tych od błędów i innych



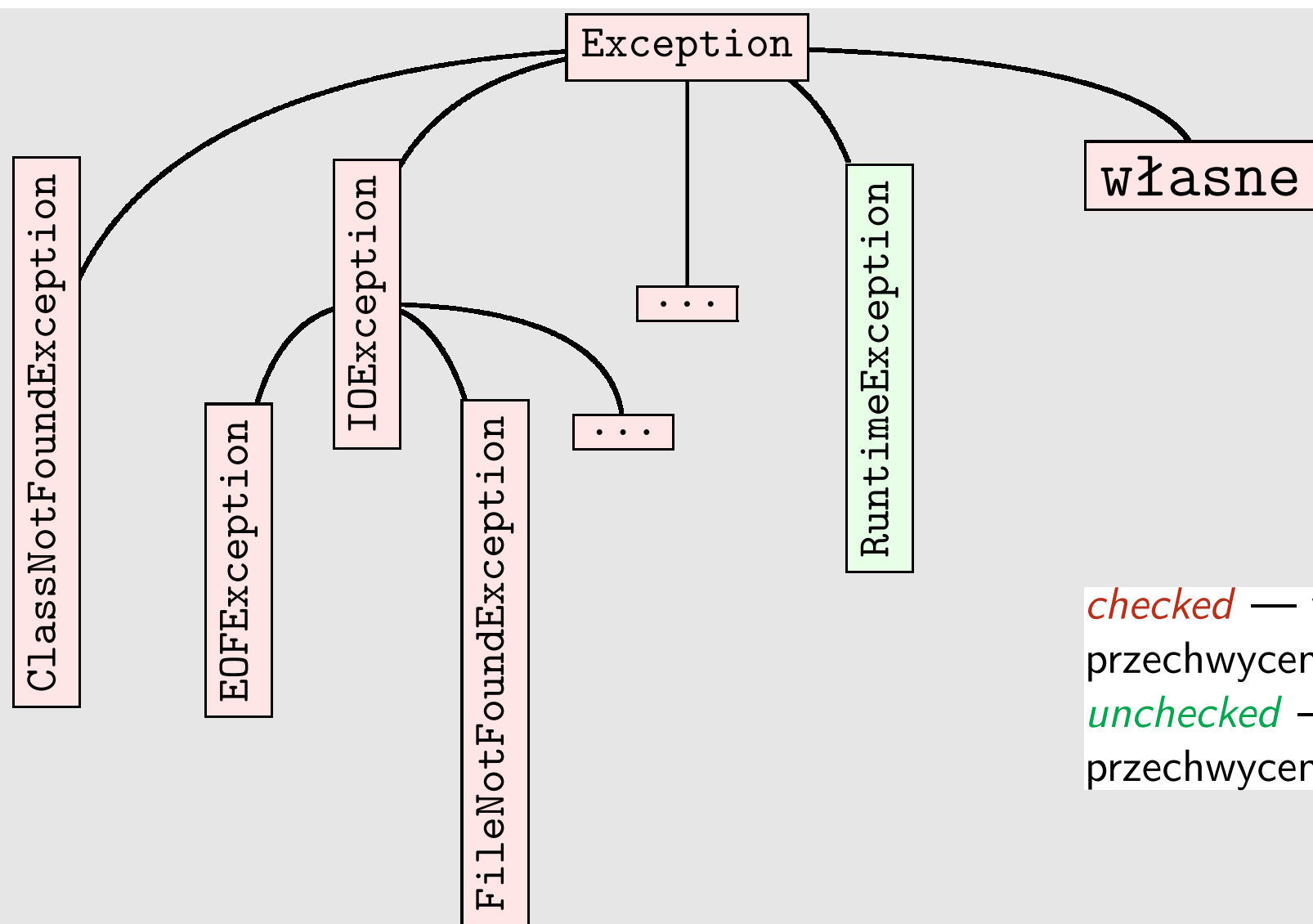
checked — wymagające przechwycenia lub ostrzeżenia

Klasy wyjątków — tych od błędów i innych



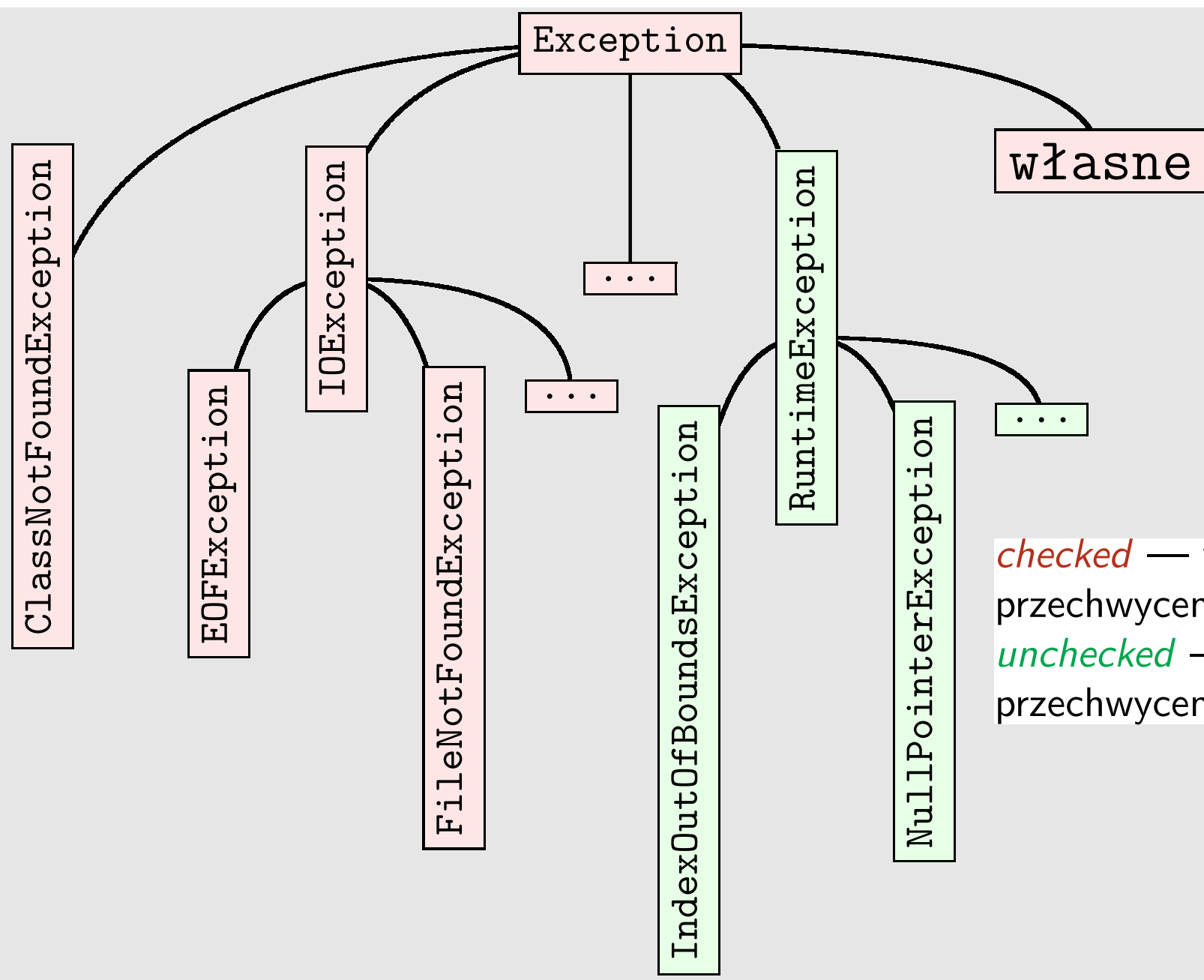
checked — wymagające
przechwycenia lub ostrzeżenia

Klasy wyjątków — tych od błędów i innych



checked — wymagające
przechwycenia lub ostrzeżenia
unchecked — niewymagające
przechwycenia ani ostrzeżenia

Klasy wyjątków — tych od błędów i innych



Wyjątki

Używać czy nie używać i jak używać wyjątków?
(dobre rady dziadka)



Wyjątki

Używać czy nie używać i jak używać wyjątków?
(dobre rady dziadka)

```
if (0<=ind && ind<ls.size())  
    ls.remove(ind);
```



Wyjątki

Używać czy nie używać i jak używać wyjątków?
(dobre rady dziadka)

```
if (0<=ind && ind<ls.size())  
    ls.remove(ind);
```

czy

```
try { ls.remove(ind); }  
catch (IndexOutOfBoundsException ex) {}
```



Wyjątki

Używać czy nie używać i jak używać wyjątków?
(dobre rady dziadka)



```
if (0<=ind && ind<ls.size())  
    ls.remove(ind);
```

czy

```
try { ls.remove(ind); }  
catch (IndexOutOfBoundsException ex) {}
```

wielokrotnie
szybsze

Wyjątki

Używać czy nie używać i jak używać wyjątków?
(dobre rady dziadka)



```
if (0<=ind && ind<ls.size())  
    ls.remove(ind);
```

czy

```
try { ls.remove(ind); }  
catch (IndexOutOfBoundsException ex) {}
```

wielokrotnie
szybsze

NIE UŻYWAĆ

Wyjątki

Używać czy nie używać i jak używać wyjątków?
(dobre rady dziadka)

```
for (i=0; i<100; i++) {  
    try { komenda A } catch(WyjatekA ex) {...stop...}  
    try { komenda B } catch(WyjatekB ex) {...stop...}  
}
```



Wyjątki

Używać czy nie używać i jak używać wyjątków?
(dobre rady dziadka)

```
for (i=0; i<100; i++) {  
    try { komenda A } catch(WyjątekA ex) {...stop...}  
    try { komenda B } catch(WyjątekB ex) {...stop...}  
}
```

czy

```
try {  
    for (i=0; i<100; i++) {  
        komenda A; komenda B;  
    }  
}  
catch(WyjątekA ex) {...stop...}  
catch(WyjątekB ex) {...stop...}
```



Wyjątki

Używać czy nie używać i jak używać wyjątków?
(dobre rady dziadka)



```
for (i=0; i<100; i++) {  
    try { komenda A } catch(WyjątekA ex) {...stop...}  
    try { komenda B } catch(WyjątekB ex) {...stop...}  
}
```

czy

```
try {  
    for (i=0; i<100; i++) {  
        komenda A; komenda B;  
    }  
}  
catch(WyjątekA ex) {...stop...}  
catch(WyjątekB ex) {...stop...}
```

prostsze
lepiej oddziela normalne
od wyjątkowego

Wyjątki

Używać czy nie używać i jak używać wyjątków?
(dobre rady dziadka)



```
for (i=0; i<100; i++) {  
    try { komenda A } catch(WyjątekA ex) {...stop...}  
    try { komenda B } catch(WyjątekB ex) {...stop...}  
}
```

czy

```
try {  
    for (i=0; i<100; i++) {  
        komenda A; komenda B;  
    }  
}  
catch(WyjątekA ex) {...stop...}  
catch(WyjątekB ex) {...stop...}
```

prostsze
lepiej oddziela normalne
od wyjątkowego

UŻYWAĆ DRUGIEGO