Politechnika Wrocławska

Platformy Programistyczne Wykład z Javy dla zaawansowanych

Agata Migalska

20 maja 2014

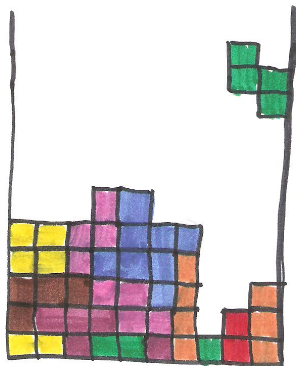
Plan wykładu

1. Polimorfizm i dziedziczenie
2. Życie i śmierć obiektu
3. Poziomy oraz modyfikatory dostępu
4. Obsługa wyjątków

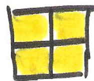
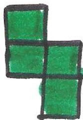
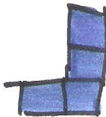
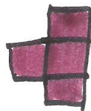
Polimorfizm i dziedziczenie

1. Polimorfizm i dziedziczenie
   - Tetris w klasycznej odsłonie
   - Tetris - nowe wymagania
2. Życie i śmierć obiektu
3. Poziomy oraz modyfikatory dostępu
4. Obsługa wyjątków

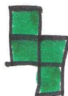
Tetris

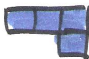


Klocki potrafią się narysować

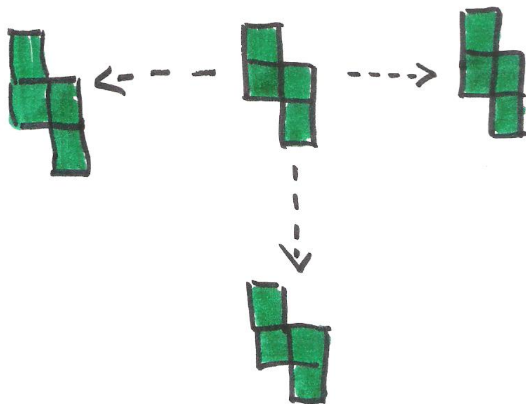





Klocki obracają się

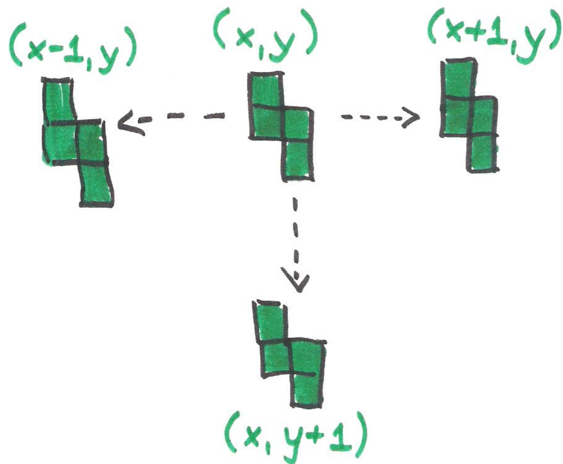




Klocki przesuwają się w dół i na boki



Klocki znają swoją pozycję



Klocki - ich zmienne i metody

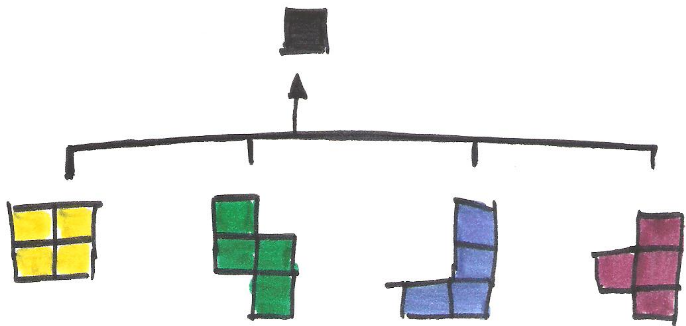
```
Kwadrat
-  pozycja
+  narysuj() :void
+  przesuńWBok() :void
+  przesuńWDół() :void
+  obróć() :void
```

```
Krzesło
-  pozycja
+  narysuj() :void
+  przesuńWBok() :void
+  przesuńWDół() :void
+  obróć() :void
```

```
But
-  pozycja
+  narysuj() :void
+  przesuńWBok() :void
+  przesuńWDół() :void
+  obróć() :void
```

```
Trójkąt
-  pozycja
+  narysuj() :void
+  przesuńWDół() :void
+  przesuńWBok() :void
+  obróć() :void
```

Wspólna abstrakcja



Wspólna abstrakcja

«abstract»
Klocek
- pozycja
+ przesuńWBok() :void
+ przesuńWDół() :void
+ obróć() :void

Kwadrat
+ narysuj() :void

Krzesło
+ narysuj() :void

But
+ narysuj() :void

Trójkąt
+ narysuj() :void

Relacje JEST i MA

Gdy jedna klasa dziedziczy po drugiej...

mówimy, że klasa pochodna *rozszerza* klasę bazową.

Test relacji JEST

Test sprawdzający czy jedna rzecz powinna rozszerzać inną.

- Kwadratowy klocek JEST klockiem ✔
- Człowiek JEST ssakiem ✔
- Pozycja klocka JEST klockiem ✘
- Wanna JEST łazienką ✘

Relacje JEST i MA

Relacja MA

Klocek MA swoją pozycję ✔
Łazienka MA wannę ✔

Dziedziczenie a kompozycja

Jeżeli **A JEST B**, korzystamy z dziedziczenia.
Jeżeli **A MA B**, korzystamy z kompozycji.

Jak to zaimplementować?

```
public abstract class Klocek {

  private Wspolrzedne pozycja;

  public abstract void narysuj();

  public void obroc(){  ... }
  public void przesunWBok(){ ... }
  public void przesunWDol(){ ... }
}
```

Jak to zaimplementować?

```
public class Kwadrat extends Klocek {

  public void narysuj() {
      ...
  }

}
```

Klasa konkretna

Konkretna klasa pochodna musi zaimplementować wszystkie metody abstrakcyjne, które nie zostały zaimplementowane w jej abstrakcyjnej klasie bazowej.

Drzewo dziedziczenia

```
public abstract class Foo {
  public abstract String moo();
  public abstract void boo();
}

public abstract class FooMoo extends Foo {
  public String moo() { return "moo"; }
}

public class FooMooBoo extends FooMoo {
  public void boo(){ ... }
}
```

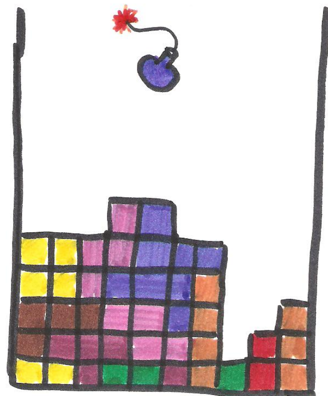
Jeszcze o dziedziczeniu

- Jeśli zadeklarujesz metodę jako abstrakcyjną, w ten sam sposób MUSISZ zadeklarować całą klasę. Abstrakcyjne metody nie mogą istnieć w "nieabstrakcyjnych" klasach.
- Klasa zadeklarowana jako abstrakcyjna NIE MUSI mieć ani jednej metody abstrakcyjnej.

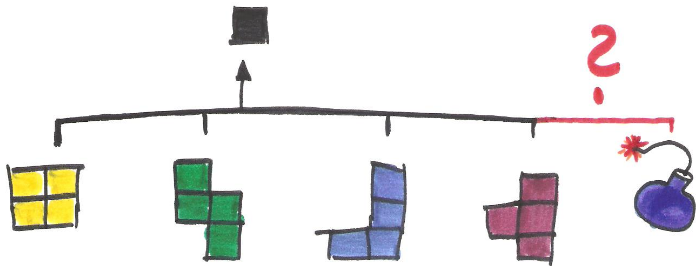
Dygresja na temat nazewnictwa

- Nazwy klas rozpoczynamy z wielkiej litery
- Nazwa pliku *.java jest taka sama jak nazwa klasy w tym pliku
- Nazwy zmiennych i metod rozpoczynamy z małej litery
- W nazwach wykorzystujemy tzw. camel case, czyli kolejneWyrazyRozpoczamyZWielkiejLitery

Tetris - nowe wymagania



Czy bomba jest klockiem?



O czym bomba wie i co potrafi

- ✔ zna swoją pozycję
- ✔ potrafi się narysować
- ✔ przesuwa się w dół i na boki

- ✔ cecha bomby występująca u klocka
- ✘ cecha bomby nie występująca u klocka

O czym bomba wie i co potrafi

- ✔ zna swoją pozycję
- ✔ potrafi się narysować
- ✔ przesuwa się w dół i na boki
- ✘ nie obraca się
- ✘ wybucha

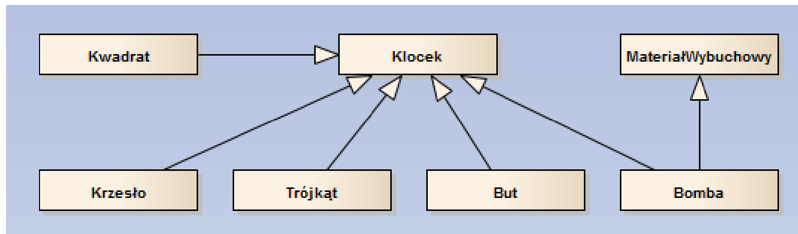
- ✔ cecha bomby występująca u klocka
- ✘ cecha bomby nie występująca u klocka

O czym bomba wie i co potrafi

- ✔ zna swoją pozycję
- ✔ potrafi się narysować
- ✔ przesuwa się w dół i na boki
- ✘ nie obraca się brak obrotu to specyficzny sposób obrotu...
- ✘ wybucha klocki nie wybuchają...

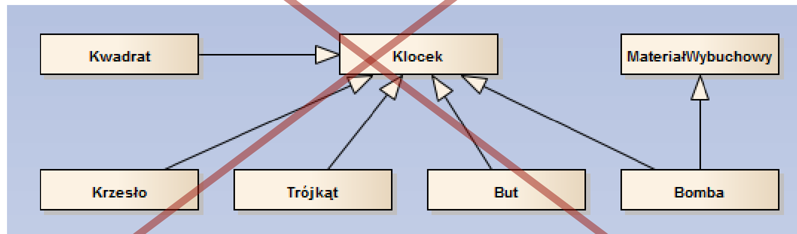
- ✔ cecha bomby występująca u klocka
- ✘ cecha bomby nie występująca u klocka

Gdyby w Javie istniało wielodziedziczenie

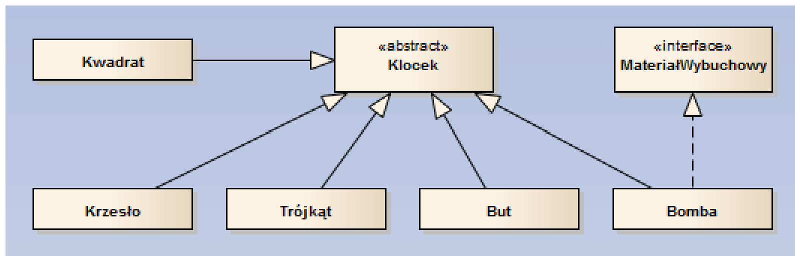


Politechnika Wrocławska

Gdyby w Javie istniało wielodziedziczenie



Interfejs



Implementacja Bomby

```
public interface MaterialWybuchowy {
  public abstract void zdetonujSie();
}
```

```
public class Bomba extends Klocek
   implements MaterialWybuchowy {

  public void zdetonuj() {
    // efektowny wybuch
  }
  public void obroc() {
    // nie rob absolutnie nic
  }
}
```

Gdy jeden interfejs to za mało

```
public class Bomba extends Klocek implements
  MaterialWybuchowy, Atomowa, Samonaprowadzalna {
     ...
}
```

Dylemat: co wybrać?

- Utwórz klasę, która nie rozszerza żadnej innej, jeżeli nie przechodzi ona testu relacji JEST z żadną inną klasą.
- Utwórz klasę pochodną jedynie wtedy, gdy potrzebujesz **bardziej wyspecjalizowanej** wersji klasy i musisz przesłonić jakieś zachowania lub dodać nowe.
- Użyj klasy abstrakcyjnej, jeśli chcesz zdefiniować pewien **wzorzec** dla klas pochodnych i jeśli chcesz mieć pewność, że nikt nie będzie mógł tworzyć obiektów tego typu.
- Utwórz interfejs, jeśli chcesz zdefiniować *funkcję*, jaką mają pełnić inne klasy i to niezależnie od ich położenia w drzewie dziedziczenia.

Operator instanceof

Operatof instanceof

- Przeprowadza test relacji JEST.
- Sprawdza czy dany obiekt jest obiektem danej klasy lub klasy pochodnej.
- Sprawdza czy dany obiekt implementuje dany interfejs.

```
public String zrobCos(Klocek klocek) {

  if (klocek instanceof Bomba) {
    zdetonuj();
  }
}
```

Operator instanceof

```java
public class Bomba implements Wybuchowy { }
public class BombaAtomowa extends Bomba { }
```

```java
public String wybuchnij(Wybuchowy wybuchowy) {

  if (wybuchowy instanceof BombaAtomowa) {
    zniszczWszystko();
  }
}
```

Życie i śmierć obiektu

1. Polimorfizm i dziedziczenie
2. Życie i śmierć obiektu
3. Poziomy oraz modyfikatory dostępu
4. Obsługa wyjątków

Obszary pamięci - Stos i Sterta

Stos

Tutaj żyją wywołania metod i zmienne lokalne.

Sterta

Tutaj żyją wszystkie obiekty.

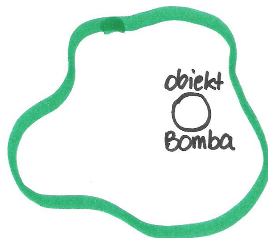
Bombowy kod 1

```
public class Bomba{
  public void zdetonuj(){
    double liczba = Math.random();
    eksploduj( liczba < 0.5 );
  }
  public void eksploduj( boolean napewno ){
    zniszczPlansze( 4, 5 );
    System.out.println("Zniszczono.");
  }
  public void zniszczPlansze( int x, int y ) {
    // niszczy plansze
  }
}
```

Bombowy kod na Stosie

STOS

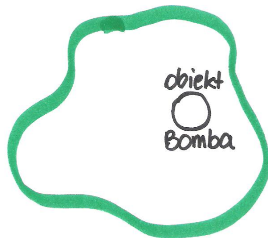
STERTA



Bombowy kod na Stosie

STOS

STERTA

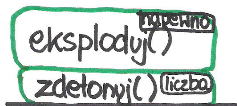


zdetonuj() liczba

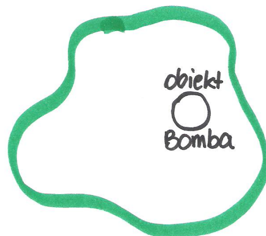
```
public void zdetonuj(){
  double liczba = Math.random();
  eksploduj( liczba < 0.5 );
}
```

Bombowy kod na Stosie

STOS

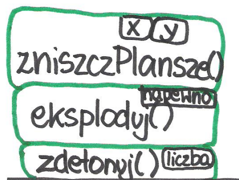


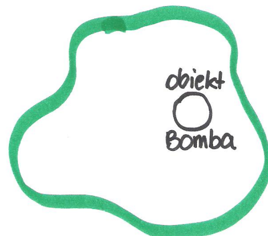
STERTA




```
public void eksploduj( boolean napewno ){
  zniszczPlansze( 4, 5 );
  System.out.println("Zniszczono.");
}
```

Bombowy kod na Stosie

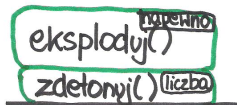




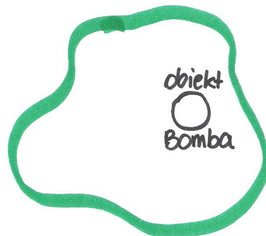

```
public void zniszczPlansze( int x, int y ) {
  // niszczy plansze

}
```

Bombowy kod na Stosie

STOS



STERTA

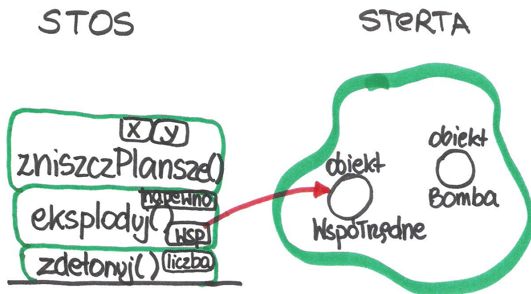



```java
public void eksploduj( boolean napewno ){
  zniszczPlansze( 4, 5 );
  System.out.println("Zniszczono.");
}
```

Bombowy kod 2

```java
public class Bomba{
  public void zdetonuj(){
    double liczba = Math.random();
    eksploduj( liczba < 0.5 );
  }
  public void eksploduj( boolean napewno ){
    Wspolrzedne wsp= new Wspolrzedne();
    zniszczPlansze( wsp.getX(), wsp.getY() );
    System.out.println("Zniszczono.");
  }
  public void zniszczPlansze( int x, int y ) {
    // niszczy plansze
  }
}
```

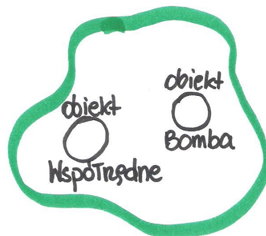
Bombowy kod na Stosie i Stercie




```
public void eksploduj( boolean napewno ){
  Wspolrzedne wsp= new Wspolrzedne();
  zniszczPlansze( wsp.getX(), wsp.getY() );
}
```

Bombowy kod na Stosie i Stercie

STOS




```
// Metoda zdetonuj() wykonala sie i zostala zdjeta
// ze stosu. Zadna referencja nie wskazuje na
// obiekt Wspolrzedne
```

Stos i Sterta - Podsumowanie

1. W Javie są dwa obszary pamięci - stos i sterta.
2. Wszystkie zmienne lokalne istnieją na stosie, w ramce odpowiadającej metodzie, w której zostały zadeklarowane.
3. Zmienne referencyjne zachowują się tak samo jak zmienne typów prostych - jeżeli odwołanie zostało zdeklarowane jako zmienna lokalna, to będzie przechowywane na stosie.
4. Wszystkie obiekty przechowywane są na stercie, niezależnie od tego, czy odwołania wskazujące na nie są zmiennymi egzemplarza czy też zmiennymi lokalnymi.

Kiedy umierają obiekty na Stercie?

Życie obiektu na Stercie

Obiekt istnieje, dopóki istnieją aktywne odwołania (referencje) do niego.

Gotowość do ośmiecenia

Obiekt zostaje uznany za nadający się do odśmiecenia, kiedy przestanie istnieć ostatnie aktywne odwołanie wskazujące na niego.

Garbage Collector

Kiedy programowi zacznie brakować pamięci, Garbage Collector (Odśmiecacz) usunie część lub wszystkie obiekty przeznaczone do odśmiecenia.

Poziomy oraz modyfikatory dostępu

1. Polimorfizm i dziedziczenie
2. Życie i śmierć obiektu
3. Poziomy oraz modyfikatory dostępu
4. Obsługa wyjątków

Poziomy dostępu

Dostęp do elementu w zależności od poziomu

- **prywatny** (*private*) - wyłącznie kod należący do tej samej klasy. Element jest prywatny w ramach klasy, a nie obiektu. Jeden Pies ma dostęp do prywatnych elementów drugiego Psa, jednak Kot nie będzie mógł z nich korzystać.

Poziomy dostępu

Dostęp do elementu w zależności od poziomu

- **prywatny** (*private*) - wyłącznie kod należący do tej samej klasy. Element jest prywatny w ramach klasy, a nie obiektu. Jeden Pies ma dostęp do prywatnych elementów drugiego Psa, jednak Kot nie będzie mógł z nich korzystać.
- **pakietowy / domyślny** (*default*) - wyłącznie kod należący do tego samego pakietu

Poziomy dostępu

Dostęp do elementu w zależności od poziomu

- **prywatny** (*private*) - wyłącznie kod należący do tej samej klasy. Element jest prywatny w ramach klasy, a nie obiektu. Jeden Pies ma dostęp do prywatnych elementów drugiego Psa, jednak Kot nie będzie mógł z nich korzystać.
- **pakietowy / domyślny** (*default*) - wyłącznie kod należący do tego samego pakietu
- **chroniony** (*protected*) - kod należący do tego samego pakietu oraz wszystkie klasy pochodne (niezależnie od pakietu)

Poziomy dostępu

Dostęp do elementu w zależności od poziomu

- **prywatny** (*private*) - wyłącznie kod należący do tej samej klasy. Element jest prywatny w ramach klasy, a nie obiektu. Jeden Pies ma dostęp do prywatnych elementów drugiego Psa, jednak Kot nie będzie mógł z nich korzystać.
- **pakietowy / domyślny** (*default*) - wyłącznie kod należący do tego samego pakietu
- **chroniony** (*protected*) - kod należący do tego samego pakietu oraz wszystkie klasy pochodne (niezależnie od pakietu)
- **publiczny** (*public*) - dowolny kod umieszczony w dowolnym miejscu programu

Modyfikatory dostępu

Modyfikatory dostępu

- private
- protected
- public

Modyfikatory dostępu

Modyfikatory dostępu

- private
- protected
- public

A gdzie czwarty modyfikator?

Pakietowy poziom dostępu jest stosowany domyślnie i nie ma swojego modyfikatora.

Poziomy i modyfikatory dostępu

public

Tego modyfikatora możesz używać do określania poziomu dostępu do klas, stałych, konstruktorów oraz metod, które chcesz udostępnić całemu światu.

Poziomy i modyfikatory dostępu

public

Tego modyfikatora możesz używać do określania poziomu dostępu do klas, stałych, konstruktorów oraz metod, które chcesz udostępnić całemu światu.

private

Powinien być stosowany do zmiennych klasy oraz metod, które nie powinny być wywoływane przez kod spoza klasy.

Poziomy i modyfikatory dostępu

default

Pakiety są zazwyczaj projektowane jako grupy współpracujących, powiązanych ze sobą klas. Często stosowany, żeby umożliwić przetestowanie metody.

Poziomy i modyfikatory dostępu

default

Pakiety są zazwyczaj projektowane jako grupy współpracujących, powiązanych ze sobą klas. Często stosowany, żeby umożliwić przetestowanie metody.

protected

Daje możliwość, żeby klasy pochodne, należące do innego pakietu niż klasa bazowa, miały dostęp (poprzez dziedziczenie) do obiektów chronionych.

Obsługa wyjątków

1. Polimorfizm i dziedziczenie
2. Życie i śmierć obiektu
3. Poziomy oraz modyfikatory dostępu
4. Obsługa wyjątków

Obsługa wyjątków

Zgłaszanie wyjątków

Metoda zgłasza wyjątek, aby przekazać informację, że stało się coś złego.

Metoda zgłaszająca wyjątek

FileInputStream

```
public FileInputStream(String name)
                throws FileNotFoundException
```

Throws:

`FileNotFoundException` - if the file does not exist, is a directory rather than a regular file, or for some other reason cannot be opened for reading.

`SecurityException` - if a security manager exists and its `checkRead` method denies read access to the file.

Wywołanie ryzykownej metody

```
try {
  file = new FileInputStream(fileName);
  x = (byte) file.read();
} catch (FileNotFoundException ex) {
  ex.printStackTrace();
}
```

Hierarchia wyjątków

«interface»
Throwable

\+ printStackTrace() :void
\+ getMessage() :String

Exception

RuntimeException

Wyjątki

Obsługa wyjątków przez kompilator

1. Jeśli zgłaszasz wyjątek, musisz w deklaracji metody umieścić słowo kluczowe `throws`.
2. Jeśli wywołujesz metodę zgłaszającą wyjątek, musisz potwierdzić, że zdajesz sobie sprawę z ryzyka.

Wyjątki

Obsługa wyjątków przez kompilator

1. Jeśli zgłaszasz wyjątek, musisz w deklaracji metody umieścić słowo kluczowe `throws`.
2. Jeśli wywołujesz metodę zgłaszającą wyjątek, musisz potwierdzić, że zdajesz sobie sprawę z ryzyka.

Wyjątki niesprawdzane (*ang. unchecked exceptions*)

Wyjątki typu RuntimeException NIE są sprawdzane przez kompilator. Można zgłaszać, przechwytywać i deklarować wyjątki tego typu, jednak nie trzeba.

RuntimeException

```
try {
  x = Integer.parseInt("2");
} catch (ParseException e) {
  x = 3;
}
```

RuntimeException czyli wyjątki czasu wykonania programu

Te wyjątki najczęściej wynikają z błędów w implementacji.

- NullPointerException
- IndexOutOfBoundsException

Przechwytywanie wyjątków

```
try {
  // jakis kod
} catch (Wyjatek e) {
  // zrob cos
}
```

Przechwytywanie wyjątków

```
try {
  // jakis kod
} catch (Wyjatek e) {
  // zrob cos
} finally {
  // wywolaj niezaleznie od wyniku
}
```

Czy wyjątek został przechwycony?

```
try {
  // jakis kod
} finally {
  // wykonaj niezaleznie od wyniku
}
```

Rezygnacja z obsługi wyjątku

```
public void mojaMetoda() throws Wyjatek {

  try {
    // jakis kod
  } finally {
    // wykonaj niezaleznie od wyniku
  }

}
```

Rezygnacja z obsługi wyjątku

```
public void mojaMetoda() throws Wyjatek {
 // blok try-catch-finally mozna pominac
}
```

Co zostanie wypisane na konsolę?

```
public void mojaMetoda() throws Wyjatek {
  try {
    System.out.print("a");
    throw new WyjatekA();
    System.out.print("b");
  } catch (WyjatekA ex) {
    System.out.print("c");
  } finally {
    System.out.print("d");
  }
  throw new Wyjatek();
  System.out.print("e");
}
```

Co zostanie wypisane na konsolę?

```
public void mojaMetoda() throws Wyjatek {
  try {
    System.out.print("a");
    throw new WyjatekA();
    System.out.print("b");
  } catch (WyjatekA ex) {
    System.out.print("c");
  } finally {
    System.out.print("d");
  }
  throw new Wyjatek();
  System.out.print("e");
}
```

Odp.: **acd**

Dziękuję za uwagę.