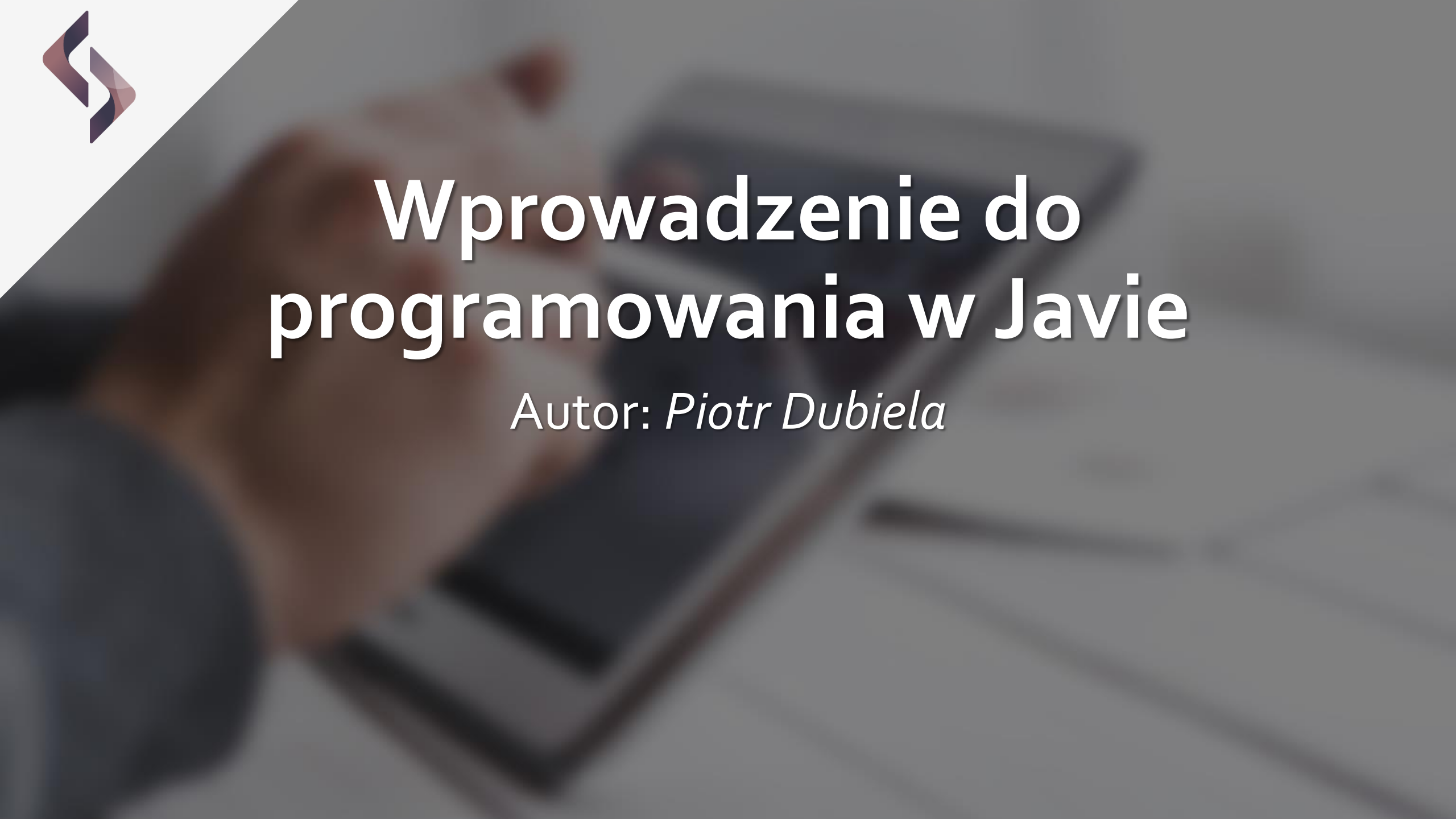
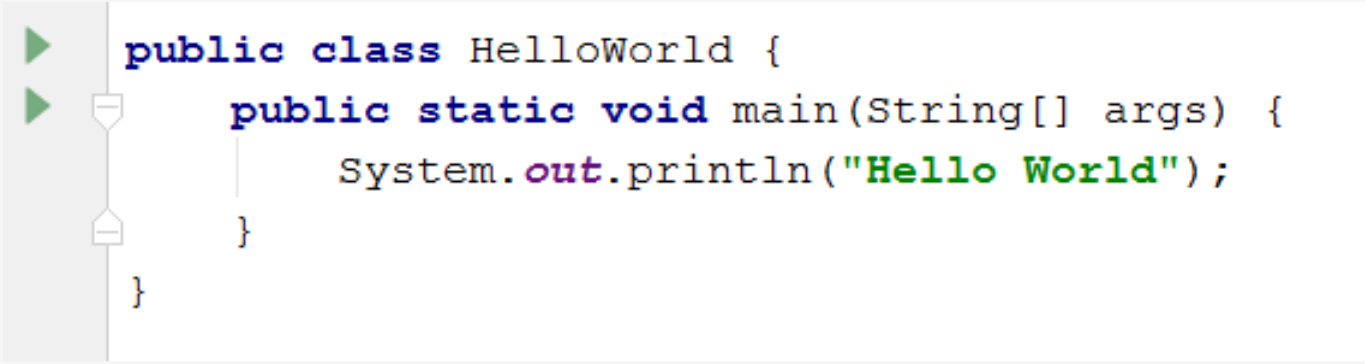


Wprowadzenie do programowania w Javie

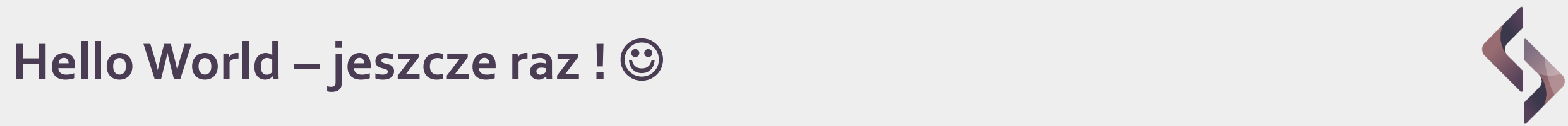
Autor: *Piotr Dubiela*

Hello World – jeszcze raz ! ☺



```java
public class HelloWorld {
    public static void main(String[] args) {
        System.out.println("Hello World");
    }
}
```

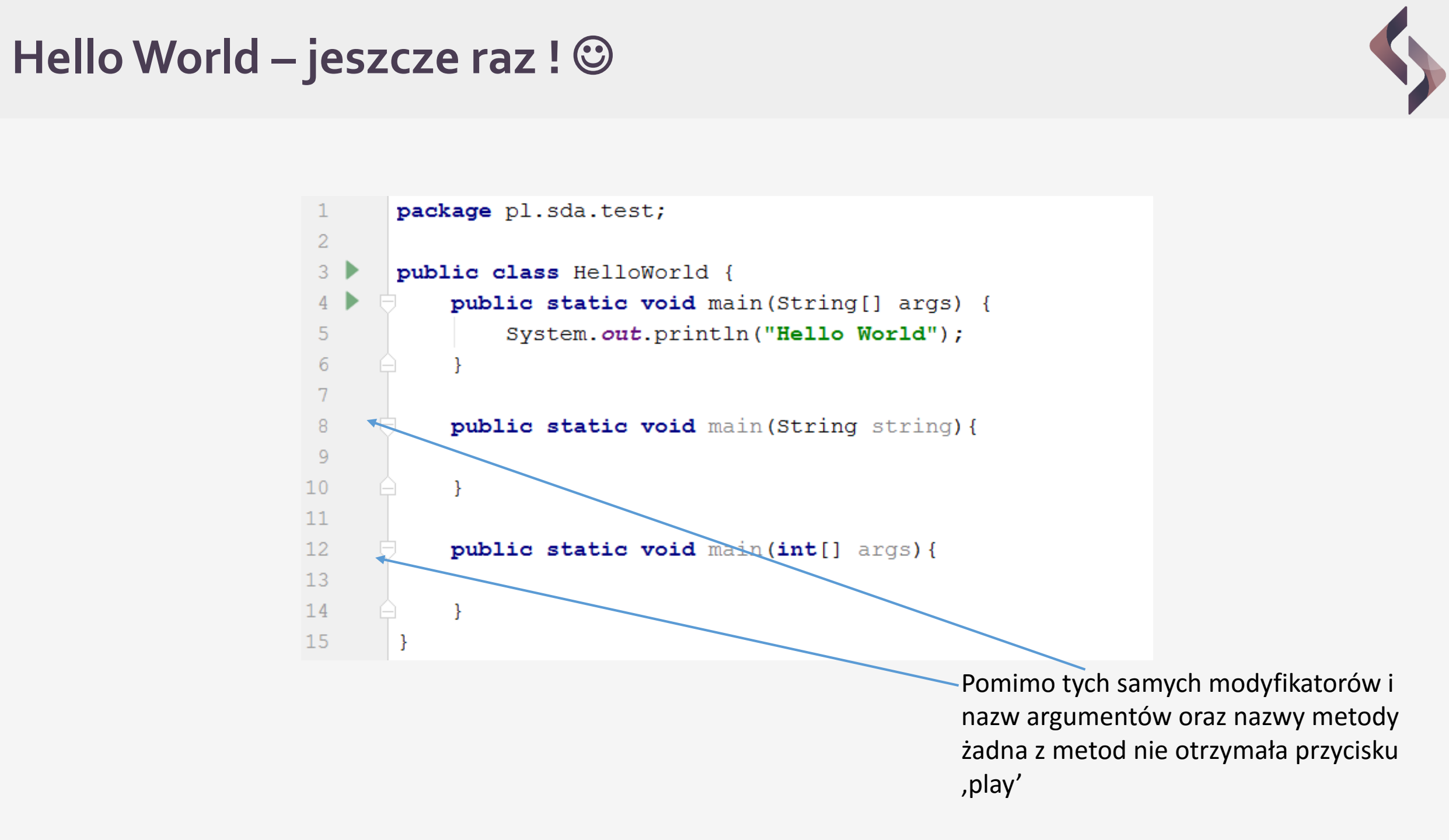
Hello World – jeszcze raz ! ☺



play oznacza, że możemy uruchomić program przez uruchomienie pojedynczej metody

```
public class HelloWorld {
    public static void main(String[] args) {
        System.out.println("Hello World");
    }
}
```

Hello World – jeszcze raz ! ☺



```
1   package pl.sda.test;
2
3   public class HelloWorld {
4       public static void main(String[] args) {
5           System.out.println("Hello World");
6       }
7
8       public static void main(String string){
9
10      }
11
12      public static void main(int[] args){
13
14      }
15  }
```

Pomimo tych samych modyfikatorów i nazw argumentów oraz nazwy metody żadna z metod nie otrzymała przycisku ‚play'

Hello World – jeszcze raz ! ☺

Metoda ‚main' - publiczna, statyczna, bez typu zwracanego, przyjmująca tablicę znaków stanowi punkt wejścia programu w Javie

```
1   package pl.sda.test;
2
3   public class HelloWorld {
4       public static void main(String[] args) {
5           System.out.println("Hello World");
6       }
7
8       public static void main(String string){
9
10      }
11
12      public static void main(int[] args){
13
14      }
15  }
```

Hello World – jeszcze raz ! ☺

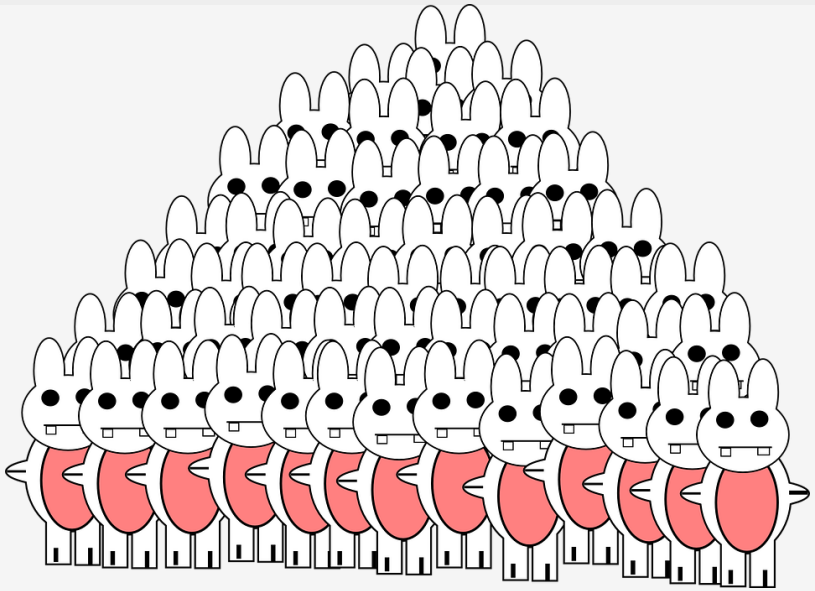
Czym jest zatem argument punktu wejścia ?

- Zbiór tzw. danych wejściowych dla programu
- Pozwala na automatyczne wykorzystanie programu np. program do rozsyłania powiadomień mailowych, uruchamiający się cyklicznie dla zadanych parametrów np. o 10 powiadomienia dla grupy A, o 11 dla grupy B itp.
- Maksymalnie jeden punkt wejścia w 1 klasie

Hello World – zadanie

1. *Utworz klasę DodawanieParametrami*
2. *Utwórz metodę psvm*
3. *Zsumuj otrzymane na wejściu liczby*
    1. *Utwórz metodę zamienNaLiczby(String [] znaki):int[]*
    2. *Skorzystaj z metody Integer.parseInt(String str);*
4. *Wyświetl wynik : Suma wprowadzonych liczb to {wynik}*
5. *Uruchom program z IntelliJ*
6. ** Dodaj program do artefaktów*
7. ** Zbuduj archiwum jar*
8. ** Przetestuj w konsoli*

Var Args



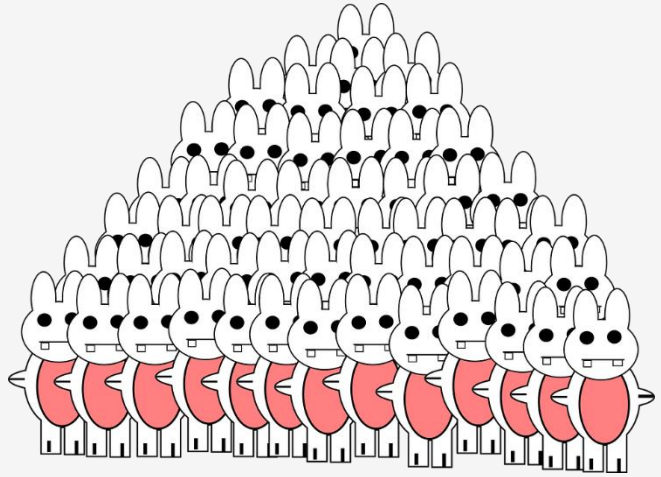
Czym jest varargs w Javie?

Varargs – pozwala na posiadanie metody mogącej przyjąć bliżej nieokreśloną liczbę argumentów (od 0 do n) tego samego typu.
Wprowadzone do Javy od JDK 1.5

Var Args

Przed wprowadzeniem varargs:

- Konieczność przeciążania metod dla każdego przypadku
- public void wydrukuj(){...}
- public void wydrukuj(**String text**){...}
- public void wydrukuj(**String text1, String text2**){...}

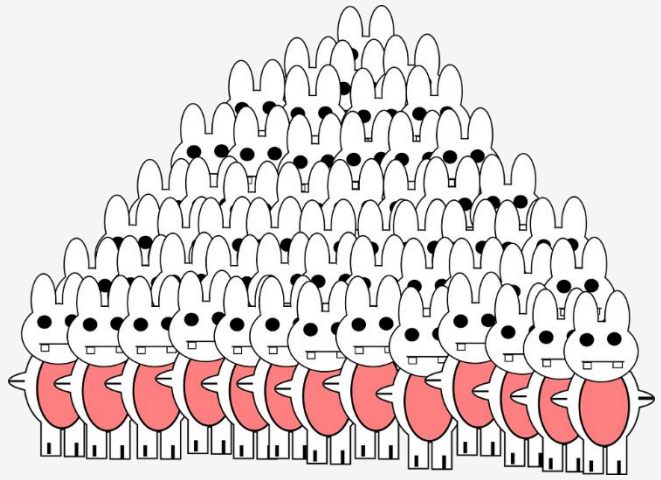


Var Args

Po dodaniu varargs:

- Pojedyncza metoda dla każdego przypadku:

```
public void wydrukuj(String... słowa){
//....
}
```



Var Args

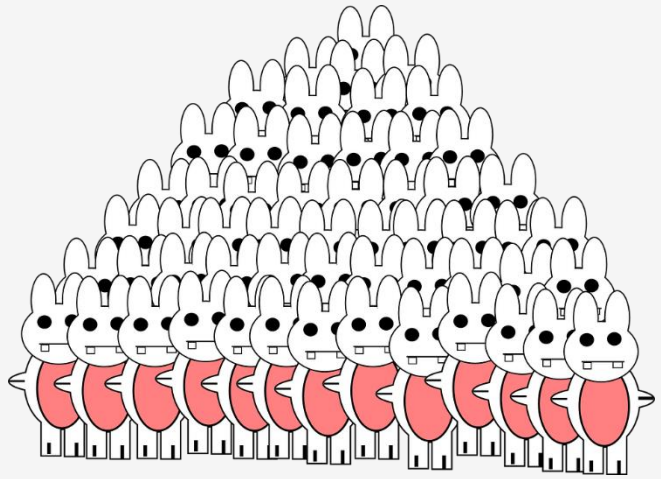
Po dodaniu varargs:

- Pojedyncza metoda dla każdego przypadku:

```
public void wydrukuj(String... słowa){
//....
}
```

- Użycie:

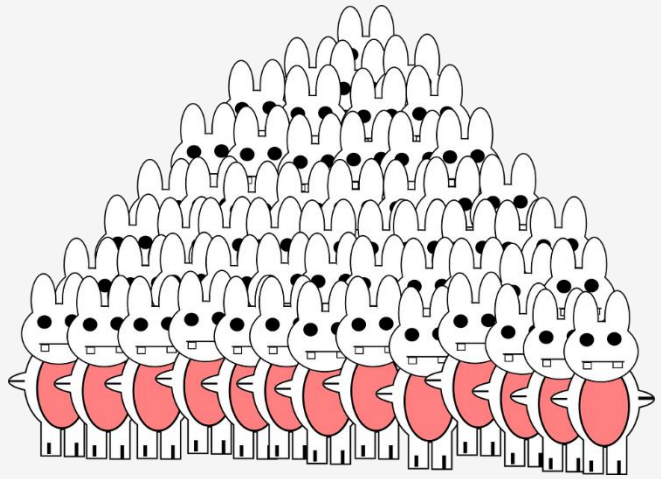
```
wydrukuj();
wydrukuj(„Ala”, „Ma”, „Kota”);
```



Var Args

Ograniczenia varargs:

- Każda metoda może mieć tylko jeden argument typu varargs (np. int..., String...)
- Varargs musi być podane jako ostatni argument tj:
  - wydrukuj(**String ... tekst, int a**) → 
  - wydrukuj(**int a, String... tekst**) → 



Var Args – zadanie

1. *Utworz klasę Kalkulator*
2. *Dodaj metodę **nie-statyczną** dodaj:int* przyjmującą dowolną ilość argumentów typu int
3. *Dodaj metodę **nie-statyczną** odejmij:int* przyjmującą dowolność ilość argumentów typu int
4. Utwórz klasę *Main*
5. W metodzie psvm klasy Main zainicjalizuj obiekt Kalkulatora
6. Przetestuj działanie metod
7. * Zamień klasę Main w samodzielny program pobierający dowolną ilość argumentów na wejściu, gdzie pierwszy argument oznacza typ działania + (dodawanie) – (odejmowanie)
8. * Zarchiwizuj projekt do postaci runnable-jar
9. * Przetestuj program z użyciem konsoli

Wyjątki

Czym są wyjątki w Javie?

Wyjątki – zdarzenia, które zakłocają normalny przebieg wykonywania programu

Wyjątki

Wyróżniamy 3 rodzaje wyjątków:

- **Wyjątki jawne (*checked exceptions*)**
  - Informują nas o potencjalnych problemach jakie mogą wystąpić przy korzystaniu z danej metody
  - Wymuszają obsługę błędu

Wyjątki

Wyróżniamy 3 rodzaje wyjątków:

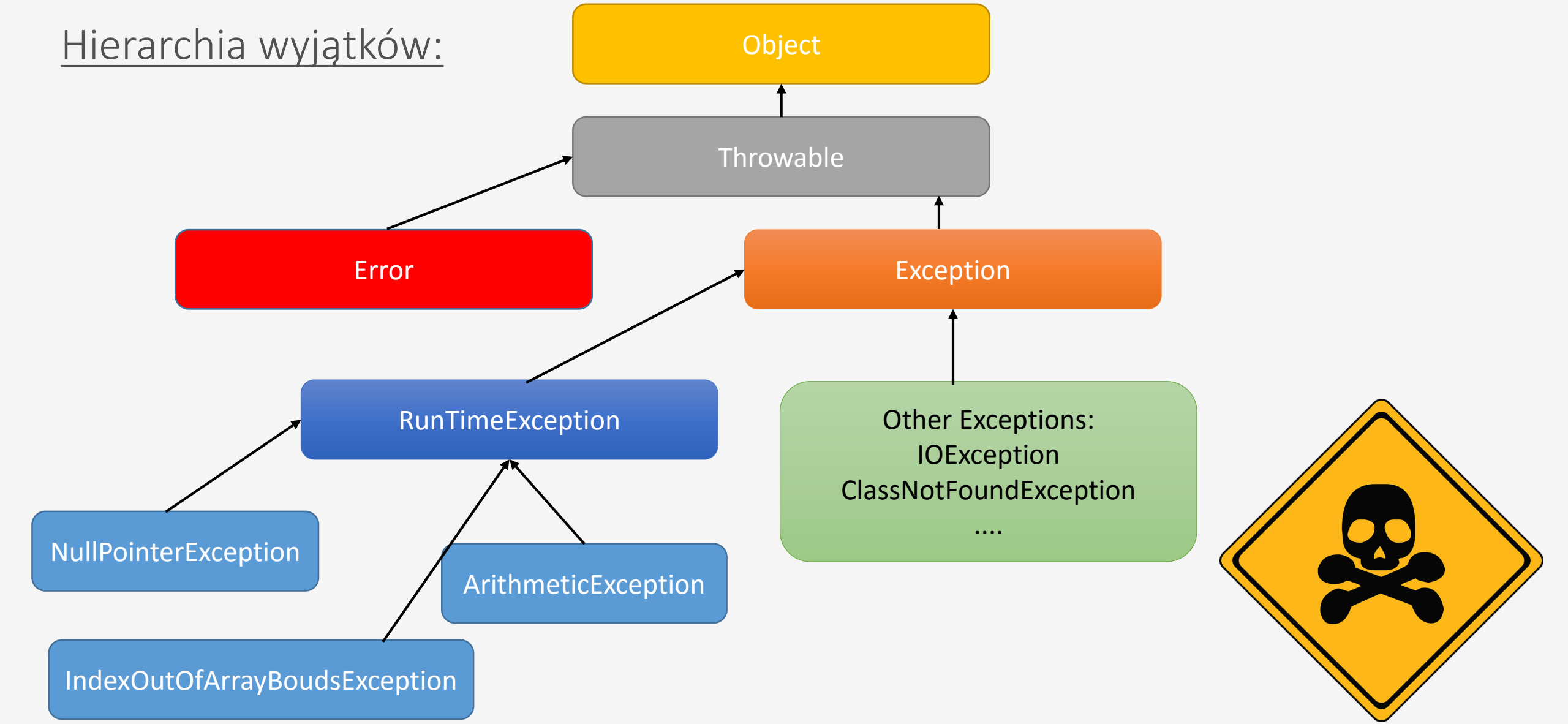
- **Wyjątki jawne (*checked exceptions*)**
- **Błędy (*Error*)**
  - Nieoczekiwane wydarzenia związane z problemami niezależnymi od napisanej aplikacji
  - Oracle rekomenduje, aby ich nie obsługiwać i zamiast tego wypisać w logach cały *stacktrace* zdarzenia

Wyjątki

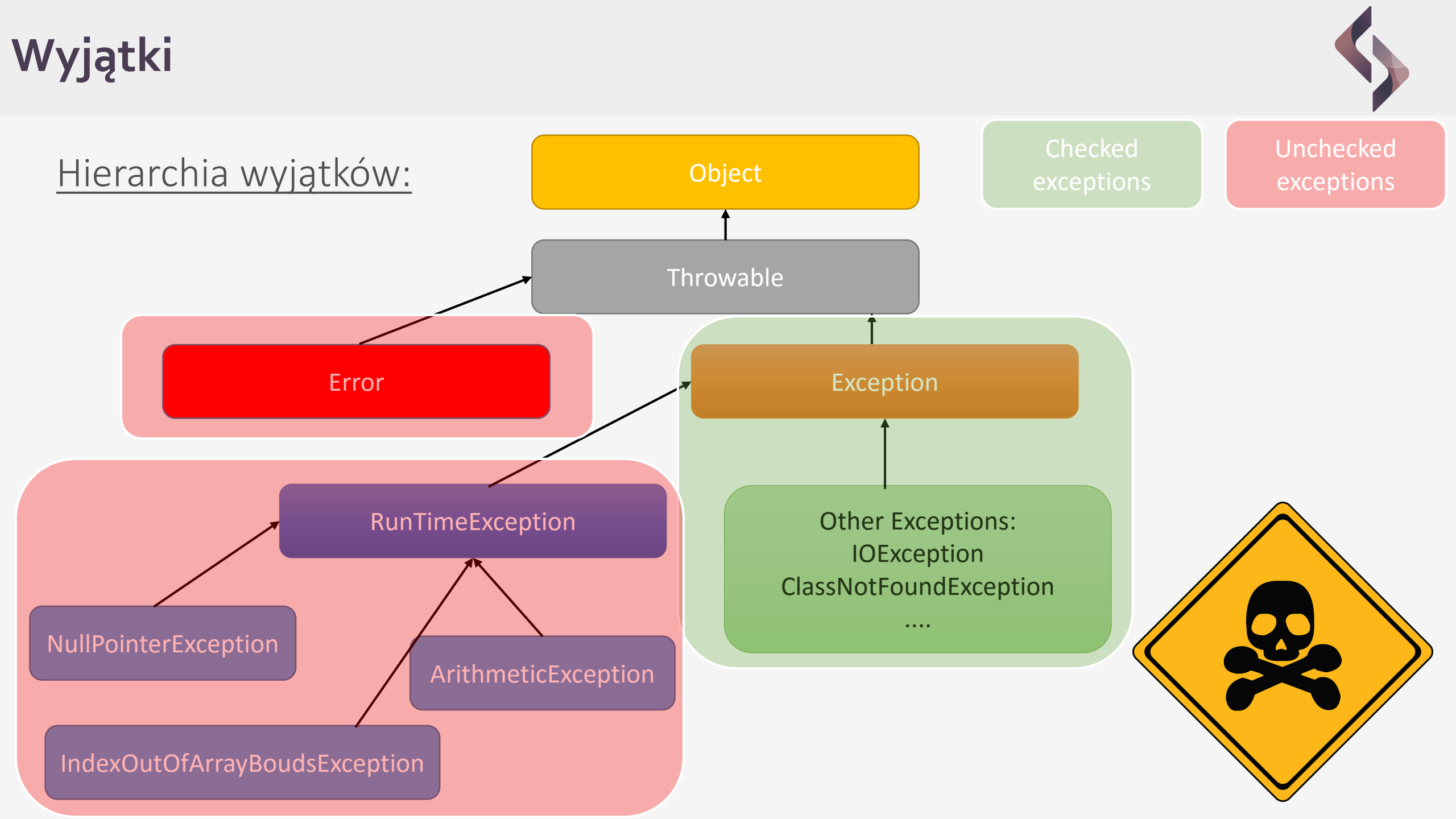
Wyróżniamy 3 rodzaje wyjątków:

- **Wyjątki jawne (*checked exceptions*)**
- **Błędy (*Error*)**
- **Wyjątki niejawne (*unchecked/runtime exceptions)***
  - Zdarzenia wynikające z logiki aplikacji, z którymi aplikacja nie powinna być sobie w stanie poradzić, takie jak:
    - Próba wykonania metody na pustej referencji (null)
    - Próba podzielenia przez zero
    - Odwołanie się do nieistniejącego indeksu tablicy

Wyjątki


Hierarchia wyjątków:
Object
Throwable
Error
Exception
RunTimeException
Other Exceptions:
IOException
ClassNotFoundException
....
NullPointerException
ArithmeticException
IndexOutOfArrayBoudsException

Wyjątki


Hierarchia wyjątków:
Object
Throwable
Error
Exception
RunTimeException
NullPointerException
ArithmeticException
IndexOutOfArrayBoudsException
Other Exceptions:
IOException
ClassNotFoundException
....
Checked exceptions
Unchecked exceptions

Wyjątki

Wyjątki jawne:

- Wywoływanie wyjątków odbywa się poprzez słowo kluczowe *throw*
- Wyjątki są obiektami dlatego przy ich wywoływaniu korzystamy z standardowego słówka *new*
- Wyjątki mogą posiadać opcjonalną wiadomość

```
throw new Exception(„Nie dziel przez zero!”);
```

Wyjątki

Wyjątki jawne:

- Deklaracja metody musi posiadać informację o potencjalnych wyjątkach jawnych

```
public int podziel(int a, int b) throws Exception{
...
}
```

Wyjątki

Obsługa wyjątków

- Wyjątki obsługujemy poprzez ich przechwycenie, a następnie wykonanie instrukcji w bloku catch
- Niezłapane wyjątki są propagowane do kolejnych metod wyższych (callStackTrace) aż do metody main

```
try{
...
}catch (ClassNotFoundException e){
...
}
```

Wyjątki

Obsługa wyjątków

- Wyjątki obsługujemy poprzez ich przechwycenie, a następnie wykonanie instrukcji w bloku catch
- Niezłapane wyjątki są propagowane do kolejnych metod wyższych (callStackTrace) aż do metody main

```
try{
...
}catch (ClassNotFoundException e){
...
}
```

Potencjalne miejsce wywołania wyjątku

Wyjątki

Obsługa wyjątków

- Wyjątki obsługujemy poprzez ich przechwycenie, a następnie wykonanie instrukcji w bloku catch
- Niezłapane wyjątki są propagowane do kolejnych metod wyższych (callStackTrace) aż do metody main

```
try{
...
}catch (ClassNotFoundException e){
...
}
```

Informacja jaki wyjątek chcemy obsłużyć

Wyjątki

Obsługa wyjątków

- Wyjątki obsługujemy poprzez ich przechwycenie, a następnie wykonanie instrukcji w bloku catch
- Niezłapane wyjątki są propagowane do kolejnych metod wyższych (callStackTrace) aż do metody main

```
try{
...
}catch (ClassNotFoundException e){
...
}
```

Blok kodu, który zostanie wykonany w celu poprawnej obsługi wyjątku

Wyjątki

Obsługa wielu wyjątków

- Możemy obsłużyć w różny sposób różne wyjątki
- W zależności od typu wyjątku zostanie wykonany właściwy fragment kodu

```
try{
...
}catch (ClassNotFoundException e){
...
} catch (IOException e){
...
}
```

Wyjątki

Dodatkowy blok finally

- Blok jest wykonywany niezależnie od wystąpienia wyjątku lub jego braku

```
try{
...
}catch (ClassNotFoundException e){
...
} catch (IOException e){
...
} finally {
...
}
```

Wyjątki – zadanie

1. *Dodaj klasę Main podaną przez prowadzącego do swojego projektu*
2. *Zaimplementuj klasę Konto, tak aby kod się kompilował*
3. *Zidentyfikuj miejsca gdzie działanie aplikacji może zostać zaburzone*

Wyjątki – zadanie

1. *Dodaj klasę Main podaną przez prowadzącego do swojego projektu*
2. *Zaimplementuj klasę Konto, tak aby kod się kompilował*
3. *Zidentyfikuj miejsca gdzie działanie aplikacji może zostać zaburzone*
4. *Dodaj blok try catch w metodzie pobierzLiczbe, tak aby:*
    1. *Wyświetlić komunikat o nieprawidłowej wartości wprowadzonej przez użytkownika*
    2. *Zwrócić liczbę typu int z powrotem do bloku main*
5. *Przetestuj rozwiązanie*
6. *Rzuć wyjątkiem jawnym gdy użytkownik próbuje podać kwotę większą od swojego stanu konta*
7. ** Obsłuż odpowiednio rzucany wyjątek z punktu 6*

Wyjątki

Wyjątki niejawne:

- Użycie podobne jak przy wyjątkach jawnych
- Brak deklaracji *‚throws’* w metodzie
- Kompilator nie wymusza obsługi wyjątku
- Nieobsłużony wyjątek wpadając do bloku *main* zakończy działanie programu

```
throw new RunTimeException(„Nie dziel przez zero!”);
```

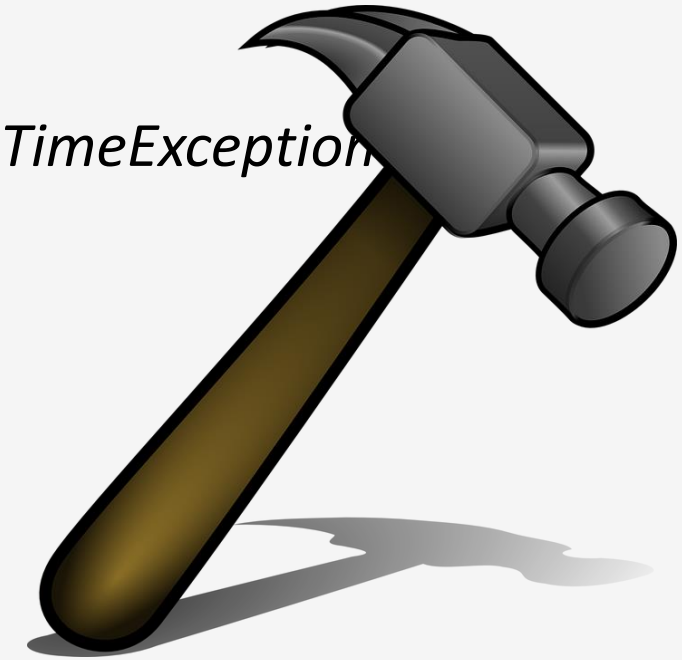
Wyjątki

Tworzenie własnych wyjątków:

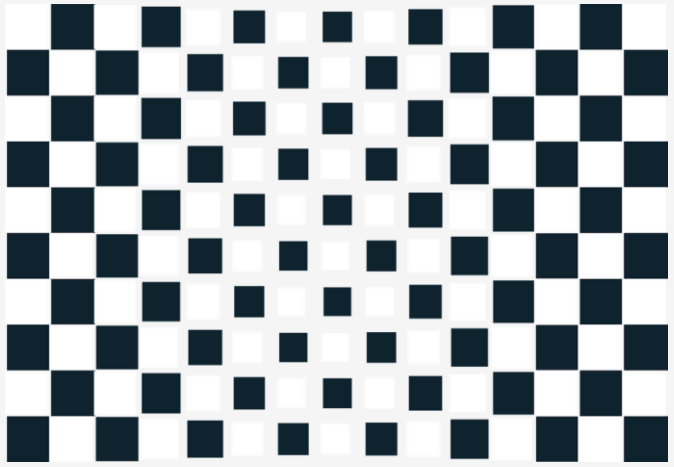
- Zamiast korzystać z predefiniowanych wyjątków możemy bardzo łatwo tworzyć własne
- Samodzielnie utworzony wyjątek może stanowić bardziej czytelny błąd
- Pomaga również unikać podejścia *‚złap je wszystkie’* 
- W celu utworzenia własnego wyjątku wystarczy odziedziczyć po innym wyjątku ☺

Wyjątki – zadanie

1. *Utwórz nowy wyjątek w pakiecie pl.sda.wyjatki.bank*
2. Nazwij go *InsufficientFundsException*
3. *Odziedzicz po klasie Exception*
4. *Rzuć utworzony wyjątek gdy użytkownik próbuje wybrać wyższą kwotę niż aktualny stan jego konta*
5. *Zmień oczekiwany wyjątek w bloku catch*
6. * Zobacz jak zmieni się użycie przy oddziedziczeniu po *RunTimeException*



Wyrażenia regularne



Czym są wyrażenia regularne?

Wyrażenia regularne (*regular expressions, regex/regexp)* – wzorce opisujące łańcuchy symboli. Wyrażenie regularne może określać zbiór pasujących łańcuchów lub wyszczególniać istotne części łańcucha

Posiadają implementację w większości języków programowania w tym w Javie

Wyrażenia regularne

Klasa String:

- Stanowi **niezmienną tablicę znaków**
- Możemy odwoływać się bezpośrednio do wartości (literał) :

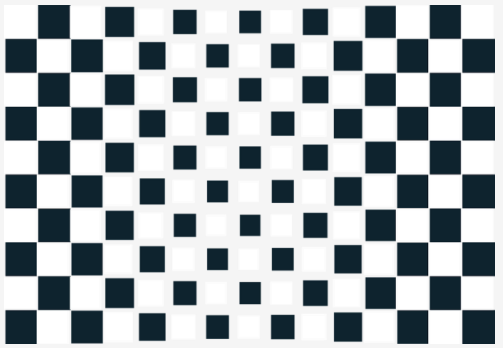
```
String str1 = „Hello World!”; // literał
```

- Jak również poprzez obiekt

```
String str2 = new String(„Hello World!”);
```

- == vs *equals* :

```
str1==str2; //false
str1.equals(str2); //true
```

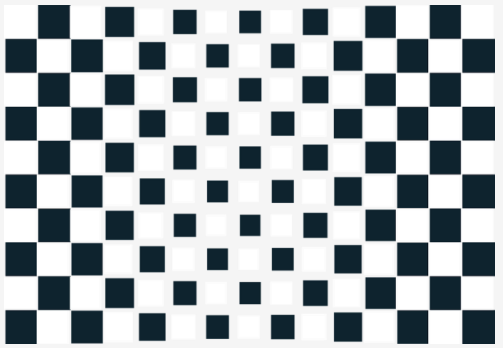


Wyrażenia regularne

Klasa String:

- Klasa jest finalna
- Nie posiada metody typu **set**
- Oznacza to że klasa jest *niezmienna (immutable)*
- Wszystkie operacje wykonywane na klasie String powodują utworzenie nowego obiektu typu String

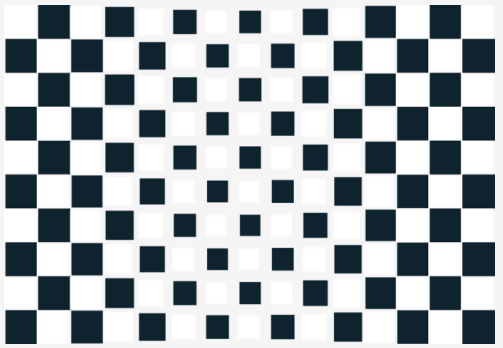
```
String hello = „Hello”;
System.out.println(hello.concat(„ World!”));
//Hello World!
System.out.println(hello);
//Hello
```



Wyrażenia regularne

Istotne metody w klasie String:

- length(): int – zwraca długość łańcucha znaków
- charAt(): char – zwraca znak o danym indeksie
- indexOf(): int – zwraca indeks pierwszego wystąpienia znaku
- substring(): String – zwraca wybrany fragment ciągu znaków
- toLowerCase (): String – zamienia na małe
- toUpperCase(): String – zamienia na duże znaki
- equals(): boolean – czy równe
- equalsIgnoreCase(): boolean – czy równe bez względu na wielkość liter
- startsWith(): boolean – czy łańcuch znaków zaczyna się od ...
- endsWith(): boolean – czy łańcuch znaków kończy się na ...
- contains(): boolean – czy łańcuch znaków zawiera wybrany tekst
- replace(): String – zamiana podłańcucha na wybrany inny
- trim(): String – obcina białe znaki z obu stron łańcucha

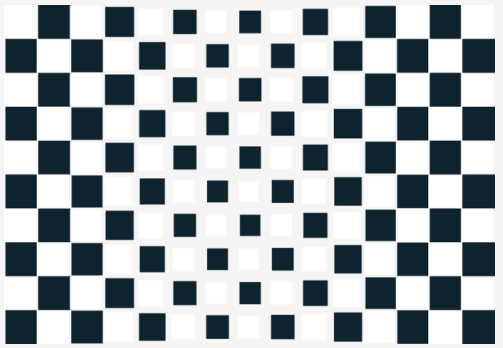


Wyrażenia regularne

String – łącznie metod:

- Na wyniku działania jednej metody możemy wywołać następną co nazywa się łączeniem metod (*method chaining*)

```
String str = „ Hello World! ”;
System.out.println(str.trim().replace(„World”, „Poland”)
                .toUpperCase());
// „HELLO POLAND!”
```

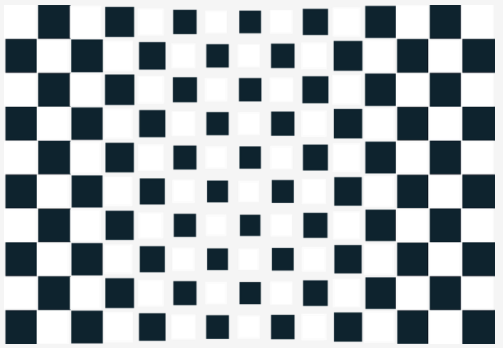


Wyrażenia regularne

StringBuilder vs StringBuffer:

- StringBuffer – poprzednik StringBuildera, napisany z myślą o wielowątkowości (metody są synchronizowane)
- StringBuilder – dodany w JDK 1.5, szybszy od StringBuffera, ale nie zapewnia odpowiedniej obsługi w aplikacjach wielowątkowych

```
StringBuilder builder = new StringBuilder(„123”);
System.out.println(builder.reverse().toString()); // 321
```

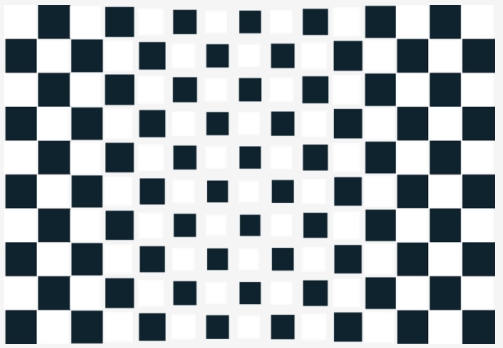


Wyrażenia regularne

Wyrażenia regularne

- Pozwalają opisywać wzorce tekstu – określać wzór łańcucha znaków
- Stosowane przy wyszukiwaniu oraz modyfikowaniu tekstu
- Przykład – wyszukiwanie tekstu Hello ... ! , gdzie pomiędzy hello a wykrzyknikiem jest sam tekst:

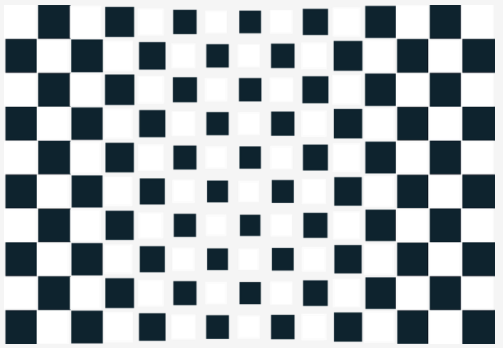
Hello \D*!



Wyrażenia regularne

Składnia – wybór znaków

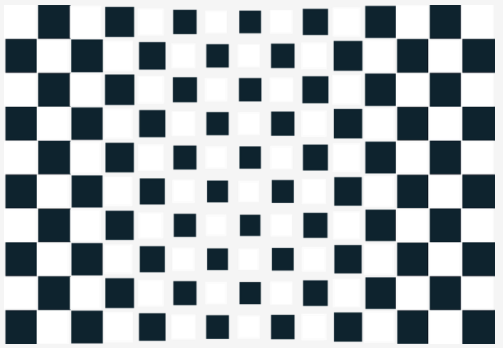
- . – dowolny znak
- \d – cyfra [0-9]
- \D – inny znak niż cyfra ([^0-9])
- \s – znak biały
- \S – inny znak niż znak biały
- \w – znak [a-zA-Z_0-9]
- \W – inny znak niż \w (np. .';,{}[]@!)



Wyrażenia regularne

Składnia – wybór liczebników

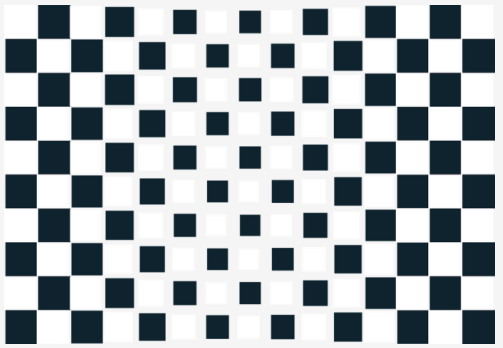
- ? – 0 lub 1 wystąpienie
- * – 0 lub więcej wystąpień
- + – 1 lub więcej wystąpień
- {n} – dokładnie n wystąpień (np. {5})
- {n,} – przynajmniej n wystąpień
- {n,m} – przynajmniej n lecz nie więcej niż m wystąpień



Wyrażenia regularne

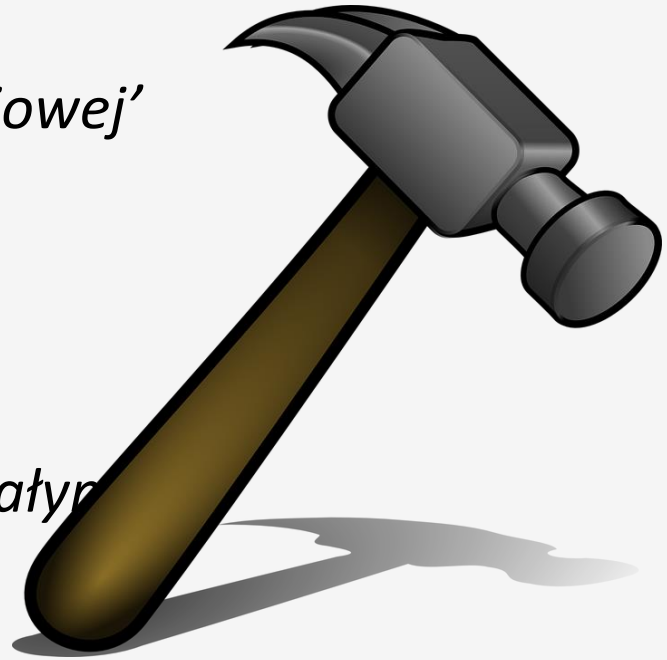
Składnia – meta znaki

- \ – wskazanie, że interesuje nas konkretny znak (np. szukając kropki - \.)
- ^ – oznaczenie początku linii
- $ – oznaczenie końca linii
- | – alternatywa (np. a|s → albo ‚a' albo ‚s')
- () – grupowanie znaków (np. (Hello|World!) → albo ‚Hello', albo ‚World!')
- [] – zbiór znaków (np. \s[a-z]+\s? → wszystkie małe wyrazy oddzielone spacją



Wyrażenia regularne – zadanie

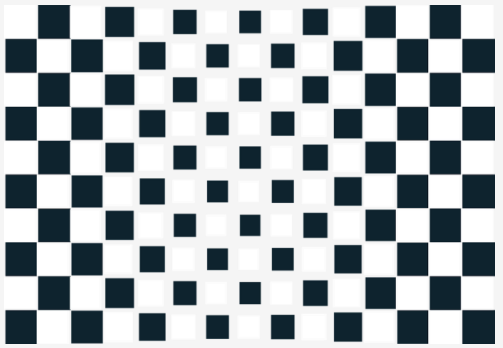
1. *Przejdź na stronę https://regex101.com*
2. *Napisz wyrażenie regularne dla odkrywania wieku osoby z opisu:*
    1. *Roman, lat 30 studiuje prawo i ćwiczy cross-fit*
    2. *Antoni, w wieku 101 lat odpoczywa na emeryturze*
3. *Napisz wyrażenie regularne dla pobrania kodu pocztowego XX-XXX*
4. *Napisz wyrażenie regularne dla pobrania nazwy ulicy:*
    1. *Mieszkam na ul. Armii Krajowej 15A/8 → ‚ul. Armii Krajowej'*
    2. *Przenoszę się na ul. Dworcowa 1 → ‚ul. Dworcowa'*
5. *Napisz wyrażenie regularne pobierające datę*
    1. *Data ur: 11.11.2000 → 11.11.2000*
    2. *Data dzisiejsza: 09.01.2017 → 09.01.2017*
6. *Napisz wyrażenie regularne pobierające słowa napisane małymi literami w zakresie a-k tj:*
    1. *„Bogdan ma dziś kaca" → „kaca"*
    2. *„Tomasz je pizze" → „je"*



Wyrażenia regularne

Wyrażenia regularne w Javie

- Metody w klasie String:
  - matches(**String regexp**):boolean – zwraca true jeśli zadany łańcuch znaków posiada chociaż 1 wystąpienie wzorca
  - split(**String regexp**):String[] – rozbija łańcuch znaków poprzez wycięcie wzorca i odseparowanie tak powstałych łańcuchów w postać tablicy
  - replaceAll(**String regexp, String replacement**) –podmienia wystąpienie wzorca z wskazanym łańcuchem znaków
  - replaceFirst(**String regexp, String replacement**) – podmienia jedynie pierwsze wystąpienie wzorca z wskazanym łańcuchem znaków

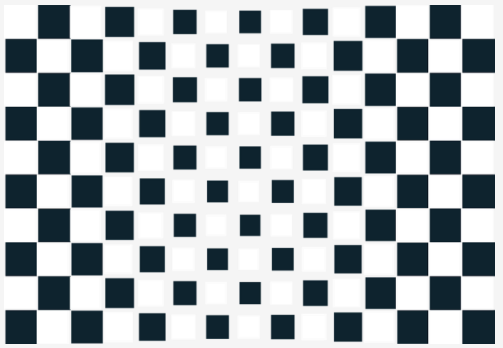


Wyrażenia regularne

Wyrażenia regularne w Javie

- *java.util.regex.Pattern* – pozwala na zapis poszukiwanego wzorca
- *java.util.regex.Matcher* – służy do odnajdywania wzorca wewnątrz wybranego łańcucha znaków

```
Pattern pattern = Pattern.compile("(Hello|World!)");
Matcher matcher = pattern.matcher( input: "Hey hop Hello World! World! Hello");
while(matcher.find()){
    System.out.println(matcher.group());
}
```



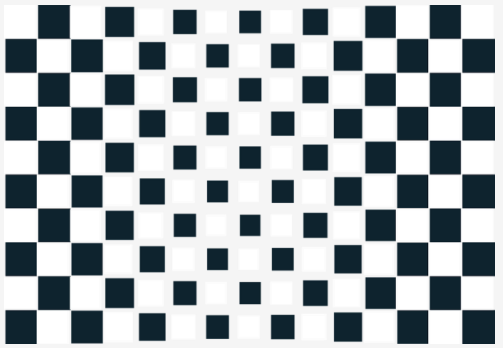
Wyrażenia regularne

Wyrażenia regularne w Javie

- *java.util.regex.Pattern* – pozwala na zapis poszukiwanego wzorca
- *java.util.regex.Matcher* – służy do odnajdywania wzorca wewnątrz wybranego łańcucha znaków

```
Pattern pattern = Pattern.compile("(Hello|World!)");
Matcher matcher = pattern.matcher( input: "Hey hop Hello World! World! Hello");
while(matcher.find()){
    System.out.println(matcher.group());
}
```

Hello
World!
World!
Hello



Wyrażenia regularne – zadanie 2

1. *Utwórz nowy projekt ‚Wyrazenia Regularne'*
2. *Utwórz pakiet pl.sda.regexp.cwiczenia*
3. *Utwórz klasę WyrazeniaRegularne*
4. *Utwórz metodę ‚zlacz' – która pobiera 2 ciągi typu string, usuwa z nich białe znaki na początku i końcu a następnie usuwa 2 litery z początku i końca i zwraca połączenie obu tych łańcuchów*
5. *Utwórz metodę do usuwania samogłosek z wybranego ciągu znaków*
6. *Utwórz metodę do usuwania liczb większych niż 9 podanych w zadanym ciągu znaków*
7. *Utwórz metodę zamieniającą słowa z dużej litery na wybrany tekst (2 parametr metody) np. „Ala ma kota","???" → „??? ma kota"*
8. *Utwórz metodę zamieniającą koniec zdania na koniec zdania + znak nowej linii.*
9. ** Usprawnij metodę z 8 zadania, tak aby to był rzeczywiście koniec zdania a nie ‚np. X' → ‚np.\nX'*