



Czym są wyjątki w Javie?

Wyjątki – zdarzenia, które zakłócają normalny przebieg wykonywania programu



Wyróżniamy 3 rodzaje wyjątków:

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





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





Wyróżniamy 3 rodzaje wyjątków:

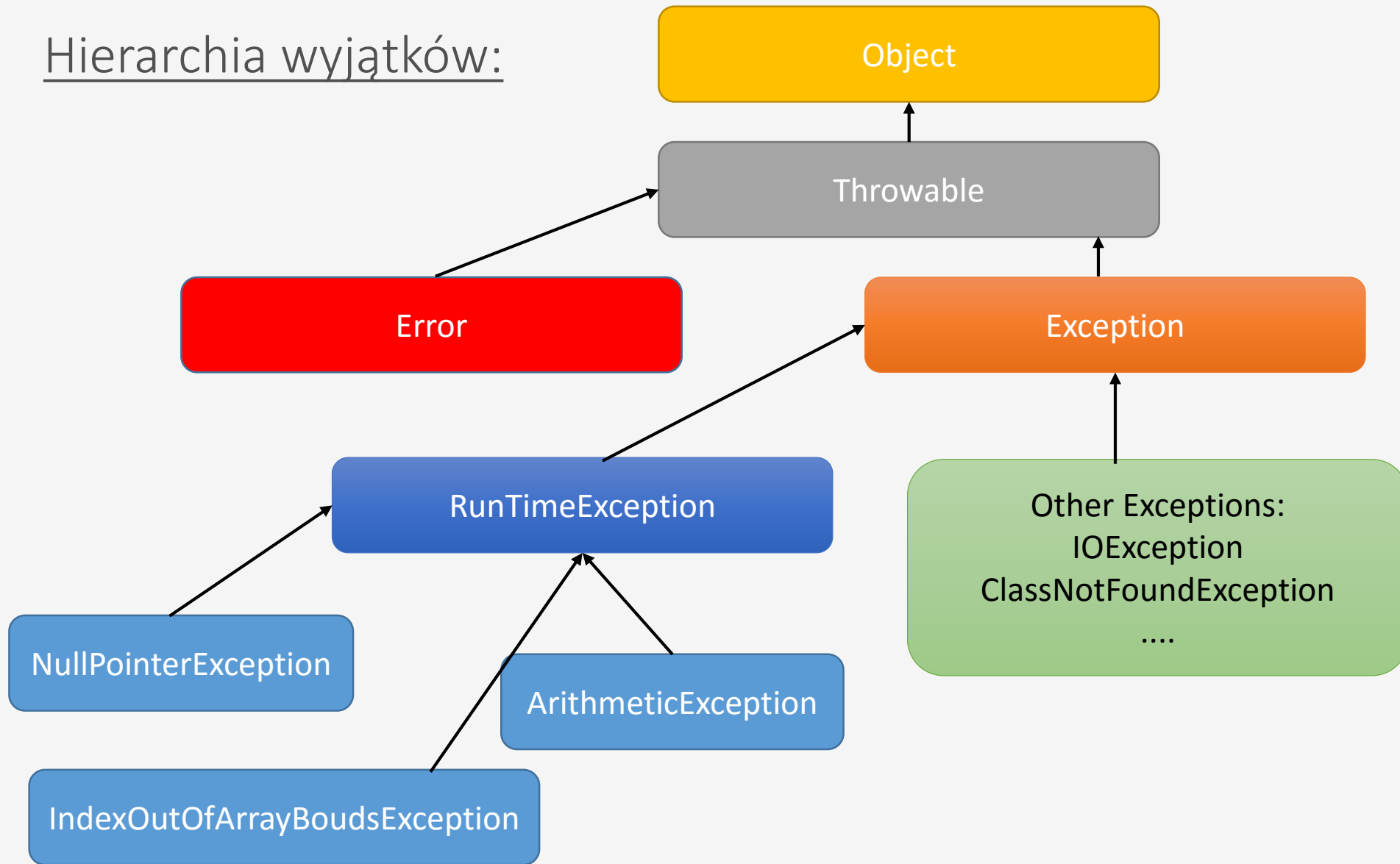
- Wyjątki jawne (*checked exceptions*)
- Błędy (*Error*)
- Wyjątki niejawne (*unchecked/runtime exceptions*)
  - Zdarzenia wynikające z logiki aplikacji, z którymi aplikacja nie powinna być sobie w stanie poradzić, takie jak:
    - Próba wykonania metody na pustej referencji (null)
    - Próba podzielenia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



# Wyjątki



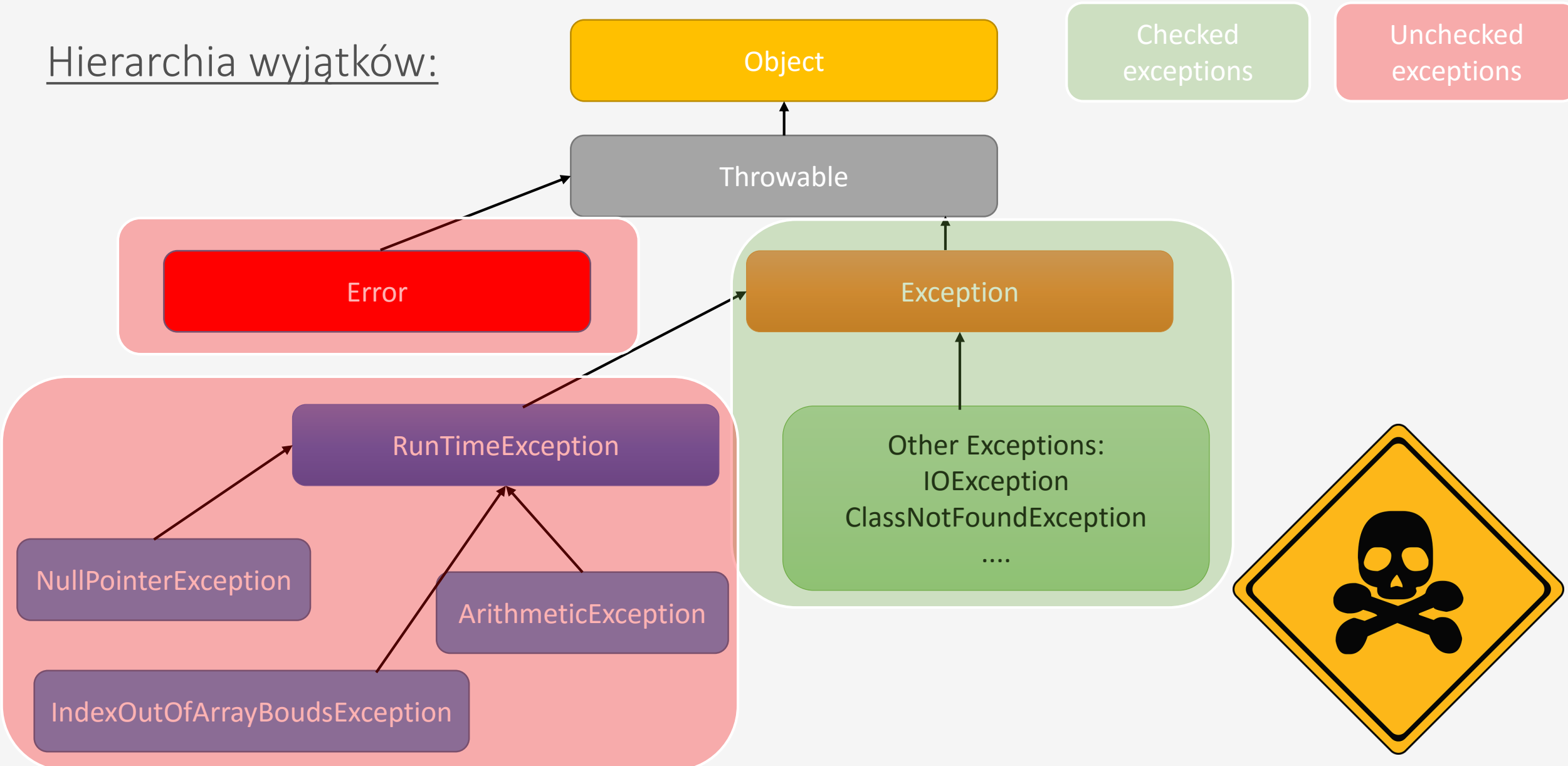
Hierarchia wyjątków:



# Wyjątki



## Hierarchia wyjątków:





## 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 jawne:

- Deklaracja metody musi posiadać informację o potencjalnych wyjątkach jawnych

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

```
...
```

```
}
```







## 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){  
    ...  
}
```





## 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





## 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





## 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





## 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){  
...  
}
```





## 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 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!");
```





## 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`*

