	Wydział Politechniczny **Katedra Informatyki**
Prowadzący	Tomasz Gądek
Kurs	Narzędzia i środowiska programistyczne
Rok / Semestr	1 / Letni
Temat	Lab04: Podstawy programowania w języku Kotlin cz. 4.

Data ostatniej modyfikacji: **01-10-2024**

Wyjątek

Wyjątki w języku Kotlin są to sytuacje, które występują podczas działania programu i powodują przerwanie normalnego przebiegu programu. Mogą być one wynikiem błędnych danych podczas wykonywania programu. W Kotlinie, jak w wielu innych językach programowania, wyjątki są obsługiwane za pomocą bloków **try/catch/finally**.

```
fun main() {
   print("Get digit: ")
   val userInput = readln()
   val number: Int = userInput.toInt()

   println("It is: $number")
}
```

Gdy użytkownik poda znak innego typu niż liczba całkowita (np. **a**), próba rzutowania **userInput** na typ **Int** spowoduje wystąpienie wyjątku.

```
Exception in thread "main" java.lang.NumberFormatException: For
input string: "a"
    at
java.base/java.lang.NumberFormatException.forInputString(NumberFo
rmatException.java:67)
    at java.base/java.lang.Integer.parseInt(Integer.java:665)
    at java.base/java.lang.Integer.parseInt(Integer.java:781)
    at MainKt.main(Main.kt:4)
    at MainKt.main(Main.kt)
```

W tym przypadku, gdy użytkownik poda znak **a**, próba rzutowania **userInput** na typ **Int** spowoduje wystąpienie wyjątku **NumberFormatException**, ponieważ nie można przekonwertować **a** na liczbę całkowitą.

Obsługa wyjątków za pomocą **try/catch/finally**:

```
fun main() {
    try {
        print("Get digit: ")
        val userInput = readln()
        val number: Int = userInput.toInt()

        println("It is: $number")
    } catch (e: NumberFormatException) {
        println("We have a problem: ${e.message}")
    } finally {
        println("End")
    }
}
```

W powyższym przykładzie używamy bloku **try** do wykonania kodu, który może spowodować wyjątek. Jeśli wystąpi wyjątek **NumberFormatException**, zostanie on przechwycony przez blok **catch**, który wyświetli odpowiedni komunikat. Blok **finally** zostanie zawsze wykonany, niezależnie od tego, czy wystąpił wyjątek czy nie. Dzięki temu zapewniamy, że program będzie miał odpowiednią obsługę błędów i zawsze będzie kończył swoje działanie w odpowiedni sposób.

Spróbuj wykonać testy programu podając prawidłowe / nieprawidłowe dane wejściowe. Zaobserwuj wyniki działania programu.

Funkcja, ponowne spojrzenie

Parametry domyślne

Funkcja **greet()** została zaprojektowana w taki sposób, aby można było powitać użytkownika za pomocą dowolnego tekstu przy użyciu domyślnych parametrów. W języku Kotlin możemy zdefiniować parametry funkcji z wartościami domyślnymi, co oznacza, że jeśli nie podamy wartości dla tych parametrów, zostaną one automatycznie użyte.

```
fun greet(user: String = "Anonymous", greeting: String = "Hello")
{
    println("$greeting, $user!")
}

fun main() {
    greet("Mike", "Good morning")
    greet("Mike")
    greet()
}
```

Rezultat jest następujący.

```
Good morning, Mike!
Hello, Mike!
Hello, Anonymous!
```

Gdy wywołujemy funkcję **greet()**, możemy przekazać nazwę użytkownika, którego chcemy powitać, oraz tekst powitania. Jeśli nie podamy tych wartości, funkcja automatycznie przyjmie wartości domyślne: **Anonymous** jako użytkownik i **Hello** jako tekst powitania.

Funkcja jednowyrażeniowa

Jeżeli funkcja ma zwrócić wartość obliczoną na podstawie jednego wyrażenia, można użyć składni jednowyrażeniowej. Nie trzeba pisać nawiasów klamrowych ani instrukcji powrotu **return**. Kotlin automatycznie zwróci wartość z ostatniego wyrażenia w funkcji.

Oto prosty przykład funkcji jednowyrażeniowej, która oblicza sumę liczb całkowitych.

```
fun add(a:Int, b:Int) = a + b

fun main() {
   val a = 1
   val b = 2
   println("$a + $b = ${add(a, b)}")
}
```

Rezultat jest następujący.

```
1 + 2 = 3
```

W powyższym przykładzie **a + b** jest jedynym wyrażeniem w funkcji **add()**, więc nie musimy używać nawiasów klamrowych ani słowa kluczowego **return**. Gdy funkcja **add()** zostanie wywołana, zwróci sumę liczb przekazanych jako argumenty.

Funkcje jednowyrażeniowe są przydatne w przypadku prostych operacji, gdzie nie ma potrzeby definiowania rozbudowanego ciała funkcji. Pomagają one upraszczać i skracać kod, co sprawia, że jest bardziej czytelny i zwięzły.

Nazwane argumenty funkcji

Funkcja **greet()** przyjmuje trzy argumenty: **name**, **age** i **city**. Wywołując tę funkcję, możemy przekazać wartości do tych parametrów w dowolnej kolejności, podając ich nazwy.

```
fun greet(name: String, age: Int, city: String) {
   println("Hello, $name! You are $age years old and you live in
$city.")
}

fun main() {
   greet("John", 25, "New York")

   println()

   greet(age = 30, name = "Emily", city = "London")
   greet(age = 30, city = "London", name = "Emily")
   greet(city = "London", age = 30, name = "Emily")
}
```

Rezultat.

```
Hello, John! You are 25 years old and you live in New York.

Hello, Emily! You are 30 years old and you live in London.
Hello, Emily! You are 30 years old and you live in London.
Hello, Emily! You are 30 years old and you live in London.
```

Przeciążanie nazwy funkcji

Przeciążanie nazwy funkcji w języku Kotlin oznacza tworzenie wielu funkcji o tej samej nazwie, ale różniących się zestawem parametrów. Wywołując funkcję o konkretnej nazwie, kompilator Kotlin może rozpoznać, która wersja funkcji zostanie wywołana na podstawie przekazanych argumentów.

Aby funkcja **add()** mogła dodawać liczby całkowite oraz liczby zmiennoprzecinkowe, możemy zdefiniować dwie różne wersje tej funkcji: jedną dla typów **Int** i drugą dla typów **Double**.

Oto przykład.

```
fun add(x: Int, y: Int): Int {
   return x + y
}

fun add(x: Double, y: Double): Double {
   return x + y
}

fun main() {
   val sum1 = add(3, 5)
   val sum2 = add(2.5, 3.7)

   println(sum1)
   println(sum2)
}
```

Rezultat działania programu jest następujący.

```
8
6.2
```

Kolekcja - lista

W języku Kotlin lista jest to kolekcja, która może być modyfikowana i przechowuje elementy w określonej kolejności.

Podstawowe operacje na liście.

```
fun main() {

    print("1) Tworzenie pustej listy, która ")
    println("przechowuje ciągi tekstowe (String)...")
    val names = mutableListOf<String>()
    // val names = mutableListOf("Mike", "Tom")

    println("2) Dodawanie elementów do listy...")
    names.add("Mike")
    names.add("Tom")

    println("3) Wyświetlanie całej listy...")
    println(names)

    println("4) Dodawanie elementu w konkretne miejsce...")
    println("Dodaje element 'Monica' za 'Mike': ")
    names.add(1, "Monica")
    println(names)

    println("5) Usuwam element o idx = 0...")
    names.removeAt(0)
    println(names)

    print("6) Aktualizuje element ('Monica' zostanie ")
    println("nadpisana przez 'Alex') o idx = 0...")
    names[0] = "Alex"
    println(names)

    println("7) Pobieram drugi element z listy: ${names[1]}")

    print("8) Czy lista zawiera imię 'Alex'?: ")
    println("${names.contains("Alex")}")

    print("9) Wywołanie funkcji 'showElements()'. ")
    println()"Parametrem jest lista 'names'..")
    showElements(names)
}

private fun showElements(names: MutableList<String>) {
    for(name in names) {
        println(name)
    }
}
```

Rezultat.

```
1) Tworzenie pustej listy, która przechowuje ciągi tekstowe
(String)..
2) Dodawanie elementów do listy..
3) Wyświetlanie całej listy..
[Mike, Tom]
4) Dodawanie elementu w konkretne miejsce..
Dodaje element 'Monica' za 'Mike':
[Mike, Monica, Tom]
5) Usuwam element o idx = 0...
[Monica, Tom]
6) Aktualizuje element ('Monica' zostanie nadpisana przez 'Alex')
o idx = 0...
[Alex, Tom]
7) Pobieram drugi element z listy: Tom
8) Czy lista zawiera imię 'Alex'?: true
9) Wywołanie funkcji 'showElements()'. Parametrem jest lista
'names'..
Alex
Tom
```

Kolekcja - mapa

W języku Kotlin mapa jest to struktura danych, która przechowuje pary **klucz-wartość**. Każdy element w mapie składa się z klucza i odpowiadającej mu wartości. Klucze w mapie są unikalne, co oznacza, że każdy klucz może być przypisany tylko do jednej wartości. Mapy w Kotlinie są bardzo przydatne do przechowywania danych w formie par klucz-wartość i umożliwiają szybki dostęp do wartości na podstawie ich klucza.

Podstawowe operacje na mapie:

```
fun main() {

    print("1) Tworzenie mapy z gotowymi elementami ")
    println("key (String) -> value (Int)...")
    val map = mutableMapOf("one" to 1, "two" to 2)
    // val map = mutableMapOf<String, Int>()
    println(map)

    println("2) Dodanie elementu do mapy...")
    map["three"] = 3
    println(map)

    println("3) Wyswietlanie mapy...")
    println(map)

    println("4) Aktualizacja elementu w mapie...")
    map["three"] = 33
    println(map)

    print("5) Usuwanie elementu o zadanym ")
    println("kluczu ('three')...")
    map.remove("three")
    println(map)

    print("6) Pobranie elementu o kluczu 'one' ")
    println("z mapy: ${map["one"]}")
    print("7) Pobranie elementu o kluczu, który ")
    println("nie istnieje: ${map["three"]}")

    print("8) Dostęp do wszystkich ")
    println("kluczy: ${map.keys}")
    print("9) Dostęp do wszystkich wartosci: ")
    prinltn("${map.values}")

    print("10) Wywołanie funkcji 'showElements()'. ")
    prinltn("Parametrem jest mapa 'map'..")
    showElements(map)
}

private fun showElements(elements: Map<String, Int>) {
    for((key, value) in elements) {
        println("key = $key, value = $value")
    }
}
```

Rezultat.

```
1) Tworzenie mapy z gotowymi elementami key (String) -> value
(Int)..
{one=1, two=2}
2) Dodanie elementu do mapy..
{one=1, two=2, three=3}
3) Wyswietlanie mapy..
{one=1, two=2, three=3}
4) Aktualizacja elementu w mapie..
{one=1, two=2, three=33}
5) Usuwanie elementu o zadanym kluczu ('three')..
{one=1, two=2}
6) Pobranie elementu o kluczu 'one' z mapy: 1
7) Pobranie elementu o kluczu, który nie istnieje: null
8) Dostęp do wszystkich kluczy: [one, two]
9) Dostęp do wszystkich wartosci: [1, 2]
10) Wywołanie funkcji 'showElements()'. Parametrem jest mapa
'map'..
key = one, value = 1
key = two, value = 2
```

Zadania

- Napisz funkcję jednowyrażeniową, która wyznaczy najmniejszy element spośród przekazanej pary argumentów typu **Int**.
- Napisz funkcję, która usunie liczby nieparzyste z listy (funkcja powinna przyjmować listę liczb całkowitych oraz zwracać listę, która posiada jedynie liczby parzyste).
- Napisz funkcję, która przyjmuje jako parametr listę imion (**String**), a zwróci sumę znaków wszystkich imion znajdujących się w kolekcji.
- Napisz funkcję, która konwertuje cyfry rzymskie na arabskie od I do IX (Mapa - klucz to **String**, a wartość to **Int** - zaimplementuj mapę z kluczami od "I" do "IX"). Funkcja powinna zwracać **Int?** (jeżeli podana cyfra nie istnieje w mapie to zwracamy **null**). Funkcja powinna przyjmować cyfrę rzymską np. **"I"**. W odpowiedzi powinniśmy otrzymać cyfrę arabską, czyli **1**.
- Napisz funkcję, która zasymuluje logowanie użytkownika do systemu (Mapa - klucz to **String / Login**, a wartość to **String / Hasło**). Funkcja powinna przyjmować **login** i **hasło** użytkownika. Funkcja powinna zwracać **true**, gdy logowanie się powiedzie. W przeciwnym wypadku funkcja powinna zwracać **false**.

Czy wiesz, że...

Pierwsza strona internetowa została stworzona przez Tima Bernersa-Lee, który jest uważany za twórcę **World Wide Web** (**WWW**). Berners-Lee jest brytyjskim informatykiem, który pracował w Europejskim Ośrodku Badań Jądrowych (**CERN**) w Genewie, w Szwajcarii. To właśnie tam, w 1989 roku, Berners-Lee opracował koncepcję **WWW** jako sposobu udostępniania i przeglądania dokumentów za pośrednictwem internetu.

Pierwsza strona internetowa została uruchomiona publicznie 6 sierpnia 1991 roku.

http://info.cern.ch (adres pierwszej strony internetowej).

Warto zauważyć, że pierwsza strona internetowa była bardzo prosta w porównaniu z dzisiejszymi standardami. Składała się głównie z tekstu i prostych linków. Jednakże jej powstanie otworzyło drogę do rewolucji w sposobie, w jakim ludzie komunikują się, zdobywają informacje i korzystają z zasobów Internetu.