Wykład 1

Podstawy, przegląd biblioteki standardowej, Java w zastosowaniach

---

JDK – **Java Development Kit** (zawiera kompilator: javac)

JRE– **Java Runtime Environment**, środowisko uruchomieniowe dla programów napisanych w języku Java. Składa się z wirtualnej maszyny Javy, biblioteki standardowej oraz plików pomocniczych. (zawiera komendę 'java' do uruchamiania programu, ale **nie ma kompilatora**)

PRYMITYWNE TYPY DANYCH:

- byte (8-bit), short (16-bit), int (32-bit), long (64-bit) — *typ całkowitoliczbowy*
- float (32-bit), double (64-bit) — *liczby rzeczywiste*
- boolean (1-bit) - flaga — *typ logiczny prawda lub fałsz*
- char (16-bit) – znak w unikodzie, np. \u015b — *znak zapisany w unikodzie, w tym przypadku "ś"*

Reszta to **OBIEKTOWE** typy danych (String, PrintStream, ...)

Struktura danych "*args*" (argumenty programu) posiada atrybut length

```
for(i=0; i<args.length; i++)
    System.out.printf(Locale.US, "%.2f\n", args[i]);
```

- metody typu length() można wywoływać na obiektach, na tablicach można wywoływać atrybuty

Locale

W języku Java, **Locale** jest klasą, która reprezentuje ustawienia regionalne, takie jak język i kraj. ***Locale.US*** to jedna z predefiniowanych stałych Locale w klasie Locale oznaczająca *Stany Zjednoczone*. Ustawienia regionalne (Locale) są używane w różnych kontekstach w Javie, takich jak formatowanie liczb, dat, walut itp. Tutaj Locale.US wprowadza kropkę do liczby zmiennoprzecinkowej zamiast znaku przecinka.

Różnica między *Equals()* a użyciem ==

- **Equals() porównuje zawartość**

czy zawartość stringu1 jest równa stringowi2

- **== porównuje czy adresy obiektu w pamięci(referencje) wskazują na ten sam obiekt**

chociaż we współczesnych wersjach javy == też by zadziałało, ponieważ od pewnego momentu w historii, gdy dwa obiekty przechowują ten sam string (*tyczy się to właśnie tylko stringów!*), drugi string nie jest tworzony tak samo jak pierwszy, tylko zwraca on referencję do pierwszego obiektu, który zawiera ten string

W javie występują tylko referencje (przeciwnie do języka C/C++ gdzie były jeszcze wskaźniki i obiekty same w sobie)

Continue, break, return

W języku Java, continue, break, i return są instrukcjami sterującymi, które wpływają na przebieg programu w różny sposób.

Break:

- Instrukcja break jest używana do przerwania pętli (np. for, while, do-while) lub instrukcji switch. Gdy interpreter natrafi na break, natychmiastowo **opuszcza daną pętlę lub instrukcję switch**.

Continue:

- Instrukcja continue jest używana do przeskakiwania aktualnej iteracji pętli i przechodzenia do następnej iteracji. W przeciwieństwie do break, continue **nie kończy całej pętli, ale tylko bieżącą iterację**.

Return:

- Instrukcja return jest używana do zakończenia wykonywania metody i zwrócenia wartości. Gdy interpreter napotyka return, opuszcza bieżącą metodę, a jeśli jest zdefiniowana wartość zwracana, przekazuje ją do miejsca, z którego została wywołana metoda.

Static vs Non-Static

Static

Są to pola i metody odnoszące sie do klasy jako całości. Dla każdego stworzonego obiektu będą takie same.

```
public class Cat{
    public static final MAX_LIVES = 9;
    private static catCount = 0;
    String name;
    int age;
    int livesRemaning;
}
```

- Nie potrzeba faktycznej instancji obiektu żeby dostać się do tych pól / użyć tych metod

```
System.out.println(Cat.catCount);
```

- Statyczne pola i metody są jak zamknięty podsystem w klasie. **Statyczne metody mogą korzystać tylko ze statycznych pól**
- metody statyczne można wywołać przez konkretne obiekty ale nie jest to zalecane - mozemy dostać warning

Podsumowując, static w Javie oznacza, że dany atrybut lub metoda jest wspólna dla wszystkich instancji klasy, a nie zależna od konkretnego obiektu tej klasy.

Non-Static

- jest wywolywana tylko na **konkretnym** obiekcie, nigdy na klasie

```
Cat myCat = new Cat();
myCat.meow();
myCat.name = "Puszek";
myCat.age = 5;
```

NIE MA SENSU:

```
Cat.meow();
```

Klasa kot nie może miaczeć!

- metody niestatyczne mogą korzystać ze statycznych zmiennych

Słowo Final

Dla Klas

- Żadna klasa nie może rozszerzać tej klasy.

Dla Metod

- Metoda ze słówkiem final nie może być nadpisana w klasie potomnej.

Nie możemy użyć *@Override*. Używamy więc wówczas, gdy chcemy aby funkcjonalność danej metody była taka sama dla każdej z podklas.

Dla Pól

- Zmienna final to zmienna, do której możesz przypisac wartość tylko raz

final odpowiednik ***const*** (z C++)

Nie można drugi raz stworzyć tego samego obiektu.
Kamelcase wygląda tak, że takie stałe piszemy w całości z dużych liter.

```
public static final MAX_LIVES = 9;
```

Skrócony zapis if else

```
roots[0] = (delta==0)?1:2;
```

Należy czytać w ponizszy sposób:

?— if

:— else

```
roots[0] = (delta==0) if 1 else 2;
```