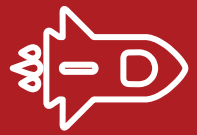
Object Oriented Programming

Witajcie!

Łukasz Włodarski
neonelect91@gmail.com

WAŻNE POJĘCIE

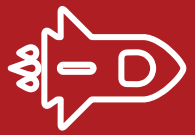


Krótkie wyjaśnienie …

1.
Programowanie obiektowe
Object oriented programming

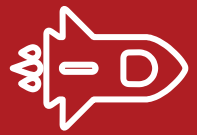
2.
Klasa
Class

Obiekt *(object)*



Jest odzwierciedleniem rzeczywistego bytu w programie.

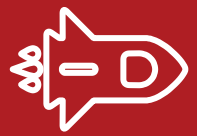
Pole *(field)*



Jest atrybutem (pojedynczą cechą) klasy.

Metoda

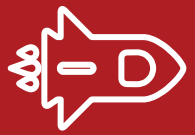
(method)



Zawiera jedno konkretne zachowanie klasy.

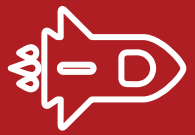
Konstruktor

(constructor)



Zawiera ”instrukcję” jak tworzyć obiekt.

Przeciążanie *(overloading)*



Ta sama nazwa, ale inne parametry

Warto wspomnieć o …

- Klasa anonimowa,
- Klasa abstrakcyjna

POJO

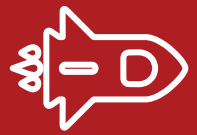
(Plain Old Java Object)

Klasa zawierająca pola i metody, ale nie dziedzicząca po żadnej klasie i nie implementująca żadnego interfejsu. Nie używająca żadnych adnotacji.

Inaczej prosty obiekt javy!

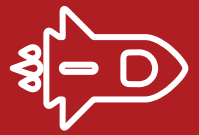
3.
Tworzenie obiektów
Object initializing

new



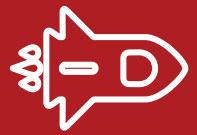
Operator rezerwuje pamięć dla nowo tworzonego obiektu.

Stan i zachowanie



Stan pól i zachowanie metod.

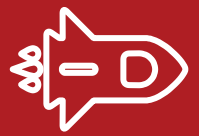
Porównywanie obiektów



equals porównuje stany wewnątrz dwóch obiektów
== porównuje referencje w pamięci

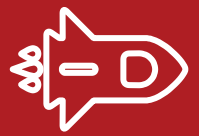
UWAGA: Nigdy nie porównujemy obiektów za pomocą == !

Hash code



Unikalny identyfikator generowany na podstawie pól danej klasy.

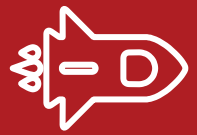
static



Zmienna - taki sam stan zmiennej w całym programie.

Metoda – przywiązana do klasy, nie obiektu. Można ją wywołać przez referencje do klasy

final



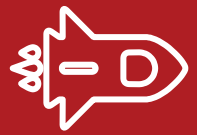
Zmienna – nie można zmienić jej wartości

Metoda – nie można jej nadpisać

Klasa – nie można po niej dziedziczyć

4.
Generyczność
(Generics)

Type

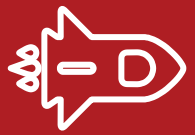


Narzucamy typ ogólny, by z każdym wykorzystaniem definiować konkretny.

Typów ogólnych możemy definiować wiele!

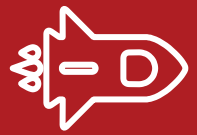
5.
Dziedziczenie
Inheritance

public



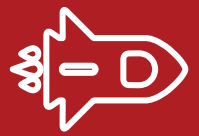
Klasa/metoda widoczna wszędzie w programie.

Package protected



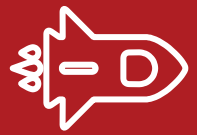
Klasa/metoda widoczna w zasięgu pakietu, którego się znajduje.

protected



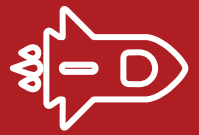
Metoda dostępna tylko w klasach dziedziczących.

private



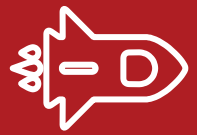
Klasa/metoda dostępna tylko w zasięgu klasy/pliku, w którym się znajduje.

enkapsulacja



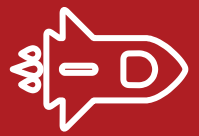
Prawidłowa enkapsulacja **gwarantuje nam, że jedynym obiektem odpowiedzialnym za zmianę stanu jest sam obiekt** i nie ma możliwości modyfikacji jego stanu z zewnątrz.

polimorfizm



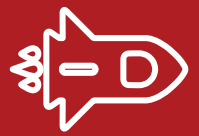
Wyodrębnienie warstwy abstrakcyjnej, która może być użyta na wiele sposobów.

extends



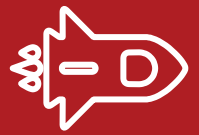
Jednoznacznie wskazuje klasę, po której dziedziczymy.
Java pozwala na dziedziczenie tylko **jednej klasy!**

Przesłanianie *(overriding)*



Nowe zachowanie metody w klasie dziedziczącej lub anonimowej.

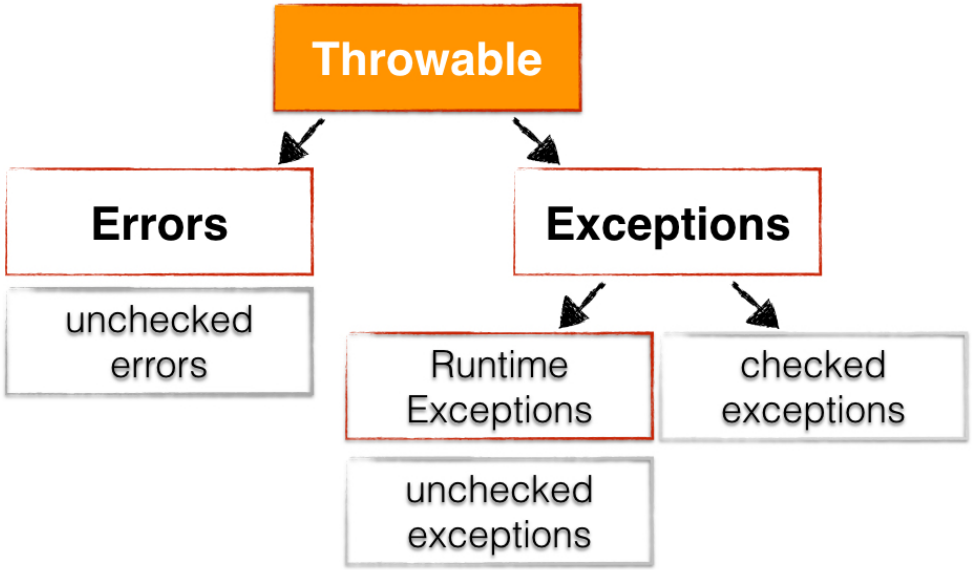
abstract



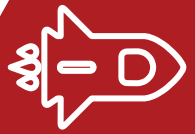
Klasa – nie możemy tworzyć instancji; musimy po niej dziedziczyć
Metoda – musimy zapewnić implementację w klasie dziedziczącej

6.
Wyjątki
Exceptions

Podział


Throwable
Errors
unchecked errors
Exceptions
Runtime Exceptions
unchecked exceptions
checked exceptions

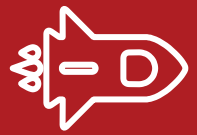
try/catch/finally



Klauzule do obsługiwania wyjątków.

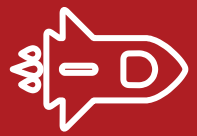
UWAGA: finally wykona się **zawsze**!

obsługiwane (checked)



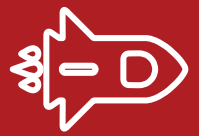
Wyjątki, które musimy obsłużyć za pomocą ***try/catch***.

nieobsługiwane (unchecked)



Wyjątki, które możemy ale nie musimy obsługiwać.

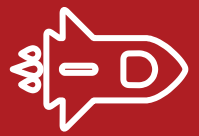
throw/throws



throw – rzuca wyjątek w metodzie,
throws – deklaruje w sygnaturze metody, który wyjątek ta metoda rzuca

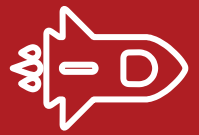
7.
Interfejsy
Interfaces

interface



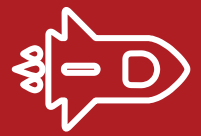
Można implementować wiele interfejsów!

Dziedziczenie, a interfejsy



Interfejsy też mogą dziedziczyć!

Interfejsy, a klasy abstrakcyjne



Klasa abstrakcyjna może implementować interfejs, ale **nie musi** zapewnić implementacji jego metodom.

8.
Garbage collector

Dzieki!!

Łukasz Włodarski