# Programowanie w języku Java - Hermetyzacja

mgr inż. Maciej Lasota <m.lasota@tu.kielce.pl>

Version 1.0, 03-03-2017

# Spis treści

# Hermetyzacja (enkapsulacja)

Java jest językiem obiektowym. Języki obiektowe posługują się pojęciem obiektu i klasy. **Obiekt** – to konkretny lub abstrakcyjny byt, wyróżnialny w modelowanej rzeczywistości, posiadający określone *granice* i *atrybuty (właściwości)* oraz mogący świadczyć określone usługi, czyli wykonywać określone działania lub przejawiać określone zachowanie. Obiekty współdziałają ze sobą wymieniając komunikaty. **Komunikat** - to wąski i dobrze określony interfejs, opisujący współzależność działania obiektów. Komunikaty zwykle żądają od obiektów wykonania określonych (właściwych im) usług.

**Hermetyzacja** nazywana też **enkapsulacja** jest jednym z podstawowych założeń *paradygmatu programowania obiektowego*. Polega ona na **ukrywaniu** lub **udostępnianiu** określonych *atrybutów (pól)* oraz *metod* obiektów danej klasy tak, aby były one dostępne tylko metodom wewnętrznym danej klasy lub metodom zaprzyjaźnionym wybranych obiektów.

W języku Java dostęp do składowych klasy regulują tzw. *specyfikatory widoczności (dostępu)*, których używamy w deklaracjach **zmiennych**, **stałych** i **metod**.

Główne założenia Hermetyzacji (enkapsulacji):

1. Ochrona przed **zepsuciem** (zazwyczaj pola powinny być prywatne, wyjątkiem są stałe). Użytkownik klasy **nie ma dostępu** do prywatnych pól i nic nie popsuje (nieświadomie).
2. Zapewnienie **właściwego interfejsu** (metody "robocze" winny być prywatne). Użytkownik klasy ma do dyspozycji tylko niezbędne (klarowne) metody, co ułatwia mu korzystanie z klasy.
3. Ochrona przed konsekwencjami **zmiany implementacji**. Twórca klasy może zmienić zestaw i implementację prywatnych metod, nie zmieniając interfejsu publicznego: wszystkie programy napisane przy wykorzystaniu tego interfejsu nie będą wymagały żadnych zmian.

# Modyfikatory widoczności (dostępu)

## Dla klas:

- **private** - klasa dostępna jest tylko wewnątrz pakietu (pliku); brak w definicji klasy jakiegokolwiek modyfikatora powoduje automatycznie uznanie klasy za private;
- **public** - udostępniana na zewnątrz pakietu *(ang. package)*, w którym się znajduje; w jednym pliku może wystąpić tylko *jedna* klasa typu public; aby użyć jej poza pakietem, w którym się znajduje, należy odwołać się: `NazwaKlasy.java`;

## Dla metod i pól danych:

- **public** - pole/metoda dostępna jest dla wszystkich klas;
- **private** - dostęp do pól/metod posiadają jedynie inne metody tej samej klasy;
- **protected** - powoduje, że pole/metoda będzie widoczne (dostępne) w klasie, wszystkich

podklasach (klasach dziedziczących z klasy zawierających pole/metodę) i w całym pakiecie;

- **package** *(zaprzyjaźniona)* - modyfikator domyślny, wszystkie pola/metody bez modyfikatora dostępu traktowane są jako typu package; pola/metody mogą być używane przez inne klasy danego pakietu

# Modyfikatory właściwości

## Dla klas:

- **abstract** - definiuje klasę abstrakcyjną zawierającą chociaż *jedna* metodę abstrakcyjną (niezaimplementowaną); nie można tworzyć żadnych obiektów tej klasy; klasa taka jest użyteczna podczas budowania hierarchii klas (dziedziczenie): stanowi wzór do tworzenia klas pochodnych;
- **final** - uniemożliwia tworzenie klas pochodnych; stosuje się w celu uniemożliwienia klasa potomna "podszycia się" pod klasę bazową oraz czasami do oznaczenia klas, nad którymi prace zostały ukończone;
- **synchronizable** - klasa z mechanizmem obsługi wątków (ang. threads);

## Dla metod:

- **abstract** - metody niezaimplementowane (bez kodu i zmiennych); użyteczna podczas budowania hierarchii klas (dziedziczenie): stanowi wzór dla metod klas pochodnych, które wymagają jej różnych implementacji;
- **static** - metoda wspólna dla wszystkich obiektów danej klasy; należy ona do klasy, a nie do obiektu i może być wywoływana bez tworzenia obiektu danej klasy;
- **final** - uniemożliwia klasom pochodnym "przesłanianie" metody;
- **synchronized** - metoda blokująca dostęp do obiektu, do którego należy i odblokowująca go, gdy zakończy działanie; jeżeli dostęp do obiektu został wcześniej zablokowany, to metoda oczekuje na jego odblokowanie zanim zacznie się wykonywać; mechanizm bardzo istotny w przypadku programów wielowątkowych (ang. multitheads);
- **native** - oznacza funkcję z wykorzystaniem nieprzenośnych cech danej platformy: programy napisane w Javie są niezależne od platformy sprzętowej i systemowej; jeżeli konieczne jest skorzystanie z mechanizmów specyficznych dla danej platformy, to modyfikator native powoduje, że program będzie mógł być uruchamiany tylko na tej *jednej* dedykowanej platformie;

## Dla pól danych:

- **final** - wartość pola jest ustalana podczas tworzenia obiektu i nie może być później modyfikowana: pole danych jest stałą;
- **static** - pole danych wspólne dla wszystkich obiektów danej klasy; należy do klasy, a nie do obiektu i zmiana wartości w jednym obiekcie, powoduje, że zmienia się jego wartość dla wszystkich obiektów;
- **transient** - pole danych nie jest trwałą częścią obiektu i nie będzie zachowane przy archiwizacji

obiektu;

- **volatile** - oznacza pole, które może być modyfikowane asynchronicznie, przez konkurencyjne wątki w programach wielowątkowych.

# Pakiety

Program w języku Java to w pewnym uproszczeniu – **zbiór klas**. Nie jest to jednak zbiór bezładny. Klasy pogrupowane są w tzw. *pakiety*. Klasy dzielimy na pakiety by pogrupować je według ich znaczenia – analogicznie do tego, jak dzielimy pliki na *katalogi* na dysku twardym naszego komputera.

**Pakiety** podobnie jak katalogi – mają *strukturę hierarchiczną*, tj. każdy z pakietów może zawierać kolejne pakiety, podobnie jak katalogi mogą zawierać podkatalogi. Każdy pakiet, oprócz dowolnej liczby innych pakietów może zawierać także *dowolną liczbę klas* – podobnie do katalogu, który oprócz innych katalogów może zawierać dowolnie wiele plików.

Nazwa pakietu odpowiada strukturze katalogów.

```
package odwrocona.nazwa.domeny.pakiet;
```

```
odwrocona
\--nazwa
   \--domeny
      \--pakiet
         |--Klasa.java
         |  ...
         \--Test.java
```

## Korzystanie z pakietu:

Importowanie wszystkich klas z pakietu

```
import nazwa.pakietu.*;
```

Importowanie określonej klasy

```
import nazwa.pakietu.Klasa;
```

Dostęp do klasy uprzednio zaimportowanej

```
Klasa
```

Dostęp do klasy niezaimportowanej (przez nazwę pakietu)

```
nazwa.pakietu.Klasa
```

# Bibliografia


- **Bruce Eckels**, *"Thinking in Java. Edycja polska. Wydanie IV"*, wydawnictwo Helion.
- **Cay S. Horstmann, Gary Cornell**, *"Java. Podstawy. Wydanie IX"*, wydawnictwo Helion.
- **Cay S. Horstmann, Gary Cornell**, *"Java. Techniki zaawansowane. Wydanie IX"*, wydawnictwo Helion.
- **Krzysztof Barteczko**, *"Podstawy programowania w języku Java, PJWSTK"*, http://edu.pjwstk.edu.pl/wyklady/ppj/scb/
- **Konrad Kurczyna**, *"Laboratorium Java"*, Politechnika Świętokrzyska w Kielcach.
- **Mariusz Lipiński**, *"Nauka Javy"*, http://www.naukajavy.pl/