Refleksje (odbicia)

Co to jest? Jest to mechanizm, który pozwala nam

1) na dostęp do metod i pól obiektów, do których posiadamy referencje,
2) na dostęp do metod i pól statycznych klas, do których nie posiadamy referencji,
3) na używanie obiektów, których definicji nie znamy w momencie pisania naszego kodu.

Najlepszym przykładem jest załadowanie sterownika jdbc. Ładujemy tutaj klasę ze ścieżki ClassLoadera, której typu nie znamy w momencie tworzenia kodu.

```
Class.forName("com.mysql.jdbc.Driver");
```

Metoda **forName()** ładuje klasę (której typu nie znamy w momencie tworzenia kodu), która jest w **ClassLoaderze**. ClassLoader to zbiór wszystkich bibliotek, komponentów i klas, które są potrzebne do skompilowania naszej aplikacji.

Ciekawostka: Jedyny dostęp do JDBC jest przez refleksje. Nigdzie nie jest zapisywana referencja do obiektu klasy. Nie robimy tego, bo sterownik JDBC sam rejestruje się w mechanizmach JDBC.

Wypisanie metod klasy

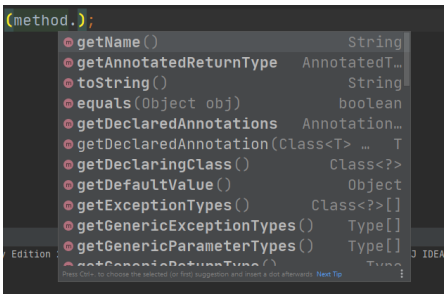
https://github.com/idzikpro/JavaBasics/blob/master/src/main/java/pl/idzikpro/reflection/Person.java

https://github.com/idzikpro/JavaBasics/blob/master/src/main/java/pl/idzikpro/reflection/GetDeclaredMethodsFromPerson.java

```
for (Method method:Person.class.getDeclaredMethods()
    ) {
   System.out.println(method);
}
```

Zobaczymy wszystkie metody, nawet te prywatne i odziedziczone.

Jakie informacje o metodzie można jeszcze zobaczyć?



Gdzie są stosowane refleksje?

1) Przyciemnianie kodu (biblioteka do mavena) – w kodzie skompilowanym nazwy metod i klas są haszowane
2) Gdy nie jesteśmy w st+anie utworzyć obiektu danej klasy.
3) IntelliJIdea wykorzystuje refleksje, aby podpowiadać składnię.

Uruchamiane metod klasy za pomocą refleksji

https://github.com/idzikpro/JavaBasics/blob/master/src/main/java/pl/idzikpro/reflection/InvokeMethodFromPerson.java

```
Object myObject = Person.class.getConstructor(String.class,Integer.class).newInstance("Tomasz",20);
Method myMethod = myObject.getClass().getMethod("toString");
Object myReturn = myMethod.invoke(myObject);
```

W przykładzie została wywołana metoda **toString** po uprzednim utworzeniu obiektu za pomocą konstruktora dwuparametrowego. Jeśli zachodzi potrzeba wywołania metody parametrowej, to w **invoke** po obiekcie jako kolejne parametry należy podawać argumenty wywołania tej metody.

Przykład pokazujący wszystkie metody klasy String

https://github.com/idzikpro/JavaBasics/blob/master/src/main/java/pl/idzikpro/reflection/GetStringMethods.java

```
Class myString=Class.forName("java.lang.String");
for (Method method:myString.getDeclaredMethods()
    ) {
   System.out.println(method.getName());
}
```

Można też zrobić tak

```
Class myString="String".getClass();
```

albo tak

```
Class myString=String.class;
```

W przypadku metod, czy pól prywatnych, domyślnych z innym pakiecie, musimy przed wywołaniem metody czy dostępem do pola zmienić jego przy pomocy metody **setAccessible(boolean)**