Programowanie obiektowe

Laboratorium 2. Założenia programowania obiektowego.

Podejście obiektowe

W języku Java wszystko traktujemy jako obiekt, niezależnie od stopnia złożoności danych czy metod na nich wywoływanych. Identyfikator obiektu który modyfikujemy jest referencją do obiektu. Przykładowo:

```
String tekst; // tworzymy referencję typu String

String tekst = "jakiś tekst"; // tworzymy i inicjalizujemy referencję typu String,
```

Powyższy zapis jest jednak uproszczony dla typu String. Pełen mechanizm tworzenia obiektu danej klasy wygląda tak (posługujemy się operatorem **new**):

```
String tekst;
tekst  = new String ("jakiś tekst)";
```

Typy podstawowe

Istnieje grupa typów, które nie wymagają użycia operatora **new**, a co za tym idzie, nie są tworzone obiekty, a jedynie zmienne automatyczne, nie będące referencją – jest to uzasadnione względami pamięciowymi. Typy te wymienione są poniżej, wraz z odpowiadającymi im nazwami klas, o ile istnieje potrzeba stworzenia obiektu danej klasy (z reguły można obejść się bez tego):

```
boolean    Boolean
char       Character
byte       Byte
short      Short
int        Integer
long       Long
float      Float
double     Double
void       Void
```

Przykładowo, stworzyć możemy obiekt typu podstawowego char:

```
char c = 'x';
```

lub też stworzyć obiekt klasy Character:

```
char c = new Character ('x');
```

Idea programowania obiektowego polega na umożliwieniu programowi dostosowania się do specyficznego języka danego problemu poprzez dodanie nowych typów obiektów. Każdy obiekt posiada swój wewnętrzny stan (dane) oraz zestaw operacji, które może wykonać (metody).

Podejście obiektowe według Alana Kaya (twórcy języka Smalltalk):

1. *Wszystko jest obiektem*. Można wyobrazić sobie obiekt jako wymyślną zmienną – taką, która nie tylko przechowuje dane, ale może także realizować żądania, czyli wykonywać na sobie pewne operacje. Teoretycznie, dowolne pojęcia ze świata rozwiązywanego problemu (psy, budynki, usługi itd.) można reprezentować w programie za pomocą obiektu.

2. *Program jest zbiorem obiektów, które poprzez wysyłanie komunikatów mówią sobie nawzajem, co robić*. „Wysłanie komunikatu" do obiektu, to inaczej zażądanie wykonania przez niego pewnej operacji. Można też myśleć o komunikacie jako o wywołaniu funkcji przynależnej do konkretnego obiektu.

3. *Każdy obiekt posiada swą własną pamięć, na którą składają się inne obiekty*. Innymi słowy, nowy obiekt tworzymy, łącząc w jeden pakiet grupę już istniejących obiektów. Budujemy w ten sposób złożone programy, ukrywając jednocześnie ich złożoność za prostotą obiektów.

4. *Każdy obiekt posiada swój typ*. Mówiąc potocznie, każdy obiekt jest *instancją* pewnej *klasy*, gdzie „klasa" jest synonimem „typu". Najistotniejszą cechą wyróżniającą klasę jest odpowiedź na pytanie: „Jakie komunikaty można do niej wysłać?".

5. *Wszystkie obiekty danego typu mogą otrzymywać te same komunikaty*. Ponieważ obiekt typu „okrąg" jest jednocześnie obiektem typu „figura", „okrąg" może otrzymywać komunikaty odpowiednie dla „figury". Oznacza to, iż możliwe jest pisanie kodu odwołującego się do „figury", który będzie automatycznie obsługiwał wszystkie obiekty pasujące do opisu typu „figura". Ta *zastępowalność* jest jedną z najpotężniejszych idei programowania obiektowego.

Abstrakcyjne typy danych (klasy) zachowują się prawie tak samo jak typy podstawowe wymienione wcześniej: można tworzyć zmienne takiego typu (w OOP nazywane obiektami lub instancjami danej klasy), a następnie manipulować nimi (proces ten nazywamy wysyłaniem komunikatów lub *żądań* do obiektów).

Składowe danej klasy posiadają pewne cechy wspólne (np. cechą wszystkich instancji klasy „kwadrat" w naszym programie będzie posiadane pole oraz obwód), równocześnie każdy obiekt danej klasy może posiadać inny stan (każdy „kwadrat" będzie mógł mieć inną wartość pola i obwodu).

Możliwość tworzenia nowych klas przez programistę, zamiast konieczności bazowania wciąż na ograniczonej liczbie wbudowanych typów, pozwala na dostosowanie programu do rzeczywistych potrzeb problemu.

Każdy obiekt posiada interfejs.

Interfejs definiuje listę żądań, jakie można skierować pod adresem obiektu danego typu. Kod realizujący te żądania, wraz z ukrytymi danymi, składa się na *implementację*. Przykładowo interfejs typu wyglądałby tak:

Nazwa typu (klasa):	`Zarowka`
Interfejs:	`zapal()`
	`zgas()`
	`rozjaśnij()`
	`przyciemnij()`

Dla powyższego przykładu napisać można kod:

```
Zarowka zr = new Zarowka();
zr.zapal();
```

Klasą (typem) jest tutaj `Zarowka`, nazwą konkretnego obiektu jest `zr`, a interfejs definiuje cztery żądania, jakie można skierować do obiektów typu `Zarowka`. Stworzenie obiektu typu `Zarowka` polega na zdefiniowaniu referencji `zr` dla tego obiektu, a następnie wywołaniu operatora new, realizującego żądanie stworzenia nowego obiektu danego typu. W celu wysłania do obiektu komunikatu, piszemy jego nazwę (danego obiektu) i łączymy ją za pomocą kropki z nazwą komunikatu.