Opis programowania agantowego

**Programowanie obiektowe kontra programowanie agentowe**

Programowanie obiektowego (ang. object-oriented programming, OOP) jest obecnie najpopularniejszym i najważniejszym paradygmatem stosowanym do tworzenia oprogramowania. Stanowi on również podstawę dla innych paradygmatów takich jak programowanie aspektowe (ang. aspect-oriented progrming, AOP) oraz programowanie generyczne (ang. generic programming). [1] Programowanie obiektowe może przybrać również bardziej wyspecjalizowaną formę nazywaną programowaniem agentowym (ang. agent-oriented progrming, AOP). [2] Poniższe akapity(podrozdziały?) przedstawią oba paradygmaty oraz ich porównanie.

**Definicja i opis programowania Obiektowego**

Programowanie obiektowe jest paradygmatem, w którym do tworzenia programów wykorzystuje się elementy zwane obiektami. Obiekty stanowią abstrakcyjne byty programistyczne, które stanowią przestrzeń zagadnień rozwiązywanych przez program. Ten fakt nadaje programowi cechę dostosowywania się do języka danego zagadnienia. Oznacza to rozpatrywanie zagadnień we właściwych, bliższych myśleniu człowieka, kategoriach, w odróżnieniu od kategorii maszyny, na której uruchomiony jest program. Przykładem może być prosta układanka składająca się z pewnej liczby elementów przesuwanych po planszy przez użytkownika. Elementy układanki, jak i sama plansza mogą stanowić obiekty, które programista jest w stanie wykorzystać do napisania programu. Każdy obiekt może posiadać pewne dane wewnętrzne odpowiadające za jego stan, zestaw funkcji lub metod będącymi jego zachowaniem oraz możliwość odróżnienia danego obiektu od innego, zwana tożsamością. Informacje na ten temat przechowywane są w klasie. Zmienne należące do klasy nazywane są polami klasy, a obiekt utworzony na podstawie klasy jej instancją. Przykład kodu klasy dla języka Java został zawarty w listingu 1.

**public** **class** Account {

**private** **double** balance;

**public** **double** getBalance() {

**return** balance;

}

**public** **void** deposit(**double** depositValue)

{

balance += depositValue;

}

**public** **boolean** withdraw(**double** withdrawValue)

{

**if**(withdrawValue <= balance){

balance -= withdrawValue;

**return** **true**;

}

**return** **false**;

}

}

Listing 1. Klasa stanowiąca implementacje konta bankowego.

Przykładowa klasa zaprezentowana na listingu 1 posiada zmienną typu zmiennoprzecinkowego, która stanowi wartość depozytu bankowego. Każda instancja tej klasy posiadać będzie własną wartość tego pola. Zachowanie obiektów klasy Account zdefiniowane jest w funkcji withdraw i metodzie deposit służących do zmiany wartości konta oraz w metodzie getBalance zwracającej aktualny stan konta. Wspomniane metody i funkcje stanowią interfejs klasy Account, ponieważ określają jakie akcje lub żądania mogą zostać wykonane względem obiektów tej klasy. Z tego wynika, że zachowanie i rola obiektów w programie zależy od klasy którą dany obiekt reprezentuje. Obiekty tworzące program mogą komunikować się ze sobą w celu wykonywania zaprogramowanych dla nich zadań. Sposób w jaki to robią polega na wzajemnym wywoływaniu przez nie operacji . Najczęściej wykorzystywaną techniką umożliwiająca komunikację między obiektami jest posiadanie jednego obiektu przez inny. Powyższe twierdzenie może zilustrować może dowolna klasa posiadająca na przykład pole typ Account. Obiekty takiej klasy mogłyby komunikować się z posiadanymi przez nie obiektami poprzez wartości argumentów, wywoływanych przez nie, metod i funkcji .

Obecnie OOP dzieli się na dwa podtypy które odróżniają się odmiennym sposobem tworzenia obiektów. Jest to programowanie oparte o wspomniane wyżej klasy. Ten podtyp jest wykorzystywany w językach takich jak Java, C++, C# i wiele innych. [1] Drugim podtypem jest programowanie oparte o prototyp(ang. Prototype-based programming) gdzie nowy obiekt tworzony jest w oparciu o już istniejący, wykorzystywanym w głównej mierze w językach skryptowych, takich jak Perl, Java Script czy LUA. [3]

Najważniejsze założenia paradygmatu obiektowego to:

• Abstrakcja – polega na pomijaniu pewnych szczegółów w celu uproszczenia sposobu rozwiązywania problemów. Przykładem może być obiekt reprezentujący samochód, który w zależności od systemu może zawierać cechy takie jak barwa, numer rejestracyjny i dane właściciela lub szczegółowe wymiary, numer silnika i dane fabryki która go wyprodukowała. Oba przypadki odrzucają pewne cechy i skupiają się tylko na tych które program potrzebuje. [2]

• Hermetyzacja – pomaga chronić obiekty przed niepowołaną zmianą stanu lub użyciem funkcji poprzez wprowadzenie kontroli dostępu do wszystkich jej składników. To wymusza korzystanie z klasy tylko w założony przez jej twórcę sposób. Eliminuje to konieczność zabezpieczania programu przed skutkami ubocznymi użycia klasy w błędny sposób oraz pozwala na zmiany wewnętrznych, chronionych mechanizmów bez utraty kompatybilności z resztą programu. Z punktu widzenia użytkownika klasy, hermetyzacja pozwala na odróżnienie elementów dla niego istotnych, do których ma dostęp, od tych które powinien zignorować. W obiektowych językach programowania, takich jak Java lub C++ używa się najczęściej trzech modyfikatorów dostępu. Public oznacza, że dana funkcja lub pole jest widoczne poza klasą, private – tylko w obrębie danej klasy albo protected, kiedy funkcja, czy pole są chronione jedynie przed dostępem z zewnątrz.[1]

• Dziedziczenie – jest mechanizmem, który pozwala rozszerzyć lub zmienić zachowanie klasy tworząc na jej podstawie nową klasę lub klasy. Klasa, po której następuje dziedziczenie jest nazywana klasą bazową, a klasa dziedzicząca klasą pochodną. Dziedziczenie, nie tylko pozwala na rozszerzanie klas, ale tworzy też pewną hierarchę. Przykładem może być klasa „zwierzę” dla której klasami pochodnymi są klasy „słoń” , „lew” i „zebra”. Klasy pochodne (te które nie są chronione przed dziedziczeniem) posiadają cechy klasy „zwierzę” oraz dodają do nich swoje bardziej wyspecjalizowane cechy lub zachowania. Ukazaną hierarchię można określić jako przejście od jednego ogólnego typu do wielu wyspecjalizowanych typów. W zależności od języka programowania klasa pochodna może dziedziczyć tylko z jednej lub z wielu klas bazowych. [1]

• Polimorfizm – jest związany bezpośrednio z dziedziczeniem. Pozwala on na używanie instancji klas pochodnych, tak jakby były typu bazowego, jednak obiekt wciąż zachowuje swoje oryginalne zachowanie. Oznacza to że obecny typ obiektu ma wpływ jedynie na to jakie funkcje możemy użyć, a nie na to z jakiej klasy zostaną wywołane. Jest to możliwe dzięki mechanizmowi zwanemu późnym wiązaniem (ang. late binding) pozwalającym na wykonanie odpowiedniego kodu dopiero w czasie działania programu. Polimorfizm umożliwia bardzo elastyczne rozszerzanie programu o nowe typy pochodne, bez konieczności zmiany implementacji funkcji, które z nich korzystają. [1]

**[1]-** [**http://www.americanscientist.org/issues/pub/the-post-oop-paradigm**](http://www.americanscientist.org/issues/pub/the-post-oop-paradigm)

**[2] – Agent-oriented progrming –Yoav Shoham**

**[3] -http://en.wikipedia.org/wiki/Prototype-based\_programming**

**http://informatyka.umcs.lublin.pl/files/bylina.pdf**