

Wstęp

Pisanie kodu programu na tzw. *'wysokim poziomie abstrakcji'* jest podstawową umiejętnością programisty. 'Wysoki poziom abstrakcji' - mowa tu o tym, aby pisać tak kod programów, aby były jak najogólniejsze. Dzięki temu wytwarza się oprogramowanie, które można elastycznie dostosowywać do różnych projektów i łatwo dodawać nową funkcjonalność. Takie programowanie zbliżone jest do tworzenia 'uogólnionych struktur programistycznych' (ang. frameworks), które można 'wypełniać' funkcjonalnością dostosowaną do bieżących zadań. Wybrane struktury programistyczne, stanowią tzw. wzorce projektowe. Współczesne projektowanie oprogramowania, zakładające prace w grupie, zakłada wykorzystanie różnych typów wzorców projektowych.

Projektując swoje klasy, powinieneś uwzględnić jak najwięcej uogólnień, ponieważ wzmocni to 'elastyczność' twojego projektu – czyli możliwość jego rozbudowy.

Pamiętaj, obecnie nie piszemy 'zamknięte' projekty – zakładamy stałą możliwość rozbudowy oprogramowania.

W poprawnie zbudowanej hierarchii dziedziczenia klasami konkretnymi, czyli tymi, których instancje faktycznie wykorzystujesz, powinny być liśćmi drzewa dziedziczenia. Im bliżej korzenia, tym większe uogólnienie.

Polimorfizm – czyli wielość form.

To jest element kluczowy w programowaniu zorientowanym obiektowo. Cała idea polimorfizmu polega na tworzeniu generalizacji (dziedziczenia – odpowiedzi na relacje 'JEST' pomiędzy obiektami) – dzięki temu, jeśli np. zbudowałeś hierarchię dziedziczenia: Zwierze (klasa abstrakcyjna) – Pies i Kot (klasy konkretne, rozszerzające klasę Zwierze), to pracując na kolekcjach danych typu Zwierze, możesz do nich wczytywać instancje klas Pies i Kot – a więc traktujesz psa ogólnie jako zwierze lub jako psa w razie potrzeby. Tworząc metody w klasach nadrzędnych (bliżej korzenia dziedziczenia) oraz przesłaniając je w klasach niżej położonych w hierarchii dziedziczenia, umożliwiasz uruchomienie mechanizmu polimorfizmu – czyli pracujemy na obiektach typu zwierze,

„ZPR PWr – Zintegrowany Program Rozwoju Politechniki Wrocławskiej”

ale uruchamiając metody klasy zwierze, jeśli są one przesłonięte w klasach pies i kot, kompilator orientuje się, jaka dokładnie instancja chce uruchomić określoną metodę i uruchamia tę właściwą wersję. Generuje to bardzo dużo możliwości oraz umożliwia pisanie kodu na ‘wysokim poziomie abstrakcji’ – mechanizm polimorfizmu już potrafi skonkretyzować w razie potrzeby.

Ćwiczenie pierwsze wykonaj, rozpoznając jak najwięcej klas oraz tworząc jak najwięcej generalizacji – jest to kluczowy element ćwiczenia.

Ćwiczenie drugie zaprojektuj na wzór działań z ćwiczenia pierwszego, a następnie zaprezentuj kod swojego rozwiązania.

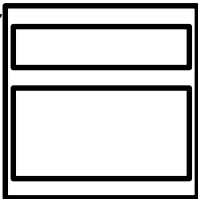
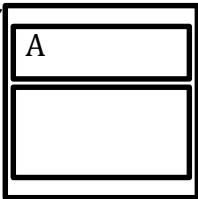

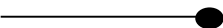
Lista 5

1. (zadanie projektowe)

Przygotuj szczegółowo rozbudowaną hierarchię dziedziczenia, prezentującą możliwość zarządzania klientami Banku – tzn. rozpoznaj różnych klientów typowego banku (studentów, klientów zatrudnionych na umowę o pracę lub innego typu, przedsiębiorstwa itd.). Opisz każdą wykorzystaną klasę w tabeli (poniżej szablon tabeli).

'Nazwa klasy'		
Składowe klasy:	<i>Nazwa składowej</i>	<i>Opis składowej</i>
	...	
Metody klasy:	<i>Nazwa metody</i>	<i>Opis metody</i>
	...	

Dodatkowo przygotuj schemat prezentujący zależności między klasami, wykorzystując poniższe elementy graficzne:

Klasa	<div> <i>'Nazwa klasy'</i>  </div>
Klasa abstrakcyjna	<div> <i>'Nazwa klasy'</i>  </div>
Dziedziczenie	
Zawieranie obiektów ¹	

¹ Uwaga: na tym etapie prac, nie zakładamy rozróżnienia agregacji i kompozycji obiektów (na tzw. diagramach UML oznaczane są one rombem pustym lub pełnym).

