Lista 9

Programowanie obiektowe: związek IS-A i jego konsekwencje – dziedziczenie i polimorfizm

Zadania z niniejszej listy mogą być rozwiązane w dowolnym języku programowania, który wspiera paradygmat obiektowy (np. OCaml, Scala, Java, C++, C#, ...) – wybór należy do rozwiązującego. Wymagane jest jednak, by paradygmat obiektowy w ramach wybranego języka wspierał zarówno dziedziczenie, jak również polimorfizm związany z podtypowaniem (patrz dalej).

Celem niniejszej listy jest zaznajomienie ze związkiem IS-A. Związek ten określany jest również mianem *generalizacja-specjalizacja* lub *podtypowania* (ang. subtyping) i dotyczy dwóch klas, w ramach których jedna pełni rolę klasy nadrzędnej (nadklasy), a druga podrzędnej (podklasy).
Związek IS-A posiada dwie istotne konsekwencje, które istnieją w większości popularnych języków programowania obiektowego:

- dziedziczenie,
- polimorfizm związany z podtypowaniem (ang. *subtype polymorphism*).

Mechanizm dziedziczenia pozwala na konstruowanie struktur, w ramach których atrybuty oraz operacje definiowane są w ramach nad klasy, a ze względu na związek dziedziczenia, są również one zdefiniowane w podklasie.
Związek IS-A pozwala również na pewnego rodzaju polimorfizm. Języki programowania dostarczają mechanizmów takich, jak metody wirtualne, które pozwalają definiować zachowanie w ramach nadklas, które to może być przedefiniowywane w ramach podklas.

Zadanie 1 (max. 4 pkt)

Przyjrzyj się poniższemu diagramowi. Prostokąty reprezentują klasy, a strzałki z niewypełnionym grotem związek generalizacja-specjalizacja. Strzałki z wypełnionym rombem reprezentują związek kompozycji. Nazwy klas pisane kursywą reprezentują klasy abstrakcyjne[1].

[1] Klasy, które nie mogą posiadać bezpośrednich instancji.

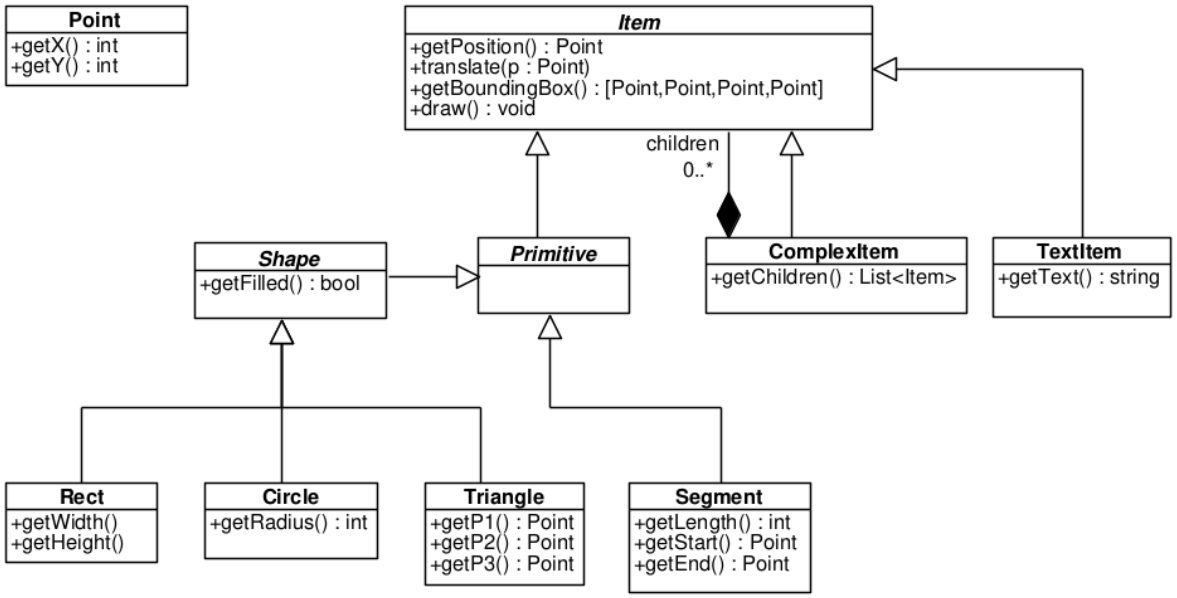


Diagram przedstawia hierarchię klas uproszczonego systemu grafiki wektorowej. Na diagramie przedstawiono kilka metod dostępowych (tzw. "*gettery*"), które charakteryzują dane możliwe do pozyskania z obiektów przedstawionych klas. Krótki opis klas przedstawiono w tabeli poniżej.

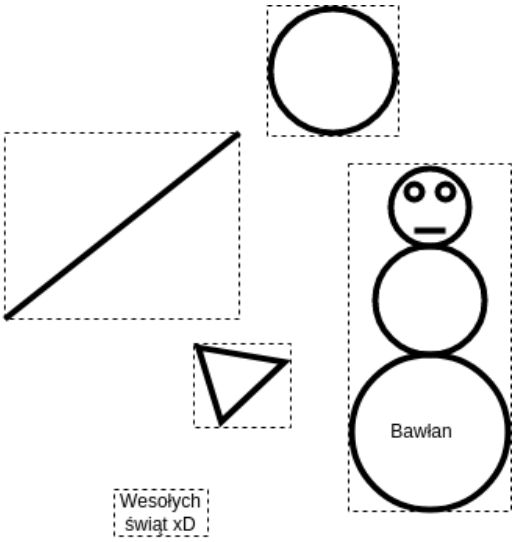
Klasa	**Opis**
Point	Klasa reprezentująca punkt w dwuwymiarowej przestrzeni. Punkt jest opisany dwoma współrzędnymi: X i Y, które są liczbami całkowitymi.
Item	Klasa abstrakcyjna reprezentująca pewien element będący składnikiem pewnej sceny. Element posiada swoją pozycję Position określoną przez punkt. Przyjęto, iż pozycja elementu określa współrzędne lewego, górnego rogu prostokąta otaczającego element (BoundingBox).
Primitive	Klasa abstrakcyjna określająca elementy pierwotne, niepodzielne.
Shape	Klasa stanowiąca abstrakcję nad konkretnymi kształtami, takimi jak okrąg, prostokąt, trójkąt, itd. Kształt domyślnie stanowi sam obrys, lecz może również posiadać wypełnienie w środku (Filled).
Rect, Circle, Triangle	Klasy specjalizujące Shape i reprezentujące konkretne figury. Klasy posiadają właściwości i metody charakterystyczne dla reprezentacji istotnych danych pozwalających zdefiniować

	daną figurę [np. wysokość (Height) i szerokość (Width) w przypadku Rect].
ComplexItem	Klasa reprezentująca element złożony, tj. składający się z listy elementów pierwotnych (Children). Wszelkie operacje wykonywane na tym elemencie powinny być przekazywane kolejno do jego wszystkich składowych. W szczególności, pozycja elementów składowych powinna być obliczana względem posiadającego je elementu złożonego.
TextItem	Klasa reprezentująca element tekstowy (napis).
Segment	Klasa reprezentująca odcinek.

W wybranym języku programowania zaimplementuj klasy widoczne na diagramie oraz związki pomiędzy nimi. Zadbaj o to, aby struktura klas definiowała wymagane pola oraz aby mechanizm dziedziczenia przenosił je na wszystkie potrzebne poziomy w hierarchii klas. Zapewnij możliwość instancjonowania klas i ustalania wartości atrybutów, np. przez konstruktory.

Następnie zapewnij możliwość realizacji zachowania poprzez implementację metod:

- metoda getBoundingBox() – każda z nieabstrakcyjnych klas: Rect, Circle, Triangle, Segment, ComplexItem, TextItem powinna dostarczać implementację obliczającą ograniczający prostokąt na bazie struktury obiektu, jak na rys. poniżej.



- metoda translate(p : Point), która pozwoli zmodyfikować pozycję elementu poprzez przesunięcie go o zadany punkt.

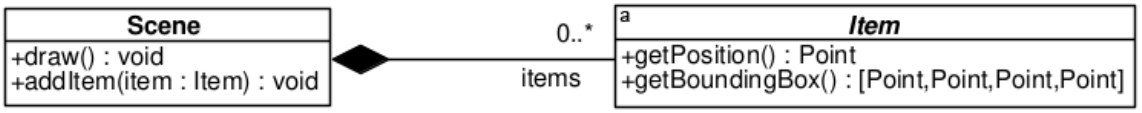
Zadanie 2 (max 2 pkt)

Zaproponuj i zaimplementuj własną specjalizację klasy Shape reprezentującą inny, nietrywialny obiekt wzbogacający opracowany system grafiki wektorowej. Dla przykładu: krzywa Béziera, ścieżka, wielokąt gwiaździsty, spirala.

Zadanie 3 (max. 4 pkt)

Skonstruuj mechanizm renderowania obiektów dla opracowanej hierarchii klas. W tym celu:

- utwórz klasę Scene reprezentującą scenę. Umieść w niej metody do dodawania elementów oraz rysowania sceny.



- rozszerz klasę *Item* o metodę (np. abstrakcyjną/czysto wirtualną) draw(...). W każdej z konkretnych klas zapewnij jej implementację, która pozwoli go narysować.
- w wybrany sposób zobrazuj działanie metod wyświetlających scenę, dodanie obiektów na scenę, przesunięcie obiektów na scenie.

Mechanizm renderowania może być oparty o tryb tekstowy (konsolę) lub wykonany z wykorzystaniem dowolnej, wybranej biblioteki do renderowania grafiki (np. SDL, OpenCV, ...).