6 podchwytliwych pytań rekrutacyjnych z języka UML

Szymon Zioło
e-mail: sziolo@redpill.com.pl
https://www.redpill.com.pl
Kursy wideo on-line: https://kursy.redpill.com.pl

LinkedIn YouTube

redpill
szkolenia it konsulting



0. Wprowadzenie

Jesteś analitykiem lub projektantem IT. Piszesz w CV, że znasz UML. I rzeczywiście znasz. Albo przynajmniej wydaje ci się że znasz. Myślisz o nowej pracy. Idziesz na rozmowę rekrutacyjną. Jeśli rekruter jest osobą merytoryczną, a nie „panią z HR”, to może próbować sprawdzić Twoją znajomość języka UML. I czasami zadaje pytania z kategorii „podchwytliwych”, to znaczy takich, na które odpowiedzi możesz nie znać lub nie pamiętać, nawet jeśli nieźle sobie radzisz z UML-em.

Podana tutaj lista podchwytliwych pytań jest oczywiście subiektywna. Zawiera pytania, które sam miewałem na rozmowach kwalifikacyjnych, gdy szukałem pracy. Gdy sam zacząłem „maglować” kandydatów do pracy, też lubiłem ich zaginać tymi pytaniami.

Mam nadzieję, że ten dokument pozwoli Ci lepiej poznać niuanse języka UML i lepiej przygotować się do następnej rozmowy kwalifikacyjnej, na której… nie dasz się zagiąć ☺

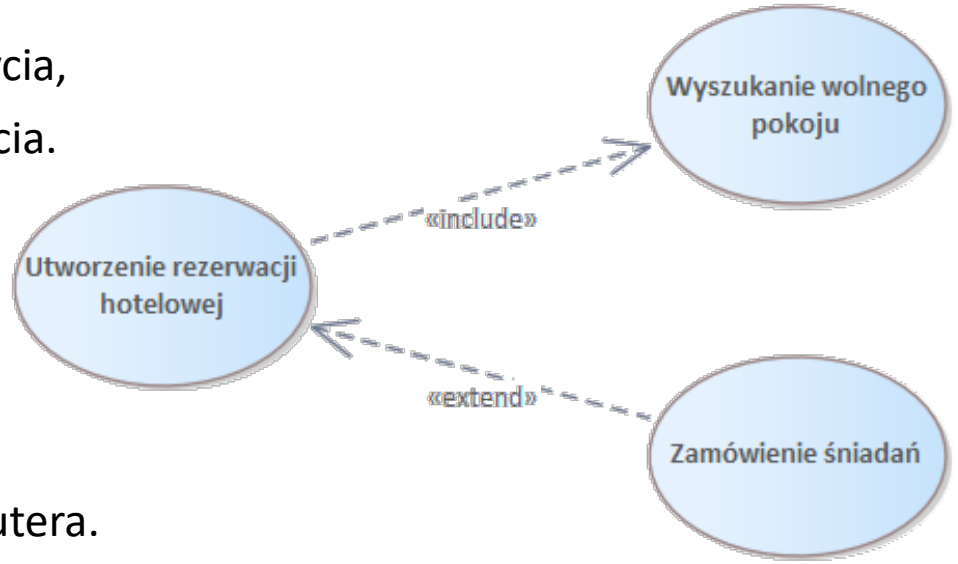
Powodzenia!

© RedPill – szkolenia IT konsulting

1. W którą stronę powinny być strzałki w relacjach *include* i *extend* na diagramach przypadków użycia?

<<include>> i <<extend>> to dość podobne relacje używane na diagramach przypadków użycia.
Problem w tym, że strzałki w tych relacjach są (pozornie) niespójne:

- strzałka w relacji <<include>> wskazuje na podrzędny przypadek użycia,
- strzałka w relacji <<extend>> wskazuje na nadrzędny przypadek użycia.



Na przykład na pokazanym diagramie:

- przypadek użycia „Utworzenie rezerwacji hotelowej” wywołuje (*include*) przypadek użycia „Wyszukanie wolnego pokoju”,
- przypadek użycia „Zamówienie śniadań” rozszerza (*extend*) przypadek użycia „Utworzenie rezerwacji hotelowej”

Trudno to zapamiętać. Łatwo się pomylić. I paść ofiarą złośliwego rekrutera.

Ale jest na to rada: wystarczy zapamiętać prostą mnemotechnikę:

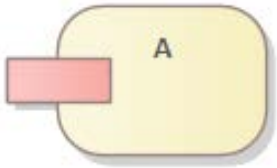
include – include’owany, extend – extend’owany

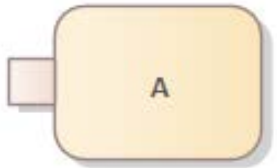
A tłumacząc bardziej po ludzku:

- include pokazuje na include’owany (dołączany, podrzędny) przypadek użycia,
- extend pokazuje na extend’owany (rozszerzany, nadrzędny) przypadek użycia.

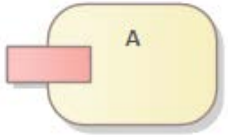
© RedPill – szkolenia IT konsulting

2. Co to za elementy?

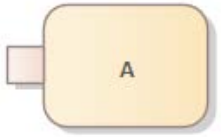




Wyglądają podobnie. Występują na diagramach czynności (aktywności) w UML. Gdyby nie „wypustki”, byłyby nie do odróżnienia (bo stopień zaokrąglenia rogów nie jest istotnym aspektem w notacji UML). Jak łatwo się domyślić, należy je odróżnić po tych „wypustkach”.

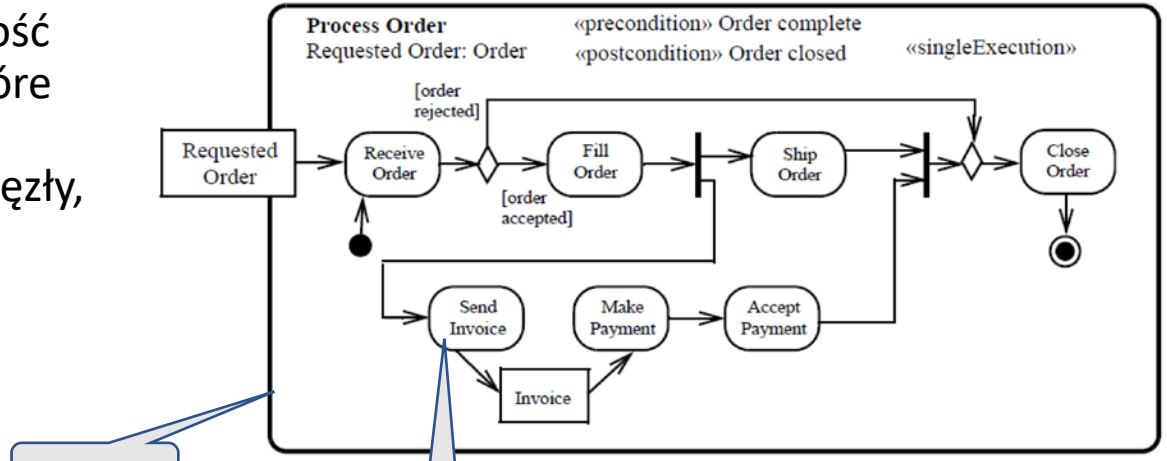


Prostokąt częściowo zatopiony w elemencie o nazwie A to **parametr** czynności (ActivityParameterNode). Parametry służą do przekazywania wartości do i z wykonania czynności. Skoro tak, to element A musi być **czynnością** (*activity*).



Kwadrat przyczepiony do brzegu elementu o nazwie A to **pin**. Pin oznacza dane wejściowe przekazywane do akcji (pin wejściowy – InputPin) lub dane wyjściowe z akcji (pin wyjściowy – OutputPin). Skoro tak, to element A musi być **akcją** (*action*).

A po co w ogóle w UML-u są dwa rodzaje elementów – czynność (*activity*) i akcja (*action*)? Czynność to zachowanie złożone, które modelujemy na diagramie używając akcji. Czyli na diagramie czynności umieszczamy **akcje** (nie czynności!), bramki i inne węzły, łącząc je przepływami. Czynność może na diagramie pełnić co najwyżej rolę „opakowania”, w którym jest zawarty model złożony z akcji, bramek i przepływów. Czynności nie łączymy przepływami sterowania!



Źródło: UML 2.5 Specification

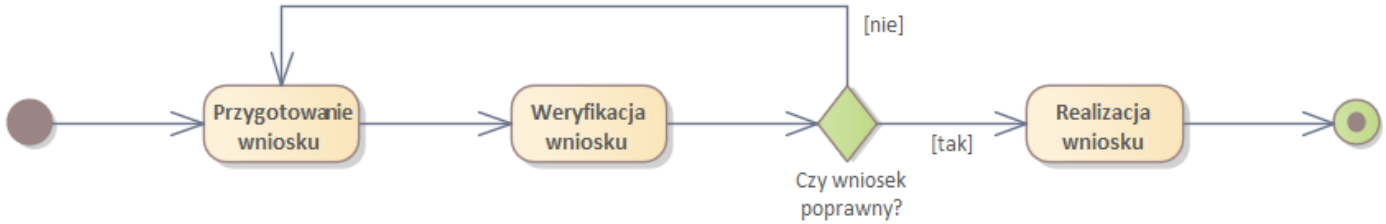
© RedPill – szkolenia IT konsulting

3. Ile razy wykona się akcja, do której wchodzi kilka przepływów sterowania?

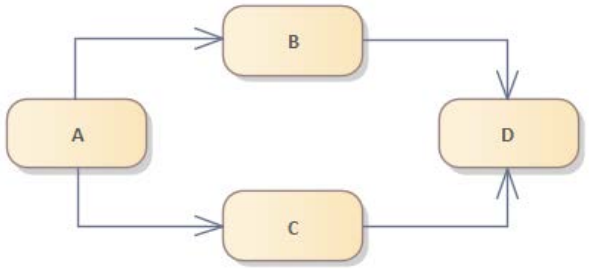
Pytanie może brzmieć różnie, ale dotyczy diagramów czynności (aktywności) i sytuacji, gdy z jednej akcji wychodzi kilka przepływów lub do jednej akcji wchodzi kilka przepływów. Bez wykorzystania bramek decyzyjnych ani rozgałęzień Fork/Join. Specyfikacja UML mówi jasno:

- wykonanie akcji rozpoczyna się, gdy na wszystkich wchodzących przepływach sterowania są oferowane tokeny (niejawne połączenie – *implicit join*),
- po zakończeniu akcji tokeny są oferowane na wszystkich przepływach z niej wychodzących (niejawne rozgałęzienie – *implicit fork*).

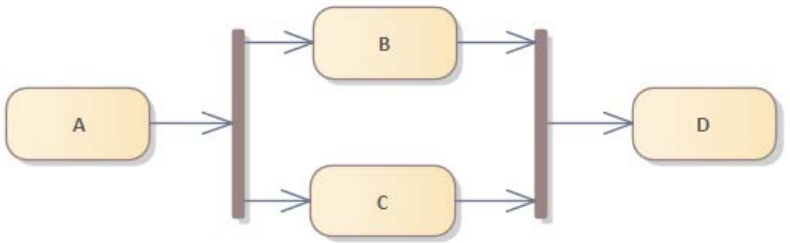
Zatem odpowiedź na pytanie, ile razy wykona się akcja „Przygotowanie wniosku” brzmi: **ani razu**. Do akcji tej wchodzą dwa przepływy, ale nigdy na obu jednocześnie nie pojawią się tokeny. Ten model blokuje się na samym początku.



Odpowiedź na pytanie, ile razy wykona się akcja D, brzmi: **jeden raz**. Z akcji A wychodzą dwa tokeny, i oba ostatecznie trafiają do akcji D i ją uruchamiają. To tak, jakbyśmy mieli niejawne węzły Fork (rozgałęzienie) i Join (połączenie równoległych wątków).



Ukryte rozgałęzienie i połączenie



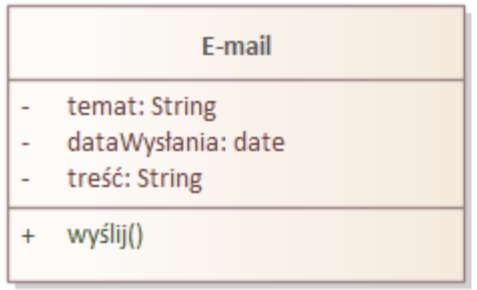
Model równoważny

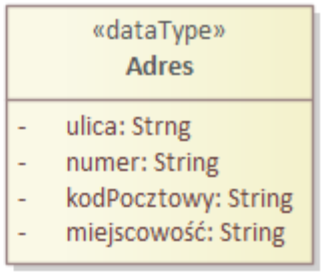
© RedPill – szkolenia IT konsulting

4. Czym się różni klasa od typu danych?

Na diagramach klas możemy używać dwóch rodzajów elementów: klas (*class*) i typów danych (*data type*). Czym się od siebie różnią?

Pytanie jest nieco podchwytliwe, szczególnie gdy patrzymy na podany przykład klasy E-mail i typu danych Adres. Nie, różnica nie polega na tym, że klasy mogą mieć metody, a typy danych nie. Zarówno klasy, jak i typy danych to klasyfikatory. Jedne i drugie mogą mieć zarówno atrybuty, jak i metody.





Różnica jest następująca:

- Obiekty klas mają **tożsamość** (*identity*). Dwa obiekty danej klasy, nawet jeśli mają identyczne wartości atrybutów, są różnymi obiektami (ich porównanie zwróci fałsz). Z drugiej strony, jeśli zmienimy wartości atrybutów danego obiektu, on nadal pozostaje tym samym obiektem. Tak jak człowiek – może zmienić imię czy nazwisko, ale nadal jest tą samą osobą.
- Wartości typów danych nie mają tożsamości i są porównywane po wartościach atrybutów. Jeśli dwie wartości typu danych mają identyczne atrybuty, to są sobie równe (ich porównanie zwróci prawdę). Jeśli w wartości typu mienimy jakiś atrybut, to otrzymamy nową wartość, różną od poprzedniej.

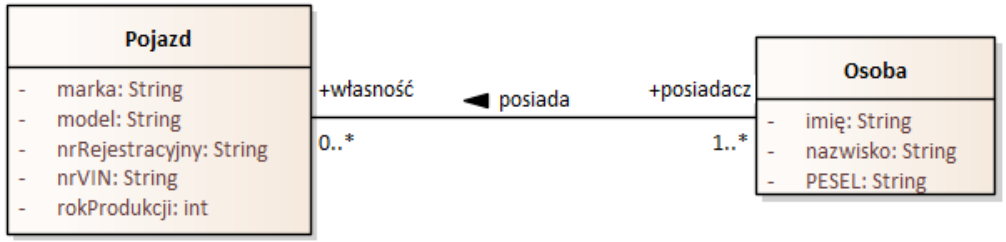
© RedPill – szkolenia IT konsulting

5. Co to są *cardinality* i *multiplicity*? Czym się od siebie różnią?

Pytanie dotyczy dwóch terminów wykorzystywanych w specyfikacji UML, głównie w kontekście diagramów klas i obiektów. Problem polega na tym, że w języku polskim nie mamy na te dwa terminy osobnych słów i zwykle posługujemy się jednym słowem – **liczebność**.

Multiplicity stosujemy na diagramach klas. Składa się z dolnego i górnego ograniczenia, np.:

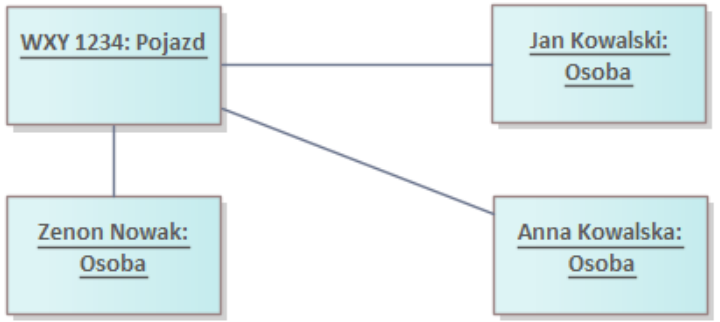
- 0..1 – od zera do jednego obiektu,
- 1..* - od jednego do dowolnie wielu obiektów.



Multiplicity ogranicza liczbę obiektów połączonych asocjacją. Zgodnie z przykładowym diagramem:

- jedna osoba może posiadać od zera do dowolnie wielu (0..*) pojazdów,
- jeden pojazd może być posiadany przez od jednej do dowolnie wielu (1..*) osób.

Natomiast ***cardinality*** to liczba obiektów w konkretnej kolekcji. Możemy ją zobaczyć na diagramie obiektów, gdzie pokazujemy konkretne obiekty – egzemplarze klas – i powiązania między nimi. Na przykładowym diagramie pojazd WXY 1234 ma połączone 3 obiekty klasy Osoba. Zatem *cardinality* kolekcji osób połączonych relacją posiada z pojazdem WXY 1234 wynosi **3**.



Można więc powiedzieć, że ***multiplicity* to ograniczenie nałożone na dopuszczalne wartości *cardinality***.

© RedPill – szkolenia IT konsulting

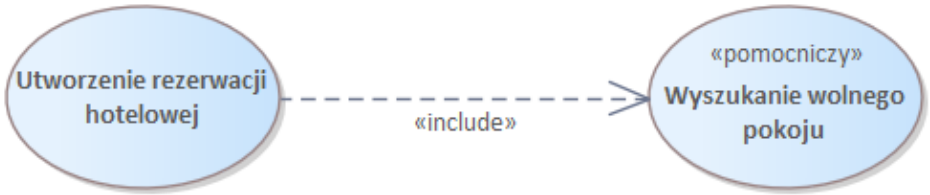
6. Co to za napisy: <<include>>, <<extend>>, <<enumeration>>, <<device>>?

Odpowiedziałbyś, że to stereotypy? Niestety nie, to pułapka. To tak zwane słowa kluczowe (*keywords*) w UML. Problem polega na tym, że słowa kluczowe i stereotypy zapisuje się dokładnie tak samo – w nawiasach << >>. Ale różnica jest zasadnicza:

- Słowa kluczowe pozwalają odróżnić od siebie różne pojęcia UML o tym samym symbolu graficznym. Na przykład relacje include i extend na diagramach przypadków użycia mają tą samą postać graficzną. Gdyby nie słowa kluczowe, nie odróżnilibyśmy ich od siebie. Lista słów kluczowych jest zdefiniowana w specyfikacji UML. Nie można tworzyć własnych słów kluczowych.
- Stereotypy to mechanizm oznaczania elementów modelu przez twórcę jego własnymi etykietami. Element oznaczony stereotypem nie zmienia swojego podstawowego znaczenia. Przypadek użycia ze stereotypem pozostaje przypadkiem użycia. Zyskuje jednak dodatkowe znaczenie, określone przez twórcę stereotypu. Można tworzyć dowolne własne stereotypy.

Na przykładowym diagramie:

- <<include>> jest słowem kluczowym UML, pozwalającym odróżnić relację include od innych relacji o tej samej postaci graficznej (np. extend),
- <<pomocniczy>> jest stereotypem zdefiniowanym przez twórcę modelu.



© RedPill – szkolenia IT konsulting