Overlapping

Istotą tego dziedziczenia jest fakt, że jeden obiekt może posiadać wiele wariantów, a konkretnie należeć do wielu typów. Wyobraźmy sobie taki scenariusz

Public abstract class Person

Public class Employee extends Person

Public class Student extends Person

Student może pracować na uczelni. Pracownik na uczelni może studiować. Tzn. Że każdy obiekt może należeć do obu typów równocześnie. Można wykonać to na jeden z dwóch sposobów.

Pierwszy polega na tzw. Spłaszczeniu. Polega ono na usunięciu podklas (Employee, Student) i przeniesieniu ich metod oraz pól do nadklasy (Person). Oczywiście przy takim scenariuszu klasa Person przestaje być abstrakcyjna.
Następnie tworzymy Enum który będzie definiować nam nasze typy (Employee, Student) a w klasie Person dodajemy EnumSet który przechowuje wybrane enumy. Enum służy Państwu do dodania weryfikacji w metodach dostępnych, aby upewnić się czy obiekt może pobrać wartość.

Załóżmy, że Employee miał pole salary które przechowuje jego pensję. Jeśli dany obiekt w EnumSet nie ma wartości Employee to nie można pobrać jego pensji bo jej fizycznie nie posiada.

Druga metoda dotyczy wykorzystania kompozycji. Każde dziedziczenie zamieniamy na kompozycję. O tym jakiego typu/typów dany obiekt jest decyduje to która kompozycja jest utworzona. Jeśli osoba jest dwóch typów obie kompozycje są utworzone. Tu należy pamiętać, że elementy które powinny być dziedziczone trzeba obsłużyć. Np. firstName I lastName musza się jakoś znaleźć w klasie Employee skoro nie może ich pobrać z dziedziczenia. Można przez referencje, można powielać pola zależy co w danej sytuacji jest lepsze.

Wielodziedziczenie

Wielodziedziczenie poza kilkoma wyjątkami jak np. C++ nie istnieje w programowaniu więc należy to obejść. Jedną z prostszych metod jest wykorzystanie interfejsów ponieważ tych możemy implementować w klasie ile chcemy.

Załóżmy, że mamy klasy ElectricCar i CombustionCar. Do tego mamy klasę HybridCar która RÓWNOCZEŚNIE dziedziczy z obu nadklas. Aby to osiągnąć wybieramy nadklasę, która jest najmniej złożona i tworzymy odpowiedni interfejs, np. ICombustionCar. Ten interfejs implementujemy do dwóch klas: HybridCar i CumbustionCar.

W HybridCar implementujemy interfejs dla zasymulowania dziedziczenia. Uzyskujemy klasę HybridCar która równocześnie jest typu ElectricCar i ICombustionCar. Teraz dla jeszcze wierniejszego symulowania dziedziczenie ten interfejs dodajmy również do klasy CombustionCar. Dzięki temu CombustionCar i HybridCar są tego samego typu ICombustionCar.

Docelowo klasy powinny wyglądać tak:
ElectricCar extends Car
CombustionCar extends Car implements ICumbustionCar
HybridCar extends ElectricCar implements ICumbustionCar

Wieloaspektowe

Dziedziczenie wieloaspektowe jest bardzo specyficzne gdyż gałąź dziedziczenie zależy do aspektu. Tzn. Na diagramie UML nie traktujemy tego jako dwa dziedziczenie które mają miejsce równocześnie, a takie które zależy od aspektu na jaki patrzymy. W związku z tym jeśli mamy klasy

Abstract Person

Employee extends Person
Student extends Person

Male extends Person
Female extends Person

Jak widzicie mamy 4 klasy które dziedziczą po klasie Person, ale te dziedziczenia istnieją tylko w zależności od kontekstu. Powinniśmy to rozumieć tak, że tylko 2 klasy dziedziczą równocześnie po Person co zakładam, że brzmi lekko nielogicznie. Skoro brzmi nielogicznie to jak to zaprogramować? Rozwiązane jest analogiczne jak przy overlapping tzn. Musimy się czegoś pozbyć. Wybieramy tą gałąź dziedziczenia, która jest dla nas najmniej problematyczna. W naszym przykładzie będzie to aspekt płci (drugi aspekt to zajęcie osoby). Teraz mogą Państwo albo dokonać spłaszczenia i klasy Male i Female zawrzeć w klasie Person która przestaje być abstrakcyjna. Albo dodać nową klasę która uwzględnia Male i Female oraz kompozycję. W przypadku wieloaspektowego na potrzeby MP sugerują opcję ze spłaszczeniem. Na potrzeby egzaminu proszę również poczytać o metodzie z kompozycją.

Dynamiczne

Na potrzeby dynamicznego posłużę się tym samym przykładem co w overlapping aby dobrze zobrazować Państwu różnicę.

Abstract Person
Employee extends Person
Student extends Person

Dziedziczenie dynamiczne polega na tym, że obiekt może zmieniać typ w trakcie działania programu, ale może być tylko jednego typu równocześnie. Dla przypomnienia overlapping dopuszcza posiadanie wielu typów równocześnie. Dlatego nie można i nie należy próbować prezentować tych dwóch dziedziczeń na jednym przykładzie. To nie działa wtedy prawidłowo.

Zarówno metoda Employee jak i Student powinny zawierać konstruktor, który jako parametr przyjmuje obiekt drugiego typu

Employee(Student student, [pola wymagane dla employe])
Student (Employee employee, [pola wymagane dla student])

Następnie w konstruktorze przepisujemy wartości pól wspólnych, uzupełniamy pola wymagane i kasujemy stary obiekt. W ten sposób nasz obiekt zmienił typ dynamicznie.