- **Zaprojektowano klasy Rectange i Square i w „naturalny” sposób relację dziedziczenia między nimi (każdy kwadrat jest prostokątem). (...) Co można powiedzieć o spełnianiu przez taką hierarchię zasady LSP w kontekście poniższego kodu klienckiego?**

Dana hierarchia klas nie spełnia zasady LSP, bo warunek wyjścia `set` dla `Width` i `Height` jest osłabiony w klasie `Square` względem klasy `Rectangle` (tj. wartościami przypisanymi do `Width` i `Height` niekoniecznie są te, które zostały podane przez klienta). Sprawia to, że obiekty klasy `Rectangle` nie mogą być zastępowane obiektami klasy `Square`, gdyż w przypadku `Square` *settery* nie zachowują się w oczekiwany dla `Rectangle` sposób (ustawienie `Width = x` i `Height = y` niekoniecznie spowoduje, że pole figury to `x * y`, jak byśmy oczekiwali dla dowolnych prostokątów).

- **Jak należałoby zmodyfikować przedstawioną hierarchię klas, żeby zachować zgodność z LSP w kontekście takich wymagań? Jak potraktować klasy Rectangle i Square? Odpowiedź zilustrować działającym kodem.**

Jedną z możliwości mogłoby być usunięcie relacji dziedziczenia pomiędzy `Rectangle` i `Square`, gdyż `Square` w rzeczywistości nie rozszerza `Rectangle` o żadną nową funkcjonalność a próbuje jedynie wyegzekwować pewien niezmiennik dla szczególnych przypadków obiektów klasy `Rectangle`.

Samo usunięcie dziedziczenia spowodowałoby jednak, że kliencki kod do liczenia pola musiałby oddzielnie obsługiwać oba typy. Rozwiazaniem jest wprowadzenie nowej, abstrakcyjnej klasy `Shape`, po której dziedziczyły by zarówno klasa `Rectangle` jak i `Square`.

Rzeczą, która łączy wszystkie figury dwuwymiarowe jest posiadanie pola, więc klasa `Shape` będzie posiadała metodę `GetArea`.

Takie rozwiązanie powoduje, że w dowolnym kontekście `Shape` możemy zastępować obiektami podklas `Rectangle` i `Square` bez wprowadzania błędów. Jednocześnie dodatkowo przestrzegamy *Open-Closed Principle*, bo klasa `Shape` umożliwia rozszerzanie poprzez tworzenie nowych klas potomnych.