|  |
| --- |
| **InterpreterProduct (The Product Class)** |
| **Line** | **Code** |
| **1****2****3****4****5****6****7****8****9****10****11****12****13****14****15****16****17****18****19****20****21****22****23****24****25****26****27****28****29****30****31****32****33****34****35****36****37****38****39****40****41****42****43** | **using** System;**namespace** InterpreterProduct { **public enum** ProductColor { Red, Green, Blue, Cyan, Magenta, Yellow, White, Black }; **public class** Product { **public readonly string** Name; **private float** price; **private float** weight; **private** ProductColor color; **public float** Price{ **get** { return price; } **private set** { **if** (**value** < 0) **throw new** Exception("Price can't be -ve!"); price = **value**; } } **public** ProductColor Color { **get** { return color; } **private set** { color = **value**; } } **public float** Weight { **get** { return weight; } **private set** { **if** (**value** <= 0) **throw new** Exception("Weight must be +ve!"); weight = **value**; } } **public** Product(**string** name, **float** price, **float** weight, ProductColor color){ Name = name; Price = price; Weight= weight; Color = color; } **public void** Show(){ Console.WriteLine(Name); } }} |

|  |
| --- |
| **InterpreterProduct** |
| **Line** | **Code** |
| **1****2****3****4****5****6****7****8****9****10****11****12****13****14****15****16****17****18****19****20****21****22****23****24****25****26****27****28****29****30****31****32****33****34****35****36****37****38****39****40****41****42****43****44****45****46****47****48****49****50****51****52****53****54****55****56****57****58****59****60****61****62****63****64****65****66****67****68****69****70****71****72****73****74****75****76****77****78****79****80****81****82****83****84****85****86****87****88****89****90****91****92****93****94****95****96****97****98****99****100****101****102****103****104****105****106****107****108****109****110****111****112****113****114****115****116****117****118****119****120****121****122****123****124****125****126****127****128****129****130****131****132****133** | **using** System;**using** System.Collections.Generic;**namespace** InterpreterProduct { **class** Program { **static void** Main(**string**[] args) { Product[] ps = { **new** Product("A", 20F, 0.56F, ProductColor.Blue), **new** Product("B", 25F, 1.66F, ProductColor.Green), **new** Product("C", 17F, 0.56F, ProductColor.Blue), **new** Product("D", 29F, 2.47F, ProductColor.Red), **new** Product("E", 31F, 3.53F, ProductColor.Black), **new** Product("F", 22F, 0.66F, ProductColor.Green), **new** Product("G", 14F, 4.12F, ProductColor.Blue), }; ProductFinder pf = **new** ProductFinder(ps); //showBlueProducts(pf); //Q1:Blue products where price < $15 **foreach** (Product p **in**  pf.byColorAndBelowPrice(ProductColor.Blue,15)) p.Show(); //Q2:Products < 1KG but not Green color //Q3:Blue products > $15 but < 1KG Console.ReadKey(); } **static void** showBlueProducts(ProductFinder pf) { **foreach** (Product p **in** pf.byColor(ProductColor.Blue)) p.Show(); } } **abstract class** Spec { **abstract public bool** isSatisfiedBy(Product p); } **class** ColorSpec : Spec { ProductColor color; **public** ColorSpec(ProductColor color) { **this**.color = color; } **override public bool** isSatisfiedBy(Product p) { **return** (color == p.Color); } } **class** BelowPriceSpec : Spec { **float** threasholdPrice; **public** BelowPriceSpec(**float** thPrice) { threasholdPrice = thPrice; } **override public bool** isSatisfiedBy(Product p) { **return** p.Price < threasholdPrice; } } **class** NotSpec : Spec { Spec specToNegate; **public** NotSpec(Spec spec) { specToNegate = spec; } **override public bool** isSatisfiedBy(Product p) { **return** !specToNegate.isSatisfiedBy(p); } } **class** AndSpec : Spec { Spec lhs, rhs; **public** AndSpec(Spec lhs, Spec rhs) { **this**.lhs = lhs; **this**.rhs = rhs; } **override public bool** isSatisfiedBy(Product p) { **return** lhs.isSatisfiedBy(p) && rhs.isSatisfiedBy(p); } } **class** OrSpec : Spec { **private** Spec lhs, rhs; **public** OrSpec(Spec lhs, Spec rhs) { **this**.lhs = lhs; **this**.rhs = rhs; } **override public bool** isSatisfiedBy(Product p) { **return** lhs.isSatisfiedBy(p) || rhs.isSatisfiedBy(p); } } **class** ProductFinder { Product[] products; **public** ProductFinder(Product[] ps) { products = ps; } **public** IEnumerable<Product> by(Spec spec) { **foreach** (Product p **in** products) { **if** (spec.isSatisfiedBy(p)) **yield return** p; } } **public** IEnumerable<Product> byColor(ProductColor color) { **return** by(**new** ColorSpec(color)); } **public** IEnumerable<Product> byColorAndBelowPrice( ProductColor color, **float** price) { **return** by(**new** AndSpec(**new** ColorSpec(color), **new** BelowPriceSpec(price))); } **public** IEnumerable<Product> byBelowPriceAvoidingAColor( **float** price, ProductColor color) { **return** by(**new** AndSpec(**new** BelowPriceSpec(price), **new** NotSpec(**new** ColorSpec(color)))); } **public** IEnumerable<Product> byAll(**params** Spec[] specs) { **foreach** (Product p **in** products) { **int** i; **for** (i = 0; i < specs.Length; i++) { **if** (!specs[i].isSatisfiedBy(p)) **break**; } **if** (i == specs.Length) **yield return** p; } } **public** IEnumerable<Product> byAny(**params** Spec[] specs) { **foreach** (Product p **in** products) { **foreach** (Spec spec **in** specs) { **if** (spec.isSatisfiedBy(p)) { **yield return** p; **break**; } } } } }} |