**TESTAREA PROIECTULUI AFERENT DISCIPLINEI „PROIECTAREA SISTEMELOR SOFTWARE COMPLEXE”**

**Autor: Mădălin-Florin, Goian**

**Email: madalin.goian@student.upt.ro**

04 Ianuarie 2024

# INTRODUCERE

În acest referat se va prezenta testarea proiectului realizat pentru disciplina „Proiectarea Sistemelor Software Complexe”, cel îl are ca profesor titular pe domnul ș.l.dr.ing. Daniel IERCAN, folosind diferite tehnici în realizarea testelor unitare.

* 1. **PREZENTARE GENERALĂ A PROIECTULUI**

Proiectul este creat în contextul plasării unei comenzi în cadrul unui magazin online. În proiectare s-au folosit principii ce țin de „Domain Driven Design”. Parafrazând din cursul domnului profesor D. Iercan, sistemul este bazat pe evenimente ale domeniul care creează procese, ce la rândul lor generează o altă listă de evenimente [1]. Aplicat pe proiectul curent, utilizatorul va introduce de la tastatură o listă de produse, acțiune ce va reprezenta primul eveniment. Aceasta va crea o comandă care va declanșa fluxul de lucru ce va prelucra lista introdusă. La rândul său, fluxul de lucru va genera fie un eveniment de încheiere cu succes, fie un eveniment de încheiere cu eșec. Întreg procesul poate fi vizualizat în figura 1.

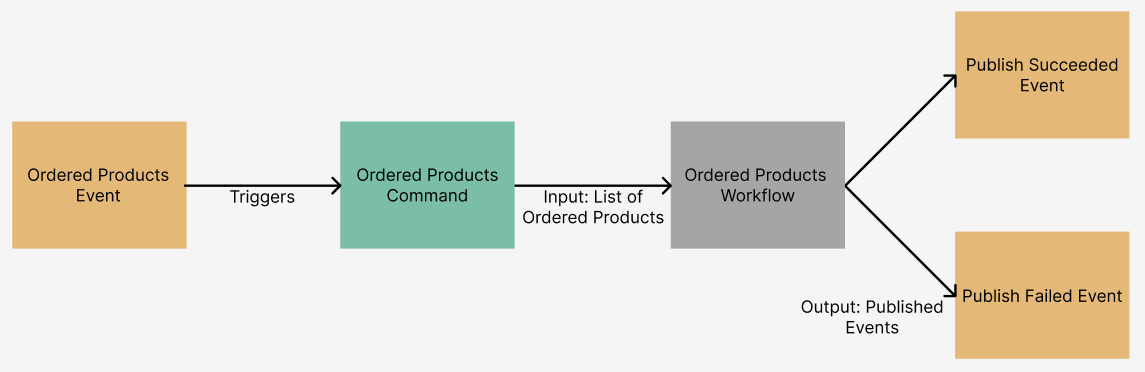


Fig. 1 – Fluxul de execuție al aplicației

Fluxul de lucru al aplicției execută operații precum validarea sau invalidarea comenzii, calcularea prețurilor totale ale produselor comandate, iar mai apoi publicarea comenzii valide.

Pentru ca o comandă să fie validă trebuie ca și codul produsului și cantitatea introduse de la tastatură să fie valide. Pentru ca și codul produsului să fie valid, acesta trebuie să fie format dintr-un șir de cinci cifre și să existe într-un „depozit”, aflat în fișierul „ProductRepository.cs”, vizibil și în figura 2. Pentru ca și cantitatea introdusă să fie validă, aceasta trebuie să fie număr întreg pozitiv, diferit de zero.

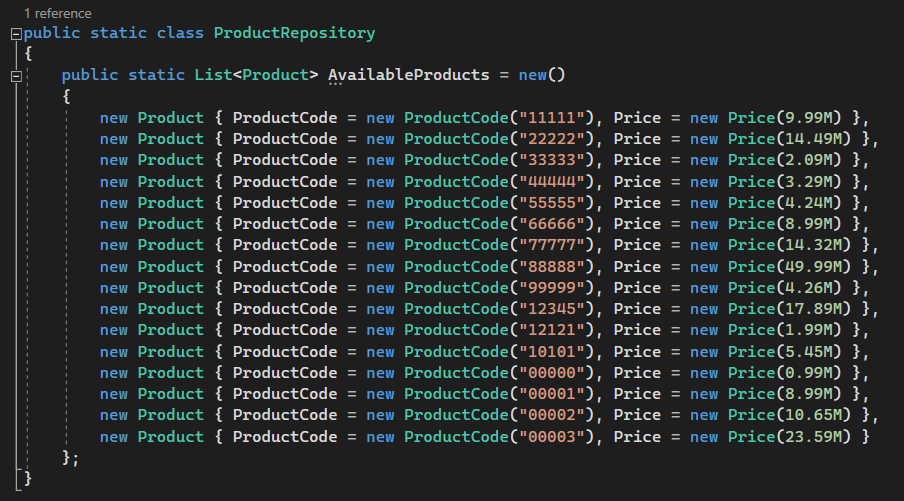


Fig. 2 – Depozitul de produse, realizat în cod

1. **TESTARE**
   1. **CLASA „PRICE”**

Această clasă are ca și membru o proprietate publică „Value”, un constructor, în care verificăm dacă parametrul introdus este valid, în caz contrar aruncându-se excepție, metodă „ToString”, o metodă statică de conversie la tipul „Price”, „TryParsePrice”, și o metodă privată ce verifică să fie introdus un număr întreg pozitiv, diferit de zero.

Au fost create două teste pentru a verifica dacă validarea datelor în constructor se efectuează corespunzător, după cum se poate observa și în figura 3. Parametrii testelor au fost selectați respectând principiul de testare al domeniului, discutate la curs [2], alegând astfel mereu câte cinci valori pentru fiecare subdomeniu.

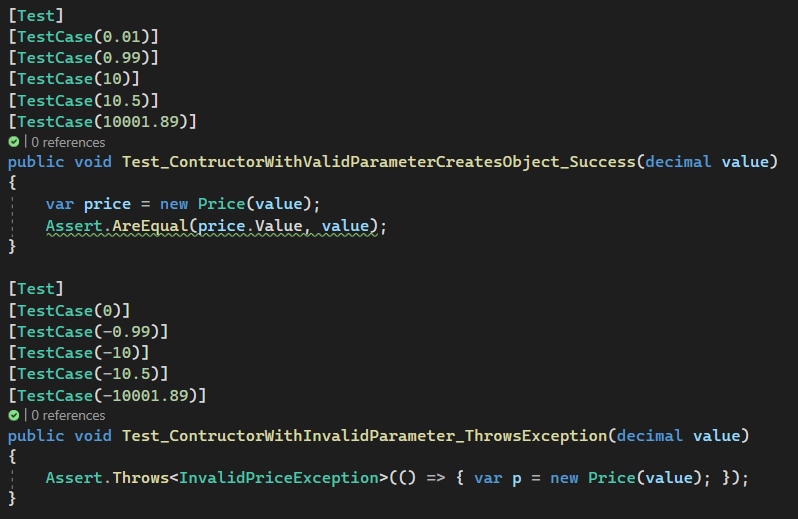


Fig. 3 – Testele constructorului

În cazul metodei „TryParsePrice” s-au considerat pentru cazurile de testare în care nu se poate face conversia două subdomenii: cel al tipurilor neconvertibile la tipul „decimal” și cel al numerelor aparținând intervalului (-∞, 0]. Acest test poate fi observat, împreună cu cel pentru conversia reușită, în figura 4. De asemenea, s-a realizat un test și pentru metoda „ToString”.

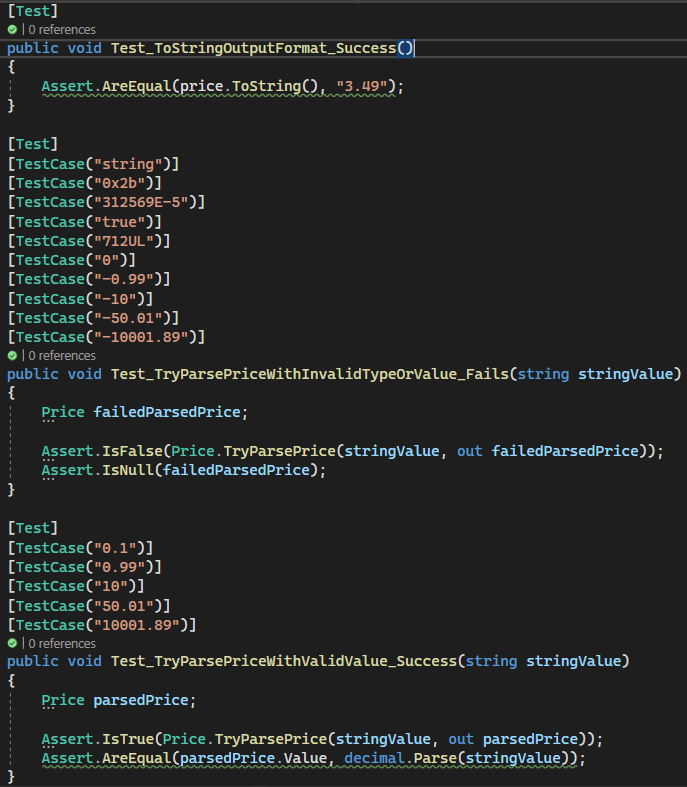


Fig. 4 – Testele metodelor „TryParsePrice” și „ToString”

* 1. **CLASA „PRODUCT CODE”**

Asemănător clasei „Price”, clasa „ProductCode” are ca și membru proprietatea publică „Value”, metodă „ToString”, constructorul în care se verifică validitatea parametrului introdus și metoda „TryParse”. Pe lângă acestea mai avem o metodă privată care compară parametrul introdus în constructor cu un șablon de tip regex.

Pentru alegerea cazurilor de testare s-a ținut cont de același principiu discutat la punctul 2.1. Testele pot fi observate în figurile 5 și 6.



Fig. 5 – Testele constructorului și a metodei „ToString”

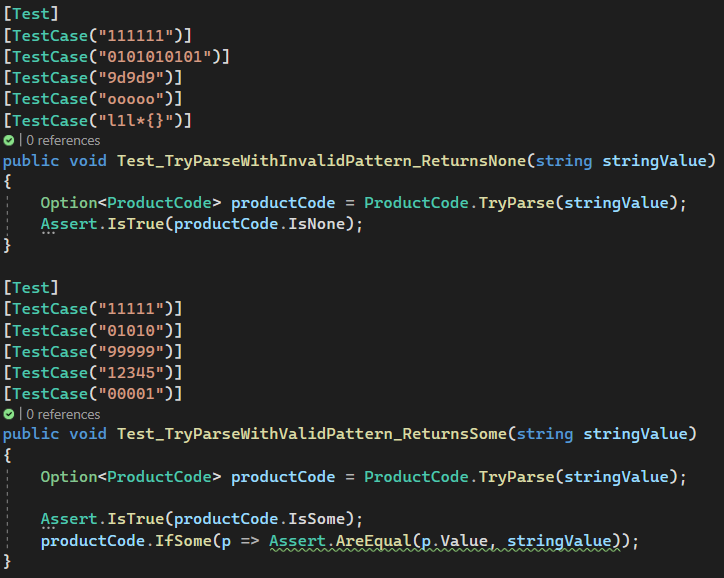


Fig. 6 – Testele metodei „TryParse”

* 1. **CLASA „QUANTITY”**

Asemănător celor două clase menționate la punctele 2.1 și 2.2, și aceasta are o proprietate „Value”, constructor în care se face validarea, metodă „TryParse”, „ToString” și metoda ce verifică să fie valid parametrul din constructor. Condiția de data aceasta este ca numărul să fie întreg pozitiv, diferit de 0.

Testele sunt realizate într-o manieră asemănătoare subpunctului 2.2 și se pot observa în figurile 7 și 8.

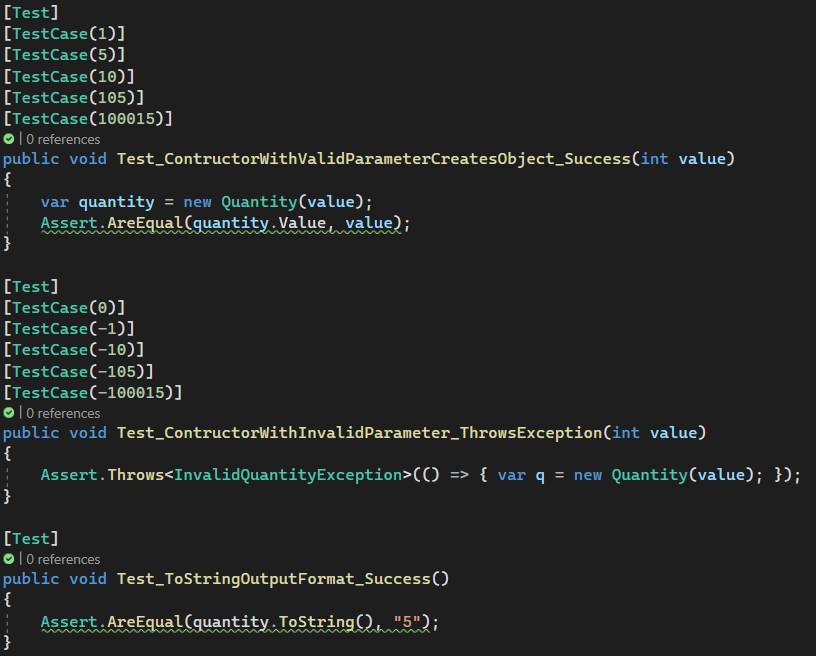


Fig. 7 – Testele constructorului și a metodei „ToString”

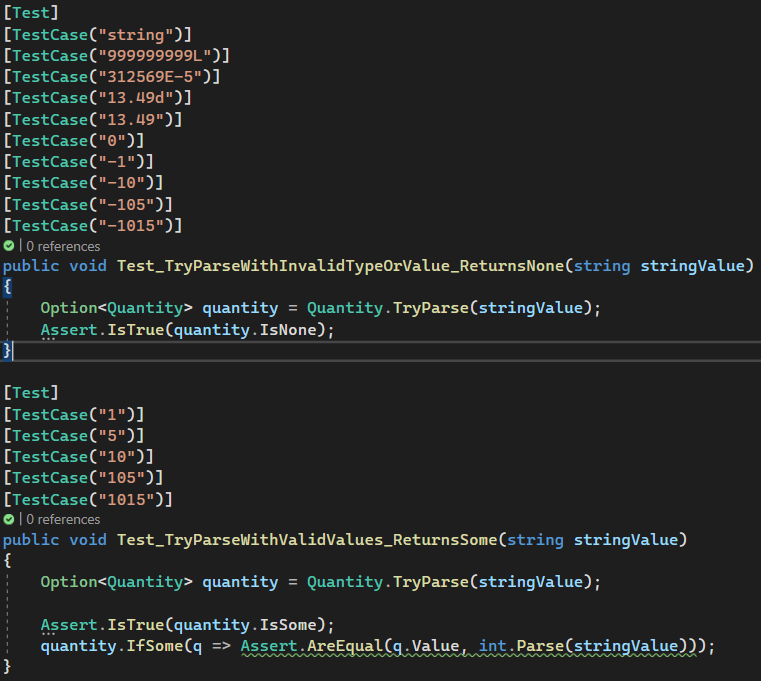


Fig. 8 – Testele metodei „TryParse”

* 1. **CLASA STATICĂ „ORDERED PRODUCT OPERATIONS”**

Clasa statică „OrderedProductsOperations” funcționează mai degrabă ca și un „namespace” și are definită în ea funcțiile ce se apelează în interiorul fluxului de lucru. Principalele funcții sunt „ValidateOrderedProducts”, „CalculateOrderedProducts” și „PublishOrderedProducts”.

Clasa de test a fost implementată folosind mai multe tehnici de testare discutate la curs [2], precum șablonul de proiectare „Dependency Injection”, mock-uri și „Dummy Objects”, tehnici pe care le voi descrie mai amănunțit în paragrafele următoare.

Funcția „ValidateOrderedProducts” primește ca parametrii două funcții ce ar trebui să facă anumite verificări, precum dacă obiectul există în depozit și dacă obiectul mai este în stoc. Ca și idee de extindere, aceste funcții ar fi urmat să facă aceste verificări intr-o bază de date. Funcțiile sunt membrii ai interfeței „IProductValidator”, vizibilă în figura 9.

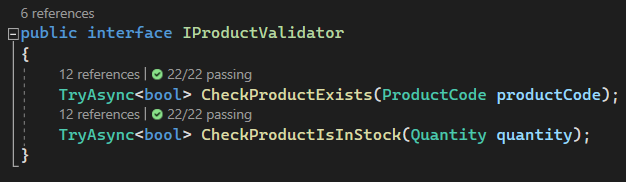


Fig. 9 – Interfața „IProductValidator”

În clasa de test s-a realizat o injecție a acestei interfețe, care este manipulată prin intermediul unui mock (figura 10). Cele două metode vor returna astfel orice valori de care avem nevoie în realizarea testelor unitare (figura 11).

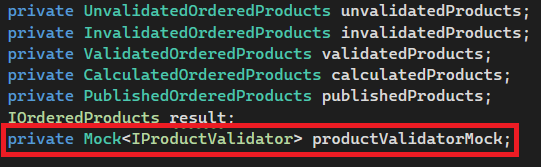


Fig. 10 – Injectarea dependinței

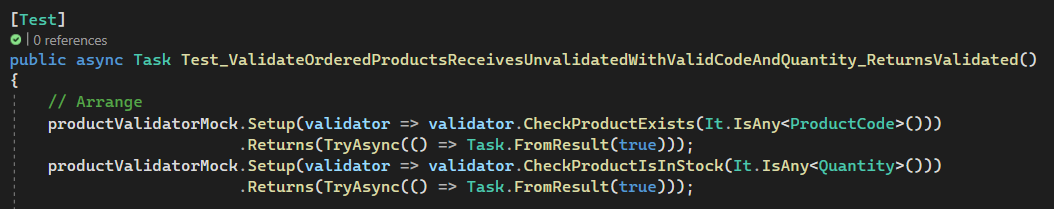


Fig. 11 – Manipularea interfeței prin mock

Tot în clasa de test avem mai multe „dummy objects”, de care ne folosim fie în crearea tipurilor de date ce reprezintă stările sistemului, fie pentru compararea rezultatului actual cu cel așteptat (figura 12).

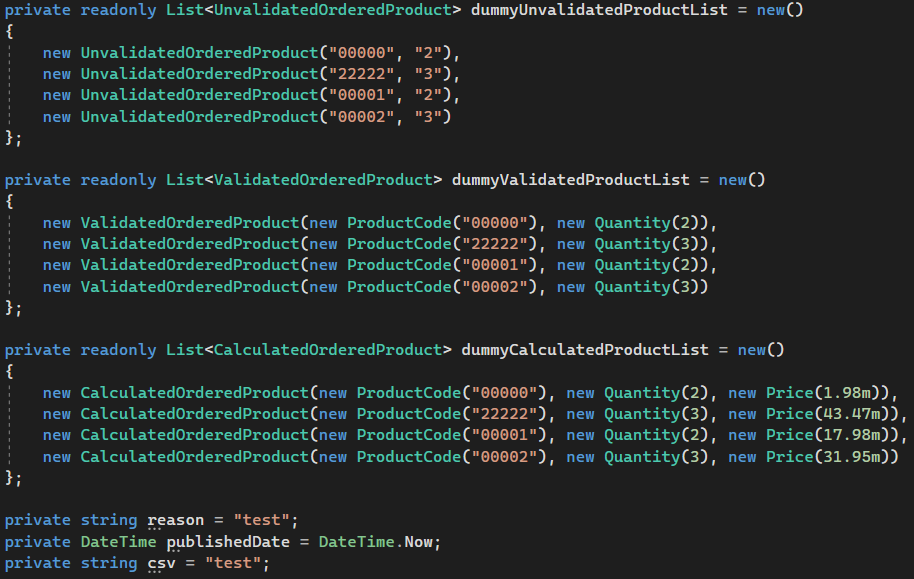


Fig. 12 – „Dummy Objects”

Și aici s-a ținut cont de principiul de testare al domeniului, cu un exemplu de test vizibil în figura 13.



Fig. 13 – Exemplu de test parametrizat

* 1. **CLASA „ORDER PRODUCTS WORKFLOW”**

Această clasă conține doar o metodă „ExecuteAsync” ce reprezintă însuși fluxul de lucru. Metoda primește ca și parametrii comanda de execuție, precum și cele două metode ale interfeței „IProductValidator” și returnează unul dintre cele două evenimente (fie „ProductOrderPublishedFailedEvent”, fie „ProductOrderPublishedSucceededEvent”).

Și în acest caz s-au folosit aceleași tehnici prezentate la subpunctul 2.4, precum injecția dependinței, mock și „dummy object”, după cum se poate observa și în figura 14.

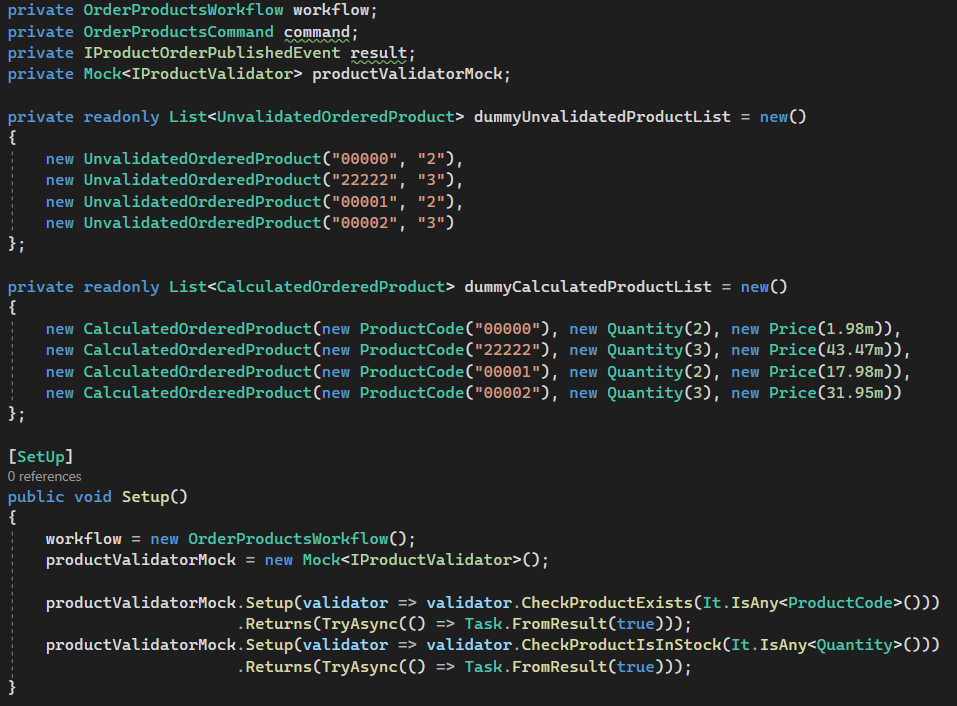


Fig. 14 – Diferitele tehnici de testare ale fluxului de lucru

Și în testele parametrizate ale acestei clase s-a ținut cont de principiul testării domeniului, cu un exemplu de test vizibil în figura 15.



Fig. 15 – Exemplu de test parametrizat

1. **CONCLUZII**

Realizarea testării s-a realizat folosind diferite tehnici de testare precum injecția dependinței, mock-uri și obiecte „dummy”. De asemenea s-a ținut cont de principiul testării domeniului de valori. Proiectul a ajuns cu aceste teste unitare să aibă o acoperire totală de 68,6%.

# BIBLIOGRAFIE

1. D. Iercan, „Proiectarea Sistemelor Software Complexe,” Dec. 2023
2. O. Baniaș, „Testarea Aplicațiilor Software,” Ian. 2024