ANDRAS Alexandru-Razvan

Anul 4, I.S

Grupa 1.1

TEMA 2 - PSSC

Domain driven design – Value Object

VALUE OBJECT(obiect de valoare) este un tip care *se distinge numai de starea proprietăților sale*. Aceasta este, spre deosebire de o entitate care are un identificator unic și rămâne distinctă, chiar dacă proprietățile ei sunt identice, două obiecte de valoare cu aceleași proprietăți pot fi considerate egale.

Value objects reprezintă un model descris pentru prima dată în cartea de design Evans " Domain-Driven Design" și explicată mai departe în cursul lui Smith și Lerman, " Domain-Driven Design Fundamentals" .

Pentru a produce un tip imuabil în C #, tipul trebuie să aibă toată starea în care este construit. Orice proprietăți trebuie să fie doar pentru citire, ceea ce se poate realiza folosind setteri privați, ca în acest exemplu:

```
public class SomeValue
{
  public SomeValue(int value1, string value2)
  {
    this.Value1 = value1;
    this.Value2 = value2;
  }

  public int Value1 { get; private set; }
  public string Value2 { get; private set; }
}
```

Un alt exemplu de implementare in C++:

```
class Point {

  public:
            Point( int x, int y ): _x(x), _y(y) {}
            int getX() const { return _x; }
            int getY() const { return _y; }
            void setX( int x ) { _x = x; }
            void setY( int y ) { _y = y; }

  private:
            int _x;
            int _y;
            }
```

Acum putem declara obiectele:

```
Point aMutablePoint( 4, 10 );
const Point anImmutablePoint( 5, 5 );
```

Fiind imuabil, valoarea obiectelor nu poate fi schimbată odată ce acestea sunt create. Modificarea este una conceptuală, la fel ca și eliminarea celei vechi și crearea uneia noi. În mod frecvent, obiectul poate defini metode de ajutor (sau metode de extensii) care ajută la astfel de operațiuni. Obiectul încorporat în cadrul .NET este un bun exemplu de tip imuabil. Conversia unui șir într-o anumită manieră, cum ar fi facerea majusculă prin ToUpper (), nu schimbă efectiv șirul original, ci creează mai degrabă un șir nou. De asemenea, concatenarea a două șiruri nu modifică nici șirul original, ci creează mai degrabă un al treilea șir.

Deoarece obiectele de valoare nu au identitate, ele pot fi comparate pe baza stării lor colective. Dacă toate proprietățile componentelor lor sunt egale unul cu altul, atunci se poate spune că două obiecte de valoare sunt egale. Din nou, aceasta este aceeași ca și cu tipurile de șir.

Obiectele de valoare pot fi deosebit de utile ca mijloc de descriere a conceptelor într-o aplicație care are reguli intrinseci, dar care nu sunt ele însele entități. În multe aplicații, unele concepte care sunt descrise ca entități ar fi mai bine implementate ca obiecte de valoare. De exemplu, o adresă de expediere ar putea fi tratată ca o Entitate sau ca Obiect de Valoare, dar dacă ați compară două instanțe ale unei adrese care erau atât "123 Main St., Anytown, OH, 12345, USA" ei să fie egali. Două obiecte de valoare ar fi, dar două entități nu ar fi (din moment ce fiecare ar avea o identitate diferită). Acest lucru poate complica aplicația, deoarece verificarea duplicatelor devine acum o preocupare (care nu ar exista dacă s-ar fi folosit obiecte de valoare).

În general, validarea obiectelor de valoare nu ar trebui să aibă loc în constructorul lor. Constructorii de regulă nu ar trebui să includă logica, ci ar trebui să atribuie pur și simplu valori. Dacă este necesară validarea, ar trebui mutată într-o metodă din fabrică și, într-adevăr, este un model comun pentru a face constructorii Value Objects "privați" și pentru a oferi una sau mai multe metode statice publice pentru crearea obiectului de valoare. Acest lucru permite separarea preocupărilor , deoarece construirea unei instanțe dintr-un set de valori este o preocupare separată față de asigurarea validității valorilor.

Note:

1: În design-ul bazat pe domeniu, clasificarea Evans contrastează obiectele de valoare cu entitățile. Consider că entitățile sunt o formă obișnuită de obiect de referință, dar folosesc termenul "entitate" numai în cadrul modelelor de domeniu, în timp ce dihotomia de referință / valoare este utilă pentru toate codurile.

2: Cele mai multe comparații de obiecte în Java se fac cu equals- ceea ce este el însuși un pic ciudat, deoarece trebuie să amintească să se folosească acest operator, mai degrabă decât pe cel egal ==. Acest lucru este enervant, dar programatorii Java se obisnuiesc cu asta, deoarece String-ul se comportă în același mod. Alte limbaje OOP pot evita acest lucru - Ruby foloseste operatorul ==, dar permite ca acesta sa fie suprasolicitat.

3: Imutabilitatea este valoroasă și pentru obiectele de referință - dacă o comandă de vânzări nu se schimbă în timpul unei solicitări, atunci transformarea acesteia într-o formă imuabilă este valoroasă; și asta ar face în siguranță copierea, dacă ar fi util. Dar acest lucru nu ar face ca ordinul de vânzări să fie un obiect valoric dacă determin determinarea egalității pe baza unui număr unic de comandă.