Niklas Björk, WU21

Reflektioner inlämningsuppgift: En natt på museet, 5 Yhp

Kommandon

Alla verb utan ett argument jobbar på den plats i hierarkin som användaren befinner sig.
D.v.s. är du i där alla byggnader finns så appliceras det på byggnader. Går du in i en byggnad, appliceras det på rum i byggnaden och så kan det fortsätta. I detta fall kan man inte gå längre än in i en specifik byggnad. Attributen till verben används bara när användaren valt en specifik byggnad med [select]. Tillgängliga verb:

select [byggnadsnamn]
Gör att du kan gå in i en byggnad du skapat och där hantera rum samt konstverk.
select ..
Tar dig ut ur en byggnad som du gått in i. ”..” är klassiska tecken för att hoppa ner ett steg i en hierarki. Kanske skulle kunna bytas till t.ex. argument ”-out” om det är användare som inte är så vana vid terminal-kommando.

list
Listar alla byggnader om ingen speciell byggnad är vald med verbet [select]. Är en byggnad vald, listar [list] alla rum samt rummens konstverk i den valda byggnaden.
list [rumsnamn]
Om du befinner dig i en byggnad som du tidigare valt med [select], så kan du med [list] i kombination med [rumsnamn] lista bara konstverken i det angivna rummet

add
I kombination med [byggnadsnamn]/[rumsnamn] lägger till byggnad eller rum, beroende på vart användaren befinner sig i hierarkin.
add -art [rumsnamn/titel/beskrivning/upphovsmakare]
Med argumentet **-art** i kombination med namnet på ett rum, konstverkstitel, beskrivning och upphovsmakare, kan du lägga till ett konstverk i det angivna rummet. Argumentet är valt med tanke på vad som ska läggas till. Lägg till (add) konstverk (-art) i rummet (rumsnamn) samt konstverkets egenskaper, ger ett logiskt kommando.

delete
I kombination med [byggnadsnamn]/[rumsnamn] tar bort byggnad eller rum, beroende på vart användaren befinner sig i hierarkin.

delete -art [rumsnamn/titel]

Med argumentet **-art** i kombination med namnet på ett rum samt konstverkstitelkan du ta bort ett konstverk i det angivna rummet. Argumentet är valt med tanke på vad som ska tas bort. Ta bort (delete) konstverk (-art) i rummet (rumsnamn) samt konstverkets titel, ger ett logiskt kommando.

help

Visar hjälp för kommandon i applikationen, vart du än befinner dig i hierarkin.

preset

Lägger automatiskt till en uppsättning av byggnader, rum och konstverk. Endast för att kunna testa applikationen. Kvarlämnad som hjälp vid test och bedömning.

Seperation

I programstrukturen används System.Console-klassen endast i min UI-klass ViritualMuseum. UI-klassen kommunicerar med övriga klasser genom metoder som har skapats i de andra klasserna. Metoder som inte innehåller UI-specifika metoder som t.ex. System.Console-klassen.
Metoderna i de andra klasserna utför uppgifter som anropas från UI-klassen. Ibland returnerar metoden ett svar och ibland bara utför en uppgift utan gensvar. Det gör att skulle man vilja ändra UI, så behöver inte klasserna skrivas om.
Jag har dock metoder i några av klasserna som returnerar ett kommunikationssvar genom en text-sträng. Metoden i sig kan t.ex. kolla om ett visst objekt finns i en lista och returnera ett textsvar. När jag funderat lite mer över detta, så anser jag att det är bättre att UI-klassen även bestämmer om svaret ska vara en text eller något annat. Skulle man ändra UI, och svaret kanske ska vara en blinkande röd lampa, är det bättre med ett bool-svar, än en textsträng. Visst går det i UI:t jämföra en textsträng med en annan textsträng och få en lampa att blinka, men jag anser att även i detta fallet så ska UI-specifika meddelanden ligga i UI-klassen och kommunikationen hållas till t.ex. bool-svar.
Jag har några av dessa metoder kvar i min programkod, men skulle jag förbättra koden ytterligare, så hade jag gjort om dessa metoder och lagt all textkommunikation i UI-klassen.

Testning

Jag har tre enhetstester i applikationen:

[NoMoreArtCanBeAddedInARoom()] kollar så att det endast går att ha tre konstverk i ett rum. I detta test skapas en ny BuildingCollection och i den en byggnad. I byggnaden skapas ett rum, och i rummet försöker testet sedan lägga till fyra konstverk. För varje tillägg av ett konstverk så returnerar metoden en textsträng som bekräftar att konstverket lagts till, om konstverkstiteln redan finns i listan eller om rummet redan nått sin maxgräns av konstverk. När det fjärde konstverket läggs till så sparas textmeddelandet i en variabel, som sedan jämförs med den förväntade textsträngen.

För att också dubbelkolla att det ändå inte finns fler än tre konstverk i rummet, kollas också längden på listan med konstverk i det specifika rummet med det förväntade antalet konstverk i listan.

[NotAbleToEraseRoomWithArt()] kollar så att det inte går att ta bort ett rum som innehåller konstverk. I detta test skapas en ny BuildingCollection och i den en byggnad. I byggnaden skapas ett rum samt ett konstverk i rummet. Metoden i programkoden, som sedan ska ta bort ett rum ur byggnadens rum-lista anropas. Även denna metod returnerar meddelande beroende på om rummet innehåller konst eller inte. Det returnerade svaret sparas i en variabel och jämförs med det förväntade svaret. Även här görs en dubbelkoll mot längden på rummets lista på konstverk och förväntat antal, så att konsten inte ändå togs bort.

[CountAndCompareBulidingsRoomsAndArt()] kollar så att det går att lägga till flera byggnader, med olika rum samt olika konstverk i rummen. I detta test skapas en ny BuildingCollection och i den skapas två byggnader, två rum i ena byggnaden och ett i andra. I de olika rummen skapas flera olika konstverk samt olika antal konstverk i de olika rummen. Efter det så skapas variabler med den förväntade mängden byggnader, mängden olika rum i de olika byggnaderna, samt mängden konstverk i de olika rummen för varje byggnad.
Sedan med de metoder som finns i program-klasserna, kollas mängden objekt i de olika listorna. Listornas antal objekt sparas i variabler som jämförs med den förväntade mängden. Eftersom det skapas en ny lista för rum i varje byggnad, och en ny lista för konstverk i varje rum och metoderna hämtar listorna beroende på byggnadsnamn och rumsnamn, så visar testet att det funkar att skapa flera byggnader med olika uppsättningar rum och konstverk.

Testerna som är skapade är inte fristående utan använder befintliga klasser och metoder i programkoden. Det gör att de både visar att den verkliga programkoden funkar samt att om den skulle ändras på fel sätt, d.v.s. så att de uppställda kraven inte längre stämmer, så kommer testerna visa att något är fel i koden. Det är alltså inte bara tester som kan användas vid ett tillfälle, utan kan användas för felsökning även vid utveckling av koden.

En natt på museet

- class Program
- class VirtualMuseum - UI klass
  - new BuildingCollection
- class BuildingCollection
  - Fields
    - List <Building> buildingList = List <Building> ();
  - Method
    - CheckIfBuildingInList
    - AddBuilding
    - GetBuildingInBuildingList
    - GetIndexForBuilding
    - SetAllBuildingsToNotActive
    - ListAllBuildings
    - ListAllRoomsInABuilding
    - GetBuildingInBuildingList (overload)
- class Building
  - Fields
    - List <Room> roomList = List <Room> ();
    - string buildningName
    - enterBuilding
  - Method
    - GetBuildingName
    - AddRoomInBuilding
    - DeleteRoomInBuliding
    - GetRoomInRoomList (overload)
- class Room
  - Fields
    - string roomName
    - int maxAmountArtWorks
    - List <Art> artList = List <Art> ();
  - Methods
    - AddArtInRoom
    - GetRoomName
    - GetRoomList
    - MaxAmountOfArtworkReached
    - RoomContainArt
    - ArtWithTheSameName
- class Art
  - Fields
    - string artTitle
    - string artDescription
    - string artAuthor
  - Methods
    - GetArtTitle
    - override ToString

Enhetstest: NoMoreArtCanBeAddedInARoom

Enhetstest: NotAbleToEraseRoomWithArt

Enhetstest: CountAndCompareBulidingsRoomsAndArt