Reflektion – En natt på museet – Johannes Jakobsson

Kommandon

Select - är mitt första verb i switchen. Eftersom det första man gör bör vara att välja vilket museum man ska arbeta med. Efter verbet select behöver man lägga till ett argument, det ska innehålla ett namn på ett befintligt museum i programmet. Man skriver alltså mu select [museum-namn]. Med ett namn som argument blir det tydligt vilket val man vill göra.

Show - är nästa verb. De olika argumenten för show är museums, rooms och room. Argumentet museums visar helt enkelt alla museum som finns tillgängliga och vilket som du har "selectat". Argumentet rooms visar alla rum för det museum du "besöker". Argumentet room har ytterligare ett argument där du får skriva in ett namn för ett redan befintligt rum i museet, då skrivs all information om konstverken som befinner sig i rummet ut på skärmen. Argumenten är tydligt valda i plural och singular för att göra det så tydligt som möjligt vad som ska skrivas ut och visas för användaren.

Add – Om användaren vill lägga till museum, rum eller konstverk så behöver man vända sig till det här verbet. Kort, koncist och tydligt. Argumenten som finns är just museum, room och art. Med argumentet museum behöver du lägga till ytterligare ett argument, det är helt enkelt önskat namn på det nya museet. Dock har jag inte lyckats listat ut hur man bör jobba med mellanslag i *command line arguments,* därför är det endast tillgängligt att skriva ett namn utan mellanslag. Samma gäller för argumentet room, nästkommande argument man behöver är önskat rumsnamn. Det nya rummet hamnar i museet där man befinner sig nu. Sist men inte minst argumentet art, här behöver du också lägga till ytterligare ett argument och det är i vilket rum du vill lägga till konstverket i. När du sedan klickar enter så får du fylla i tre frågor som tyvärr innefattar Console.ReadLine då jag återigen inte hittade någon lösning för mellanslagsfunktioner och det krävs när man ska lägga till mycket information om ett konstverk. Även här är argumenten tydligt valda för vad det är man vill lägga till.

Delete – Om användaren vill ta bort ett rum eller konstverk är det här rätt verb. Argumenten är room och art. Till argumentet room behöver man ett till argument som ska innehålla rumsnamnet på det rum man vill ta bort, rummet måste dock vara tomt för att man ska kunna ta bort det. Efter man har skrivit argumentet art så behöver man lägga till vilket rum konstverket befinner sig i, men även lägga till på vilken plats konstverket ligger, exempel: [mu delete art konstverksrummet 2], det här tar bort konstverket som ligger på plats två i konstverksrummet. Det enda argumentet som jag är aningen missnöjd med skulle väl vara att man behöver välja platsen på konstverket man ska ta bort istället för exempelvis ett id, en förkortning av konstverket eller liknande. Men därav finns det ju mer utvecklingspotential att jobba mer med exempelvis dictionaries.

Help – Det här verbet ger användaren en närmare förklaring av vilka kommandon som finns och innefattar inget ytterligare argument.

Seperation

Jag har valt att enbart ha "console-kommandon" i VirtualMuseum.cs för att separera det från alla modellklasser. Istället för att ha "console-kommandon" i modellklasserna så har jag skapat metoder i respektive klass som returnerar olika former av strings. För att få det mer överskådligt och tydligt i VirtualMuseum.cs har jag byggt in de flesta "console-kommandona" och "funktionaliten" i olika metoder som kallas på beroende av användarens val i simulatorn.

Testning

Jag har gjort 10 tester i den här uppgiften, vissa mer relevanta än andra. När jag bygger ett projekt så har jag börjat utgå från testet, bygger “arrangen, acten och asserten”, i acten skriver jag en metod som inte finns, men med ett relevant namn som bör göra det jag vill göra. När det är gjort så skapar jag metoden för att testet ska gå igenom och bli “grönt”. Därför finns ett gäng fler tester än vad uppgiften kräver. Men jag tänker att jag beskriver relevansen av de testerna som krävdes i uppgiftsbeskrivningen.

CheckIfRoomIsFull är en metod för att göra det lätt att styra så att användaren inte kan lägga till fler konstverk än vad som är möjligt för ett rum. Med CheckIfRoomIsFull kan man enkelt med hjälp av en if-sats styra så att det inte går att lägga till fler konstverk om ett rum är fullt.

ShouldNotBeAbleToDeleteRoomWithArtInIt är ett test skapat för att se att en exception throwas när man försöker ta bort ett rum som fortfarande innehåller ett konstverk. Metoden man kontrollerar är egentligen DeleteContent, om den fungerar som den ska så ska den skicka ett felmeddelande om konst finns kvar i rummet och förklara att det inte går att ta bort rummet. Här ser vi tydligt att det inte går att ta bort ett rum.

AddMuseumToDictionary är ett test som visar att man kan skapa nya museum och lägga till dessa i en dictionary via AddContent metoden. Är väldigt relevant för att möjliggöra att kunna skapa nya museum eller byggnader om man väljer att kalla det så. Tillsammans med alla övriga test som är presenterade i enhetstestet tycker jag att det utgör en tydlig bild av att det går att skapa nya museum, lägga till & ta bort olika rum och olika konstverk med hjälp av de klasser och metoder som finns i detta program.

Egen kommentar

Jag har gjort en “överflödig” klass som heter TestMuseum som innehåller ett enklare exempel av ett museum. Syftet med den är enbart att hjälpa dig vid rättning så du slipper skapa egna museum och rum från start. Är det fel tänkt så kan du ju bara tänka att den inte finns där, alternativt ta bort den!

Såhär i efterhand skulle man ha byggt programmet med fler dictionaries istället för listor. Hade varit lättare att komma åt delar på ett annat sätt känns det som och slippa lite av foreach-loopandet. Är även medveten om att jag hade kunnat bygga in mer av foreach-looparna i de specifika klasserna som metoder istället för att lägga dessa i VirtualMuseum.cs.