Relationer mellan objekt

Delat aggregaat, shared aggregation

Hämta klasserna Account och ListOfAccounts från kurssidan. ListOfAccounts är en klass som lagrar kontoobjekt.
Lägg till följande funktionalitet i klassen ListOfAccounts

- En metod, `getNoOfAccounts`, som returnerar antalet objekt som för tillfället finns i listan
- En metod, `getAccount`, som tar ett index som argument och returnerar objektet på denna position, utan att ta bort det från listan. Om index är felaktigt kan du returnera null (eller använda assert).
- En metod, `remove`, som givet ett index tar bort, och returnerar, objektetet från listan. Denna metod ska se till att listan inte blir fragmenterad när objektet tas bort och att antalet objekt i listan uppdateras.

Skriv sedan ett main där du skapar ett ListOfAccounts-objekt, lägger till kontobjekt och testar de nyametoderna i klassen.

Komposition, composite aggregation

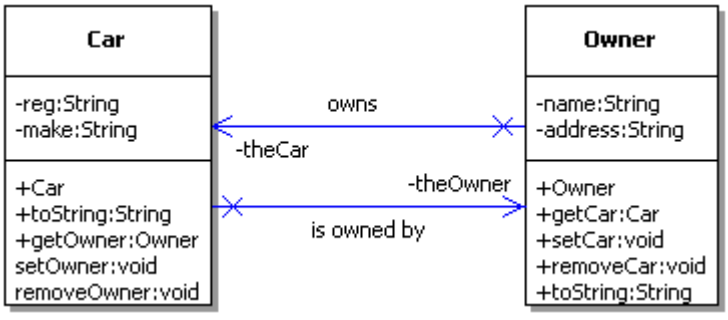
a) Skriv en enkel klass Time som representerar ett klockslag med h, m och s. Det ska finnas en metod för att ändra tiden, `setTime(int h, int m, int s)`.
Skriv sedan en klass Flight som representerar en avgång på en flygplats. Datamedlemmar är, destination, fligthnr och departureTime (avgångstid), det senare ett Time-objekt. Det ska finnas en metod, getDeparture, som returnerar avgångstiden som ett objekt.

b) Ändra klassen Flight så att

- konstruktorn skapar det inkapslade objektet departureTime
- metoden getDeparture så att den returnerar en kopia av tidobjektet istället för en referens (så att det inte går att ändra i den interna avgångstiden via den referens som returneras).

Dubbelriktad association

Mellan en bil och dess ägare råder en dubbelriktad association, se klassdiagrammet nedan.



Konstruera klassen Car som beskriver en bil. En bil har ett regnr och en tillverkare samt *känner till* sin ägare, Owner (när den väl sålts).
Skriv också klassen Owner, som kan äga en eller ingen bil.

I klassen Owner finns de publika metoderna setCar och removeCar, som ser till att associationen skapas (eller bryts*) i båda riktningarna*. När en ägare kopplas till en bil ska alltså bilen automatiskt kopplas till ägaren samt motsvarande när ägaren gör sig av med bilen.
Det är lämpligt att skriva metoder i klassen Car för kopplingen bil -> ägare som anropas av motsvarande metoder i ägarklassen. Låt dessa metoder ha synligheten ”package” i bilklassen så att klasser utanför paketet med klasserna bil och ägare inte kan använda dessa metoder.
Att ge synligheten package innebär att metoden (eller datat) är tillgängligt för alla andra klasser i samma paket.
I java fås synligheten package genom att man varken anger private eller public.

Det är lämpligt att skapa ett separat paket, carpark, där du placerar klasserna Car och Owner. I Eclipse gör du detta genom att välja File, New, package. När du sedan skapar klasserna Car och Owner anger du att de ska tillhöra paketet carpark (de ska då vara märkta med package carpark; på första raden).
Main-klassen läggs utanför paketet carpark (i defaultpaketet). På så sätt blir package-metoderna inte tillgängliga i main, endast de publika. Skriv main och skapa en bil samt en ägare, och associera dessa med varandra.