|  |
| --- |
|  |

Övningsuppgifter

Övning 1

Lägg till ett nytt projekt som heter bibblan i er solution.

Här är UML:en för projektet.



Konstruktorn till Bok skall ta alla medlemsvariablerna som inparametrar.

SkrivUt skall skriva ut värdet på medlemsvariablerna på skärmen samt vad de är (tex författare: …).

Metoderna CompareTo och Equals tar object (dvs klassen Object som alla klasser ärver av i grund och botten) som inparameter men detta kan vi inte använda utan måste typkonvertera detta till Bok klassen, detta görs väldigt enkelt:

Bok b = (Bok)o;

När man implementerar CompareTo så brukar det vara god praxis att samtidigt överlagra (override) Equals som vi ”ärver” ifrån object, så att den ger motsvarande resultat som CompareTo när vi jämför två ”likadana” Bok-objekt. Samma gäller metoden GetHashCode, som skall ge ett ”unikt” värde för att kunna lägga objektet på ”rätt” plats i en hashtabell. Eftersom vi vill att ISBN skall vara unikt och string är redan implementerat i IComparable så kan vi enkelt utnyttja det i vår egen CompareTo.

CompareTo kommer att returnera resultatet av ett anrop till CompareTo:

return this.ISBN.CompareTo(b.ISBN);

Vilket är ungefär “jämför mig med det där objektet på denna punkten”.

Ni skall få Equals och GetHashCode av mig:

public override bool Equals(object o)

{

Bok b = (Bok)o;

Return this.ISBN.Equals(b.ISBN);

}

public override int GetHashCode(object o)

{

Return ISBN.GetHashCode();

}

Biblioteksklassen har en menyfunktion (static void) som skall ha följande alternativ:

1 Ny Bok

9 Avsluta

21 Lista böcker (ArrayList)

22 Sök bok (ArrayList)

23 Sortera böcker efter ISBN

24 Sortera böcker efter författare/titel

31 Lista böcker (Hashtable)

32 Sök bok (Hashtable)

Lämpligt är att sätta rubrik för de olika talgrupperingarna (ental, tjugotal och trettiotal) tex: Meny, ArrayList och Hashtable.

Biblioteksklassen har två static variabler:

En som heter bokList, och en som heter bokHash, bokList skall vara av typen ArrayList och bokHash skall vara av typen HashTable.

Vi kommer att lägga in alla instanser av bok objekten i båda samlingarna så att vi kan se om det blir någon skillnad mellan Hashtable eller ArrayList.

Sedan skall vi ha en Main function (i bibliteksklassen)

Den börjar med att anropa Meny()

Sedan i en evig loop bör vi be användaren om dess menyval och ta emot detta i en lokal variabel som vi omvandlar från en sträng till en int.

Gör en switch sats som tar detta menyval (int)

Case 1 anropar NyBok()

Case 9 skall bara ”göra” return. Vilket i praktiken gör att programmet avslutas.

Case 21 anropar ListaList()

Case 22 skall be användaren mata in ett ISBN, som vi tar emot, skapar ett Bok objekt som tilldelas resultatet från ett anrop till funktionen SökList(med isbn strängen som användaren matade in som parameter).

Sedan gör vi en koll så att vårt bok objekt inte är null (dvs att vi fått en träff)

Skriver ut resultatet.

Case 23 anropar SortListISBN()

Case 24 anropar SortListNamn()

Case 31 anropar ListaHash()

Case 32 Gör som case 22 men anropar SökHash(isbn), kontrollera så att inte bok objektet är null, om inte skriv ut resultatet.

Funktionen NyBok skall be användaren mata in alla nödvändiga parametrar som vi behöver för att skapa en ny bok.

Därefter skall vi lägga till vårt nya bokobjekt till bokList och till bokHash med ISBN som nyckel.

Funktionen ListaList()

Skall loopa igenom varje förekomst av objekt i bokList samlingen och skriva ut detta.

SökList funktionen är static och skall returnera ett bok objekt, ta en sträng (ISBN) som inparameter och vi skall återigen gå igenom hela bokList samlingen och för varje förekomst av objekt skall vi kolla om detta objektets ISBN är lika med (Equals) isbn strängen som är inparameter om så är fallet returnerar vi detta objekt annars returnerar vi null.

Funktionen SortListISBN är static och använder sig av den inbyggda sort funktionen i bokList samlingen.

SortListNamn är static och skall anropa Sort funktionen på bokList samlingen, inparameter till Sort är en ny instans av BokComparator().

Funktionen ListaHash() är static och skall skapa en instans av ICollection som tilldelas värdet av bokHash ICollection e = bokHash.Values;

Och för varje bok objekt i samlingen e skall vi skriva ut.

SökHash funktionen är static och tar en sträng som inparameter (ISBN för denna skall ju vara unik) och returnerar det itemet (som egentligen är ett bok objekt) som finns på plats [isbn] i samlingen bokHash, returvärdet måste typkonverteras till ett bok objekt.

Till sist kommer klassen BokComparator som ärver IComparer.

Har bara en metod:

Compare(tar två objekt som parametrar), dessa objekt måste typkonverteras till Bok objekt.

Skapa en sträng som tilldelas värdet av författare och titel ifrån objekt 1,

Gör sedan en strän 2 som får likadant ifrån objekt 2 . Sedan returnerar man jämförelsen av sträng 1 med sträng 2 (dvs funktionen skall returnera en int).

Övning 2

I Main skapa en lista av typen int, därefter lägg till 3 olika värden (ett och ett). 5,15,25

Lägg till 6 st olika värden på en gång. (1,2,3,4,5,6)

Skapa en for loop som loopar igenom och skriver ut index och värdet i listan på plats [index].

Använd en av de inbyggda funktionerna till att kontrollera om det finns ett värde i listan (välj ett värde som finns), därefter anropa samma funktion och leta efter ett värde som inte finns.

Detta görs lättast synligt genom att anropa Console.WriteLine(”någon lämplig text” + lista.Contains(värde));

Ta bort den första förekomsten av ett värde som du har minst 2 av

Ta bort värdet på index plats 3

Skriv en loop som skriver ut värdena i listan (det finns en variant på loop där man inte kan manipulera värdena i listan utan bara titta på dem och läsa dem, använd denna loop typ).

Övning 3

Förbättra ovanstående program med fyra nya funktions anrop av de inbyggda funktionerna som man kan använda genom List <T> klassen och ta med det i utskrifter och visa mig.

Övning 4

Ni skall skapa motsvarande ifrån övning 2 och 3 men med Dictionary<key,value> där nyckel skall vara string (namnet på en student) och value är int (betyget på studenten).

Övning 5

Vi skall skapa ett program som samlar 1000 tärningsslag och skriver ut resultatet av detta i ett konsol fönster. Använd er av en samlingsklass lämpligtvis en dictionary<int, int> och använd er av Random klassen, ni bör också lägga in en kontroll så att ni inte lägger in samma tärningsslag flera gånger utan bara ökar värdet (förekomsten av det).

Slumpningen bör göras så att min värdet är ett och maxvärdet är sju, dvs den skall inte gå över sju utan vara i detta intervall.

Övning 6

Gör om övning 2 men fixa så att den blir sorterad.

Övning 7

Nu skall ni använda Queue klassen skapa en kö som innehåller studenter i en klass.

Lämpligt är att återanvända person och student klasserna. Skapa upp några student objekt och köa dessa objekt, därefter loopa igenom kön och skriv ut informationen som finns där. Sedan skall ni skapa en ny loop där ni skriver ut till konsolen att studenten på första platsen har klarat av utbildningen och sedan skall ni ”av köa” studenten.

Övning 8

För att få ett underlag att söka eller sortera på gör följande metod:



Skapa en klass med en funktion som gör en brute force sökning. Funktionen tar en samling av element och en variabel av typen int som inparameter. Samlingen kan vara en lista av typen int, tänk på att ni inte får använda de inbyggda funktionerna till hjälp typ Sort() funktionen.

Resultatet skall visas upp för mig.

Övning 9

Nu skall du utöka den klass du skapade i förra övningen med ytterligare en funktion. En funktion som söker enligt algoritmen för binär sökning. Även detta resultat skall visas upp för mig. En förutsättning för att detta skall fungera är att listan är sorterad. För att kunna testa denna måste vi sortera listan först, här får vi använda den inbyggda sorterings funktion eftersom vi inte har gjort någon egen ännu.

Den skall en lista av typen int och en variabel av typen int (målvärdet, det som vi skall söka efter) som inparametrar

Övning 10

Ny ska du få utöka din klass ytterligare vi börjar med en sk Swap funktion som skall vara en public void och ta tre parametrar(lista av typen int, int a och int b).



Sedan skall vi skapa funktionen public void MyBubbleSort().

Som inparameter skall den ta en lista av typen int.

I denna funktion deklarerar du två variabler:

* En bool som heter osorterad som du sätter till true.
* En int som heter end och tilldelas mängden – 1 av den lokala listan som skapas inne i denna funktion.

Sedan skapar du en loop som skall iterera så länge som variabeln osorterad är true (dvs sann).

Inne i loopen börjar du med att sätta variabeln osorterad till false, sedan skapar du en for loop med villkoren j = 0, j < end, j++.

Du gör en kontroll om värdet i listan på plats j är större än värdet i listan på plats j+1, om så är fallet anropar du din Swap metod med parametrarna (listan, j, j + 1) sätt sedan osorterad till true.

Efter for loopen (dvs när den är slut) räknar du ner end variabeln med ett (end--).

Övning 11

Vi använder oss av vår ”gamla” klass igen och utökar den med ytterligare en public void funktion som heter ExchangeSort().

Vi skall återigen ha en lista av typen int som inparameter.

Den skall innehålla två loopar (nästlade), innan den innersta loopen börjar skall vi deklarera en variabel av typen int som vi tilldelar indexet ifrån vår yttersta loop.

När vi skapar den andra loopen så skall vi inte sätta indexet till noll utan det sätts till (indexet ifrån den yttersta loopen + 1).

Inne i den innersta loopen gör vi en kontroll om värdet i vår lista på plats minst är större än det värdet på plats ifrån indexet ifrån den innersta loopen, om så är fallet så skall variabeln minst tilldelas värdet på indexet ifrån den innersta loopen.

Efter den innersta loopen så gör vi en kontroll till där vi kollar om indexet ifrån den yttersta loopen är mindre än värdet på variabeln minst, om så är fallet anropar vi vår funktion Swap(den lokala list variabeln, variabeln minst, och indexet ifrån den yttersta loopen (för den innersta har vi gått ur och når inte längre dess variabler som fanns där)).