Inlämningsuppgift 3

Uppgift 1

Skriv en sökningsalgoritm som kombinerar linjär sökning och binär sökning på följande sätt:

-Anta att arrayen är sorterad

-Arrayen är av element av typen Comparable

-Om arrayen har mindre än 20 element skall linjär sökning genomföras annars binär sökning.

-Algoritmen skall returnera platsen där elementen hittas annars skall algoritmen returnera -1.

1)Vad blir tidskomplexiteten för algoritmen? Resonera och motivera!

2) Skriv JUnit test för algoritmen som bevisar att algoritmen är korrekt.

Uppgift 2

Dynamiska datastrukturer är mycket viktiga. En typ av dynamisk datastruktur är länkade listor.

Det går inte att förstå länkade listor utan att själv skriva en. Här skall du få chansen

I kurslitteratur, på föreläsningen samt på nätet finns mycket information om länkade listor. Se till att du förberedder dig innan du börjar med uppgiften,
http://www.mycstutorials.com/articles/data_structures/linkedlists

Ladda ner klasserna **ListNode** och **SimpleLinkedList** som vi har pratat om på föreläsningen. Försök förstå hur ”länkningen” mellan noder implementeras.

I klassen **SimpleLinkedList** finns några metoder som inte är färdiga. Implementera dessa enligt beskrivningen som ni hittar i klassen. Se till att du testar dessa metoder så att du säkert vet att de fungerar.

Det kan uppfattas något "krångligt" att använda ”länkar” men dessa används vid skapandet av dynamiska datastrukturer som är mycket viktigt oavsett vilket programmeringsspråk man använder.

Uppgift 3

RFID-tekniken (Radio Frequency ID) används i allt fler applikationer, från passersystem och biltullstranspondrar till varuetiketter.

Kommunikationen mellan RFID tagen (passiv teknik) och RFID läsaren, sker med hjälp av induktion.

När taggen kommer inom läsarens fält laddas taggen så att den kan sända informationen (transponderns unika ID-kod för read-only chip) till läsaren.

Till skillnad mot streckkoden behöver taggen inte vara synlig från sändare/mottagare enheten, men den får heller inte vara för långt borta. Dagens läsavstånd är mellan två centimeter och tio meter beroende på antennans storlek hos läsaren.

Tekniken passar mycket väll för ett modern lager. Tänkt till exempel en byggfirma där verktyg lånas ut och lämnas tillbaka. Det försvinner mycket tid på att ta reda på om ett visst verktyg finns eller inte finns i lagret.

Ett sjukhus har också mycket medicinsk utrusning som "vandrar" mellan olika avdelningar och man vill veta om utrustningen är utlånat eller inte samt eventuell vem som har lånat den senast. Inte minst för att medicinsk utruskning kan sprida sjukdomar mellan olika avdelningar. Då vill man kunna spåra utlåningen.

Du behöver inte veta mycket mer om RFID. I vår uppgift förenklar vi scenariot på följande sätt.

Ladda ner paket med klasser **Labb3.** Observera att några klasser är redan delvis implementerade.

En vara definieras i klassen Item (se koden i filen Item.java). Ett Item-objekt består av namn, RIFD nummer samt ett datum stämpel när vara har blivit utlånad.

I klassen Item finns också en tom main metod så att du kan testa klassen Item. Skapa två Item- objekt. Skriv ut objekten. Reflektera.

Ett lager definieras av en dynamisk datastruktur, i detta fall av en länkad lista. Det speciella med länkade listor är att datan i listan "läggs" i så kallade noder som sedan länkas till varandra.

En sådan nod hittar du färdigimplementerad i klassen **Node.java.**

En Nod består av ett Item-objekt samt länkar till nästa Nod.

Själva länkade listan definieras i klassen ItemList som är från början en "tom" Nod. Allt eftersom "Items" lånas ut eller lämnas tillbaka skall länkade listan uppdateras genom att ta bort eller lägga till objekten till listan.

a) För att dessa operationer skall kunna utföras måste du först implementera metoderna från klassen ItemList.java enligt beskrivningen som du finner framför varje metod.

Testning av ItemList klassen

I den givna koden finns metodstumpar för metoderna du skall implementera. Utveckla och testa metoderna. Ha som exempel testfilen från förra labben.

Läs om hur man använder JUnit test i JGrasp om du inte har redan gjort i förra labben.

http://www.cs.washington.edu/education/courses/cse143/11wi/handouts/junit-jgrasp.pdf

eller Eclips

http://help.eclipse.org/helios/index.jsp?topic=%2Forg.eclipse.jdt.doc.user%2FgettingStarted%2Fqs-junit.htm

b) Skriv ett program (i klassen LagerProgram finns början) där du simulerar det tidigare beskrivna ”lagermjukvara”. Du kan välja fritt om du skall har ett lager för en byggfirma eller sjukhus.

Se filen *Medisinskutr* och *Verktyg* som innehåller några namn på verktyg. Du skall läsa objekt från en fil när du sätter upp ditt lager.

Nedanstående funktioner är obligatoriska. Utöver får du själv lägga till andra.

Använd en meny, liknade den som du fick i Inlämningsuppgift 1.

- Skapa lager lista, och utlåninglista dvs. två ItemList- objekt.

- Fyll lagret med varor. Läs data som gäller för dina Item- objekt från en av filerna du har fått. Komplettera gärna filerna med flera varor. Ett enkelt sätt att läsa från fil är med hjälp av Scanner. Exempel hittar du som kommentar sist i filen LagerProgram.java.

- Låna ut vara (eftersom vi inte har riktiga RFID kort och kortläsare, matar du in RFID numret från tangentbordet). När vara lånas ut, tas den från inStore (lager)listan, datum stämplas (setDeliverDate(Date d) och sedan läggs i outStrore (utlåning) listan.

- Lämna tillbaka varan. Då läggs tillbaka varan i inStore samt sätt "delivered" variabeln till null.

- Sök efter en viss vara i inStore (använd varans namn för sökning). Skriv ut hur många objekt finns i lagret. Tänkt på effektiv sökning.

- Skriv ut alla utlånade varor

- Skriv ut alla varor som finns i lagret

Lägg till programmet andra funktioner om du vill.

c) Som ni vet, blir data mycket mer överskådlig om den är sorterad och sökningen mycket mer effektiv om man kan tillämpa binär sökning på data vi undersöker.

Komplettera klassen ItemList med en metod addSort() som lägger Item-objekten i listan i sorterad ordning utifrån objektens namnn.

Innan du gör det skall du låta klassen Item implementera interfacen **Comparable.**

Gör om ditt program så att den använder addSort() istället för sort.
Ändra ditt program så att när programmet stängs ner skall all data sparar till fil.

Extra:

1. Implementera 2-3 separata klasser som implementerar **Comparator**<Item> (liknande det du gjorde för klassen Land). Detta gör att du skulle kunna skriva ut filer sorterade utifrån olika data, RIFD nummer och Item- namn eller Date om du önskar.
2. Ändra i programmet så att du kan skapa ”loggfiler” för dina objekt. Men hjälp av dessa loggfiler skulle du kunna se hur objekten hade lånats, mm. Kanske en separat fil för varje objekt

Frågor att besvara:

1. I java biblioteket finns metoden Collections.binarySearch().Vad är tidskomplexiteten för metoden? Vilken typ av datastruktur kan sökas igenom med metoden. Vad måste ändras i klasen ItemList så att den kan sökas igenom med Collections.binarySearch() metoden?

2. I vilka fall föredrar du länkade listor istället för array baserade listor?