Algoritmer och datastrukturer
DB4002
1 April 2025

HÖGSKOLAN I HALMSTAD

Uppgiftspaket 1

Allmänna instruktioner: Hur och när man lämnar in och vad som ska lämnas in förklaras i inlämningssidan på Blackboard. Denna dokument beskriver uppgifterna.

För varje uppgift och i visa fall deluppgift finns en *.java* fil. Vad filen heter anges i respektive uppgift / deluppgift.

OBS! Du behöver använda flera av de klasser som finns i Blackboard under modulerna för veckorna 1 och 2. Det som finns programmerad i dessa klasser ska du inte programmera om, du ska bara använda klasserna. För att göra det behöver du ladda ner klasserna till den mapp där du skriver dina program för inlämning.

Denna information finns som en del av kursbeskrivningen som finns i Blackboard:

> Alla uppgiftspaketen är uppdelade i två delar benämnda Del 1 och Del 2. Del 1 göres för att få godkänt på omgången, medan Del 2 göres för att få bonuspoäng för högre betyg (4, 5) på tentans andra del (se informationen nedan om tentans struktur). Alltså, Del 1 måste göras för att få godkänt, medan Del 2 endast göres för att samla bonuspoäng för högre betyg på tentan.
>
> Det totala antalet bonuspoäng man kan erhålla genom att lösa uppgifterna på Del 2 för respektive uppgiftspaket är 10. Alltså, man kan samla ihop hälften av poängen för högre betyg på tentan genom att lösa dessa Del 2-uppgifter. Bonuspoängen man får med sig till tentan är lika med summan av bonuspoängen på man har erhållit från lösningarna av Del 2-uppgifterna.

Inlämningsuppgifter - del 1

1. Denna uppgift handlar om att implementera en datatyp för *studenter* som kan användas i studieadministrativa sammanhang. I det sammanhanget har en student ett namn (för- och efternamn), ett personnummer och ett antal högskolepoäng hen har avklarat.

   Man vill bland annat kunna hantera listor med studenter (till exempel från ett program eller kurs) som man sorterar efter namn. Men man kan även vilja sortera studenter efter avklarade högskolepoäng, till exempel när det ska avgöras vilka studenter har tillräcklig med poäng för att flyttas till nästa årskurs eller för att kunna bli registrerade på examensarbetet.

   (a) **(4 poäng)** I filen *`Student.java`* finns början på en implementation av en datatyp för studenter som du ska komplettera. Filen kompilerar men metoden `main` som testar din implementation ger för tillfället fel utskrift

   ```
   [Student]/Randomsdotter, Student, pn: 0101019999, credits: 0.0/ equals
   [Student]/Randomsdotter, Student, pn: 0101019999, credits: 0.0/ is false

   [Student]/Studentdotter, She, pn: 0202028888, credits: 0.0/ equals
   [Student]/Studentsson, He, pn: 0302017777, credits: 0.0/ is false

   Array with students:
   [Student]/Randomsdotter, Student, pn: 0101019999, credits: 0.0/
   [Student]/Studentdotter, She, pn: 0202028888, credits: 0.0/
   [Student]/Studentsson, He, pn: 0302017777, credits: 0.0/
   [Student]/Randomsson, Student, pn: 0302016666, credits: 0.0/

   Sorted array:
   [Student]/Randomsdotter, Student, pn: 0101019999, credits: 0.0/
   [Student]/Studentdotter, She, pn: 0202028888, credits: 0.0/
   [Student]/Studentsson, He, pn: 0302017777, credits: 0.0/
   [Student]/Randomsson, Student, pn: 0302016666, credits: 0.0/

   Added credits:
   [Student]/Randomsdotter, Student, pn: 0101019999, credits: 0.0/
   [Student]/Studentdotter, She, pn: 0202028888, credits: 0.0/
   [Student]/Studentsson, He, pn: 0302017777, credits: 0.0/
   [Student]/Randomsson, Student, pn: 0302016666, credits: 0.0/
   ```

   Det du behöver göra är att programmera metoderna `addCredits`**,** `credits`**,** `equals` **och** `compareTo`. Den sistnämnda metoden behövs för att kompilator ska acceptera att `Student implements Comparable<Student>`.

   Klassens API beskriver vad dessa metoder ska göra:

   ```
   public class Student implements Comparable<Student>
   ---------------------------------------------------------------------
       // Konstrueraren skapar en student med givna
       //  personnummer, förnamn och efternamn.
       // Antal credits sätts till 0.
       // (OBS: redan implementerad)
       public Student(String pn, String fn, String ln)

       // Metoden toString beräknar en text med all information om studenten.
       // Texten ser ut:
       // [Student]/lastname, firstname, pn: pnumber, credits: acquired/
       // (OBS: redan implementerad)
       public String toString()
   ```

```
    // Metoden equals beräknar det booleska värdet av att jämföra för likhet
    // *denna* student (med instansvariablerna) med studenten *that* som är
    //  metodens argument.
    // Två studenter är lika om de har samma namn, samma efternamn och samma
    // personnummer.
    public boolean equals(Student that)

    // Metoden compareTo beräknar ett heltal som ska vara:
    // -1 om denna student är mindre än studenten that
    //  0 om denna student är lika (enligt equals) med studenten that
    //  1 om denna student är större än studenten that
    // Ordningen bestäms av lexikografisk ordning av
    // efternamn, namn och personnummer:
    // Bara om två studenter har samma efternamn tittar man på namn
    // och bara om två studenter har samma efternamn och namn tittar
    // man på personnummer  för att bestämma ordningen.
    public int compareTo(Student that)

    // Metoden addCredits används för att lägga till högskolepoäng
    // som studenten har klarat.
    // Värdet c används för att öka värdet i instansvariabeln credits.
    public void addCredits(double c)

    // Metoden credits returnerar värdet i variabeln credits.
    public double credits()
```

En korrekt implementation producerar följande utskrift (du ska inte ändra i `main` metoden. Du ska bara implementera metoderna som används på ett korrekt sätt.):

```
java Student
[Student]/Randomsdotter, Student, pn: 0101019999, credits: 0.0/ equals
[Student]/Randomsdotter, Student, pn: 0101019999, credits: 0.0/ is true

[Student]/Studentdotter, She, pn: 0202028888, credits: 0.0/ equals
[Student]/Studentsson, He, pn: 0302017777, credits: 0.0/ is false

Array with students:
[Student]/Randomsdotter, Student, pn: 0101019999, credits: 0.0/
[Student]/Studentdotter, She, pn: 0202028888, credits: 0.0/
[Student]/Studentsson, He, pn: 0302017777, credits: 0.0/
[Student]/Randomsson, Student, pn: 0302016666, credits: 0.0/

Sorted array:
[Student]/Randomsdotter, Student, pn: 0101019999, credits: 0.0/
[Student]/Randomsson, Student, pn: 0302016666, credits: 0.0/
[Student]/Studentdotter, She, pn: 0202028888, credits: 0.0/
[Student]/Studentsson, He, pn: 0302017777, credits: 0.0/

Added credits:
[Student]/Randomsdotter, Student, pn: 0101019999, credits: 7.5/
[Student]/Randomsson, Student, pn: 0302016666, credits: 3.0/
[Student]/Studentdotter, She, pn: 0202028888, credits: 15.0/
[Student]/Studentsson, He, pn: 0302017777, credits: 4.5/
```

(b) **(1 poäng)** I filen *StudentCreditsComparator.java* ska du programmera en `Comparator` som jämför studenter med hänsyn till hur många högskolepoäng de har: studenter som har färre `credits` är mindre än studenter med fler `credits`, studenter med samma antal `credits` jämförs som vanliga studenter, enligt efternamn, namn och personnummer.
En korrekt implementation producerar följande utskrift (du ska inte ändra i `main` metoden!):

```
java StudentCreditsComparator
Array with students:
[Student]/Randomsdotter, Student, pn: 0101019999, credits: 7.5/
[Student]/Studentdotter, She, pn: 0202028888, credits: 15.0/
[Student]/Studentsson, He, pn: 0302017777, credits: 4.5/
[Student]/Randomsson, Student, pn: 0302016666, credits: 3.0/

Sorted array:
[Student]/Randomsson, Student, pn: 0302016666, credits: 3.0/
[Student]/Studentsson, He, pn: 0302017777, credits: 4.5/
[Student]/Randomsdotter, Student, pn: 0101019999, credits: 7.5/
[Student]/Studentdotter, She, pn: 0202028888, credits: 15.0/
```

(c) **(1 poäng)** I filen *StudentAgeComparator.java* ska du programmera en `Comparator` som jämför studenter med hänsyn till personnummer: studenter med högre personnummer är mindre (yngre) än studenter med lägre personnummer.
En korrekt implementation producerar följande utskrift. I detta fall måste du komplettera `main` metoden för att sortera och skriva ut det sorterade fältet.

```
java StudentAgeComparator
Array with students:
[Student]/Randomsdotter, Student, pn: 0101019999, credits: 7.5/
[Student]/Studentdotter, She, pn: 0202028888, credits: 15.0/
[Student]/Studentsson, He, pn: 0302017777, credits: 4.5/
[Student]/Randomsson, Student, pn: 0302016666, credits: 3.0/

Sorted array:
[Student]/Studentsson, He, pn: 0302017777, credits: 4.5/
[Student]/Randomsson, Student, pn: 0302016666, credits: 3.0/
[Student]/Studentdotter, She, pn: 0202028888, credits: 15.0/
[Student]/Randomsdotter, Student, pn: 0101019999, credits: 7.5/
```

Tips: Det finns ingen metod i klassen `Student` för att ta reda på en students personnummer. Du ska inte ändra i klassen `Student`. Tänk på att metoden `toString` producerar en text som innehåller personnummer: använd det för att ta reda på en students personnummer.

(d) **(1 poäng)** Skriv ett program `SortStudents` som läser in studenter från standard input och genererar en sorterad utskrift. Programmet ska vara en klass `SortStudents` i en fil *SortStudents.java*.
Antal studenter som skal läsas in anges som den första kommandoradsargument. Sorteringen kan göras enligt namn, ålder eller avklarade högskolepoäng. Vilket beror på vad som anges som den andra kommandoradsargument: *name*, *age* eller *credits*.
En student läses in som ett personnummer, ett efternamn, ett förnamn och ett antal högskolepoäng.
Här ser du utskrifter som ska genereras när man dirigerar om standard input till filen *few-students.txt* som innehåller

```
0101019999 Randomsdotter Student 7.5
0202028888 Studentdotter She 15
0302017777 Studentsson He 4.5
0302016666 Randomsson Student 3.0
```

```
java SortStudents 4 name < few-students.txt
[Student]/Randomsdotter, Student, pn: 0101019999, credits: 7.5/
[Student]/Randomsson, Student, pn: 0302016666, credits: 3.0/
[Student]/Studentdotter, She, pn: 0202028888, credits: 15.0/
[Student]/Studentsson, He, pn: 0302017777, credits: 4.5/
```

```
java SortStudents 4 age < few-students.txt
[Student]/Studentsson, He, pn: 0302017777, credits: 4.5/
[Student]/Randomsson, Student, pn: 0302016666, credits: 3.0/
[Student]/Studentdotter, She, pn: 0202028888, credits: 15.0/
[Student]/Randomsdotter, Student, pn: 0101019999, credits: 7.5/
```

```
java SortStudents 4 credits < few-students.txt
[Student]/Randomsson, Student, pn: 0302016666, credits: 3.0/
[Student]/Studentsson, He, pn: 0302017777, credits: 4.5/
[Student]/Randomsdotter, Student, pn: 0101019999, credits: 7.55/
[Student]/Studentdotter, She, pn: 0202028888, credits: 15.0/
```

2. Denna uppgift handlar om att implementera en datatyp som är ganska likt ett fält (array). Den stora skillnaden är att man inte behöver bestämma storleken, istället bestäms storleken av antalet element som lagras i den. Denna datatyp är en så kallad `datastruktur`: datatypen motsvarar inte något som en färg, en bild, komplexa tal, studenter eller andra typer av objekt man kan vilja använda i ett program. Istället används datatypen för att lagra data (så som arrayer gör!).

   Denna datastruktur brukar kallas för *extendable* (eller *extensible* eller *dynamic*) *array*: man kan tänka på extendable arrays som fält som går att förlänga under programmets exekvering. De program som använder denna datastruktur är program som behöver använda fält men där man inte vet innan programmet körs hur många platser kommer att behövas (typisk många element som läses från en fil eller program där man behöver ändra antal element under exekvering).

   I filen *`Xarray.java`* finns början till en implementation av en datatyp för *extendable arrays*: fält som går att förlänga under programmets exekvering.

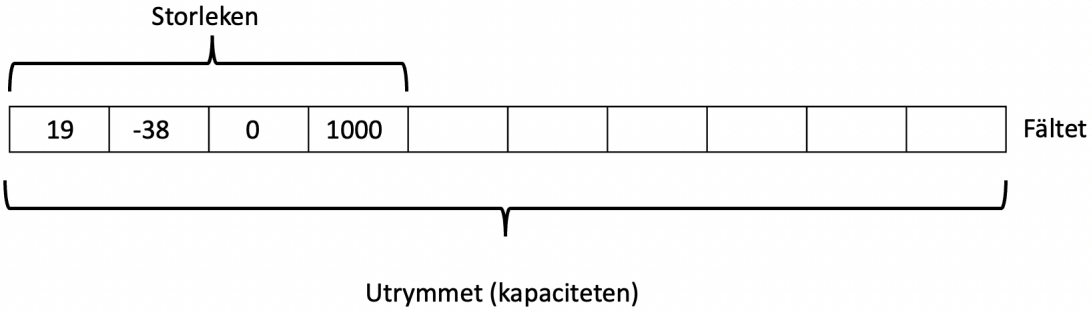
   Datatypens API är följande (mer utförliga förklaringar följer i deluppgifterna):

```
public class Xarray<T>
-----------------------------------------------------------------------
    // Konstrueraren skapar ett fält av T med 10 platser
    // för att kunna lagra elementen.
    // Antalet element i fältet är 0, men har utrymme för att
    // lagra 10 element.
    public Xarray()

    // Metoden add lägger till värdet x på första lediga platsen i fältet.
    // Om det inte finns tillräckligt med plats
    // skapas ett större fält och befintliga element kopieras ditt.
    public void add(T x)

    // Metoden lookup returnerar värdet i position i i fältet
    public T lookup(int i) throws IndexOutOfBoundsException

    // Metoden update ändrar värdet i position i i fältet till x
    public void update(int i, T x) throws IndexOutOfBoundsException

     // Metoden size returnerar antal element i fältet (storleken)
     public int size()
```

   Du ska programmera klassen `Xarray`. För att göra det ska du:

   (a) **(1 poäng)** Använda tre instansvariabler:
   - en variabel av typ `T[]` (alltså ett ordinarie fält) för att lagra elementen i `Xarray`,
   - en variabel av typ `int` för att hålla reda på fältets utrymme (längden på första variabel) och
   - en variabel av typ `int` för att hålla reda på hur många element som finns lagrade i fältet. Detta är storleken på `Xarray`.

   Bilden visar hur vi tolkar dessa tre variabler:



(b) **(1 poäng)** Programmera konstrueraren så att när ett klientprogram skapar en instans med `new Xarray()` initieras instansvariablerna till ett fält med 10 platser, storleken till 0 och kapaciteten till 10.

(c) **(1 poäng)** Programmera metoden `add` för att lägga till ett element. Elementet ska placeras på första lediga platsen i fältet. Första lediga platsen kan man räkna ut med hjälp av instansvariabeln som håller reda på storleken. När man lägger till ett element måste man ändra storleken (den ökar med ett).

Om metoden anropas då storleken är samma som kapaciteten (det finns ingen ledig plats för det nya elementet) ska implementationen

- skapa ett fält med dubbelt så många platser,
- kopiera alla värden i instansvariabeln (fältet) till det nya fältet,
- uppdatera instansvariabeln (fältet) till det nya längre fält,
- uppdatera instansvariabeln för kapaciteten,
- lägga till det nya elementet på första lediga platsen och
- uppdatera instansvariabeln för storleken.

(d) **(1 poäng)** Programmera metoden `lookup` så att den returnerar det värdet som finns på den plats som anges i argumentet. Om argumentet inte är en position där det finns ett element (mindre än 0 eller större eller lika med storleken) ska metoden kasta ett `IndexOutOfBoundsException`.

(e) **(1 poäng)** Programmera metoden `update` så att värdet i positionen som anges får det värdet som anges. Om positionen inte är en position där det finns ett element (mindre än 0 eller större eller lika med storleken) ska metoden kasta ett `IndexOutOfBoundsException`.

(f) **(0 poäng)** Programmera metoden `size` som returnerar antalet element i fältet.

(g) **(1 poäng)** Skriv ett program `StudentXarray` som läser in studenter från standard input, skapar ett `Xarray` med dessa studenter och skriver ut studenterna sorterade på avklarade högskolepoäng. Programmet ska vara en klass som heter `StudentXarray` i en fil *`StudentXarray.java`*. Programmet ska inte använda kommandoradsargument eller annan input för att ta reda på hur många studenter som ska läsas in.
Du kan testa programmet med filen *`many-students.txt`* där du kan dirigera programmets standard input. **OBS!** För att kunna sortera med `Arrays.sort` och `StudentCreditsComparator` kommer du behöva skapa ett Java fält med alla element i den `Xarray` du skapade med alla inmatade studenter.

Inlämningsuppgifter - del 2

(2 poäng) Komplettera metoden `main` i klassen `SandC.java` som läser in studenter och anonymiseringskoder från standard input och kan producera två utskrifter:

1. antingen en utskrift som är sorterad enligt studenternas efternamn
2. eller en utskrift som är sorterad enligt anonymiserings koderna

Programmet ska använda ett kommandoradsargument som bestämmer hur utskriften ska vara sorterad: antingen *code* eller *student*

Studenter och koder läses in som fyra `String`: personnummer, efternamn, förnamn och kod.

Klassen för par `Pair` finns i Blackboard, med materialet för vecka 1 i kursen.

Du ska definiera en `Comparator` för klassen `Pair` där ordningen bestäms i första hand av andra komponenten i paren. Då kan ni använda `GenericSorting.insertionSort` för att sortera enligt första komponenten och `Arrays.sort` med den nya `Comparator` för att sortera enligt den andra komponenten.

Ni kan testa programmet genom att dirigera om standard input till filen *SandC.txt* som har 100 studenter med anonymiserings koder. Kontrollera att utdatan är sorterad som det önskas enligt kommandoradsargumentet.