Instruktioner för distanstenta

Anvisningar:

Alla lösningar bör skrivas efter varandra i ***en*** textfil (.txt).

Filen måste börja med studentens namn och en kort fritext där studenten intygar att alla lösningar i filen är skrivna på egenhand utan hjälp från andra och utan kopering från internet.

Varje svar måste börja med ***en tydlig rubrik*** i textfilen, t.ex.:

```
UPPGIFT 3
=========

Del a)

...

Del b)

...


UPPGIFT 4
=========

...
```

Man får ta kod från föreläsningspresentationerna, sina egna labblösningar och man får söka information om Javas API med Google.

Man får inte kommunicera med andra studenter eller på annat sätt be om lösningar. Man får inte söka lösningar på internet.

Lösningar bör skickas in som en textfil (dvs en fil som slutar med .txt) via Canvas Assignments innan tentan tar slut, dvs innan klockan 12:30.

Lycka till!

Kontakt:

Vid frågor går det att få tag på examinatorn Magnus Myréen med att ringa 0729744943 under tentans gång.

Data och Informationsteknik / Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
Ansvarig lärare: Magnus Myréen

DAT050 Objektorienterad programmering

Tentamen, torsdag, 2020-10-29, 08:30-12:30

Vitsordsgränser: 3=24p, 4=36p, 5=48p, max 60p.

Kom ihåg att inte fastna på en uppgift. Bestäm i förväg din egen tidsgräns per uppgift. ***Lycka till!***

Uppgift 1: [10 poäng totalt] *att skriva klass, equals, interface*

Denna uppgift handlar om att skriva en klass `Nat` för naturliga tal, dvs heltal som är icke-negativa: 0, 1, 2, 3, 4, osv. Koden ska vara skriven så att:

- Instaner av klassen är *immutable*.
- Värdet lagras som en instansvariabel av typen `int`.

Klassen ska ha följande konstruktor:

- En konstrukor som skapar en `Nat` instans på basis av en given `int` parameter. Ifall det givna värdet är negativt ska ett exception kastas. [2 poäng]

Klassen ska ha följande metoder:

- En instansmetod för addition som adderar det nuvarande naturliga talet med ett givet naturligt tal. Kasta en *exception* ifall talet inte längre ryms i en `int`. [2 poäng]

  Tips: Om man adderar två positiva `int` tal och resultatet inte längre ryms i en `int`, då är resultatet av additionen ett negativt `int` värde, t.ex. följande kod skriver ut: `:-)`

  ```
  int i = 2000000000;
  if (i + i < 0) { System.out.println(":-)"); }
  ```

- En liknande instansmetod för subtraktion. Skriv koden så att subtraktion stannar vid noll, dvs m – n = 0 om n > m. Exempel: 5 – 10 = 0. [2 poäng]
- En `equals`-metod som följer standard Java riklinjer för `equals`. [2 poäng]
- En `toString` metod. [1 poäng] ***Tips:*** `String.valueOf`
- Se till att klassen implementerar `java.lang.Comparable<Nat>`. [1 poäng]

Uppgift 2: [6 poäng totalt] *testning, exceptions*

Denna uppgift handlar om att skriva kod som testar följande metod:

```
/** Returnerar alltid en ny int array med slumpat innehåll.
    Kastar aldrig en exception. */
public int[] createRandomArray() { ... }
```

Din uppgift är att skriva testkod som kollar (så gott det går) att koden som är gömd i `...` ovan implementerar specifkiationen som står skrivet i Javadoc kommentaren ovan.

Hur kollar man att resultatet är slumpat? För slumpningsegenskapen räcker det att man kollar om 100 anrop till `createRandomArray` ger samma resultat. Ifall 100 anrop ger samma resultat, då kan testkoden dra slutsatsen att `createRandomArray` inte slumpar.

Obs: Testkoden får inte kasta exception oberoende om `createRandomArray` gör det eller ej.

Uppgift 3: [15 poäng totalt] *användning av standard klasser*

Denna uppgift handlar om att skriva kod som använder följande klasser.

```
/** Data om ett nätverk av landsvägar */
public class RoadMap {
  ...
  public Set<RoadSegment> getRoadsFrom(Place p) { ... }
}

/** En enkelriktad vägstump som har en längd och en ändpunkt */
public class RoadSegment {
  ...
  public double getLength() { ... }   // alltid > 0.0
  public Place getEndPoint() { ... }
}

/** En klass för platser */
public class Place {
  ...
  public boolean equals(Object obj) { ... }
}
```

`Set<E>` interfacet från Javas API (nedan) representerar en matematisk mängd av element av typen `E`. Man kan skapa en tom mängd av typen `Set<E>` med `new HashSet<E>()`.

```
public interface Set<E> {

  /** Lägger till e till mängden */
  public boolean add(E e);

  /** Kollar om Objektet o redan finns i mängden */
  public boolean contains(Object o);

  /** Returnerar en Iterator som räknar upp mängdens element */
  public Iterator<E> iterator();

  ...
}
```

Obs: Du kan anta att det inte finns `null` någonstans och att alla vägar har längd > 0.

Uppgiften handlar om att skriva nya instansmetoder för `RoadMap` klassen:

a) Skriv en metod `distance` som, givet en `Place from` och en `Place to`, returnerar distansen av en väg som *leder direkt* från plasten `from` till platsen `to`. Ifall det finns flera direkta vägar, får koden välja vilken som helst. Ett exception ska kastas om det inte går att köra *direkt* från plasten `from` till platsen `to`. [4 poäng]

b) Skriv en metod `distance` som givet en `path` array (av typen `Place[]`) räknar ut vad distansen är om man kör vägar som förbinder platserna enligt `path`, dvs om man kör från `path[i]` direkt till `path[i+1]` för alla värden av `i`. Ett exception ska kastas om det inte går att köra enligt `path` eller om `path` är tom. [4 poäng]

c) Skriv en metod `reachableWithin` som givet en `Place p` och en distans `double d` returnerar en mängd `Set<Place>` som består av alla platser man kan nå inom distans `d` om man börjar köra från plats `p`. [7 poäng]

   Tips: Tänk rekursivt! ”Vart kan jag komma inom 500 m från A?” Om jag kan köra från A till B på 100 m, då frågar jag mig: ”Vart kan jag komma inom 400 m från B?”

   Obs: Din kod för del c) behöver inte vara snabb, men koden måste vara lättläst och returnera korrekt resultat.

   Obs: Du får gärna definiera egna privata hjälpmetoder.

Uppgift 4: [23 poäng totalt] *interface, kopiering, Java Generics*

Nedan är ett interface `Iter` som liknar (men inte är samma som) `java.util.Iterator`.

```
public interface Iter<A> {

   /** Returnerar true om det finns ett till element. */
   public boolean hasNext();

   /** Returnerar det nästa elementet om hasNext() ger true.
       Kan göra precis vad som helst ifall hasNext() ger falskt. */
   public A next();

   /** Returnerar en djup kopia av sig själv. */
   public Iter<A> clone();

}
```

Denna uppgift handlar om detta `Iter` interface. Vi använder `Iter` för att representera uppräkningar av värden. Uppräkningar kan vara *ändliga* eller *oändliga*.

a) Skriv kod för en klass `ThreesIter` som implementerar `Iter<Integer>` och representerar en oändliga uppräkningen av talet tre: 3, 3, 3, 3, 3, ... [2 poäng]

b) Skriv kod för en klass `OneTwoThreeIter` som implementerar `Iter<Integer>` och representerar den ändliga nummer uppräkningen: 1, 2, 3. [2 poäng]

c) Skriv en klass `EmptyIter` som implementerar `Iter<A>` och representerar den tomma uppräkningen, dvs en `Iter` som inte har några element alls. [2 poäng]

d) Skriv en klass `AppendIter` som slår samman två uppräkningar representerade som `Iter<A>`. Klassen bör ha en konstruktor med följande signatur. [5 poäng]

```
public AppendIter(Iter<A> first, Iter<A> second) { ... }
```

Obs: Kom ihåg att `Iter` instanser är mutable. Var försiktig med aliasing i din kod! Tänk t.ex. att följande kod bör resultera i att värdet i variabeln `a` är en instans som representerar uppräkningen 1, 2, 3, 1, 2, 3.

```
Iter<Integer> a = new OneTwoThreeIter();
a = new AppendIter(a, a);
```

Obs: Koden i `AppendIter` klassen får inte gå in i en oändlig loop om det går att undvika.

e) Skriv en *klassmetod* `appendAll` till `AppendIter` som givet en array av typen `Iter[]` slår samman alla givna `Iter` instanser och returnerar en `Iter` som representerar den sammanslagna uppräkningen av värden. Du bör hantera varje `null` som om den representerar den tomma uppräkningen. [5 poäng]

f) Skriv kod för en klass `OnlyEvenIter` som implementerar `Iter<Integer>` och har en konstruktor med följande signatur. [7 poäng]

```
public OnlyEvenIter(Iter<Integer> it) { ... }
```

Idén är att `OnlyEvenIter` är samma som `it` men lämnar bort alla udda tal. Exempel: `new OnlyEvenIter(new OneTwoThreeIter())` producerar en siffra en gång: 2.

Obs: Din kod för `OnlyEvenIter` får inte fastna i en oändlig loop om det går att undvika.

Uppgift 5: [6 poäng totalt] *händelsehantering, arv, GUI programmering*

Denna uppgift handlar om att implementera en `JFrame` variant som heter `ShakeFrame`. `ShakeFrame` ska fungera precis som en vanlig `JFrame`, men den ska ha en extra egenskap: varje gång musen rör sig över `ShakeFrame` fönstret ska fönstret byta plats till en slumpmässig plats som är 5 pixlar i x- och y-led från koordinaterna 50, 50. Effekten av denna egenskap är att det ska se ut som om fönstret skakar när musen åker över fönstret.

Tips: Använd `MouseMotionListener`.