Laboration 5 – rekursion

Mål: Du ska träna på att skriva program med rekursiva algoritmer.

Förberedelser

F1. Läs och se filmer om rekursion i modul 10.

F2. Läs igenom texten under rubriken ”Bakgrund”.

F3. Antag att följande anrop görs: `fractalLine(4, 810, 0);`

a) Hur många gånger nås basfallet, dvs. hur många linjer ritas? **Svar:**

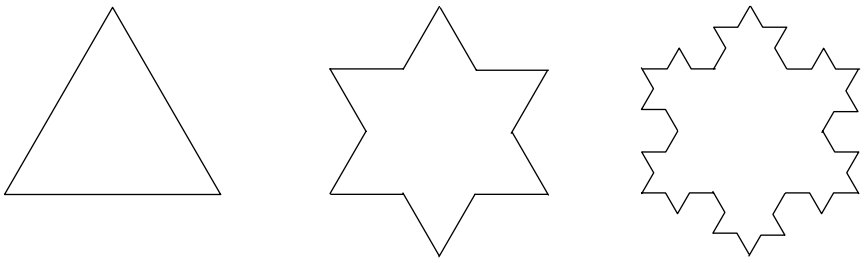
b) Hur långa är linjerna? **Svar:**

Bakgrund

Uppgiften i denna laboration är att skriva ett program som ritar fraktala figurer. Fraktal, en term som myntades av Mandelbrot 1975, är benämningen på bilder som i motsats till t.ex räta linjer, cirklar och trianglar är starkt sönderbrutna. De är uppbyggda av olika element med samma struktur.

Det finns många exempel på fraktaler i naturen såsom berglandskap, kustlinjer och virvelbilning i vattenfall. Inom matematiken använder man fraktaler för att beskriva sådana verkliga fenomen.

Studera fig 1 som visar fraktalen Kochs snöflinga (uppkallad efter den svenska matematikern Helge von Koch).



Figur 1: Kochs fraktal av ordning 0, 1 och 2.

Varje ny figur har åstadkommits genom att varje linje ersatts med en figur bestående av fyra nya linjer.

För att rita fraktalen Kochs snöflinga utgår man från en liksidig triangel. För att få en figur av ordning 1 ersätter man var och en av de tre linjerna med fyra nya linjer enligt fig 2. För att få en figur av ordning 2 ersätts varje linje i figuren av ordning 1 med fyra nya linjer osv.
En linje med längden length och riktningen alpha (vinkeln mellan linjen och x-axeln) ersätts alltså med fyra nya linjer som har följande längd och riktning:

- length/3, alpha
- length/3, alpha - 60°
- length/3, alpha + 60°
- length/3, alpha

Följande metod och tillhörande rekursiva hjälpmetod (i pseudokod) ritar Kochs snöflinga av en godtycklig ordning:

Figur 2: En linje ersätts med fyra nya linjer.

```
public void draw(int order, double length) {
    fractalLine(order, length, 0);
    fractalLine(order, length, 120);
    fractalLine(order, length, 240);
}

private void fractalLine(int order, double length, double alpha) {
    if (order == 0) {
        "rita en linje med längden length och riktningen alpha"
    } else {
        fractalLine(order-1, length/3, alpha);
        fractalLine(order-1, length/3, alpha-60);
        fractalLine(order-1, length/3, alpha+60);
        fractalLine(order-1, length/3, alpha);
    }
}
```

För att kunna rita olika fraktaler och fraktaler av olika ordning (grad av sönderbrytning) under laborationen finns det ett grafiskt användargränssnitt. Avsikten är att man som användare av det färdiga programmet ska kunna välja vilken fraktal man vill se ur en meny och kunna påverka fraktalens ordning genom att klicka på knappar. Användargränssnittet är i stora delar färdigt. Dock finns det bara en enda typ av fraktal att välja i menyn. Under laborationen kommer gränssnittet att behöva kompletteras så att man kan välja ytterligare en fraktaltyp.

D1. I projektet för laborationen finns tre paket: `fractal`, `koch` och `mountain`. I paketet `fractal` finns klasser för det grafiska användargränssnittet. Där finns bland annat en abstrakt klass `Fractal` som ska vara superklass till de egna ”fraktalklasser” du skapar. Vidare finns klassen `TurtleGraphics` med metoder för att rita linjer i användargränssnittets fönster. De övriga klasserna i paketet beskriver användargränssnittets fönster med dess meny och knappar.

Huvudprogrammet finns i klassen `FractalApplication`. Kör detta program. Då öppnas ett fönster på skärmen:

- Fönstret har en meny med namnet Fraktaler och texten ”Kochs triangel ordning 0” syns på fönstret.
- Om du öppnar menyn så syns det ett val: Kochs triangel. Om du väljer detta alternativ ur menyn så händer det ingenting. Det beror på att programmet vid detta val försöker rita Kochs fraktal av ordning 0, men denna metod gör ingenting förrän du själv kompletterat koden (uppgift D2).
- Fönstret har också knappar för att öka resp. minska den valda fraktalens ordning och rita den på nytt. Klicka på knappen pil uppåt. Texten i fönstret ändras då till ”Kochs triangel ordning 1” men fortfarande ser man ingen fraktal av de skäl som nämnts ovan.

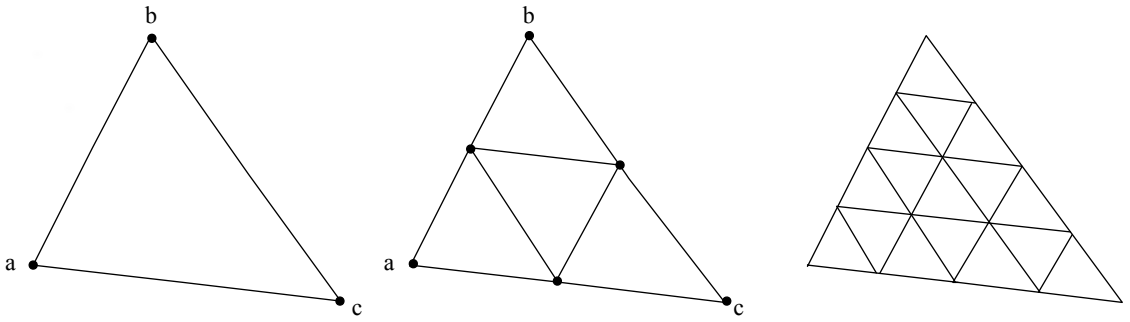
  Om du inte ser några knappar längst ned i fönstret så justera fönstrets höjd (attributet `fractalHeight` i klassen `FractalApplication`).

D2. I paketet `koch` finns en påbörjad klass `Koch` med metoder för att rita Kochs snöflinga. Fyll i de rader som saknas i metoden `fractalLine`.

Observera att koden för att rita Kochs snöflinga här i texten är pseudokod och att du i den riktiga koden även behöver ha med ett objekt av klassen `TurtleGraphics` som parameter för att rita linjer.

Kör huvudprogrammet. Nu ska du se Kochs fraktal av ordning 0 på fönstret då programmet startar och du ska kunna se samma fraktal av högre ordningar genom att använda knapparna.

D3. I denna uppgift ska du lägga till ännu en fraktal till ditt tidigare program. Denna fraktal ska åskådliggöra ett bergsmassiv. En figur av ordning 0 utgörs av en triangel (gärna något sned). För att få nästa ordning ersätts varje triangel av fyra nya trianglar enligt fig 3.



Figur 3: Bergfraktal av ordning 0, 1 och 2.

I paketet `mountain` ska du lägga till en klass (liknande `Koch` i paketet `koch`) med metoder för att rita bergsfraktalen. Lämpliga parametrar till konstruktorn kan vara de tre startpunkterna. Till din hjälp finns den färdiga klassen `Point`, som beskriver en punkt.

OBS! Bergsfraktalen ritas på liknande sätt som Kochs snöflinga, men det finns ett par viktiga skillnader. Kochs snöflinga byggs upp av tre linjer. I varje rekursiv nivå ersätts en linje av fyra nya. En linje har en längd och en riktning. Bergsfraktalen består av en *triangel. I varje rekursiv nivå ersätts en triangel av* fyra *nya trianglar. En triangel beskrivs av* tre *punkter.*

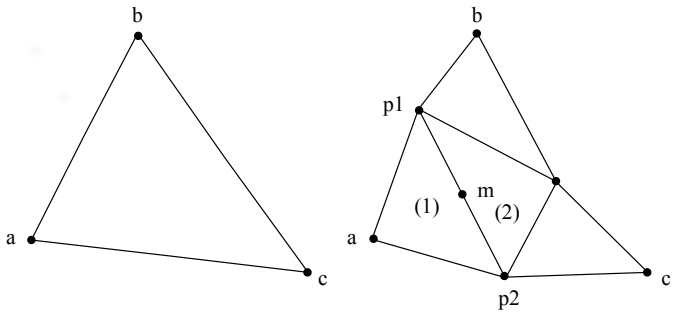
För att din nya fraktal ska synas i användargränsnittets meny och kunna ritas upp behöver du bara ändra i `main`-metoden i klassen `FractalApplication`. Öka vektorn `fractals` storlek och lägg in ett objekt av din nya fraktalklass i den. När du provkör programmet kommer du att se att det i menyn dyker upp ett alternativ till med det namn som metoden `getTitle()` i den nya fraktalklassen returnerar. Välj detta alternativ för att testa ritning av bergsmassiv.

Tips! Lägg in det nya fraktalobjektet först i vektorn så kommer det att visas när programmet startar. Du kommer att provköra det en hel del gånger.

D4. Fraktalen i föregående uppgift blir för regelbunden för att likna ett bergsmassiv. Inför uppdelningen av en triangel i fyra nya, mindre trianglar ska därför mittpunkten förskjutas i y-led. Se fig. 4.

Förskjutningens storlek bestäms av funktionen `randFunc` som ger ett slumptal enligt en viss fördelning med avvikelsen `dev`:

```
public static double randFunc(double dev) {
   double t = dev * Math.sqrt(-2 * Math.log(Math.random()));
   if (Math.random() < 0.5) {
      t = -t;
   }
   return t;
}
```



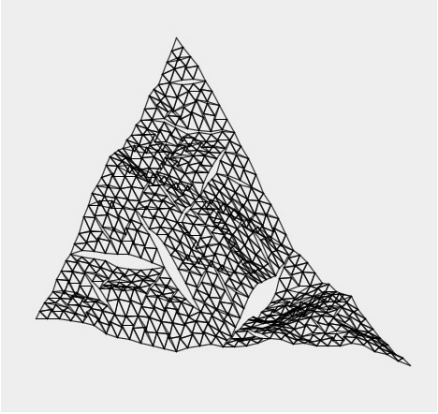
Figur 4: Bergfraktal av ordning 0 och 1. Mittpunkterna förskjuts - y-led innan en triangel delas upp i fyra nya trianglar.

För varje nivå ska parametern `dev` till `randFunc` halveras. Om man glömmer att halvera denna parameter blir figuren för taggig. Låt gärna startvärdet på `dev` vara parameter till `Mountains` kontruktor.

De förskjutna mittpunkterna kommer att tillsammans med triangelns ursprungliga hörn att utgöra hörn i de fyra nya trianglarna.

Metoden `randFunc` är färdig att använda och finns i klassen `RandomUtilities`.

Berget kommer nu att ha lite mer oregelbundna former och se mer naturligt ut. Men det kommer att finnas vita fält här och var i figuren (se fig. 5), vilket ska rättas till i nästa deluppgift.



Figur 5: Bergfraktal av ordning 5 med förskjutning av alla mittpunkter.

D5. Av den högra figuren i fig. 4 ser vi att en speciell svårighet uppstår genom att trianglarna har vissa sidor gemensamma. När triangel (1) ska delas in i fyra mindre trianglar så ska mittpunkten m förskjutas. När senare triangel (2) ska delas in får inte m förskjutas en gång till. Så här kan man göra för att klara av denna svårighet:

Implementera först en klass `Side` som håller reda på en triangelsidas ändpunkter.

Skapa i klassen `Mountain` en map av typen `HashMap<Side, Point>` där `Side`-objekt kan lagras tillsammans med sin beräknade mittpunkt.

När en mittpunkt ska beräknas söker man först i mappen efter en sida med ändpunkterna p1, p2. Om en sådan sida finns använder man den redan beräknade mittpunkten. I annat fall beräknar man mittpunkten på samma sätt som tidigare och lagrar sidan med p1, p2 i mappen tillsammans med den beräknade mittpunkten.

Eftersom en sida bara används högst två gånger (första gången då mittpunkten beräknas och andra gången då den hittas i mappen) kan man ta bort sid-mittpunktsparet från mappen när man använt den. Sökningen blir snabbare då.

I mappen används klassen `Side` som nyckel. För att det ska fungera måste man där skugga metoderna `equals` och `hashCode` i klassen `Side`. Dessa två metoder används för att hitta nyckeln i mappen. (Hur mappen är implementerad och hur dessa två metoder används kommer att behandlas senare i kursen. När du tidigare har använt klassen `HashMap` har någon av Javas standardklasser används som nyckel. I dessa klasser är redan `equals` och `hashCode` skuggade.)

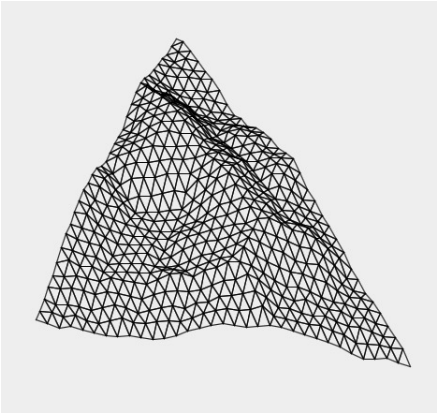
Metoden `hashCode` ska översätta `Side`-objektet till ett heltal och kan se ut så här:

```
@Override
public int hashCode() {
    return p1.hashCode() + p2.hashCode();
}
```

Inuti metoden `equals` är det sidornas ändpunkter som ska jämföras. Tänk på att man inte säkert vet ordningen på sidans ändpunkter.

Tips! `@Override` är ett direktiv till kompilatorn att kontrollera så att man skuggar en metod som verkligen finns i en superklass. Lägg till det före metoden `equals` så försäkrar du dig om att metodnamnet är rättstavat och att parametern har rätt typ.

Efter dessa ändringar ska figuren se ut liknande den i fig. 6



Figur 6: Bergfraktal av ordning 5.

D6. Fundera igenom följande, och diskutera med din handledare.

- Är det något speciellt i din kod du vill ha feedback på?