Laboration 3: Rationella tal

Syfte

Att få förståelse för de grundläggande objektorienterade begreppen.

Redovisning

Källkoden för uppgifterna skall lämnas in via Fire senast onsdag 10/10. Lägg upp källkoden direkt i Fire (dvs paketera inte filerna i arkiv (såsom zip eller tar).

Uppstart

Börja med att skapa en ny mapp Lab3. Ladda ner filen filesLab3.zip från kursens hemsida och kopiera klasserna i denna fil till mappen Lab3.

Uppgift

Skriv en klass Rational för att hantera *rationella tal*. Ett rationellt tal uttrycks som bekant som kvoten mellan två heltal. Vi skriver rationella tal på formen 1/3, 4/7, -1/5. Klassen skall skapas i mappen Lab3.

Steg 1: Metod som beräknar största gemensamma divisor

När man beskriver ett rationellt tal är det bäst att *normalisera* talet, d.v.s. att förkorta bort alla gemensamma faktorer i täljaren och nämnare. Exempelvis skall talet 15/20 representeras som 3/4. För att kunna göra denna förkortning behöver vi en metod som beräknar den *största gemensamma divisorn* (eng. greatest common divisor) till två positiva heltal m och n. Börja med att skriva en sådan metod och placera metoden i en klass med namnet Rational. Metoden skall ha signaturen

```
public static int gcd(int a, int b)
```

Observera att största gemensamma divisorn till k och 0 är k för alla $k \neq 0$, men att största gemensamma divisorn för 0 och 0 är odefinierat. Vi har alltså en partiell funktion, eftersom den inte är definierad för alla argument. Om båda parametrarna a och b till metoden gcd är 0 skall därför metoden kasta en *exception* (som förklaras mer i detalj senare i kursen) av typen ArithmeticException. Detta görs med satsen

```
throw new ArithmeticException("Rational.gcd(0,0) is undefined");
```

För att beräkna största gemensamma divisor för heltalen m och n är det lämpligt att använda *Euclides algoritm* som har följande utseende:

1. Sätt m till det största av de två talen och n till det minsta av talen.
2. Dividera m med n och beteckna resten vid divisionen med r.
3. Om r = 0 så är beräkningen klar och resultatet finns i n.
4. Sätt annars m till n och n till r. Upprepa från punkt 2.

Denna algoritm fungerar endast om talen m och n båda är >= 1. Metoden gcd skall dock acceptera både positiva och negativa argument.

Test: Testa metoden gcd med det färdiga huvudprogrammet i filen RationalTest1.java. Kompilera och kör programmet!

Steg 2: Konstruktorer och accessmetoder

I objekt av klassen Rational skall täljaren och nämnaren lagras i var sin instansvariabler av typen **int**. Täljaren och nämnaren skall alltid sakna gemensamma faktorer, d.v.s. vara normaliserade. Nämnaren måste alltid vara större än noll. Negativa rationella tal representeras således med en negativ täljare.

I detta steg skall du förse klassen Rational med följande konstruktorer och metoder:

- En konstruktor

  ```
  public Rational(int a, int b)
  ```

  med två parametrar a och b, vilka betecknar täljaren respektive nämnaren.

  Om b är lika med noll skall en *exception* av typen IllegalStateException genereras. Detta göras med satsen

  ```
  throw new IllegalStateException("Rational.Rational(int,int): zero denominator");
  ```

  Dividera annars a och b med deras största gemensamma divisor (med hjälp av metoden gcd) innan respektive instansvariabel sätts.

- En konstruktor

  public Rational()

  utan parametrar som initierar det aktuella talet till 0/1.

- En konstruktor

  public Rational(**int** a)

  med en parameter a, vilken betecknar täljaren. Det aktuella talet initieras till a/1.

  Tips: Vid närmare eftertanke inser du säkert att den andra och tredje konstruktorn är specialfall av den första. I en konstruktor kan man anropa en annan konstruktor och Java har en speciell syntax för detta. Man använder det reserverade ordet **this**. Ex.

  ```
  public Rational(int x) {
      this(x,1); // den första konstruktorn anropas
  }
  ```

  Med denna teknik kan man bygga sina konstruktorer hierarkiskt.

- En konstruktor (s.k. *kopieringskonstruktor*)

  public Rational(Rational other)

  som har ett annat rationellt tal other som parameter. Det aktuella talet initieras så att det innehåller samma täljare och nämnare som other.

- En metod

  public int getNumerator()

  som avläser och returnerar det aktuella talets täljare.

- En metod

  public int getDenominator()

  som avläser och returnerar det aktuella talets nämnare.

Test: Testa klassen Rational med det färdiga huvudprogrammet i filen RationalTest2.java. Kompilera och kör programmet!

Steg 3: Kopiering, likhet och olikhet

- En metod

  public Rational clone()

  som skapar en kopia av det aktuella talet.

- En metod

  public boolean equals(Rational other)

  som har ett annat rationellt tal other som parameter vilket jämförs med det aktuella talet. Om de två talen är lika returneras **true** annars **false**.

- En metod

  public boolean lessThan(Rational other)

  som har ett annat rationellt tal other som parameter vilket jämförs med det aktuella talet. Om det aktuella talet är mindre än other returneras **true** annars **false**

- En metod

  public boolean isZero()

  som returnerar **true** om det aktuella talet är 0, annars returneras **false**.

Test. Testa klassen Rational med det färdiga huvudprogrammet i filen RationalTest3.java. Kompilera och kör programmet!

Steg 4: Konvertering mellan rationella tal och deras textrepresentation

Klassen skall nu utökas med några metoder, vilka förenklar inläsning och utskrift av rationella tal:

- En metod

  public String toString()

  som returnerar det aktuella talet som en text med formen "a/b". Som frivillig uppgift kan metoden förbättras så att den presenterar talet med formen "k r/b", där k är kvoten av divisionen a/b och r resten. Om det är fråga om ett negativt tal skall man i detta fall presentera talet med formen "-k r/b".

- En klassmetod

  public static Rational parse(String str)

  som undersöker om parametern str innehåller en text som på ett korrekt sätt beskriver ett rationellt tal. Om så är fallet skapas ett nytt rationellt tal med det angivna värdet och en referens till talet returneras. (Jämför med metoden parseInt i standardklassen Integer.)

  Texten i parametern str får ha någon av följande fyra former, där a och b betecknar heltalskonstanter: "a/b", "–a / b", " a/-b " eller " a ". Det får alltså finnas blanktecken i strängen. Den sista formen visar att det är tillåtet att bara ange täljaren. I så fall skall nämnaren antas vara lika med 1.

  Exempel: (Rational.parse(" 2 / 6 ")).toString() returnerar strängen "1/3"

  Använd metoderna indexOf, substring och trim i standardklassen String för att beräkna delsträngarna som representerar täljaren a respektive nämnaren b. Använd sedan metoden Integer.parseInt för att avkoda täljaren respektive nämnaren.

  När Integer.parseInt som parameter får en sträng som inte kan översättas till ett heltal kastar den ett undantag av typen NumberFormatException. Vi låter detta propagera till koden som anropat parse, eftersom en komponent i talet är felformatterad och därmed hela det avsedda rationella talet . Exempelvis kommer anropet Rational.parse("tjoho/27") att kasta NumberFormatException eftersom Integer.parseInt("tjoho") gör det.

Test. Testa klassen Rational med det färdiga huvudprogrammet i filen RationalTest4.java. Kompilera och kör programmet!

Steg 5: Aritmetiska operationer

I detta steg skall du skriva några grundläggande aritmetiska operationer:

- En metod

  public Rational add(Rational other)

  som har ett annat rationellt tal other som parameter. Som resultat ges ett nytt rationell tal som innehåller summan av det aktuella talet och other.

- En metod

  public Rational sub(Rational other)

  som har ett annat rationellt tal other som parameter. Som resultat ges ett nytt rationell tal som innehåller skillnaden mellan det aktuella talet och other.

- En metod

  public Rational mul(Rational other)

  som har ett annat rationellt tal other som parameter. Som resultat ges ett nytt rationell tal som innehåller produkten av det aktuella talet och other.

- En metod

  public Rational div(Rational other)

  som har ett annat rationellt tal other som parameter. Som resultat ges ett nytt rationell tal som innehåller kvoten mellan det aktuella talet och other. Om other har värdet 0 skall ett undantag kastas

  throw new ArithmeticException("Rational.div: division by zero");

Tips. I metoderna add, sub, mul och div använder du förstås dina matematiska kunskaper. Uttrycket a/b+c/d kan t.ex. skrivas om som (ad+bc)/bd. Använd en lämplig konstruktor när du skapar det tal som skall innehålla resultatet av den matematiska operationen. Då normaliseras talet automatiskt.(Det är förstås inte tillåtet att använda flyttal vid uträkningarna!)

Test. Testa klassen Rational med det färdiga huvudprogrammet i filen RationalTest5.java. Kompilera och kör programmet!