Extra övningsuppgifter

Här finns flera övningsuppgifter för egen träning, som komplement till uppgifterna i kurskompendiet och i läroboken. Lösningsförslag finns sist.

Övning 1

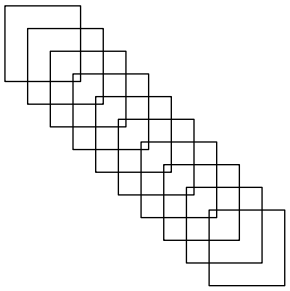
1. Klasserna `SimpleWindow` och `Square` från läroboken antas vara tillgängliga. Vad ritas på skärmen när nedanstående program exekveras? Visa med en skalenlig figur.

```
public class SquareExample {
    public static void main(String[] args) {
        SimpleWindow w = new SimpleWindow(400, 400, "Example");
        Square sq1 = new Square(100, 100, 200);
        Square sq2 = new Square(300, 300, 50);
        sq1.draw(w);
        sq2.draw(w);
    }
}
```

2. Skriv om programmet från uppgift 1 men skapa bara *ett* kvadratobjekt. Samma figur som tidigare ska ritas. Förutsätt att klassen `Square` har en operation `void setSide(int newSide)`, som ändrar sidlängden till `newSide`.

Övning 2

1. Skriv en klass med en `main`-metod som med hjälp av klasserna `SimpleWindow` och `Square` ritar följande bild på skärmen (10 bilder av en kvadrat med sidan 100, varje bild är förskjuten 30 pixlar nedåt höger):



2. Skriv ett program som i ett `SimpleWindow`-fönster ritar en liksidig triangel med sidan 100 pixlar. En av sidorna i triangeln ska vara parallell med x-axeln. Kvadratroten ur ett tal `x` får man med `Math.sqrt(x)`.

3. Nedanstående program innehåller fyra fel som upptäcks av kompilatorn. Korrigera felen.

```
public class ErrorTest {
    public static void main(Spring[] args) {
        SimpleWindow w = new SimpleWindow(600, 600);
        sq = new Square(100, 200, 50);
        while sq.getX() < 300 {
            sq.draw(w);
            sq.move(10, 10);
        }
    }
}
```

Övning 3

1. Hur kan man med operatorn `%` avgöra om heltalet `n` är udda? Hur kan man avgöra om talet slutar med en nolla?

2. Vad gör nedanstående program? Programmet innehåller ett fel och något som är fel eller åtminstone mycket tvivelaktigt. Korrigera felen.

```
public class AvgTest {
    public static void main(String[] args) {
        System.out.println("Hur många tal ska läsas?");
        Scanner scan = new Scanner(System.in);
        int n = scan.nextInt();
        System.out.println("Skriv de " + n + " talen:");
        int sum = 0;
        for (int i = 0; i < n; i++) {
            int nbr = scan.nextInt();
            sum = sum + nbr;
        }
        if (n <= 1) {
            System.out.println("För få tal!");
        } else {
            System.out.println("Resultat: " + sum / n);
        }
    }
}
```

3. Skriv ett program som beräknar följande summa:

$$\frac{1}{10\sqrt{10}} + \frac{1}{11\sqrt{11}} + \frac{1}{12\sqrt{12}} + \frac{1}{13\sqrt{13}} + \ldots$$

Man kan naturligtvis inte summera ett oändligt antal termer. I stället tar man i summan bara med termer som är större än ett tal `epsilon`, som läses in först i programmet.

4. Skriv en metod som räknar hur många primtalstvillingar (två intilliggande udda tal[1] som båda är primtal) som finns i ett intervall [a, b]. Exempel på primtalstvillingar: 2 och 3, 3 och 5, 5 och 7, 11 och 13, 17 och 19, 29 och 31, ... Använd följande färdigskrivna metod för att undersöka om ett tal är ett primtal:

```
/** Undersöker om talet nbr är ett primtal */
public static boolean isPrime(int nbr);
```

[1] Talen 2 och 3 räknas också som primtalstvillingar.

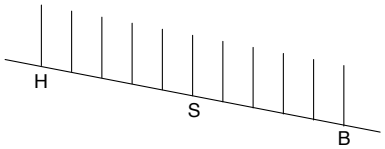
5. Skriv ett program som läser in ett x-värde och därefter beräknar e^x med 6 decimalers noggrannhet utan att utnyttja funktionen `exp` i klassen `Math`. Utnyttja i stället följande serieutveckling:

$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \cdots$$

Varje ny term ska beräknas med hjälp av den föregående termen. Till exempel beräknas $x^4/4!$ som (föregående term) $* \, x/4$, det vill säga $x^4/4! = x^3/3! * x/4$.

Övning 4

1. En man vandrar på en sluttande gata med elva jämnt utplacerade lyktstolpar:



Mannen startar sin promenad vid lyktstolpe nummer 6 (vid S i figuren) och vandrar slumpmässigt mellan lyktstolparna. Med sannolikheten 0.5 flyttar sig mannen *en* lyktstolpe *uppför* backen, med sannolikheten 0.5 *två* lyktstolpar *nedför* backen. Mannens promenad avslutas när han når till eller förbi den sista lyktstolpen i ena änden av gatan (lyktstolpe 1 eller 11, H respektive B i figuren).

Skriv ett program som simulerar ett stort antal promenader för mannen och skriver ut i hur många procent av promenaderna som mannen hamnade vid H.

2. Ett rationellt tal är ett bråktal med täljare och nämnare, till exempel 2/3 eller 54/7. Vi vill skriva ett program som kan räkna exakt med rationella tal och definierar följande klass:

```
/** Skapar ett rationellt tal med täljaren num och nämnaren denom */
Rational(int num, int denom);

/** Skriver ut talet på formen täljare/nämnare (bara täljare om
    täljaren är = 0 eller om nämnaren är = 1) */
void print();

/** Adderar det rationella talet r till detta tal */
void add(Rational r);

// ... här finns tre liknande metoder: sub, mul och div
```

Implementera klassen. Ett rationellt tal ska alltid vara förkortat så långt som möjligt; till exempel ska 4/6 representeras som 2/3, 12/144 som 1/12. När du ska utföra förkortningen får du använda följande funktion, som du inte behöver skriva:

```
/** Beräknar den största gemensamma faktorn till talen a och b.
    Till exempel är gcd(4,6)=2, gcd(12,144)=12, gcd(5,7)=1 */
static int gcd(int a, int b);
```

3. I nedanstående program utnyttjas klassen `Turtle` från datorlaboration 4. Vad ritas i programmet?

```
public class TurtleTest {
    public static void main(String[] args) {
        SimpleWindow w = new SimpleWindow(600, 600, "TurtleTest");
        Turtle t = new Turtle(w, 300, 300);
        t.right(90);
        while (t.getX() < w.getWidth()) {
            t.penDown();
            t.forward(10);
            t.penUp();
            t.forward(10);
        }
    }
}
```

4. Rita två figurer som visar variabler, objekt och attribut när satserna i nedanstående `main`-metod exekverats. Den första figuren ska visa minnessituationen när `n`-värdet 0 har lästs in, den andra ska visa situationen när `n`-värdet 2 har lästs in.

```
public class Test {
    public static void main(String[] args) {
        Scanner scan = new Scanner(System.in);
        int n = scan.nextInt();
        A pa = new A(n);
    }
}

public class A {
    private int n;
    private A next;

    public A(int n) {
        this.n = n;
        if (n > 0) {
            next = new A(n - 1);
        } else {
            next = null;
        }
    }
}
```

Övning 5

1. Ett tärningsspel går ut på att räkna antalet tärningskast som behövs för att man ska uppnå poängsumman 100. Spelaren kastar tärningen flera gånger och summerar antalet prickar som tärningen visar (summan är poängsumman). Spelaren måste hamna på exakt 100 poäng. Om spelaren till exempel har 97 poäng och slår en trea så är omgången slut; om det blir en etta eller tvåa så ökas poängsumman och spelaren fortsätter att kasta tärningen; om det blir en fyra eller femma eller sexa så fortsätter spelaren att kasta utan att öka poängsumman.

   Skriv ett program som låter en spelare genomföra 10 omgångar av spelet och skriver ut antalet kast i varje omgång. I programmet ska du utnyttja följande färdigskrivna klass, som beskriver en tärning (notera att detta inte är samma tärningsklass som beskrivs i läroboken):

`Die`

```
/** Skapar en sexsidig tärning */
Die();

/** Kastar tärningen, returnerar antalet prickar (1-6) */
int roll();
```

Uppgiften ska lösas utan någon klass `Player`. Du ska alltså bara skriva en klass med en `main`-metod.

2. Ett kortspel med två spelare går till på följande sätt: En kortlek blandas och det översta kortet tas ur leken. Den förste spelaren drar sedan kort ur leken tills han eller hon får ett kort med samma färg eller samma valör som kortet som drogs när leken hade blandats. Sedan drar den andre spelaren kort på samma sätt. Den som har dragit flest kort har vunnit spelet (det kan också bli oavgjort).

   Korten och kortleken beskrivs av klasserna `Card` och `CardDeck`. Specifikationerna av dessa klasser finns i övningsuppgift 7.6 i läroboken.

   En spelare beskrivs av följande klass:

   `Player`

```
/** Skapar en spelare med namnet name */
Player(String name);

/** Tar reda på spelarens namn */
String getName();

/** Drar kort ur leken deck tills ett kort med samma färg eller
    samma valör som firstCard påträffas, returnerar antalet
    kort som dragits */
int drawCards(CardDeck deck, Card firstCard);
```

   a) Skriv ett program där Alice och Bob spelar en omgång och namnet på vinnaren skrivs ut.
   b) Implementera klassen `Player`.

3. Förenkla följande logiska uttryck (`a` är en `int`-variabel, `ready` är en `boolean`-variabel):

```
a > 2 && a > 5
a > 2 || a > 5
! (a > 2)
! (a > 2 && a < 9)
! (a < 0 || a > 10)
ready == true
ready == false
```

4. Skriv ett program som först läser in ett `n`-värde och därefter `n` st reella tal. Därefter ska programmet skriva ut talen i omvänd ordning med ett tal per rad.

5. En vektor `v` innehåller 100 heltal. Vilken utskrift fås av nedanstående satser om a) det finns en nolla i vektorn, i vektorelementet `v[33]`, b) det inte finns någon nolla i vektorn, c) det finns 50 nollor i vektorn, i vektorelementen med udda index dvs i `v[1]`, `v[3]`, ..., `v[99]`.

```
int i = 0;
while (i < v.length && v[i] != 0) {
    i++;
}
System.out.println(i);
```

6. I följande klass beskrivs ett telefonregister i vilket man kan lagra namn och telefonnummer för ett antal personer:

```
/** Skapar ett tomt telefonregister med plats för nMax personer */
TelephoneDirectory(int nMax);

/** Lägger in en ny person med namnet name och telefonnumret
    telNbr i registret. Om det redan finns en person med
    detta namn ska ingenting inträffa. Du får förutsätta att
    det finns plats i registret */
void add(String name, String telNbr);

/** Tar bort personen med namnet name ur registret. Om det
    inte finns någon person med detta namn ska ingenting
    inträffa */
void remove(String name);

/** Tar reda på telefonnumret för personen med namnet name.
    Om det inte finns någon person med detta namn ska en
    tom sträng returneras */
String findNbr(String name);
```

a) Det är lämpligt att lagra namn och telefonnummer för en person i ett objekt av en klass `Person`. Skriv en sådan klass.

b) Implementera klassen `TelephoneDirectory` enligt följande anvisningar:

- Personerna i registret ska lagras i en vektor. Objekten ska alltid finnas i början av vektorn; det får alltså inte finnas några ”hål” i vektorn. När man lägger in en ny person ska denne läggas sist i vektorn, när man tar bort en person måste hålet som uppstår fyllas på något sätt.
- I alla tre metoderna `add`, `remove` och `findNbr` ska man leta upp en person med ett givet namn. Inför en privat metod för detta.
- När man söker efter en person med ett givet namn måste man testa om två strängar (namn) är lika. Om man har två strängar `s1` och `s2` gör man det med `s1.equals(s2)`, som ger `true` om de båda strängarna är lika långa och innehåller samma tecken, `false` annars. (`s1 == s2` fungerar inte, eftersom den konstruktionen testar om `s1` och `s2` refererar till samma objekt.)

7. Skriv ett program som läser ett positivt heltal och beräknar antalet *olika* siffror som finns i talet. Exempel: talet 8640876 innehåller 5 olika siffror.

Övning 6

1. Implementera följande metod:

```
/** Undersöker om vektorerna v1 och v2 är lika dvs om de är
    lika långa och alla v1[i] == v2[i] */
public static boolean equal(int[] v1, int[] v2);
```

   Undersök inte fler tal än nödvändigt.

2. Med ”spåret” av en kvadratisk matris menas summan av elementen i huvuddiagonalen (från övre vänstra hörnet till nedre högra hörnet). Skriv en metod med anropet `trace(a)` som beräknar spåret av `double`-matrisen `a`.

3. I ett experiment mäter man ett antal värden. Experimentet beskrivs av följande klass:

```
public class Experiment {
    private double[] values; // de uppmätta värdena
    private int n;           // antalet värden i values

    /** Skapar ett experiment med plats för max mätvärden */
    public Experiment(int max) {
        values = new double[max];
        n = 0;
    }

    /** Lagrar ett nytt mätvärde newValue */
    public void measurement(double newValue) {
        values[n] = newValue;
        n++;
    }

    /** Ritar en kurva över mätvärdena */
    public void plot() {
        ...
    }
}
```

   När man ritar upp kurvan över mätvärdena märker man att kurvan är ”taggig” på grund av slumpmässiga mätfel. Man vill därför jämna ut kurvan genom att ersätta varje mätvärde (utom det första och det sista) med medelvärdet av det aktuella mätvärdet och de båda intillliggande värdena. Skriv en metod som utför utjämningen.

4. En student vid en högskola beskrivs av följande klass (vi använder bara en av metoderna i klassen):

```
Student
/** Tar reda på antalet poäng som studenten har klarat under
    året year */
int getPoints(int year);
```

   Alla studenterna beskrivs av ett objekt av följande klass:

```
public class StudentRegister {
    private int nbrStudents;    // antalet studenter
    private Student[] students; // vektor med nbrStudents studenter

    // konstruktorer och metoder där vektorn students bildas
```

```
    /** Beräknar medelantalet poäng som studenterna klarade
        under året year */
    public double getAveragePoints(int year) {
        ...
    }

    /** Räknar hur många studenter som under året year klarade
        0-9, 10-19, ..., 50-59, >=60 poäng, skriver ut antalen */
    public void printStatistics(int year) {
        ...
    }
}
```

Implementera operationerna `getAveragePoints` och `printStatistics`.

5. En mängd är en samling av element där varje element förekommer bara en gång. De grundläggande operationerna på en mängd är att lägga in ett element i mängden, att ta bort ett element och att undersöka om ett element finns i mängden. I denna uppgift ska du implementera en klass som beskriver mängder av heltal. Alla talen ligger i intervallet [0, max). ([0, max) betyder att 0 ingår i intervallet men inte max.) Vi förutsätter att max inte är alltför stort.

   En effektiv metod att implementera en klass som beskriver en mängd med ett begränsat antal element är att använda en `boolean`-vektor med `max` element och låta varje vektorelement visa om motsvarande tal ingår i mängden eller inte. Nedanstående figur visar hur mängderna {1, 3, 4, 5, 8} och {0, 7} representeras (`max` är 10, `T` står för `true`, `F` för `false`):

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
F	T	F	T	T	T	F	F	T	F

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
T	F	F	F	F	F	F	T	F	F

Klassen har följande specifikation:

`LimitedIntSet`

```
/** Skapar en tom mängd för tal i intervallet [0,max) */
LimitedIntSet(int max);

/** Lägger in talet nbr i mängden */
void insert(int nbr);

/** Tar bort talet nbr ur mängden */
void remove(int nbr);

/** Undersöker om talet nbr finns i mängden */
boolean contains(int nbr);

/** Tar reda på antalet tal i mängden */
int size();
```

Implementera klassen. Om parametern `nbr` i ett anrop är felaktig, till exempel om `nbr` är < 0, ska `insert` och `remove` inte göra någonting, `contains` returnera `false`.

Övning 7

1. Betrakta följande klass:

```
public class RegisterExample {
    private static final int MIN = 1;
    private static final int MAX = 100;
    private int[] v; // n st tal, alla i intervallet MIN..MAX
    private int n;   // antalet tal i vektorn v

    // här finns konstruktor och metoder som lägger in tal i vektorn

    /** Skriver ut antalet tal i vektorn som är = MIN, MIN+1, ..., MAX */
    public void printFrequencies() {
        for (int nbr = MIN; nbr <= MAX; nbr++) {
            int sum = 0;
            for (int i = 0; i < n; i++) {
                if (v[i] == nbr) {
                    sum++;
                }
            }
            System.out.println("Antalet tal = " + nbr + " är " + sum);
        }
    }
}
```

Metoden `printFrequencies` fungerar visserligen, men den är *inte* bra. Varför inte? Skriv om metoden så att den blir bättre.

2. Följande klass beskriver en tärning:

```
/** Skapar en tärning */
Die();

/** Kastar tärningen, returnerar antalet prickar (1-6) */
int roll();
```

I ett statistikexperiment vill man kasta *två* tärningar och undersöka hur många gånger som summan av prickarna på tärningarna blir 2, 3, 4, . . . , 11, 12. Experimentet beskrivs av följande klass:

```
/** Skapar ett experiment där tärningarna d1 och d2 kastas */
Experiment(Die d1, Die d2);

/** Kastar tärningarna n gånger, räknar antalet gånger summan blev
    2, 3, 4, ..., 11, 12. Returnerar antalen i en vektor */
int[] makeExperiment(int n);
```

Implementera klassen.

3. I ett program har man lagrat uppgifter om ett antal personers ålder i en heltalsvektor `ages`. Man vill räkna hur många av personerna som är 0–5 år, 6–10 år, ..., 46–50 år, > 50 år. Detta har man gjort på följande sätt:

```
int[] nbr = new int[11]; // nbr[0] = antal 0-5-åringar,
                         // nbr[1] = antal 6-10-åringar,
                         // ...,
                         // nbr[9] = antal 46-50-åringar,
                         // nbr[10] = antal >50-åringar
for (int i = 0; i < ages.length; i++) {
    int age = ages[i];
    if (age < 6) {
        nbr[0]++;
    } else if (age < 11) {
        nbr[1]++;
    } else if (age < 16) {
        nbr[2]++;
    } else if (age < 21) {
        nbr[3]++;
    } else if (age < 26) {
        nbr[4]++;
    } else if (age < 31) {
        nbr[5]++;
    } else if (age < 36) {
        nbr[6]++;
    } else if (age < 41) {
        nbr[7]++;
    } else if (age < 46) {
        nbr[8]++;
    } else if (age < 51) {
        nbr[9]++;
    } else {
        nbr[10]++;
    }
}
```

Detta fungerar men det är *inte* snyggt, och det blir ännu värre om man ska ta med flera intervall. Förbättra programmet!

Övning 8

1. Ett program som hanterar utlåningen av böcker i ett bibliotek ska skrivas. Biblioteket håller reda på sina böcker, sina låntagare och de utlåningar som gjorts. Böckerna är försedda med en streckkod som identifierar boken (kallas `bookCode`). Låntagarnas lånekort är försedda med en streckkod som identifierar låntagaren (kallas `personCode`). Böckerna, låntagarna och utlåningarna beskrivs av följande klasser:

```
/** Skapar en bok med koden bookCode, titeln title och författaren
    author */
Book(String bookCode, String title, String author);

// här finns operationerna getBookCode(), getTitle() och getAuthor()
```

```
/** Skapar en person med koden personCode, namnet name och adressen
    address */
Person(String personCode, String name, String address);

// här finns operationerna getPersonCode(), getName() och getAddress()
```

```
/** Skapar ett utlåningsobjekt som beskriver att boken b lånades av
    personen p under dagen d */
Loan(Book b, Person p, Date d);

// här finns operationerna getBook(), getPerson() och getDate()
```

`Date` är en standardklass i Java som beskriver datum.

Följande klass håller ordning på biblioteket:

```
/** Skapar ett nytt (tomt) bibliotek */
Library();

/** Registrerar att boken med kod bCode lånas ut till personen
    med kod pCode (se också nedan) */
int lendBook(String bCode, String pCode);

/** Registrerar att boken med kod bCode lämnas tillbaka (se också
    nedan) */
int returnBook(String bCode);

// här finns ytterligare operationer för att till exempel lägga in
// nya böcker och nya personer. Dessa operationer ska du inte skriva.
```

Implementera klassen `Library` enligt följande anvisningar:

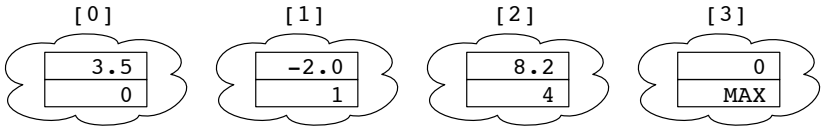
- Böckerna, låntagarna och utlåningsobjekten ska lagras i tre olika `ArrayList`-objekt. Kalla listorna `books`, `persons` respektive `loans`.
- När en bok lånas ut ska ett utlåningsobjekt skapas och läggas in i listan `loans`. Med konstruktorn `Date()` skapar man ett datumobjekt som representerar dagens datum. Metoden ska returnera `0` om boken och personen existerar, `-1` om det inte finns någon bok med kod `bCode`, `-2` om det inte finns någon person med kod `pCode`. Det kan förutsättas att boken inte redan är utlånad.
- När en bok lämnas tillbaka ska motsvarande utlåningsobjekt tas bort ur listan `loans`. Om utlåningsobjektet fanns i listan ska metoden returnera `0`, annars `-1`.

Övning 9

1. En klass som beskriver polynom i en variabel kan ha följande specifikation (bara några få av operationerna har tagits med):

```
/** Skapar ett tomt polynom */
Polynom();

/** Lägger in en ny term med koefficienten coeff och gradtalet degree
    i polynomet. Det förutsätts att det inte tidigare finns en term
    med detta gradtal */
void addTerm(double coeff, int degree);

/** Beräknar polynomets värde i punkten x */
double getValue(double x);
```

I klassen ska polynomet representeras genom att koefficient och gradtal för de termer vars koefficient inte är noll lagras i en `ArrayList`. Listan är sorterad efter växande gradtal. Efter dataelementen i listan finns en vaktpost med gradtal = `Integer.MAX_VALUE`. Till exempel representeras polynomet $3.5 - 2x + 8.2x^4$ av följande lista:



Elementen i listan är objekt av klassen `Term`:

```
/** Skapar en term med koefficienten coeff och gradtalet degree */
Term(double coeff, int degree);

/** Tar reda på koefficientvärdet */
double getCoeff();

/** Tar reda på gradtalet */
int getDegree();
```

Implementera klassen `Polynom`. Exempel: `getValue(0)` på polynomet $3.5 - 2x + 8.2x^4$ ska ge resultatet 3.5, `getValue(2.0)` ska ge resultatet 130.7 ($3.5 - 2 \cdot 2 + 8.2 \cdot 2^4$).

2. Implementera följande operation i klassen `Polynom`:

```
/** Adderar detta polynom och polynomet p, returnerar resultatet */
Polynom add(Polynom p);
```

Exempel: om `p1` $= 3.5 - 2x + 8.2x^4$ och `p2` $= 1 + x^2 - 8.2x^4 + 10x^7$ ska resultatet av `p1.add(p2)` vara ett nytt polynom $4.5 - 2x + x^2 + 10x^7$.

Ledning: gå igenom listorna ”parallellt”. Börja från början i båda listorna. Om de aktuella termerna har samma gradtal, addera koefficienterna, bilda en ny term och stega fram i båda listorna. Annars, bilda en ny term med koefficienten och gradtalet från termen med det mindre gradtalet och stega fram i den listan, och så vidare.

Övning 10

1. Betrakta följande programavsnitt:

```
int x = 100;
int y = 25;
System.out.println("Summa: " + x + y);
System.out.println("Summa: " + (x + y));
System.out.println(x + y + " är summan");
```

Vilken utskrift fås när satserna exekveras? Förklara vad som händer.

2. Antag att metoderna `equals` och `compareTo` i klassen `String` inte är tillgängliga. Skriv följande metod:

```
/** Undersöker om strängarna s1 och s2 är lika långa och alla
    tecknen i s1 är lika med motsvarande tecken i s2 */
public static boolean equals(String s1, String s2);
```

3. Skriv en metod `split` som tar en sträng och som resultat ger en `ArrayList` med varje tecken i strängen i ett `Character`-objekt. Exempel: `split("abc")` ska ge en `ArrayList` med tre element innehållande `'a'`, `'b'`, `'c'` i denna ordning.

   Skriv också en metod `join` som utför omvändningen till `split`. Metoden ska alltså ta en `ArrayList` med `Character`-objekt och bilda en sträng innehållande tecknen i listan. Om `s` är en godtycklig sträng ska det naturligtvis gälla att `join(split(s)).equals(s)`.

4. Ett visst ordbehandlingsprogram kan föreslå avstavningar av ord. I programmet utnyttjas följande funktion:

```
/** Bildar en sträng där de möjliga avstavningarna av ordet word
    är markerade med bindestreck. I vektorn pos finns positionerna
    för de bokstäver efter vilka avstavning kan göras */
public static String hyphenation(String word, int[] pos);
```

Exempel:

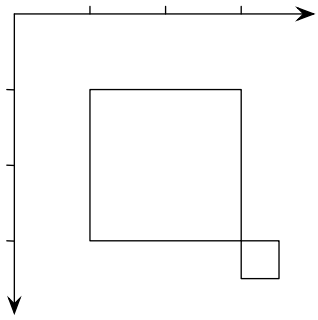
word	pos	resultat
"student"	(2)	"stu-dent"
"programmering"	(2, 6, 8)	"pro-gram-me-ring"
"sträng"	()	"sträng"

Skriv funktionen `hyphenation`. Utnyttja en strängbuffert när du bildar den nya strängen. Observera att det kan inträffa att det inte är möjligt att avstava ett ord; i så fall har vektorn `pos` längden noll.

Lösningsförslag

Övning 1

1. Följande ritas:



2.
```
public class NewSquareExample {
    public static void main(String[] args) {
        SimpleWindow w = new SimpleWindow(400, 400, "NewExample");
        Square sq1 = new Square(100, 100, 200);
        sq1.draw(w);
        sq1.move(200, 200);
        sq1.setSide(50);
        sq1.draw(w);
    }
}
```

Övning 2

1.
```
public class ManySquares {
    public static void main(String[] args) {
        SimpleWindow w = new SimpleWindow(500, 500, "ManySquares");
        Square sq = new Square(50, 50, 100);
        sq.draw(w);
        for (int i = 0; i < 9; i++) {
            sq.move(30, 30);
            sq.draw(w);
        }
    }
}
```

2.
```
public class TriangleDrawing {
    public static void main(String[] args) {
        SimpleWindow w = new SimpleWindow(600, 600, "Triangle");
        w.moveTo(100, 100);
        w.lineTo(200, 100);
        int dy = (int) Math.round(Math.sqrt(100 * 100 - 50 * 50));
        w.lineTo(150, 100 + dy);
        w.lineTo(100, 100);
    }
}
```

3.
```
public class ErrorTest {
    public static void main(String[] args) { // inte Spring
        SimpleWindow w = new SimpleWindow(600, 600,
                                          "ErrorTest"); // parameter
        Square sq = new Square(100, 200, 50); // deklaration
        while (sq.getX() < 300) { // parentes
            sq.drawSquare(w);
            sq.move(10, 10);
        }
    }
}
```

Övning 3

1.
```
if (n % 2 != 0)  { ... }
if (n % 10 == 0)  { ... }
```

2. Programmet beräknar medelvärdet av `n` inlästa tal. Testen i den sista `if`-satsen är felaktig: man kan mycket väl beräkna medelvärdet av 1 tal. Vid beräkningen av medelvärdet får man heltalsdivision, vilket man antagligen inte önskar. Den sista `if`-satsen ska ha utseendet:

```
if (n < 1) {
    System.out.println("För få tal!");
} else {
    System.out.println("Resultat: " + (double) sum / n);
}
```

3.
```
public class Sum {
    public static void main(String[] args) {
        System.out.println("Vilken noggrannhet önskas i summan?");
        Scanner scan = new Scanner(System.in);
        double epsilon = scan.nextDouble();
        double sum = 0;
        int k = 10;
        double term = 1 / (k * Math.sqrt(k));
        while (term > epsilon) {
            sum = sum + term;
            k = k + 1;
            term = 1 / (k * Math.sqrt(k));
        }
        System.out.println("Summa: " + sum);
    }
}
```

4.
```
public static int countPrimeTwins(int a, int b) {
    int nbrTwins = 0;
    if (a <= 2 && b >= 3) {
        nbrTwins = 1; // specialfall om 2,3 ingår i [a,b]
    }
    if (a % 2 == 0) {
        a++;
    }
    boolean aIsPrime = isPrime(a);
    int next = a + 2;
```

```
        while (next <= b) {
            boolean nextIsPrime = isPrime(next);
            if (aIsPrime && nextIsPrime) {
                nbrTwins++;
            }
            a = next;
            aIsPrime = nextIsPrime;
            next += 2;
        }
        return nbrTwins;
    }
```

5.
```
    public class ExpComputer {
        public static void main(String[] args) {
            double epsilon = 0.5E-6;
            System.out.println("Vilket tal ska exponentieras?");
            Scanner scan = new Scanner(System.in);
            double x = scan.nextDouble();
            double sum = 0;
            double term = 1;
            int k = 0;
            while (Math.abs(term) > epsilon) {
                sum += term;
                k++;
                term = term * x / k;
            }
            System.out.println("exp(" + x + ") är ungefär " + sum);
        }
    }
```

Övning 4

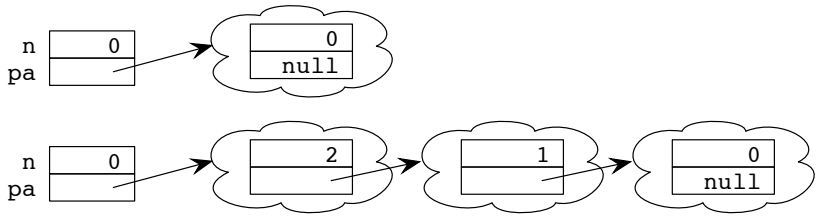
1.
```
    public class RandomWalk {
        public static void main(String[] args) {
            final int NBR_WALKS = 100000;
            final int HOME_POS = 1;
            final int AWAY_POS = 11;
            final int START_POS = (HOME_POS + AWAY_POS) / 2;
            Random rand = new Random();
            int nbrH = 0;
            for (int i = 0; i < NBR_WALKS; i++) {
                int pos = START_POS;
                while (pos > HOME_POS && pos < AWAY_POS) {
                    if (rand.nextDouble() < 0.5) {
                        pos += 2;
                    } else {
                        pos -= 1;
                    }
                }
                if (pos <= HOME_POS) {
                    nbrH++;
                }
            }
            System.out.println("Antal procent av promenaderna som " +
                               "mannen hamnade i H: " +
                               (double) nbrH / NBR_WALKS * 100);
        }
    }
```

2.
```
public class Rational {
    private int num;   // täljare
    private int denom; // nämnare

    public Rational(int num, int denom) {
        this.num = num;
        this.denom = denom;
        lowestTerms();
    }

    public void print() {
        System.out.print(num);
        if (denom != 1) {
            System.out.print("/" + denom);
        }
    }

    public void add(Rational r) {
        num = num * r.denom + denom * r.num;
        denom = denom * r.denom;
        lowestTerms();
    }

    public void sub(Rational r) {
        num = num * r.denom - denom * r.num;
        denom = denom * r.denom;
        lowestTerms();
    }

    public void mul(Rational r) {
        num = num * r.num;
        denom = denom * r.denom;
        lowestTerms();
    }

    public void div(Rational r) {
        num = num * r.denom;
        denom = denom * r.num;
        lowestTerms();
    }

    /** Förkortar talet så långt som möjligt */
    private void lowestTerms() {
        int div = gcd(num, denom);
        num /= div;
        denom /= div;
    }
}
```

I ovanstående lösning tas inte hänsyn till att nämnaren kan bli 0 (kan inträffa i konstruktorn och i operationen `div`). Om detta sker bör det rapporteras genom att ett undantagsobjekt (`ArithmeticException`) kastas.

3. En horisontell streckad linje från mitten av fönstret till fönstrets högra kant ritas (dvs från punkten 300,300 till 600,300). Både strecken och mellanrummen har längden 10 pixlar.

4. Minnessituation:



Övning 5

1.
```
public class DiceGame {
    public static void main(String[] args) {
        Die die = new Die();
        for (int i = 0; i < 10; i++) {
            int sum = 0;
            int rolls = 0;
            while (sum != 100) {
                int dots = die.roll();
                rolls++;
                if (sum + dots <= 100) {
                    sum += dots;
                }
            }
            System.out.println("Antal kast i omgång " + (i + 1) +
                               ": " + rolls);
        }
    }
}
```

2.
```
public class CardGame {
    public static void main(String[] args) {
        Player p1 = new Player("Alice");
        Player p2 = new Player("Bob");
        CardDeck deck = new CardDeck();
        deck.shuffle();
        Card firstCard = deck.getCard();
        int p1Points = p1.drawCards(deck, firstCard);
        int p2Points = p2.drawCards(deck, firstCard);
        if (p1Points > p2Points) {
            System.out.println(p1.getName() + " vann");
        } else if (p2Points > p1Points) {
            System.out.println(p2.getName() + " vann");
        } else {
            System.out.println("Det blev oavgjort");
        }
    }
}

public class Player {
    private String name;

    public Player(String name) {
        this.name = name;
    }

    public String getName() {
        return name;
    }
```

```
    public int drawCards(CardDeck deck, Card firstCard) {
        int nbrCards = 0;
        Card c;
        do {
            c = deck.getCard();
            nbrCards++;
        } while (c.getSuit() != firstCard.getSuit() &&
                 c.getRank() != firstCard.getRank());
        return nbrCards;
    }
}
```

3.
```
a > 5
a > 2
a <= 2
a <= 2 || a >= 9
a >= 0 && a <= 10
ready
! ready
```

4.
```
public class ReversePrinter {
    public static void main(String[] args) {
        System.out.println("Skriv ett n-värde och därefter n reella tal:");
        Scanner scan = new Scanner(System.in);
        int n = scan.nextInt();
        double[] nbrs = new double[n];
        for (int i = 0; i < n; i++) {
            nbrs[i] = scan.nextDouble();
        }
        for (int i = n - 1; i >= 0; i--) {
            System.out.println(nbrs[i]);
        }
    }
}
```

5.
a) 33

b) 100

c) 1

6.
```
public class Person {
    private String name;
    private String telNbr;

    /** Skapar en person med namnet name och telefonnumret telNbr */
    public Person(String name, String telNbr) {
        this.name = name;
        this.telNbr = telNbr;
    }

    /** Tar reda på namnet */
    public String getName() {
        return name;
    }

    /** Tar reda på telefonnumret */
    public String getTelNbr() {
        return telNbr;
    }
}
```

```
public class TelephoneDirectory {
    private Person[] persons;
    private int n;

    public TelephoneDirectory(int nMax) {
        persons = new Person[nMax];
        n = 0;
    }

    public void add(String name, String telNbr) {
        int pos = findPersonIndex(name);
        if (pos < 0) {
            persons[n] = new Person(name, telNbr);
            n++;
        }
    }

    public void remove(String name) {
        int pos = findPersonIndex(name);
        if (pos >= 0) {
            persons[pos] = persons[n - 1]; // fyll "hålet"
            persons[n - 1] = null;
            n--;
        }
    }

    public String findNbr(String name) {
        int pos = findPersonIndex(name);
        return (pos >= 0) ? persons[pos].getTelNbr() : "";
    }

    /** intern metod: ger index i vektorn persons för personen
        med namnet name, -1 om det inte finns någon sådan person */
    private int findPersonIndex(String name) {
        int i = 0;
        while (i < n && ! persons[i].getName().equals(name)) {
            i++;
        }
        return (i < n) ? i : -1;
    }
}
```

7.

```
public class DifferentDigits {
    public static void main(String[] args) {
        int[] count = new int[10];
        System.out.println("Skriv ett positivt heltal");
        Scanner scan = new Scanner(System.in);
        int origNbr = scan.nextInt();
        int nbr = origNbr;
        do {
            int digit = nbr % 10;
            count[digit]++;
            nbr /= 10;
        } while (nbr > 0);
        int nbrDifferent = 0;
        for (int digit = 0; digit < count.length; digit++) {
            if (count[digit] > 0) {
                nbrDifferent++;
            }
        }
        System.out.println("Talet " + origNbr + " innehåller " +
                           nbrDifferent + " olika siffror");
    }
}
```

Övning 6

1.
```
public boolean equal(int[] v1, int[] v2) {
    if (v1.length != v2.length) {
        return false;
    }
    int i = 0;
    while (i < v1.length && v1[i] == v2[i]) {
        i++;
    }
    return i >= v1.length;
}
```

2.
```
public double trace(double[][] a) {
    double sum = 0;
    for (int i = 0; i < a.length; i++) {
        sum += a[i][i];
    }
    return sum;
}
```

3.
```
public void smooth() {
    double prev = values[0];
    for (int i = 1; i < values.length - 1; i++) {
        double current = values[i];
        values[i] = (prev + current + values[i + 1]) / 3;
        prev = current;
    }
}
```

4.
```
public double getAveragePoints(int year) {
    int sum = 0;
    for (int i = 0; i < nbrStudents; i++) {
        sum += students[i].getPoints(year);
    }
    return (double) sum / nbrStudents;
}

public void printStatistics(int year) {
    int[] nbrs = new int[7];
    int index;
    for (int i = 0; i < nbrStudents; i++) {
        index = students[i].getPoints(year) / 10;
        if (index >= 7) {
            index = 6;
        }
        nbrs[index]++;
    }
    for (index = 0; index < nbrs.length; index++) {
        System.out.println(nbrs[index]);
    }
}
```

5.
```
public class LimitedIntSet {
    private boolean[] isPresent; // isPresent[nbr] är true om nbr
                                 // finns i mängden

    public LimitedIntSet(int max) {
        isPresent = new boolean[max];
    }
```

```
    public void insert(int nbr) {
        if (isInRange(nbr)) {
            isPresent[nbr] = true;
        }
    }

    public void remove(int nbr) {
        if (isInRange(nbr)) {
            isPresent[nbr] = false;
        }
    }

    public boolean contains(int nbr) {
        return isInRange(nbr) ? isPresent(nbr) : false;
    }

    public int size() {
        int nbrElems = 0;
        for (int i = 0; i < isPresent.length; i++) {
            if (isPresent[i]) {
                nbrElems++;
            }
        }
        return nbrElems;
    }

    private boolean isInRange(int nbr) {
        return nbr >= 0 && nbr < isPresent.length;
    }
}
```

Övning 7

1. I metoden undersöker man varje talvärde 100 gånger (`MAX-MIN+1` gånger). Det räcker att undersöka varje tal en gång, enligt följande (vanlig registrering):

```
public void printFrequencies() {
    int[] nbrs = new int[MAX - MIN + 1];
    for (int i = 0; i < n; i++) {
        int index = v[i] - MIN;
        nbrs[index]++;
    }
    for (int index = 0; index < nbrs.length; index++) {
        System.out.println("Antalet tal = " + (index + MIN) +
                           " är " + nbrs[index]);
    }
}
```

2.
```
public class Experiment {
    private Die die1;
    private Die die2;

    public Experiment(Die d1, Die d2) {
        this.die1 = d1;
        this.die2 = d2;
    }
```

```
        public int[] makeExperiment(int n) {
            int[] result = new int[11];
            for (int i = 0; i < n; i++) {
                int index = die1.roll() + die2.roll() - 2;
                result[index]++;
            }
            return result;
        }
    }
```

3.
```
        int[] nbr = new int[11];
        for (int i = 0; i < ages.length; i++) {
            int index;
            if (ages[i] == 0) {
                index = 0;
            } else if (ages[i] < 51) {
                index = (ages[i] - 1) / 5;
            } else {
                index = 10;
            }
            nbr[index]++;
        }
```

Övning 8

1.
```
    public class Library {
        private ArrayList<Book> books;
        private ArrayList<Person> persons;
        private ArrayList<Loan> loans;

        public Library() {
            books = new ArrayList<Book>();
            persons = new ArrayList<Person>();
            loans = new ArrayList<Loan>();
        }

        public int lendBook(String bCode, String pCode) {
            int i = 0;
            while (i < books.size() &&
                   ! books.get(i).getBookCode().equals(bCode)) {
                i++;
            }
            if (i == books.size()) {
                return -1;
            }

            int k = 0;
            while (k < persons.size() &&
                   ! persons.get(k).getPersonCode().equals(pCode)) {
                k++;
            }
            if (k == persons.size()) {
                return -2;
            }

            loans.add(new Loan(books.get(i), persons.get(k), new Date()));
            return 0;
        }
```

```
    public int returnBook(String bCode) {
        int i = 0;
        while (i < loans.size()) &&
               ! loans.get(i).getBook().getBookCode().equals(bCode)) {
            i++;
        }
        if (i == loans.size()) {
            return -1;
        }
        loans.remove(i);
        return 0;
    }
}
```

Övning 9

1.
```
class Polynom {
    private ArrayList<Term> terms;

    public Polynom() {
        terms = new ArrayList();
        terms.add(new Term(0, Integer.MAX_VALUE));
    }

    public void addTerm(double coeff, int degree) {
        if (coeff == 0) {
            return;
        }
        int i = 0;
        while (terms.get(i).getDegree() < degree) {
            i++;
        }
        terms.add(i, new Term(coeff, degree));
    }

    public double getValue(double x) {
        double result = 0;
        int i = 0;
        Term t = terms.get(i);
        while (t.getDegree() != Integer.MAX_VALUE) {
            result += t.getCoeff() * Math.pow(x, t.getDegree());
            i++;
            t = terms.get(i);
        }
        return result;
    }
}
```

2.
```
public Polynom add(Polynom p) {
    Polynom result = new Polynom();
    result.terms.remove(0); // tag bort vaktposten tillfälligt
    int i = 0; // index i this.terms
    int k = 0; // index i p.terms
    Term t1 = terms.get(i);   // aktuell term i terms
    Term t2 = p.terms.get(k); // aktuell term i p.terms
```

```
        while (t1.getDegree() != Integer.MAX_VALUE ||
               t2.getDegree() != Integer.MAX_VALUE) {
            if (t1.getDegree() == t2.getDegree()) {
                double sum = t1.getCoeff() + t2.getCoeff();
                if (sum != 0) {
                    result.terms.add(new Term(sum, t1.getDegree());
                }
                i++;
                k++;
            }
            else if (t1.getDegree() < t2.getDegree()) {
                result.terms.add(new Term(t1.getCoeff(), t1.getDegree());
                i++;
            }
            else {
                result.terms.add(new Term(t2.getCoeff(), t2.getDegree());
                k++;
            }
            t1 = terms.get(i);
            t2 = p.terms.get(k);
        }
        result.terms.add(new Term(0, Integer.MAX_VALUE));
        return result;
    }
```

Övning 10

1. Följande utskrift fås:

```
Summa: 10025
Summa: 125
125 är summan
```

Operatorer med samma prioritet utförs från vänster till höger. I det första fallet konverteras `x` till `String` och konkateneras med strängen `"Summa: "`, därefter konverteras `y` till `String` och konkateneras.

I det andra fallet står `x+y` inom parentes och heltalsadditionen utförs först. Sedan konverteras resultatet till `String` och konkateneras.

Också i det tredje fallet utförs `x+y` före konkateneringen.

2.
```
    public static boolean equals(String s1, String s2) {
        if (s1.length() != s2.length()) {
            return false;
        }
        int i = 0;
        while (i < s1.length() && s1.charAt(i) == s2.charAt(i)) {
            i++;
        }
        return i >= s1.length();
    }
```

3.
```
    public static ArrayList<Character> split(String s) {
        ArrayList list = new ArrayList<Character>();
        for (int i = 0; i < s.length(); i++) {
            list.add(new Character(s.charAt(i)));
        }
        return list;
    }
```

```
public static String join(ArrayList<Character> list) {
    StringBuilder sb = new StringBuilder();
    for (int i = 0; i < list.size(); i++) {
        Character elem = list.get(i);
        sb.append(elem.charValue());
    }
    return sb.toString();
}
```

4.
```
public static String hyphenation(String word, int[] pos) {
    int p = 0;
    StringBuilder sb = new StringBuilder();
    for (int i = 0; i < word.length(); i++) {
        sb.append(word.charAt(i));
        if (p < pos.length && i == pos[p]) {
            sb.append('-');
            p++;
        }
    }
    return sb.toString();
}
```