Lösningar till övningsuppgifterna

Lösningar till övningsuppgifterna som hämtats från läroboken finns i boken. Observera att man kan lösa uppgifter på många olika sätt; detta är *förslag* till lösningar.

För att spara plats har vi utelämnat alla dokumentationskommentarer — det får man inte göra i riktiga program.

Övning 1

1. Uppgift 2.2 i boken (sidan 266).

2. a) Parentesen runt `c + d` i tilldelningssatsen `int e = ...` är onödig. Om den utelämnas görs ändå `c + d` först, eftersom den additionen står först.

   b) `6 2 6 8 9`

   c) `7 2 10 4 9`

3.
```
public class TimeDifference {
    public static void main(String[] args) {
        System.out.println("Skriv starttid och sluttid (timmar och minuter):");
        Scanner scan = new Scanner(System.in);
        int startHour = scan.nextInt();
        int startMin = scan.nextInt();
        int stopHour = scan.nextInt();
        int stopMin = scan.nextInt();
        int minutes = 60 * (stopHour - startHour) + (stopMin - startMin);
        System.out.println("Tidsavstånd i minuter: " + minutes);
    }
}
```

   För att skriva ut tidsavståndet i timmar och minuter subtraherar vi timmarna för sig, minuterna för sig. Vi måste ta hänsyn till att minutskillnaden kan bli negativ och i så fall korrigera det. Nya satser (efter inläsningssatserna):

```
int diffHour = stopHour - startHour;
int diffMin = stopMin - startMin;
if (diffMin < 0) {
    diffMin = diffMin + 60;
    diffHour = diffHour - 1;
}
System.out.println("Tidsavstånd: " + diffHour +
                   " timmar, " + diffMin + " minuter");
```

   Den andra varianten, med heltalsdivision:

```
int minutes = 60 * (stopHour - startHour) + (stopMin - startMin);
int diffHour = minutes / 60;
int diffMin = minutes % 60;
System.out.println("Tidsavstånd: " + diffHour +
                   " timmar, " + diffMin + " minuter");
```

4. Vi använder en `while`-sats för att upprepa beräkningarna:

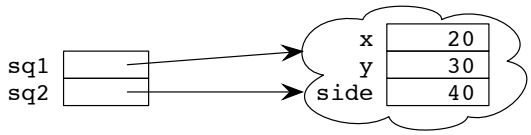
```
public class TimeDifference {
    public static void main(String[] args) {
        System.out.println("Skriv starttid och sluttid (timmar och minuter)");
        System.out.println("(skriv ett negativt tal för att avsluta)");
        Scanner scan = new Scanner(System.in);
        int startHour = scan.nextInt();
        while (startHour >= 0) {
            int startMin = scan.nextInt();
            int stopHour = scan.nextInt();
            int stopMin = scan.nextInt();
            int minutes = 60 * (stopHour - startHour) + (stopMin - startMin);
            System.out.println("Tidsavstånd i minuter: " + minutes);
            System.out.println("Skriv nya tider:");
            startHour = scan.nextInt();
        }
    }
}
```

5. Man måste använda en variabel (här kallar vi den `temp`) för att ”mellanlagra” det ena värdet. Satser:

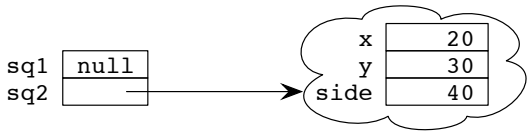
```
int temp = a;
a = b;
b = temp;
```

Övning 2

1. Efter satsen `sq2 = sq1` refererar båda referensvariablerna till samma objekt:



Satsen `sq1 = null` sätter *referensvariabeln* till `null` — objektet som variabeln refererar till påverkas inte:



2. Man kan börja med en `if`-sats som skiljer ut männen och kvinnorna och sedan skriva separata beräkningar för män och kvinnor. Så gör vi inte här: i stället använder vi variabler som sätts till olika värden beroende på könet hos personen och gör sedan beräkningarna på ett ställe. Detta är en allmän regel — man ska bara skriva samma sak en gång.

```
public int howWellFed() {
    int coeff1;
    double coeff2;
    if (sex == 'm') {
        coeff1 = 110;
```

```
            coeff2 = 0.9;
        } else {
            coeff1 = 115;
            coeff2 = 0.95;
        }
        int normWeight = height - coeff1;
        int result;
        if (weight < coeff2 * normWeight) {
            result = -1;
        } else if (weight <= 1.05 * normWeight) {
            result = 0;
        } else {
            result = +1;
        }
        return result;
    }
```

3. Uppgift 3.3 i boken (sidan 269).

4.
```
public class LineDrawing {
    public static void main(String[] args) {
        SimpleWindow w = new SimpleWindow(500, 500, "LineDrawing");
        w.moveTo(0, 0);
        while (true) {
            w.waitForMouseClick();
            w.lineTo(w.getMouseX(), w.getMouseY());
        }
    }
}
```

Övning 3

1.
```
public class CollatzSequence {
    public static void main(String[] args) {
        System.out.println("Skriv startvärdet i talföljden (a0)");
        Scanner scan = new Scanner(System.in);
        int a = scan.nextInt();
        System.out.println(a);
        int length = 1;
        while (a != 1) {
            if (a % 2 == 0) {
                a = a / 2;
            } else {
                a = 3 * a + 1;
            }
            System.out.println(a);
            length = length + 1;
        }
        System.out.println("Antal tal = " + length);
    }
}
```

2. Uppgift 3.4 i boken (sidan 269).

3. Uppgift 3.8 i boken (sidan 271).

4. En groda måste hålla reda på sin position (koordinaterna `x` och `y`) och antalet hopp som den har gjort (`jumps`). `jumps++` betyder detsamma som `jumps = jumps + 1` (7.2 i boken).

```
public class Frog {
    private int x;
    private int y;
    private int jumps;

    public Frog() {
        x = 0;
        y = 0;
        jumps = 0;
    }

    public int getX() {
        return x;
    }

    public int getY() {
        return y;
    }

    public int getJumps() {
        return jumps;
    }

    public void jumpNorth() {
        y = y + 1;
        jumps++;
    }

    public void jumpEast() {
        x = x + 1;
        jumps++;
    }

    public void jumpSouth() {
        y = y - 1;
        jumps++;
    }

    public void jumpWest() {
        x = x - 1;
        jumps++;
    }
}
```

5. Uppgift 3.14 i boken (sidan 274).

Övning 4

1. Uppgift 3.10 i boken (sidan 272).

2. När en metod anropas ska de aktuella parametrarna (som man skriver i anropet) föras över till de formella parametrarna (de som finns i metoden). Detta görs genom att de aktuella parametrarnas *värden* beräknas och kopieras till de formella parametrarna. I exemplet:

    1. Värdet av den aktuella parametern `j` beräknas (1).
    2. Den formella parametern `i` tilldelas detta värde.
    3. `incr` anropas.
    4. `i` ökas med 1 och skrivs ut.

5. Efter anropet skrivs värdet av `j` ut. `j` har samma värde som det hade före anropet av `incr`.

Utskriften blir alltså:

```
2
1
```

3. Inuti metoden `isOlderThan` har man tillgång till två `Person`-objekt: det som man utför operationen på (det objekt som står före punkten) och det objekt som skickats som parameter. I det första objektet kommer man åt attributet `age` utan någon punktnotation, som vanligt. I parameterobjektet kommer man åt `age` med punktnotation trots att det är deklarerat `private`, eftersom de båda objekten är av samma klass.

   Implementering:

```
public class Person {
    private int age;

    public Person(int age) {
        this.age = age;
    }

    public boolean isOlderThan(Person p) {
        return age > p.age;
    }
}
```

I stället för `return age > p.age` kan man skriva så här (fast det är inte lika snyggt):

```
    if (age > p.age) {
        return true;
    } else {
        return false;
    }
```

4. Notera villkoret i `while`-satsen, som uttrycker att grodan befinner sig på gräsmattan. När något av de fyra delvillkoren blir `false` blir hela villkoret `false` och `while`-satsen lämnas. För att bestämma i vilken riktning som grodan ska hoppa drar vi vi ett slumptal mellan 0 och 3 och utnyttjar en `switch`-sats (se avsnitt 7.4 i läroboken) för att testa värdet av talet. Det går precis lika bra att testa med flera `if`-satser efter varandra.

```
public class FrogJump {
    public static void main(String[] args) {
        Random rand = new Random();
        Frog f = new Frog();
        while (f.getX() >= -5 && f.getX() <= 5 &&
               f.getY() >= -5 && f.getY() <= 5) {
            int rnd = rand.nextInt(4);
            switch (rnd) {
                case 0: f.jumpNorth(); break;
                case 1: f.jumpEast(); break;
                case 2: f.jumpSouth(); break;
                case 3: f.jumpWest(); break;
            }
        }
        System.out.println("Grodan gjorde " + f.getJumps() + " hopp");
    }
}
```

5. Uppgift 3.11 i boken (sidan 272).

Övning 5

1. Uppgift 6.2 i boken (sidan 276).

2. Uppgift 6.3 i boken (sidan 276).

3. När referenser förs över som parametrar används värdeanrop, precis som för alla typer av parametrar. Men det är värdet av *referensen* som kopieras till metoden — det skapas inte någon kopia av objektet. I metoden `use` kommer alltså variabeln `pa` att referera till samma objekt som `a`, och när man utför `pa.set(10)` så ändras innehållet i objektet.

   Utskrift (kommentarerna inom parentes skrivs inte ut):

```
value = 5  (före anropet av use)
value = 5  (första print i use)
value = 10 (andra print i use)
value = 10 (efter anropet av use)
```

4. Satsen `pa = new A(10)` i `use` medför att den formella parametern `pa` sätts att referera till ett nytt objekt. Variabeln `a` påverkas inte. När man lämnar `use` så försvinner alla parametrar och lokala variabler — det medför att objektet med `value` = 10 inte längre kan nås och att det kommer att tas bort av skräpsamlaren.

   Utskrift:

```
value = 5
value = 5
value = 10
value = 5
```

5.
```
public class MinMax {
    public static void main(String[] args) {
        int maxOdd = Integer.MIN_VALUE;
        int minEven = Integer.MAX_VALUE;
        System.out.println("Skriv tal (minst ett udda och minst ett jämnt):");
        Scanner scan = new Scanner(System.in);
        while (scan.hasNextInt()) {
            int nbr = scan.nextInt();
            if (nbr % 2 != 0) {
                if (nbr > maxOdd) {
                    maxOdd = nbr;
                }
            } else {
                if (nbr < minEven) {
                    minEven = nbr;
                }
            }
        }
        System.out.println("Största udda tal = " + maxOdd);
        System.out.println("Minsta jämna tal = " + minEven);
    }
}
```

6. Uppgift 6.7 i boken (sidan 277).

Övning 6

1. Metoden har tillgång till attributen i klassen: vektorn `nbrs` och antalet tal `n`. Den behöver alltså inga parametrar.

```
public void checkSorting() {
    int max = Integer.MIN_VALUE;
    for (int i = 0; i < n; i++) {
        if (nbrs[i] < max) {
            System.out.println("Tal nr " + (i + 1) +
                    " är felsorterat, dess värde är " + nbrs[i]);
        } else {
            max = nbrs[i];
        }
    }
}
```

2. Uppgift 8.3 i boken (sidan 283).

3. I `remove` flyttas det sista elementet i `apartments`-vektorn till ”hålet” där det borttagna elementet fanns. Det går bra att göra så, eftersom ordningen mellan elementen inte har någon betydelse. I `countApartments` görs två genomgångar av vektorn: först en för att hitta det maximala antalet rum, sedan en för att registrera antalet rum.

```
public class Register {
    private Apartment[] apartments;
    private int n;

    public Register(int maxSize) {
        apartments = new Apartment[maxSize];
        n = 0;
    }

    public void add(Apartment a) {
        apartments[n] = a;
        n++;
    }

    public void remove(int id) {
        int i = 0;
        while (i < n && apartments[i].getId() != id) {
            i++;
        }
        if (i < n) {
            apartments[i] = apartments[n - 1];
            n--;
        }
    }

    public int[] countApartments() {
        int max = 0;
        for (int i = 0; i < n; i++) {
            if (apartments[i].getNbrOfRooms() > max) {
                max = apartments[i].getNbrOfRooms();
            }
        }
```

```
        int[] count = new int[max];
        for (int i = 0; i < n; i++) {
            int rooms = apartments[i].getNbrOfRooms();
            if (rooms > 0) {
                count[rooms - 1]++;
            }
        }
        return count;
    }
}
```

4. De fyra lösningsvarianterna i uppgiften beskrivs i metoderna `print1`, `print2`, `print3` och `print4`:

```
public class LetterPrinter {;
    public static void main(String[] args) {
        char[] letters = new char[26];
        for (int i = 0; i < letters.length; i++) {
            letters[i] = (char) ('A' + i);
        }
        Random rand = new Random();

        print1(letters, rand); // eller print2 eller print3 eller print4

        System.out.println();
    }

    private static void print1(char[] letters, Random rand) {
        for (int i = 0; i < letters.length; i++) {
            int k;
            do {
                k = rand.nextInt(letters.length);
            } while (letters[k] == ' ');
            System.out.print(letters[k]);
            letters[k] = ' ';
        }
    }

    private static void print2(char[] letters, Random rand) {
        int max = letters.length;
        for (int i = 0; i < letters.length; i++) {
            int k = rand.nextInt(max);
            System.out.print(letters[k]);
            for (int j = k; j < max - 1; j++) {
                letters[j] = letters[j + 1];
            }
            max--;
        }
    }

    private static void print3(char[] letters, Random rand) {
        int max = letters.length;
        for (int i = 0; i < letters.length; i++) {
            int k = rand.nextInt(max);
            System.out.print(letters[k]);
            letters[k] = letters[max - 1];
            max--;
        }
    }
```

```
    private static void print4(char[] letters, Random rand) {
        for (int i = letters.length - 1; i > 0; i--) {
            int k = rand.nextInt(i + 1);
            char temp = letters[k];
            letters[k] = letters[i];
            letters[i] = temp;
        }
        for (int i = 0; i < letters.length; i++) {
            System.out.print(letters[i]);
        }
    }
}
```

När antalet bokstäver som ska skrivas ut är så litet som 26 har det inte någon betydelse vilken metod man använder. Men om antalet (kalla det `n`) skulle vara större blir det annorlunda:

`print1` När `i` börjar närma sig `n` kommer man att behöva dra många slumptal innan man hittar en plats i vektorn som inte är blank. Detta tar naturligtvis tid.

`print2` Detta är den sämsta metoden för stora `n` — för varje slumptal måste man flytta många element i vektorn.

`print3` Detta är den bästa metoden. Man gör bara en genomgång av vektorn.

`print4` Denna metod är lika bra som metod 3..

Övning 7

1. Metoden behöver två parametrar: vektorn som ska sorteras och antalet tal. Metoden får alla sina indata via parametrarna och bör därför vara en statisk metod.

```
public static void sort(int[] sequence, int n) {
    for (int i = 1; i < n; i++) {
        int nbr = sequence[i];
        int k = i - 1;
        while (k >= 0 && sequence[k] > nbr) {
            sequence[k + 1] = sequence[k];
            k--;
        }
        sequence[k + 1] = nbr;
    }
}
```

2.
```
public class CharList {
    private ArrayList<Character> chars;

    public CharList() {
        char = new ArrayList<Character>();
    }

    public void insert(char ch) {
        chars.add(new Character(ch));
    }
```

```
    public void trim() {
        // Inledande blanka tas bort.
        while (!chars.isEmpty() && chars.get(0).charValue() == ' ') {
            chars.remove(0);
        }
        // Avslutande blanka tas bort.
        int lastIndex = chars.size() - 1;
        while (lastIndex >= 0 && chars.get(lastIndex).charValue() == ' ') {
            chars.remove(lastIndex);
            lastIndex--;
        }
    }

    public int getLastLetterIndex() {
        int i = chars.size() - 1;
        while (i >= 0 && !(chars.get(i).charValue() >= 'a' &&
                chars.get(i).charValue() <= 'z')) {
            i--;
        }
        return i;
    }
}
```

3. Uppgift 12.2 i boken (sidan 294).

Övning 8

1.
```
public class PriorityQueue {
    private ArrayList<Person> persons;

    public PriorityQueue() {
        persons = new ArrayList<Person>();
    }

    public int size() {
        return persons.size();
    }

    public void insert(Person p) {
        persons.add(p);
    }

    public Person getOldest() {
        int oldestPersonIndex = -1;
        int maxAge = Integer.MIN_VALUE;
        for (int i = 0; i < persons.size(); i++) {
            Person p = persons.get(i);
            if (p.getAge() > maxAge) {
                maxAge = p.getAge();
                oldestPersonIndex = i;
            }
        }
        return persons.remove(oldestPersonIndex);
    }
}
```

2. Uppgift 12.3 i boken (sidan 295).

3.
```
public class IntStack {
    private ArrayList<Integer> v;

    public IntStack() {
        v = new ArrayList<Integer>();
    }

    public void push(int nbr) {
        v.add(new Integer(nbr));
    }

    public int pop() {
        Integer elem = v.remove(v.size() - 1);
        return elem.intValue();
    }

    public boolean empty() {
        return v.isEmpty();
    }
}
```

Övning 9

1. a) Man anger arv med `extends` och anropar superklassens konstruktor med `super`:

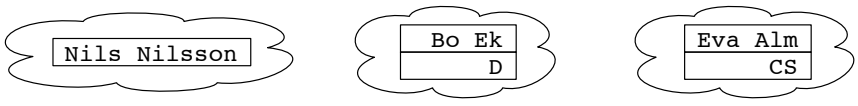
```
public class Person {
    private String name;

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

public class Student extends Person {
    private String program;

    public Student(String name, String program) {
        super(name);
        this.program = program;
    }
}
```

b) Observera `Student`- och `Teacher`-objekten: attributet `name` har ärvts från `Person`, men det syns inte på objekten.



c) 1) `p = s` är korrekt — `p` får referera till `Student`-objekt eftersom `Student` är en subklass till `Person`, och `s` refererar garanterat till ett `Student`-objekt; 2) `s = p` är felaktigt — `p` kan ju referera till andra slags objekt än `Student`-objekt; 3) `s = t` är felaktigt — `s` får inte referera till `Teacher`-objekt; 4) `s = (Student) p` är korrekt — referensen konverteras explicit till `Student` (men det kan bli fel under exekvering).

d) `s = (Student) p` ger `ClassCastException` om `p` under exekvering inte refererar till ett `Student`-objekt (eller till ett objekt av en subklass till `Student`).

e) `super.toString()` betyder att `toString`-metoden i superklassen anropas. Utskrift:

```
Nils Nilsson
Bo Ek, D
Bo Ek, D
```

Notera att samma sats, `System.out.println(p.toString())`, ger två olika resultat. Det beror på att Java använder dynamisk bindning för metodanrop: första gången refererar `p` till ett `Person`-objekt och `Person.toString()` anropas, andra gången refererar `p` till ett `Student`-objekt och `Student.toString()` anropas.

Anmärkning: `toString` anropas automatiskt när man ger ett objekt som parameter till `print` eller `println`. I stället för `System.out.println(p.toString())` kan man alltså skriva `System.out.println(p)`.

2.
```
public class BlogPost {
    private String title;
    private String text;
    private String date;
    private ArrayList<String> tags;

    public BlogPost(String title, String text, String date) {
        this.title = title;
        this.text = text;
        this.date = date;
        tags = new ArrayList<String>();
    }

    public void addTag(String tag) {
        if (!tags.contains(tag)) {
            tags.add(tag);
        }
    }

    public boolean allTagsMatch(ArrayList<String> words) {
        for (int i = 0; i < words.size(); i++) {
            if (!tags.contains(words.get(i))) {
                return false;
            }
        }
        return true;
    }

    public String toString() {
        return title + " " + date + "\n" + text;
    }
}
```

3.
```
    public ArrayList<BlogPost> getMatchingPosts(ArrayList<String> words) {
        ArrayList<BlogPost> result = new ArrayList<BlogPost>();
        for (int i = 0; i < posts.size(); i++) {
            if (posts.get(i).allTagsMatch(words)) {
                result.add(posts.get(i));
            }
        }
        return result;
    }
```

4.
```
public static void partition(int[] v, Condition cond) {
    int first = 0;
    while (first < v.length && cond.evaluate(v[first])) {
        first++;
    }
    for (int i = first + 1; i < v.length; i++) {
        if (cond.evaluate(v[i])) {
            int temp = v[i];
            v[i] = v[first];
            v[first] = temp;
            first++;
        }
    }
}
```

Övning 10

1. Jämför med lösningen till uppgift 11.1.e i boken: här är strängen parameter till funktionen. Lösningen är något annorlunda (bokens lösning är kanske bättre, för där anropas `isWhiteSpace` bara en gång för varje tecken). Idén är dock densamma — man identifierar början av varje ord.

```
public static int getNbrOfWords(String s) {
    int words = 0;
    char prevChar = ' ';
    for (int i = 0; i < s.length(); i++) {
        if (!Character.isWhiteSpace(s.charAt(i)) &&
                Character.isWhiteSpace(prevChar)) {
            words++;
        }
        prevChar = s.charAt(i);
    }
    return words;
}
```

Genom att utnyttja en `Scanner` får man en betydligt enklare lösning (en `Scanner` kan ju läsa från en sträng, och metoden `hasNext` läser förbi whitespace).

```
public static int getNbrOfWords(String s) {
    int words = 0;
    Scanner scan = new Scanner(s);
    while (scan.hasNext()) {
        String word = scan.next();
        words++;
    }
    return words;
}
```

2.
```
public static String encrypt(String plainText, long key) {
    Random rand = new Random(key);
    StringBuilder sb = new StringBuilder();
    for (int i = 0; i < plainText.length(); i++) {
        int newCode = (plainText.charAt(i) + rand.nextInt(256)) % 256;
        sb.append((char) newCode);
    }
    return sb.toString();
}
```

3. Uppgift 11.4 i boken (sidan 292).

4. Uppgift 11.5 i boken (sidan 292).