Övningsuppgifter

I kursen ingår 10 övningar (det är ingen övning vecka 7 i läsperiod ht2). Under övningarna ska du självständigt eller tillsammans med en kamrat lösa övningsuppgifterna. Övningsledaren ger tips om lösning, hjälper till om du får problem och diskuterar dina lösningar med dig.

Före varje övning måste du förbereda dig genom att läsa i läroboken, titta igenom uppgifterna och gärna försöka lösa en del av dem.

Du som har programmerat tidigare kanske tycker att uppgifterna är enkla och gör dem på egen hand. Kontrollera då dina lösningar mot lösningsförslagen och övertyga dig om att du förstår allting.

Till uppgifterna som är hämtade från läroboken finns lösningsförslag i boken. Till de övriga uppgifterna finns lösningsförslag på kursens hemsida.

Det finns många fler övningsuppgifter: dels i läroboken, dels på kursens hemsida (”extra övningsuppgifter”).

Övning 1

Läs i läroboken: avsnitt 2.1–2.5, 3.2.

1. Uppgift 2.2 i läroboken (sidan 21).

2. Betrakta följande program, där fyra heltal läses från tangentbordet och några (ointressanta) beräkningar görs:

```
public class Example1 {
    public static void main(String[] args) {
        Scanner scan = new Scanner(System.in);
        int a = scan.nextInt();
        int b = scan.nextInt();
        int c = scan.nextInt();
        int d = scan.nextInt();
        int e = (c + d) - (a + b);
        c = c + 2;
        a = (2 * e + c) / 4;
        System.out.println(a + " " + b + " " + c + " " + d + " " + e);
    }
}
```

   a) En av parenteserna i programmet är onödig. Vilken? Varför?
   b) Vilka värden skrivs ut när talen 1, 2, 4 och 8 läses? Använd en tabell där du noterar de successiva variabelvärdena.
   c) Vilka värden skrivs ut när talen 1, 2, 8 och 4 läses?

3. Skriv ett program (en klass med en `main`-metod) som först läser en starttid (två tal, timmar och minuter, till exempel 12 41) och därefter en sluttid (två andra tal, till exempel 16 28) och därefter beräknar och skriver ut hur många minuter det är mellan tiderna. Du kan förutsätta att sluttiden är större än starttiden. Ledning: det behövs fyra `int`-variabler för de fyra inlästa talen. Ge dessa variabler vettiga namn.

   Ändra sedan programmet så att tidsavståndet skrivs ut i timmar och minuter i stället för i minuter. Detta kan man göra på två sätt: antingen genom att använda en `if`-sats eller genom att använda heltalsdivision och operatorn `%` (avsnitt 6.3 i läroboken).

4. Ändra programmet i uppgift 3 så att beräkningarna upprepas gång på gång tills ett negativt tal läses in (som timmarna i starttiden).

5. Två heltalsvariabler `a` och `b` har deklarerats och fått värden. Skriv satser som byter värde på `a` och `b` (”swappar” värdena). Exempel:

```
int a = 10;
int b = 25;
// ... dina satser
System.out.println(a + " " + b); // ger utskriften 25 10
```

Övning 2

Läs i läroboken: avsnitt 2.6–2.8, 3.3–3.4, appendix C.

1. Rita en bild av minnessituationen (vilka variabler som finns, variablernas värden, vilka objekt som finns, attributens värden) när följande satser har exekverats:

```
Square sq1 = new Square(20, 30, 40);
Square sq2 = sq1;
sq1 = null;
```

2. Betrakta följande klass, som beskriver en person (datatypen `char` representerar ett tecken, se läroboken avsnitt 6.6):

```
public class Person {
    private int weight; // personens vikt i kg
    private int height; // längd i cm
    private char sex;   // kön, 'm' = man, 'k' = kvinna

    /** Skapar en person med vikten weight (kg), längden height (cm)
        och könet sex ('m'=man, 'k'=kvinna) */
    public Person(int weight, int height, char sex) {
        this.weight = weight;
        this.height = height;
        this.sex = sex;
    }

    /** Undersöker om personen är undernärd (-1), ordinär (0)
        eller välnärd (+1) */
    public int howWellFed() {
        ...
    }
}
```

   Implementera operationen `howWellFed`. Följande regler gäller: en man har normalvikten $normWeight = height - 110$ kg. Om hans verkliga vikt är mindre än $0.9 * normWeight$ är han undernärd, om den är större än $1.05 * normWeight$ är han välnärd. Motsvarande gäller kvinnor om 110 ersätts med 115 och 0.9 med 0.95.

   För att skilja mellan män och kvinnor gör man så här:

```
if (sex == 'm') {
    ... // man
} else {
    ... // kvinna
}
```

3. Uppgift 3.3 i läroboken (sidan 54).

4. Skriv ett program med vars hjälp man kan rita linjer i ett `SimpleWindow`-fönster genom att klicka med musen. Den första linjen ska börja i punkten (0,0) och sluta där man klickar första gången. Den andra linjen ska börja där den första slutade och sluta där man klickar andra gången, osv. Ritandet ska fortsätta ”i oändlighet”, alltså tills man bryter programmet.

Övning 3

Läs i läroboken: avsnitt 2.6, 3.6, 3.9, 6.2–6.3.

1. Ett heltal a_0 är givet. Man beräknar en talföljd med följande formler:

$$a_{k+1} = \begin{cases} a_k/2, & \text{om } a_k \text{ är jämnt} \\ 3a_k + 1, & \text{annars} \end{cases}$$

   För alla a_0 större än 0 blir a_k förr eller senare $= 1$. (Detta har ingen lyckats bevisa — det kallas Collatz problem — men de flesta tror att det är så.) Exempel:

   $a_0 = 3$ ger 10, 5, 16, 8, 4, 2, 1 (8 tal i talföljden)
   $a_0 = 4$ ger 2, 1 (3 tal)
   $a_0 = 5$ ger 16, 8, 4, 2, 1 (6 tal)
   $a_0 = 6$ ger 3, 10, 5, 16, 8, 4, 2, 1 (9 tal)
   $a_0 = 7$ ger 22, 11, 34, 17, 52, 26, 13, 40, 20, 10, 5, 16, 8, 4, 2, 1 (17 tal)

   Skriv ett program som läser in ett a_0 och skriver ut den talföljd som bildas med formlerna, till och med den avslutande 1-an. Skriv också ut antalet tal i talföljden.

2. Uppgift 3.4 i läroboken (sidan 54).

3. Uppgift 3.8 i läroboken (sidan 56).

4. En groda hoppar omkring på en gräsmatta. Grodan kan hoppa åt norr, öster, söder och väster. Varje hopp är en meter långt. Från början befinner sig grodan i mitten av gräsmattan, dvs i en punkt med koordinaterna (0, 0). Grodan beskrivs av en klass med följande specifikation:

   `Frog`

```
/** Skapar en groda som befinner sig i punkten (0,0) */
Frog();

/** Tar reda på x-koordinaten (i meter) för grodans position */
int getX();

/** Tar reda på y-koordinaten (i meter) för grodans position */
int getY();

/** Tar reda på hur många hopp grodan gjort sedan den skapades */
int getJumps();

/** Låter grodan hoppa norrut */
void jumpNorth();

// ... här finns tre ytterligare operationer: jumpEast,
// ... jumpSouth och jumpWest
```

   Implementera klassen `Frog`. Observera att grodans hopp inte ska ritas ut någonstans.

5. Uppgift 3.14 i läroboken (sidan 58).

Övning 4

Läs i läroboken: avsnitt 3.7.1, 3.9–3.10, 3.12, 6.4, 6.10.

1. Uppgift 3.10 i läroboken (sidan 57).

2. Betrakta följande enkla klass (increment betyder ”öka”):

```
public class A {
    public static void increment(int i) {
        i = i + 1;
        System.out.println(i);
    }
}
```

   Vilken utskrift fås när nedanstående programrader exekveras? Rita aktiveringsposter och förklara resultatet (se avsnitt 3.12 i läroboken).

```
int j = 1;
A.increment(j);
System.out.println(j);
```

   Notera att metoden `incr` är statisk — den använder inte något attribut från något objekt av klassen som den finns i. Statiska metoder anropar man genom att skriva klassnamnet före punkten.

3. Klassen `Person` beskriver personer (klassen har fler operationer, men dem är vi inte intresserade av här):

```
/** Skapar en person med åldern age */
Person(int age);

/** Undersöker om denne person är äldre än personen p. Returnerar
    då true, annars false */
boolean isOlderThan(Person p);
```

   Exempel på användning av klassen:

```
Person p1 = new Person(23);
Person p2 = new Person(21);
if (p1.isOlderThan(p2)) { ... }
```

   Implementera klassen.

4. Fortsättning på uppgift 4, övning 3: skriv ett program som placerar grodan i mitten av en 10×10 m stor gräsmatta och låter grodan hoppa omkring slumpmässigt (med samma sannolikhet i alla riktningar) tills den hamnar utanför gräsmattan. Skriv ut antalet hopp som grodan gjorde.

5. Uppgift 3.11 i läroboken (sidan 57).

Övning 5

Läs i läroboken: avsnitt 2.6, 3.12, 6.3–6.4, 6.10, 7.5 (De Morgans lagar), 7.8, 7.13.

1. Uppgift 6.2 i läroboken (sidan 93).

2. Uppgift 6.3 i läroboken (sidan 93).

3. Betrakta följande klass:

```
public class A {
    private int value;

    public A(int value) {
        this.value = value;
    }

    public void set(int newValue) {
        value = newValue;
    }

    public void print() {
        System.out.println("value = " + value);
    }
}
```

   Vilken utskrift fås när nedanstående `main`-metod exekveras? Rita aktiveringsposter, förklara resultatet.

```
public class Test {
    public static void use(A pa) {
        pa.print();
        pa.set(10);
        pa.print();
    }

    public static void main(String[] args) {
        A a = new A(5);
        a.print();
        use(a);
        a.print();
    }
}
```

4. Vilken utskrift fås av satserna i uppgift 3 om metoden `use` har följande definition:

```
public void use(A pa) {
    pa.print();
    pa = new A(10);
    pa.print();
}
```

5. Skriv ett program som från tangentbordet läser ett antal heltal (använd `hasNextInt()` för att undersöka om det finns fler tal att läsa). I programmet ska du beräkna och skriva ut det största av de udda talen och det minsta av de jämna talen. Exempel: om talen är 4, 7, 9, 4, 2, 66, 41, 3, 12 så ska talen 41 och 2 skrivas ut. Du får förutsätta att det finns minst ett udda tal och minst ett jämnt tal.

6. Uppgift 6.7 i läroboken (sidan 94).

Övning 6

Läs i läroboken: avsnitt 6.4, 6.10, 8.1–8.4, 8.8, 8.10.

1. (Detta är uppgift 7.4 i boken, omformulerad så att talen finns i en vektor.) En klass har två attribut: en heltalsvektor `nbrs` och ett heltal `n`, som anger hur många tal som finns i `nbrs`. Skriv en metod som kontrollerar att talen är sorterade i växande ordning. Om det är fel i sorteringsordningen ska utskrift ske enligt nedanstående exempel.

   Talen i vektorn (här är $n = 10$):

   ```
   1  1  2  7  4  5  12  11  8  23
   ```

   ska ge utskriften:

   ```
   Tal nr 5 är fel, dess värde är 4
   Tal nr 6 är fel, dess värde är 5
   Tal nr 8 är fel, dess värde är 11
   Tal nr 9 är fel, dess värde är 8
   ```

   Observera att 5-an betraktas som felsorterad trots att 5 är större än 4. Man jämför 5-an med 7-an, som är det hittills största värdet.

2. Uppgift 8.3 i läroboken (sidan 140).

3. En bostadslägenhet beskrivs av följande klass (det finns fler operationer, men de används inte i denna uppgift):

   `Apartment`

   ```
   /** Tar reda på lägenhetens nummer */
   int getId();

   /** Tar reda på antalet rum i lägenheten */
   int getNbrOfRooms();
   ```

   En kommun har ett register över samtliga lägenheter i kommunen. Registret beskrivs av följande klass:

   `Register`

   ```
   /** Skapar ett tomt register med plats för maxSize lägenheter */
   Register(int maxSize);

   /** Lägger in lägenheten a i registret. Det förutsätts att det
       finns plats och att lägenheten inte redan finns i registret */
   void add(Apartment a);

   /** Tar bort lägenheten med nummer id ur registret. Om lägenheten
       inte finns ska ingenting inträffa */
   void remove(int id);

   /** Räknar antalet lägenheter med 1, 2, 3, ... rum, returnerar
       antalen i en vektor (antalet 1-rummare i [0], osv) */
   int[] countApartments();
   ```

   Implementera klassen `Register`. Vektorn som returneras i `countApartments` ska vara lika lång som antalet rum i den största lägenheten som finns i registret.

4. Skriv ett program som skriver ut de 26 bokstäverna A–Z i slumpmässig ordning. Varje bokstav ska förekomma exakt en gång i utskriften.

   Man kan lösa uppgiften på många olika sätt — alla börjar med att man lägger bokstäverna i ordning i en vektor. Fyra varianter:

   1. Välj en av bokstäverna slumpmässigt, skriv ut den, lägg ett blanktecken på bokstavens plats. Fortsätt tills alla bokstäverna är utskrivna. Man måste dra slumptal tills man hamnar på en plats i vektorn som inte är blank.
   2. Välj en av bokstäverna slumpmässigt, skriv ut den, ta bort den ur vektorn genom att flytta de efterföljande bokstäverna. Välj en ny bokstav på samma sätt, ...
   3. Som metod 2, men flytta den sista bokstaven i vektorn till platsen för det utskrivna talet.
   4. Blanda alla bokstäverna slumpmässigt. Skriv sedan ut hela vektorn.

   Diskutera för- och nackdelar med metoderna. Implementera metoden som du tycker är bäst.

Övning 7

Läs i läroboken: avsnitt 7.13, 8.9, 12.1–12.4.

1. (Detta är uppgift 8.7 i boken, omformulerad så att vektorn som ska sorteras är parameter.) ”Insättningssortering” är en bra sorteringsmetod om antalet tal som ska sorteras inte är för stort. I metoden går man igenom talen i tur och ordning och sorterar in varje tal på sin rätta plats bland de redan sorterade talen.

   Exempel, där talföljden består av 7 tal:

13	4	7	5	27	12	2

   När man har klarat av de tre första talen och ska sortera in det fjärde talet ser det ut så här:

4	7	13	5	27	12	2

   När nu 5-an ska sorteras in ska man dels finna talets rätta plats (till höger om 4-an), dels flytta de tal som ligger till höger om denna plats (talet 7 och talet 13) ett steg åt höger. När detta är klart kan man lägga in 5-an på den lediga platsen.

   Implementera en metod som sorterar en vektor med insättningssortering. Vektorn och antalet tal i vektorn som ska sorteras ska vara parametrar. Tänk på att metoden också ska klara fallen när det tal som ska sorteras in är mindre än eller större än alla de redan sorterade talen.

2. Standardklassen `Character` beskriver ett tecken (jämför med standardklassen `Integer` som beskriver ett heltal). Klassen har följande utseende (det finns fler attribut, till exempel konstanterna `MIN_VALUE` och `MAX_VALUE`, och fler operationer):

   ```
   public class Character {
       private char value;
       public Character(char value) { this.value = value; }
       public char charValue() { return value; }
   }
   ```

   Implementera följande klass, som beskriver en lista av tecken:

   ```
   /** Skapar en tom lista */
   CharList();

   /** Lägger in tecknet ch sist i listan */
   void insert(char ch);

   /** Tar bort alla blanktecken som ligger i början eller i slutet
       av listan */
   void trim();

   /** Ger index för det sista elementet som innehåller en bokstav a-z,
       -1 om det inte finns någon bokstav i listan */
   int getLastLetterIndex();
   ```

3. Uppgift 12.2 i läroboken (sidan 229).

Övning 8

Läs i läroboken: avsnitt 7.13, 12.1–12.4.

1. Framför en kassa i ett postkontor finns en kö där det står ett antal personer. När en ny person anländer till postkontoret ställer han eller hon sig sist i kön. När en person ska betjänas väljs av någon anledning alltid den äldste personen i kön.

   En person beskrivs av följande klass:

   `Person`

   ```
   // konstruktor och operationer som inte utnyttjas i denna uppgift

   /** Tar reda på personens ålder */
   int getAge();
   ```

   Kön beskrivs av följande klass:

   `PriorityQueue`

   ```
   /** Skapar en tom kö */
   PriorityQueue();

   /** Tar reda på antalet personer i kön */
   int size();

   /** Lägger in personen p sist i kön */
   void insert(Person p);

   /** Tar ut den äldste personen ur kön, returnerar en referens
       till denne. Du kan förutsätta att det finns minst en person
       i kön. Om det finns flera personer som är äldst så får vem
       som helst av dem returneras */
   Person getOldest();
   ```

   Implementera klassen `PriorityQueue`. Kön ska implementeras med en `ArrayList` som innehåller `Person`-objekt.

2. Uppgift 12.3 i läroboken (sidan 230). Observera: när man ska jämföra två ord (strängar) med varandra kan man inte använda ==. I stället använder man metoden `equals`: uttrycket `w1.equals(w2)` är true om strängarna `w1` och `w2` är lika. Se avsnitt 11.2 i läroboken.

3. En stack är en lista där inläggning av element bara kan göras i ena änden av listan. Borttagning av element kan bara göras i samma ände av listan. Man säger ofta att man lägger element ”på” stacken och tar bort det ”översta” elementet från stacken. En klass som beskriver en stack med heltal kan ha nedanstående utseende. Stacken har en begränsad storlek: högst `maxSize` element kan lagras. Man kontrollerar inte om man försöker lagra fler element eller om man försöker ta bort ett element från en tom stack.

   ```
   public class IntStack {
       private int[] v; // talen på stacken
       private int top; // index för den första lediga platsen i v

       /** Skapar en tom heltalsstack med plats för maxSize element */
       public IntStack(int maxSize) {
           v = new int[maxSize];
           top = 0;
       }
   ```

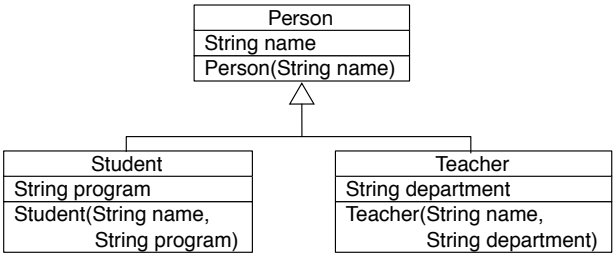
```
    /** Lägger talet nbr på stacken */
    public void push(int nbr) {
        v[top] = nbr;
        top++;
    }

    /** Tar bort det översta talet från stacken, returnerar det */
    public int pop() {
        top--;
        return v[top];
    }

    /** Undersöker om stacken är tom */
    public boolean isEmpty() {
        return top == 0;
    }
}
```

Skriv om klassen så att stacken kan innehålla ett godtyckligt antal element. Konstruktorn behöver då naturligtvis ingen parameter som anger den maximala storleken.

Övning 9

Läs i läroboken: avsnitt 9.1–9.3, 9.5, 9.8, 11.6, 12.1–12.4.

1. Personer, studenter och lärare har ordnats i följande klasshierarki:



   a) Implementera klasserna `Person` och `Student`.

   b) Följande tilldelningssatser är självklart korrekta:

```
Person p = new Person("Nils Nilsson");
Student s = new Student("Bo Ek", "D");
Teacher t = new Teacher("Eva Alm", "CS");
```

   Åskådliggör i en bild hur objekten ser ut.

   c) Satserna i föregående uppgift har utförts. Vilka av följande satser är korrekta, åtminstone så långt kompilatorn kan avgöra det?

```
p = s;
s = p;
s = t;
s = (Student) p;
```

   d) Satsen `s = (Student) p;` kan ge ett fel under exekvering. När inträffar felet? Vad kallas felet?

   e) Alla klasser har en operation `toString()` som ska ge en ”läsbar representation” av objektet (läroboken avsnitt 11.6). I klasserna `Person` och `Student` har metoden implementerats enligt följande:

```
public class Person {
    public String toString() {
        return name;
    }
}

public class Student extends Person {
    public String toString() {
        return super.toString() + ", " + program;
    }
}
```

   Vad betyder `super.toString()`? Vilken utskrift får man av nedanstående satser?

```
Person p = new Person("Nils Nilsson");
Student s = new Student("Bo Ek", "D");
System.out.println(p.toString());
System.out.println(s.toString());
p = s;
System.out.println(p.toString());
```

2. I vissa bloggar kan författaren förse sina inlägg med ”taggar” som talar om vad inlägget handlar om. En tagg är *ett* ord. Implementera följande klass, som beskriver ett blogginlägg:

`BlogPost`

```
/** Skapar ett blogginlägg med titeln title, texten text och
    publiceringsdatum date (på formen yyyy-mm-dd) */
BlogPost(String title, String text, String date);

/** Lägger till taggen tag, om den inte redan finns */
void addTag(String tag);

/** Undersöker om alla orden i listan words ingår bland inläggets
    taggar. Returnerar true i så fall, annars false */
boolean allTagsMatch(ArrayList<String> words);

/** Returnerar en sträng med titel, datum och text. Mellan titel
    och datum ska det vara ett blanktecken, mellan datum och text
    ny rad ("\n") */
String toString();
```

3. Alla blogginläggen samlas i ett objekt av klassen `Blog`:

```
public class Blog {
    private ArrayList<BlogPost> posts;

    /** Skapar en blogg */
    public Blog() {
        posts = new ArrayList<BlogPost>();
        // ... här läses blogginläggen från en databas och lagras i
        // ... listan posts
    }

    /** Ger en lista med de blogginlägg för vilka det gäller att alla
        orden i listan words ingår bland inläggets taggar */
    public ArrayList<BlogPost> getMatchingPosts(ArrayList<String> words) {
        // ... implementera
    }
}
```

Implementera operationen `getMatchingPosts`.

4. En algoritm som man ibland har nytta av är ”partitionering”, vilket innebär att man delar upp element i två grupper: de element som uppfyller ett villkor och de element som inte uppfyller villkoret. I fortsättningen förutsätter vi att elementen är heltal som är lagrade i vektorer och att partitioneringen innebär att elementen flyttas till början eller slutet av vektorn. Exempel: partitionering av vektorn $\{1, 2, 3, 4, 5, 6\}$ med villkoret ”jämnt tal” ska medföra att vektorn blir $\{2, 4, 6, 1, 5, 3\}$ (ordningen mellan de tre första talen är godtycklig, liksom mellan de tre sista).

   När man implementerar algoritmen i Java vill man kunna ”plugga in” olika sorters partitioneringsvillkor i algoritmen. Ett sätt att göra det är att utnyttja följande klass:[1]

```
public abstract class Condition {
    /** Returnerar true om x uppfyller villkoret, false annars */
    public abstract boolean evaluate(int x);
}
```

[1] Egentligen borde klassen ha varit ett gränssnitt, ett `interface`.

För varje specifikt villkor skriver man sedan en subklass till `Condition`. Till exempel ser en klass för villkoret ”jämnt tal” ut så här:

```
public class EvenCondition extends Condition {
    public boolean evaluate(int x) {
        return x % 2 == 0;
    }
}
```

Nu kan man skicka med ett `EvenCondition`-objekt till algoritmen för att partitionera enligt villkoret jämnt tal och objekt av andra subklasser för att partitionera enligt andra villkor.

Implementera partitioneringsalgoritmen i en metod med följande rubrik:

```
/** Flyttar om talen i vektorn v så att de tal som uppfyller villkoret
    cond hamnar före de tal som inte uppfyller villkoret */
public static void partition(int[] v, Condition cond);
```

Observera att man bara behöver gå igenom vektorn en gång.

Övning 10

Läs i läroboken: avsnitt 6.6, 6.10, 11.1–11.6.

1. I en sträng finns en text med ett antal ord. Mellan orden finns det ett eller flera blanktecken eller radframmatningar (”whitespace”). Dessa tecken kan också förekomma före det första ordet och efter det sista ordet. Skriv en metod som räknar antalet ord i en sträng.

2. Slumptal kan användas för att kryptera texter. Man behöver en slumptalsgenerator som kan initieras med ett slumptalsfrö så att den kan upprepa följden av slumptal. Klassen `java.util.Random` är en sådan klass — man använder den konstruktor som har ett slumptalsfrö som parameter.

   Kryptering av en text går till på följande sätt:

   - Välj en krypteringsnyckel, ett `long`-tal `key`.
   - Skapa slumptalsgeneratorn med `key` som slumptalsfrö.
   - För varje tecken i texten: dra ett slumptal, addera det till tecknet.

   Vi förutsätter här att teckenkoderna för tecknen ligger i intervallet [0, 256) och att slumptalen ligger i samma intervall. För att också de krypterade tecknen ska hålla sig inom intervallet ska additionen göras ”cykliskt”, dvs efter tecknet med koden 255 anses tecknet med koden 0 komma. Exempel (teckenkoderna har skrivits i decimal form):

Tecken:	A	t	t	a	c	k	!
Teckenkod:	65	116	116	97	99	107	33
Slumptal:	4	207	6	1	12	255	8
Krypterad kod:	69	67	122	98	111	106	41
Krypterat tecken:	E	C	z	b	o	j	)

   Dekryptering av texten görs genom att man initierar slumptalsgeneratorn med *samma* krypteringsnyckel och subtraherar slumptalen från teckenkoderna.

   Skriv en metod som krypterar en text `plainText` utgående från krypteringsnyckeln `key`. Den krypterade texten ska returneras som resultat. Metoden ska ha följande rubrik:

   ```
   public static String encrypt(String plainText, long key);
   ```

3. Uppgift 11.4 i läroboken (sidan 203).

4. Uppgift 11.5 i läroboken (sidan 203).