Innhold

Nyttige metoder for String-klassen: 2
Nyttige metoder for List-klassen: 3
Nyttige metoder for HashMap-klassen: 4
Nyttige metoder for HashSet-klassen: 4
Nyttige metoder for Integer-klassen: 5
Nyttige metoder for Scanner-klassen: 5
Når en skanner skal lese fra terminal: 5
Nyttige metoder for File-klassen: 6
Lage tester som printer fargekoder til terminal: 6
Eksempel liste: 6
Eksempel Lenkeliste: 6
Eksempel IndeksertListe: 9
Eksempel Prioriteteskø: 12
Hvordan å dele opp Strings og legge dem i Array: 13
Viktig catch å inkludere når man sender en fil inn i en Scanner: 13
Boolsk metode å sjekke at en String matcher et spesifikt format: 13
Formater for printf: 13
Flettetråd: 14
Lesetråd: 15
Monitor: 15
Bruke tråder i en oppgave: 17
Subsekvenser og hashmaps: 21
GUI eksempel 1: 23
GUI eksempel 2: 27
Timer og TimerTask: 29
Eksamen V19: 30

Nyttige metoder for String-klassen:

char	**charAt(int index)** Returns the char value at the specified index.
int	**compareTo(String anotherString)** Compares two strings lexicographically.
int	**compareToIgnoreCase(String str)** Compares two strings lexicographically, ignoring case differences.
String	**concat(String str)** Concatenates the specified string to the end of this string.
boolean	**contains(CharSequence s)** Returns true if and only if this string contains the specified sequence of char values.
boolean	**contentEquals(CharSequence cs)** Compares this string to the specified CharSequence.
boolean	**contentEquals(StringBuffer sb)** Compares this string to the specified StringBuffer.
static String	**copyValueOf(char[] data)** Equivalent to **valueOf(char[])**.
static String	**copyValueOf(char[] data, int offset, int count)** Equivalent to **valueOf(char[], int, int)**.
boolean	**endsWith(String suffix)** Tests if this string ends with the specified suffix.
boolean	**equals(Object anObject)** Compares this string to the specified object.
boolean	**equalsIgnoreCase(String anotherString)** Compares this String to another String, ignoring case considerations.

int	**indexOf(int ch)** Returns the index within this string of the first occurrence of the specified character.
int	**indexOf(int ch, int fromIndex)** Returns the index within this string of the first occurrence of the specified character, starting the search at the specified index.
int	**indexOf(String str)** Returns the index within this string of the first occurrence of the specified substring.
int	**indexOf(String str, int fromIndex)** Returns the index within this string of the first occurrence of the specified substring, starting at the specified index.

int	**lastIndexOf(int ch)** Returns the index within this string of the last occurrence of the specified character.
int	**lastIndexOf(int ch, int fromIndex)** Returns the index within this string of the last occurrence of the specified character, searching backward starting at the specified index.
int	**lastIndexOf(String str)** Returns the index within this string of the last occurrence of the specified substring.
int	**lastIndexOf(String str, int fromIndex)** Returns the index within this string of the last occurrence of the specified substring, searching backward starting at the specified index.
int	**length()** Returns the length of this string.
boolean	**matches(String regex)** Tells whether or not this string matches the given **regular expression**.

char[]	**toCharArray()** Converts this string to a new character array.
String	**toLowerCase()** Converts all of the characters in this String to lower case using the rules of the default locale.
String	**toLowerCase(Locale** locale**)** Converts all of the characters in this String to lower case using the rules of the given Locale.
String	**toString()** This object (which is already a string!) is itself returned.
String	**toUpperCase()** Converts all of the characters in this String to upper case using the rules of the default locale.
String	**toUpperCase(Locale** locale**)** Converts all of the characters in this String to upper case using the rules of the given Locale.
String	**trim()** Returns a string whose value is this string, with any leading and trailing whitespace removed.

Nyttige metoder for List-klassen:

boolean	**add(E** e**)** Appends the specified element to the end of this list (optional operation).
void	**add(**int index, **E** element**)** Inserts the specified element at the specified position in this list (optional operation).
boolean	**addAll(Collection<?** extends **E>** c**)** Appends all of the elements in the specified collection to the end of this list, in the order that they are returned operation).
boolean	**addAll(**int index, **Collection<?** extends **E>** c**)** Inserts all of the elements in the specified collection into this list at the specified position (optional operation).
void	**clear()** Removes all of the elements from this list (optional operation).
boolean	**contains(Object** o**)** Returns true if this list contains the specified element.
boolean	**containsAll(Collection<?>** c**)** Returns true if this list contains all of the elements of the specified collection.
boolean	**equals(Object** o**)** Compares the specified object with this list for equality.
E	**get(**int index**)** Returns the element at the specified position in this list.

int	**indexOf(Object** o**)** Returns the index of the first occurrence of the specified element in this list, or -1 if this list does not contain the element.
boolean	**isEmpty()** Returns true if this list contains no elements.
Iterator<E>	**iterator()** Returns an iterator over the elements in this list in proper sequence.
int	**lastIndexOf(Object** o**)** Returns the index of the last occurrence of the specified element in this list, or -1 if this list does not contain the element.
ListIterator<E>	**listIterator()** Returns a list iterator over the elements in this list (in proper sequence).
ListIterator<E>	**listIterator(**int index**)** Returns a list iterator over the elements in this list (in proper sequence), starting at the specified position in the list.
E	**remove(**int index**)** Removes the element at the specified position in this list (optional operation).
boolean	**remove(Object** o**)** Removes the first occurrence of the specified element from this list, if it is present (optional operation).
boolean	**removeAll(Collection<?>** c**)** Removes from this list all of its elements that are contained in the specified collection (optional operation).
default void	**replaceAll(UnaryOperator<E>** operator**)** Replaces each element of this list with the result of applying the operator to that element.
boolean	**retainAll(Collection<?>** c**)** Retains only the elements in this list that are contained in the specified collection (optional operation).
E	**set(**int index, **E** element**)** Replaces the element at the specified position in this list with the specified element (optional operation).
int	**size()** Returns the number of elements in this list.

Alle metodene over gjelder også for ArrayList.

Nyttige metoder for HashMap-klassen:

void	**clear()** Removes all of the mappings from this map.
Object	**clone()** Returns a shallow copy of this HashMap instance: the keys and values themselves are not cloned.

boolean	**containsKey(Object key)** Returns true if this map contains a mapping for the specified key.
boolean	**containsValue(Object value)** Returns true if this map maps one or more keys to the specified value.
Set<Map.Entry<K,V>>	**entrySet()** Returns a **Set** view of the mappings contained in this map.
void	**forEach(BiConsumer<? super K,? super V> action)** Performs the given action for each entry in this map until all entries have been processed or the action throws an exception.
V	**get(Object key)** Returns the value to which the specified key is mapped, or null if this map contains no mapping for the key.
V	**getOrDefault(Object key, V defaultValue)** Returns the value to which the specified key is mapped, or defaultValue if this map contains no mapping for the key.
boolean	**isEmpty()** Returns true if this map contains no key-value mappings.
Set<K>	**keySet()** Returns a **Set** view of the keys contained in this map.
V	**merge(K key, V value, BiFunction<? super V,? super V,? extends V> remappingFunction)** If the specified key is not already associated with a value or is associated with null, associates it with the given non-null value.

V	**remove(Object key)** Removes the mapping for the specified key from this map if present.
boolean	**remove(Object key, Object value)** Removes the entry for the specified key only if it is currently mapped to the specified value.
V	**replace(K key, V value)** Replaces the entry for the specified key only if it is currently mapped to some value.
boolean	**replace(K key, V oldValue, V newValue)** Replaces the entry for the specified key only if currently mapped to the specified value.
void	**replaceAll(BiFunction<? super K,? super V,? extends V> function)** Replaces each entry's value with the result of invoking the given function on that entry until all entries have been processed or the function throws an exception.
int	**size()** Returns the number of key-value mappings in this map.
Collection<V>	**values()** Returns a **Collection** view of the values contained in this map.

Nyttige metoder for HashSet-klassen:

boolean	**add(E e)** Adds the specified element to this set if it is not already present.
void	**clear()** Removes all of the elements from this set.
Object	**clone()** Returns a shallow copy of this HashSet instance: the elements themselves are not cloned.
boolean	**contains(Object o)** Returns true if this set contains the specified element.
boolean	**isEmpty()** Returns true if this set contains no elements.
Iterator<E>	**iterator()** Returns an iterator over the elements in this set.
boolean	**remove(Object o)** Removes the specified element from this set if it is present.
int	**size()** Returns the number of elements in this set (its cardinality).
Spliterator<E>	**spliterator()** Creates a ***late-binding*** and *fail-fast* **Spliterator** over the elements in this set.

Nyttige metoder for Integer-klassen:

static int	**max**(int a, int b) Returns the greater of two int values as if by calling **Math.max**.
static int	**min**(int a, int b) Returns the smaller of two int values as if by calling **Math.min**.

static int	**parseInt**(String s) Parses the string argument as a signed decimal integer.

Nyttige metoder for Scanner-klassen:

void	**close**() Closes this scanner.

boolean	**hasNextBoolean**() Returns true if the next token in this scanner's input can be interpreted as a boolean value using a case insensitive pattern created from the string "true|false".
boolean	**hasNextByte**() Returns true if the next token in this scanner's input can be interpreted as a byte value in the default radix using the **nextByte()** method.
boolean	**hasNextByte**(int radix) Returns true if the next token in this scanner's input can be interpreted as a byte value in the specified radix using the **nextByte()** method.
boolean	**hasNextDouble**() Returns true if the next token in this scanner's input can be interpreted as a double value using the **nextDouble()** method.
boolean	**hasNextFloat**() Returns true if the next token in this scanner's input can be interpreted as a float value using the **nextFloat()** method.
boolean	**hasNextInt**() Returns true if the next token in this scanner's input can be interpreted as an int value in the default radix using the **nextInt()** method.
boolean	**hasNextInt**(int radix) Returns true if the next token in this scanner's input can be interpreted as an int value in the specified radix using the **nextInt()** method.
boolean	**hasNextLine**() Returns true if there is another line in the input of this scanner.
boolean	**hasNextLong**() Returns true if the next token in this scanner's input can be interpreted as a long value in the default radix using the **nextLong()** method.
boolean	**hasNextLong**(int radix) Returns true if the next token in this scanner's input can be interpreted as a long value in the specified radix using the **nextLong()** method.
boolean	**hasNextShort**() Returns true if the next token in this scanner's input can be interpreted as a short value in the default radix using the **nextShort()** method.
boolean	**nextBoolean**() Scans the next token of the input into a boolean value and returns that value.
byte	**nextByte**() Scans the next token of the input as a byte.
byte	**nextByte**(int radix) Scans the next token of the input as a byte.
double	**nextDouble**() Scans the next token of the input as a double.
float	**nextFloat**() Scans the next token of the input as a float.
int	**nextInt**() Scans the next token of the input as an int.
int	**nextInt**(int radix) Scans the next token of the input as an int.
String	**nextLine**() Advances this scanner past the current line and returns the input that was skipped.
long	**nextLong**() Scans the next token of the input as a long.
long	**nextLong**(int radix) Scans the next token of the input as a long.
short	**nextShort**() Scans the next token of the input as a short.

Når en skanner skal lese fra terminal:

```
Scanner scanner = new Scanner(System.in);
```

Nyttige metoder for File-klassen:

boolean	**canExecute()** Tests whether the application can execute the file denoted by this abstract pathname.
boolean	**canRead()** Tests whether the application can read the file denoted by this abstract pathname.
boolean	**canWrite()** Tests whether the application can modify the file denoted by this abstract pathname.
int	**compareTo(File pathname)** Compares two abstract pathnames lexicographically.
boolean	**createNewFile()** Atomically creates a new, empty file named by this abstract pathname if and only if a file with this name does not yet exist.
static File	**createTempFile(String prefix, String suffix)** Creates an empty file in the default temporary-file directory, using the given prefix and suffix to generate its name.
static File	**createTempFile(String prefix, String suffix, File directory)** Creates a new empty file in the specified directory, using the given prefix and suffix strings to generate its name.
boolean	**delete()** Deletes the file or directory denoted by this abstract pathname.
void	**deleteOnExit()** Requests that the file or directory denoted by this abstract pathname be deleted when the virtual machine terminates.
boolean	**equals(Object obj)** Tests this abstract pathname for equality with the given object.
boolean	**exists()** Tests whether the file or directory denoted by this abstract pathname exists.

Lage tester som printer fargekoder til terminal:

```
private static void testNavn(Legemiddel[] e, String[] navn) {
        System.out.println("Tester navn på legemidler:");
        for (int i = 0; i < e.length; i++) {
            System.out.printf("Kjører test nr %d: [",i+1);
            if (sjekkNavn(e[i], navn[i])) {
                System.out.printf("\u001B[32mPASS\u001B[0m");
            } else {
                System.out.printf("\u001B[31mFAIL\u001B[0m");
            }
            System.out.printf("] Fikk: %s\tForventet: %s\n",e[i].hentNavn(),
navn[i]);
        }
    }
```

Eksempel liste:

```
interface Liste <E> extends Iterable<E> {
    int stoerrelse ();
    void leggTil (E x);
    E hent ();
    E fjern ();
}
```

Eksempel Lenkeliste:

```
import java.util.Iterator;
import java.util.NoSuchElementException;


abstract class Lenkeliste <E> implements Liste<E> {
```

```
    protected int stoerrelse;
    //Invariant: Antall objekter i listen er alltid lik stoerrelse

    class Node {
        Node neste = null;
        Node forrige = null;
        E data;

        public Node(E x) {  data = x;   }
    }

    @Override
    public Iterator<E> iterator() {
        return new LenkelisteIterator();
    }


    class LenkelisteIterator implements Iterator<E> {

        private Node sjekker;

        public LenkelisteIterator() {
            sjekker = start;
        }

        @Override
        public boolean hasNext() {
            return sjekker != null;
        }

        @Override
        public E next() { //forstår ikke hvorfor det står T next() i oppgaven,
skal vel bruke E
            if (!hasNext()) {
                throw new NoSuchElementException(); //Bruker denne exception
ettersom det står i Oracle sine dokumenter
            }
            E data = sjekker.data;
            sjekker = sjekker.neste;
            return data;

        }
    }



    protected Node start = null;
    protected Node slutt = null;
```

```
    //Start og slutt vil alltid peke til null eller det første og siste
objektet respektivt



    public int stoerrelse() {
        return stoerrelse;
    }



    public void leggTil(E x) {
        Node ny = new Node(x);
        if (start == null) {
            start = ny;
            slutt = ny;
        }

        else if (slutt == start) {
            slutt = ny;
            start.neste = slutt;
            slutt.forrige = start;
        }

        else {
            slutt.neste = ny;
            ny.forrige = slutt;
            slutt = ny;
        }

        stoerrelse++;
    }
    //Invarianten gjelder


    public E hent() {
        if (start == null) {
            return null;
        }
        return start.data;
    }


    public E fjern() {
        if (start == null) {
            throw new UgyldigListeindeks(-1);
        }
        E ut = start.data;
        start = start.neste;
```

```
        stoerrelse--;
        return ut;
    }
    //Invarianten gjelder

    @Override
    public String toString() {
        String s = "";
        Node sjekk = start;
        int teller = 0;


        while (sjekk != null) { //Foretrekker å printe ut ett element per
linje for god oversikt
            s = String.format("%sElement [%d] --> %s\n", s, teller,
sjekk.data);
            teller++;   sjekk = sjekk.neste;
        }

        return s;
    }


}
```

Eksempel IndeksertListe:

```
class IndeksertListe <E> extends Lenkeliste<E> {


    public void leggTil (int pos, E x) {
        Node ny = new Node(x);
        if (pos > stoerrelse || pos < 0) { //Tillater pos å være like stor som
størrelse siden vi kan putte inn noe på slutten av listen
            throw new UgyldigListeindeks(pos);
        }


        if (start == null && pos == 0) { //kjører kun for tomme lister
            start = ny;
            slutt = ny;
        }

        else if (pos == 0) { //Kjører når noe skal i indeks 0
            ny.neste = start;
            start.forrige = ny;
            start = ny;
        }
```

```
        else if (pos == stoerrelse) { //kjører når noe skal til slutt
            ny.forrige = slutt;
            slutt.neste = ny;
            slutt = ny;
        }
        //De neste to betingelsene er bare for å a mest effektiv kode

        else if (pos < (stoerrelse/2)) { //Om pos er mindre en halve
størrelsen av listen starter vi fra starten
            Node sjekk = start;
            for (int i = 0; i < pos; i++) {
                sjekk = sjekk.neste;
            }
            ny.neste = sjekk;
            ny.forrige = sjekk.forrige;
            sjekk.forrige = ny;
            ny.forrige.neste = ny;
        }

        else { //Om pos er større en halve størrelsen av listen går vi fra
slutten og bakover
            Node sjekk = slutt;
            for (int i = stoerrelse; i > pos; i--) {
                sjekk = sjekk.forrige;
            }
            ny.forrige = sjekk;
            ny.neste = sjekk.neste;
            sjekk.neste = ny;
            ny.neste.forrige = ny;
        }

        stoerrelse++;
    }


    public void sett(int pos, E x) {
        if (pos >= stoerrelse || pos < 0) {
            throw new UgyldigListeindeks(pos);
        }

        Node sjekk = start;
        for (int i = 0; i < pos; i++) {
            sjekk = sjekk.neste;
        }
        sjekk.data = x;
    }
```

```
    public E hent(int pos) {
        if (pos > stoerrelse || pos < 0) {
            throw new UgyldigListeindeks(pos);
        }
        Node sjekk = start;

        for (int i = 0; i < pos; i++) {
            sjekk = sjekk.neste;
        }

        return sjekk.data;
    }


    public E fjern(int pos) {
        if (start == null) {
            throw new UgyldigListeindeks(-1);
        }

        if (pos >= stoerrelse || pos < 0) {
            throw new UgyldigListeindeks(pos);
        }

        E ut = null;
        if (pos == 0) {
            ut = start.data;
            start = start.neste;
        }


        else if (pos == stoerrelse-1) { //Må ha minus en ettersom vi ser på
indekser (som starter på 0)
            ut = slutt.data;
            slutt = slutt.forrige;
        }


        else { //Loop som finner en det ønskede objektet og sletter det
            Node sjekk = start;
            for (int i = 0; i < pos; i++) {
                sjekk = sjekk.neste;
            }
            sjekk.neste.forrige = sjekk.forrige;
            sjekk.forrige.neste = sjekk.neste;
            ut = sjekk.data;
        }
        stoerrelse--;
        return ut;
    }
```

```
}
```

Eksempel Prioriteteskø:

```
class Prioritetskoe<E extends Comparable<E>> extends Lenkeliste<E> {
/*
    public int compareTo(E x) {
        return this.compareTo(x);
    }*/


    public void leggTil(E x) {
        Node ny = new Node(x);
        stoerrelse++;
        if (start == null) {
            start = ny;
            slutt = ny;
            return;
        }


        Node sjekk = start;

        while (x.compareTo(sjekk.data) > 0) { //Denne testen kjører til vi
finner en verdi som er mindre enn input
            if (sjekk.neste == null) { // Om vi er på slutten av listen så
finnes ingen lavere verdier
                sjekk.neste = ny;
                ny.forrige = sjekk;
                slutt = ny;
                return;
            }
            sjekk = sjekk.neste;

        }

        if (sjekk == start) { //Må ha denne i tilfelle input er større enn
første indeks
            ny.neste = sjekk;
            sjekk.forrige = ny;
            start = ny;
            return;
        }


        if (sjekk == slutt) { //Må ha denne i tilfelle input er større enn
siste indeks
            ny.forrige = sjekk.forrige;
            ny.forrige.neste = ny;
```

```
            ny.neste = sjekk;
            sjekk.forrige = ny;
            return;

        }


        ny.neste = sjekk; //I alle andre tilfeller fungerer denne koden helt
fint
        ny.forrige = sjekk.forrige;
        ny.forrige.neste = ny;
        sjekk.forrige = ny;


    }
}
```

Hvordan å dele opp Strings og legge dem i Array:

String[] liste = linje.trim().split(",");

Viktig catch å inkludere når man sender en fil inn i en Scanner:

```
} catch (FileNotFoundException e) {
            System.out.printf("Fant ikke filen. se feilmelding:\n%s\n", e);
        }
```

Boolsk metode å sjekke at en String matcher et spesifikt format:

navn.matches("[a-zA-Z]+")

fdnr.matches("[0-9]+")

Formater for printf:

Format Specifiers

Let's look at the available format specifiers available for `printf`:

- **%c** character
- **%d** decimal (integer) number (base 10)
- **%e** exponential floating-point number
- **%f** floating-point number
- **%i** integer (base 10)
- **%o** octal number (base 8)
- **%s** String
- **%u** unsigned decimal (integer) number
- **%x** number in hexadecimal (base 16)
- **%t** formats date/time
- **%%** print a percent sign
- **\%** print a percent sign

Escape Characters

Following are the escape characters available in `printf()`:

- \b backspace
- \f next line first character starts to the right of current line last character
- \n newline
- \r carriage return
- \t tab
- \\ backslash

Tråder:

Flettetråd:

```java
import java.io.File;
import java.util.HashMap;

public class FletteTrad implements Runnable{
    private Monitor2 monitor;
    private volatile boolean running = true;

    public FletteTrad(Monitor2 monitor) {
        this.monitor = monitor;
    }

    public void stopThread() {
        monitor.stop();
        running = false;
    }

    @Override
    public void run() {
        while (running) {
            HashMap<String, Subsekvens>[] hashes = null;
            try {
                hashes = monitor.getHash();
            } catch (InterruptedException e) {
                System.out.println("Noe gikk galt");
            }

            if (hashes == null) {
                break;
            }
            monitor.addHash(Monitor2.mergeHash(hashes[0], hashes[1]));
        }
    }
}
```

Lesetråd:

```
import java.io.File;
import java.util.HashMap;

public class FletteTrad implements Runnable{
    private Monitor2 monitor;
    private volatile boolean running = true;

    public FletteTrad(Monitor2 monitor) {
        this.monitor = monitor;
    }

    public void stopThread() {
        monitor.stop();
        running = false;
    }

    @Override
    public void run() {
        while (running) {
            HashMap<String, Subsekvens>[] hashes = null;
            try {
                hashes = monitor.getHash();
            } catch (InterruptedException e) {
                System.out.println("Noe gikk galt");
            }

            if (hashes == null) {
                break;
            }
            monitor.addHash(Monitor2.mergeHash(hashes[0], hashes[1]));
        }
    }
}
```

Monitor:

```
import java.util.concurrent.locks.Lock;
import java.util.concurrent.locks.ReentrantLock;
import java.util.concurrent.locks.Condition;
import java.util.HashMap;
import java.io.File;

public class Monitor2 {
    private SubsekvensRegister register;
    private Lock lock = new ReentrantLock();
    private Condition greaterThanOne = lock.newCondition();
    protected boolean running = true;
```

```
    public Monitor2() {
        register = new SubsekvensRegister();
    }

    void stop() {
        lock.lock();
        try {
            running = false;
            greaterThanOne.signalAll();
        }
        finally {
            lock.unlock();
        }
    }

    void addHash(HashMap<String, Subsekvens> hash) {
        lock.lock();
        try {
            register.addHash(hash);
            if (register.length() > 1) {
                greaterThanOne.signal(); //Signaliserer at det er 2+ hashmaps
i listen
            }
        }
        finally {
            lock.unlock();
        }
    }

    HashMap<String, Subsekvens>[] getHash() throws InterruptedException {
        lock.lock();
        try {

            while (register.length() <= 1 && running) { //Har med running her
slik at ingen tråder sitter fast i await
                greaterThanOne.await(); //Passiv venting til lengden er over
to
            }
            if (!running) {
                return null;
            }
            @SuppressWarnings("unchecked")
            HashMap<String, Subsekvens>[] hashes = new HashMap[2];

            hashes[0] = register.getHash(0);
            hashes[1] = register.getHash(0);

            return hashes;
        }
```

```
        finally {
            lock.unlock();
        }
    }

    HashMap<String, Subsekvens> getHash(int index) {
        lock.lock();
        try {
            return register.getHash(index);
        }
        finally {
            lock.unlock();
        }
    }

    int length() {
        lock.lock();

        try {
            return register.length();
        }
        finally {
            lock.unlock();
        }
    }
    static HashMap<String, Subsekvens> makeHash(File file) {
        return SubsekvensRegister.makeHash(file);
    }

    static HashMap<String, Subsekvens> mergeHash(HashMap<String, Subsekvens>
hash1, HashMap<String, Subsekvens> hash2) {
        return SubsekvensRegister.mergeHash(hash1, hash2);
    }
}
```

Bruke tråder i en oppgave:

```
import java.util.HashMap;
import java.io.File;
import java.util.Scanner;
import java.io.FileNotFoundException;
import java.util.ArrayList;

class Oblig5Hele {
    public static void main(String[] args) {
        if (args.length != 1) {
            System.err.println("Vennligst kun legg ved (KUN) en mappelokasjon
for filer!");
            System.exit(1);
```

```
        }

        long start = System.currentTimeMillis();

        HashMap<String, Subsekvens> hashSyke = new HashMap<>();
        HashMap<String, Subsekvens> hashFriske = new HashMap<>();
        File fil = null;
        String filnavn = null;
        boolean fulltFilNavn = false;

        //Gir muligheten til å skrive inn enten hele filplasseringen eller å
skrive
        //Kun navnet på mappen gitt at den ligger i samme overmappe som denne
mappen.
        if (!(args[0].matches("[a-zA-Z]+"))) {
            filnavn = args[0] + "/metadata.csv";
            fulltFilNavn = true;
        }
        else {
            filnavn = "../" + args[0] + "/metadata.csv";
        }

        try {
            fil = new File(filnavn);

        }
        catch (NullPointerException e) {
            System.err.printf("MANGLER FILPLASSERING! \nfeilkode: %s\n", e);
            System.exit(1);
        }

        Monitor2 syke = new Monitor2();
        Monitor2 friske = new Monitor2();
        Scanner scanner = null;

        try {
            scanner = new Scanner(fil);
        }
        catch (FileNotFoundException e) {
            System.err.printf("Filen eksisterer ikke. \nfeilkode: %s\n", e);
            System.exit(1);
        }

        //Lager to Array med tråder, et for syke og et for friske:
        Thread[] sykeFletteTrad = new Thread[8];
        FletteTrad[] sykeFlette = new FletteTrad[8];

        Thread[] friskeFletteTrad = new Thread[8];
        FletteTrad[] friskeFlette = new FletteTrad[8];
```

```
        for (int i = 0; i < 8; i++) {
            //Grunnen til at jeg lager først flettetrådene og så sender det
inn i
            //tråd-klassen er fordi jeg kjører lesing og fletting fra filene
samtidig.
            //Dermed må jeg ha spesiell implementasjon for å slutte
flettetrådene når
            //jeg er ferdig å lese filer. For å gjøre det må jeg ha tilgang
til mine
            //Runnable-funksjoner som jeg ikke har tilgang til om jeg sender
inn direkte i thread.
            sykeFlette[i] = new FletteTrad(syke);
            sykeFletteTrad[i] = new Thread(sykeFlette[i]);
            sykeFletteTrad[i].start();

            friskeFlette[i] = new FletteTrad(friske);
            friskeFletteTrad[i] = new Thread(friskeFlette[i]);
            friskeFletteTrad[i].start();
        }

        System.out.println("FletteTråder startet");
        //Lager så lesetrådene:
        ArrayList<Thread> leseTraader = new ArrayList<>();

        while (scanner.hasNextLine()) {
            String[] hentFilNavn = scanner.nextLine().split(",");
            Monitor2 monitor = null; //skal peke til den gjeldende monitoren

            if (fulltFilNavn) {
                filnavn = args[0] + "/" + hentFilNavn[0];
            }
            else {
                filnavn = "../" + args[0] + "/" + hentFilNavn[0];
            }


            if (hentFilNavn[1].contains("True")) {
                monitor = syke;
            }
            else {
                monitor = friske;
            }

            //Lager og starter lesetrådene fortløpende
            leseTraader.add(new Thread(new LeseTrad(filnavn, monitor)));
            leseTraader.get(leseTraader.size()-1).start();
        }
        System.out.println("Alle leseTråder startet");
```

```
        //Fletter så trådene sammen:
        int leseTraaderStoerrelse = leseTraader.size()-1;
        for (int i = 0; i < leseTraader.size(); i++) {
            try {
                leseTraader.get(i).join();
                System.out.printf("Lesetråd nummer [%d/%d] avsluttet\n", i,
leseTraaderStoerrelse);
            } catch (Exception e){
                System.err.printf("Et problem oppstod under kombinering av
filene\n%s", e);
            }
        }
        System.out.println("Alle leseTråder avsluttet");

        while (syke.length() != 1 && friske.length() != 1) {
            //Kjører litt aktiv venting til flettetrådene er ferdig
            System.out.println("Venter");
        }

        for (int i = 0; i < sykeFlette.length; i++) {
            sykeFlette[i].stopThread();
            friskeFlette[i].stopThread();
        }


        hashSyke = syke.getHash(0);
        hashFriske = friske.getHash(0);

        String sekvens = null;
        int verdiSyke = 0;
        int verdiFriske = 0;
        int maksdiff = 0;

        for (String s : hashSyke.keySet()) {
            if (!(hashFriske.containsKey(s))) {
                verdiFriske = 0;
            }
            else {
                verdiFriske = hashFriske.get(s).hentForekomster();
            }


            verdiSyke = hashSyke.get(s).hentForekomster();


            //Sjekker om verdiene er større og skriver isåfall ut
            //Den under kan brukes i TestData eller TestDataLiten om ønsket
            /*if (verdiSyke > verdiFriske) {
```

```
                System.out.printf("Sekvensen %s forekommer %d ganger hos
friske.\nSekvensen %s forekommer %d ganger hos syke.\n", s, verdiFriske, s,
verdiSyke);
            }*/

            if ((verdiSyke-verdiFriske) > 6) {
                System.out.printf("Sekvensen %s forekommer %d ganger hos
friske.\nSekvensen %s forekommer %d ganger hos syke.\n", s, verdiFriske, s,
verdiSyke);
            }

            if (sekvens == null || maksdiff < (verdiSyke-verdiFriske)) {
                sekvens = s;
                maksdiff = verdiSyke - verdiFriske;
            }

        }

        System.out.printf("\n\n\n\n\nDen største differensen i forekomster er
i sekvens %s.\nSyke hadde sekvensen %d hyppigere enn friske.\n", sekvens,
maksdiff);


        long finish = System.currentTimeMillis();
        long timeElapsed = finish - start;
        System.out.printf("Tid brukt i millisekunder: %d", timeElapsed);

    }
}
```

Subsekvenser og hashmaps:

```
import java.util.Scanner;
import java.io.File;
import java.io.FileNotFoundException;
import java.util.ArrayList;
import java.util.HashMap;


class SubsekvensRegister {
    ArrayList<HashMap<String, Subsekvens>> register = new ArrayList<>();

    void addHash(HashMap<String, Subsekvens> hash) {
        register.add(hash);
    }

    HashMap<String, Subsekvens> getHash(int index) {
        if (index >= register.size() || index < 0) {
            return null;
```

```
        }
        return register.remove(index);
    }


    int length() {
        return register.size();
    }

    static HashMap<String, Subsekvens> makeHash(File file) {
        HashMap<String, Subsekvens> map = new HashMap<>();
        try {
            Scanner scanner = new Scanner(file);
            while (scanner.hasNextLine()) {
                String line = scanner.nextLine();
                if (line.length() < 3) {
                    scanner.close();
                    System.err.println("Ugyldig DNA-sekvens, program
termineres");
                    System.exit(1);
                }

                for (int i = 0; i < line.length() - 2; i++) {
                    String sekvens = "" + line.charAt(i) + line.charAt(i+1) +
line.charAt(i+2);
                    Subsekvens subsekvens = new Subsekvens(sekvens);

                    map.put(sekvens, subsekvens);
                }
            }
        }
        catch (FileNotFoundException e) {
            System.err.printf("Filen eksisterer ikke. \nfeilkode: %s\n", e);
            System.exit(1);
        }
        return map;
    }

    static HashMap<String, Subsekvens> mergeHash(HashMap<String, Subsekvens>
hash1, HashMap<String, Subsekvens> hash2) {
        //Velger å sette hash2 inn i hash1
        for (String s : hash2.keySet()) {
            if (hash1.containsKey(s)) {
                hash1.get(s).endreForekomster(hash1.get(s).hentForekomster()+h
ash2.get(s).hentForekomster());
            }
            else {
                hash1.put(s, hash2.get(s));
            }
```

```
        }
        return hash1;
    }
}
```

GUI eksempel 1:

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

class GUI {
    JFrame vindu;
    JPanel celler;
    JPanel kontrollknapper;
    JLabel tekst;
    StartKnapp startknapp;
    SluttKnapp sluttknapp;
    int levendeCeller = 0;
    Kontroller kontroller;
    CelleKnapp[][] knapper;



    void lagFrame() {
        vindu = new JFrame("Game Of Life");
        vindu.setLocationRelativeTo(null);
        vindu.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        vindu.setLayout(new BorderLayout());

        celler = new JPanel();
        kontrollknapper = new JPanel();

        kontrollknapper.setLayout(new FlowLayout());
        startknapp = new StartKnapp(this, kontroller);
        startknapp.setFocusPainted(false);
        startknapp.setBorderPainted(false);
        startknapp.initGUI();

        tekst = new JLabel("Antall levende celler: ");
        kontrollknapper.add(tekst);
        kontrollknapper.add(startknapp);

        sluttknapp = new SluttKnapp();
        sluttknapp.initGUI();
        kontrollknapper.add(sluttknapp);

    }
```

```
    void lagCeller(Rutenett rutenett) {
        celler.setLayout(new GridLayout(rutenett.antRader,
rutenett.antKolonner));
        knapper = new CelleKnapp[rutenett.antRader][rutenett.antKolonner];

        for (int i = 0; i < rutenett.antRader; i++) {
            for (int j = 0; j < rutenett.antKolonner; j++) {
                CelleKnapp knapp = new CelleKnapp(rutenett.rutene[i][j],
this);
                knapper[i][j] = knapp;
                knapp.initGUI();
                celler.add(knapp);
                if (knapp.celle.erLevende()) {
                    levendeCeller++;
                }
            }
        }
        tekst.setText(tekst.getText() + levendeCeller);
        vindu.add(celler, BorderLayout.CENTER);
        vindu.add(kontrollknapper, BorderLayout.PAGE_START);
        vindu.pack();
    }

    void visVindu() {
        vindu.setVisible(true);
    }

    void oppdaterTekst() {
        tekst.setText("Antall levende celler: " + levendeCeller);
    }

    void settKontroller(Kontroller kontroller) {
        this.kontroller = kontroller;
        startknapp.kontroller = kontroller;
        sluttknapp.kontroller = kontroller;
    }


    void oppdatering() {
        for (CelleKnapp[] kol : knapper) {
            for (CelleKnapp knapp : kol) {
                knapp.oppdaterStatus();
            }
        }
        oppdaterTekst();
    }

}
```

```
class CelleKnapp extends JButton {
    Celle celle;
    GUI gui;

    CelleKnapp(Celle celle, GUI gui) {
        this.celle = celle;
        this.gui = gui;
        if (celle.erLevende()) {
            this.setText("*");
        }
        else {
            this.setText("");
        }
    }

    void endreStatus() {
        if (this.getText().equals("")) {
            this.setText("*");
            this.celle.settLevende();
            gui.levendeCeller++;
        }
        else {
            this.setText("");
            this.celle.settDoed();
            gui.levendeCeller--;
        }
        gui.oppdaterTekst();
    }

    void oppdaterStatus() {
        if (celle.erLevende()) {
            if (this.getText().equals("")) {
                gui.levendeCeller++;
            }
            this.setText("*");
        }
        else {
            if (this.getText().equals("*")) {
                gui.levendeCeller--;
            }
            this.setText("");
        }
    }

     class EndreKnapp implements ActionListener {
        @Override
        public void actionPerformed(ActionEvent e) {
            CelleKnapp knapp = (CelleKnapp) e.getSource();
```

```
            knapp.endreStatus();
        }
     }

     void initGUI() {
        addActionListener(new EndreKnapp());
     }
}

class StartKnapp extends JButton {
    GUI gui;
    boolean kjorer = false;
    Kontroller kontroller;

    StartKnapp(GUI gui, Kontroller kontroller) {
        super("Start");
        this.gui = gui;
        this.kontroller = kontroller;
        this.setBackground(Color.GREEN);
    }

    class StartStopp implements ActionListener {
        @Override
        public void actionPerformed(ActionEvent e) {
            StartKnapp knapp = (StartKnapp) e.getSource();

            if (knapp.kjorer) {
                knapp.kjorer = false;
                knapp.setText("Start");
                knapp.setBackground(Color.GREEN);
                kontroller.stopTimer();
            }
            else {
                knapp.kjorer = true;
                knapp.setText("Stopp");
                knapp.setBackground(Color.YELLOW);
                kontroller.startTimer();
            }
        }

    }

    void initGUI() {
        addActionListener(new StartStopp());
    }
}

class SluttKnapp extends JButton {
    Kontroller kontroller;
```

```
    SluttKnapp() {
        super("Avslutt");
        this.setBackground(Color.RED);
        this.setBorderPainted(false);
    }

    class Avslutt implements ActionListener {
        @Override
        public void actionPerformed(ActionEvent e) {
            SluttKnapp knapp = (SluttKnapp) e.getSource();
            knapp.kontroller.avslutt();
        }
    }

    void initGUI() {
        addActionListener(new Avslutt());
    }

}
```

GUI eksempel 2:

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

class Oblig7Gui {
    JFrame vindu;
    JPanel ruter;
    RuteKnapp[][] ruteknapper;
    Labyrint labyrint;


    void lagFrame() {
        vindu = new JFrame("Labyrint");
        vindu.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        vindu.setLayout(new BorderLayout());

        ruter = new JPanel();
    }


    void lagRuter(Labyrint labyrint) {
        this.labyrint = labyrint;
        ruter.setLayout(new GridLayout(labyrint.antRader,
labyrint.antKolonner));
        ruteknapper = new RuteKnapp[labyrint.antRader][labyrint.antKolonner];
    }
```

```
    void lagRuteKnapper() {
        for (int i = 0; i < labyrint.antRader; i++) {
            for (int j = 0; j < labyrint.antKolonner; j++) {
                RuteKnapp knapp = new RuteKnapp(labyrint.labyrint[i][j],
this);
                knapp.setPreferredSize(new Dimension(50, 50));
                knapp.initGUI();
                ruteknapper[i][j] = knapp;
                ruter.add(knapp);
            }
        }
    }

    void visVindu() {
        vindu.add(ruter, BorderLayout.CENTER);
        vindu.pack();
        vindu.setLocationRelativeTo(null);
        vindu.setVisible(true);
    }

    void visVei() {
        //System.out.println("Viser vei!");
        if (labyrint.kortesteVei.isEmpty()) return;
        int teller = 0;
        int stoerrelse = labyrint.kortesteVei.size();
        for (Rute rute : labyrint.kortesteVei) {
            ruteknapper[rute.rad][rute.kol].setBackground(new Color(100, (int)
(255*teller/stoerrelse), (int) (255*teller/stoerrelse)));
            teller++;
        }
    }

    void fjernVei() {
        if (labyrint.kortesteVei.isEmpty()) return;
        for (RuteKnapp[] kol : ruteknapper) {
            for (RuteKnapp knapp : kol) {
                knapp.oppdater();
            }
        }
    }

}


class RuteKnapp extends JButton {
    Rute rute;
    Labyrint labyrint;
    Oblig7Gui gui;
```

```
    RuteKnapp(Rute rute, Oblig7Gui gui) {
        this.rute = rute;
        labyrint = rute.labyrint;
        this.gui = gui;
        this.setFocusPainted(false);
        this.setBorderPainted(false);
        if (rute instanceof HvitRute) {
            this.setBackground(Color.WHITE);
        } else {
            this.setBackground(Color.BLACK);
            this.setEnabled(false);
        }
    }

    void oppdater() {
        if (rute instanceof HvitRute) {
            this.setBackground(Color.WHITE);
            rute.forrige = null;
            return;
        }
        //this.setBackground(Color.BLACK);
    }

    class EndreKnapp implements ActionListener {
        @Override
        public void actionPerformed(ActionEvent e) {
            RuteKnapp knapp = (RuteKnapp) e.getSource();
            if (knapp.rute instanceof SortRute) return;
            //System.out.println("Fjerner vei!");
            gui.fjernVei();
            //System.out.println("Ferdig å fjerne vei!");
            knapp.labyrint.kjorAStar(knapp.rute);
            //System.out.println("Ferdig å finne korteste rute");
            gui.visVei();
        }
    }
    void initGUI() {
        addActionListener(new EndreKnapp());
    }
}
```

Timer og TimerTask:

```
import java.util.Timer;
import java.util.TimerTask;
    void startTimer() {
        timer = new Timer();
        TimerTask task = new TimerTask() {
```

```
            @Override
            public void run() {
                oppdaterCeller();
            }
        };
        timer.scheduleAtFixedRate(task, 0, 20);
    }

    void stopTimer() {
        timer.cancel();
    }
```

Eksamen V19:

```
import java.util.NoSuchElementException;
import java.util.Iterator;

class Hund implements Comparable<Hund> {
    String navn;
    Kull mittKull;
    Tidspunkt minFodselstid;
    Hund neste = null;
    Hund (Kull k, String n, Tidspunkt fodt) {
        navn = n;
        mittKull = k;
        minFodselstid = fodt;
    }

    public Hund mor() {
    return mittKull.mor;
    }

    public Hund far() {
    return mittKull.far;
    }


    @Override
    public int compareTo(Hund h) {
        return this.minFodselstid.compareTo(h.minFodselstid);
    }


    public boolean erHelsosken(Hund h) {
        if (mittKull.mor == h.mittKull.mor && mittKull.far == h.mittKull.far
&& mittKull.mor != null && mittKull.far != null) {
            return true;
        }
```

```
        return false;
    }

    public boolean erHalvsosken(Hund h) {
        if (!erHelsosken(h)) {
            if ((mittKull.mor == h.mittKull.mor || mittKull.far ==
h.mittKull.far) && (mittKull.mor != null || mittKull.far != null)) {
                return true;
            }
        }
        return false;
    }


    public Hund finnEldsteKjenteOpphav() {
        if (mittKull.mor == null && mittKull.far == null) {
            return this;
        }
        Hund morSinEldsteFar = null;
        Hund morSinEldsteMor = null;
        if (mittKull.mor != null) {
            Hund mor = mittKull.mor;
            morSinEldsteMor = mor;
            if (mor.mittKull.mor != null) {
                morSinEldsteMor = mor.mittKull.mor.finnEldsteKjenteOpphav();
            }
            if (mor.mittKull.far != null) {
                morSinEldsteFar = mor.mittKull.far.finnEldsteKjenteOpphav();
            }

        }
        Hund farSinEldsteMor = null;
        Hund farSinEldsteFar = null;
        if (mittKull.far != null) {
            Hund far = mittKull.far;
            farSinEldsteFar = far;
            if (far.mittKull.mor != null) {
                farSinEldsteMor = far.mittKull.mor.finnEldsteKjenteOpphav();
            }
            if (far.mittKull.far != null) {
                farSinEldsteFar = far.mittKull.far.finnEldsteKjenteOpphav();
            }
        }

        Hund eldsteMorside = null;
        if (morSinEldsteFar != null) {
            if (morSinEldsteFar.compareTo(morSinEldsteMor) > 0) {
                eldsteMorside = morSinEldsteFar;
            }
```

```
            else {
                eldsteMorside = morSinEldsteMor;
            }
        }
        else {
            eldsteMorside = morSinEldsteMor;
        }

        Hund eldsteFarside = null;
        if (farSinEldsteMor != null) {
            if (farSinEldsteMor.compareTo(farSinEldsteFar) > 0) {
                eldsteFarside = farSinEldsteMor;
            }
            else {
                eldsteFarside = farSinEldsteFar;
            }
        }
        else {
            eldsteFarside = farSinEldsteFar;
        }

        if (eldsteMorside != null) {
            if (eldsteMorside.compareTo(eldsteFarside) > 0) {
                return eldsteMorside;
            }
            else {
                return eldsteFarside;
            }
        }
        return eldsteFarside;
    }

}

abstract class Kull implements Iterable<Hund> {
    Hund mor, far;

    Kull (Hund mor, Hund far) {
        this.mor = mor;
        this.far = far;
        }


    public void skrivUtAlle () {
        for (Hund h: this)System.out.println("  " + h.navn);
    }


    public abstract void settInn (Hund h);
```

```
    public abstract Iterator<Hund> iterator();

}


class Tidspunkt implements Comparable<Tidspunkt> {
    int[] tidspunkt = new int[6];

    Tidspunkt(int aar, int mnd, int dag, int time, int min, int sek) {
        tidspunkt[0] = aar;
        tidspunkt[1] = mnd;
        tidspunkt[2] = dag;
        tidspunkt[3] = time;
        tidspunkt[4] = min;
        tidspunkt[5] = sek;
    }


    @Override
    public int compareTo(Tidspunkt t2) {
        if (t2 == null) return 1;
        for (int i = 0; i < 6; i++) {
            if (this.tidspunkt[i] > t2.tidspunkt[i]) {
                return -1;
            }
            else if (this.tidspunkt[i] < t2.tidspunkt[i]) {
                return 1;
            }
        }
        return 0;
    }
}


class KullArray extends Kull {
    Hund[] array = new Hund[60];

    public KullArray(Hund mor, Hund far) {
        super(mor, far);
    }

    @Override
    public void settInn(Hund h) {
        int sek = h.minFodselstid.tidspunkt[5];
        if (array[sek] == null) {
            array[sek] = h;
            return;
        }
```

```
        Hund neste = array[sek];
        while (neste.neste != null) {
            neste = neste.neste;
        }
        neste.neste = h;
    }

    @Override
    public Iterator<Hund> iterator() {
        return new KullArrayIterator();
    }


    class KullArrayIterator implements Iterator<Hund> {

        private Hund sjekker;
        private int arrayPos = 0;

        KullArrayIterator() {
            sjekker = array[arrayPos];
            while (sjekker == null) {
                arrayPos++;
                sjekker = array[arrayPos];
            }
        }



        public boolean hasNext() {
            return sjekker != null;
        }


        public Hund next() {
            //System.out.println("dette skjer");
            if(!hasNext()) {
                throw new NoSuchElementException();
            }
            Hund denne = sjekker;

            sjekker = sjekker.neste;
            if (sjekker == null && arrayPos < 59) {
                arrayPos++;
                sjekker = array[arrayPos];
            }
            while (sjekker == null) {
                if (arrayPos == 59) {
                    return denne;
                }
```

```
                arrayPos++;
                sjekker = array[arrayPos];

            }
            return denne;
        }
    }
}


class KullListe extends Kull {
    Hund start = null;

    public KullListe(Hund mor, Hund far) {
        super(mor, far);
    }

    @Override
    public void settInn(Hund h) {
        if (start == null) {
            start = h;
            return;
        }

        else if (start.compareTo(h)< 0) {
            h.neste = start;
            start = h;
            return;
        }

        Hund neste = start;
        while (neste != null) {
            if (neste.neste == null) {
                neste.neste = h;
                return;
            }

            if (neste.neste.compareTo(h) < 0) {
                h.neste = neste.neste;
                neste.neste = h;
                return;
            }
            neste = neste.neste;

        }
    }

    @Override
    public Iterator<Hund> iterator() {
```

```
        return new KullListeIterator();
    }

    class KullListeIterator implements Iterator<Hund> {

        private Hund sjekker;

        KullListeIterator() {
            sjekker = start;
        }


        public boolean hasNext() {
            return sjekker != null;
        }


        public Hund next() {
            if(!hasNext()) {
                throw new NoSuchElementException();
            }
            Hund denne = sjekker;
            sjekker = sjekker.neste;
            return denne;
        }
    }
}
```