Geavanceerd Objectgeoriënteerd Programmeren

Bijeenkomst 4

Stijn de Gouw

Open Universiteit
www.ou.nl

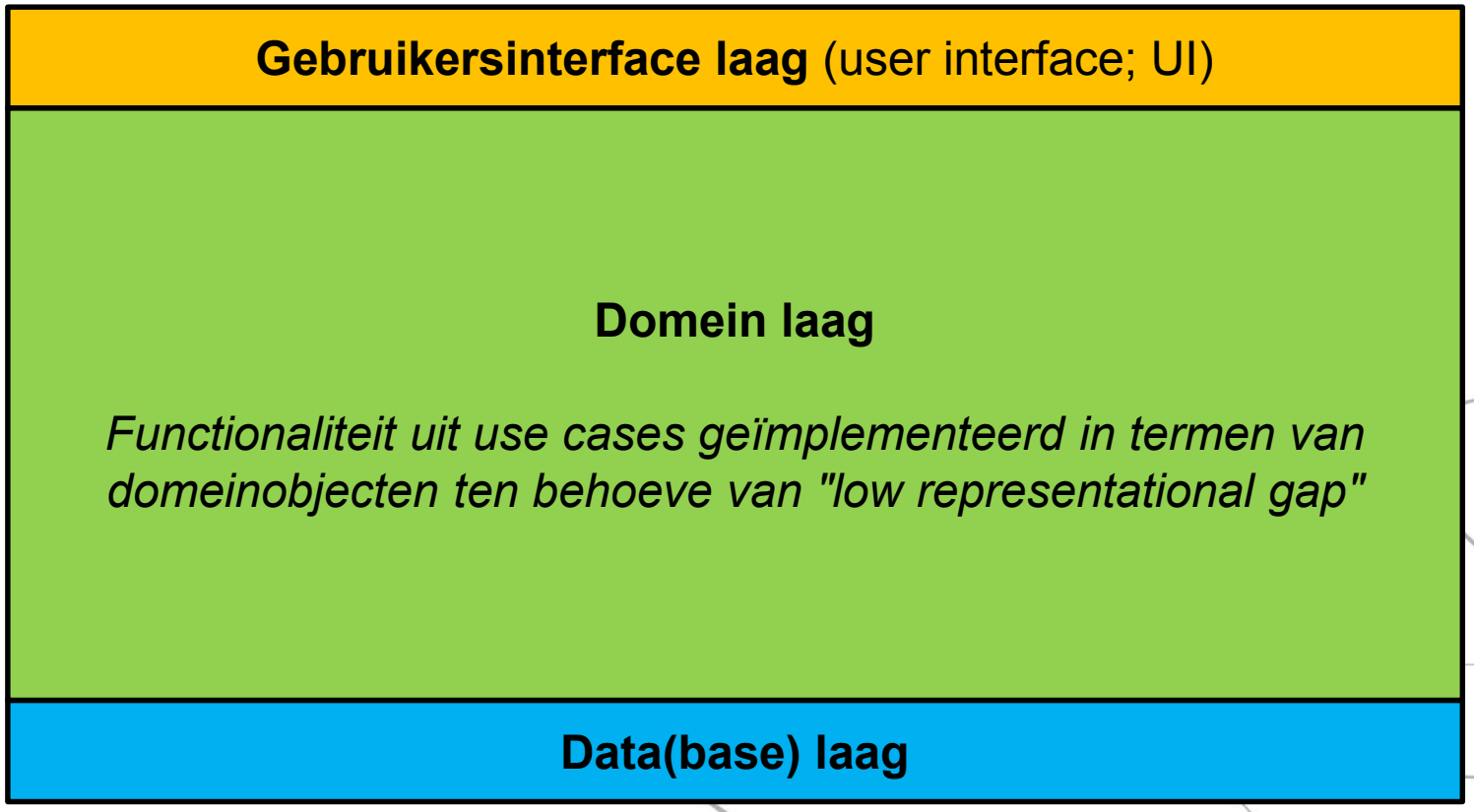
Programma

Activiteit	Datum	Leereenheden	Opdracht
~~Bijeenkomst 1~~	~~04-04-2019~~	~~1*, 2, 3 [18u]~~	
~~Bijeenkomst 2~~	~~18-04-2019~~	~~4, 5* [14u]~~	
~~Bijeenkomst 3~~	~~09-05-2019~~	~~6, 7 [13u]~~	~~1 (definitief)~~ ~~2 (voorlopig)~~
Bijeenkomst 4	23-05-2019	8, 9* [15u]	2 (definitief) 3 (voorlopig)
Bijeenkomst 5	06-06-2019	10*,11*,12 [21u]	3 (definitief) 4 (voorlopig)
Toets	20-06-2019		

(*) Zie rooster op yOUlearn voor toelichting

Open Universiteit
www.ou.nl

Programma



Open Universiteit
www.ou.nl

Threads

LEEREENHEID 8

Programma

- Inleiding
- Theorie over threads
  - achtergrond
  - waarom en wanneer gebruiken
  - problemen en oplossingen
- Threads in Java

Open Universiteit
www.ou.nl

Theorie over *threads*

Inleiding

- **Sequentiële programma's**
  - "sequentieel" = "in een reeks"
  - Eén activiteit tegelijk uitvoeren

Inleiding

- **Sequentiële programma's**
  - "sequentieel" = "in een reeks"
  - Eén activiteit tegelijk uitvoeren

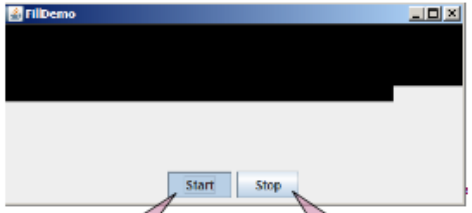
Wat is concurrency?

- **Sequentiële programma's**
  - "sequentieel" = "in een reeks"
  - Eén activiteit tegelijk uitvoeren

- **Concurrent programma's**
  - "concurrent" = "gelijktijdig" (geen gemeengoed)
  - Meerdere activiteiten tegelijk uitvoeren
    - Wij kunnen lopen, praten, ademen, kijken, horen, ruiken …. Allemaal tegelijkertijd
    - Computers kunnen dit ook: downloaden file, printen file, ontvangen email, klok runnen …. min of meer parallel

Motivatie

1. **Responsiviteit**
   - Zo snel mogelijk op de gebruiker reageren door achtergrondactiviteiten gelijktijdig uit te voeren
   - **Voorbeeld:** Spellchecken, GUIs, webapplicaties
     - Sequentieel: Activiteiten één-voor-één uitvoeren
     - Concurrent: Activiteiten gelijktijdig uitvoeren



Startknop: Scherm vult zich geheel met zwarte blokjes.

Stopknop: Werkt pas als het programma (het vullen met zwarte blokjes) is afgerond!

Motivatie

2. Efficiëntie

- Twee activiteiten tegelijk = Twee keer zo snel klaar (*Echt waar..?*)

- **Voorbeeld:** Bestand zoeken
  - Sequentieel: Alle bestanden en mappen één-voor-één aflopen
  - Concurrent: Alle bestanden één-voor-één aflopen, maar nieuwe zoekactiviteit starten voor elke map

Motivatie

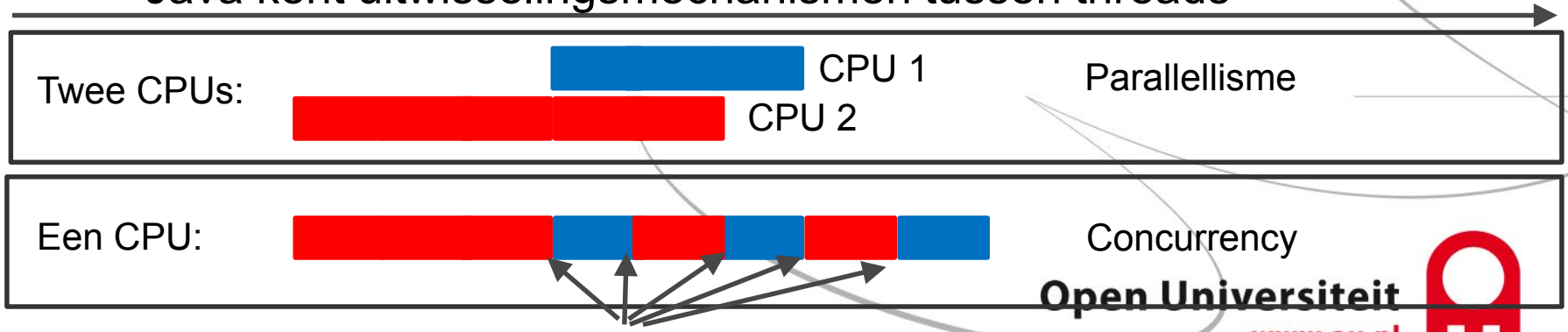
3. Natuurlijke modellering

- Sommige concepten in de werkelijkheid zijn natuurlijker te representeren met concurrency

- **Voorbeeld:** Banksimulatie, restaurant (practicum opdracht)
  - Sequentieel: Rekeninghouders één-voor-één bankactiviteiten laten uitvoeren
  - Concurrent: Bankactiviteiten van verschillende rekeninghouders gelijktijdig uitvoeren

Threads

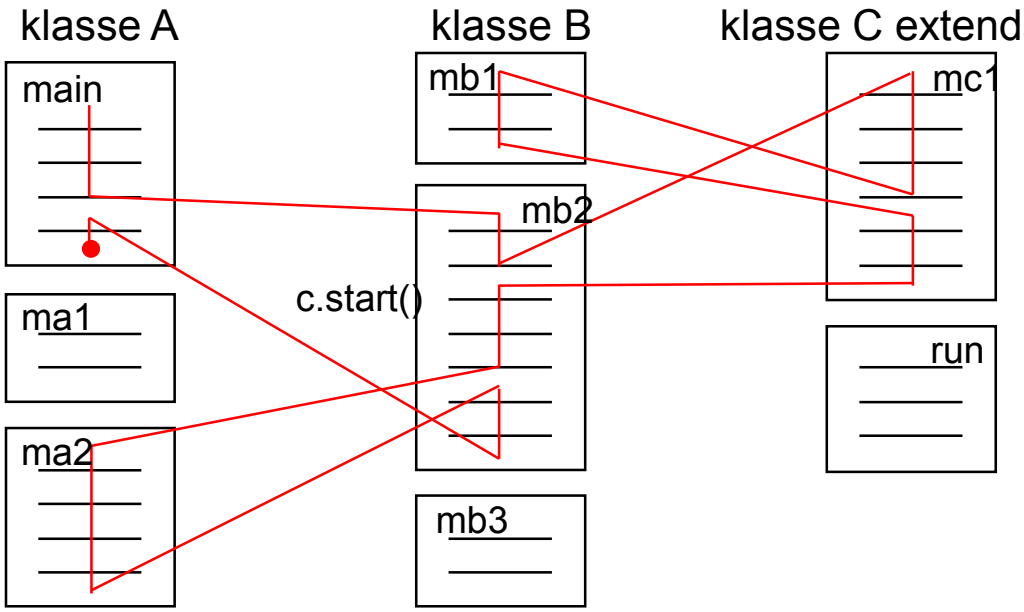
- Veelgebruikt: Java, (Objective-)C(++), Swift, Cobol, Python, ...
- Threads in Java maken het mogelijke meerdere taken “tegelijkertijd” te laten uitvoeren:
- Het lijkt alsof de verschillende threads gelijktijdig worden uitgevoerd, en bij multicore machine kan dat ook zo zijn
- De JVM bevat een scheduler die de processortijd verdeelt over de verschillende threads in een Java-programma
- Java kent uitwisselingsmechanismen tussen threads



Wat is een thread?

- Een thread is een sequentiële uitvoering van de instructie van een programma
- Deze opdrachten kunnen zich in verschillende objecten van verschillende klassen bevinden
- Gedeeld geheugen
  - Simpele vorm van communicatie tussen threads
  - Maar moeilijk (foutgevoelig) om mee te programmeren
- Elk programma heeft minstens een main-thread, die de main methode uitvoert
- Iedere thread heeft zijn eigen callstack
- Applicatie met *n* threads benut maximaal *n* CPU cores

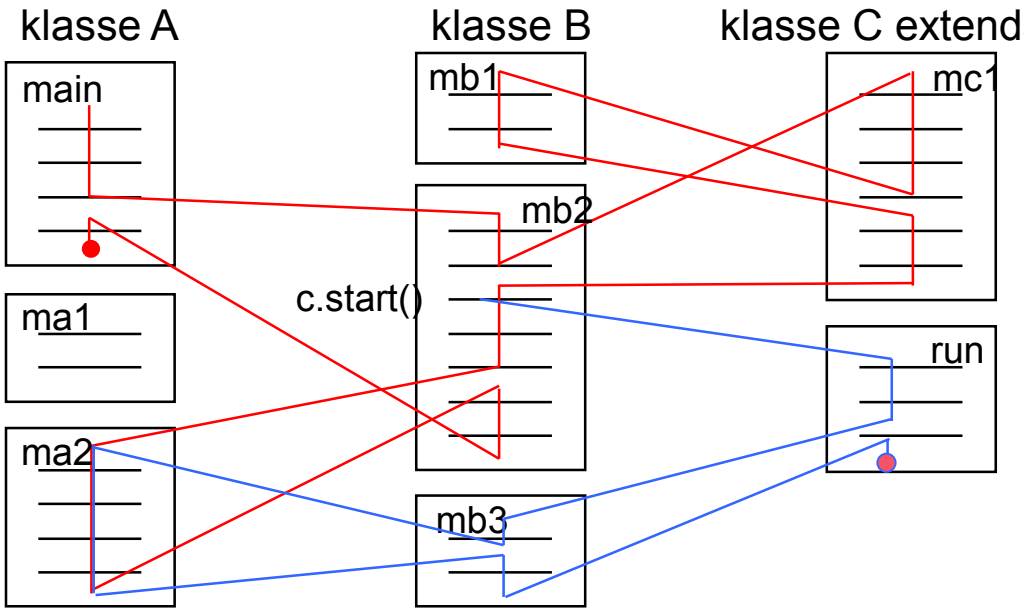
Voorbeeld threads



– JVM start main in eigen draad (rood)

– Hoofddraad is afgelopen als main is afgelopen

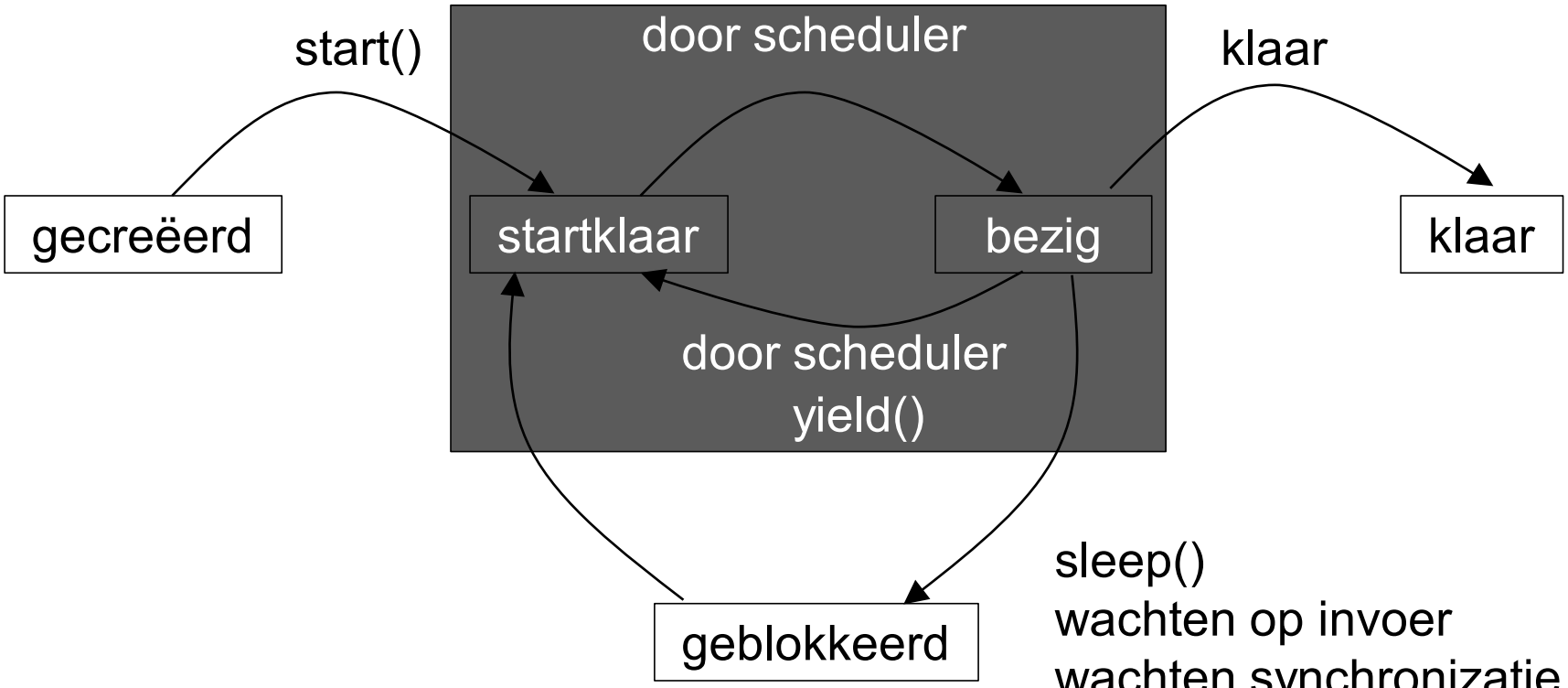
Voorbeeld threads



- JVM start main in eigen draad (rood)
- c.start() zorgt voor starten tweede control-flow draad (blauw)
- De opdrachten uit die draad worden ‘tegelijk’ met die uit de hoofddraad uitgevoerd

- Hoofddraad is afgelopen als main is afgelopen
- Hulpdraad is afgelopen als run is afgelopen
- Er zijn twee callstacks!

Levensloop van een draad



Interferentie

- i++ is niet atomair:
  LOAD @i, r0; load the value of 'i' from memory into register r0
  ADD r0, 1; increment the value in register r0
  STORE r0, @i; write the updated value back to memory

T1	T2	R/W	i
			0
Read		←	0
+ 1			0
Write		→	1
	Read	←	1
	+ 1		1
	Write	→	2

T1	T2	R/W	i
			0
Read		←	0
	Read	←	0
+ 1			0
	+ 1		0
Write		→	1
	Write	→	1

Open Universiteit
www.ou.nl

Opgave

Gegeven 4 integer variabelen: a, b, x, y, allen met beginwaarde 0. Welke waarden zijn mogelijk voor **x, y** na uitvoering van onderstaande threads?

Thread 1

```
a = 1;
x = b;
```

Thread 2

```
b = 1;
y = a;
```

x	y	mogelijk?
0	0	
0	1	
1	0	
1	1	

Uitwerking

Gegeven 4 integer variabelen: a, b, x, y, allen met beginwaarde 0. Welke waarden zijn mogelijk voor **x, y** na uitvoering van onderstaande threads?

Thread 1

```
a = 1;
x = b;
```

Thread 2

```
b = 1;
y = a;
```

x y	mogelijk?
0 0	nee?
0 1	ja, bv volgorde main mem: a=1; x=b; b=1; y=a;
1 0	ja, bv volgorde main mem: b=1; y=a; a=1; x=b;
1 1	ja, bv volgorde main mem: a=1; b=1; x=b; y=a;

Uitwerking

Gegeven 4 integer variabelen: a, b, x, y, allen met beginwaarde 0. Welke waarden zijn mogelijk voor **x, y** na uitvoering van onderstaande threads?

Thread 1

```
a = 1;
x = b;
```

Thread 2

```
b = 1;
y = a;
```

x y	mogelijk?
0 0	ja! bv reordering, of als variabele a in CPU core cache zit. Volgorde main mem: x=b; b=1; y=a; a=1;
0 1	ja, bv volgorde main mem: a=1; x=b; b=1; y=a;
1 0	ja, bv volgorde main mem: b=1; y=a; a=1; x=b;
1 1	ja, bv volgorde main mem: a=1; b=1; x=b; y=a;

Terminologie

- **Thread-safe:** Een klasse is thread-safe als elke instantie zich correct gedraagt als hij in meerdere threads gebruikt wordt, *onafhankelijk van timing/interleaving*
  - Correct betekent: aan alle specificaties (klasse-invarianten, contracten van alle methoden) wordt voldaan
  - Specificaties staan vaak (informeel) in Javadoc, of zijn daaruit af te leiden. Bijvoorbeeld: ArrayList invariant: size ≥0.

```
/**
 * The size of the ArrayList (the number of elements it contains).
 */
private int size;
```

- **Race conditie:** situatie dat de correctheid afhangt van de timing of interleaving van threads tijdens uitvoering

Race conditie voorbeeld

- Stel var is een gedeelde variabele. Is onderstaande code thread-safe?

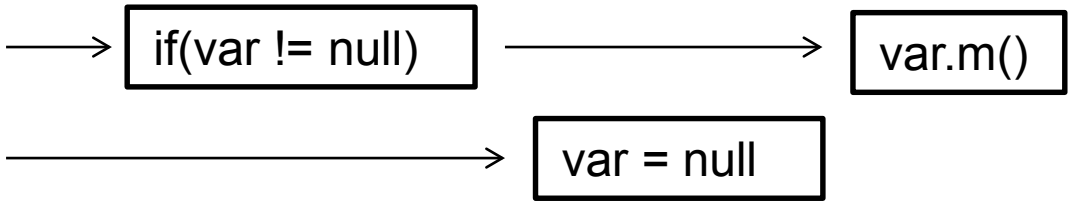
```
if (var !=null)  {   // Check
   var.m();          // Act
}
```


Open Universiteit
www.ou.nl

Race conditie voorbeeld

- ‘check-then-act’ problemen:

```
if (var !=null)  {   // Check
   var.m();          // Act
}
```

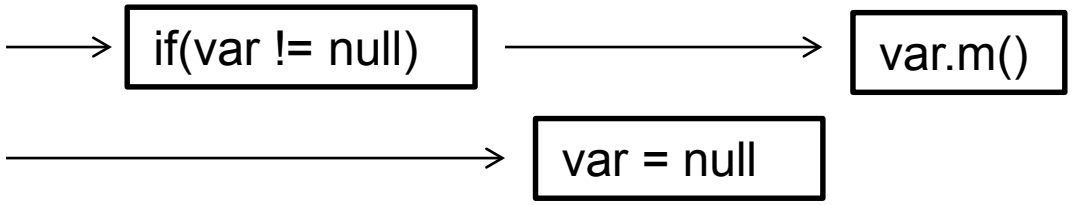


Open Universiteit
www.ou.nl

Race conditie voorbeeld

- ‘check-then-act’ problemen:

```
if (var !=null)  {   // Check
   var.m();          // Act
}
```



- Oplossing: synchronisatie
  - Mechanisme om reeksen van statements atomair (= zonder interferentie) uit te voeren
    - Afdwingen dat i++ atomair wordt uitgevoerd
    - Afdwingen dat check en act samen atomair worden uitgevvoeren
  - Biedt programmeurs *enige* controle over de scheduler ➔ Nondeterminisme indammen!
  - Meest gebruikte synchronisatie optie: locking

Open Universiteit
www.ou.nl

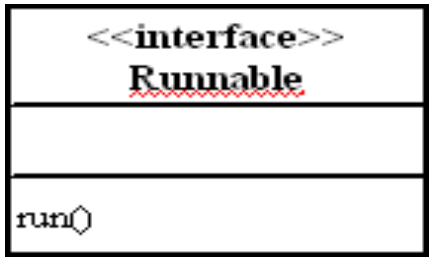
Threads in Java

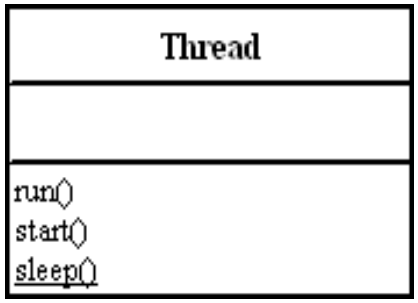
Open Universiteit
www.ou.nl

Threads maken en starten

Bedenk welke opdrachten je in een aparte draad wilt uitvoeren

1. Maak nieuwe Runnable subklasse met opdrachten in run()





2. Of: herdefinieer run() in nieuwe subklasse van klasse Thread

- run() is voor extra draad wat main() is voor hoofddraad
- Bij optie 1: creëer Runnable en nieuwe thread, en geef runnable mee aan thread constructor: Thread t = **new** Thread(new Crunnable());
- Bij optie 2: instantieer nieuwe subklasse: Thread t = **new** SubThread();
- Start draad door aanroep van methode start(): t.start();
- Nieuwe draad is klaar als run klaar is

Threads maken en starten - voorbeeld

```
public class TelAf implements Runnable {
  private int n = 0;

  public TelAf(int n) {
    this.n = n;
  }

  public void run() {
    while (n > 0) {
      n = n-1;
      if (n % 100 == 0) {
        System.out.println("n=" + n);
      }
    }
  }
}
```

```
public class Rekenmachine {

  public static void main(String[] args) {
    TelAf taf = new TelAf(10000);
    Thread t = new Thread(taf);
    t.start();
    System.out.println("main klaar");
  }

}
```


Open Universiteit
www.ou.nl

Threads “pauzeren”

- Klasse Thread heeft statische methode sleep

```
try {
  Thread.sleep(1000);
} catch (InterruptedException exc) {…}
```

- Laat huidige draad gedurende aangegeven aantal milliseconden wachten en geeft processor vrij voor andere draden
- Kan gebruikt worden om taken in bepaalde snelheid uit te voeren
- Houdt rekening met het optreden van interrupts:
  - sleep gooit checked exceptie InterruptedException

OPGAVE 1

Threads stoppen

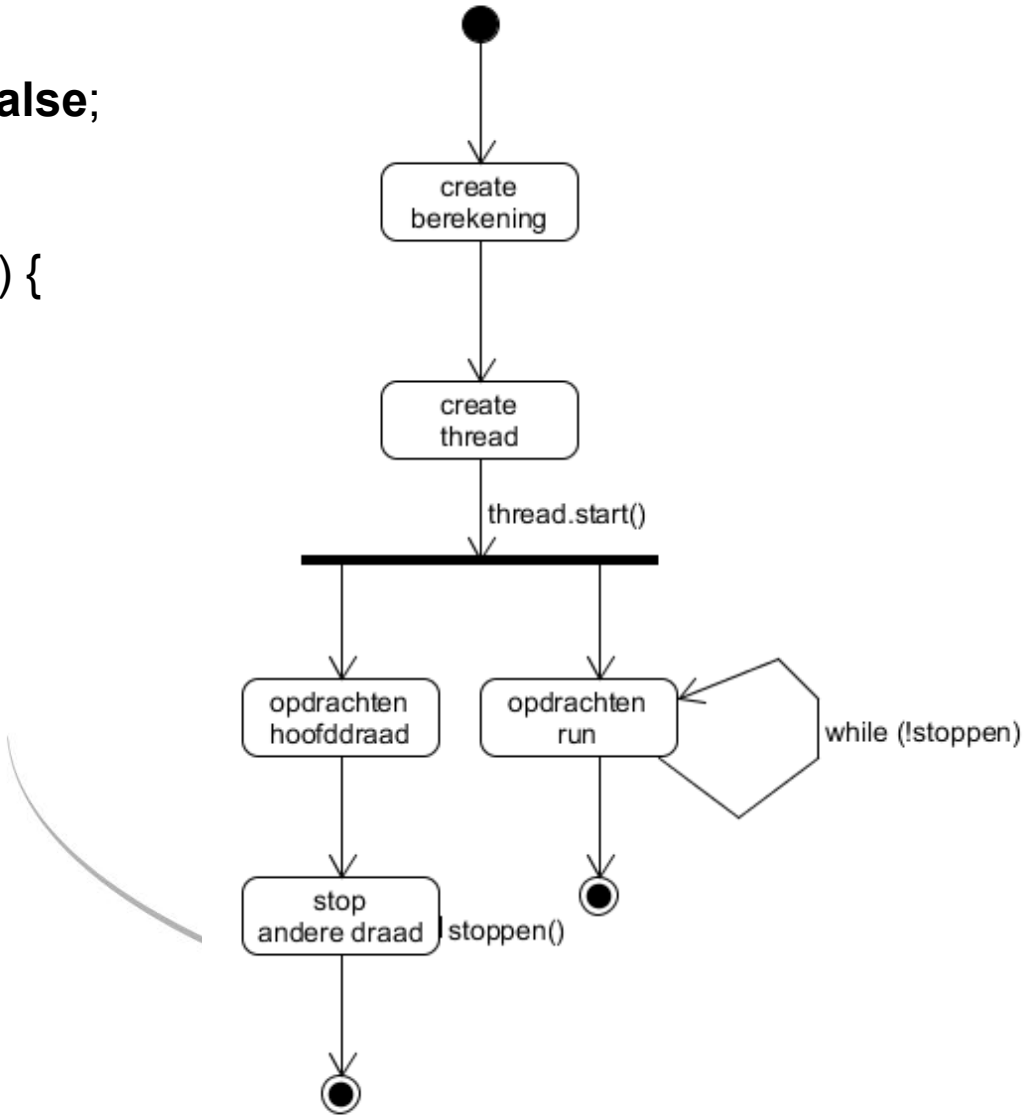
- Als run beëindigd is, is de thread automatisch gestopt
- Soms zal run een oneindige lus bevatten zodat draad niet vanzelf stopt: while(true) { …}
- Meestal in zo'n lus: Thread.sleep(SLAAPTIJD);
- Als andere draad controle wil hebben over moment van stoppen:
  - Methode Thread.stop() is deprecated; niet gebruiken
  - In plaats daarvan Futures, of lus in run met gedeelde var stoppen:

    while (!stoppen) {…}

    En introduceer een methode om stoppen te veranderen

Threads stoppen

```
private boolean stoppen = false;


public void run(String[] args) {
  while (!stoppen) {
    …
  }
}
public void stoppen() {
  stoppen = true;
}
```



Onverwacht gedrag

```
private boolean stoppen = false;

private void stopKnopAction() {
  stoppen();
  System.out.println("StopKnopAction:  " + stoppen);
}

public void run() {
  stoppen = false;
  waardeLabel.setText("");
  tijdLabel.setText("");
  long starttijd = System.currentTimeMillis();
  long teller = 0;
  while (!stoppen && teller < MAXTELLER) {
    teller++;
  }
  long eindtijd = System.currentTimeMillis();
  waardeLabel.setText("teller = " + teller);
  tijdLabel.setText("tijd: " + (eindtijd - starttijd) / 1000 + " seconden");
}

private  void stoppen() {
  stoppen = true;
}
```

In draad 1

In draad 2

Onverwacht gedrag

Exam

Waarde van attribuut stoppen lijkt niet in draad van run door te dringen

- Mogelijke verklaring
  - Compiler ziet in run lus waarin attribuut stoppen niet wijzigt
  - Compiler optimaliseert door eenmalig de waarde van stoppen te lezen en in cache van draad te stoppen
  - Wanneer andere draad stoppen wijzigt ziet de run-draad dit niet
  - Compilerversie en JVM afhankelijk wat er exact gebeurt
    - Vaak werkt het weer wel door een sleep toe te voegen aan lus, maar dit is niet gegarandeerd

Oplossing

- Maak dit soort variabelen volatile

  private volatile boolean stoppen;

- Declaring a volatile Java variable means:
  “The value of this variable will never be cached thread-locally: alle reads and writes will go straight to main-memory.”

OPGAVE 2

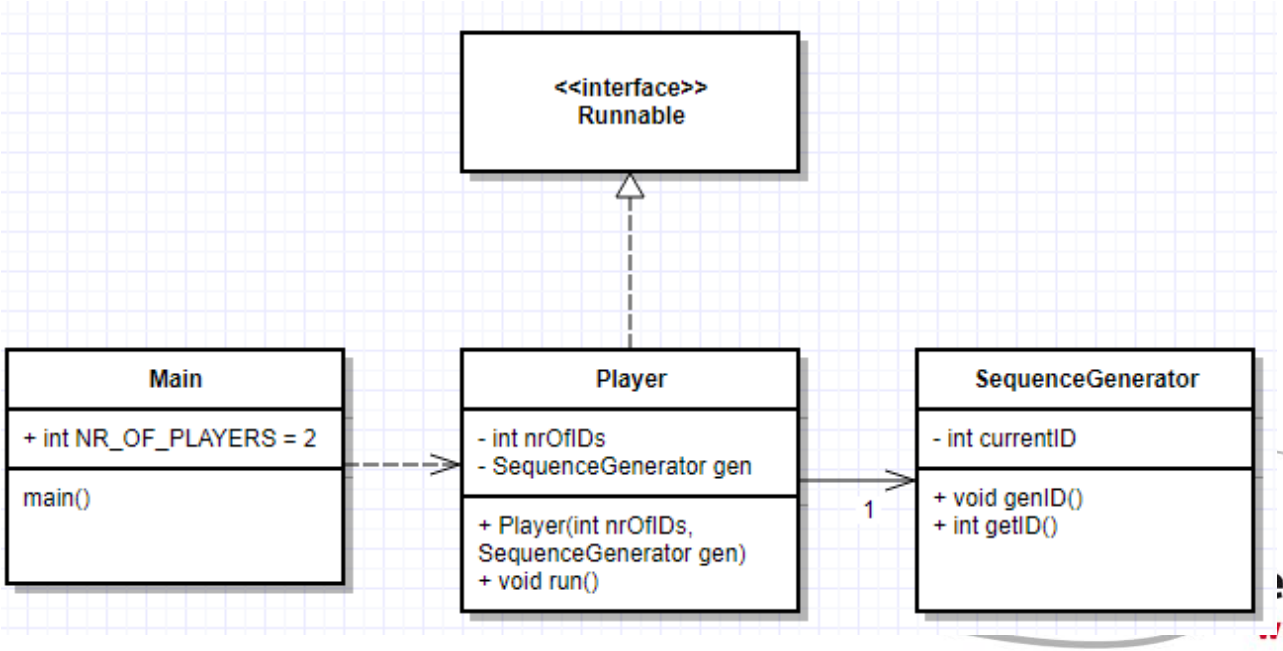
Synchronisatie in Java

- Instructies atomair uitvoeren dmv het keyword *synchronized*. Dit keyword kan voor methoden en om blocks staan.
- De hele methode/block wordt aangemerkt als atomair
  - ```
    public synchronized void inc() {
      i++;
    }
    ```
  - ```
    public void inc() {
      synchronized (this) {
                i++;
      }
    }
    ```
- Werkt per object, niet per klasse!
- Alleen de gemarkeerde methoden/blocks worden beschermd
- Synchronized locks zijn reentrant: thread die lock heeft kan hem meerdere keren krijgen (aantal x bijgehouden in integer teller)

Open Universiteit
www.ou.nl

Simulatie van een multi-threaded spel

- Er is een SequenceGenerator die unieke opeenvolgende integers genereert (0, 1, 2, …). Twee Players vragen ID's op van dezelfde generator. Na het opvragen moet het ID getoond worden. Hiervoor is een multi-threaded programma gemaakt waarbij iedere speler in zijn eigen Thread een ID laat genereren, en dat ID vervolgens afdrukt



Broncode Player, SequenceGenerator en Main

```java
/**
 * Generates a sequence of unique, consecutive IDs 0, 1, 2, 3, ...
 *
 */
public class SequenceGenerator {
    private int currentID=0;

    /**
     * Generates the next (consecutive) ID in the sequence.
     */
    public void genID() {
        currentID++;
    }

    /**
     * Returns the current ID in the sequence.
     * @return the current ID
     */
    public int getID() {
        return currentID;
    }
}
```

```java
/**
 * A player that retrieves integer IDs from a sequence generator
 *
 */
public class Player implements Runnable {
    private int nrOfIDs;
    private SequenceGenerator generator;

    /**
     * Creates the player that retrieves id's
     * @param nrOfIDs the number of id's to retrieve
     * @param gen the sequence generator that generates the id's
     */
    public Player(int nrOfIDs, SequenceGenerator gen) {
        this.nrOfIDs = nrOfIDs;
        this.generator = gen;
    }

    /**
     * Prints a list of unique id's in ascending order (not necessarily consecutive).
     */
    @Override
    public void run() {
        for(int i=0; i<nrOfIDs; i++) {
                generator.genID();
                System.out.println(generator.getID());
        }

    }

}
```

```java
public class Main {
    public static final int NR_OF_PLAYERS = 2;

    public static void main(String[] args) {
        SequenceGenerator gen = new SequenceGenerator();
        for (int i=0; i<NR_OF_PLAYERS; i++) {
            Player p = new Player(5, gen);
            Thread t1 = new Thread(p);
            t1.start();
        }
    }
}
```


Open Universiteit
www.ou.nl

Opgave

1. Is de klasse SequenceGenerator thread-safe? Zo niet:
   a) Geef aan waarom niet (waar kan interferentie optreden?)
   b) Zo niet: welke aanpassingen zijn nodig om de klasse thread-safe te maken?

2. Zelfde vraag voor klasse Player

Open Universiteit
www.ou.nl

SequenceGenerator

SequenceGenerator is **niet** thread-safe: als uit twee verschillende threads genID() wordt aangeroepen, kan hetzelfde ID twee keer worden gegenereerd, want currentID++ is geen atomische actie.

Oplossen door genID() en getID() **synchronized** te maken

public synchronized void genID() { … }

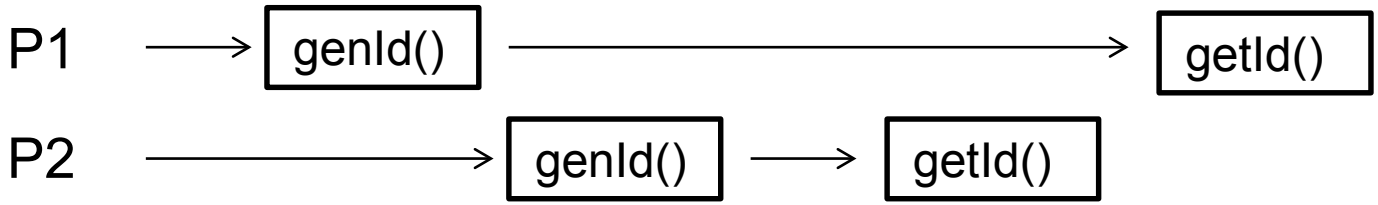
public synchronized void getID() { … }

Ook mogelijk: attribuut currentId **volatile** maken ipv genID() **synchronized**

Open Universiteit
www.ou.nl

Player

Player is **niet** thread-safe: als 2 players dezelfde gedeelde generator gebruiken om IDs te genereren en daarna te printen, dan is niet gegarandeerd dat het door die speler gegenereerde ID wordt getoond:



Thread safe maken als volgt:

1. Optie a: Maak een compound synchronized methode/block die gooit *en* het aantal ogen teruggeeft in klasse SequenceGen.
2. Optie b: Gebruik een synchronized blok in klasse Player om genID() en getID() zonder interferentie uit te voeren

Open Universiteit
www.ou.nl

Player – thread safe

- Optie a (compound methode in klasse SequenceGenerator):

```
public synchronized int generateAndReturnID() {
        currentID++;
        return currentID;
}
```

- Optie b (synchronized block):

```
@Override public void run() {
        for(int i=0; i<nrOfIDs; i++) {
                synchronized(generator) {
                        generator.genID();
                        System.out.println(generator.getID());
                }
        }
}
```


Open Universiteit
www.ou.nl

Player – thread safe

- Optie b – alternatief (synchronized block op private lock object):

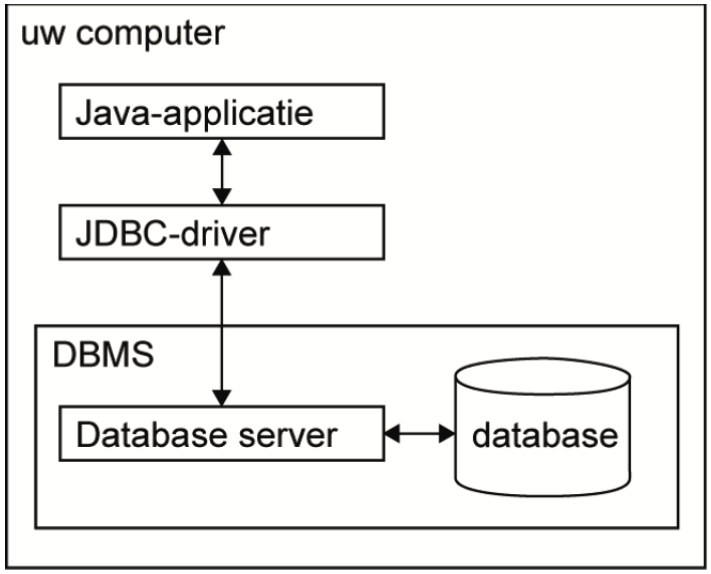
```
public class Player implements Runnable {
    private static final Object lock = new Object();

    …
    @Override public void run() {
        for(int i=0; i<nrOfIDs; i++) {
                synchronized(lock) {
                        generator.genID();
                        System.out.println(generator.getID());
                }
        }

    }
}
```

- Moet lock static zijn? Waarom?
- Moet lock final zijn? Waarom?

Koppeling met databases

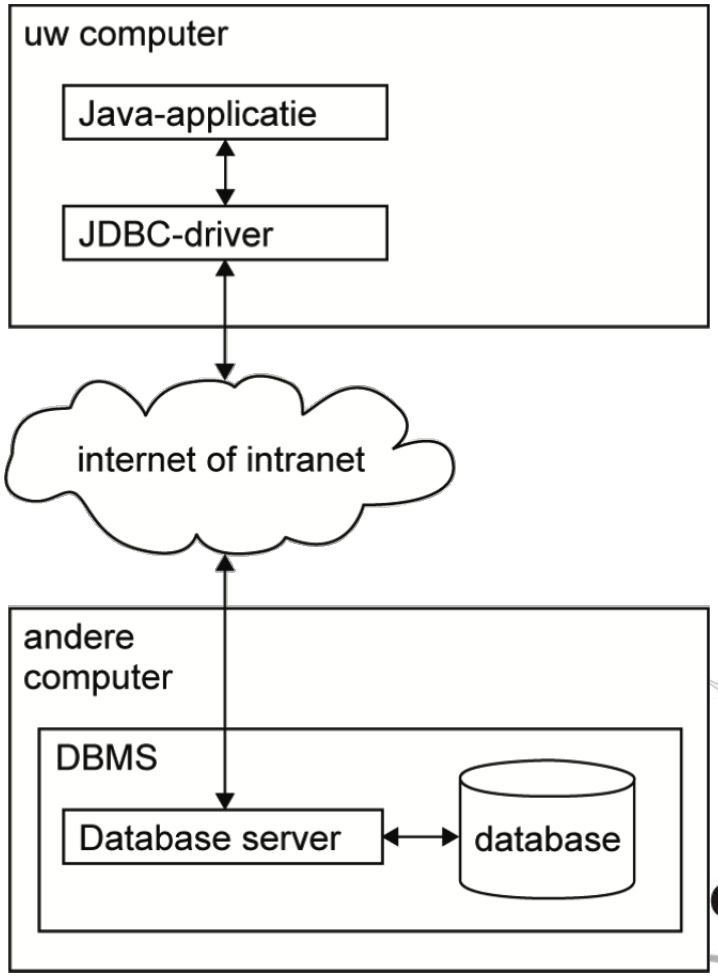
LEEREENHEID 9

§2.1 – JDBC-drivers



Open Universiteit
www.ou.nl

§2.1 – JDBC-drivers



Open Universiteit
www.ou.nl

§2.1 – JDBC-drivers

- Specificatie van JDBC (Java database connectivity) in packages java.sql en javax.sql
- Deze packages bestaan voor een deel uit interfaces
- Voor elk type database is er een implementatie: de JDBC-driver
  - Geleverd door DBMS-fabrikant

§3 – De JDBC-API

Algemene opzet

1. Verbinding maken met de database [§3.1]
2. SQL-opdrachten formuleren en versturen naar de database [§3.2.1, §3.2.3]
3. De resultaten van een SQL-opdracht opvragen [§3.2.2]
4. Verbinding verbreken [§3.1]

Aanname: De database server is *gestart* en *toegankelijk* dmv een gebruikersnaam en wachtwoord

§3.1 – Verbinding maken en verbinding verbreken

- Constanten

```
String DRIVER = "org.firebirdsql.jdbc.FBDriver"
String URL = "jdbc:firebirdsql://localhost/cds";
String GEBRUIKERSNAAM = "admin";
String WW = "foobar";
```

- Driver laden

```
Class.forName(DRIVER);
```

- Connectie openen/verbreken

```
Connection con = DriverManager.getConnection(
                    URL, GEBRUIKERSNAAM, WW);
                // SQLException?
con.close();
```

§3.2.1 – Versturen van SELECT-opdrachten

- SQL Opdracht formuleren

```
String sql = "SELECT * FROM CD WHERE code = ?";
PreparedStatement prep = con.prepareStatement(sql);
```

- Waarden invullen (voor iedere ?, tellen vanaf 1)

```
prep.setString(1, "CDS7870362-1");
```

- Opdracht uitvoeren

```
ResultSet res = prep.executeQuery();  // select
```

Of

```
prep.executeUpdate(); // update, delete, insert
```

§3.2.2 – Opvragen van gegevens uit een resultaattabel

- Resultaten zijn beschikbaar via een ResultSet-object

```
ResultSet res = prep.executeQuery();
```

- res is een soort tabel die doorlopen kan worden

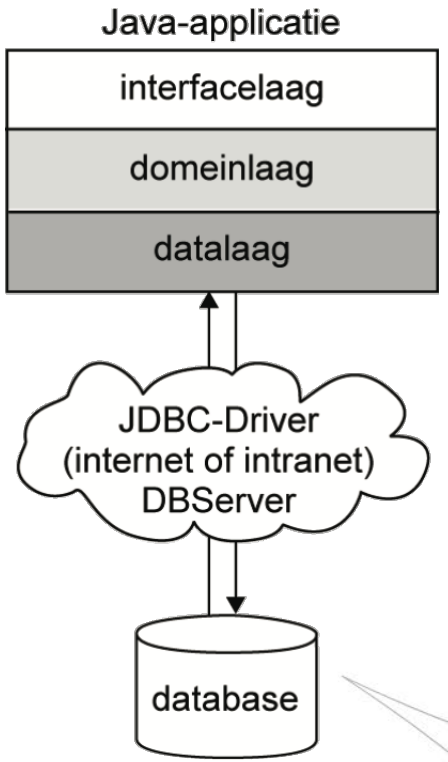
```
while (res.next()) {
  /* ... */
}
```

- Informatie van een record kan per kolom worden opgevraagd (met kolomnaam, of index)

```
String titel = res.getString("titel");
String titel = res.getString(2); // tellen vanaf 1
```

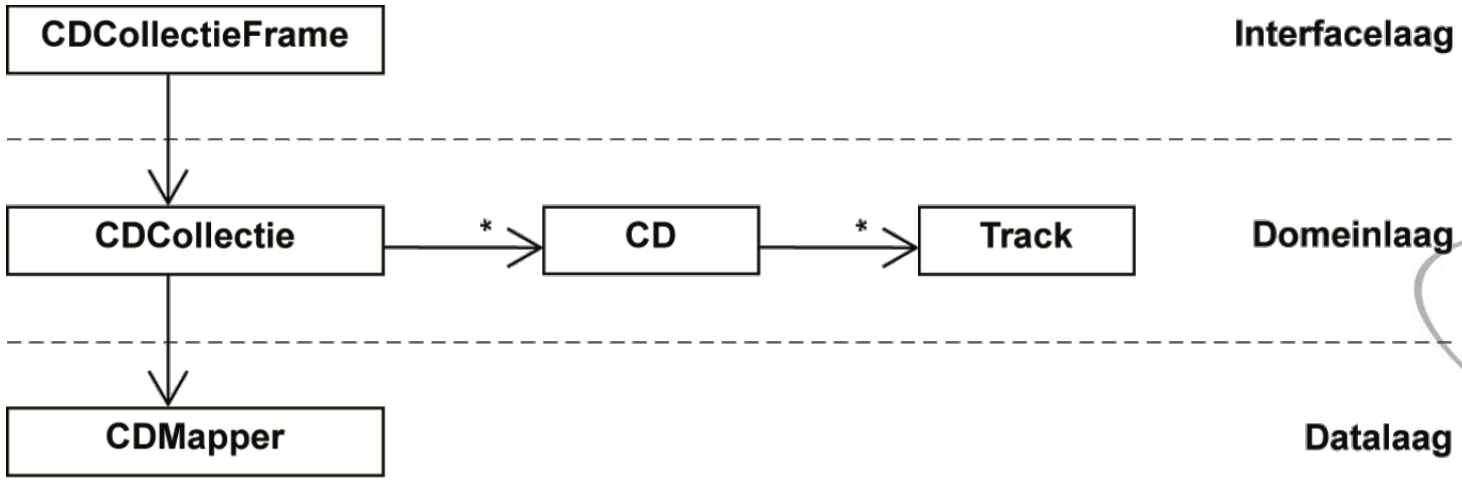
§4 – Ontwerpaspecten

- Ontwerp in drie lagen:



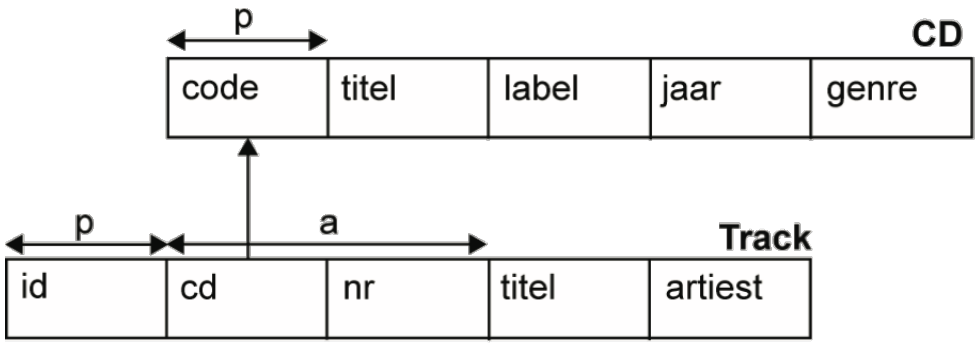
Open Universiteit
www.ou.nl

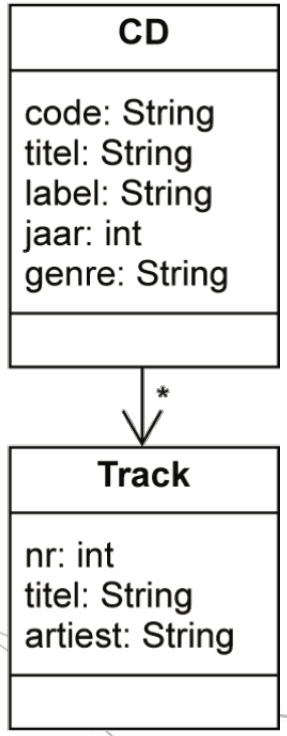
§5.1 – Een datalaag voor een cd-databa…



Open Universiteit
www.ou.nl

§5.1 – Een datalaag voor een cd-databa…





Open Universiteit
www.ou.nl

§5.1 – Een datalaag voor een cd-databa…

```
public class CDMapper {
  private PreparedStatement selectCdsStatement = null;
  private PreparedStatement selectTracksStatement = null;
  private Connection connection = null;

  public CDMapper() throws CDException {
    initConnection(); // Maak verbinding
    initStatements(); // Prepareer statements
  }

  ...
```

§5.1 – Een datalaag voor een cd-databa…

```
private void initConnection() throws CDException {
  try {
    Class.forName(DBConst.DRIVERNAAM); // CNFException?
    connection = DriverManager.getConnection(
      DBConst.URL,
      DBConst.GEBRUIKERSNAAM,
      DBConst.WACHTWOORD); // SQLException?
  }
  catch (ClassNotFoundException e) {
    throw new CDException("Driver niet geladen");
  }
  catch (SQLException e) {
    throw new CDException("Verbinden mislukt");
} }

...
```

Open Universiteit
www.ou.nl

§5.1 – Een datalaag voor een cd-databa…

```
private void initStatements() throws CDException {
  try {
    selectCdsStatement = connection.prepareStatement(
      "SELECT * FROM CD");
    selectTracksStatement = connection.prepareStatement(
      "SELECT * FROM Track WHERE cd = ?");
  }
  catch (SQLException e) {
    closeConnection(); // Nu eerst verbinding sluiten!
    throw new CDException("Fout bij het formuleren van SQL opdracht");
} }

...
```

§5.1 – Een datalaag voor een cd-databa…

```
  public List<CD> leesAlleCDs() throws CDException {
    List<CD> cds = new ArrayList<CD>();
    try {
      ResultSet res = selectCdsStatement.executeQuery();
      while (res.next()) {
        String code = res.getString("code");
        String titel = res.getString("titel");
        String label = res.getString("label");
        int jaar = res.getInt("jaar");
        String genre = res.getString("genre");
        cds.add(new CD(code, titel, label, jaar, genre));
      }
      return cds;
    } catch (SQLException e) {
      throw new CDException("Fout bij inlezen van CD's");
} } }
```

OPGAVE 3

OPDRACHT 3

Aandachtspunten

- Controleer of je implementatie matcht met het ontwerp uit de opdrachtbeschrijving: public/private, zelfde parameters bij methoden, zelfde associaties (tijdelijk/permanent), etc
- Vergeet niet te beschrijven in een apart document waar synchronisatie nodig is en hoe dat gerealiseerd is in de code.
- Ter herinnering: in het algemeen synchronizatie nodig bij methoden op een object dat in in meerdere threads tegelijkertijd gebruikt kan worden → zorg dat klasse invarianten en methode contracten worden gerespecteerd!
- De methode `stop` van Thread is deprecated: gebruik deze dus niet.
- Zorg dat traceerbaar is wat er gebeurd. Geef ook meldingen bij starten/stoppen van Ober/Kok, en welke ober/kok aan welk gerecht werkt.

Wat kan hier mis gaan?

- In run() van Ober:

```
if (balie.isLeeg()) {
     tijdVerstrijkt(WACHTTIJD, "de balie is leeg - wachten");
} else {
     serveer(balie.pakMaaltijd());
}
```

- Wordt dit opgelost door methoden `synchronized` te maken?

TOT SLOT

Open Universiteit
www.ou.nl

Opdracht 3+4

- Voorlopige versie opdracht 4 (04-06-2019, 23:59)
- Definitieve versie opdracht 3 (06-06-2019, 23:59)

- Inleveren via yOUlearn

Open Universiteit
www.ou.nl