Stichting NIOC en de NIOC kennisbank

Stichting NIOC (www.nioc.nl) stelt zich conform zijn statuten tot doel: het realiseren van congressen over informatica onderwijs en voorts al hetgeen met een en ander rechtstreeks of zijdelings verband houdt of daartoe bevorderlijk kan zijn, alles in de ruimste zin des woords.

De stichting NIOC neemt de archivering van de resultaten van de congressen voor zijn rekening. De website www.nioc.nl ontsluit onder "Eerdere congressen" de gearchiveerde websites van eerdere congressen. De vele afzonderlijke congresbijdragen zijn opgenomen in een kennisbank die via dezelfde website onder "NIOC kennisbank" ontsloten wordt.

Op dit moment bevat de NIOC kennisbank alle bijdragen, incl. die van het laatste congres (NIOC2023, gehouden op donderdag 30 maart 2023 jl. en georganiseerd door NHL Stenden Hogeschool). Bij elkaar bijna 1500 bijdragen!

We roepen je op, na het lezen van het document dat door jou is gedownload, de auteur(s) feedback te geven. Dit kan door je te registreren als gebruiker van de NIOC kennisbank. Na registratie krijg je bericht hoe in te loggen op de NIOC kennisbank.

Het eerstvolgende NIOC vindt plaats op donderdag 27 maart 2025 in Zwolle en wordt dan georganiseerd door Hogeschool Windesheim. Kijk op www.nioc2025.nl voor meer informatie.

Wil je op de hoogte blijven van de ontwikkeling rond Stichting NIOC en de NIOC kennisbank, schrijf je dan in op de nieuwsbrief via

www.nioc.nl/nioc-kennisbank/aanmelden nieuwsbrief

Reacties over de NIOC kennisbank en de inhoud daarvan kun je richten aan de beheerder:

R. Smedinga kennisbank@nioc.nl.

Vermeld bij reacties jouw naam en telefoonnummer voor nader contact.

Het methodologisch ontwerpen en implementeren van concurrent programma's

Lex Bijlsma[1], Kees Huizing[2], Harrie Passier[1],

Ruurd Kuiper[2], Harold Pootjes[1] en Sjaak Smetsers[3]

1) Department of Computer Science, OUNL
2) Department of Mathematics and Computing Science, TUE
3) Institute for Computing and Information Sciences, RU

NIOC 2015

Radboud University

TU/e Technische Universiteit Eindhoven University of Technology

Open Universiteit
www.ou.nl

Probleem

- Programmeren is moeilijk
- Het maken van een concurrent-programma is nog moeilijker
  - Vereist kennis van het onderliggende executiemodel
  - Vereist inzicht in de onvoorspelbare interactie tussen concurrerende threads (bijvoorbeeld alle mogelijke executie scenario's)
  - Vereist kennis van primitieven voor synchronisatie en communicatie en bijbehorende (Java) syntax
- Alleen creativiteit is onvoldoende; studieboeken geven weinig houvast

Enkele specifieke oorzaken

- Docenten gebruiken ad-hoc tekenmethoden om de werking van een concurrent programma weer te geven
- Het grote aantal potentiële executiepaden maakt het moeilijk om het dynamisch gedrag zich voor te stellen en te begrijpen
- Studenten halen vaak methode-executie en -scheduling door elkaar
- Studenten vinden het vaak moeilijk om geschikte synchronisatiemechanismen en -primitieven te kiezen om racecondities te voorkomen

Algemene Oorzaken

- Bijna alle boeken op het gebied van programmeren behandelen alleen de syntax en geven vervolgens enkele voorbeelden
- Op *hoe* een probleem moet worden aangepakt wordt niet of nauwelijks ingegaan
- Voor studenten blijft weinig anders over dan de trial-and-error methode
- Er zijn uitzonderingen, bijvoorbeeld:
  - How to design programs (Felleisen)
  - UP voor het modelleren in UML

Enkele citaten

… *Programming instructors often assume that students can take their general problem-solving skills and discover specific programming design skills on their own. Thus, students learn program design through unguided discovery* ... (Linn and Clancy, 1992)

… *We need to look seriously at how we teach programming. The purpose of an education should not simply be to pour facts into students, but rather to teach them to think* … (Gries, 2006)

Twee ervaringen

- Enkele hardop-denken-sessies
  - De chaos bij het ontbreken van een aanpak; het niet gebruiken van de aangeleerde concepten
  - De rust bij het aanwezig zijn van een aanpak

- Observaties tijdens een pair-programming sessie
  - Het zien dat een aanpak ontbreekt
  - Het hardop uitspreken van: … *ik probeer net zo lang tot dat ie het doet, en dan stop ik …*

Vakdidactiek Informatica: Concurrency

- Gekeken naar een algemene didactische aanpak
- Deze algemene didactiek invullen voor het ontwerpen en implementeren van concurrent programma's

Een algemene didactische aanpak

Ten Steps to Complex learning (Merrienboer, 2009)

- Taakbeschrijving
  - Wat moet de student doen, wat moet worden bereikt?
- **Ondersteunende informatie**
  - Conceptuele kennis nodig die nodig is om het probleem op te lossen en te weten wanneer een oplossing goed is
- **Procedurele informatie**
  - Een beschrijving van de (vuist-)regels, stappen, procedures om het probleem op te lossen
- Oefening
  - Van voorbeelden tot zelf problemen oplossen, gebruikmakend van ondersteunende en procedurele kennis
  - Authentieke taken

Ondersteunende informatie (I)

- Wat is concurrency? (concurrency vs parallellisme)
- Wanneer toepassen? (simulatie, responsiviteit, efficiency, inherent parallel)
- Wat zijn threads?
  - Call stack, control flow, systeem, proces, aanroep, toestanden van een thread, …
  - Non-determinisme
  - Atomiciteit en scheduling (`i++;`)

Ondersteunende informatie (II)

- Java threads
  - Toestanden van een thread
  - Creëren van een thread (interface runnable)
  - Starten van een thread
  - Laten slapen van een thread
  - Joinen van threads
  - Stoppen van threads
  - Problemen met threads

Open Universiteit
www.ou.nl

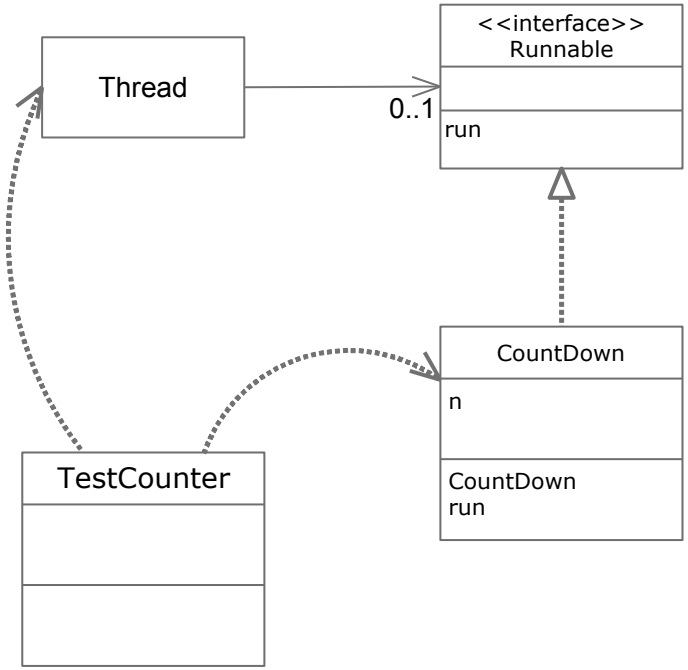
Ondersteunende informatie (III)

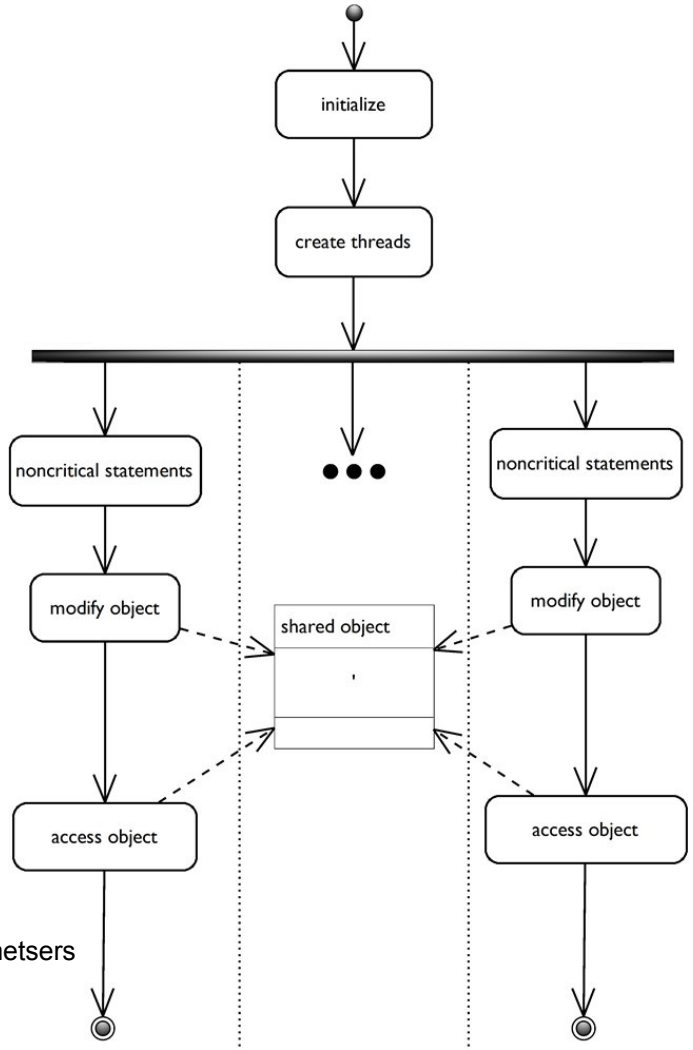
Met threads kunnen we problemen oplossen, maar worden makkelijk nieuwe problemen geïntroduceerd:

- Race conditions
  - Mogelijke incorrecte waarde van een variabele door een ongelukkig scheduler scenario (`i++;`)
  - Oplossing: gelimiteerde toegang tot kritieke secties (locking – in Java d.m.v. `synchronized` blok en methode)
- Deadlock
  - Volgt later

Ondersteunende informatie (IV)

Modelleergereedschappen: UML klasse- en activiteitendiagram





Procedurele informatie

1. **Analyse**: Is het probleem geschikt om met theads op te lossen? (Efficiëntie, inherent parallel, responsiviteit, simulatie)
2. **Ontwerp**: Teken een UML klassendiagram voor de situatie zonder concurrency
3. **Ontwerp**: Breid het diagram uit door aan te geven welke klasse interface Runnable implementeert en welke klasse de thread(s) creëert en start
4. **Ontwerp**: Teken een hoog niveau activitity diagram voor het programmadeel waar methode run actief is. Geef specifiek aan welke objecten worden gedeeld d.m.v. access en update aanroepen
5. **Analyse/ontwerp**: Check op race conditions en pas op de juiste manier synchronisatie toe (het Java monitorpatroon)
6. **Implementeer**
7. **Reflecteer** over gekozen oplossing

Voorbeeld opdracht

Ontwerp en implementeer een program waarin continu met een dobbelsteen wordt gegooid en waarin op bepaalde tijden het *actuele* aantal ogen van de dobbelsteen wordt opgevraagd

Druk bijvoorbeeld in de console 20 keer het aantal ogen af dat om de 100 ms is gegooid.

In deze opdracht hoeft u het proces waarin de dobbelsteen wordt gegooid niet te stoppen.

Classificatie

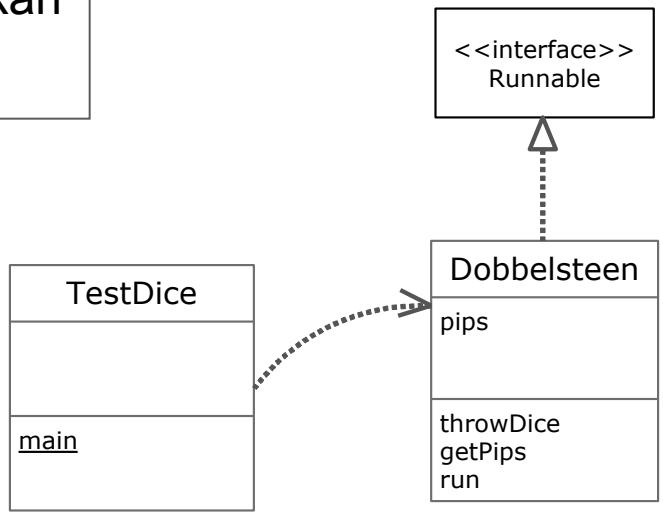
- Het probleem is een voorbeeld van simulatie

Analyse/ontwerp

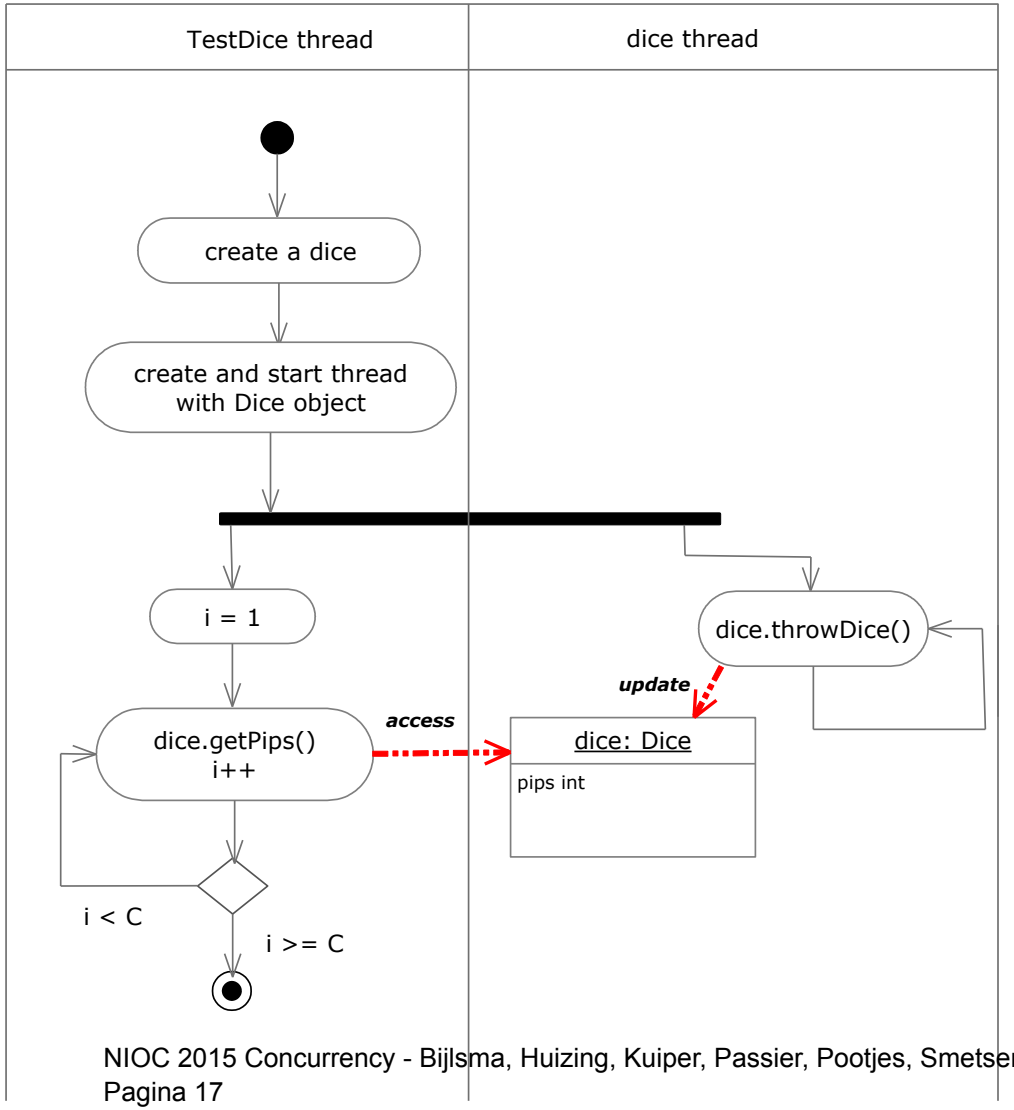
Twee klassen zijn nodig: Dice en TestDice die herhaald het aantal gegooide ogen opvraagt. Elke keer dat de dobbelsteen is gegooid wordt het aantal ogen opgeslagen in een attribuut `pips`. Het aantal ogen kan opgevraagd worden via de methode `getPips`.

De dobbelsteen moet continu worden gegooid. Op bepaalde momenten wordt het aantal ogen opgevraagd. Beide processen vinden plaats in een eigen thread.

Klasse Dice implementeert interface Runnable: het continue gooien wordt gedaan in methode `run`. In TestDice wordt de thread gecreëerd en gestart.



Check op race conditions



Een race condition kan optreden.

Oplossing: maak de update en access methoden `synchronized`.

Code

```
public class Dice implements Runnable {

 private int pips;
 private Random rand = new Random();

 public Dice(int pips) {
  this.pips = pips;
 }

 private synchronized void throwDice() {
  pips = rand.nextInt(6) + 1;
 }

 public synchronized int getPips() {
  return pips;
 }

 public void run() {
  while (true) {
   throwDice();
  }
 }

}
```

```
public class TestDice {

 public static void main(String[] args) {
  Dice dice = new Dice(1);
  Thread thread =  new Thread(dice);
  thread.start();
  for (int i = 0 ; i < 10 ; i++) {
    try {
     Thread.sleep(100);
    } catch (InterruptedException e) {}
    System.out.println(dice.getPips());
  }
 }

}
```

Validatie

- Een trimester (mei – juni 2015) eerstejaars cursus Java
  - TUE, RU: collegezaal (geluidsopname, uitgebreide set sheets en kleine oefenopgaven)
  - OU: Collaborate (scherm- en geluidsopname, idem)
  - Aansluitend werkcollege (3 opgaven)
- Per universiteit worden 15 studenten geselecteerd
  - Maken van tentamenopgave
  - Een aantal studenten wordt geïnterviewd (open en gesloten deel)
  - Drie paren studenten werken aan een opgave in de vorm van pair-programming (video- en geluidsopname) gevolgd door een video-stimulated recall interview

Vervolg

- Producer – Consumer probleem
  - Analyse en voorkomen van deadlock

Open Universiteit
www.ou.nl

Meer info

- Er verschijnt binnenkort een technisch rapport: Didactics for Methodical Concurrency Design

Vragen?