



Міністерство освіти та науки України  
Національний технічний університет України  
«Київський політехнічний інститут імені Ігоря Сікорського»  
Факультет інформатики та обчислювальної техніки  
Кафедра інформатики і програмної інженерії

**Звіт**  
з дисципліни «Технології розподілених обчислень»  
Комп'ютерний практикум №2  
«Розробка паралельних  
програм з використанням механізмів синхронізації: синхронізовані  
методи, локери, спеціальні типи»

**Виконав:**

*Студент III курсу*

*гр. ІІІ-33*

Соколов О. В.

**Прийняв:**

Дифучин А. Ю.

“27” Лютого 2026 р.

## **КОМП'ЮТЕРНИЙ ПРАКТИКУМ №2**

**Тема:** Розробка паралельних програм з використанням механізмів синхронізації: синхронізовані методи, локери, спеціальні типи.

**Мета:** Здобути практичні навички використання механізмів синхронізації потоків у Java (синхронізовані методи, локери, спеціальні типи), реалізувати приклад Producer-Consumer, електронний журнал оцінок та кероване виведення символів у потоках.

### **Завдання:**

1. Реалізуйте програмний код, даний у лістингу (AsynchBankTest), та протестуйте його при різних значеннях параметрів. Модифікуйте програму, використовуючи методи управління потоками, так, щоб її робота була завжди коректною. Запропонуйте три різних варіанти управління. 30 балів.

2. Реалізуйте приклад Producer-Consumer application (див. <https://docs.oracle.com/javase/tutorial/essential/concurrency/guardmeth.html> ). Модифікуйте масив даних цієї програми, які читаються, у масив чисел заданого розміру (100, 1000 або 5000) та протестуйте програму. Зробіть висновок про правильність роботи програми. 20 балів.

3. Реалізуйте роботу електронного журналу групи, в якому зберігаються оцінки з однієї дисципліни трьох груп студентів. Кожного тижня лектор і його 3 асистенти виставляють оцінки з дисципліни за 100-бальною шкалою. 20 балів.

4. Створіть три потоки, один з яких виводить на консоль символ '|', другий – символ '\', а третій – символ '/'. Запустіть потоки в основній програмі так, щоб вони виводили свої символи в рядок. Виведіть на консоль 90 таких рядків. Поясніть виведений результат. 10 балів. Використовуючи методи управління потоками, добийтесь почергового виведення на консоль символів так, щоб утворювався рядок |\\|\\|\\... . 15 балів.

5. Зробіть висновки про використання методів управління потоками в Java. 5 балів.

## ВИКОНАННЯ

### Завдання 1

За лістингом із завдання було реалізовано консольну програму AsynchBankTest, яка моделює банк із NACCOUNTS = 10 рахунками та початковим балансом INITIAL\_BALANCE = 10000 на кожному рахунку. Для кожного рахунку створюється окремий потік TransferThread, який багаторазово виконує випадкові перекази коштів між рахунками методом transfer(from, to, amount). Після кожних NTEST = 10000 транзакцій виконується перевірка test(), що обчислює суму всіх балансів і виводить у консоль кількість транзакцій та поточну суму.

Було реалізовано чотири варіанти банку: один небезпечний (UnsafeBank) та три коректні варіанти керування потоками. У варіанті UnsafeBank масив accounts[] змінюється без будь-якої синхронізації, тому одночасні операції accounts[from] -= amount та accounts[to] += amount можуть «перекривати» одна одну, що призводить до втрати оновлень і зміни загальної суми коштів.

The screenshot shows a terminal window titled "parallel-computing-technologies" running on a Mac OS X system. The command executed is "AsynchBankTest". The output demonstrates three separate runs of the test, each showing multiple parallel transactions being processed simultaneously. The results show significant divergence from the expected total sum of 100000, with each run producing a different sum due to the lack of synchronization.

```
parallel-computing-technologies — qwertannov@Oleksandrs-Laptop — -zsh — 100x52
..-technologies
-/Doc/K/77/parallel-computing-technologies Lab2 *5 !1 ?3 cd /Users/qwertannov/Documents/kpi/3_course/2_sem/parallel-computing-technologies ; /usr/bin/env /Library/Java/JavaVirtualMachines/temurin-21.jdk/Contents/Home/bin/java -XX:+ShowCodeDetailsInExceptionMessages -cp /Users/qwertannov/Library/Application\ Support/Cursor/User/workspaceStorage/60f719d4e1a0e8dfce06207006dbe5ac/redhat.java/jdt_ws/parallel-computing-technologies_6b714d94/bin part_1.bank.AsyncBankTest
AsynchBankTest: 10 accounts, initial balance 10000
Total expected sum: 100000

1) UNSAFE variant (no synchronization, race condition):
   (Run multiple times to see sum often diverge from 100000)

   Run 1:
UnsafeBank → transactions: 10045, sum: 97112
UnsafeBank → transactions: 20017, sum: 94106
UnsafeBank → transactions: 30001, sum: 89832
UnsafeBank → transactions: 40008, sum: 85017
UnsafeBank → transactions: 50001, sum: 84837
UnsafeBank → transactions: 60000, sum: 83392
UnsafeBank → transactions: 70008, sum: 81963
UnsafeBank → transactions: 80005, sum: 81652
UnsafeBank → transactions: 90006, sum: 82071
UnsafeBank → transactions: 100001, sum: 80146
UnsafeBank → transactions: 110002, sum: 79757
UnsafeBank → transactions: 120003, sum: 78849
UnsafeBank → transactions: 130000, sum: 77834
UnsafeBank → transactions: 132567, sum: 77431

   Run 2:
UnsafeBank → transactions: 10007, sum: 99986
UnsafeBank → transactions: 20003, sum: 99295
UnsafeBank → transactions: 30012, sum: 96285
UnsafeBank → transactions: 40000, sum: 95947
UnsafeBank → transactions: 50022, sum: 95695
UnsafeBank → transactions: 60007, sum: 95014
UnsafeBank → transactions: 70006, sum: 94311
UnsafeBank → transactions: 80002, sum: 94534
UnsafeBank → transactions: 90002, sum: 90980
UnsafeBank → transactions: 100012, sum: 91203
UnsafeBank → transactions: 110006, sum: 90630
UnsafeBank → transactions: 118981, sum: 90630

   Run 3:
UnsafeBank → transactions: 10001, sum: 99281
UnsafeBank → transactions: 20072, sum: 99841
UnsafeBank → transactions: 30017, sum: 100058
UnsafeBank → transactions: 40118, sum: 100058
UnsafeBank → transactions: 50011, sum: 100058
UnsafeBank → transactions: 60004, sum: 100058
UnsafeBank → transactions: 70002, sum: 100834
UnsafeBank → transactions: 80013, sum: 100421
UnsafeBank → transactions: 90000, sum: 99850
UnsafeBank → transactions: 90850, sum: 99850
```

Рисунок 1.1 – Консольний вивід програми AsynchBankTest у варіанті  
UnsafeBank (без синхронізації)

При відсутності синхронізації в UnsafeBank операції над елементами масиву accounts не є атомарними та можуть виконуватися паралельно в різних потоках. Декілька потоків можуть одночасно зчитувати старий

баланс, змінювати його та записувати результат, «перетираючи» оновлення один одного. Унаслідок такої гонки потоків (race condition) частина транзакцій втрачається, а сумарний баланс зменшується та стає непередбачуваним – саме це видно у виводі на рисунку 1.1.

```
parallel-computing-technologies — qwertannov@Oleksandrs-Laptop — -zsh — 100x33
..-technologies

2) SYNCHRONIZED methods variant:
SynchronizedBank → transactions: 10000, sum: 100000
SynchronizedBank → transactions: 20000, sum: 100000
SynchronizedBank → transactions: 30000, sum: 100000
SynchronizedBank → transactions: 40000, sum: 100000
SynchronizedBank → transactions: 50000, sum: 100000
SynchronizedBank → transactions: 60000, sum: 100000
SynchronizedBank → transactions: 70000, sum: 100000
SynchronizedBank → transactions: 80000, sum: 100000
SynchronizedBank → transactions: 83400, sum: 100000

3) ReentrantLock variant:
LockBank → transactions: 10000, sum: 100000
LockBank → transactions: 20000, sum: 100000
LockBank → transactions: 30000, sum: 100000
LockBank → transactions: 40000, sum: 100000
LockBank → transactions: 50000, sum: 100000
LockBank → transactions: 56007, sum: 100000

4) AtomicIntegerArray + AtomicLong variant:
AtomicBank → transactions: 10000, sum: 100000, expected: 100000
AtomicBank → transactions: 20000, sum: 100000, expected: 100000
AtomicBank → transactions: 30000, sum: 100000, expected: 100000
AtomicBank → transactions: 40000, sum: 100000, expected: 100000
AtomicBank → transactions: 50000, sum: 100000, expected: 100000
AtomicBank → transactions: 60000, sum: 100000, expected: 100000
AtomicBank → transactions: 70000, sum: 100000, expected: 100000
AtomicBank → transactions: 76589, sum: 100000, expected: 100000
```

Рисунок 1.1 – Консольний вивід програми AsynchBankTest у варіантах SyncronizedBank, LockBank та AtomicBank

У варіанті SynchronizedBank коректність досягається завдяки монітору об'єкта: одночасно в межах одного екземпляра банку може виконуватися лише один метод transfer() або test(), тому операції над масивом accounts[] виконуються послідовно без втрати оновлень. У

LockBank використано явний локер ReentrantLock, який дає той самий рівень взаємного виключення, але з більш гнучким керуванням блокуванням. У AtomicBank баланси рахунків зберігаються в AtomicIntegerArray, а лічильник транзакцій – в AtomicLong; разом із глобальним ReentrantLock це забезпечує як потокобезпечність оновлень, так і збереження інваріанту суми (sum завжди дорівнює очікуваному значенню). Усі три синхронізовані варіанти гарантують збереження суми, але відрізняються простотою синтаксису та можливостями для розширення (локери й атомарні типи зручніші в складніших сценаріях).

## Завдання 2

Реалізовано приклад Producer-Consumer за патерном guarded blocks: спільний об'єкт BoundedBuffer з методами put(int) та take(), які використовують synchronized, цикл while (умова) { wait(); } та notifyAll() після зміни стану. Буфер має заданий розмір і зберігає масив цілих чисел; при повному буфері producer блокується, при порожньому – consumer. Потік Producer послідовно записує числа 0, 1, 2, ... до заданої кількості; потік Consumer читає їх у буфер і зберігає в масив. Після завершення обох потоків виконується перевірка: кількість отриманих значень, їхня сума (має дорівнювати  $0+1+\dots+(n-1)$ ), відсутність дублікатів і наявність усіх чисел з діапазону. Програма ProducerConsumerDemo запускає тест для кожного розміру буфера і виводить результат перевірки та час виконання.

```
parallel-computing-technologies -- qwertannov@Oleksandrs-Laptop -- -zsh -- 100x31
..-technologies
~/.../.../parallel-computing-technologies lab2 *5 ?2 cd /Users/qwertannov/Documents/kpi/3_course/2_sem/parallel-computing-technologies ; /usr/bin/env /Library/Java/JavaVirtualMachines/temurin-21.jdk/Contents/Home/bin/java -XX:+ShowCodeDetailsInExceptionMessages -cp /Users/qwertannov/Library/Application\ Support/Cursor/User/workspaceStorage/60f719d4e1a0e8dfce06207006dbe5ac/redhat.java/jdt_ws/parallel-computing-technologies_6b714d94/bin_part_2.producer_consumer.ProducerConsumerDemo
Producer-Consumer (guarded blocks: wait/notifyAll)
Total items per run: 10000

Buffer size: 100
Count: 10000 (expected 10000)
Sum: 49995000 (expected 49995000)
No duplicates, all 0..9999: true
Result: CORRECT
Time: 10 ms

Buffer size: 1000
Count: 10000 (expected 10000)
Sum: 49995000 (expected 49995000)
No duplicates, all 0..9999: true
Result: CORRECT
Time: 4 ms

Buffer size: 5000
Count: 10000 (expected 10000)
Sum: 49995000 (expected 49995000)
No duplicates, all 0..9999: true
Result: CORRECT
Time: 3 ms
```

Рисунок 2.1 – Консольний вивід Producer-Consumer із різними розмірами буферау

При усіх трьох розмірах буфера (100, 1000, 5000) програма працює коректно: споживач отримує рівно стільки чисел, скільки виробив виробник, без втрат і дублювань; сума та перевірка діапазону збігаються з очікуваними. Це підтверджує, що використання guarded blocks (wait/notifyAll) та одного монітора для put/take забезпечує правильну взаємодію потоків і збереження даних у обмеженому буфері.

### Завдання 3

Реалізовано електронний журнал. Модель даних: три групи, у кожній по studentsPerGroup студентів, оцінки з однієї дисципліни за 100-балльною шкалою за кожен із weeksCount тижнів. Дані зберігаються в масиві grades[група][студент][тиждень]. Один потік – лектор (LecturerThread): кожного тижня виставляє оцінку кожному студенту в усіх трьох групах (діапазон 60–100). Три потоки – асистенти (AssistantThread): асистент виставляє оцінки лише студентам групи за всі тижні (50–100). Усі чотири потоки працюють одночасно; для однієї комірки (група, студент, тиждень) останній запис «перекриває» попередній. Синхронізація: у класі Gradebook методи setGrade та getGrade (і формування таблиці) використовують один ReentrantLock – кожен запис і читання виконуються під замком, тому немає втрати оновлень і читання бачить узгоджений стан.

```

parallel-computing-technologies — qwertannov@Oleksandrs-Laptop — -zsh — 100x41
..-technologies
[-/Doc/k/3/parallel-computing-technologies] Lab2 *5 !1 ?1 cd /Users/qwertannov/Documents/kpi/3_course/2_sem/parallel-computing-technologies ; /usr/bin/env /Library/Java/JavaVirtualMachines/temurin-21.jdk/Contents/Home/bin/java -XX:+ShowCodeDetailsInExceptionMessages -cp /Users/qwertannov/Library/Application\ Support/Cursor/User/workspaceStorage/60f719d4e1a0e8dfce0620700dbe5ac/redhat.java/jdt_ws/parallel-computing-technologies_6b714d94/bin part_3.gradebook.GradebookDemo
Electronic gradebook: 1 lecturer + 3 assistants
3 groups, 5 students per group, 4 weeks
Grades 0-100. All four threads write concurrently (ReentrantLock).

Electronic gradebook (3 groups, 5 students, 4 weeks)

--- Group 1 ---
Student Week1 Week2 Week3 Week4 Avg
1 76 83 70 61 72.5
2 97 76 63 82 79.5
3 71 96 78 78 80.8
4 50 57 74 71 63.0
5 80 99 74 65 79.5
Group avg: 75.1

--- Group 2 ---
Student Week1 Week2 Week3 Week4 Avg
1 90 89 89 95 90.8
2 57 87 51 100 73.8
3 52 71 89 86 74.5
4 54 75 52 97 69.5
5 90 98 86 92 91.5
Group avg: 80.0

--- Group 3 ---
Student Week1 Week2 Week3 Week4 Avg
1 72 70 99 88 82.3
2 81 64 66 65 69.0
3 75 54 92 85 76.5
4 87 92 76 70 81.3
5 63 88 83 87 80.3
Group avg: 77.9

```

Рисунок 3.1 – Консольний вивід після завершення потоків

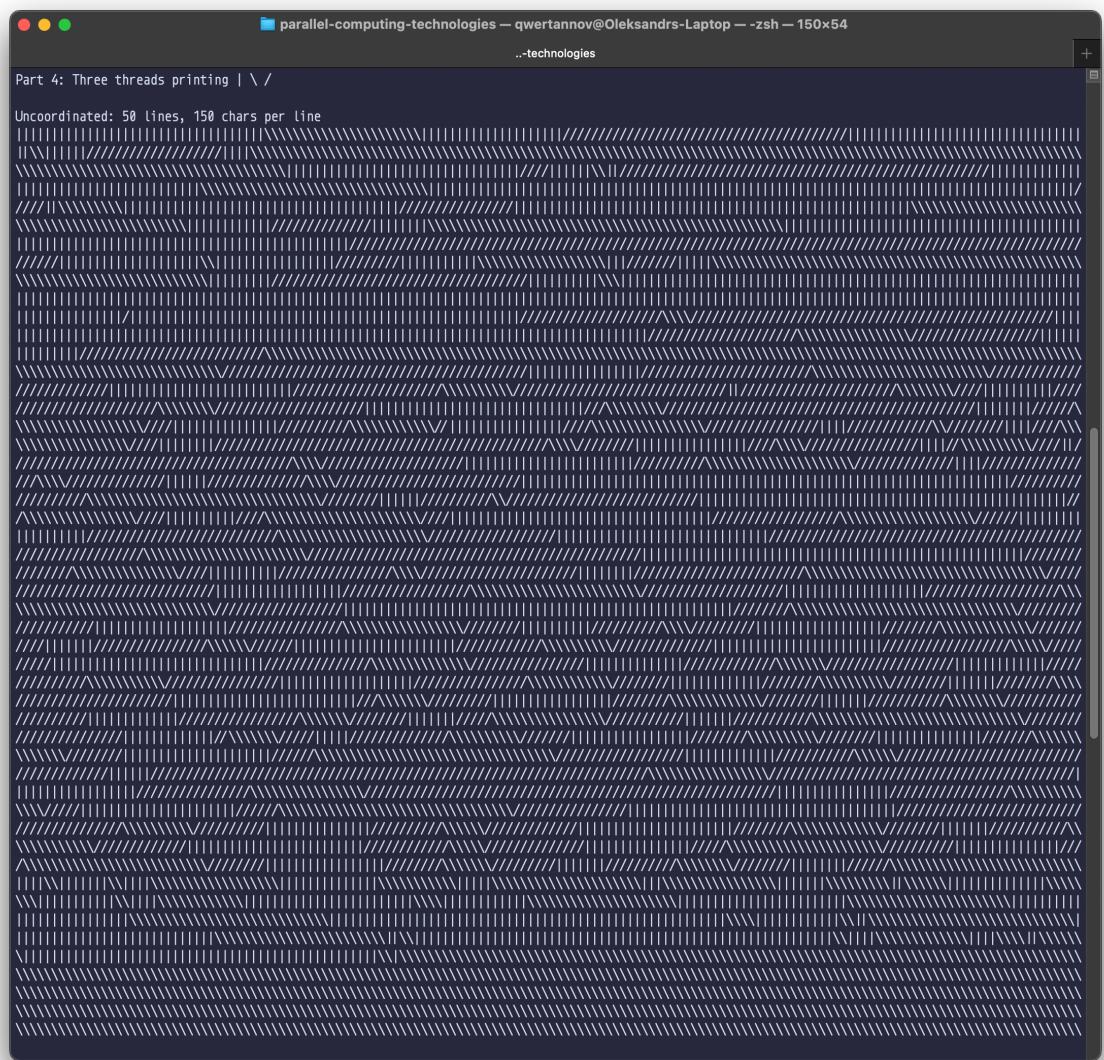
Синхронізація через один загальний замок (ReentrantLock) гарантує, що в момент запису оцінки в комірку жоден інший потік не читає й не змінює цю комірку одночасно. Усі оновлення від лектора та асистентів послідовні з точки зору доступу до спільногого стану журналу, тому не виникає гонок і втрачених записів; після join усіх потоків таблиця містить коректні остаточні значення.

## Завдання 4

Реалізовано програму, в якій три потоки виводять символи '|', ',', '/' відповідно. Виведено задану кількість рядків по заданій кількості символів у рядку. Дві версії:

Частина 1 – без координації: Кожен потік у циклі виводить свій символ. Синхронізація використовується лише для спільногого підрахунку символів та виводу переведення рядка після кожних N символів; порядок виводу символів між потоками не узгоджується. Потоки виконуються паралельно, планувальник вирішує, хто виконується в даний момент, тому послідовність символів у рядку непередбачувана (хаотичне чергування |, , /).

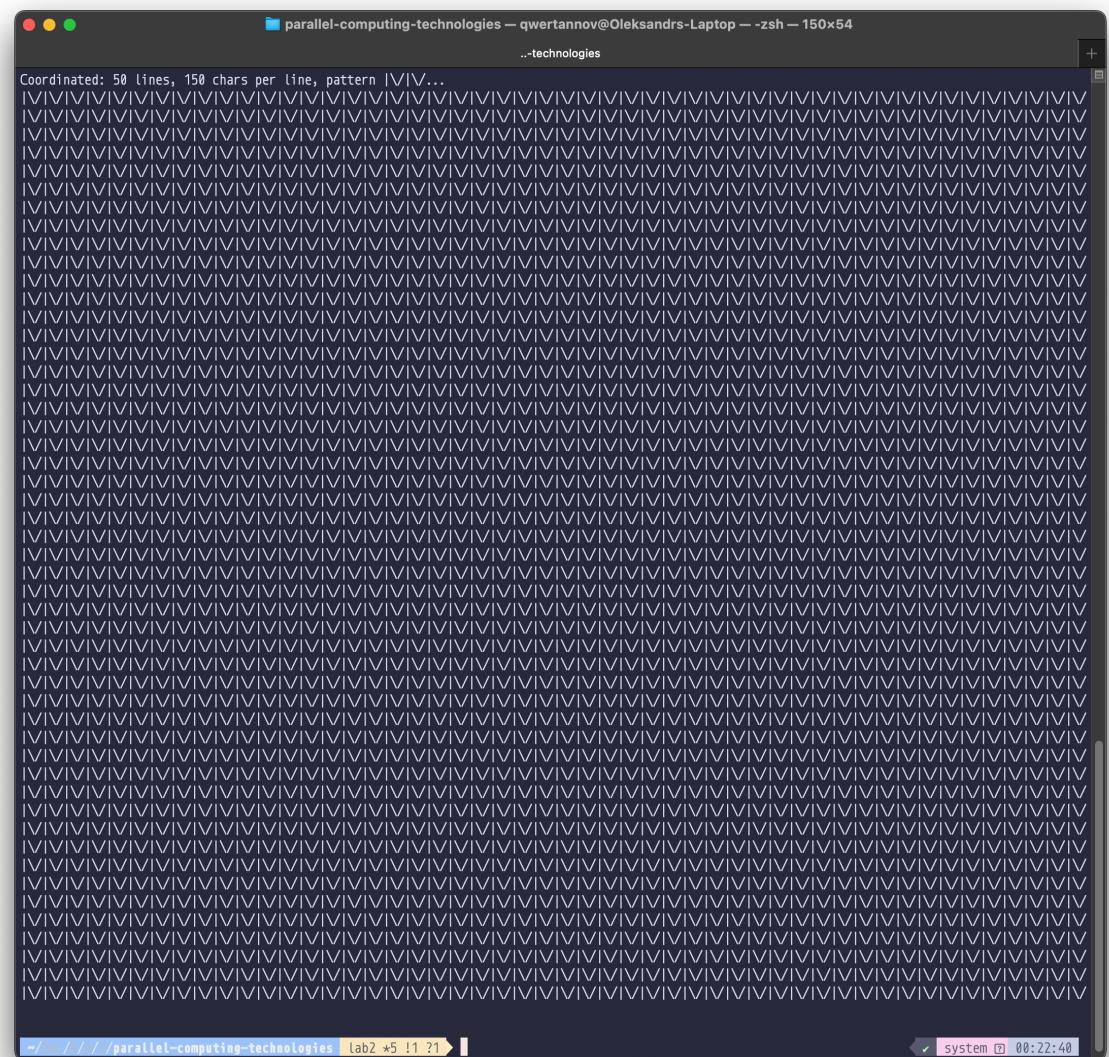
Частина 2 – почерговий вивід |||/...: Використано один спільний замок і змінну стану (черга). Кожен потік під замком чекає, поки не настане його черга (за умовою в циклі while і викликом wait), виводить один символ, оновлює стан і сповіщає інші потоки (notifyAll). Після кожних N символів виводиться переведення рядка. Таким чином досягається суворий порядок: спочатку |, потім , потім /, знову |, , / тощо у кожному рядку повторюється патерн |||/... .



The screenshot shows a terminal window titled "parallel-computing-technologies — qwertannov@Oleksandrs-Laptop — -zsh — 150x64". The title bar also includes ".-technologies". The terminal content starts with "Part 4: Three threads printing | \ /". Below this, the text "Uncoordinated: 50 lines, 150 chars per line" is displayed. The main area of the terminal is filled with a dense, uncoordinated pattern of diagonal slashes (forward and backward) and other characters, creating a visual representation of multiple threads printing simultaneously without synchronization.

Рисунок 4.1 – Консольний вивід без синхронізації

Порядок символів без координації непередбачуваний: потоки виконуються паралельно, планувальник надає їм кванту часу в довільному порядку, вивід у консоль не синхронізований між потоками, тому символи змішуються випадково.



The screenshot shows a terminal window titled "parallel-computing-technologies — qwertannov@Oleksandrs-Laptop — -zsh — 150x54". The window contains a command-line interface with the following text:  
Coordinated: 50 lines, 150 chars per line, pattern \/\|...  
The terminal displays a continuous sequence of characters: backslash (\), forward slash (/), vertical bar (|), and another backslash (\). This sequence repeats across 50 lines, indicating a synchronized output from multiple threads or processes. The terminal window has a dark background with light-colored text, and the status bar at the bottom shows the path "/Users/qwertannov/parallel-computing-technologies/lab2" and the time "00:22:40".

Рисунок 4.2 – Консольний вивід із синхронізацією (wait/notify)

Методи `wait()` та `notifyAll()` у поєднанні з перевіркою умови в циклі `while` забезпечують жорстку черговість: лише один потік за раз виводить символ, і лише коли настало його черга. Після виводу черга переходить до наступного потоку, тому виходить патерн `\|\|/...`.

## Завдання 5

Синхронізовані методи та блоки (`synchronized`) є найпростішим способом забезпечення взаємного виключення. Для цього достатньо позначити метод ключовим словом `synchronized` або використати блок `synchronized`(об'єкт). Монітор у такому випадку прив'язаний до конкретного об'єкта, і лише один потік одночасно може виконувати синхронізований код для цього об'єкта. Перевагами підходу є простота реалізації та зрозумілість коду. Водночас існують обмеження: неможливо зручно реалізувати умовне очікування (наприклад, «поки буфер не порожній»), складно комбінувати кілька умов або використовувати таймаути.

Явні замки, зокрема `ReentrantLock`, забезпечують той самий механізм взаємного виключення, але надають більшу гнучкість. Вони дозволяють використовувати метод `tryLock`, встановлювати таймаути при спробі захоплення замка та працювати з умовними чергами через інтерфейс `Condition`. Це робить їх зручними у випадках, коли необхідна складніша логіка очікування або робота з кількома умовами. Водночас програміст повинен обов'язково звільнити замок за допомогою методу `unlock()`, бажано в блоці `finally`, щоб уникнути взаємного блокування.

Методи `wait()`, `notify()` та `notifyAll()` використовуються разом із `synchronized` для реалізації так званих `guarded blocks` – блоків коду з очікуванням певної умови. Класичний шаблон передбачає використання циклу `while` (умова) { `wait();` } після захоплення монітора, а після зміни стану – виклик `notifyAll()`. Такий підхід широко застосовується у задачах типу `producer-consumer` або для організації черговості виконання потоків. Важливо перевіряти умову саме в циклі `while`, а не в `if`, щоб уникнути помилок через спонтанні пробудження потоків.

Атомарні типи, такі як `AtomicInteger`, `AtomicLong`, `AtomicIntegerArray` та інші, забезпечують виконання атомарних операцій над змінними без використання явних замків. Вони зручні для реалізації лічильників, пропорців або простих оновлень значень. Проте у випадках складнішої

логіки, наприклад під час переказу коштів між рахунками з перевіркою балансу, зазвичай необхідно додатково використовувати механізми блокування, щоб зберегти інваріант між кількома змінними.

У межах практичної роботи синхронізовани методи та ReentrantLock було застосовано в завданні 1 (модель банку), механізм wait/notify у завданнях 2 (буфер) та 4 (черговість виведення символів), а атомарні типи – як один із варіантів реалізації в завданні 1. Таким чином, вибір конкретного механізму синхронізації залежить від характеру задачі: для простого захисту критичної ділянки коду достатньо synchronized, для очікування виконання певної умови – wait/notify, а для складніших сценаріїв з кількома умовами або таймаутами доцільно використовувати ReentrantLock і Condition.

## **Висновок**

У межах комп'ютерного практикуму №2 отримано практичні навички використання механізмів синхронізації потоків у Java та реалізовано чотири частини завдань.

У частині 1 реалізовано модель банку з багатьма потоками переказів; продемонстровано, що без синхронізації виникає гонка і сума по рахунках перестає збігатися з очікуваною. Запропоновано три коректні варіанти: синхронізовані методи, явний замок ReentrantLock та використання атомарних типів у поєднанні з замком – усі вони зберігають інваріант суми. У частині 2 реалізовано приклад producer-consumer з обмеженим буфером цілих чисел та патерном guarded blocks (wait/notifyAll); перевірено коректність при різних розмірах буфера. У частині 3 реалізовано електронний журнал оцінок: три групи, лектор і три асистенти виставляють оцінки паралельно; один ReentrantLock забезпечує узгодженість записів. У частині 4 реалізовано три потоки, що виводять символи |, , /: спочатку без координації (хаотичний вивід), потім з координацією через wait/notify для отримання патерну |||/.... .

Підсумовуючи, виконані завдання охоплюють синхронізовані методи та блоки, явні замки (ReentrantLock), методи wait/notify для умовного очікування та атомарні типи. Кожен підхід має свою область застосування; правильний вибір механізму синхронізації забезпечує коректність і передбачуваність багатопотокових програм.

## Код програми

Посилання на GitHub репозиторій:

<https://github.com/sokolovgit/parallel-computing-technologies>

### Задача 1

```
for f in *.java; do
    echo "===== $f ====="
    cat "$f"
    echo
done
===== AsynchBankTest.java =====
package part_1.bank;

/**
 * Asynchronous bank test based on the listing from the lab assignment.
 * Contains one unsafe implementation and three different synchronized variants.
 */
public class AsynchBankTest {

    public static final int NACCOUNTS = 10;
    public static final int INITIAL_BALANCE = 10_000;
    /**
     * How many transfer operations each thread performs.
     */
    public static final int REPS = 100_000;

    public static void main(String[] args) throws InterruptedException {
        System.out.println("AsynchBankTest: " + NACCOUNTS + " accounts, initial balance " +
        INITIAL_BALANCE);
        System.out.println("Total expected sum: " + (NACCOUNTS * INITIAL_BALANCE));

        runVariantUnsafe();
        runVariantSynchronized();
        runVariantLock();
        runVariantAtomic();
    }

    private static void runVariantUnsafe() throws InterruptedException {
        System.out.println();
        System.out.println("1) UNSAFE variant (no synchronization, race condition):");
        System.out
            .println(" (Run multiple times to see sum often diverge from " + (NACCOUNTS *
        INITIAL_BALANCE) + ")");

        for (int run = 1; run <= 3; run++) {
            System.out.println(" Run " + run + ":");
            Bank bank = new UnsafeBank(NACCOUNTS, INITIAL_BALANCE);
            startAndJoinAllThreads(bank);
            System.out.println();
        }
    }

    private static void runVariantSynchronized() throws InterruptedException {
```

```

System.out.println();
System.out.println("2) SYNCHRONIZED methods variant:");
Bank bank = new SynchronizedBank(NACCOUNTS, INITIAL_BALANCE);
startAndJoinAllThreads(bank);
System.out.println();
}

private static void runVariantLock() throws InterruptedException {
    System.out.println();
    System.out.println("3) ReentrantLock variant:");
    Bank bank = new LockBank(NACCOUNTS, INITIAL_BALANCE);
    startAndJoinAllThreads(bank);
    System.out.println();
}

private static void runVariantAtomic() throws InterruptedException {
    System.out.println();
    System.out.println("4) AtomicIntegerArray + AtomicLong variant:");
    Bank bank = new AtomicBank(NACCOUNTS, INITIAL_BALANCE);
    startAndJoinAllThreads(bank);
    System.out.println();
}

private static void startAndJoinAllThreads(Bank bank) throws InterruptedException {
    TransferThread[] threads = new TransferThread[NACCOUNTS];
    for (int i = 0; i < NACCOUNTS; i++) {
        threads[i] = new TransferThread(bank, i, INITIAL_BALANCE, REPS);
        threads[i].setPriority(Thread.NORM_PRIORITY + i % 2);
        threads[i].start();
    }
    for (TransferThread t : threads) {
        t.join();
    }

    // final consistency check
    if (bank instanceof TestableBank testableBank) {
        testableBank.test();
    }
}
}

===== Bank.java =====
package part_1.bank;

import java.util.concurrent.locks.Lock;
import java.util.concurrent.locks.ReentrantLock;
import java.util.concurrent.atomic.AtomicIntegerArray;
import java.util.concurrent.atomic.AtomicLong;

interface Bank {
    void transfer(int from, int to, int amount);

    int size();
}

interface TestableBank {
    void test();
}

abstract class AbstractBank implements Bank, TestableBank {

```

```

protected static final int NTEST = 10_000;

protected final int[] accounts;
protected long ntransacts = 0;

protected AbstractBank(int n, int initialBalance) {
    accounts = new int[n];
    for (int i = 0; i < accounts.length; i++) {
        accounts[i] = initialBalance;
    }
}

protected void doTransfer(int from, int to, int amount) {
    if (from == to) {
        return;
    }
    if (amount <= 0) {
        return;
    }

    if (accounts[from] < amount) {
        return;
    }

    accounts[from] -= amount;
    accounts[to] += amount;
    ntransacts++;
    if (ntransacts % NTEST == 0) {
        test();
    }
}

@Override
public int size() {
    return accounts.length;
}

@Override
public void test() {
    int sum = 0;
    for (int account : accounts) {
        sum += account;
    }
    System.out.println(
        getClass().getSimpleName()
        + " -> transactions: " + ntransacts
        + ", sum: " + sum);
}

/**
 * Completely unsafe bank: no synchronization at all.
 * This version demonstrates race conditions and incorrect sums.
 */
class UnsafeBank extends AbstractBank {

    UnsafeBank(int n, int initialBalance) {
        super(n, initialBalance);
    }
}

```

```
@Override
public void transfer(int from, int to, int amount) {
    doTransfer(from, to, amount);
}
}

< /**
 * Bank using synchronized methods (implicit monitor on this).
 */
class SynchronizedBank extends AbstractBank {

    SynchronizedBank(int n, int initialBalance) {
        super(n, initialBalance);
    }

    @Override
    public synchronized void transfer(int from, int to, int amount) {
        doTransfer(from, to, amount);
    }

    @Override
    public synchronized void test() {
        super.test();
    }
}

< /**
 * Bank using an explicit ReentrantLock as a single global lock.
 */
class LockBank extends AbstractBank {

    private final Lock lock = new ReentrantLock();

    LockBank(int n, int initialBalance) {
        super(n, initialBalance);
    }

    @Override
    public void transfer(int from, int to, int amount) {
        lock.lock();
        try {
            doTransfer(from, to, amount);
        } finally {
            lock.unlock();
        }
    }

    @Override
    public void test() {
        lock.lock();
        try {
            super.test();
        } finally {
            lock.unlock();
        }
    }
}
```

```

* Bank that stores balances in AtomicIntegerArray and uses AtomicLong
* for the transaction counter. All modifications and tests are still
* protected by a ReentrantLock to keep the global invariant (constant sum).
*/
class AtomicBank implements Bank, TestableBank {

    private static final int NTEST = 10_000;

    private final AtomicIntegerArray accounts;
    private final AtomicLong ntransacts = new AtomicLong(0);
    private final int expectedTotal;
    private final Lock lock = new ReentrantLock();

    AtomicBank(int n, int initialBalance) {
        accounts = new AtomicIntegerArray(n);
        for (int i = 0; i < n; i++) {
            accounts.set(i, initialBalance);
        }
        expectedTotal = n * initialBalance;
    }

    @Override
    public void transfer(int from, int to, int amount) {
        if (from == to || amount <= 0) {
            return;
        }

        lock.lock();
        try {
            int fromBalance = accounts.get(from);
            if (fromBalance < amount) {
                return;
            }
            accounts.addAndGet(from, -amount);
            accounts.addAndGet(to, amount);
            long t = ntransacts.incrementAndGet();
            if (t % NTEST == 0) {
                test();
            }
        } finally {
            lock.unlock();
        }
    }

    @Override
    public int size() {
        return accounts.length();
    }

    @Override
    public void test() {
        lock.lock();
        try {
            int sum = 0;
            for (int i = 0; i < accounts.length(); i++) {
                sum += accounts.get(i);
            }
            System.out.println(
                getClass().getSimpleName()
                + " -> transactions: " + ntransacts.get()
            );
        }
    }
}

```

```
        + ", sum: " + sum
        + ", expected: " + expectedTotal);
    } finally {
        lock.unlock();
    }
}
```

===== TransferThread.java =====

```
package part_1.bank;
```

```
class TransferThread extends Thread {
```

```
    private final Bank bank;
    private final int fromAccount;
    private final int maxAmount;
    private final int reps;
```

```
    TransferThread(Bank bank, int from, int maxAmount, int reps) {
        this.bank = bank;
        this.fromAccount = from;
        this.maxAmount = maxAmount;
        this.reps = reps;
    }
```

```
@Override
```

```
public void run() {
    for (int i = 0; i < reps; i++) {
        int toAccount = (int) (bank.size() * Math.random());
        int amount = (int) (maxAmount * Math.random() / 10);
        bank.transfer(fromAccount, toAccount, amount);
    }
}
```

## Задача 2

```
for f in *.java; do
    echo "===== $f ====="
    cat "$f"
    echo
done
===== BoundedBuffer.java =====
package part_2.producer_consumer;

< /**
 * Bounded buffer of integers using the guarded block pattern (wait/notifyAll).
 * Producer blocks when buffer is full; consumer blocks when buffer is empty.
 *
 * @see <a href=
 *      "https://docs.oracle.com/javase/tutorial/essential/concurrency/guardmeth.html">Guarded
 *      Blocks</a>
 */
public class BoundedBuffer {

    private final int[] buffer;
    private final int capacity;
    private int count;
    private int putIndex;
    private int takeIndex;

    public BoundedBuffer(int capacity) {
        this.capacity = capacity;
        this.buffer = new int[capacity];
        this.count = 0;
        this.putIndex = 0;
        this.takeIndex = 0;
    }

    /**
     * Inserts the value. Blocks until the buffer is not full.
     */
    public synchronized void put(int value) throws InterruptedException {
        while (count == capacity) {
            wait();
        }

        buffer[putIndex] = value;
        putIndex = (putIndex + 1) % capacity;
        count++;

        notifyAll();
    }

    /**
     * Removes and returns the next value. Blocks until the buffer is not empty.
     */
    public synchronized int take() throws InterruptedException {
        while (count == 0) {
            wait();
        }
    }
}
```

```
        int value = buffer[takeIndex];
        takeIndex = (takeIndex + 1) % capacity;
        count--;

        notifyAll();
        return value;
    }

    public int getCapacity() {
        return capacity;
    }
}
```

===== Consumer.java =====

```
package part_2.producer_consumer;
```

```
/***
 * Consumes exactly totalCount integers from the buffer and stores them for verification.
 */
public class Consumer extends Thread {

    private final BoundedBuffer buffer;
    private final int totalCount;
    private final int[] received;

    public Consumer(BoundedBuffer buffer, int totalCount) {
        this.buffer = buffer;
        this.totalCount = totalCount;
        this.received = new int[totalCount];
    }

    @Override
    public void run() {
        try {
            for (int i = 0; i < totalCount; i++) {
                received[i] = buffer.take();
            }
        } catch (InterruptedException e) {
            Thread.currentThread().interrupt();
            throw new RuntimeException(e);
        }
    }

    /**
     * Returns the received values (valid only after this thread has finished).
     */
    public int[] getReceived() {
        return received;
    }
}
```

===== Producer.java =====

```
package part_2.producer_consumer;
```

```
/***
 * Produces a sequence of integers 0, 1, 2, ..., (totalCount - 1) into the buffer.
 */
public class Producer extends Thread {
```

```

private final BoundedBuffer buffer;
private final int totalCount;

public Producer(BoundedBuffer buffer, int totalCount) {
    this.buffer = buffer;
    this.totalCount = totalCount;
}

@Override
public void run() {
    try {
        for (int i = 0; i < totalCount; i++) {
            buffer.put(i);
        }
    } catch (InterruptedException e) {
        Thread.currentThread().interrupt();
        throw new RuntimeException(e);
    }
}

```

===== ProducerConsumerDemo.java =====

```
package part_2.producer_consumer;
```

```

/**
 * Tests the Producer-Consumer pattern with a bounded buffer of integers.
 * Runs tests for buffer sizes 100, 1000, and 5000 and verifies correctness
 * (count, sum, no duplicates, correct sequence).
 */
public class ProducerConsumerDemo {

    /**
     * Number of integers to produce/consume per test (must be >= max buffer size).
     */
    public static final int TOTAL_ITEMS = 10_000;

    public static void main(String[] args) throws InterruptedException {
        System.out.println("Producer-Consumer (guarded blocks: wait/notifyAll)");
        System.out.println("Total items per run: " + TOTAL_ITEMS);
        System.out.println();

        runTest(100);
        runTest(1000);
        runTest(5000);
    }

    private static void runTest(int bufferSize) throws InterruptedException {
        System.out.println("Buffer size: " + bufferSize);

        BoundedBuffer buffer = new BoundedBuffer(bufferSize);
        Producer producer = new Producer(buffer, TOTAL_ITEMS);
        Consumer consumer = new Consumer(buffer, TOTAL_ITEMS);

        long start = System.nanoTime();

        producer.start();
        consumer.start();
        producer.join();
        consumer.join();
    }
}
```

```

long elapsedMs = (System.nanoTime() - start) / 1_000_000;

int[] received = consumer.getReceived();

// Expected sum: 0+1+...+(TOTAL_ITEMS-1) = TOTAL_ITEMS * (TOTAL_ITEMS-1) / 2
long expectedSum = (long) TOTAL_ITEMS * (TOTAL_ITEMS - 1) / 2;
long actualSum = 0;
boolean[] seen = new boolean[TOTAL_ITEMS];
boolean orderOk = true;

for (int i = 0; i < TOTAL_ITEMS; i++) {
    int v = received[i];
    actualSum += v;
    if (v < 0 || v >= TOTAL_ITEMS || seen[v]) {
        orderOk = false;
    } else {
        seen[v] = true;
    }
}
boolean noDuplicates = true;
for (int i = 0; i < TOTAL_ITEMS; i++) {
    if (!seen[i]) {
        noDuplicates = false;
        break;
    }
}

boolean ok = (actualSum == expectedSum && noDuplicates && received.length ==
TOTAL_ITEMS && orderOk);

System.out.println(" Count: " + received.length + " (expected " + TOTAL_ITEMS + ")");
System.out.println(" Sum: " + actualSum + " (expected " + expectedSum + ")");
System.out.println(" No duplicates, all 0.." + (TOTAL_ITEMS - 1) + ":" + noDuplicates);
System.out.println(" Result: " + (ok ? "CORRECT" : "INCORRECT"));
System.out.println(" Time: " + elapsedMs + " ms");
System.out.println();
}
}

```

### Задача 3

```
for f in *.java; do
    echo "===== $f ====="
    cat "$f"
    echo
done
===== AssistantThread.java =====
package part_3.gradebook;

import java.util.Random;

< /**
 * Thread representing one assistant: assigns grades for a single group
 * (all students, all weeks). Assistant index 0 -> group 0, etc.
 */
public class AssistantThread extends Thread {

    private final Gradebook gradebook;
    private final int assistantIndex;
    private final Random rnd = new Random();

    public AssistantThread(Gradebook gradebook, int assistantIndex) {
        this.gradebook = gradebook;
        this.assistantIndex = assistantIndex;
    }

    @Override
    public void run() {
        int group = assistantIndex;
        if (group >= Gradebook.NUM_GROUPS) {
            return;
        }
        for (int week = 0; week < gradebook.getWeeksCount(); week++) {
            for (int student = 0; student < gradebook.getStudentsPerGroup(); student++) {
                int grade = 50 + rnd.nextInt(51); // 50–100
                gradebook.setGrade(group, student, week, grade);
            }
        }
    }
}

===== Gradebook.java =====
package part_3.gradebook;

import java.util.concurrent.locks.Lock;
import java.util.concurrent.locks.ReentrantLock;

< /**
 * Electronic gradebook: grades for one discipline, 3 groups of students, by week.
 * Grades are on a 0–100 scale. All updates are protected by a single lock
 * so that lecturer and assistants can write concurrently without lost updates.
 */
public class Gradebook {

    public static final int NUM_GROUPS = 3;
```

```

public static final int MIN_GRADE = 0;
public static final int MAX_GRADE = 100;

private final int studentsPerGroup;
private final int weeksCount;
/** grades[group][student][week] */
private final int[][][] grades;
private final Lock lock = new ReentrantLock();

public Gradebook(int studentsPerGroup, int weeksCount) {
    this.studentsPerGroup = studentsPerGroup;
    this.weeksCount = weeksCount;
    this.grades = new int[NUM_GROUPS][studentsPerGroup][weeksCount];
}

public void setGrade(int group, int student, int week, int value) {
    if (group < 0 || group >= NUM_GROUPS || student < 0 || student >= studentsPerGroup
        || week < 0 || week >= weeksCount) {
        return;
    }
    int grade = Math.max(MIN_GRADE, Math.min(MAX_GRADE, value));
    lock.lock();
    try {
        grades[group][student][week] = grade;
    } finally {
        lock.unlock();
    }
}

public int getGrade(int group, int student, int week) {
    lock.lock();
    try {
        return grades[group][student][week];
    } finally {
        lock.unlock();
    }
}

public int getStudentsPerGroup() {
    return studentsPerGroup;
}

public int getWeeksCount() {
    return weeksCount;
}

/** Builds a string table of all grades and group averages. */
public String formatTable() {
    lock.lock();
    try {
        StringBuilder sb = new StringBuilder();
        sb.append("Electronic gradebook (3 groups, ").append(studentsPerGroup)
            .append(" students, ").append(weeksCount).append(" weeks)\n\n");
        for (int g = 0; g < NUM_GROUPS; g++) {
            sb.append("--- Group ").append(g + 1).append(" ---\n");
            sb.append("Student ");
            for (int w = 0; w < weeksCount; w++) {
                sb.append(" Week").append(w + 1).append(" ");
            }
        }
    }
}

```

```
sb.append(" Avg\n");

long groupSum = 0;
int groupCount = 0;
for (int s = 0; s < studentsPerGroup; s++) {
    sb.append(String.format("%4d    ", s + 1));
    int rowSum = 0;
    for (int w = 0; w < weeksCount; w++) {
        int v = grades[g][s][w];
        sb.append(String.format("%4d  ", v));
        rowSum += v;
        groupSum += v;
        groupCount++;
    }
    double avg = weeksCount > 0 ? (double) rowSum / weeksCount : 0;
    sb.append(String.format(" %5.1f\n", avg));
}
double groupAvg = groupCount > 0 ? (double) groupSum / groupCount : 0;
sb.append("Group avg: ").append(String.format("%.1f", groupAvg)).append("\n\n");
}

return sb.toString();
} finally {
    lock.unlock();
}
}
```

## ===== GradebookDemo.java =====

package part 3.gradebook;

/\* \*

\* Runs the electronic gradebook: one lecturer and three assistants assign grades concurrently; after they finish, the grade table is printed.  
\*/

### public class Gradebook

```
public static final int STUDENTS_PER_GROUP = 5;  
public static final int WEEKS_COUNT = 4;
```

```
public static void main(String[] args) throws InterruptedException {
```

```
System.out.println("Electronics gradebook: 1 lecturer + 3 assistants")
```

```
System.out.println("Electronic gradebook. " + LECTURE + " lectures");
System.out.println("3 groups, " + STUDENTS_PER_GROUP + " students per group, " +
WEEKS_COUNT + " weeks");
```

System.out.println("Grades 0-100. All four threads write concurrently (ReentrantLock)\n");

Gradebook gradebook = new Gradebook(STUDENTS, PER, GROUP, WEEKS, COUNT);

```
LecturerThread lecturer = new LecturerThread(gradebook);
```

```
LecturerThread lecturer = new LecturerThread(gradebook);  
AssistantThread assistant0 = new AssistantThread(gradebook, 0);
```

```
AssistantThread assistant0 = new AssistantThread(gradebook, 0);  
AssistantThread assistant1 = new AssistantThread(gradebook, 1);
```

```
AssistantThread assistant1 = new AssistantThread(gradebook, 1);  
AssistantThread assistant2 = new AssistantThread(gradebook, 2);
```

lecturer.start();

```
lecturer.start(),  
assistant0.start();
```

assistant0.start(),  
assistant1.start();

assistant1.start(),  
assistant2.start();

```
lecturer.join();
```

lecturer.join(),  
assistant.join();

```
assistant0.join(),  
assistant1.join();
```

```

        assistant2.join();

        System.out.println(gradebook.formatTable());
    }

===== LecturerThread.java =====
package part_3.gradebook;

import java.util.Random;

/**
 * Thread representing the lecturer: each week assigns a grade (0–100) to every
 * student in every group.
 */
public class LecturerThread extends Thread {

    private final Gradebook gradebook;
    private final Random rnd = new Random();

    public LecturerThread(Gradebook gradebook) {
        this.gradebook = gradebook;
    }

    @Override
    public void run() {
        for (int week = 0; week < gradebook.getWeeksCount(); week++) {
            for (int group = 0; group < Gradebook.NUM_GROUPS; group++) {
                for (int student = 0; student < gradebook.getStudentsPerGroup(); student++) {
                    int grade = 60 + rnd.nextInt(41); // 60–100
                    gradebook.setGrade(group, student, week, grade);
                }
            }
        }
    }
}

```

#### Задача 4

```
for f in *.java; do
    echo "===== $f ====="
    cat "$f"
    echo
done
===== PrinterDemo.java =====
package part_4.printers;

/**
 * Task 4: Three threads print '|', '\', '/'
 */
public class PrinterDemo {

    public static final int LINES = 50;
    public static final int CHARS_PER_LINE = 150;
    private static final int PRINTS_PER_THREAD = LINES * CHARS_PER_LINE / 3;

    public static void main(String[] args) throws InterruptedException {
        System.out.println("Part 4: Three threads printing | \\ \n");
        runUncoordinated();
        runCoordinated();
    }

    private static final Object uncoordLock = new Object();
    private static int uncoordCount = 0;

    static class UncoordinatedSymbolThread extends Thread {
        private final String symbol;

        UncoordinatedSymbolThread(String symbol) {
            this.symbol = symbol;
        }

        @Override
        public void run() {
            for (int i = 0; i < PRINTS_PER_THREAD; i++) {
                synchronized (uncoordLock) {
                    System.out.print(symbol);
                    uncoordCount++;
                    if (uncoordCount % CHARS_PER_LINE == 0) {
                        System.out.println();
                    }
                }
            }
        }
    }

    private static void runUncoordinated() throws InterruptedException {
        System.out.println("Uncoordinated: " + LINES + " lines, " + CHARS_PER_LINE + " chars per
line");
        uncoordCount = 0;

        Thread t1 = new UncoordinatedSymbolThread("|");
        Thread t2 = new UncoordinatedSymbolThread("\\");
        Thread t3 = new UncoordinatedSymbolThread("/");
    }
}
```

```

t1.start();
t2.start();
t3.start();

t1.join();
t2.join();
t3.join();

System.out.println();
}

private static final Object lock = new Object();
private static int state = 0;

static class CoordinatedSymbolThread extends Thread {
    private final String symbol;
    private final int targetState;

    CoordinatedSymbolThread(String symbol, int targetState) {
        this.symbol = symbol;
        this.targetState = targetState;
    }

    @Override
    public void run() {
        for (int i = 0; i < PRINTS_PER_THREAD; i++) {
            synchronized (lock) {
                while (state % 3 != targetState) {
                    try {
                        lock.wait();
                    } catch (InterruptedException e) {
                        e.printStackTrace();
                    }
                }
                System.out.print(symbol);
                state++;
                if (state % CHARS_PER_LINE == 0) {
                    System.out.println();
                }
                lock.notifyAll();
            }
        }
    }
}

private static void runCoordinated() throws InterruptedException {
    System.out.println(
        "Coordinated: " + LINES + " lines, " + CHARS_PER_LINE + " chars per line, pattern "
        + "\\\\" + "\\\\" + "\\\\" + "\\\\"");
    state = 0;

    Thread t1 = new CoordinatedSymbolThread("|", 0);
    Thread t2 = new CoordinatedSymbolThread("\\\\", 1);
    Thread t3 = new CoordinatedSymbolThread("/", 2);

    t1.start();
    t2.start();
    t3.start();
    t1.join();
}

```

```
t2.join();
t3.join();
System.out.println();
}
```