МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ФАКУЛЬТЕТ ПРИКЛАДНОЙ МАТЕМАТИКИ И ИНФОРМАТИКИ

СЕРГИЕНКО ЛЕВ ЭДУАРДОВИЧ

Отчет по
Лабораторная работа 7
Параллельные вычисления в Java Модели создания и функционирования потоков: Производитель и потребитель

Преподаватель

Кондратьева О.М.

2025

Задание 1.

Вычисление числа Пи методом Монте-Карло, модель Производитель / Потребитель.

Последовательная программа.

```
package lab7;

import java.util.Random;

public class PiSequential {
    public static void main(String[] args) {
        long iterations = 100_000_000;
        long pointsInCircle = 0;
        Random random = new Random(100000000L);

        long startTime = System.currentTimeMillis();
        for (long i = 0; i < iterations; i++) {
            double x = random.nextDouble();
            double y = random.nextDouble();

            if (x * x + y * y <= 1) {
                pointsInCircle++;
            }
        }
        double pi = 4.0 * pointsInCircle / iterations;

        long endTime = System.currentTimeMillis();

        System.out.println("Приближенное значение pi: " + pi);
        System.out.println("Точное значение pi: " + Math.PI);
        System.out.printf("Погрешность: %.10f%n", Math.abs(pi - Math.PI));
        System.out.println("Время выполнения (мс): " + (endTime - startTime));
    }
}
```

Многопоточная программа.

```
package lab7;

import java.util.Random;
import java.util.concurrent.BlockingQueue;
import java.util.concurrent.LinkedBlockingQueue;
import java.util.concurrent.atomic.AtomicLong;

public class PiParallel {
    private static final int NUM_POINTS = 100_000_000;
    private static final int NUM_PRODUCERS = 4;
```

```
private static final int NUM_CONSUMERS = 4;
private static final int QUEUE_CAPACITY = 1000;

private static final Random random = new Random(100000000L);

private static final Point POISON_PILL = new Point(2, 2);

static class Point {
    double x, y;

    public Point(double x, double y) {
        this.x = x;
        this.y = y;
    }
}

public static void main(String[] args) throws InterruptedException {
    BlockingQueue<Point> queue = new LinkedBlockingQueue<>(QUEUE_CAPACITY);
    AtomicLong insideCircle = new AtomicLong(0);
    AtomicLong producedCount = new AtomicLong(0);

    // Создаём потоки-производители
    Thread[] producers = new Thread[NUM_PRODUCERS];
    for (int i = 0; i < NUM_PRODUCERS; i++) {
        producers[i] = new Thread(() -> {
            while (true) {
                long current = producedCount.getAndIncrement();
                if (current >= NUM_POINTS) {
                    break;
                }
                double x = random.nextDouble();
                double y = random.nextDouble();
                Point p = new Point(x, y);
                try {
                    queue.put(p);
                } catch (InterruptedException e) {
                    Thread.currentThread().interrupt();
                    break;
                }
            }
        });
    }

    // Создаём потоки-потребители
    Thread[] consumers = new Thread[NUM_CONSUMERS];
    for (int i = 0; i < NUM_CONSUMERS; i++) {
        consumers[i] = new Thread(() -> {
            while (true) {
                try {
                    Point p = queue.take();
                    if (p == POISON_PILL) {
```

```
                        break;
                    }
                    if (p.x * p.x + p.y * p.y <= 1) {
                        insideCircle.incrementAndGet();
                    }
                } catch (InterruptedException e) {
                    Thread.currentThread().interrupt();
                    break;
                }
            }
        });
    }

    long startTime = System.currentTimeMillis();
    for (int i = 0; i < NUM_PRODUCERS; i++) {
        producers[i].start();
    }
    for (int i = 0; i < NUM_CONSUMERS; i++) {
        consumers[i].start();
    }

    for (Thread producer : producers) {
        producer.join();
    }

    for (int i = 0; i < NUM_CONSUMERS; i++) {
        queue.put(POISON_PILL);
    }

    for (Thread consumer : consumers) {
        consumer.join();
    }
    long endTime = System.currentTimeMillis();

    double piEstimate = 4.0 * insideCircle.get() / NUM_POINTS;

    System.out.println("Количество потоков производителей: " +
NUM_PRODUCERS);
    System.out.println("Количество потоков потребителей: " + NUM_CONSUMERS);

    System.out.println("Приближенное значение pi: " + piEstimate);
    System.out.println("Точное значение pi: " + Math.PI);
    System.out.printf("Погрешность: %.10f%n", Math.abs(piEstimate -
Math.PI));
    System.out.println("Время выполнения (мс): " + (endTime - startTime));
  }
}
```

Таблица с результатами экспериментов.

Размерность задачи	Время выполнения последовательной программы, мс	Параллельная программа - 1 с / 1 р			Параллельная программа - 1 с / 2 р			Параллельная программа - 2 с / 1 р			Параллельная программа - 2 с / 2 р		
		Время выполнения	Ускорение	Эффективность	Время выполнения	Ускорение	Эффективность	Время выполнения	Ускорение	Эффективность	Время выполнения	Ускорение	Эффективность
1 000 000	30	184	0,1630434783	0,08152173913	205	0,1463414634	0,0487804878	197	0,152284264	0,05076142132	193	0,1554404145	0,03886010363
10 000 000	205	1927	0,1063829787	0,05319148936	1976	0,1037449393	0,03458164642	1542	0,1329442283	0,04431474276	1904	0,1076680672	0,02691701681
100 000 000	1619	15850	0,1021451104	0,05107255521	15961	0,1014347472	0,0338115824	12267	0,1319801092	0,04399336975	16247	0,099649166	0,0249122915

Задание 2.

Разработать многопоточное приложение. Использовать слово synchronized. Все сущности, желающие получить доступ к ресурсу, должны быть потоками.

Автостоянка. Доступно несколько машиномест. На одном месте может находиться только один автомобиль.

Текст программы.

```
package lab7;

import java.util.ArrayList;
import java.util.List;
import java.util.Random;
import java.util.concurrent.TimeUnit;

import static java.lang.System.out;

public class ParkingLotSynchronized {
    private static final int PARKING_CAPACITY = 5; // Количество мест на стоянке
    private static final int CARS_COUNT = 10; // Количество автомобилей
    private static final int MAX_PARKING_TIME = 3; // Максимальное время стоянки в секундах
    private static final int MAX_DRIVING_TIME = 5; // Максимальное время вождения в секундах

    // Массив парковочных мест: 0 - свободно, иначе - ID припаркованного автомобиля
```

```
    private final int[] parkingSpots = new int[PARKING_CAPACITY];

    public synchronized int parkCar(int carId) {
        for (int i = 0; i < parkingSpots.length; i++) {
            if (parkingSpots[i] == 0) {
                parkingSpots[i] = carId;
                System.out.println("Автомобиль #" + carId + "
припарковался на месте " + (i + 1));
                printParkingStatus();
                return i;
            }
        }
        System.out.println("Автомобиль #" + carId + " не нашел
свободного места");
        return -1;
    }

    public synchronized void leaveParkingLot(int carId, int spotIndex) {
        if (spotIndex >= 0 && spotIndex < parkingSpots.length &&
parkingSpots[spotIndex] == carId) {
            parkingSpots[spotIndex] = 0;
            System.out.println("Автомобиль #" + carId + " покинул место
" + (spotIndex + 1));
            printParkingStatus();
            notify();
        }
    }

    private void printParkingStatus() {
        final String RESET = "\u001B[0m";
        final String GREEN = "\u001B[32m";
        final String RED = "\u001B[31m";

        int occupiedCount = 0;
        StringBuilder status = new StringBuilder("Статус мест: [");

        for (int i = 0; i < parkingSpots.length; i++) {
            if (parkingSpots[i] != 0) {
                status.append(RED).append("Место ").append(i +
1).append(": авто #").append(parkingSpots[i])
                        .append(RESET);
                occupiedCount++;
            } else {
                status.append(GREEN).append("Место ").append(i +
1).append(": свободно").append(RESET);
            }
```

```
            if (i < parkingSpots.length - 1) {
                status.append(", ");
            }
        }
        status.append("]");

        out.println(status);
        out.println("Статус стоянки: занято " + RED + occupiedCount +
RESET + " из " + GREEN + PARKING_CAPACITY + RESET
                + " мест");
    }

    static class Car extends Thread {
        private final int carId;
        private final ParkingLotSynchronized parkingLot;
        private final Random random = new Random();
        private int parkedSpotIndex = -1; // Индекс места, где
припаркован автомобиль (-1 если не припаркован)

        public Car(int carId, ParkingLotSynchronized parkingLot) {
            this.carId = carId;
            this.parkingLot = parkingLot;
            setName("Car-" + carId);
        }

        @Override
        public void run() {
            try {
                while (!Thread.interrupted()) {
                    synchronized (parkingLot) {
                        while ((parkedSpotIndex =
parkingLot.parkCar(carId)) == -1) {
                            System.out.println("Автомобиль #" + carId +
" ожидает освобождения места...");
                            parkingLot.wait(); // Ожидаем, пока не
освободится место
                        }
                    }

                    // Стоим на парковке
                    int parkingTime = random.nextInt(MAX_PARKING_TIME) +
1;
                    System.out.println("Автомобиль #" + carId + " будет
стоять " + parkingTime + " сек.");
                    TimeUnit.SECONDS.sleep(parkingTime);
```

```
                    // Покидаем парковку
                    parkingLot.leaveParkingLot(carId, parkedSpotIndex);
                    parkedSpotIndex = -1;

                    // Ездим по городу
                    int drivingTime = random.nextInt(MAX_DRIVING_TIME) +
1;
                    System.out.println("Автомобиль #" + carId + " уехал
на " + drivingTime + " сек.");
                    TimeUnit.SECONDS.sleep(drivingTime);
                }
            } catch (InterruptedException e) {
                System.out.println("Автомобиль #" + carId + " завершил
работу");
                // Если прерываем авто, когда оно на парковке, освободим
место
                if (parkedSpotIndex != -1) {
                    parkingLot.leaveParkingLot(carId, parkedSpotIndex);
                }
            }
        }
    }

    public static void main(String[] args) {
        ParkingLotSynchronized parkingLot = new
ParkingLotSynchronized();
        List<Car> cars = new ArrayList<>();

        for (int i = 1; i <= CARS_COUNT; i++) {
            Car car = new Car(i, parkingLot);
            cars.add(car);
            car.start();
        }

        try {
            TimeUnit.SECONDS.sleep(10);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }

        for (Car car : cars) {
            car.interrupt();
            try {
                car.join(1000); // Даем потоку секунду на завершение
            } catch (InterruptedException e) {
```

```
                e.printStackTrace();
            }
        }

        System.out.println("Программа завершена");
    }
}
```

Снимок окна работы приложения.

```
Автомобиль #1 припарковался на месте 1
Статус мест: [Место 1: авто #1, Место 2: свободно, Место 3: свободно, Место 4: свободно, Место 5: свободно]
Статус стоянки: занято 1 из 5 мест
Автомобиль #10 припарковался на месте 2
Автомобиль #1 будет стоять 3 сек.
Статус мест: [Место 1: авто #1, Место 2: авто #10, Место 3: свободно, Место 4: свободно, Место 5: свободно]
Статус стоянки: занято 2 из 5 мест
Автомобиль #10 будет стоять 1 сек.
Автомобиль #9 припарковался на месте 3
Статус мест: [Место 1: авто #1, Место 2: авто #10, Место 3: авто #9, Место 4: свободно, Место 5: свободно]
Статус стоянки: занято 3 из 5 мест
Автомобиль #9 будет стоять 2 сек.
Автомобиль #8 припарковался на месте 4
Статус мест: [Место 1: авто #1, Место 2: авто #10, Место 3: авто #9, Место 4: авто #8, Место 5: свободно]
Статус стоянки: занято 4 из 5 мест
Автомобиль #8 будет стоять 2 сек.
Автомобиль #7 припарковался на месте 5
Статус мест: [Место 1: авто #1, Место 2: авто #10, Место 3: авто #9, Место 4: авто #8, Место 5: авто #7]
Статус стоянки: занято 5 из 5 мест
Автомобиль #7 будет стоять 3 сек.
Автомобиль #6 не нашел свободного места
Автомобиль #6 ожидает освобождения места...
Автомобиль #5 не нашел свободного места
Автомобиль #5 ожидает освобождения места...
Автомобиль #4 не нашел свободного места
Автомобиль #4 ожидает освобождения места...
Автомобиль #2 не нашел свободного места
Автомобиль #2 ожидает освобождения места...
Автомобиль #3 не нашел свободного места
Автомобиль #3 ожидает освобождения места...
Автомобиль #10 покинул место 2
Статус мест: [Место 1: авто #1, Место 2: свободно, Место 3: авто #9, Место 4: авто #8, Место 5: авто #7]
Статус стоянки: занято 4 из 5 мест
Автомобиль #6 припарковался на месте 2
Статус мест: [Место 1: авто #1, Место 2: авто #6, Место 3: авто #9, Место 4: авто #8, Место 5: авто #7]
Статус стоянки: занято 5 из 5 мест
Автомобиль #6 будет стоять 2 сек.
Автомобиль #10 уехал на 1 сек.
Автомобиль #9 покинул место 3
Статус мест: [Место 1: авто #1, Место 2: авто #6, Место 3: свободно, Место 4: авто #8, Место 5: авто #7]
Статус стоянки: занято 4 из 5 мест
Автомобиль #9 уехал на 1 сек.
Автомобиль #5 припарковался на месте 3
Статус мест: [Место 1: авто #1, Место 2: авто #6, Место 3: авто #5, Место 4: авто #8, Место 5: авто #7]
Статус стоянки: занято 5 из 5 мест
Автомобиль #5 будет стоять 2 сек.
Автомобиль #8 покинул место 4
Статус мест: [Место 1: авто #1, Место 2: авто #6, Место 3: авто #5, Место 4: свободно, Место 5: авто #7]
Статус стоянки: занято 4 из 5 мест
Автомобиль #8 уехал на 3 сек.
Автомобиль #4 припарковался на месте 4
Статус мест: [Место 1: авто #1, Место 2: авто #6, Место 3: авто #5, Место 4: авто #4, Место 5: авто #7]
Статус стоянки: занято 5 из 5 мест
Автомобиль #4 будет стоять 2 сек.
Автомобиль #10 не нашел свободного места
Автомобиль #10 ожидает освобождения места...
Автомобиль #1 покинул место 1
Статус мест: [Место 1: свободно, Место 2: авто #6, Место 3: авто #5, Место 4: авто #4, Место 5: авто #7]
Статус стоянки: занято 4 из 5 мест
Автомобиль #1 уехал на 1 сек.
Автомобиль #2 припарковался на месте 1
Статус мест: [Место 1: авто #2, Место 2: авто #6, Место 3: авто #5, Место 4: авто #4, Место 5: авто #7]
Статус стоянки: занято 5 из 5 мест
Автомобиль #2 будет стоять 1 сек.
Автомобиль #7 покинул место 5
Статус мест: [Место 1: авто #2, Место 2: авто #6, Место 3: авто #5, Место 4: авто #4, Место 5: свободно]
Статус стоянки: занято 4 из 5 мест
```

Задание 3.

Разработать многопоточное приложение. Использовать Семафор. Все сущности, желающие получить доступ к ресурсу, должны быть потоками. **Автостоянка**. Доступно несколько машиномест. На одном месте может находиться только один автомобиль.

Текст программы.

```
package lab7;

import java.util.ArrayList;
import java.util.List;
import java.util.Random;
import java.util.concurrent.Semaphore;
import java.util.concurrent.TimeUnit;
import java.util.concurrent.locks.Lock;
import java.util.concurrent.locks.ReentrantLock;

import static java.lang.System.out;

public class ParkingLotSemaphore {
    private static final int PARKING_CAPACITY = 5;
    private static final int CARS_COUNT = 10;
    private static final int MAX_PARKING_TIME = 3;
    private static final int MAX_DRIVING_TIME = 5;

    private final int[] parkingSpots = new int[PARKING_CAPACITY];

    // для контроля доступа к парковочным местам
    private final Semaphore semaphore = new Semaphore(PARKING_CAPACITY,
true);
    // для синхронизации доступа к массиву parkingSpots
    private final Lock spotsLock = new ReentrantLock();

    public int parkCar(int carId) throws InterruptedException {
        semaphore.acquire();

        spotsLock.lock();
        try {

            for (int i = 0; i < parkingSpots.length; i++) {
                if (parkingSpots[i] == 0) {
                    parkingSpots[i] = carId;
                    System.out.println("Автомобиль #" + carId + "
```

```
припарковался на месте " + (i + 1));
                    printParkingStatus();
                    return i;
                }
            }

            // Эта ситуация не должна произойти, так как семафор
гарантирует наличие
            // свободного места
            System.out.println("ОШИБКА: Автомобиль #" + carId + "
получил разрешение, но все места заняты");
            semaphore.release();
            return -1;
        } finally {
            spotsLock.unlock();
        }
    }

    public void leaveParkingLot(int carId, int spotIndex) {
        spotsLock.lock();
        try {
            if (spotIndex >= 0 && spotIndex < parkingSpots.length &&
parkingSpots[spotIndex] == carId) {
                parkingSpots[spotIndex] = 0;
                System.out.println("Автомобиль #" + carId + " покинул
место " + (spotIndex + 1));
                printParkingStatus();

                semaphore.release();
            }
        } finally {
            spotsLock.unlock();
        }
    }

    private void printParkingStatus() {
        final String RESET = "\u001B[0m";
        final String GREEN = "\u001B[32m";
        final String RED = "\u001B[31m";

        int occupiedCount = 0;
        StringBuilder status = new StringBuilder("Статус мест: [");

        for (int i = 0; i < parkingSpots.length; i++) {
            if (parkingSpots[i] != 0) {
                status.append(RED).append("Место ").append(i +
```

```
1).append(": авто #").append(parkingSpots[i])
                        .append(RESET);
                occupiedCount++;
            } else {
                status.append(GREEN).append("Место ").append(i +
1).append(": свободно").append(RESET);
            }

            if (i < parkingSpots.length - 1) {
                status.append(", ");
            }
        }
        status.append("]");

        out.println(status);
        out.println("Статус стоянки: занято " + RED + occupiedCount +
RESET + " из " + GREEN + PARKING_CAPACITY + RESET
                + " мест (доступно разрешений: " + GREEN +
semaphore.availablePermits() + RESET + ")");
    }

    static class Car extends Thread {
        private final int carId;
        private final ParkingLotSemaphore parkingLot;
        private final Random random = new Random();
        private int parkedSpotIndex = -1;

        public Car(int carId, ParkingLotSemaphore parkingLot) {
            this.carId = carId;
            this.parkingLot = parkingLot;
            setName("Car-" + carId);
        }

        @Override
        public void run() {
            try {
                while (!Thread.interrupted()) {
                    // Пытаемся припарковаться
                    System.out.println("Автомобиль #" + carId + " ищет
место для парковки...");
                    parkedSpotIndex = parkingLot.parkCar(carId);

                    if (parkedSpotIndex >= 0) {
                        // Стоим на парковке
                        int parkingTime =
random.nextInt(MAX_PARKING_TIME) + 1;
```

```
                        System.out.println("Автомобиль #" + carId + "
будет стоять " + parkingTime + " сек.");
                        TimeUnit.SECONDS.sleep(parkingTime);

                        // Покидаем парковку
                        parkingLot.leaveParkingLot(carId,
parkedSpotIndex);
                        parkedSpotIndex = -1;

                        // Ездим по городу
                        int drivingTime =
random.nextInt(MAX_DRIVING_TIME) + 1;
                        System.out.println("Автомобиль #" + carId + "
уехал на " + drivingTime + " сек.");
                        TimeUnit.SECONDS.sleep(drivingTime);
                    }
                }
            } catch (InterruptedException e) {
                System.out.println("Автомобиль #" + carId + " завершил
работу");
                if (parkedSpotIndex != -1) {
                    parkingLot.leaveParkingLot(carId, parkedSpotIndex);
                }
            }
        }
    }

    public static void main(String[] args) {
        ParkingLotSemaphore parkingLot = new ParkingLotSemaphore();
        List<Car> cars = new ArrayList<>();

        System.out.println("Запуск симуляции автостоянки с использованием
семафора");
        System.out.println("Количество мест на стоянке: " +
PARKING_CAPACITY);
        System.out.println("Количество автомобилей: " + CARS_COUNT);

        for (int i = 1; i <= CARS_COUNT; i++) {
            Car car = new Car(i, parkingLot);
            cars.add(car);
            car.start();
        }

        try {
            TimeUnit.SECONDS.sleep(10);
        } catch (InterruptedException e) {
```

```
            e.printStackTrace();
        }

        System.out.println("Завершение работы автомобилей...");
        for (Car car : cars) {
            car.interrupt();
            try {
                car.join(1000);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }

        System.out.println("Программа завершена");
    }
}
```

Снимок окна работы приложения.

```
Запуск симуляции автостоянки с использованием семафора
Количество мест на стоянке: 5
Количество автомобилей: 10
Автомобиль #1 ищет место для парковки...
Автомобиль #5 ищет место для парковки...
Автомобиль #10 ищет место для парковки...
Автомобиль #8 ищет место для парковки...
Автомобиль #2 ищет место для парковки...
Автомобиль #9 ищет место для парковки...
Автомобиль #7 ищет место для парковки...
Автомобиль #4 ищет место для парковки...
Автомобиль #3 ищет место для парковки...
Автомобиль #6 ищет место для парковки...
Автомобиль #1 припарковался на месте 1
Статус мест: [Место 1: авто #1, Место 2: свободно, Место 3: свободно, Место 4: свободно, Место 5: свободно]
Статус стоянки: занято 1 из 5 мест (доступно разрешений: 0)
Автомобиль #5 припарковался на месте 2
Автомобиль #1 будет стоять 2 сек.
Статус мест: [Место 1: авто #1, Место 2: авто #5, Место 3: свободно, Место 4: свободно, Место 5: свободно]
Статус стоянки: занято 2 из 5 мест (доступно разрешений: 0)
Автомобиль #5 будет стоять 1 сек.
Автомобиль #10 припарковался на месте 3
Статус мест: [Место 1: авто #1, Место 2: авто #5, Место 3: авто #10, Место 4: свободно, Место 5: свободно]
Статус стоянки: занято 3 из 5 мест (доступно разрешений: 0)
Автомобиль #10 будет стоять 2 сек.
Автомобиль #8 припарковался на месте 4
Статус мест: [Место 1: авто #1, Место 2: авто #5, Место 3: авто #10, Место 4: авто #8, Место 5: свободно]
Статус стоянки: занято 4 из 5 мест (доступно разрешений: 0)
Автомобиль #8 будет стоять 1 сек.
Автомобиль #2 припарковался на месте 5
Статус мест: [Место 1: авто #1, Место 2: авто #5, Место 3: авто #10, Место 4: авто #8, Место 5: авто #2]
Статус стоянки: занято 5 из 5 мест (доступно разрешений: 0)
Автомобиль #2 будет стоять 3 сек.
Автомобиль #8 покинул место 4
Статус мест: [Место 1: авто #1, Место 2: авто #5, Место 3: авто #10, Место 4: свободно, Место 5: авто #2]
Статус стоянки: занято 4 из 5 мест (доступно разрешений: 0)
Автомобиль #5 покинул место 2
Статус мест: [Место 1: авто #1, Место 2: свободно, Место 3: авто #10, Место 4: свободно, Место 5: авто #2]
Автомобиль #8 уехал на 3 сек.
Статус стоянки: занято 3 из 5 мест (доступно разрешений: 0)
Автомобиль #5 уехал на 3 сек.
Автомобиль #9 припарковался на месте 2
Статус мест: [Место 1: авто #1, Место 2: авто #9, Место 3: авто #10, Место 4: свободно, Место 5: авто #2]
Статус стоянки: занято 4 из 5 мест (доступно разрешений: 0)
Автомобиль #9 будет стоять 1 сек.
Автомобиль #7 припарковался на месте 4
Статус мест: [Место 1: авто #1, Место 2: авто #9, Место 3: авто #10, Место 4: авто #7, Место 5: авто #2]
Статус стоянки: занято 5 из 5 мест (доступно разрешений: 0)
Автомобиль #7 будет стоять 3 сек.
Автомобиль #1 покинул место 1
Статус мест: [Место 1: свободно, Место 2: авто #9, Место 3: авто #10, Место 4: авто #7, Место 5: авто #2]
Статус стоянки: занято 4 из 5 мест (доступно разрешений: 0)
Автомобиль #1 уехал на 3 сек.
Автомобиль #10 покинул место 3
Статус мест: [Место 1: свободно, Место 2: авто #9, Место 3: свободно, Место 4: авто #7, Место 5: авто #2]
Статус стоянки: занято 3 из 5 мест (доступно разрешений: 0)
Автомобиль #10 уехал на 4 сек.
Автомобиль #3 припарковался на месте 1
Статус мест: [Место 1: авто #3, Место 2: авто #9, Место 3: свободно, Место 4: авто #7, Место 5: авто #2]
Статус стоянки: занято 4 из 5 мест (доступно разрешений: 0)
Автомобиль #3 будет стоять 3 сек.
Автомобиль #4 припарковался на месте 3
Статус мест: [Место 1: авто #3, Место 2: авто #9, Место 3: авто #4, Место 4: авто #7, Место 5: авто #2]
Статус стоянки: занято 5 из 5 мест (доступно разрешений: 0)
Автомобиль #4 будет стоять 2 сек.
Автомобиль #9 покинул место 2
Статус мест: [Место 1: авто #3, Место 2: свободно, Место 3: авто #4, Место 4: авто #7, Место 5: авто #2]
Статус стоянки: занято 4 из 5 мест (доступно разрешений: 0)
Автомобиль #6 припарковался на месте 2
Статус мест: [Место 1: авто #3, Место 2: авто #6, Место 3: авто #4, Место 4: авто #7, Место 5: авто #2]
Статус стоянки: занято 5 из 5 мест (доступно разрешений: 0)
```

Задание 4.

Разработать многопоточное приложение. Использовать Условную переменную. Все сущности, желающие получить доступ к ресурсу, должны быть потоками.

Автостоянка. Доступно несколько машиномест. На одном месте может находиться только один автомобиль.

Текст программы.

```
package lab7;

import java.util.ArrayList;
import java.util.List;
import java.util.Random;
import java.util.concurrent.TimeUnit;
import java.util.concurrent.locks.Condition;
import java.util.concurrent.locks.Lock;
import java.util.concurrent.locks.ReentrantLock;

import static java.lang.System.out;

public class ParkingLotCondition {
    private static final int PARKING_CAPACITY = 5;
    private static final int CARS_COUNT = 10;
    private static final int MAX_PARKING_TIME = 3;
    private static final int MAX_DRIVING_TIME = 5;

    private final int[] parkingSpots = new int[PARKING_CAPACITY];

    private int occupiedSpots = 0;

    // Блокировка для синхронизации доступа к состоянию парковки
    private final Lock lock = new ReentrantLock();

    // Условная переменная для ожидания освобождения места
    private final Condition spotAvailable = lock.newCondition();

    public int parkCar(int carId) throws InterruptedException {
        lock.lock();
        try {
            // Ждем, пока не появится свободное место
            while (occupiedSpots >= PARKING_CAPACITY) {
                System.out.println("Автомобиль #" + carId + " ожидает
освобождения места...");
```

```
                spotAvailable.await();
            }

            for (int i = 0; i < parkingSpots.length; i++) {
                if (parkingSpots[i] == 0) {
                    parkingSpots[i] = carId;
                    occupiedSpots++;
                    System.out.println("Автомобиль #" + carId + "
припарковался на месте " + (i + 1));
                    printParkingStatus();
                    return i;
                }
            }

            System.out.println(
                    "ОШИБКА: Автомобиль #" + carId + " не нашел
свободного места, хотя счетчик показывает наличие");
            return -1;
        } finally {
            lock.unlock();
        }
    }

    public void leaveParkingLot(int carId, int spotIndex) {
        lock.lock();
        try {
            if (spotIndex >= 0 && spotIndex < parkingSpots.length &&
parkingSpots[spotIndex] == carId) {
                parkingSpots[spotIndex] = 0;
                occupiedSpots--;
                System.out.println("Автомобиль #" + carId + " покинул
место " + (spotIndex + 1));
                printParkingStatus();

                spotAvailable.signal();
            }
        } finally {
            lock.unlock();
        }
    }

    private void printParkingStatus() {
        final String RESET = "\u001B[0m";
        final String GREEN = "\u001B[32m";
        final String RED = "\u001B[31m";
```

```
        StringBuilder status = new StringBuilder("Статус мест: [");

        for (int i = 0; i < parkingSpots.length; i++) {
            if (parkingSpots[i] != 0) {
                status.append(RED).append("Место ").append(i +
1).append(": авто #").append(parkingSpots[i])
                        .append(RESET);
            } else {
                status.append(GREEN).append("Место ").append(i +
1).append(": свободно").append(RESET);
            }

            if (i < parkingSpots.length - 1) {
                status.append(", ");
            }
        }
        status.append("]");

        out.println(status);
        out.println("Статус стоянки: занято " + RED + occupiedSpots +
RESET + " из " + GREEN + PARKING_CAPACITY + RESET
                + " мест (свободно: " + GREEN + (PARKING_CAPACITY -
occupiedSpots) + RESET + ")");
    }

    static class Car extends Thread {
        private final int carId;
        private final ParkingLotCondition parkingLot;
        private final Random random = new Random();
        private int parkedSpotIndex = -1;

        public Car(int carId, ParkingLotCondition parkingLot) {
            this.carId = carId;
            this.parkingLot = parkingLot;
            setName("Car-" + carId);
        }

        @Override
        public void run() {
            try {
                while (!Thread.interrupted()) {
                    System.out.println("Автомобиль #" + carId + " ищет
место для парковки...");
                    parkedSpotIndex = parkingLot.parkCar(carId);

                    if (parkedSpotIndex >= 0) {
```

```
                        int parkingTime =
random.nextInt(MAX_PARKING_TIME) + 1;
                        System.out.println("Автомобиль #" + carId + "
будет стоять " + parkingTime + " сек.");
                        TimeUnit.SECONDS.sleep(parkingTime);

                        parkingLot.leaveParkingLot(carId,
parkedSpotIndex);
                        parkedSpotIndex = -1;

                        int drivingTime =
random.nextInt(MAX_DRIVING_TIME) + 1;
                        System.out.println("Автомобиль #" + carId + "
уехал на " + drivingTime + " сек.");
                        TimeUnit.SECONDS.sleep(drivingTime);
                    }
                }
            } catch (InterruptedException e) {
                System.out.println("Автомобиль #" + carId + " завершил
работу");
                if (parkedSpotIndex != -1) {
                    parkingLot.leaveParkingLot(carId, parkedSpotIndex);
                }
            }
        }
    }

    public static void main(String[] args) {
        ParkingLotCondition parkingLot = new ParkingLotCondition();
        List<Car> cars = new ArrayList<>();

        System.out.println("Запуск симуляции автостоянки с использованием
условных переменных");
        System.out.println("Количество мест на стоянке: " +
PARKING_CAPACITY);
        System.out.println("Количество автомобилей: " + CARS_COUNT);

        for (int i = 1; i <= CARS_COUNT; i++) {
            Car car = new Car(i, parkingLot);
            cars.add(car);
            car.start();
        }

        try {
            TimeUnit.SECONDS.sleep(10);
        } catch (InterruptedException e) {
```

```
            e.printStackTrace();
        }

        System.out.println("Завершение работы автомобилей...");
        for (Car car : cars) {
            car.interrupt();
            try {
                car.join(1000);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }

        System.out.println("Программа завершена");
    }
}
```

Снимок окна работы приложения.

```
Запуск симуляции автостоянки с использованием условных переменных
Количество мест на стоянке: 5
Количество автомобилей: 10
Автомобиль #2 ищет место для парковки...
Автомобиль #1 ищет место для парковки...
Автомобиль #3 ищет место для парковки...
Автомобиль #6 ищет место для парковки...
Автомобиль #4 ищет место для парковки...
Автомобиль #7 ищет место для парковки...
Автомобиль #5 ищет место для парковки...
Автомобиль #8 ищет место для парковки...
Автомобиль #9 ищет место для парковки...
Автомобиль #10 ищет место для парковки...
Автомобиль #2 припарковался на месте 1
Статус мест: [Место 1: авто #2, Место 2: свободно, Место 3: свободно, Место 4: свободно, Место 5: свободно]
Статус стоянки: занято 1 из 5 мест (свободно: 4)
Автомобиль #1 припарковался на месте 2
Статус мест: [Место 1: авто #2, Место 2: авто #1, Место 3: свободно, Место 4: свободно, Место 5: свободно]
Автомобиль #2 будет стоять 3 сек.
Статус стоянки: занято 2 из 5 мест (свободно: 3)
Автомобиль #1 будет стоять 1 сек.
Автомобиль #6 припарковался на месте 3
Статус мест: [Место 1: авто #2, Место 2: авто #1, Место 3: авто #6, Место 4: свободно, Место 5: свободно]
Статус стоянки: занято 3 из 5 мест (свободно: 2)
Автомобиль #6 будет стоять 3 сек.
Автомобиль #3 припарковался на месте 4
Статус мест: [Место 1: авто #2, Место 2: авто #1, Место 3: авто #6, Место 4: авто #3, Место 5: свободно]
Статус стоянки: занято 4 из 5 мест (свободно: 1)
Автомобиль #3 будет стоять 3 сек.
Автомобиль #4 припарковался на месте 5
Статус мест: [Место 1: авто #2, Место 2: авто #1, Место 3: авто #6, Место 4: авто #3, Место 5: авто #4]
Статус стоянки: занято 5 из 5 мест (свободно: 0)
Автомобиль #4 будет стоять 3 сек.
Автомобиль #7 ожидает освобождения места...
Автомобиль #5 ожидает освобождения места...
Автомобиль #8 ожидает освобождения места...
Автомобиль #9 ожидает освобождения места...
Автомобиль #10 ожидает освобождения места...
Автомобиль #1 покинул место 2
Статус мест: [Место 1: авто #2, Место 2: свободно, Место 3: авто #6, Место 4: авто #3, Место 5: авто #4]
Статус стоянки: занято 4 из 5 мест (свободно: 1)
Автомобиль #7 припарковался на месте 2
Автомобиль #1 уехал на 4 сек.
Статус мест: [Место 1: авто #2, Место 2: авто #7, Место 3: авто #6, Место 4: авто #3, Место 5: авто #4]
Статус стоянки: занято 5 из 5 мест (свободно: 0)
Автомобиль #7 будет стоять 2 сек.
Автомобиль #6 покинул место 3
Статус мест: [Место 1: авто #2, Место 2: авто #7, Место 3: свободно, Место 4: авто #3, Место 5: авто #4]
Статус стоянки: занято 4 из 5 мест (свободно: 1)
Автомобиль #6 уехал на 1 сек.
Автомобиль #3 покинул место 4
Статус мест: [Место 1: авто #2, Место 2: авто #7, Место 3: свободно, Место 4: свободно, Место 5: авто #4]
Статус стоянки: занято 3 из 5 мест (свободно: 2)
Автомобиль #3 уехал на 4 сек.
Автомобиль #2 покинул место 1
Статус мест: [Место 1: свободно, Место 2: авто #7, Место 3: свободно, Место 4: свободно, Место 5: авто #4]
Статус стоянки: занято 2 из 5 мест (свободно: 3)
Автомобиль #2 уехал на 2 сек.
Автомобиль #5 припарковался на месте 1
Статус мест: [Место 1: авто #5, Место 2: авто #7, Место 3: свободно, Место 4: свободно, Место 5: авто #4]
Статус стоянки: занято 3 из 5 мест (свободно: 2)
Автомобиль #5 будет стоять 2 сек.
Автомобиль #8 припарковался на месте 3
Статус мест: [Место 1: авто #5, Место 2: авто #7, Место 3: авто #8, Место 4: свободно, Место 5: авто #4]
Статус стоянки: занято 4 из 5 мест (свободно: 1)
Автомобиль #8 будет стоять 1 сек.
Автомобиль #9 припарковался на месте 4
Статус мест: [Место 1: авто #5, Место 2: авто #7, Место 3: авто #8, Место 4: авто #9, Место 5: авто #4]
Статус стоянки: занято 5 из 5 мест (свободно: 0)
Автомобиль #9 будет стоять 3 сек.
```

Задание 5.

Задания к главе 12, Вариант А, Задача 3 [2, стр. 424]
Разработать многопоточное приложение. Использовать возможности, предоставляемые пакетом java.util.concurrent. Не использовать слово synchronized. Все сущности, желающие получить доступ к ресурсу, должны быть потоками.
Автостоянка. Доступно несколько машиномест. На одном месте может находиться только один автомобиль. Если все места заняты, то автомобиль не станет ждать больше определенного времени и уедет на другую стоянку.
Подсказка. Читать «Семафор» [2, стр. 403-407]

Текст программы.

```
package lab7;

import java.util.ArrayList;
import java.util.List;
import java.util.Random;
import java.util.concurrent.Semaphore;
import java.util.concurrent.TimeUnit;

import static java.lang.System.out;

public class ParkingLotSemaphore2 {
    private static final int PARKING_CAPACITY = 5;
    private static final int CARS_COUNT = 10;
    private static final int MAX_PARKING_TIME = 3;
    private static final int MAX_DRIVING_TIME = 5;
    private static final int MAX_WAITING_TIME = 2000;

    private final int[] parkingSpots = new int[PARKING_CAPACITY];
    private int occupiedSpots = 0;

    private final Semaphore semaphore = new Semaphore(PARKING_CAPACITY, true);

    public int parkCar(int carId, long maxWaitMillis) throws ParkingException {
        try {
            if (semaphore.tryAcquire(maxWaitMillis, TimeUnit.MILLISECONDS)) {
                synchronized (parkingSpots) {
                    for (int i = 0; i < parkingSpots.length; i++) {
                        if (parkingSpots[i] == 0) {
                            parkingSpots[i] = carId;
                            occupiedSpots++;
                            System.out.println("Автомобиль #" + carId + "
припарковался на месте " + (i + 1));
                            printParkingStatus();
```

```
                        return i;
                    }
                }

                System.out.println("ОШИБКА: Автомобиль #" + carId
                        + " не нашел свободного места, хотя семафор
разрешил парковку");
                semaphore.release();
                return -1;
            }
        } else {
            throw new ParkingException("время ожидания превышено");
        }
    } catch (InterruptedException e) {
        Thread.currentThread().interrupt();
        throw new ParkingException("поток был прерван");
    }
}

public void leaveParkingLot(int carId, int spotIndex) {
    synchronized (parkingSpots) {
        if (spotIndex >= 0 && spotIndex < parkingSpots.length &&
parkingSpots[spotIndex] == carId) {
            parkingSpots[spotIndex] = 0;
            occupiedSpots--;
            System.out.println("Автомобиль #" + carId + " покинул место "
+ (spotIndex + 1));
            printParkingStatus();

            semaphore.release();
        }
    }
}

private void printParkingStatus() {
    final String RESET = "\u001B[0m";
    final String GREEN = "\u001B[32m";
    final String RED = "\u001B[31m";

    StringBuilder status = new StringBuilder("Статус мест: [");

    for (int i = 0; i < parkingSpots.length; i++) {
        if (parkingSpots[i] != 0) {
            status.append(RED).append("Место ").append(i + 1).append(":
авто #").append(parkingSpots[i])
                    .append(RESET);
        } else {
            status.append(GREEN).append("Место ").append(i + 1).append(":
свободно").append(RESET);
        }
```

```
            if (i < parkingSpots.length - 1) {
                status.append(", ");
            }
        }
        status.append("]");

        out.println(status);
        out.println("Статус стоянки: занято " + RED + occupiedSpots + RESET + "
из " + GREEN + PARKING_CAPACITY + RESET
                + " мест (свободно: " + GREEN + (PARKING_CAPACITY -
occupiedSpots) + RESET + ")");
    }

    public static class ParkingException extends Exception {
        public ParkingException(String message) {
            super(message);
        }
    }

    static class Car extends Thread {
        private final int carId;
        private final ParkingLotSemaphore2 parkingLot;
        private final Random random = new Random();
        private int parkedSpotIndex = -1;

        public Car(int carId, ParkingLotSemaphore2 parkingLot) {
            this.carId = carId;
            this.parkingLot = parkingLot;
            setName("Car-" + carId);
        }

        @Override
        public void run() {
            try {
                while (!Thread.interrupted()) {
                    System.out.println("Автомобиль #" + carId + " ищет место
для парковки...");

                    try {
                        long waitTime = random.nextInt(1500) + 500;
                        parkedSpotIndex = parkingLot.parkCar(carId, waitTime);

                        if (parkedSpotIndex >= 0) {
                            int parkingTime = random.nextInt(MAX_PARKING_TIME) +
1;
                            System.out.println("Автомобиль #" + carId + "
будет стоять " + parkingTime + " сек.");
                            TimeUnit.SECONDS.sleep(parkingTime);

                            parkingLot.leaveParkingLot(carId, parkedSpotIndex);
                            parkedSpotIndex = -1;
```

```
                            int drivingTime = random.nextInt(MAX_DRIVING_TIME) +
1;
                            System.out.println("Автомобиль #" + carId + "
уехал на " + drivingTime + " сек.");
                            TimeUnit.SECONDS.sleep(drivingTime);
                        }
                    } catch (ParkingException e) {
                        System.err.println("Автомобиль #" + carId + " не смог
припарковаться: " + e.getMessage());
                        System.out.println("Автомобиль #" + carId + " уехал на
другую стоянку");

                        // Уезжаем на другую стоянку на некоторое время
                        TimeUnit.SECONDS.sleep(random.nextInt(MAX_DRIVING_TIME)
+ 1);
                    }
                }
            } catch (InterruptedException e) {
                System.out.println("Автомобиль #" + carId + " завершил
работу");
                if (parkedSpotIndex != -1) {
                    parkingLot.leaveParkingLot(carId, parkedSpotIndex);
                }
            }
        }
    }

    public static void main(String[] args) {
        ParkingLotSemaphore2 parkingLot = new ParkingLotSemaphore2();
        List<Car> cars = new ArrayList<>();

        System.out.println("Запуск симуляции автостоянки 2 с использованием
семафора");
        System.out.println("Количество мест на стоянке: " + PARKING_CAPACITY);
        System.out.println("Количество автомобилей: " + CARS_COUNT);
        System.out.println("Максимальное время ожидания: " + MAX_WAITING_TIME
+ " мс");

        for (int i = 1; i <= CARS_COUNT; i++) {
            Car car = new Car(i, parkingLot);
            cars.add(car);
            car.start();
        }

        try {
            TimeUnit.SECONDS.sleep(10);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
```

```
        System.out.println("Завершение работы автомобилей...");
        for (Car car : cars) {
            car.interrupt();
            try {
                car.join(1000);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }

        System.out.println("Программа завершена");
    }
}
```

Снимок окна работы приложения

```
Запуск симуляции автостоянки 2 с использованием семафора
Количество мест на стоянке: 5
Количество автомобилей: 10
Максимальное время ожидания: 2000 мс
Автомобиль #1 ищет место для парковки...
Автомобиль #2 ищет место для парковки...
Автомобиль #3 ищет место для парковки...
Автомобиль #7 ищет место для парковки...
Автомобиль #6 ищет место для парковки...
Автомобиль #9 ищет место для парковки...
Автомобиль #10 ищет место для парковки...
Автомобиль #8 ищет место для парковки...
Автомобиль #4 ищет место для парковки...
Автомобиль #5 ищет место для парковки...
Автомобиль #1 припарковался на месте 1
Статус мест: [Место 1: авто #1, Место 2: свободно, Место 3: свободно, Место 4: свободно, Место 5: свободно]
Статус стоянки: занято 1 из 5 мест (свободно: 4)
Автомобиль #6 припарковался на месте 2
Статус мест: [Место 1: авто #1, Место 2: авто #6, Место 3: свободно, Место 4: свободно, Место 5: свободно]
Автомобиль #1 будет стоять 3 сек.
Статус стоянки: занято 2 из 5 мест (свободно: 3)
Автомобиль #6 будет стоять 1 сек.
Автомобиль #7 припарковался на месте 3
Статус мест: [Место 1: авто #1, Место 2: авто #6, Место 3: авто #7, Место 4: свободно, Место 5: свободно]
Статус стоянки: занято 3 из 5 мест (свободно: 2)
Автомобиль #7 будет стоять 2 сек.
Автомобиль #3 припарковался на месте 4
Статус мест: [Место 1: авто #1, Место 2: авто #6, Место 3: авто #7, Место 4: авто #3, Место 5: свободно]
Статус стоянки: занято 4 из 5 мест (свободно: 1)
Автомобиль #3 будет стоять 3 сек.
Автомобиль #2 припарковался на месте 5
Статус мест: [Место 1: авто #1, Место 2: авто #6, Место 3: авто #7, Место 4: авто #3, Место 5: авто #2]
Статус стоянки: занято 5 из 5 мест (свободно: 0)
Автомобиль #2 будет стоять 2 сек.
Автомобиль #10 не смог припарковаться: время ожидания превышено
Автомобиль #10 уехал на другую стоянку
Автомобиль #5 не смог припарковаться: время ожидания превышено
Автомобиль #5 уехал на другую стоянку
Автомобиль #8 не смог припарковаться: время ожидания превышено
Автомобиль #8 уехал на другую стоянку
Автомобиль #6 покинул место 2
Статус мест: [Место 1: авто #1, Место 2: свободно, Место 3: авто #7, Место 4: авто #3, Место 5: авто #2]
Статус стоянки: занято 4 из 5 мест (свободно: 1)
Автомобиль #9 припарковался на месте 2
Автомобиль #6 уехал на 2 сек.
Статус мест: [Место 1: авто #1, Место 2: авто #9, Место 3: авто #7, Место 4: авто #3, Место 5: авто #2]
Статус стоянки: занято 5 из 5 мест (свободно: 0)
Автомобиль #9 будет стоять 3 сек.
Автомобиль #4 не смог припарковаться: время ожидания превышено
Автомобиль #4 уехал на другую стоянку
Автомобиль #7 покинул место 3
Статус мест: [Место 1: авто #1, Место 2: авто #9, Место 3: свободно, Место 4: авто #3, Место 5: авто #2]
Статус стоянки: занято 4 из 5 мест (свободно: 1)
Автомобиль #7 уехал на 4 сек.
Автомобиль #2 покинул место 5
Статус мест: [Место 1: авто #1, Место 2: авто #9, Место 3: свободно, Место 4: авто #3, Место 5: свободно]
Статус стоянки: занято 3 из 5 мест (свободно: 2)
Автомобиль #2 уехал на 1 сек.
Автомобиль #1 покинул место 1
Статус мест: [Место 1: свободно, Место 2: авто #9, Место 3: свободно, Место 4: авто #3, Место 5: свободно]
Статус стоянки: занято 2 из 5 мест (свободно: 3)
Автомобиль #1 уехал на 4 сек.
Автомобиль #6 ищет место для парковки...
Автомобиль #3 покинул место 4
Статус мест: [Место 1: свободно, Место 2: авто #9, Место 3: свободно, Место 4: свободно, Место 5: свободно]
Статус стоянки: занято 1 из 5 мест (свободно: 4)
Автомобиль #3 уехал на 1 сек.
Автомобиль #6 припарковался на месте 1
Статус мест: [Место 1: авто #6, Место 2: авто #9, Место 3: свободно, Место 4: свободно, Место 5: свободно]
Статус стоянки: занято 2 из 5 мест (свободно: 3)
```