

ГЛАВА 2

2.1. Группировка потоков (потоков выполнения). Приоритет потоков выполнения.

2.1.1. Группировка потоков

Группировка потоков предоставляет механизм для манипулирования ими как единым целым. Например, мы можем запустить или приостановить все потоки в группе одним вызовом метода. Группировка потоков осуществляется с помощью класса `ThreadGroup`.

Каждый поток выполнения Java является членом группы, независимо от того, указываем ли мы это явно. Присоединение потока выполнения к определенной группе осуществляется при ее создании и становится постоянным, поэтому мы не сможем переместить поток выполнения из одной группы в другую после его создания. Если мы создаем поток выполнения, не указывая в конструкторе, к какой группе он принадлежит, он будет автоматически помещен в ту же группу, что и поток выполнения, который его создал. При запуске Java-программы автоматически создается объект типа **`ThreadGroup`** с именем **`main`**, который будет представлять группу всех потоков выполнения, созданных непосредственно из программы и не привязанных явно к другой группе. Можно полностью игнорировать размещение потоков выполнения в группах и позволить системе заняться этим, поместив их все в группу **`main`**. Существуют ситуации, когда программа создает множество потоков выполнения, и их группировка может существенно облегчить их управление. Создание потока выполнения и его размещение в группе (отличной от группы по умолчанию) осуществляется с помощью следующих конструкторов класса `Thread`:

```
public Thread(ThreadGroup group, Runnable target)
public Thread(ThreadGroup group, String name)
public Thread(ThreadGroup group, Runnable target,
               String name)
```

Каждый из этих конструкторов создает поток выполнения, инициализирует его и помещает в группу, указанную в аргументе. В следующем примере будут созданы две группы, первая с двумя потоками выполнения, а вторая с тремя потоками выполнения:

```
ThreadGroup группа1 = new ThreadGroup("Производители");
Thread p1 = new Thread(группа1, "Производитель 1");
Thread p2 = new Thread(группа1, "Производитель 2");
ThreadGroup группа2 = new ThreadGroup("Потребители");
Thread c1 = new Thread(группа2, "Потребитель 1");
Thread c2 = new Thread(группа2, "Потребитель 2");
Thread c3 = new Thread(группа2, "Потребитель 3");
```

Чтобы узнать, к какой группе принадлежит определенный поток выполнения, мы можем использовать метод `getThreadGroup` класса `Thread`. Группа может иметь в качестве родителя другую группу, что означает, что потоки выполнения могут быть размещены в иерархии групп, в которой корнем является группа `main` по умолчанию, как показано на рисунке 3.

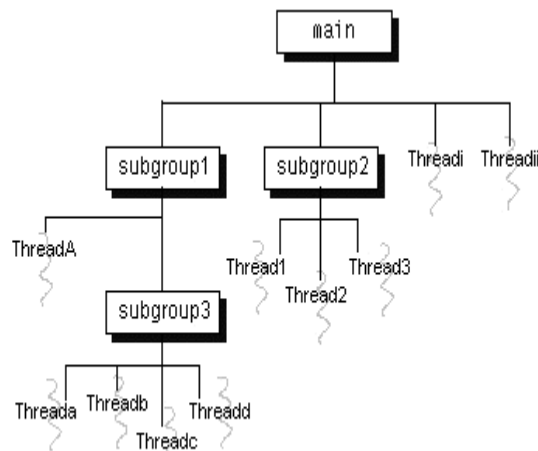


Рисунок 3. Группировка потоков выполнения

Пример: Перечисление активных потоков выполнения

```

public class EnumerateTest {
    public void listCurrentThreads() {
        ThreadGroup currentGroup =
            Thread.currentThread().getThreadGroup();
        //узнаю количество активных потоков
        int numThreads = currentGroup.activeCount();
        //помещаю в вектор ссылки на потоки выполнения.
        //активные
        Thread[] listOfThreads = new Thread[numThreads];
        currentGroup.enumerate(listOfThreads);
        //отображаю их на экране
        for (int i = 0; i < numThreads; i++)
            System.out.println("Thread #" + i + " = " +
                listOfThreads[i].getName());
    }
}
  
```

Структура потоков выполнения и групп выполнения может быть представлена в следующем виде:

```

MainGrupa{subGrupa1{subGrupa3{Tha, Thb, Thc, Thd},
ThA},subGrupa2{Th1, Th2, Th3},Th1, Th2}.
  
```

2.1.2.Приоритет потоков выполнения.

Приоритет потоков выполнения — это целое число, которое определяет относительный приоритет одного потока по отношению к другому.

Каждый поток выполнения имеет приоритет в диапазоне от `MIN_PRIORITY` до `MAX_PRIORITY`. Эти две конечные переменные объявлены в классе **Thread**. По умолчанию поток выполнения имеет приоритет `NORM_PRIORITY`, также определенный в классе **Thread**.

Среда выполнения Java упорядочивает потоки выполнения для получения контроля над центральным процессором в зависимости от их приоритета. Если существует несколько потоков с максимальным приоритетом, они упорядочиваются по алгоритму, называемому «round-robin» (по кругу). Потоки с более низким приоритетом обрабатываются только тогда, когда все потоки с высоким приоритетом находятся в состоянии «**Не выполняется**».

Приоритет потока выполнения можно определить с помощью метода `getPriority()`. Метод `getPriority()` возвращает приоритет потока выполнения, который является целым числом и представляет собой текущий приоритет потока выполнения. Для установки приоритета используется метод `setPriority()`, который принимает в качестве параметра целое число, представляющее желаемый приоритет.

Для изменения приоритета группы потоков выполнения используется метод `setMaxPriority()`.

Изменение приоритета потока выполнения опасно, если метод с высоким приоритетом не заканчивается очень быстро или не имеет частых остановок. В противном случае другие методы не смогут получить контроль над центральным процессором.

Однако есть ситуации, в которых мы можем изменить этот приоритет без опасности, например, когда у нас есть поток выполнения, который не делает ничего, кроме чтения символов от пользователя и запоминания их во временной области. В этом случае поток выполнения большую часть времени находится в состоянии «Не выполняется», потому что ожидает завершения операции ввода/вывода. В момент, когда пользователь набирает символ, поток выходит из состояния ожидания и будет первым запланированным к выполнению из-за своего высокого приоритета. Таким образом, пользователь имеет ощущение, что приложение очень быстро реагирует на его команды.

В других ситуациях нам нужно выполнить задачу с низким приоритетом. В этих случаях мы можем установить для потока выполнения, который выполняет эти задачи, низкий приоритет.

Пример создания групп с потоками выполнения и изменения приоритетов.

```
public class ThreadGroup {
public static void main(String[] args) {
    String name = null;
    ThreadGroup sys =
        Thread.currentThread().getThreadGroup();

    Thread curr = Thread.currentThread();
    curr.setPriority(curr.getPriority() + 1);
    sys.list();
    ThreadGroup g1 = new ThreadGroup(" g1");

    Thread t=null;

    t= new Thread(g1,new Fir(" A"));
    t.setPriority(Thread.MAX_PRIORITY);

    Thread t1=null;

    t1= new Thread(g1,new Fir(" B"));
    t.setPriority(Thread.MAX_PRIORITY-6);

    Thread t2=null;

    t2= new Thread(g1,new Fir(" C"));
    t.setPriority(Thread.MAX_PRIORITY-6);
    g1.list();

    ThreadGroup g2 = new ThreadGroup(g1, " g2");

    for (int i = 0; i < 5; i++){
```

```

        name=String.valueOf( Integer.toString(i));
System.out.println(name);
        Thread thread = new Thread(g2,new Fir(name));
    }
g2.list();
    System.out.println("Запуск всех потоков:");
}
}
class Fir extends Thread {
    String name;
    public Fir(String name) {
        this.name=name;
        start();
    }
    public void run() {
        System.out.println("Привет от потока "+ name);
        try {
            Thread.sleep(100);
        } catch (InterruptedException e) {
            // TODO Автоматически сгенерированный блок catch
            e.printStackTrace();
        }
    }
}
}

```

Результат выполнения:

```

java.lang.ThreadGroup[name=main,maxpri=10]
  Thread[main,6,main]
java.lang.ThreadGroup[name= g1,maxpri=10]
Привет от потока    В
Привет от потока    А
Привет от потока    С
Запуск всех потоков:
java.lang.ThreadGroup[name= g2,maxpri=10]
Привет от потока    0
Привет от потока    2
Привет от потока    1
Привет от потока    4
Привет от потока    3

```

Лабораторная работа № 2

Тема работы: Группировка нитей выполнения. Изменение приоритета.

Цель работы:

Изучение механизмов группировки потоков выполнения (thread groups) и изменения их приоритета в рамках программы, чтобы понять, как операционная система и язык программирования управляют конкуренцией и ресурсами процессора.

Цели работы:

- **Ознакомление с основными понятиями:**

- группами потоков (thread groups);
 - приоритетами групп и потоков и их влиянием на планирование.
- **Создание и управление группами потоков**, демонстрируя:
- как организовать несколько нитей в общую группу;
 - как применяются методы контроля над группой нитей.
- **Изучение механизма приоритезации** путем:
- изменения приоритета отдельных потоков;
 - анализа поведения программы, когда разные потоки имеют разные приоритеты.
- **Практическое применение** концепций путем написания и запуска программ, которые:
- создают и запускают несколько потоков параллельно;
 - используют группы потоков для организации;
 - изменяют и сравнивают приоритеты потоков.
- **Анализ результатов** для понимания:
- как приоритеты влияют на порядок выполнения;
 - различия между выполнением высокоприоритетных и низкоприоритетных потоков;
 - реальные ограничения приоритета в зависимости от планировщика операционной системы.

Пример реализации:

```
public class ThreadGroup1 {
    public static void main(String[] args) {
        ThreadGroup sys =
            Thread.currentThread().getThreadGroup();
        sys.list();
        sys.setMaxPriority(Thread.MAX_PRIORITY - 1);
        Thread curr = Thread.currentThread();
        curr.setPriority(curr.getPriority() + 1);
        sys.list();
        ThreadGroup g1 = new ThreadGroup("g1");
        g1.setMaxPriority(Thread.MAX_PRIORITY);
        Thread t = new Thread(g1, "A")
            t.start();
            t.setPriority(Thread.MAX_PRIORITY);
        g1.list();
        g1.setMaxPriority(Thread.MAX_PRIORITY - 2);
        g1.setMaxPriority(Thread.MAX_PRIORITY);
        g1.list();
        t = new Thread(g1, "B");
        t.start();
        t.setPriority(Thread.MAX_PRIORITY);
        g1.list();
        g1.setMaxPriority(Thread.MIN_PRIORITY + 2);
        t = new Thread(g1, "C");
        t.start();
        t.setPriority(t.getPriority() - 1);
        g1.list();
        ThreadGroup g2 = new ThreadGroup(g1, "g2");
        g2.list(); // (8)
        g2.setMaxPriority(Thread.MAX_PRIORITY);
        g2.list(); // (9)
    }
}
```

```

for (int i = 0; i < 5; i++)
new Thread(g2, Integer.toString(i)).start();
    sys.list(); // (10)
    System.out.println("Запуск всех потоков:");

}} ///:~

```

Результат выполнения программы:

```

(1) ThreadGroup[name=system,maxpri=10]
    Thread[main,5,system]
(2) ThreadGroup[name=system,maxpri=9]
    Thread[main,6,system]
(3) ThreadGroup[name=g1,maxpri=9]
    Thread[A,9,g1]
(4) ThreadGroup[name=g1,maxpri=8]
    Thread[A,9,g1]
(5) ThreadGroup[name=g1,maxpri=8]
    Thread[A,9,g1]
    Thread[B,8,g1]
(6) ThreadGroup[name=g1,maxpri=3]
    Поток[A,9,g1]
    Thread[B,8,g1]
    Поток[C,6,g1]
(7) ThreadGroup[name=g1,maxpri=3]
    Thread[A,9,g1]
    Поток[B,8,g1]
    Поток[C,3,g1]
(8) ThreadGroup[name=g2,maxpri=3]
(9) ThreadGroup[name=g2,maxpri=3]
(10) ThreadGroup[name=system,maxpri=9]
    Поток[main,6,system]
    ThreadGroup[name=g1,maxpri=3]
        Thread[A,9,g1]
        Thread[B,8,g1]
        Thread[C,3,g1]
    ThreadGroup[name=g2,maxpri=3]
        Thread[0,6,g2]
        Поток[1,6,g2]
        Поток[2,6,g2]
        Поток[3,6,g2]
        Поток[4,6,g2]

```

Запуск всех потоков:

Все потоки запущены

Этапы выполнения работы:

1. Студенты группы делятся на команды по два человека. Практическая работа будет выполняться в команде.
2. Реализуйте приложение (из **задания работы**). В команде решите, кто и какой уровень приложения будет реализовывать.
3. После получения инструкций от преподавателя внесите изменения в свою отдельную ветку на GitHub. Зафиксируйте изменения в коде приложения (команда git commit).
4. По завершении работы составьте отчет, который должен содержать: имя, фамилию, группу, задание и ваш вариант реализации задания, краткое описание, ссылку на исходный код на GitHub. Сохраните отчет в формате PDF или WORD и отправьте на ELSE.

Задания, предлагаемые для выполнения:

1. Задача среднего уровня сложности:

Создайте структуру в соответствии с формулой, приведенной в таблице 1, в соответствии с вариантом. Перечислите все потоки выполнения в основной группе и подгруппах, которые она содержит. Отобразите информацию об имени потока выполнения, имени группы, к которой он принадлежит, и его приоритете. Приоритет каждого потока указан в скобках () для каждого потока выполнения (см. таблицу 1). Не перезаписывайте метод run() из класса Thread.

2. Задача для продвинутого уровня:

Создайте структуру в соответствии с формулой, приведенной в таблице 1, в соответствии с вариантом. Перечислите все потоки выполнения в основной группе и подгруппах, которые она содержит. Приоритет каждого потока указан в скобках () для каждого потока выполнения (см. таблицу 1). Переопределите метод run() из класса Thread, в котором поток выполнения выводит в консоль свое имя и группу, к которой он принадлежит. Выведите приоритет потоков выполнения.

Таблица 1. Структура потоков выполнения в зависимости от варианта

№ Вариант а	Формула структуры, состоящей из потоков и групп потоков
1	Main{G1{G3{Tha(3), Thb(3), Thc(3), Thd(3)}},G2{Th1(4), Th2(5), Th3(5)},Th1(7), Th2(7), ThA(3)}
2	Main{G2G4{{G1{Tha(1), Thb(3), Thc(8), Thd(3)}}, ThA(1)},G3{Th1(4), Th2(3), Th3(5)},Th1(3), Th2(6)}
3	Main{GN{GH{Tha(4), Thb(3), Thc(6), Thd(3)}, ThA(3)},GM{Th1(2), Th2(3), Th3(3)},Th1(8), Th2(3)}
4	Main{GO{GZ{Tha(1), Thb(3), Thc(3), Thd(7)}},GV{ ThA(3)},GF{Th1(5), Th2(3), Th3(9)},Th1(3), Th2(0)}
5	Main{Th1(3),GE{GH{Tha(4),Thb(3),Thc(2),Thd(1)}, ThA(3)},GK{Th1(3), Th2(6), Th3(3)}, Th2(7)}
6	Main{G1{ThA(3)}{G3{Thf(3), Thb(7), Thc(3), Thd(3)} G2{Th8(3), Th9(4), Th3(3)},Th1(3), Th2(3)}
7	Main{ThA(3)},G2{Th1(5), Th2(3), Th33(7)},Th11(3), Th22(3), G1{G3{Thaa(2), Thbb(3), Thcc(8), Thdd(3)} }
8	Main{G4{G3{Tha(2), Thb(8), Thc(3), Thd(3), G2{Th1(3), Th2(3), Th3(3), ThA(3)},Th1(8), Th2(3)} }
9	Main{G6{ ThA(3)},Th1(4), Th2(3), G2{Th1(2), G3{Tha(2), Thb(3), Thc(4), Thd(3)}, Th2(3), Th3(3)} }
10	Main{G7{G3{Tha(6), Thb(3), Thc(6), Thd(3)}, ThA(7), ThB(6)}, Th2(3), G2{Th1(7), Th2(3), Th3(3)} }

Критерии оценки:

1. Создание схемы в соответствии с соответствующим вариантом.
2. Создание и инициализация групп и потоков из задачи среднего уровня.
3. Создание и инициализация групп и потоков из задачи для продвинутого уровня .
4. Создание графического интерфейса программы.
5. Корректность кода — проверка правильности кода , отсутствие ошибок в работе .
6. Соблюдение инструкций и требований — проверка правильности требований задачи, таких как количество нитей выполнения.
7. Оптимизация кода — оценка эффективности кода в использовании ресурсов и избегание кода.
8. Соблюдение сроков сдачи - оценка баллов за пунктуальность, если работа была сдана в установленный срок.

9. Оценка знаний - объяснения, данные о процессе выполнения работы, что может включать описание основных функций и использованной логики.

10. Использование ИИ студентом.

Для получения оценки 5-6 обязательны критерии 1,2,5,6,8,9

Для получения оценки 7–8 обязательны критерии 1, 3, 5, 6, 8, 9.

Для получения оценки 9–10 обязательны критерии 1–9.

Если был использован критерий 10, оценка снижается на 2 балла, только если студент разъяснил работу кода. В противном случае лабораторная работа не принимается.

Контрольные вопросы:

- Что такое **группа потоков (ThreadGroup)** и какова ее роль в управлении конкуренцией?
- Как создается и связывается поток выполнения с определенной группой?
- Какие методы работы с группами потоков доступны в языке Java (или в языке программирования, используемом в лаборатории)?
- Что представляет собой **приоритет потока выполнения** и как он влияет на планирование?
- Каков диапазон значений приоритета для потоков выполнения и каково значение по умолчанию?
- Каковы стандартные константы приоритета, определенные в Java (`MIN_PRIORITY`, `NORM_PRIORITY`, `MAX_PRIORITY`)?
- Как можно изменить приоритет потока в программе?
- Почему изменение приоритетов не всегда гарантирует точный порядок выполнения потоков?
- Как операционная система влияет на поведение планирования потоков, независимо от приоритетов, установленных в коде?
- Как можно визуализировать или проверить поведение потоков с разными приоритетами в экспериментальной программе?

Список литературы рекомендуется:

1. Petre Dini – *Продвинутое программирование на Java*, издательство Polirom, Яссы.
2. Университетские курсы «Операционные системы» и «Конкурентное программирование» в технических университетах Румынии и Республики Молдова (онлайн-материалы).
3. Блох Дж. – *Java. Эффективное программирование*, 3-е изд., Вильямс.
4. Хорстманн К. – *Java. Библиотека профессионала. Том 1–2*, Вильямс.
5. Герберт Шильдт – *Java: Полное руководство*, 12-е издание, McGraw-Hill.
6. Брайан Гетц и др. – *Java Concurrency in Practice*, Addison-Wesley.