ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»

Московский институт электроники и математики им. А.Н. Тихонова

Департамент компьютерной инженерии

Отчет по практической работе №1

«Потоки выполнения. Синхронизация»

по курсу «Распределенные базы данных и сетевые вычисления»

Выполнили:
Студенты группы МКС212
Журсунова Найля
Шабалина Анастасия Владимировна
Приняла:
Байбикова Татьяна Николаевна

Москва 2022 г.

Оглавление

1. Задание

Изучить теоретический материал для практики №1. Разобрать прилагаемые примеры. Создать свои примеры, в которых содержатся:

- класс ***Thread***;
- интерфейс ***Runnable***.

Подготовить отчет.

В отчет включить:

- задание;
- разработанную программу или набор программ;
- результаты работы программы;
- краткие выводы.

Работа выполнялась на языке программирования Java.

2. Потоки выполнения

В Java предоставлены две возможности создания потоков: реализация (implementing) интерфейса ***Runnable*** и расширение (extending) класса ***Thread***.

Класс ***Thread*** является основным классом при работе с потоками выполнения. Создание любого потока начинается с создания экземпляра класса ***Thread***.

Другой способ определения потока представляет реализация интерфейса ***Runnable***. Этот интерфейс имеет один метод *run()*. Методом *run()* определяется точка входа в поток. Как только завершится выполнение метода *run()*, завершится и выполнения потока. В методе *run()* можно выполнять любые операции, присущие обычным методам (объявлять переменные, использовать классы, вызывать другие методы и т.д.).

2.1. Главный поток

После запуска на выполнение программы, начинает выполняться один поток — ***главный поток*** программы. Для главного потока можно выделить следующие характерные особенности:

- главный поток начинает выполняться первым;
- из главного потока можно породить (создать) все дочерние потоки;
- желательно, чтобы главный поток завершался последним и выполнял некоторые завершающие действия при закрытии программы.

Чтобы получить доступ к главному потоку управления, нужно создать экземпляр класса ***Thread*** с помощью вызова статического метода *currentThread()*. После этого, с помощью экземпляра, можно использовать дополнительные функции управления главным потоком.

2.1.1. Реализация

В ***MainThread.java*** продемонстрирован доступ к главному потоку в программе и его использование. Все пояснения в комментариях к коду.

```
package rbd.thread;

public class MainThread {
  //Получить данные о главном потоке выполнения
  public static void main(String[] args) {
    Thread t = Thread.currentThread(); //Получаем объект главного потока

    System.out.println("Текущий поток: " + t); //Выводим данные о текущем потоке
выполнения

    //Устанавливаем новое имя потока выполнения с помощью ф-ции setName()
```

```
    t.setName("Мой поток");

    System.out.println("Текущий поток: " + t); //Снова выводим данные о потоке

    try { //Демонстрация работы потока выполнения
      for (int n=5; n>0; n--) {
        System.out.println(n);
        Thread.sleep(1000); //приостанавливаем выполнение вызывающего потока на 1
сек
      }
    }
    catch (InterruptedException e) {
      System.out.println("The main thread is interrupted.");
    }
  }
}
```

2.1.2. Результат выполнения работы

```
Текущий поток: Thread[main,5,main]
Текущий поток: Thread[Мой поток,5,main]
5
4
3
2
1
```

2.2. Класс Thread

Один из способов создания потока — наследование (расширение) класса ***Thread***. При этом способе доступны некоторые методы суперкласса ***Thread***.

2.2.1. Реализация

Об использовании методов класса ***Thread*** описывается в ***CarThreads.java***. Все пояснения в комментариях к коду.

```
package rbd.thread;

// Создание потоков с помощью расширения класса Thread
class PetrolStation extends Thread {
  private int count; //переменная для хранения кол-ва автомобилей

  public PetrolStation (String name, int carCount){   // Конструктор
        super(name);
        count = carCount;
  }

  @Override // Реализация собственного метода run()
  public void run() {
    int n=0; //переменная для подсчета уже заправленных автомобилей
```

```
    System.out.println(Thread.currentThread().getName() + " открыта...");
    try {
      for (int i=count; i>0; i--) {
        n++;
        System.out.println("Заправлено автомобилей на " +
Thread.currentThread().getName() + ": " + n);
        Thread.sleep(1000); //приостанавливаем выполнение потока на 1 сек
      }
    }
    catch (InterruptedException e) {
      System.out.println("Поток на " + Thread.currentThread().getName() + "
прерван.");
    }
    System.out.println("Поток на " + Thread.currentThread().getName() + "
завершен.");
  }
}

public class CarThreads {
  public static void main(String[] args) {
    // Демонстрация создания потоков. На вход подается имя бензоколонки и
    //кол-во автомобилей, которые должны заправиться на этой бензоколонке
    PetrolStation t1 = new PetrolStation("Бензоколонка №1", 7);
    PetrolStation t2 = new PetrolStation("Бензоколонка №2", 11);
    t1.start(); //создаем новый поток
    t2.start(); //создаем новый поток
  }
}
```

2.2.2. Результат выполнения работы

```
Бензоколонка №2 открыта ...
Бензоколонка №1 открыта ...
Заправлено автомобилей на Бензоколонка №1: 1
Заправлено автомобилей на Бензоколонка №2: 1
Заправлено автомобилей на Бензоколонка №2: 2
Заправлено автомобилей на Бензоколонка №1: 2
Заправлено автомобилей на Бензоколонка №2: 3
Заправлено автомобилей на Бензоколонка №1: 3
Заправлено автомобилей на Бензоколонка №2: 4
Заправлено автомобилей на Бензоколонка №1: 4
Заправлено автомобилей на Бензоколонка №2: 5
Заправлено автомобилей на Бензоколонка №1: 5
Заправлено автомобилей на Бензоколонка №2: 6
Заправлено автомобилей на Бензоколонка №1: 6
Заправлено автомобилей на Бензоколонка №1: 7
Заправлено автомобилей на Бензоколонка №2: 7
Заправлено автомобилей на Бензоколонка №2: 8
Поток на Бензоколонка №1 завершен.
Заправлено автомобилей на Бензоколонка №2: 9
Заправлено автомобилей на Бензоколонка №2: 10
Заправлено автомобилей на Бензоколонка №2: 11
Поток на Бензоколонка №2 завершен.
```

2.3. Интерфейс Runnable

Другим способом создания дочернего потока является реализация интерфейса ***Runnable***. Если класс использует интерфейс ***Runnable***, то в этом классе нужно реализовывать метод *run()*.

2.3.1. Реализация

В ***CarRunnable.java*** реализован интерфейс ***Runnable.*** Все пояснения в комментариях к коду.

```
package rbd.thread;

// Пример создания потока с помощью реализации интерфейса Runnable
// В интерфейсе Runnable определен метод run(), который нужно реализовать
class MyRunnable implements Runnable {
  private int count; //переменная для хранения кол-ва авто

  MyRunnable(int carCount) { // Конструктор
    count = carCount;
    System.out.println("Бензоколонка открыта...");
  }

  @Override // Реализация метода run() из интерфейса Runnable
  public void run() {
    int n=0; //переменная для подсчета уже заправленных автомобилей
    try {
      for (int i=count; i>0; i--) {
        n++;
        System.out.println("Заправлено автомобилей на бензоколонке: " + n);
        Thread.sleep(500); //приостанавливаем выполнение вызывающего потока на
0.5 сек
      }
    }
    catch (InterruptedException e) {
      System.out.println("Поток на бензоколонке прерван.");
    }
    System.out.println("Бензоколонка закрыта.");
  }
}

public class CarRunnable {
  public static void main(String[] args) {
    System.out.println("Заправка открыта...");
    // Демонстрация работы дочернего потока
    Thread t = new Thread(new MyRunnable(12), "Бензоколонка");
    t.start(); // создаем новый поток
    try {
      t.join(); // ожидаем завершения дочернего потока, чтобы главный поток
завершился последним
    }
```

```
    catch (InterruptedException e) {
      System.out.println("Поток на бензоколонке прерван.");
    }
    System.out.println("Заправка закрыта.");
  }
}
```

2.3.2. Результат выполнения работы

```
Заправка открыта ...
Бензоколонка открыта ...
Заправлено автомобилей на бензоколонке: 1
Заправлено автомобилей на бензоколонке: 2
Заправлено автомобилей на бензоколонке: 3
Заправлено автомобилей на бензоколонке: 4
Заправлено автомобилей на бензоколонке: 5
Заправлено автомобилей на бензоколонке: 6
Заправлено автомобилей на бензоколонке: 7
Заправлено автомобилей на бензоколонке: 8
Заправлено автомобилей на бензоколонке: 9
Заправлено автомобилей на бензоколонке: 10
Заправлено автомобилей на бензоколонке: 11
Заправлено автомобилей на бензоколонке: 12
Бензоколонка закрыта.
Заправка закрыта.
```

3. Выводы

В результате выполнения практической работы №1 был получен навык работы с языком программирования Java. Был реализован собственный класс ***Thread*** и интерфейс ***Runnable***, с помощью которых были созданы потоки выполнения. Были изучены методы класса ***Thread*** и применены на практике.