Вопросы по теме Создание многопоточности:

1. Как создать новый поток?
2. Сколько способов создать поток вы знаете?
3. Разница между Thread и Runnable?
4. Чем отличаются интерфейсы Runnable и Callable?
5. Что такое daemon thread?
6. Можно ли сделать основной поток программы daemon?
7. Разница между методами start() и run()?
8. Что будет, если метод start() вызвать дважды?

Создание многопоточности.

Существует 3 способа создания потоков:

– класс java.lang.**Thread**;

– интерфейс java.lang.**Runnable**;

– интерфейс java.util.concurrent.**Callable**.

Создание потока с помощью класса Thread:

```
public static void main(String[] args) {
    Thread myThread = new MyThread();
    myThread.start();
}

static class MyThread extends Thread {
    @Override
    public void run() {
        System.out.println("doing something");
    }
}
```

1. создаем новый класс и наследуемся от класса Thread;
2. переопределяем в новом классе метод run();
3. в методе run() пишем код, который будет выполнять новый поток;
4. создаем экземпляр нового класса;
5. вызываем у этого экземпляра метод start().

Создание потока с помощью интерфейса Runnable:

```
public static void main(String[] args) {
    Thread myThread = new Thread(new MyThread());
    myThread.start();
}

static class MyThread implements Runnable {
    @Override
    public void run() {
        System.out.println("doing something");
    }
}
```

```
public static void main(String[] args) {
    Thread myThread = new Thread(() -> {
        System.out.println("doing something");
    });
    myThread.start();
}
```

Виталий Жуков
unicraft1989@gmail.com
Ссылка на курс Stepik

1. создаем новый класс, который будет реализовать интерфейс Runnable и переопределяем у него метод run();
2. в методе run() пишем код, который будет выполнять новый поток;
3. создаем экземпляр класса Thread, передав в его конструктор объект класса, реализующего Runnable;
4. вызываем у этого экземпляра метод start().

Различие между Thread и Runnable:

– при использовании интерфейса Runnable решается проблема наследования;

– Runnable – это функциональный интерфейс, поэтому может использовать лямбда-выражения (пункты 1-3 можно объединить в один если при создании экземпляра класса Thread в конструктор передадим лямбда-выражение в котором укажем код, который будет выполнен в новом потоке).

Создание потока с помощью интерфейса Callable:

```
public static void main(String[] args) {
    FutureTask<String> futureTask = new FutureTask<>(new MyThread());
    new Thread(futureTask).start();
    try {
        System.out.println(futureTask.get());
    } catch (InterruptedException | ExecutionException e) {
        e.printStackTrace();
    }
}

static class MyThread implements Callable {
    @Override
    public Object call() throws Exception {
        return Thread.currentThread().getName();
    }
}
```

```
public static void main(String[] args) {
    Callable<String> callable = () -> {
        return "doing someting";
    };
    FutureTask<String> futureTask = new FutureTask<>(callable);
    new Thread(futureTask).start();
    try {
        System.out.println(futureTask.get());
    } catch (InterruptedException | ExecutionException e) {
        e.printStackTrace();
    }
}
```

1. создаем новый класс, который будет реализовать интерфейс Callable;
2. переопределяем в новом классе метод call();
3. в методе call() пишем код, который будет выполнять новый поток, данный метод возвращает результат работы кода;
4. создаем объект типа FutureTask передав в его конструктор объект класса, реализующего Callable;
5. создаем новый поток с помощью класса Thread и запускаем его методом starts();
6. получаем из объекта FutureTask результат работы потока с помощью метода get().

Различие между Runnable и Callable:

– метод run() интерфейса Runnable ничего не возвращает, а метод call() интерфейса Callable возвращает результат работы потока;

– метод run() не может выбрасывать проверяемые исключения, а метод call() может.

Метод start():

– если создаем поток с помощью Runnable и вместо метода start() вызываем метод run(), то переопределенный метод run() выполнится, но новый поток при этом создан не будет;

– если метод start() вызвать дважды, у одного и того же объекта Thread, то будет вызвано исключение IllegalThreadStateException. Отработанный поток нельзя перезапустить.

Виталий Жуков
unicraft1989@gmail.com
Ссылка на курс Stepik

Daemon thread – вспомогательный поток, работающий в фоновом режиме, который предназначен для обслуживания основного потока (выполняет вспомогательные задачи для основного потока). Например, сборщик мусора.

Если основной поток завершается, то все daemon thread, привязанные к этому потоку, завершаются принудительно. При этом JVM их просто выключает и:

– блоки finally не выполняются;

– стеки не раскручиваются.

Исходя из вышесказанного, основной поток программы нельзя сделать daemon thread.

setDaemon(true) – вызов данного метода устанавливает поток как daemon thread (нужно вызвать перед запуском потока, иначе IllegalThreadStateException).

boolean isDaemon() – проверка, является ли поток daemon thread.

Виталий Жуков
unicraft1989@gmail.com
Ссылка на курс Stepik