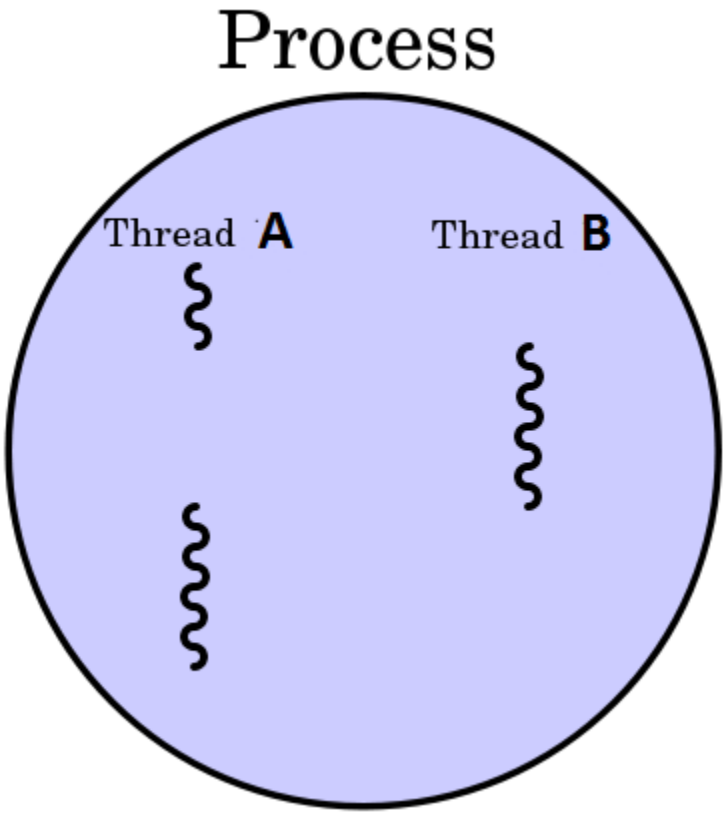
Потоки выполнения и синхронизация

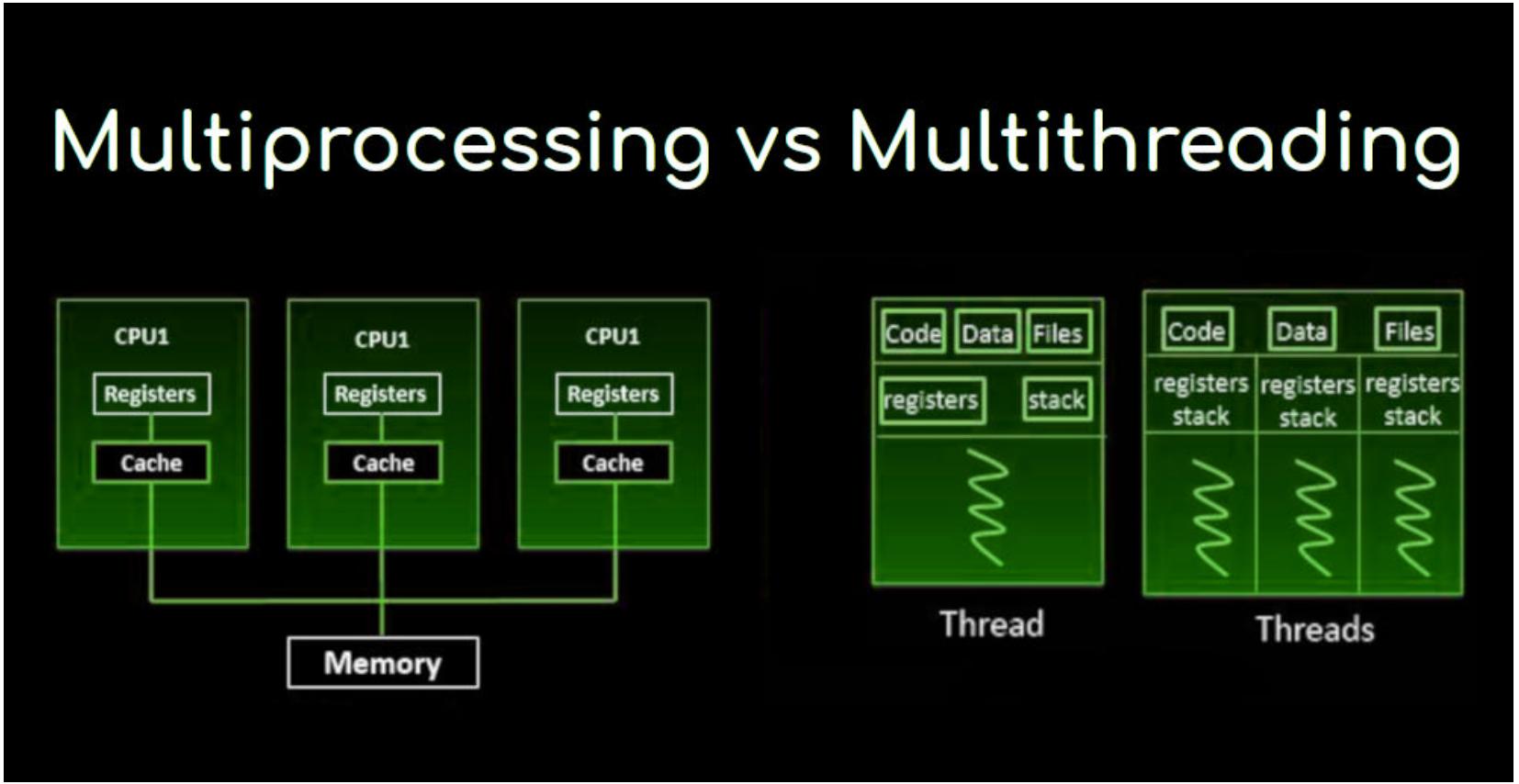
Рассматриваемые вопросы

1. Многопоточное программирование
2. Модель потоков исполнения в Java
3. Класс Thread и интерфейс Runnable
4. Использование интерфейса Callable и Future
5. Синхронизация потоков
6. Применение пакета Concurrent

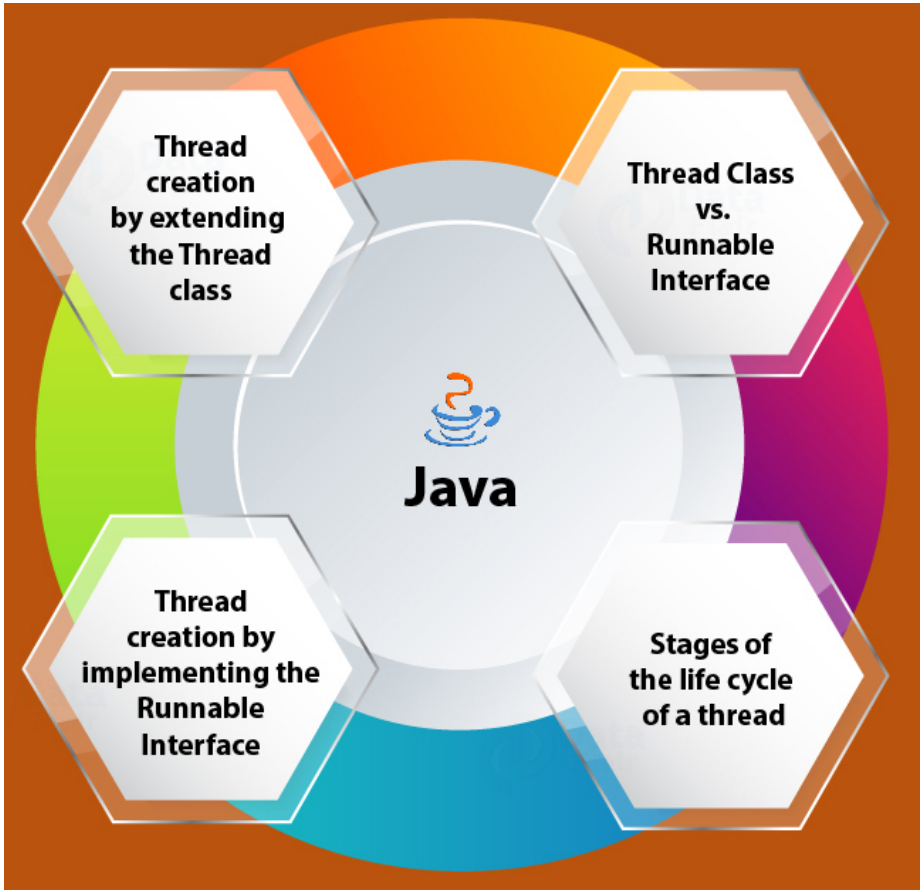
Многопоточное программирование



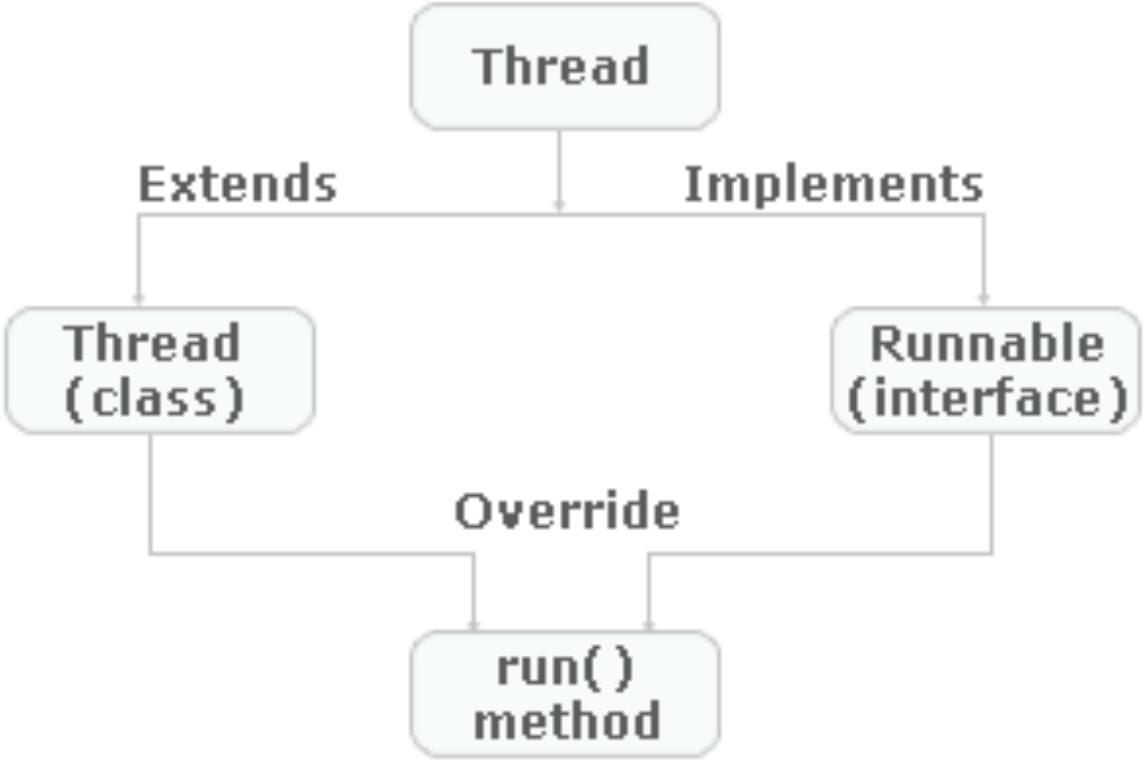
Многопоточное программирование



Модель потоков исполнения в Java



Класс Thread и интерфейс Runnable



Класс Thread и интерфейс Runnable

```
class MyRunnable implements Runnable {
    public void run(){
        System.out.println("MyRunnable running");
    }
}
```

Класс Thread и интерфейс Runnable

```
class Main {
    public static void main(String[] args) {
        Thread thread = new Thread(new MyRunnable());
        thread.start();

        Runnable myRunnable = new Runnable(){
            public void run(){
                System.out.println("Runnable running");
            }
        };
    }
}
```

Класс Thread и интерфейс Runnable

Основные методы класса Thread:

- getName - Получает имя потока исполнения
- getPriority - Получает приоритет потока исполнения
- isAlive - Определяет, выполняется ли поток
- join - Ожидает завершения потока исполнения
- run - Задает точку входа в поток исполнения
- sleep - Приостанавливает выполнение потока на заданное время
- start - Запускает поток исполнения, вызывая его метода run()

Класс Thread и интерфейс Runnable

```
class MyThread extends Thread {
    public void run(){
        System.out.println("MyThread running");
    }
}
```

Класс Thread и интерфейс Runnable

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

        Thread thread = new Thread(){
            public void run(){
                System.out.println("Thread Running");
            }
        };
        thread.start();
    }
}
```

Интерфейсы Callable и Future

Callable подобен Runnable, но с возвратом значения:

```
public interface Callable<V> {
    V call() throws Exception;
}
```

Интерфейсы Callable и Future

Future хранит результат асинхронного вычисления.

```
public interface Future<V> {
    boolean cancel(boolean mayInterruptIfRunning);
    boolean isCancelled();
    boolean isDone();
    V get() throws InterruptedException, ExecutionException;
    V get(long timeout, TimeUnit unit)
        throws InterruptedException, ExecutionException,
       TimeoutException;
}
```

Синхронизация потоков

Каждый объект в Java имеет свой монитор с операциями:

- monitorenter: захват монитора.
- monitorexit: освобождение монитора
- wait: перемещение текущего потока в ожидание и ожидание того, что произойдёт notify.
- notify(all): пробуждается один (или все) потоки, которые сейчас находятся ожидании.

Синхронизация потоков

- Метод wait() вынуждает вызывающий поток исполнения уступить монитор и перейти в состояние ожидания до тех пор, пока какой-нибудь другой поток исполнения не войдет в тот же монитор и не вызовет метод notify().
- Метод notify() возобновляет исполнение потока, из которого был вызван метод wait() для того же самого объекта.
- Метод notifyAll() возобновляет исполнение всех потоков, из которых был вызван метод wait() для того же самого объекта. Одному из этих потоков предоставляется доступ.

Применение синхронизированных методов

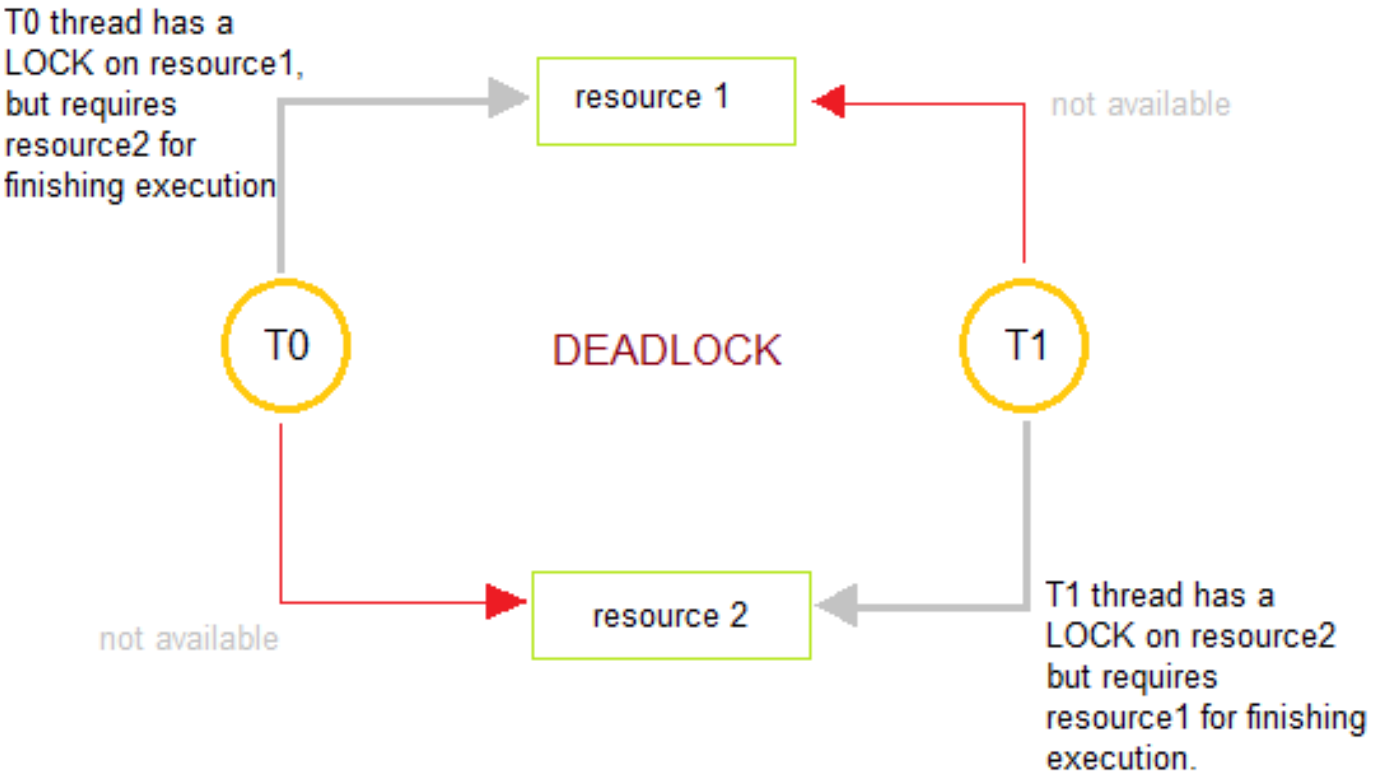
```
class SomeClass {

      synchronized void callMethod(...) {
            ...
      }
}
```

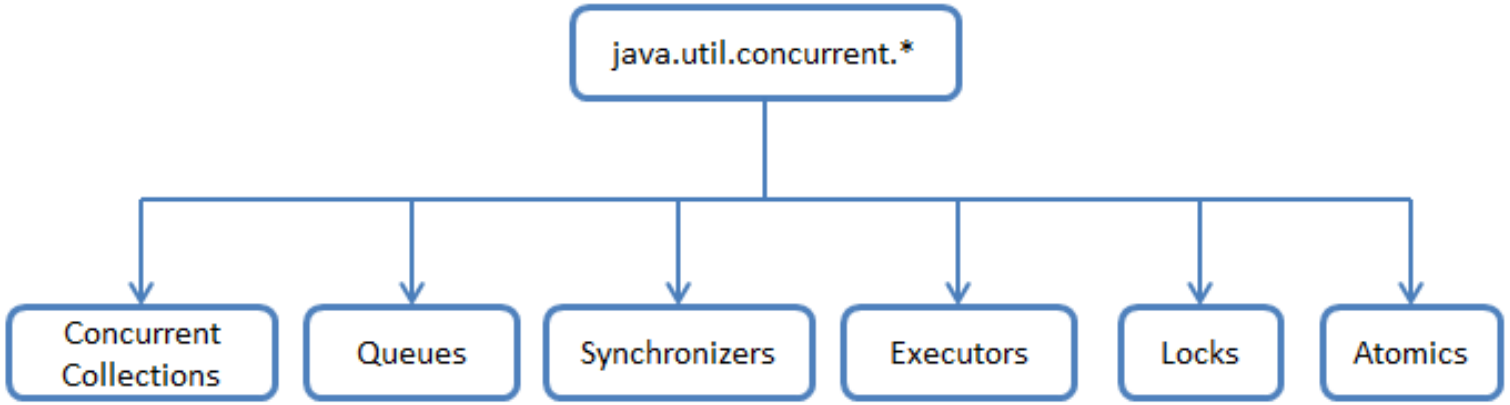
Оператор synchronized для объекта

```
synchronized(ссылка_на_объект) {
        // синхронизируемые операторы
}
```

Взаимная блокировка



Concurrency API



Вопросы для самоконтроля

1. Перечислите основные способы запуска кода в потоке
2. Перечислите основные способы синхронизации
3. Что такое Deadlock?
4. Перечислите состав пакета java.util.concurrent