Java Assignment 3
Aditya Kumar

Create and Run a Thread using Runnable Interface and Thread class.

- Use sleep and join methods with thread.
- Use a singleThreadExecutor to submit multiple threads.
- Try shutdown() and shutdownNow() and observe the difference.
- Use isShutDown() and isTerminate() with ExecutorService.
- Return a Future from ExecutorService by using callable and use get(), isDone(), isCancelled() with the Future object to know the status of task submitted.
- Submit List of tasks to ExecutorService and wait for the completion of all the tasks.
- Schedule task using schedule(), scheduleAtFixedRate() and scheduleAtFixedDelay()
- Increase concurrency with Thread pools using newCachedThreadPool() and newFixedThreadPool().
- Use Synchronize method to enable synchronization between multiple threads trying to access method at same time.
- Use Synchronize block to enable synchronization between multiple threads trying to access method at same time.
- Use Atomic Classes instead of Synchronize method and blocks.
- Coordinate 2 threads using wait() and notify().
- Coordinate mulitple threads using wait() and notifyAll()
- Use Reentract lock for coordinating 2 threads with signal(), signalAll() and wait().
- Create a deadlock and Resolve it using tryLock().

```
class MyRunnable implements Runnable{
     @Override
     public void run() {
          System.out.println("in runnable run method");
     }
}
class MyThread extends Thread{
     @Override
     public void run(){
          System.out.println("in Mythread run method");
     }
}
class MyThread1 {
     public static void main(String[] arg) {
       new Thread(new MyRunnable()).start();
         MyThread newThread=new MyThread();
       // newThread.start();
         newThread.run();
     }
}
```

```java
public class SleepAndJoinWithThread2 {
         static int counter = 0;
              public static void main(String[] args) throws InterruptedException {
                   Thread thread1 = new Thread(new Runnable() {
                        @Override
                        public void run() {
                             try {
                                  Thread.sleep(1000L);
                                  System.out.println("Running 1st Thread");
                             } catch (InterruptedException e) {
                                  e.printStackTrace();
                             }
                        }
                   });
                   Thread thread2 = new Thread(new Runnable() {
                        @Override
                        public void run() {
                             try {
                                  Thread.sleep(1000L);
                                  System.out.println("Running 2nd Thread");
                             } catch (InterruptedException e) {
                                  e.printStackTrace();
                             }
                        }
                   });
                   thread1.start();
                   thread2.start();
                   thread1.join();
                   thread2.join();
                   System.out.println("Ended....");
              }
         }
```

```java
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
public class SingleThreadExecutor3 {
    public static void main(String[] arg){
         ExecutorService executorService= Executors.newSingleThreadExecutor();
         try {
              executorService.submit(new Runnable() {
                   @Override
                   public void run() {
                        System.out.println("thread 1");
                   }
              });
              executorService.submit(new Runnable() {
                   @Override
                   public void run() {
                        System.out.println("thread 2");
                   }
              });
              executorService.submit(new Runnable() {
                   @Override
                   public void run() {
                        System.out.println("thread 3");
                   }
              });
         }
         finally{
              executorService.shutdown();
```

```
        }
        System.out.println(executorService.isShutdown());
        System.out.println(executorService.isTerminated());
        System.out.println("thread teminated successfully!");
    }
}


//
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
public class ShutDownAndShutDownNow {
    public static void main(String... arg){
        ExecutorService executorService= Executors.newSingleThreadExecutor();
        try {
            executorService.submit(new Runnable() {
                @Override
                public void run() {
                        System.out.println("thread 1");
                }
            });
            executorService.submit(new Runnable() {
                @Override
                public void run() {
                    try {
                        Thread.sleep(2000);
                        System.out.println("thread 2");
                    } catch (InterruptedException e) {
                        e.printStackTrace();
                    }
                }
            });
            executorService.submit(new Runnable() {
                @Override
                public void run() {
                    try {
                        Thread.sleep(2000);
                        System.out.println("thread 3");
                    } catch (InterruptedException e) {
                        e.printStackTrace();
                    }
                }
            });
        }
            finally{
                executorService.shutdown();
            }
            System.out.println(executorService.isShutdown());
           System.out.println(executorService.shutdownNow());
           System.out.println(executorService.isTerminated());
        }
    }


import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
public class IsShutDownAndIsTerminate5 {
    public static void main(String... arg){
        ExecutorService executorService= Executors.newSingleThreadExecutor();
        try {
            executorService.submit(new Runnable() {
                @Override
                public void run() {
```

```
                    System.out.println("thread 1");
                }
            });
            executorService.submit(new Runnable() {
                @Override
                public void run() {
                    try {
                        Thread.sleep(2000);
                        System.out.println("thread 2");
                    } catch (InterruptedException e) {
                        e.printStackTrace();
                    }
                }
            });
        }
        finally{
            executorService.shutdown();
        }
        System.out.println(executorService.isTerminated());
        System.out.println(executorService.isShutdown());
    }
}
```

```
//
import java.util.concurrent.ExecutionException;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
import java.util.concurrent.Future;
import java.util.concurrent.Callable;
public class FutureExecutorService6{
    public static void main(String[] args) throws ExecutionException,
InterruptedException {
        ExecutorService executorService = Executors.newSingleThreadExecutor();
        Future<Integer> integerFuture = executorService.submit(new Callable<Integer>() {
            @Override
            public Integer call() throws Exception {
                return 2;
            }
        });
        executorService.shutdown();
        if (integerFuture.isDone()) {
            System.out.println(integerFuture.get());
        }
        if(integerFuture.isCancelled()){
            System.out.println("Your task has been cancelled");
        }
    }
}
```

```
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
import java.util.concurrent.TimeUnit;
public class AwaitTermination7{
    public static void main(String[] args) throws InterruptedException {
        ExecutorService executorService = Executors.newSingleThreadExecutor();
        executorService.submit(()->{
            try {
                Thread.sleep(1000L);
                System.out.println("Thread Running");
            } catch (InterruptedException e) {
```

```
                    e.printStackTrace();
                }
            });
            executorService.shutdown();
            executorService.awaitTermination(500,TimeUnit.MILLISECONDS);
            if(executorService.isTerminated()){
                System.out.println("Terminated");
            }else{
                System.out.println("On or more tasks still remaining");
            }
        }
}



//
import java.util.concurrent.Executors;
import java.util.concurrent.ScheduledExecutorService;
import java.util.concurrent.TimeUnit;
public class ExecutorServiceSchedule8{
    public static void main(String[] args){
            ScheduledExecutorService executorService =
Executors.newSingleThreadScheduledExecutor();
            executorService
                    .scheduleWithFixedDelay(new Runnable() {
                                                    @Override
                                                    public void run() {
                                                        try {
                                                            Thread.sleep(2000L);

System.out.println("ScheduleWithFixedDelay Scheduled Task to executed after fixed
interval");
                                                        } catch (InterruptedException e) {
                                                            e.printStackTrace();
                                                        }
                                                    }
                                                }
                            ,
                            0,
                            1,
                            TimeUnit.SECONDS);
            executorService
                    .scheduleAtFixedRate(new Runnable() {
                                                    @Override
                                                    public void run() {
                                                        try {
                                                            Thread.sleep(2000L);

System.out.println("ScheduleAtFixedRate Scheduled Task to executed after fixed
interval");
                                                        } catch (InterruptedException e) {
                                                            e.printStackTrace();
                                                        }
                                                    }
                                                }
                            ,
                            0,
                            1,
                            TimeUnit.SECONDS);
    }
}
```

```java
import java.util.Random;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
class Process implements Runnable{
    int id;
    public Process(int id) {
        this.id = id;
    }
    @Override
    public void run() {
        System.out.println("Thread name::"+Thread.currentThread().getName()+"
Start :"+id);
        try {
            Thread.sleep(2000L);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        System.out.println("Thread name::"+Thread.currentThread().getName()+"
End :"+id);
    }
}
public class ConcurrencyThreadPool9{
    public static void main(String[] args) {
//      ExecutorService executorService= Executors.newFixedThreadPool(3);
        ExecutorService executorService= Executors.newCachedThreadPool();
        for (int i = 0; i <= 30; i++) {
            executorService.submit(new Process(i));
        }
        executorService.shutdown();
    }
}



//
import java.util.stream.IntStream;
public class SynchronizedMethod10{
    int count;
    synchronized public void incrementCount() {
        count++;
    }
    public void task1() {
        for (int iteration = 1; iteration <= 1000; iteration++) {
            incrementCount();
        }
    }
    public void task2() {
        for (int iteration2 = 1; iteration2 <= 1000; iteration2++) {
            incrementCount();
        }
    }
    public static void main(String[] args) throws InterruptedException {
        SynchronizedMethod10 synchronizeDemo = new SynchronizedMethod10();
        Thread thread1 = new Thread(new Runnable() {
            @Override
            public void run() {
                synchronizeDemo.task1();
            }
        });
        Thread thread2 = new Thread(new Runnable() {
            @Override
            public void run() {
                synchronizeDemo.task2();
            }
```

```
        });
        thread1.start();
        thread2.start();
        thread1.join();
        thread2.join();
        System.out.println(synchronizeDemo.count);
    }
}


//
import java.util.stream.IntStream;
public class SynchronizedBlock11{
    int count;
    public void incrementCount() {
        synchronized(this) {
            count++;
        }
    }
    public void task1() {
        for (int iteration = 1; iteration <= 1000; iteration++) {
            incrementCount();
        }
    }
    public void task2() {
        for (int iteration2 = 1; iteration2 <= 1000; iteration2++) {
            incrementCount();
        }
    }
    public static void main(String[] args) throws InterruptedException {
        SynchronizedBlock11 synchronizeDemo = new SynchronizedBlock11();
        Thread thread1 = new Thread(new Runnable() {
            @Override
            public void run() {
                synchronizeDemo.task1();
            }
        });
        Thread thread2 = new Thread(new Runnable() {
            @Override
            public void run() {
                synchronizeDemo.task2();
            }
        });
        thread1.start();
        thread2.start();
        thread1.join();
        thread2.join();
        System.out.println(synchronizeDemo.count);
    }
}


//
import java.util.concurrent.atomic.AtomicInteger;
import java.util.stream.IntStream;
public class SynchronisedAtomic12 {
    AtomicInteger count= new AtomicInteger();
    public void incrementCount() {
            count.incrementAndGet();
    }
    public void worker1() {
        for (int i = 1; i <= 1000; i++) {
            count.incrementAndGet();
```

```
        }
    }
    public void worker2() {
        for (int i = 1; i <= 1000; i++) {
            count.incrementAndGet();
        }
    }
    public static void main(String[] args) throws InterruptedException {
        SynchronisedAtomic12 synchronizeDemo = new SynchronisedAtomic12();
        Thread thread1 = new Thread(new Runnable() {
            @Override
            public void run() {
                synchronizeDemo.worker1();
            }
        });
        Thread thread2 = new Thread(new Runnable() {
            @Override
            public void run() {
                synchronizeDemo.worker2();
            }
        });
        thread1.start();
        thread2.start();
        thread1.join();
        thread2.join();
        System.out.println(synchronizeDemo.count);
    }
}



//
class ThreadWaitNotify13 {
    public void worker1(){
        synchronized (this) {
            System.out.println("Worker1 Started");
            try {
                wait();
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
            System.out.println("Worker1 Done");
        }
    }
    public void worker4(){
        synchronized (this) {
            System.out.println("Worker 4 Started");
            System.out.println("Worker 4 Done");
            notify();
        }
    }
    public static void main(String[] args) {
        ThreadWaitNotify13 demo = new ThreadWaitNotify13();
        new Thread(new Runnable() {
            @Override
            public void run() {
                demo.worker1();
            }
        }).start();
        new Thread(new Runnable() {
            @Override
            public void run() {
                demo.worker4();
```

```
            }
        }).start();
    }
}



//
public class ThreadWaitNotifyAll{
    public void worker1(){
        synchronized (this) {
            System.out.println("Worker1 Started");
            try {
                wait();
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
            System.out.println("Worker1 Done");
        }
    }
    public void worker2(){
        synchronized (this) {
            System.out.println("Worker 2 Started");
            try {
                wait();
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
            System.out.println("Worker 2 Done");
        }
    }
    public void worker3(){
        synchronized (this) {
            System.out.println("Worker 3 Started");
            try {
                wait();
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
            System.out.println("Worker 3 Done");
        }
    }
    public void worker4(){
        synchronized (this) {
            System.out.println("Worker 4 Started");
            System.out.println("Worker 4 Done");
            notifyAll();
        }
    }
    public static void main(String[] args) {
        ThreadWaitNotifyAll demo = new ThreadWaitNotifyAll();
        new Thread(new Runnable() {
            @Override
            public void run() {
                demo.worker1();
            }
        }).start();
        new Thread(new Runnable() {
            @Override
            public void run() {
                demo.worker2();
            }
        }).start();
```

```
        new Thread(new Runnable() {
            @Override
            public void run() {
                demo.worker3();
            }
        }).start();
        new Thread(new Runnable() {
            @Override
            public void run() {
                demo.worker4();
            }
        }).start();
    }
}


//

import java.util.concurrent.locks.Lock;
import java.util.concurrent.locks.ReentrantLock;
public class ReentractLock15 {
    Lock lock = new ReentrantLock(true);
    int count;
    void increment(){
        lock.lock();
        count++;
        lock.unlock();
    }
    public void worker1(){
        for (int i = 1; i <= 1000; i++) {
            increment();
        }
    }
    public void worker2(){
        for (int i = 1; i <= 1000; i++) {
            increment();
        }
    }
    public static void main(String[] args) throws InterruptedException {
        ReentractLock15 demo = new ReentractLock15();
        Thread thread1 = new Thread(new Runnable() {
            @Override
            public void run() {
                demo.worker1();
            }
        });
        Thread thread2 = new Thread(new Runnable() {
            @Override
            public void run() {
                demo.worker2();
            }
        });
        thread1.start();
        thread2.start();
        thread1.join();
        thread2.join();
        System.out.println(demo.count);
    }
}
```

```
//

import java.util.concurrent.locks.Condition;
import java.util.concurrent.locks.Lock;
import java.util.concurrent.locks.ReentrantLock;
public class DeadLockResolve16{
    Lock lock = new ReentrantLock(true);
    Condition condition = lock.newCondition();
    public void worker1() {
        try {
            lock.lock();
            System.out.println("worker 1 Started");
            condition.await();
            System.out.println("worker 1 Finished");
        } catch (InterruptedException e) {
            e.printStackTrace();
        } finally {
            lock.unlock();
        }
    }
    public void worker2() {
        try{
            lock.lock();
            System.out.println("worker 2 Started");
            System.out.println("worker 2 Finished");
            condition.signal();
        }finally {
            lock.unlock();
        }
    }
    public static void main(String[] args) throws InterruptedException {
        DeadLockResolve16 demo = new DeadLockResolve16();
        Thread thread1 = new Thread(new Runnable() {
            @Override
            public void run() {
                demo.worker1();
            }
        });
        Thread thread2 = new Thread(new Runnable() {
            @Override
            public void run() {
                demo.worker2();
            }
        });
        thread1.start();
        thread2.start();
        thread1.join();
        thread2.join();
    }
}



//
```