```java
package ex8;


import java.util.concurrent.*;

public class AccountWithoutSync {

    private Account account = new Account();

    public static void main(String[] args) {
        AccountWithoutSync aws = new AccountWithoutSync();
        aws.start();
    }

    public void start() {
        ExecutorService executor = Executors.newCachedThreadPool();

        // Create and launch 100 threads
        for (int i = 0; i < 100; i++) {
            executor.execute(new AddAPennyThread());
        }

        executor.shutdown();

        // Wait until all tasks are finished
        while (!executor.isTerminated()) {
        }

        System.out.println("What is the balance ? " + account.getBalance());
    }

    // An inner class of task for adding a penny to the account
    private class AddAPennyThread implements Runnable {

        public void run() {
            account.deposit(1);
        }
    }

    // An inner class for account
    private class Account {

        private int balance = 0;

        public int getBalance() {
            return balance;
        }

        public void deposit(int amount) {
            int newBalance = balance + amount;

            // This delay is deliberately added to magnify the
            // data-corruption problem and make it easy to see.
            try {
                Thread.sleep(1);
            } catch (InterruptedException ex) {
                // do nothing
            }

```

```java
            balance = newBalance;
        }
    }
}
```

```java
package ex8;


import java.util.concurrent.*;

public class AccountWithSync {

    private Account account = new Account();

    public static void main(String[] args) {
        AccountWithSync awsul = new AccountWithSync();
        awsul.start();
    }

    public void start() {
        ExecutorService executor = Executors.newCachedThreadPool();

        // Create and launch 100 threads
        for (int i = 0; i < 100; i++) {
            executor.execute(new AddAPennyThread());
        }

        executor.shutdown();

        // Wait until all tasks are finished
        while (!executor.isTerminated()) {
        }

        System.out.println("What is the balance ? " + account.getBalance());
    }

    // An inner class of task for adding a penny to the account
    private class AddAPennyThread implements Runnable {

        public void run() {
            account.deposit(1);
        }
    }

    // An inner class for account
    private class Account {

        private int balance = 0;

        public int getBalance() {
            return balance;
        }

        public synchronized void deposit(int amount) {

            int newBalance = balance + amount;

            // This delay is deliberately added to magnify the
            // data-corruption problem and make it easy to see.
            try {
                Thread.sleep(1);
            } catch (InterruptedException ex) {
                // do nothing
            }
```

```java

            balance = newBalance;
        }
    }
}

```

```java
package ex8;

import java.util.concurrent.*;
import java.util.concurrent.locks.*;

public class AccountWithSyncUsingLock {

    private Account account = new Account();

    public static void main(String[] args) {
        AccountWithSyncUsingLock awsul = new AccountWithSyncUsingLock();
        awsul.start();
    }

    public void start() {
        ExecutorService executor = Executors.newCachedThreadPool();

        // Create and launch 100 threads
        for (int i = 0; i < 100; i++) {
            executor.execute(new AddAPennyThread());
        }

        executor.shutdown();

        // Wait until all tasks are finished
        while (!executor.isTerminated()) {
        }

        System.out.println("What is the balance ? " + account.getBalance());
    }

    // An inner class of task for adding a penny to the account
    private class AddAPennyThread implements Runnable {

        public void run() {
            account.deposit(1);
        }
    }

    // An inner class for account
    private class Account {

        private ReentrantLock lock = new ReentrantLock(); // Create a lock
        private int balance = 0;

        public int getBalance() {
            return balance;
        }

        public void deposit(int amount) {
            lock.lock(); // Acquire the lock

            try {
                int newBalance = balance + amount;

                // This delay is deliberately added to magnify the
                // data-corruption problem and make it easy to see.
                Thread.sleep(5);

```

```java
                balance = newBalance;
            } catch (InterruptedException ex) {
            } finally {
                lock.unlock(); // Release the lock
            }
        }
    }
}

```

```java
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */
package ex8;

/**
 *
 * @author vanting
 */
class Counter {

    private int c = 0;

    public void increment() {
        c++;
    }

    public void decrement() {
        c--;
    }

    public int value() {
        return c;
    }

}

```

```java
package ex8;

import java.util.concurrent.*;

public class ExecutorDemo implements Runnable {

    private char c;

    ExecutorDemo(char c) {
        this.c = c;
    }

    public void run() {
        for (int i = 0; i < 10; i++) {
            System.out.print(c);
            try {
                Thread.sleep(500);        // half secs
            } catch (InterruptedException ex) {
                // do nothing
            }
        }
    }

    public static void main(String[] args) {

        // Create a fixed thread pool with maximum three threads
        ExecutorService pool = Executors.newFixedThreadPool(3);

        // Submit runnable tasks to the executor
        pool.execute(new ExecutorDemo('a'));
        pool.execute(new ExecutorDemo('b'));
        pool.execute(new ExecutorDemo('c'));
        pool.execute(new ExecutorDemo('d'));

    }
}

```

```java
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package ex8;

import java.util.ArrayList;
import java.util.List;
import java.util.concurrent.ForkJoinPool;
import java.util.concurrent.RecursiveTask;

/**
 *
 * @author vanting
 */
public class ForkJoinTest {

    public static void main(String[] args) {
        ArrayList<Double> list = new ArrayList<>(1000);
        for (int i = 0; i < 1000; i++) {
            list.add(Math.random() * 100.0);
        }

        ParallelSum task = new ParallelSum(list, 0, list.size());

        // Create a ForkJoinPool to run the task
        ForkJoinPool pool = new ForkJoinPool();

        // Submit the task to the Fork/Join pool for execution
        double sum = pool.invoke(task);

        System.out.println("Sum is " + sum);
    }
}

class ParallelSum extends RecursiveTask<Double> {

    private static final int THRESHOLD = 100;
    private List<Double> list;
    private int start, end; // sum up elements from start to end-1

    public ParallelSum(List<Double> a, int s, int e) {
        list = a;
        start = s;
        end = e;
    }

    @Override
    protected Double compute() {
        double result = 0;

        // Base case
        if (end - start < THRESHOLD) {
            for (int i = start; i < end; i++) {
                result += list.get(i);
            }
        } else {
            // Divide into 2 sub-tasks
```

```java
            int mid = (start + end) / 2;
            ParallelSum task1 = new ParallelSum(list, start, mid);
            ParallelSum task2 = new ParallelSum(list, mid, end);

            task1.fork(); // Launch the subtasks
            task2.fork();

            // Wait for subtasks to finish and combine the result
            result = task1.join()+ task2.join();
        }
        return result;
    }
}

```

```java
package ex8;

/**
 *
 * @author vanting
 */
public class HelloRunnable implements Runnable {

    public void run() {
        System.out.println("Hello from a thread!");
    }

    public static void main(String args[]) {
        HelloRunnable task = new HelloRunnable();
        Thread t = new Thread(task);
        t.start();
    }
}

```

```java
package ex8;

/**
 *
 * @author vanting
 */
public class HelloThread extends Thread {

    @Override
    public void run() {
        System.out.println("Hello from a thread!");
    }

    public static void main(String args[]) {
        Thread t = new HelloThread();
        t.start();
        //t.run();
    }
}

```

```java
package ex8;

/**
 *
 * @author vanting
 */
public class InterruptDemo1 implements Runnable {

    public void run() {
        System.err.println("Running");
        while(true) {
            if(Thread.interrupted())
                break;
        }
        System.err.println("Stopped by Interruption.");
    }

    public static void main(String args[]) throws InterruptedException {
        Thread t = new Thread(new InterruptDemo1());
        t.start();

        for(int i=10; i>0; i--) {
            Thread.sleep(1000);         // 1 sec
            System.out.print(i + " ");
        }
        System.out.println("");
        t.interrupt();
    }
}

```

```java
package ex8;

/**
 *
 * @author vanting
 */
public class InterruptDemo2 implements Runnable {

    public void run() {
        System.err.println("Start to sleep.");
        try {
            Thread.sleep(10000);        // 10 secs
        } catch (InterruptedException ex) {
            System.err.println("Woken up by Interruption.");

        }

    }

    public static void main(String args[]) throws InterruptedException {
        Thread t = new Thread(new InterruptDemo2());
        t.start();

        for(int i=5; i>0; i--) {
            Thread.sleep(1000);         // 1 sec
            System.out.print(i + " ");
        }
        System.out.println("");
        t.interrupt();
    }
}

```

```java
package ex8;

/**
 *
 * @author vanting
 */
public class JoinDemo implements Runnable {

    public void run() {
        for (int i = 0; i < 1000; i++)
            System.out.print("T");
    }

    public static void main(String args[]) throws InterruptedException {
        Thread t = new Thread(new JoinDemo());
        t.start();
        t.join(); // wait for the finish of thread t

        for (int i = 0; i < 1000; i++)
            System.out.print("M");
    }
}

```

```java
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */
package ex8;

/**
 *
 * @author vanting
 */
public class MsLunch {

    private long c1 = 0;
    private long c2 = 0;
    private Object lock1 = new Object();
    private Object lock2 = new Object();

    public void inc1() {
        synchronized (lock1) {
            c1++;
        }
    }

    public void inc2() {
        synchronized (lock2) {
            c2++;
        }
    }
}

```

```java
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package ex8;

import java.util.concurrent.Callable;
import java.util.concurrent.ExecutionException;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
import java.util.concurrent.Future;

/**
 *
 * @author vanting
 */
public class MultiThreadMaxFinder {

    public static void main(String[] args) throws InterruptedException,
ExecutionException  {

        int[] array = new int[] {8, 3, 99, 23, 14, 50, 39};
        System.out.println("The max is: " + max(array));
    }

    public static int max(int[] data) throws InterruptedException, ExecutionException
{

        if (data.length == 0) {
            throw new IllegalArgumentException();
        }

        if (data.length == 1) {
            return data[0];
        }

        // Divide the array into 2 parts.
        // Use a separate thread to find the max in each part.
        int mid = data.length / 2;
        FindMaxTask t1 = new FindMaxTask(data, 0, mid);
        FindMaxTask t2 = new FindMaxTask(data, mid, data.length);

        // Executors is a factory for creating Executor
        // ExecutorService, and ScheduledExecutorService
        ExecutorService s = Executors.newFixedThreadPool(2);

        // Asynchronous tasks
        Future<Integer> f1 = s.submit(t1);
        Future<Integer> f2 = s.submit(t2);

        // When f1.get() is called, the main thread is
        // blocked and waits for f1 to finish.
        // Similarly for f2.get().
        int result = Math.max(f1.get(), f2.get());

        s.shutdown();  // Terminate the threads
        return result;
    }
```

```java
}

class FindMaxTask implements Callable<Integer> {

    private int[] data;
    private int start, end;

    FindMaxTask(int[] d, int s, int e) {
        data = d;
        start = s;
        end = e;
    }

    // find the max value in data[start..(end-1)]
    public Integer call() {
        int max = data[start];
        for (int i = start + 1; i < end; i++) {
            if (max < data[i]) {
                max = data[i];
            }
        }

        return max;
    }
}

```

```java
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package ex8;

import java.util.concurrent.locks.Condition;
import java.util.concurrent.locks.ReentrantLock;
import java.util.logging.Level;
import java.util.logging.Logger;

/**
 *
 * @author vanting
 */


public class ProducerConsumer {

    public static Account num = new Account(0);

    public static void main(String[] args) {

        Thread add = new Thread(new MyPCThread(true));
        Thread minus = new Thread(new MyPCThread(false));

        add.start();
        minus.start();
    }

}

class Account {

    private int balance;

    private ReentrantLock lock = new ReentrantLock();   // Create a new lock
    private Condition newDeposit = lock.newCondition();     // Create a condition

    public Account(int v) {
        this.balance = v;
    }

    public int get() {
        return balance;
    }

    public int incrementRandomAndGet() {
        lock.lock();
        int amount = (int) (Math.random() * 50) + 1;
        balance += amount;
        System.out.println("Balance: " + balance + " (+" + amount + ")");
        try {
            newDeposit.signalAll();
        } finally {
            lock.unlock();
        }
        return balance;
```

```java
    }

    public int decrementRandomAndGet() {
        lock.lock();
        int amount = (int) (Math.random() * 10) + 1;
        try {
            while (balance < amount) {
                System.out.println("Not enough balance...");
                newDeposit.await();
                System.out.println("OK, enough.");
            }
            balance -= amount;
            System.out.println("Balance: " + balance + " (-" + amount + ")");
        } catch (InterruptedException ex) {
            Logger.getLogger(Account.class.getName()).log(Level.SEVERE, null, ex);
        } finally {
            lock.unlock();
        }
        return balance;
    }
}

class MyPCThread implements Runnable {

    private final boolean isAdd;

    public MyPCThread(boolean isAdd) {
        this.isAdd = isAdd;
    }

    @Override
    public void run() {
        if (isAdd) {
            while (true) {
                ProducerConsumer.num.incrementRandomAndGet();
                try {
                    Thread.sleep(3000);
                } catch (InterruptedException ex) {
                    Logger.getLogger(MyPCThread.class.getName()).log(Level.SEVERE, null, ex);
                }
            }
        } else {
            while (true) {
                try {
                    // this thread run 10x faster
                    Thread.sleep(300);
                } catch (InterruptedException ex) {
                    Logger.getLogger(MyPCThread.class.getName()).log(Level.SEVERE, null, ex);
                }
                ProducerConsumer.num.decrementRandomAndGet();
            }
        }
    }
}


```

```java
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package ex8;

import java.util.concurrent.atomic.AtomicInteger;

/**
 *
 * @author vanting
 */
class NonAtomicNum {

    // regular int variables are non-atomic
    private int value;

    public NonAtomicNum(int v) {
        this.value = v;
    }

    public int get() {
        return value;
    }

    public int incrementAndGet() {
        return ++value;
    }

    public int decrementAndGet() {
        return --value;
    }

    /*
    public synchronized int incrementAndGet() {
        return ++value;
    }

    public synchronized int decrementAndGet() {
        return --value;
    }
     */
}

class MyRaceThread implements Runnable {

    private final boolean isAdd;

    public MyRaceThread(boolean isAdd) {
        this.isAdd = isAdd;
    }

    @Override
    public void run() {
        if (isAdd) {
            for (int i = 0; i < 10000; i++) {
                RaceCondition.num.incrementAndGet();
            }
```

```java
        } else {
            for (int i = 0; i < 10000; i++) {
                RaceCondition.num.decrementAndGet();
            }
        }
    }
}

public class RaceCondition {

    public static NonAtomicNum num = new NonAtomicNum(0);
    //public static AtomicInteger num = new AtomicInteger(0);

    public static void main(String[] args) {

        Thread add = new Thread(new MyRaceThread(true));
        Thread minus = new Thread(new MyRaceThread(false));

        add.start();
        minus.start();

        /*
         * wait both threads to finish
         */
        try {
            add.join();
            minus.join();
        } catch (InterruptedException e) {
            e.printStackTrace();
        }

        // the expected output should be zero
        System.out.println(num.get());
    }

}
```

```java

package ex8;

/**
 *
 * @author vanting
 */
public class SleepDemo {

    public static void main(String args[]) throws InterruptedException {
        System.out.print("Thinking");
        for(int i=0; i<20; i++) {
            Thread.sleep(1000);      //Pause for 1 seconds
            System.out.print(".");
        }
        System.out.println("Done!");
    }
}

```

```java
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package ex8;

/**
 *
 * @author vanting
 */
public class StringBufferVsStringBuilder {

    public static void main(String[] args) {
        int N = 77777777;
        long t;

        {
            // thread safe
            StringBuffer sb = new StringBuffer();
            t = System.currentTimeMillis();
            for (int i = N; i-- > 0;) {
                sb.append("");
            }
            System.out.println(System.currentTimeMillis() - t);
        }

        {
            // not thread safe
            StringBuilder sb = new StringBuilder();
            t = System.currentTimeMillis();
            for (int i = N; i-- > 0;) {
                sb.append("");
            }
            System.out.println(System.currentTimeMillis() - t);
        }
    }
}

```

```java
package ex8;
import java.util.concurrent.*;
import java.util.concurrent.locks.*;

public class ThreadCooperation {

    private Account account = new Account();

    public static void main(String[] args) {
        ThreadCooperation tc = new ThreadCooperation();
        tc.start();
    }

    public void start() {
        // Create a thread pool with two threads
        ExecutorService executor = Executors.newFixedThreadPool(2);
        executor.execute(new DepositTask());
        executor.execute(new WithdrawTask());
        executor.shutdown();

        System.out.println("Thread 1\t\tThread 2\t\tBalance");
    }

    // A task for adding an amount to the account
    public class DepositTask implements Runnable {

        public void run() {
            try { // Purposely delay it to let the withdraw method proceed
                while (true) {
                    account.deposit((int) (Math.random() * 10) + 1);
                    Thread.sleep(1000);
                }
            } catch (InterruptedException ex) {
                ex.printStackTrace();
            }
        }
    }

    // A task for subtracting an amount from the account
    public class WithdrawTask implements Runnable {

        public void run() {
            while (true) {
                account.withdraw((int) (Math.random() * 10) + 1);
            }
        }
    }

    // An inner class for account
    private class Account {
        // Create a new lock
        private ReentrantLock lock = new ReentrantLock();        // Create a
condition
        private Condition newDeposit = lock.newCondition();
        private int balance = 0;

        public int getBalance() {
            return balance;
        }
```

```java

        public void withdraw(int amount) {
            lock.lock(); // Acquire the lock
            try {
                while (balance < amount) {
                    newDeposit.await();
                }
                balance -= amount;
                System.out.println("\t\t\tWithdraw " + amount + "\t\t" +
getBalance());
            } catch (InterruptedException ex) {
                ex.printStackTrace();
            } finally {
                lock.unlock(); // Release the lock
            }
        }

        public void deposit(int amount) {
            lock.lock(); // Acquire the lock
            try {
                balance += amount;
                System.out.println("Deposit " + amount + "\t\t\t\t\t" +
getBalance());

                // Signal thread waiting on the condition
                newDeposit.signalAll();
            } finally {
                lock.unlock(); // Release the lock
            }
        }
    }
}
```

```java
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */
package ex8;

/**
 *
 * @author vanting
 */
public class ThreadSwitching {

    public static void main(String[] args) {
        MySwitchingThread t1 = new MySwitchingThread(0, 3, 300);
        MySwitchingThread t2 = new MySwitchingThread(1, 3, 300);
        MySwitchingThread t3 = new MySwitchingThread(2, 3, 300);

        t1.start();
        t2.start();
        t3.start();
    }
}

class MySwitchingThread extends Thread {

    private int startIdx, nThreads, maxIdx;

    public MySwitchingThread(int s, int n, int m) {
        this.startIdx = s;
        this.nThreads = n;
        this.maxIdx = m;
    }

    @Override
    public void run() {
        for (int i = this.startIdx; i < this.maxIdx; i += this.nThreads) {
            System.out.println("[ID " + this.getId() + "] " + i);
        }
    }
}

```