Assignment 3:

Write a program with data structure, use atomic methods like get(),incrementAndGet(),decrementAndGet(),compareAndSet(),etc ,also use all other functionalities to make the program more responsive.

Main.java

```
import java.util.List;
import java.util.concurrent.ForkJoinPool;
import java.util.concurrent.TimeUnit;

import package6.Employe;
import package6.Employegen;
import package6.Thread;

class Main {
    public static void main(String args[]) {
        Employegen gen = new Employegen();
        List<Employe> employes = gen.generate(10);
        Thread thread = new Thread(employes, 0, employes.size(), 0.20);
        for (int i = 0; i < employes.size(); i++) {
            Employe employ = employes.get(i);
            System.out.printf("Employe %s: %f \n", employ.getName(), employ.getSalary());
        }
        System.out
                .println("------------------------------------------------------------------
-------------------------");
        System.out.println("To Increase the salary of Employes");
        System.out
                .println("------------------------------------------------------------------
-------------------------");
        ForkJoinPool pool = new ForkJoinPool();

        pool.execute(thread);
        do {
            System.out.printf("**************************************\n");
            System.out.printf("Main: Pralleism:%d\n", pool.getCommonPoolParallelism());
        } while (!thread.isDone());
        pool.shutdown();

        if (thread.isCompletedNormally()) {
            System.out.println("Main: The process has completed normally. \n");
        }
        for (int i = 0; i < employes.size(); i++) {
            Employe employ = employes.get(i);
            System.out.printf("Employe %s: %f \n", employ.getName(), employ.getSalary());
        }
    }
}
```

Employe.java

```
public class Employe {
    private int empid;
    private double empsalary;
    private String empname;

    public String getName() {
        return empname;
    }

    public void setName(String name) {
        this.empname = name;
    }

    public double getSalary() {
        return empsalary;
    }

    public void setSalary(double salary) {
        this.empsalary = salary;
    }

    public int getId() {
        return empid;
    }

    public void setId(int id) {
        this.empid = id;
    }
}
```

Employegen.java

```
import java.util.concurrent.atomic.AtomicInteger;
import java.util.*;

public class Employegen {
    public List<Employe> generate(int size) {
        List<Employe> emp = new ArrayList<Employe>();
        AtomicInteger val = new AtomicInteger(0);
        AtomicInteger val1 = new AtomicInteger(20000);
        for (int i = 0; i < size; i++) {
            Employe employe = new Employe();
            employe.setName("emp" + (i + 1));
            employe.setId(val.incrementAndGet());
            employe.setSalary(val1.decrementAndGet());
            emp.add(employe);
        }
        return emp;
    }
}
```

Thread.java

```
import java.util.*;
import java.util.concurrent.RecursiveAction;

public class Thread extends RecursiveAction {
    private List<Employe> employes;
    private int first;
    private int last;
    private double increment;

    public Thread(List<Employe> Employes, int first, int last, double increment) {
        this.employes = Employes;
        this.first = first;
        this.last = last;
        this.increment = increment;
    }

    protected void compute() {
        if (last - first < 10) {
            updateSalary();
        } else {
            int middle = (first + last) / 2;
            System.out.printf("Task pending tasks: %s\n", getQueuedTaskCount());
            Thread t1 = new Thread(employes, first, middle + 1, increment);
            Thread t2 = new Thread(employes, middle + 1, last, increment);
            invokeAll(t1, t2);

        }
    }

    private void updateSalary() {
        for (int i = first; i < last; i++) {
            Employe employe = employes.get(i);
            employe.setSalary((employe.getSalary()) * 2);
        }
    }
}
```

Output:

```
Employe emp1: 19999.000000
Employe emp2: 19998.000000
Employe emp3: 19997.000000
Employe emp4: 19996.000000
Employe emp5: 19995.000000
Employe emp6: 19994.000000
Employe emp7: 19993.000000
Employe emp8: 19992.000000
Employe emp9: 19991.000000
Employe emp10: 19990.000000
```

```
-----------------------------------------------------------------------------------------------
To Increase the salary of Employes
-----------------------------------------------------------------------------------------------
************************************
Main: Pralleism:7
************************************
Main: Pralleism:7
************************************
Task pending tasks: 0
Main: Pralleism:7
************************************
Main: Pralleism:7
************************************
Main: Pralleism:7
Main: The process has completed normally.

Employe emp1: 39998.000000
Employe emp2: 39996.000000
Employe emp3: 39994.000000
Employe emp4: 39992.000000
Employe emp5: 39990.000000
Employe emp6: 39988.000000
Employe emp7: 39986.000000
Employe emp8: 39984.000000
Employe emp9: 39982.000000
Employe emp10: 39980.000000
```