

Site : Luminy St-Charles St-Jérôme Cht-Gombert Aix-Montperrin Aubagne-SATIS
Sujet de : 1^{er} semestre 2^{ème} semestre Session 2 Durée de l'épreuve : 4h
Examen de : L2 Nom du diplôme : Licence d'informatique
Code du module : SIN3U02 Libellé du module : Programmation 2
Calculatrices autorisées : NON Documents autorisés : OUI

1 Correction de l'examen

Vous trouverez ci-dessous une correction possible de l'examen. C'est une des corrections possibles mais pas la seule.

2 La tâche de calcul : classe Job (2 points)

```
package cluster;

public class Job {
    private final int memory;
    private final int id;
    private static int jobCount = 0;

    public Job(int memory) {
        if (memory <= 0){
            throw new IllegalArgumentException("Cannot create a job with negative memory : " + getMemory());
        }
        this.memory = memory;
        id = jobCount++;
    }

    public int getMemory() {
        return memory;
    }

    public int getId() {
        return id;
    }

    public static void resetJobCount() {
        jobCount = 0;
    }

    public static int getJobCount() {
        return jobCount;
    }

    public String toString(){
        return "Job "+ getId() + " (" + getMemory() + "Mo)";
    }
}
```

3 Tester la classe Job : classe TestJob (3 points)

```
package cluster;

import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.*;
import static org.assertj.core.api.AssertionsForClassTypes.assertThat;

public class TestJob {

    @Test
    void testToString(){
        // Test with Junit
        Job job1 = new Job(100);
        String expectedStringRepresentation1 = "Job " + job1.getId() + " (100Mo)";
        assertEquals(expectedStringRepresentation1, job1.toString());

        // Test with assertJ
        Job job2 = new Job(1);
        String expectedStringRepresentation2 = "Job " + job2.getId() + " (1Mo)";
        assertThat(job2.toString()).isEqualTo(expectedStringRepresentation2);
    }

    @Test
    void testResetJobCount(){
        Job.resetJobCount();
        Job job = new Job(10);
        assertThat(job.getId()).isEqualTo(0);
    }

    @Test
    void testGetId(){
        int[] memories = {10, 1000};
        for(int memory : memories){
            Job job = new Job(memory);
            assertThat(job.getMemory()).isEqualTo(memory);
        }
    }

    @Test
    void testGetJobCount(){
        Job.resetJobCount();
        for(int jobCount = 1; jobCount < 4; jobCount++){
            Job job = new Job(10);
            assertThat(Job.getJobCount()).isEqualTo(jobCount);
        }
    }
}
```

4 Les nœuds de la ferme de calcul : classe Node (2 points)

```
package cluster;

import java.util.ArrayList;
import java.util.List;
```

```

public class Node {
    private final String name;
    private final int memoryCapacity;

    private int availableMemory;
    private final List<Job> assignedJobs = new ArrayList<>();

    Node(String name, int memoryCapacity) {
        this.name = name;
        this.memoryCapacity = memoryCapacity;
        availableMemory = memoryCapacity;
    }

    boolean canAccept(Job job){
        return availableMemory >= job.getMemory();
    }

    boolean canHandle(Job job){
        return memoryCapacity >= job.getMemory();
    }

    void accept(Job job){
        assignedJobs.add(job);
        availableMemory -= job.getMemory();
    }

    int usedMemory(){
        return memoryCapacity - availableMemory;
    }

    public String toString(){
        return "Node " + name + " (" + usedMemory() + "/" + memoryCapacity +)";
    }

    void printJobs(){
        System.out.println(this);
        for(Job job : assignedJobs){
            System.out.println(job);
        }
    }
}

```

5 Exception NotEnoughMemoryException (2 points)

```

package cluster;

public abstract class NotEnoughMemoryException extends Exception {
    private final Node node;
    private final Job job;

    public NotEnoughMemoryException(Node node, Job job) {
        this.node = node;
        this.job = job;
    }

    @Override

```

```

    public String getMessage() {
        return node + getSentenceMessage() + job;
    }

    public abstract String getSentenceMessage();
}

package cluster;

public class NotEnoughRemainingMemoryException extends NotEnoughMemoryException{
    public NotEnoughRemainingMemoryException(Node node, Job job) {
        super(node, job);
    }

    @Override
    public String getSentenceMessage() {
        return " has not enough remaining memory to handle ";
    }
}

package cluster;

public class NotEnoughTotalMemoryException extends NotEnoughMemoryException {
    public NotEnoughTotalMemoryException(Node node, Job job) {
        super(node, job);
    }

    @Override
    public String getSentenceMessage() {
        return " has not enough total memory to handle ";
    }
}

public class Node {
    void accept(Job job) throws NotEnoughMemoryException{
        if(!canHandle(job)){
            throw new NotEnoughTotalMemoryException(this, job);
        }
        if(!canAccept(job)){
            throw new NotEnoughRemainingMemoryException(this, job);
        }
        assignedJobs.add(job);
        availableMemory -= job.getMemory();
    }
}

```

6 Ordonnanceur : interface Scheduler (2 points)

```

package cluster;

import java.util.List;

public interface Scheduler<J extends Job> {
    List<J> scheduleJobs(List<J> jobs, List<Node> nodes);
}

```

7 Ordonnanceur aléatoire : classe RandomScheduler (2 points)

```
package cluster;

import java.util.ArrayList;
import java.util.List;
import java.util.Random;

public class RandomScheduler implements Scheduler<Job>{
    private Random random = new Random();

    public Node randomNode(List<Node> nodes){
        return nodes.get(random.nextInt(nodes.size()));
    }

    @Override
    public List<Job> scheduleJobs(List<Job> jobs, List<Node> nodes) {
        List<Job> nonScheduledJobs = new ArrayList<>();
        for(Job job : jobs){
            try{
                randomNode(nodes).accept(job);
            }
            catch (NotEnoughMemoryException e){
                nonScheduledJobs .add(job);
            }
        }
        return nonScheduledJobs ;
    }
}
```

8 Classe PriorityJob (2 points)

```
package cluster;

public class PriorityJob extends Job implements Comparable<PriorityJob>{
    private final int priority;

    public PriorityJob(int memory, int priority) {
        super(memory);
        this.priority = priority;
    }

    @Override
    public int compareTo(PriorityJob job) {
        return job.priority - this.priority;
    }
}
```

9 Ordonnanceur avec priorité : classe PriorityScheduler (2 points)

```
package cluster;

import java.util.ArrayList;
import java.util.Collections;
import java.util.List;
```

```

public class PriorityScheduler implements Scheduler<PriorityJob> {
    @Override
    public List<PriorityJob> scheduleJobs(List<PriorityJob> jobs, List<Node> nodes) {
        List<PriorityJob> nonScheduledJobs = new ArrayList<>();
        Collections.sort(jobs);
        for (PriorityJob job : jobs) {
            if(!scheduleJob(job, nodes)){
                nonScheduledJobs.add(job);
            }
        }
        return nonScheduledJobs;
    }

    private boolean scheduleJob(Job job, List<Node> nodes){
        for (Node node : nodes) {
            try {
                node.accept(job);
                return true;
            }
            catch (NotEnoughMemoryException e){}
        }
        return false;
    }
}

```

10 Classe Controller (3 points)

```

package cluster;

import java.util.ArrayList;
import java.util.List;

public class Controller<J extends Job> {
    private List<J> nonScheduledJobs = new ArrayList<>();
    private final List<Node> nodes = new ArrayList<>();
    private final String name;
    private final Scheduler<J> scheduler;

    void addNode(Node node){
        nodes.add(node);
    }

    void submitJob(J job){
        nonScheduledJobs.add(job);
    }

    Controller(String name, Scheduler<J> scheduler){
        this.name = name;
        this.scheduler = scheduler;
    }

    void printNodesAndNonScheduledJobs(){
        System.out.println(this);
        printNodes();
        System.out.println("Non scheduled jobs");
    }
}

```

```
    printNonScheduledJobs();
}

private void printNodes(){
    for (Node node : nodes){
        node.printJobs();
        System.out.println();
    }
}

private void printNonScheduledJobs(){
    for (Job job : nonScheduledJobs){
        System.out.println(job);
    }
}

public void scheduleJobs(){
    nonScheduledJobs = scheduler.scheduleJobs(nonScheduledJobs, nodes);
}

@Override
public String toString() {
    return "Controller " + name;
}
}
```