**Lab 4**

|  |  |  |  |
| --- | --- | --- | --- |
| Student Name | | Student CSUSM ID | Contribution percentage |
| 1 | Lauren Gonzalez | gonza823 | 50 |
| 2 | Sirena Murphree | murph135 | 50 |

**Grading Rubrics (for instructor only):**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Criteria | 1. Beginning | 2. Developing | 3. Proficient | 4. Exemplary |
| Program: functionality  *correctness* | 0-9 | 10-14 | 15-19 | 20 |
|  |  |  |  |
| Program: functionality  *Behavior Testing* | 0-9 | 10-14 | 15-19 | 20 |
|  |  |  |  |
| Program: quality ->  *Readability* | 0-9 | 10-14 | 15-19 | 20 |
|  |  |  |  |
| Program: quality ->  *Modularity* | 0-9 | 10-14 | 15-19 | 20 |
|  |  |  |  |
| Program: quality ->  *Simplicity* | 0-9 | 10-14 | 15-19 | 20 |
|  |  |  |  |
| Total Grade (100) |  | | | |

**Problems:**

Given the following design (next page), implement it in Java. Note:

1. You may add more attributes or operations to a class if necessary. Specifically, you may use meaningful operations for FBI\_Agent and CIA\_Agent classes. Remember that your CIA\_Agent and FBI\_Agent class should implement runnable interface. Each agent object has its own thread for doing the assigned tasks.
2. Read textbook to see some example code snippets for the object pool pattern (pp. 170—174).
3. Some corrections:

private ObjectPool(ObjectCreation\_IF c, int max){

instanceCount=0;

creator=c;

maxInstances=max;

pool = new Object[maxInstances];

}

public static ObjectPool getPoolInstance(ObjectCreation\_IF c, int max){

if (poolInstance==null)

poolInstance = new ObjectPool(c, max);

return poolInstance;

}

1. To demonstrate how a limited number of agents are requested to process tasks, your testing code (FBIAgentApp or CIAAgentAPP) should create a pool of 5 agents to service 10 task requesters. Each agent should leave a unique foot prints while it is serving a requester.

**Solution:**

* First, remember to zip the src folder of your project and submit the zip file to the ungraded assignment named “Lab4CodeSubmission”. One submission from each team.
* Paste a screenshot of a run of your program here.
* Also paste all you source code here.
* Save this report in PDF, then **each student** needs to submit the pdf report to the graded assignment named “Lab4ReportSubmission”.



Output Screenshots

A picture containing text

Description automatically generatedA picture containing text

Description automatically generatedA picture containing calendar

Description automatically generated

A picture containing text

Description automatically generatedText

Description automatically generated with medium confidenceA picture containing text

Description automatically generated

Agent\_IF.java

**package** AgentDemo;

**public** **interface** Agent\_IF {

**public** **void** startTask();

**public** **void** stopTask();

**public** **void** setTaskID(**int** id);

}

TaskRequester.java

**package** AgentDemo;

**import** PoolPattern.ObjectPool;

**public** **class** TaskRequester **implements** Runnable {

/\*\*

\* Singlton object pool

\*/

**private** ObjectPool server;

/\*\*

\* constructor

\* **@param** p the Object pool that we are allowed to request service providers from

\*/

**public** TaskRequester(ObjectPool p) {

**this**.server = p;

}

/\*\*

\* On run we get an available agent from the service pool, then

\* the agent start working on a task for an amount of time.

\* When time is up the agent stops working on the task and the agent is returned to the service pool.

\*/

@Override

**public** **void** run() {

Agent\_IF agent;

**try** {

agent = (Agent\_IF) server.waitForObject();

agent.setTaskID(server.getNextTask());

agent.startTask();

**try** {

Thread.*sleep*(2000);

}**catch** (InterruptedException e) {

System.***out***.println(**this**.getClass().getName());

e.printStackTrace();

}

agent.stopTask();

server.release(agent);

} **catch** (InterruptedException e1) {

e1.printStackTrace();

}

}

}

CIA\_Agent\_Creator.java

**package** AgentDemo.CIA;

**import** PoolPattern.ObjectCreation\_IF;

**public** **class** CIA\_Agent\_Creator **implements** ObjectCreation\_IF {

/\*\*

\* array if special characters

\*/

**private** String[] footPrints = {"@", "#", "$", "\*", ".", "?"};

/\*\*

\* index

\*/

**private** **int** index = 0;

/\*\*

\* creates and returns a new CIA Agent with a unique footprint

\* **@return** the CIA Agent

\*/

@Override

**public** Object create() {

CIA\_Agent agent = **new** CIA\_Agent(**this**.footPrints[(index++)%footPrints.length]);

**new** Thread(agent).start();

**return** agent;

}

}

CIA\_Agent.java

**package** AgentDemo.CIA;

**import** AgentDemo.Agent\_IF;

**public** **class** CIA\_Agent **implements** Runnable, Agent\_IF {

/\*\*

\* is the agent currently working

\*/

**private** **boolean** workingInProgress;

/\*\*

\* foot print based of special character and instance number

\*/

**private** String myFootPrint;

/\*\*

\* not sure what this is for

\*/

**private** **int** taskID = -1;

/\*\*

\* constructor

\* **@param** footPrint special character used to help distinguish agent footprint

\*/

**public** CIA\_Agent(String footPrint) {

**int** at = **this**.toString().indexOf("@")+1;

String intanceID = **this**.toString().substring(at);

**this**.myFootPrint = String.*format*("CIA - %s%s%s%s%s", footPrint, footPrint, intanceID, footPrint, footPrint);

}

/\*\*

\* start the process of working on a task

\*/

@Override

**public** **void** startTask() {

System.***out***.printf("%-15s -> %10s %d\n", myFootPrint, "Start Task", taskID);

**this**.workingInProgress = **true**;

}

/\*\*

\* finish up working on a task

\*/

@Override

**public** **void** stopTask() {

**this**.workingInProgress = **false**;

System.***out***.printf("%-15s -> %10s %d\n", myFootPrint, "End Task", taskID);

**this**.taskID = -1;

}

/\*\*

\* give the agent a dedicated task

\* **@param** the task id that the agent is to work on

\*/

@Override

**public** **void** setTaskID(**int** id) {

**this**.taskID = id;

}

/\*\*

\* while running the agent process a task if they have one.

\*/

@Override

**public** **void** run() {

**while**(**true**) {

**try** {

**if**(workingInProgress) {

processing();

Thread.*sleep*(100);

}**else** {

Thread.*sleep*(500);

}

} **catch** (InterruptedException e) {

System.***out***.println(**this**.getClass().getName());

e.printStackTrace();

}

}

}

/\*\*

\* prints a message of what is bing processed

\*/

**private** **void** processing() {

System.***out***.printf("%20s is working on task %d\n", myFootPrint, taskID);

}

}

CIAAgentApp.java

**package** AgentDemo.CIA;

**import** AgentDemo.TaskRequester;

**import** PoolPattern.ObjectPool;

**public** **class** CIAAgentApp {

**public** **static** **void** main(String[] args) {

ObjectPool server = ObjectPool.*getPoolInstance*(**new** CIA\_Agent\_Creator(), 5);

**for**(**int** i = 0; i < 10; i++) {

Thread client = **new** Thread(**new** TaskRequester(server));

client.start();

}

}

}

FBI\_Agent\_Creator.java

**package** AgentDemo.FBI;

**import** PoolPattern.ObjectCreation\_IF;

**public** **class** FBI\_Agent\_Creator **implements** ObjectCreation\_IF {

/\*\*

\* array if special characters

\*/

**private** String[] footPrints = {"@", "#", "$", "\*", ".", "?"};

/\*\*

\* index

\*/

**private** **int** index = 0;

/\*\*

\* creates and returns a new FBI Agent with a unique footprint

\* **@return** the FBI Agent

\*/

@Override

**public** Object create() {

FBI\_Agent agent = **new** FBI\_Agent(**this**.footPrints[(index++)%footPrints.length]);

**new** Thread(agent).start();

**return** agent;

}

}

FBI\_Agent.java

**package** AgentDemo.FBI;

**import** AgentDemo.Agent\_IF;

**public** **class** FBI\_Agent **implements** Runnable, Agent\_IF {

/\*\*

\* is the agent currently working

\*/

**private** **boolean** workingInProgress;

/\*\*

\* foot print based of special character and instance number

\*/

**private** String myFootPrint;

/\*\*

\* not sure what this is for

\*/

**private** **int** taskID = -1;

/\*\*

\* constructor

\* **@param** footPrint special character used to help distinguish agent footprint

\*/

**public** FBI\_Agent(String footPrint) {

**int** at = **this**.toString().indexOf("@")+1;

String intanceID = **this**.toString().substring(at);

**this**.myFootPrint = String.*format*("FBI - %s%s%s%s%s", footPrint, footPrint, intanceID, footPrint, footPrint);

}

/\*\*

\* start the process of working on a task

\*/

@Override

**public** **void** startTask() {

System.***out***.printf("%-15s -> %10s %d\n", myFootPrint, "Start Task", taskID);

**this**.workingInProgress = **true**;

}

/\*\*

\* finish up working on a task

\*/

@Override

**public** **void** stopTask() {

**this**.workingInProgress = **false**;

System.***out***.printf("%-15s -> %10s %d\n", myFootPrint, "End Task", taskID);

**this**.taskID = -1;

}

/\*\*

\* give the agent a dedicated task

\* **@param** the task id that the agent is to work on

\*/

@Override

**public** **void** setTaskID(**int** id) {

**this**.taskID = id;

}

/\*\*

\* while running the agent process a task if they have one.

\*/

@Override

**public** **void** run() {

**while**(**true**) {

**try** {

**if**(workingInProgress) {

processing();

Thread.*sleep*(100);

}**else** {

Thread.*sleep*(500);

}

} **catch** (InterruptedException e) {

System.***out***.println(**this**.getClass().getName());

e.printStackTrace();

}

}

}

/\*\*

\* prints a message of what is bing processed

\*/

**private** **void** processing() {

System.***out***.printf("%20s is working on task %d\n", myFootPrint, taskID);

}

}

FBIAgentApp.java

**package** AgentDemo.FBI;

**import** AgentDemo.TaskRequester;

**import** PoolPattern.ObjectPool;

**public** **class** FBIAgentApp {

**public** **static** **void** main(String[] args) {

ObjectPool server = ObjectPool.*getPoolInstance*(**new** FBI\_Agent\_Creator(), 5);

**for**(**int** i = 0; i < 10; i++) {

Thread client = **new** Thread(**new** TaskRequester(server));

client.start();

}

}

}

ObjectCreation\_IF.java

**package** PoolPattern;

**public** **interface** ObjectCreation\_IF {

**public** Object create();

}

ObjectPool\_IF.java

**package** PoolPattern;

**public** **interface** ObjectPool\_IF {

**public** **int** getSize();

**public** **int** getCapacity();

**public** **void** setCapacity(**int** c);

**public** Object getObject();

**public** Object waitForObject() **throws** InterruptedException;

**public** **void** release(Object o);

**public** **int** getNextTask();

}

ObjectPool.java

**package** PoolPattern;

**public** **class** ObjectPool **implements** ObjectPool\_IF {

**private** **static** Object *lockObject* = **new** Object();

/\*\*

\* the number of free objects

\*/

**private** **int** size;

/\*\*

\* the number of objects that have been created

\*/

**private** **int** instanceCount;

/\*\*

\* the maximum number of objects that may be created

\*/

**private** **int** maxInstances;

/\*\*

\* the pool of objects

\*/

**private** Object[] pool;

/\*\*

\* singleton ObjectPool

\*/

**private** **static** ObjectPool *poolInstance* = **null**;

/\*\*

\* the Object creator

\*/

**private** ObjectCreation\_IF creator;

/\*\*

\* counts all the tasks

\*/

**private** **int** taskCounter = 0;

/\*\*

\* constructor

\* **@param** c

\* **@param** max

\*/

**private** ObjectPool(ObjectCreation\_IF c, **int** max) {

**this**.creator = c;

**this**.maxInstances = max;

**this**.size = 0;

**this**.instanceCount = 0;

**this**.pool = **new** Object[maxInstances];

}

/\*\*

\* get a object pool

\* **@param** c

\* **@param** max

\* **@return** the instance of the ObjectPool

\*/

**public** **static** ObjectPool getPoolInstance(ObjectCreation\_IF c, **int** max) {

**synchronized**(*lockObject*){

**if**(*poolInstance* == **null**) {

*poolInstance* = **new** ObjectPool(c, max);

}

}

**return** *poolInstance*;

}

/\*\*

\* **@return** size - the number of free objects

\*/

@Override

**public** **int** getSize() {

**return** **this**.size;

}

/\*\*

\* **@return** capacity - the total number of objects

\*/

@Override

**public** **int** getCapacity() {

**return** pool.length;

}

/\*\*

\* set the total number of objects that make up the object pool

\* **@param** c the new capacity

\* copied from book P.172

\*/

@Override

**public** **void** setCapacity(**int** c) {

**if**(c != pool.length) {

**if**(c <= 0) {

String msg = "Capacity must be greater than zero.\n\tValue Entered:\t" + c;

**throw** **new** IllegalArgumentException(msg);

}

**synchronized**(*lockObject*){

**this**.maxInstances = c;

Object[] newPool = **new** Object[maxInstances];

System.*arraycopy*(pool, 0, newPool, 0, maxInstances);

pool = newPool;

}

}

}

/\*\*

\* get an object from the object pool

\* **@return** the object

\* copied from book P.172

\*/

@Override

**public** Object getObject() {

**synchronized**(*lockObject*){

**if**(size > 0) {

**return** removeObject();

}**else** **if**(instanceCount < maxInstances) {

**return** createObject();

}**else** {

**return** **null**;

}

}

}

/\*\*

\* get an object from the object pool when it becomes available

\* **@return** the object

\* copied from book P.173

\*/

@Override

**public** Object waitForObject() **throws** InterruptedException{

**synchronized**(*lockObject*){

**if**(size > 0) {

**return** removeObject();

}**else** **if**(instanceCount < maxInstances) {

**return** createObject();

}**else** {

**do** {

*lockObject*.wait();

}**while**(size <= 0);

**return** removeObject();

}

}

}

/\*\*

\* return an object to the object pool

\* **@param** o - the object to be placed back in the object pool

\* copied from book P.173-174

\*/

@Override

**public** **void** release(Object o) {

**if**(o == **null**) {

**throw** **new** NullPointerException();

}

**synchronized**(*lockObject*){

**if**(size < getCapacity()) {

pool[size] = o;

size++;

*lockObject*.notify();

}

}

}

/\*\*

\* remove an object from the object pool

\* **@return** the object that has been removed

\* copied from book P.173

\*/

**private** Object removeObject() {

size--;

**return** pool[size];

}

/\*\*

\* make a new object

\* **@return** the new object

\*/

**private** Object createObject() {

instanceCount++;

**return** creator.create();

}

/\*\*

\* get the next task number

\* **@return** the next task number

\*/

@Override

**public** **int** getNextTask() {

**return** ++taskCounter;

}

}