Parallel and MultiThreaded Programming

CSYE 7215

Homework 5

Due: October 18, 2020

Put all your java, compiled class files and documentation into zip file named Homework5.zip and submit it via the dropbox on canvas before the END of due date. Put your name on all .java files. There will be a short quiz on this assignment.

1. Explain:

What are possible ways Java objects become subject to garbage collection?

give example code for each case.

JVM Garbage Collector will is process all unused/unreferenced memory automatically at runtime .

There are multiple ways to unreferenced the java object

1. nulling the reference

example

test e=new test();

e=null;

1. assigning a reference to another

Test e1=new Test ();

Test e2=new Test ();

e1=e2;//now the first object referred by e1 is available for garbage collection

1. anonymous object

new test();

**What is Starvation, what is the remedy for starvation?**

* Starvation describes a situation where a thread is **unable to gain regular access to shared resources and is unable to make progress.** This happens when shared resources are made **unavailable for long periods by "greedy" threads.**
* For example, suppose an object provides a synchronized method that often takes a long time to return. If one thread invokes this method frequently, other threads that also need frequent synchronized access to the same object will often be blocked.
* While it is not possible to implement 100% fairness in Java we can still implement our synchronization constructs to increase fairness between threads.
* If we are using synchronized blocked make sure we use wait method for if the thread is running for long time. wait method will release the lock and allow other thread to work on critical section.
* use explicit lock and guard the critical section only

**What is Deadlock and Race Condition?**

**Deadlock** - Deadlock describes a situation where two or more

threads are blocked forever, waiting for each other

Example

* In Above example process p1 try to acquire lock on resource 1 but resource 1 is already locked by process p2 and process p2 try to acquire lock on resource r2 but it already locked by process p1. Now both the processes p1 and p2 are interdependent. This kind of situation known as Deadlock.
* **Race Condition**: A **race condition** occurs when two or more threads can access shared data and they try to change it at the same time. Execution order of threads are dependent on Thread scheduling algorithm, you don’t know the order in which the threads will attempt to access the shared data. Therefore, the result of the change in data is dependent on the thread scheduling algorithm, i.e. both threads are “racing” to access/change the data

**What is String Pool?**

* String Pool is a storage area in Java heap. The JVM performs some steps while initializing string literals to increase performance and decrease memory overhead. To decrease the number of String objects created in the JVM, the String class keeps a pool of strings.
* Each time a string literal is created, the JVM checks the string literal pool first. If the string already exists in the string pool, a reference to the pooled instance returns. If the string does not exist in the pool, a new String object initializes and is placed in the pool

Example

String s1 = “Hello”

String s2 = “Hello”

String s3 = “Test”

String s4 = new String(“test2”)

Heap Area

|  |
| --- |
| Hello |
| test |
|  |

S4

**What is Constant Pool?**

- Constant pool is a part of .class file (and its in-memory representation) that contains constants needed to run the code of that class.

- The constant pool is organized as an array of variable-length elements

- Each constant occupies one element in the array.

- Throughout the class file, constants are referred to by the integer index that indicates their position in the array.

- The initial constant has an index of one, the second constant has an index of two, etc.

- These constants include literals specified by the programmer and symbolic references generated by compiler.

**2. Provide four numbers (byte, short, int, long) examples. Show the results for Signed and Un-signed arithmetic. In each case, in binary show the base, position, and value in that position.**

* Signed variables, such as signed integers will allow you to represent numbers both in the positive and negative ranges.
* if the 1st bit is 1 that means number is negative else

positive

* Unsigned variables, such as unsigned integers, will only allow you to represent numbers in the positive.

Example of byte Unsigned

**10000001 = 1\*2^7+0\*2^6+……..+1\*2^0 =129**

Example of byte signed

First digit from left indicates the value is positive or negative

**10000001 = 0\*2^6+0\*2^5+………………+1\*2^0 =1 \* -1 = -1**

for short unsigned

1000000000000001 =

**(1 × 2^15 + 0 × 2^14 + ….+ 0 × 2^1 + 1 × 2^0)(10) = 32 769(10)**

for short signed

1000000000000001 =

**(0 × 2^14 + … + 1 × 2^0)(10) = -1**

for int unsigned

10000000000000000000000000000001 =

**(1× 2^31 + … + 1 × 2^0)(10) = 2147483649(10)**

for int signed

10000000000000000000000000000001 =

**(0 × 2^30 + … + 1 × 2^0)(10) = -1**

for long unsigned

1000000000000000000000000000000000000000000000000000000000000001 =

**(1× 2^63 + … + 1 × 2^0)(10) = 9223372036854775809(10)**

for int signed

1000000000000000000000000000000000000000000000000000000000000001 =

**(0 × 2^62 + … + 1 × 2^0)(10) = -1**

*\*\* (10) means 10 base.*

3. Consider Class File data structure, Explain each case with an Example:

a) magic, b) constant\_pool, c) super\_class, d) interfaces, e) fields, f) methods, g) attributes

ClassFile {

u4 magic;

u2 minor\_version;

u2 major\_version;

u2 constant\_pool\_count;

cp\_info constant\_pool[constant\_pool\_count-1];

u2 access\_flags;

u2 this\_class;

u2 super\_class;

u2 interfaces\_count;

u2 interfaces[interfaces\_count];

u2 fields\_count;

field\_info fields[fields\_count];

u2 methods\_count;

method\_info methods[methods\_count];

u2 attributes\_count;

attribute\_info attributes[attributes\_count];

}

* The Java class file contains everything a JVM needs to know about one Java class or interface.
* In their order of appearance in the class file, the major - components are:
* magic
* version
* constant pool
* access flags
* this class
* super class
* interfaces
* fields
* methods
* attributes.

**u1 u2 u4 --> unsigned 1,2,4 bytes quantity**

* **Magic**

- so first four byte in every class is always

OXCAFEBABE

- This magic number makes Java class files easier to identify, because the odds are slim

(means very less chance) that non-class files would start with the same initial four bytes

- second 4 bytes contains the major and minor version of class file

- JVM has maximum number it can load it number is higher than that then JVM will reject that class file.

* **Constant pool**

- Class file stores the constants associated with class or interfaces.

- The constant pool is organized as an array of variable-length elements. Each constant occupies one element in the array.

- Throughout the class file, constants are referred to by the integer index that indicates their position in the array.

- The initial constant has an index of one, the second constant has an index of two, etc.

- Each element of the constant pool starts with a one-byte tag specifying the type of constant at that position in the array.

* **Super class**
* The super class component, another two-byte index into the constant pool.
* Constant\_pool[super\_class] is a CONSTANT\_Class element that points to the name of the super class from which this class descends.
* **Interfaces**
* The interfaces component starts with a two-byte count of the **number of interfaces implemented by the class (or interface) defined in the file.**
* Immediately following is an array that contains one index into the constant pool for each interface implemented by the class. Each interface is represented by a CONSTANT\_Class element in the constant pool that points to the name of the interface.
* **Fields**
* The fields component starts with a two-byte count of the number of fields in this class or interface
* A field is an instance or class variable of the class or interface.
* Following the count is an array of variable-length structures, one for each field. Each structure reveals information about one field such as the field's name, type, and, if it is a final variable, its constant value. Some information is contained in the structure itself, and some is contained in constant pool locations pointed to by the structure
* The only fields that appear in the list are those that were declared by the class or interface defined in the file; no fields inherited from super classes or superinterfaces appear in the list
* **Methods**

- The methods component starts with a two-byte count of the number of methods in the class or interface.

- This count includes only those methods that are explicitly defined by this class, not any methods that may be inherited from superclasses, Following the method count are the methods themselves.

* **Attributes**
* It returns the number of attributes (instance variables) present in current class file.
* For example, one attribute is the source code attribute; it reveals the **name of the source file from which this class file was compiled**.

4. Suppose you have a two dimensional array input data;

int[][] arr = { { 9, 12, 6, 14, 10, 21, 13}, { 3, 5, 41, 16, 14, 10, 21},

{ 3, 15, 41, 17, 11, 10, 51}, { 3, 15, 41, 17, 11, 10, 51}

{ 4, 15, 35, 17, 11, 12, 55}, { 2, 16, 31, 18, 12, 11, 42} };

Write Java program: a) Create six Threads where each thread-id corresponds to a row

in array input data, for example, (tid1, row1), (tid2, row2), (tid3, row3), (tid4, row4), (tid5, row5), (tid6, row6), b) Write code for each thread to sort its row of array data us-ing sort method in Java Collections library, c) Write code to update array data with sorted values, d) Sort all rows in array data using HeapSort, e) Write code for each thread to print the sorted data. Notes: You need to protect Array data. How do you protect array data? First you sort using Collections class, and then for Heapsort. How lock mechanism works in (b) and (d)?

Approach :

* Created separate runnable class ArrayData to perform all type of sorting.
* Created six threads for each rows and perform sorting at row level

Inside the synchronized block to protect the array

* We can create all threads using **loop** or **ThreadPoolExecutor**
* once the row level sorting gets completed perform the heapsort for entire 2d array using PriorityQueue
* To check code implementation please check source code.

Output :

Text

Description automatically generated Graphical user interface, text

Description automatically generated

5. The following link provides an example of user defined class loader called CCLoad-er:

https://www.journaldev.com/349/java-classloader

a) Analyze the code and Explain as how it works.

b) Compile and run CCLoader. What are the outputs?

c) Add Student class defined in Homework2. Build CCLoader, Compile

and Run, Explain outputs.

**a) Analyze the code and Explain as how it works.**

**CCLoader Class**

- In Given example We are creating a custom class loader by extending ClassLoader class and override the loadClass() method and check the class name start with our local class file then load that class using custom class loader.

- We have also created the getClass() method which

will determined the full class name and then call the loadClassFileData() data method which will return the bytcode from the class file.

- After that we are calling defineClass() method of classloader class and reolve the class references using resolveClass() method.

CCRun Class

- In CCRun Class we are accepting the all the command line argument and processing each class with custom class Loader

- Initailly we are creating the object of custom class loader

// This will call the constructor of super class(ClassLoader)

CCLoader ccl = new CCLoader(CCRun.class.getClassLoader());

* Then we are calling the load class method which will do following things
* Check if the class name of the file if the class is local class then use custom class loader and call the getClass method.
* getclass() method will get the binary class file then call defineClass() method which is inherited from the ClassLoader class and at last resolve the class reference.

**b) Compile and run CCLoader. What are the outputs?**

Text

Description automatically generated

output shows that class name with their class Loader

for Example

**Bar ClassLoader: CCLoader@71f6f0bf**

**Foo ClassLoader: CCLoader@71f6f0bf**

Foo and Bar class load by our custom class loader.

**c) Add Student class defined in Homework2. Build CCLoader, Compile**

**Student class output:**

**A picture containing graphical user interface, text

Description automatically generated**

Explanation:

* Using custom class loader I have load the Grade Driver class of assignment 2 which h includes the student , util and grader class.
* In output screen we can see that custom class loader is loading all the depended classes.
* At the end of the code to verify the class is load successfully or not we have called the main method and the output shows that it is working fine.

6. In Homework4, you created 50 Student threads and one Grader thread and man-aged the concurrency using Explicit Locks. In this problem, consider creating threads using ThreadPoolExecutor. How would you design and solve (problem-4 in home-work4) using ThreadPoolExecutor? a) Show your design, b)Write code, compile and run.

Note: <https://howtodoinjava.com/java/multi-threading/java-thread-pool-executor-example/>

Approach :

* I have used the ThreadPoolExecutor class to create the 50 Student thread
* Assigned Student object to each Thread of ThreadPool executer

And execute the threadpool executor.

* I have implemented explicit locking to protect the file and hashmap.

Output :

A screenshot of a computer

Description automatically generated

Text

Description automatically generated

A picture containing text

Description automatically generated

Text

Description automatically generated

7. Define Student class with instance variables name, id, homework, midterm, and fi-nal-exam. The name is a string whereas others are all integers. Add a static variable nextId which is integer and statically initialized to 1. In each of them, the id should be assigned to the next available id given by nextId. The default constructor should set the name of the student object to “StudentX” where X is the next id.

Create 30 Callable Student Threads, each to be identified with string name-Thread-nextId that stores their average grade scores into ScoresHashmap, and also returns the thread string name after 1 second. What should the map hold as key/value? How do you protect the map?

Create Callable GraderThread to read grade score from ScoresHashmap and calcu-late the Final grade using letter grades A,B,C,D,E,F from ScoresHashmap (you need to build score range for each letter grade) and then send the final grade to each student. And also stores the final grade for All students in FinalGrades file. Do you need to pro-tect the map? What method do you use to send final grades to students?

Create FutureTask thread to report the final grades for all students stored in Final-Grades file by GraderThread. Compile and Run all code.

Approach:

* Created callable Student and grader class.
* Student class call method will return the student object without final grade.
* Grader class call method will return the hashmap which contains all the student thread and their final grade detail
* Created one future task to hold the output of Grader thread and later on pass that output value to student future task and set the Final grader of each student.
* For detail implementation please review code file

Output:

A screenshot of a computer

Description automatically generated

Text

Description automatically generated

Graphical user interface, text

Description automatically generated

A screenshot of a computer

Description automatically generated

8. A deadlock is when two or more threads are blocked waiting to obtain locks that some of the other threads in the deadlock are holding. Deadlock can occur when mul-tiple threads need the same locks, at the same time, but obtain them in different order. For instance, if thread1 locks A, and tries to lock B, and thread2 has already locked B, and tries to lock A, a deadlock arises. Thread1 can never get B, and thread2 can never get A. In addition, neither of them will ever know. They will remain blocked on each of their object, A and B, forever. This situation is a deadlock.

Thread-1 locks A, waits for B

Thread-2 locks B, waits for A

http://tutorials.jenkov.com/java-concurrency/deadlock.html

A) Explain As why this code Deadlocks?

B) Compile and Run this code.

C) Is Race Condition possible in this code, Yes/No, Why?

public class TreeNode {

TreeNode parent = null;

List children = new ArrayList();

public synchronized void addChild(TreeNode child){

if(!this.children.contains(child)) {

this.children.add(child);

child.setParentOnly(this);

}

}

public synchronized void addChildOnly(TreeNode child){

if(!this.children.contains(child){

this.children.add(child);

}

}

public synchronized void setParent(TreeNode parent){

this.parent = parent;

parent.addChildOnly(this);

}

public synchronized void setParentOnly(TreeNode parent){

this.parent = parent;

}

}

1. **Explain As why this code Deadlocks?**

Possible Deadlock scenario :

Thread 1: parent.addChild(child); //locks parent

--> child.setParentOnly(parent);

Thread 2: child.setParent(parent); //locks child

--> parent.addChildOnly()

* First thread 1 calls parent.addChild(child). Since addChild() is synchronized thread 1 effectively locks the parent object for access from other treads.
* Then thread 2 calls child.setParent(parent). Since setParent() is synchronized thread 2 effectively locks the child object for access from other threads.

**B) Compile and Run this code.**

Graphical user interface, text, application

Description automatically generated

**C) Is Race Condition possible in this code, Yes/No, Why?**

As all the methods are **synchronized race condition is not possible here.**