

# Assignment cover sheet

Complete all sections of this coversheet

|  |  |
| --- | --- |
| Student name | Student number |
| Mohamed Abuklal | 5902514 |

|  |  |
| --- | --- |
| Subject number and name: | Real-Time Embedded Systems ECTE331 |
| Subject coordinator: | Dr Abdsamad binkrid |
| Title of Assignment: | Project Problems (Part B) |
| Date and time due: | 18/06/2024 |
| Total number of pages: | 5 |

Student declaration and acknowledgement (must be read by all students)

By submitting this assignment online, the submitting student declares on behalf of the team that:

1. All team members have read the subject outline for this subject, and this assessment item meets the requirements of the subject detailed therein.
2. This assessment is entirely our own work, except where we have included fully documented references to the work of others. The material contained in this assessment item has not previously been submitted for assessment.
3. Acknowledgement of source information is in accordance with the guidelines or referencing style specified in the subject outline.
4. All team members are aware of the late submission policy and penalty.
5. The submitting student undertakes to communicate all feedback with the other team members.

Place compressed photo of your structure or team here (optional)

**Part B**

**Code**

**package** project;

**import** java.util.concurrent.CountDownLatch;

**public** **class** ThreadSyncExample {

**private** **static** CountDownLatch *latchA1B2* = **new** CountDownLatch(1);

**private** **static** CountDownLatch *latchB2A2* = **new** CountDownLatch(1);

**private** **static** CountDownLatch *latchA2B3* = **new** CountDownLatch(1);

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

// Creating two threads A and B for the shared variables:

Thread thread\_A = **new** Thread(() -> {

*FuncA1*(); // Thread FuncA1 Started

*latchA1B2*.countDown(); // Thread FuncA1 completed

**try** {

*latchB2A2*.await(); // Waiting for FuncB2 to be completed

} **catch** (InterruptedException e) {

e.printStackTrace();

}

*FuncA2*(); // Thread FuncA2 Started

*latchA2B3*.countDown(); // Thread FuncA2 completed

**try** {

*latchA2B3*.await(); // Waiting for FuncB3 to be completed

} **catch** (InterruptedException e) {

e.printStackTrace();

}

*FuncA3*(); // Thread FuncA3 Started

});

Thread thread\_B = **new** Thread(() -> {

**try** {

*latchA1B2*.await(); // Waiting for FuncA1 to be completed

} **catch** (InterruptedException e) {

e.printStackTrace();

}

*FuncB1*(); // Thread FuncB1 Started

*FuncB2*(); // Thread FuncB2 Started

*latchB2A2*.countDown(); // Thread FuncB2 completed

**try** {

*latchA2B3*.await(); // Wait for FuncA2 to be completed

} **catch** (InterruptedException e) {

e.printStackTrace();

}

*FuncB3*(); // Thread FuncB3 Started

});

// threads A and B Starts:

thread\_A.start();

thread\_B.start();

// Waiting for both threads to be completed:

**try** {

thread\_A.join();

thread\_B.join();

} **catch** (InterruptedException e) {

e.printStackTrace();

}

// Printing the final values of shared variables:

System.***out***.println("Latest values are:");

System.***out***.println("A1 = " + *A1*);

System.***out***.println("A2 = " + *A2*);

System.***out***.println("A3 = " + *A3*);

System.***out***.println("B1 = " + *B1*);

System.***out***.println("B2 = " + *B2*);

System.***out***.println("B3 = " + *B3*);

}

// Defining the functions of the shared variables:

**private** **static** **int** *A1*, *A2*, *A3*, *B1*, *B2*, *B3*;

**private** **static** **void** FuncA1() {

*A1* = *sum*(0, 500);

}

**private** **static** **void** FuncB1() {

*B1* = *sum*(0, 250);

}

**private** **static** **void** FuncB2() {

*B2* = *A1* + *sum*(0, 200);

}

**private** **static** **void** FuncA2() {

*A2* = *B2* + *sum*(0, 300);

}

**private** **static** **void** FuncB3() {

*B3* = *A2* + *sum*(0, 400);

}

**private** **static** **void** FuncA3() {

*A3* = *B3* + *sum*(0, 400);

}

// calculating the sum from the start to the end:

**private** **static** **int** sum(**int** start, **int** end) {

**int** sum = 0;

**for** (**int** i = start; i <= end; i++) {

sum += i;

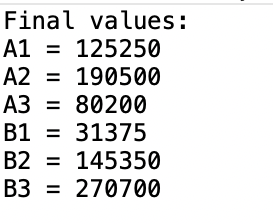
}

**return** sum;

}

}

**Output**



**Discussion**

The above Java program shows how to manage the execution order of functions across two threads, thread\_A and thread\_B, using CountDownLatch for thread synchronization. The latches make sure that certain functions in one thread wait for the completion of corresponding functions in the other thread while each thread calculates on shared variables (A1, A2, A3, B1, B2, B3). To ensure that the functions are called in the right order and depend on each other, thread\_A, for example, executes FuncA1, signals its completion, and then waits for thread\_B to finish FuncB2 before starting FuncA2.

By regulating the order of execution, this synchronization mechanism effectively prevents race conditions and guarantees thread safety. As difficulties that threads must get through, the CountDownLatch objects make sure that FuncB2 waits for FuncA1 to finish, FuncA2 waits for FuncB2, and so on. The program's coordinated execution and accurate updating of shared state appears when the main thread joins the two threads after they have finished their individual tasks and prints the final values of the shared variables.