(English below)

3. Zh – Fesztivál Szimulátor

A feladat egy fesztivál szimulálása – néhány ember elmegy a koncertekre, amik érdeklik őket, majd buliznak egy kicsit, és mennek haza.

A feladat három részből áll; alapfeladat/első rész (20 pont), második rész (8 pont), illetve harmadik rész (12 pont). **A második és harmadik rész kizárja egymást részben**, így a második részre vonatkozó részeket érdemes kikommentezni, vagy átemelni egy másik függvénybe, ha a hallgató mindkettőt teljesíti.

A kód négy osztályból áll; **FestivalSimulator**, itt van a *main(),* **Attendee**, ami egy fesztiválozót reprezentál, **Concert**, ami egy koncertet reprezentál, illetve **Bus**, ami a buszt reprezentálja, ami a harmadik részben hazaviszi a fesztiválozókat.

A kód alapfelépítését ne változtassuk, de nyugodtan ki lehet egészíteni segédfüggvényekkel és változókkal.

Alapfeladat/első rész (20 pont):

- **FestivalSimulator:**
  - Hívjuk meg a *setup()* függvényt a *main()*-ben
  - Indítsuk el a fesztiválozók *waitForConcert()*, illetve a koncertek *start()* metódusait egy-egy külön szálon
  - A **CONCERTS_DURATION_MSEC** változó segítségével várjunk annyi időt, amíg vége a koncerteknek
  - Miután vége a koncerteknek, ***maximum*** **PARTY_DURATION_MSEC** idő elteltével állítsuk le a szimulációt
- **Attendee:**
  - Implementáljuk a *waitForConcert()* metódust; addig várakozik a fesztiválozó, amíg a koncert elindul, majd meghívja a *goToConcert()* metódust
  - Implementáljuk a *goToConcert()* metódust; addig várakozik a fesztiválozó, amíg a koncertnek vége
- **Concert:**
  - Hozzunk létre egy adatszerkezetet a koncerten résztvevő **Attendee**-k számára
  - Implementáljuk a *start()* metódust; először **startTime**-ot várunk, majd értesítjük a résztvevőket, hogy elindult a koncert. Ezek után várjunk **duration**-t, majd értesítsük a résztvevőket, hogy vége a koncertnek
  - Implementáljuk az *addAttendee()* metódust; Adjuk hozza a paraméterben kapott **Attendee**-t az imént létrehozott adatszerkezethez

Második rész (8 pont):

- **FestivalSimulator:**
  - Miután vártunk **CONCERTS_DURATION_MSEC**-et, adjuk a fesztiválozók tudtára, hogy vége a fesztiválnak

- **Attendee:**
    - A *goToConcert()* metódus végén hívjuk meg a *goToParty()* metódust (miután vége a koncertnek)
    - Implementáljuk a *goToParty()* metódust; a kódban jelzett kiíratás után **WAIT_TIME_MSEC**-enként ellenőrizzük le, hogy tart-e még a fesztivál. Amennyiben vége a fesztiválnak, lépjünk ki a metódusból.

A részfeladat megoldásához ajánlott új változókat/függvényeket felvenni, hogy minél egyszerűbben meg lehessen oldani

Harmadik rész (12 pont):

A második és harmadik rész egyes szakaszai, konkrétan a *goToParty()* metódus implementálása, részben kizárja egymást. Amennyiben mindkettő teljesítve lett, érdemes a második rész kódját átemelni egy másik függvénybe, vagy kikommentezni, jelezve, hogy az a második részhez tartozik.

- **FestivalSimulator:**
    - Miután vártunk **CONCERTS_DURATION_MSEC**-et, indítsuk el a buszt egy külön szálon – a **Bus** osztály singleton osztály, így következőképp tudjuk használni: *Bus.getInstance()*
- **Attendee:**
    - A *goToConcert()* metódus végén hívjuk meg a *goToParty()* metódust (miután vége a koncertnek)
    - Implementáljuk a *goToParty()* metódust; a kódban jelzett kiíratás után a **Bus** osztály *tryToGetOnBus* metódusa segítségével próbáljon a fesztiválozó felszállni a buszra. Amennyiben ez nem sikerült, várjon **WAIT_TIME_MSEC**-et, és próbálkozzon újból. Ha fel tudott szállni a buszra, lépjünk ki a metódusból.
- **Bus:**
    - Hozzunk létre egy adatszerkezetet arra, hogy nyilván tartsuk az utasokat – az adatszerkezetnek **LIMIT**-nyi kapacitása kell, hogy legyen, ennél több utas ne utazhasson egyszerre
    - Hozzunk létre egy változót arra, hogy nyilván tartsuk, mikor fogad új utasokat a busz, és mikor nem
    - Implementáljuk a *startBus()* metódust;
        - A kódban jelzettek szerint írjuk ki, hogy jelenleg fogad utasokat
        - Az erre a célra létrehozott változó segítségével jelezzük, hogy a busz jelenleg fogad utasokat
        - Várjunk **WAIT_TIME_MSEC**-et
        - Írjuk ki a kódban jelzettek szerint, hogy a busz elindul a jelenlegi utasokkal
        - Az erre a célra létrehozott változó segítségével jelezzük, hogy a busz jelenleg **nem** fogad új utasokat
        - Várjunk **TRIP_TIME_MSEC**-et

        - Amennyiben a buszon nem volt utas, lépjünk ki a ciklusból. Ha voltak, akkor szállítsuk le őket (vegyük ki őket az adatszerkezetből), majd írjuk ki az üzenetet a kódban jelzettek szerint
    - Implementáljuk a *tryToGetOnBus(Attendee attendee)* metódust;
        - Amennyiben a busz jelenleg nem fogad új utasokat, írjuk ki a kódban jelzett üzenetet, majd térjünk vissza false-szal.
        - Ha az utas fel tud szállni, írjuk ki a kódban jelzett üzenetet, majd térjünk vissza true-val
        - Ha az utas nem tud felszállni (tehát már tele van a busz), írjuk ki a kódban jelzett üzenetet, majd térjünk vissza false-szal

Exam 3 – Festival Simulator

The task is to simulate a festival – a couple of people attend concerts they want to visit, then party a bit, and after they go home.

The exam is made of 3 parts; base/first part (20 points), second part (8 points), and a third part (12 points). **The second and third part has some overlaps**, so if the student wants to complete both parts, they should comment out their solution of the second part, or put it into a different function, for clarity.

The code has four classes; **FestivalSimulator**, which has the *main()* method, **Attendee**, which represents a person going to the festival, **Concert**, which represents a concert, and the **Bus**, which represents the bust that takes the attendees home in the third part of the exam.

The structure of the code should not be changed, but students should feel free to create any kind of variables and/or functions that may help them complete the tasks.

Base/first part (20 points):

- **FestivalSimulator:**
  - Invoke the *setup()* function in the *main()* function
  - Start the attendees' *waitForConcert()* and the concerts' *start()* methods on different threads
  - Using the **CONCERTS_DURATION_MSEC** constant wait until the concerts are finished
  - After the concerts are finished, wait **PARTY_DURATION_MSEC** time ***at most*** and stop the simulation
- **Attendee:**
  - Implement the *waitForConcert()* method; the attendee waits until the concert start, then invokes the *goToConcert()* method
  - Implement the *goToConcert()* method; the attendee waits until the concert is over
- **Concert:**
  - Create a collection for the **Attendee**s attending the concert
  - Implement the *start()* method; wait **startTime**, then tell the attendees that the concert has started. After that wait **duration** and let the attendees know that the concert is over
  - Implement the *addAttendee()* method; put the **Attendee** given in the parameter to the collection created earlier

Second part (8 points):

- **FestivalSimulator:**
  - After waiting **CONCERTS_DURATION_MSEC** let the attendees know that the festival is over
- **Attendee:**

- At the end of the *goToConcert()* method invoke the *goToParty()* method (after the concert is over)
- Implement the *goToParty()* method; after printing the message specified in the code, check every **WAIT_TIME_MSEC** whether the festival is over or not. If it's over, exit the function.

It's highly recommended to create new variables/functions to make this part easier to complete.

Third part (12 points):

Some parts of the second and third part of this exam overlap, mainly the implementation of the *goToParty()* method. If both are completed, it's recommended that the student moves the solution of the second part into another function, or comments it out, clarifying that that code belongs to the solution of the second part.

- **FestivalSimulator:**
  - After waiting **CONCERTS_DURATION_MSEC** start the bus on a separate thread – the **Bus** is a singleton, which means it can be used like this: *Bus.getInstance()*
- **Attendee:**
  - At the end of the *goToConcert()* method invoke the *goToParty()* method (after the concert is over)
  - Implement the *goToParty()* method; after printing the message specified in the code, the attendee should try to get on the bus via the **Bus**' *tryToGetOnBus* method. If it's unsuccessful, try again after waiting **WAIT_TIME_MSEC**. If the attempt was successful, exit the method.
- **Bus:**
  - Create a collection to keep track of the passengers – the collection should have **LIMIT** as capacity, meaning it can only hold that number of passengers at once
  - Create a variable to keep track whether the bus is taking on new passengers or not
  - Implement the *startBus()* method;
    - Print the message specified in the code to signal that the bus is taking on new passengers
    - Use the variable created for this purpose to reflect that the bus is taking on new passengers
    - Wait **WAIT_TIME_MSEC**
    - Print the message specified in the code to signal that the bus is taking off with the current passengers
    - Use the variable created for this purpose to reflect that the bus is **not** taking on new passengers
    - Wait **TRIP_TIME_MSEC**
    - If the bus had no passengers when taking off, exit the loop. If it had passengers, take them home (remove from the collection) and print the message specified in the code
  - Implement the *tryToGetOnBus(Attendee attendee)* method;

- If the bus currently isn’t taking new passengers, print the message specified in the code, then return false
- If the passenger can get onboard, print the message specified in the code, then return true
- If the passenger could not get onboard (meaning the bus is at capacity), print the message specified in the code, then return true