Exercise Playing with coroutines

The following program has two co-operating coroutines:

```
const val N = 100

class MainActivity : ComponentActivity() {
   class Account {
       private var amount: Double = 0.0

       suspend fun deposit(amount: Double) {
           val x = this.amount
           delay(1)     // simulates processing time
           this.amount = x + amount
       }
       fun saldo(): Double = amount
   }

   /* Approximate measurement of the given block's execution time */
   fun withTimeMeasurement(title:String, isActive:Boolean=true, code:() -> Unit) {
       if(!isActive) return
       val time = measureTimeMillis { code() }
       Log.i("MSU", "operation in '$title' took ${time} ms")
   }

   data class Saldos(val saldo1: Double, val saldo2: Double)
   fun bankProcess(account: Account): Saldos {
       var saldo1: Double = 0.0
       var saldo2: Double = 0.0

       /* we measure the execution time of one deposit task */
       withTimeMeasurement("Single coroutine deposit $N times") {
           runBlocking {
               launch {
                   for (i in 1..N)
                       account.deposit(0.0)
               }
           }
           saldo1 = account.saldo()
       }

       /* then we measure the execution time of two simultaneous deposit tasks using
coroutines */
       withTimeMeasurement("Two $N times deposit coroutines together", isActive = true) {
           runBlocking {
               launch {
                   for (i in 1..N) account.deposit(1.0)
               }
               launch {
                   for (i in 1..N) account.deposit(1.0)
               }
               saldo2 = account.saldo()
           }
       }

       return Saldos(saldo1, saldo2)
   }

   override fun onCreate(savedInstanceState: Bundle?) {
       super.onCreate(savedInstanceState)
```

```
        val results = bankProcess(Account())
        setContent {
            AndroidsensorlabsTheme {
                // A surface container using the 'background' color from the theme
                Surface(
                    modifier = Modifier.fillMaxSize(),
                    color = MaterialTheme.colors.background
                ) {
                    ShowResults(saldo1 = results.saldo1, saldo2 = results.saldo2)
                }
            }
        }
    }
}

@Composable
fun ShowResults(saldo1: Double, saldo2: Double) {
    Column {
        Text(text = "Saldo1: $saldo1")
        Text(text = "Saldo2: $saldo2")
    }
}
```

runBlocking {} is a coroutine function. By not providing any context, it will get run on the main thread. Runs a new coroutine and blocks the current thread interruptible until its completion. This function should not be used from a coroutine. It is designed to bridge regular blocking code to libraries that are written in suspending style, to be used in main functions and in tests. Note that it is not recommended to use in Android, because it will block the UI thread. In this case we use it only to allow easy execution time measurements.

launch {} is a coroutine builder. It launches a new coroutine concurrently with the rest of the code, which continues to work independently.

There is a problem in the given code. Two coroutines increase the value (saldo2, saldo1 should be 0) of the account by 100. Therefore the Saldo2 at the output should have the value of 200. But when you run the code, the value will be 0 or 100.

Fix the program, so that the final saldo of the account will be 200, and you are still using those two independent coroutines (and the deposit() operation reads the saldo to the variable x, delays 1ms, and the writes the incremented value back to the saldo). Do not use withContex() in your solution.

Hint 1: There are two separate errors in the program.

Hint 2: Mutex may be a useful tool.