UBIT: _____@buffalo.edu

Name: _____

CSE116 - Fall 2019

Level 3 Quiz

Q1. Actors [15 points]

```
case object Start
case class Value(n: Int)

class QuizActor(next: ActorRef) extends Actor {
  def receive: Receive = {
    case value: Value =>
      next ! Value(value.n - 2)
    case Start =>
      next ! Value(10)
  }
}

class QuizActor2() extends Actor {

  var storage: Int = 0

  def receive: Receive = {
    case value: Value =>
      this.storage += value.n
  }
}

object Q1 {

  def main(args: Array[String]): Unit = {
    val system = ActorSystem("QuizSystem")

    val mainActor = system.actorOf(Props(classOf[QuizActor2]))

    val a1 = system.actorOf(Props(classOf[QuizActor], mainActor))
    val a2 = system.actorOf(Props(classOf[QuizActor], a1))
    val a3 = system.actorOf(Props(classOf[QuizActor], a2))

    a3 ! Start
    a2 ! Value(5)
  }
}
```

When this program runs, what is the value the *storage* variable in *mainActor* after all messages are processed?

Continue on back

Q2. Web Sockets [20 points]

```
class QuizServer() {

  var values: Map[SocketIOClient, Int] = Map()

  val config: Configuration = new Configuration {
    setHostname("localhost")
    setPort(8080)
  }
  val server: SocketIOServer = new SocketIOServer(config)

  server.addConnectListener(new ConnectionListener(this))
  server.addDisconnectListener(new DisconnectionListener(this))
  server.addEventListener("one", classOf[String], new ListenerOne(this))
  server.addEventListener("two", classOf[Nothing], new ListenerTwo(this))
  server.start()
}
object QuizServer {
  def main(args: Array[String]): Unit = {
    new QuizServer()
  }
}
class ConnectionListener(server: QuizServer) extends ConnectListener {
  override def onConnect(socket: SocketIOClient): Unit = {
    server.values += socket -> 0
  }
}
class DisconnectionListener(server: QuizServer) extends DisconnectListener {
  override def onDisconnect(socket: SocketIOClient): Unit = {
    server.values -= socket
  }
}
class ListenerOne(server: QuizServer) extends DataListener[String] {
  override def onData(socket: SocketIOClient, data: String, ackRequest: AckRequest): Unit = {
    server.values += socket -> (server.values(socket) + data.length)
    if(data.length > 5){
      socket.sendEvent("update")
    }
  }
}
class ListenerTwo(server: QuizServer) extends DataListener[Nothing] {
  override def onData(socket: SocketIOClient, data: Nothing, ackRequest: AckRequest): Unit = {
    socket.sendEvent("value", server.values(socket).toString)
  }
}
```

This web socket server is running during all the questions on the following page.

```
class QuizValueListener() extends Emitter.Listener {
  override def call(objects: Object*): Unit = {
    val message = objects.apply(0).toString
    println(message)
  }
}
class QuizUpdateListenerOne(socket: Socket) extends Emitter.Listener {
  override def call(objects: Object*): Unit = {
    socket.emit("one", "four")
  }
}
class QuizUpdateListenerTwo(socket: Socket) extends Emitter.Listener {

  var index: Int = 0
  val messages: List[String] = List("1234567", "123456", "12345", "1234")

  override def call(objects: Object*): Unit = {
    socket.emit("one", messages.apply(index))
    index += 1
  }
}
```

What is printed to the screen after each of the following snippets of code is executed and all web socket messages are processed?

```
val socket: Socket = IO.socket("http://localhost:8080/")
socket.on("value", new QuizValueListener())

socket.connect()
socket.emit("two")
```

```
val socket: Socket = IO.socket("http://localhost:8080/")
socket.on("value", new QuizValueListener())

socket.connect()
socket.emit("one", "hello")
Thread.sleep(1000) // Wait for 1 second to allow messages to resolve in a predictable order
socket.emit("two")
```

```
val socket: Socket = IO.socket("http://localhost:8080/")
socket.on("value", new QuizValueListener())
socket.on("update", new QuizUpdateListenerOne(socket))

socket.connect()
socket.emit("one", "123456789")
Thread.sleep(1000) // Wait for 1 second to allow messages to resolve in a predictable order
socket.emit("two")
```

```
val socket: Socket = IO.socket("http://localhost:8080/")
socket.on("value", new QuizValueListener())
socket.on("update", new QuizUpdateListenerTwo(socket))

socket.connect()
socket.emit("one", "12345678")
Thread.sleep(1000) ) // Wait for 1 second to allow messages to resolve in a predictable order
socket.emit("two")
```

Continue on back

Q3. Testing Actors [15 points]

```
case object SendValue
case class Value(value: Int)
case class ShareValue(otherActor: ActorRef)

class QuizActor3(var value: Int) extends Actor {

  def receive: Receive = {
    case SendValue => sender() ! Value(this.value)
    case share: ShareValue => share.otherActor ! Value(this.value)
    case valueMessage: Value => this.value += valueMessage.value
  }

}
```

```
class Q3Test() extends TestKit(ActorSystem("TestQuizActor3"))
  with ImplicitSender
  with WordSpecLike
  with Matchers
  with BeforeAndAfterAll {

  override def afterAll: Unit = {
    TestKit.shutdownActorSystem(system)
  }

  "A quiz actor III" must {
    "update and share it's value appropriately" in {

      val actorA = system.actorOf(Props(classOf[QuizActor3], 10))
      val actorB = system.actorOf(Props(classOf[QuizActor3], 0))

      actorA ! Value(7)
      actorB ! Value(3)

      expectNoMessage(100.millis)

      actorA ! ShareValue(actorB)
      actorB ! SendValue

      val value: Value = expectMsgType[Value](2000.millis)

      assert(value == Value(20))
    }
  }

}
```

This test runs without error, but the assert statement fails when I run it on my laptop. What is causing the test to fail? How would you fix this test?