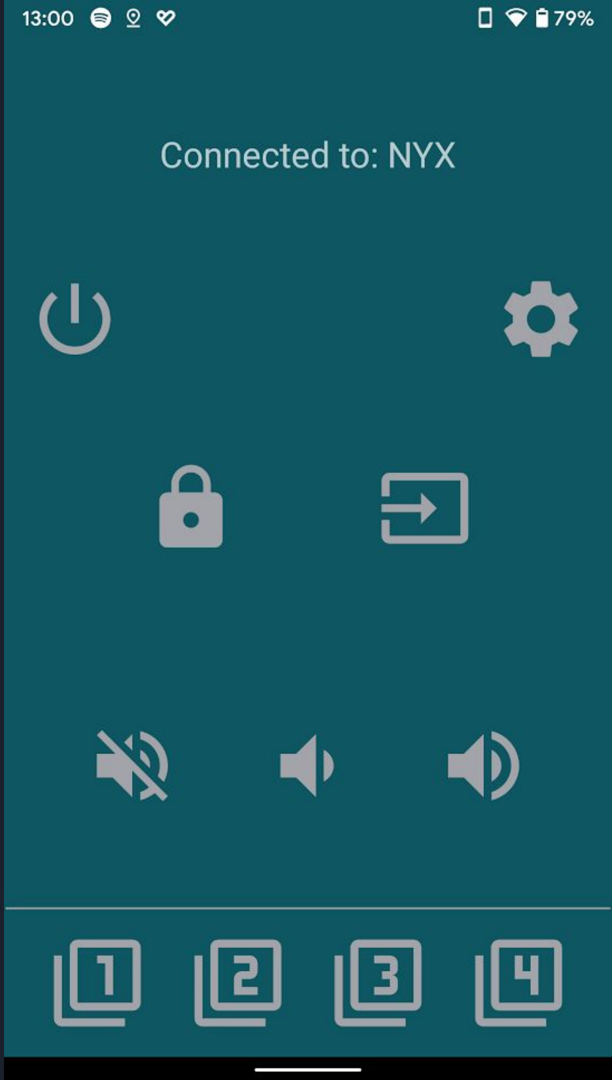
Lynkr:
PC Control from Android over Bluetooth Connection

Jacob Schwartz
12/9/2019

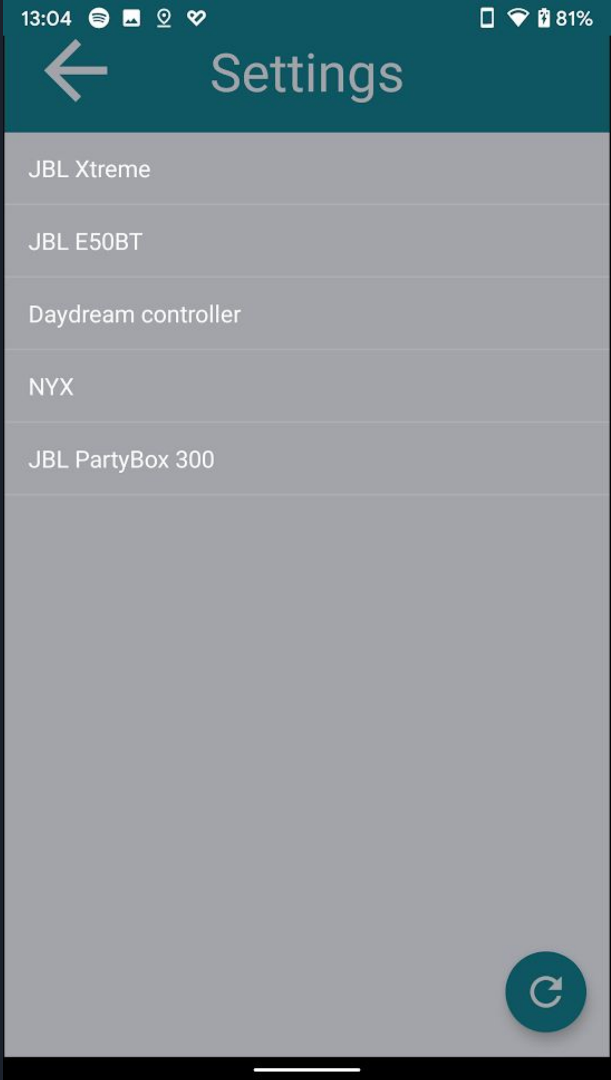
Android UI - Main

- Shutdown
- Lock
- Logout
- Volume Control
  - Mute
  - Decrease
  - Increase
- Keystrokes



Android UI - Settings

- Only device choice, not connection
- Will prompt to enable Bluetooth if disabled
- Refresh required for newly connected devices, or after enabling



JBL Xtreme

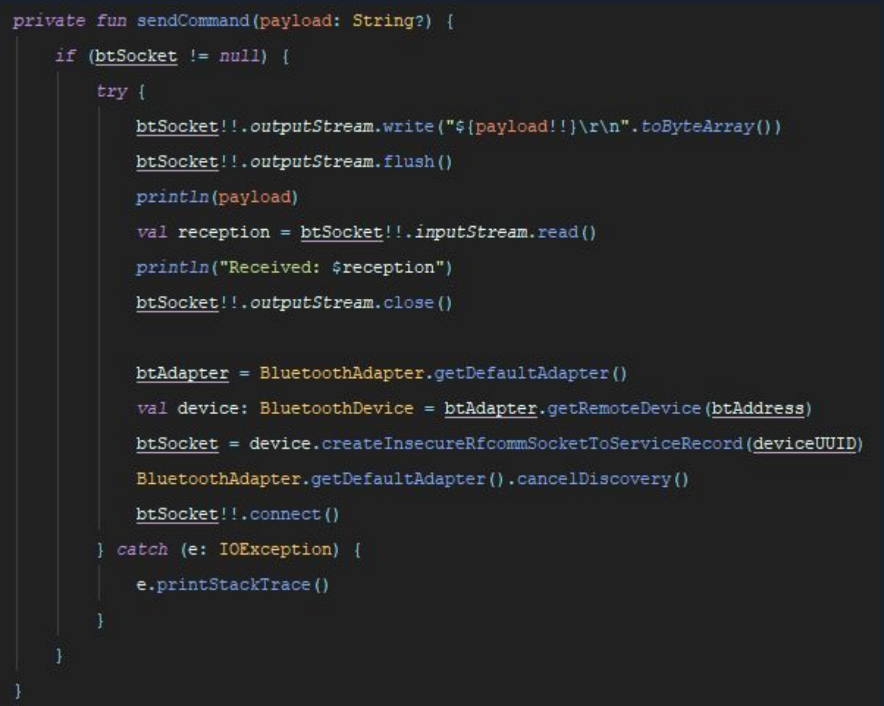
JBL E50BT

Daydream controller

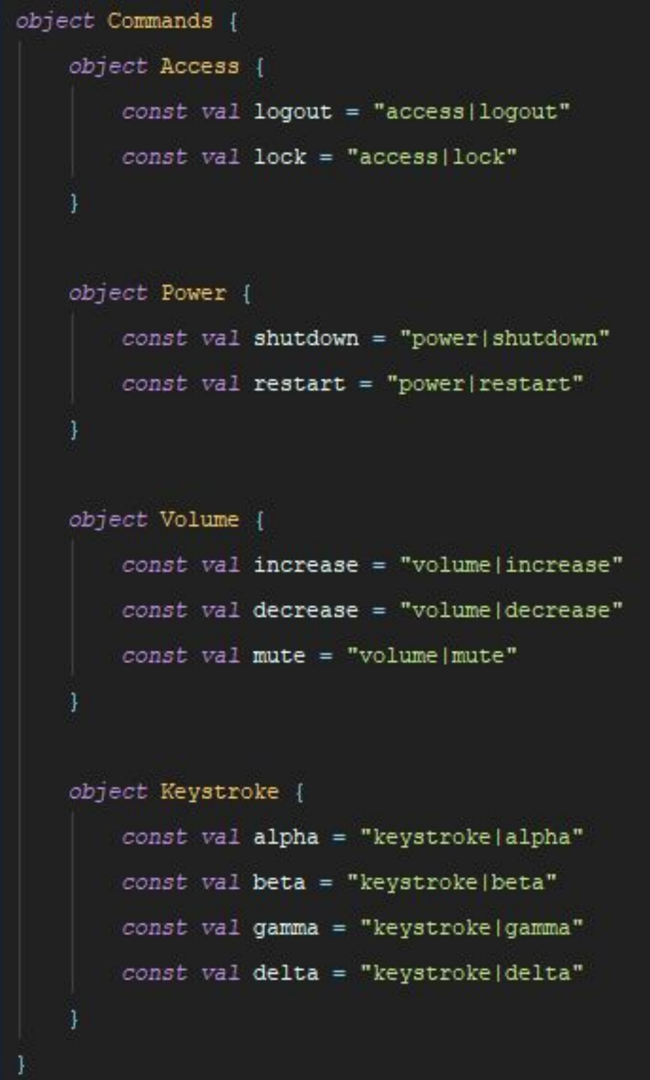
NYX

JBL PartyBox 300

Android - Communication

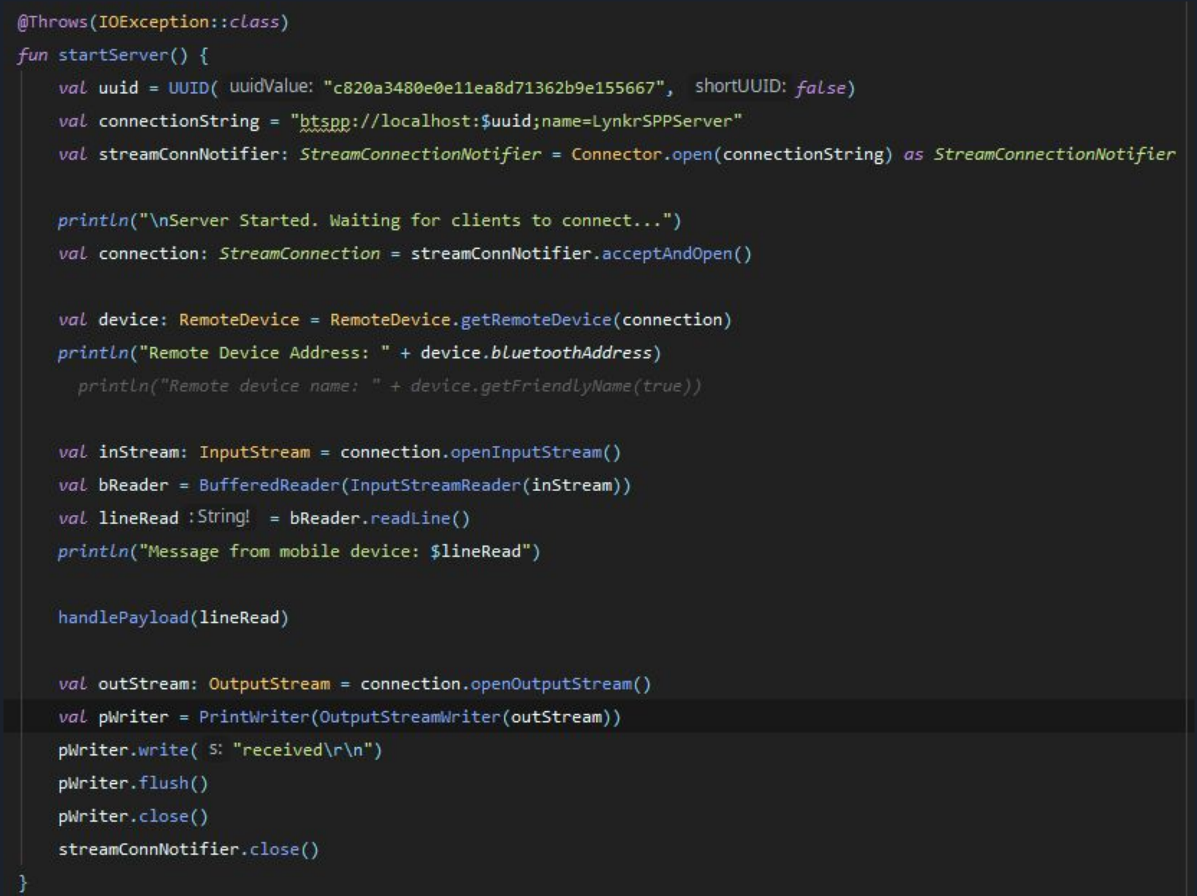


```kotlin
private fun sendCommand(payload: String?) {
    if (btSocket != null) {
        try {
            btSocket!!.outputStream.write("${payload!!}\r\n".toByteArray())
            btSocket!!.outputStream.flush()
            println(payload)
            val reception = btSocket!!.inputStream.read()
            println("Received: $reception")
            btSocket!!.outputStream.close()

            btAdapter = BluetoothAdapter.getDefaultAdapter()
            val device: BluetoothDevice = btAdapter.getRemoteDevice(btAddress)
            btSocket = device.createInsecureRfcommSocketToServiceRecord(deviceUUID)
            BluetoothAdapter.getDefaultAdapter().cancelDiscovery()
            btSocket!!.connect()
        } catch (e: IOException) {
            e.printStackTrace()
        }
    }
}
```



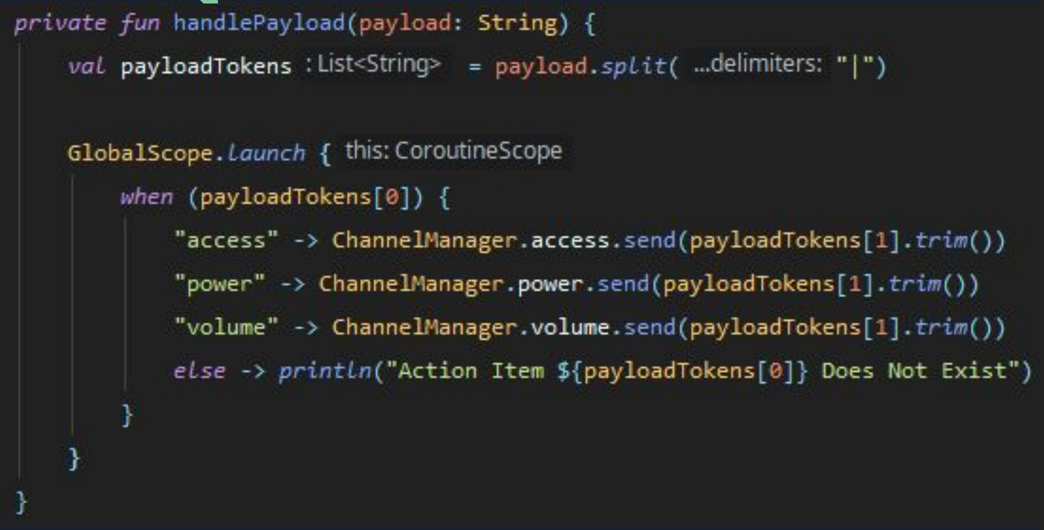
```kotlin
object Commands {
    object Access {
        const val logout = "access|logout"
        const val lock = "access|lock"
    }

    object Power {
        const val shutdown = "power|shutdown"
        const val restart = "power|restart"
    }

    object Volume {
        const val increase = "volume|increase"
        const val decrease = "volume|decrease"
        const val mute = "volume|mute"
    }

    object Keystroke {
        const val alpha = "keystroke|alpha"
        const val beta = "keystroke|beta"
        const val gamma = "keystroke|gamma"
        const val delta = "keystroke|delta"
    }
}
```

Windows - Communication

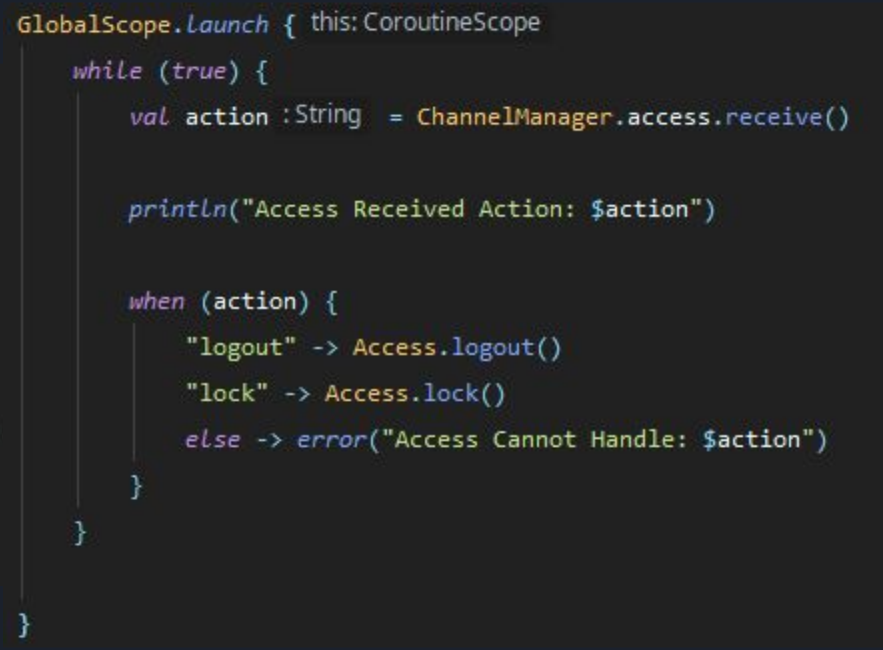


```
@Throws(IOException::class)
fun startServer() {
    val uuid = UUID( uuidValue: "c820a3480e0e11ea8d71362b9e155667",  shortUUID: false)
    val connectionString = "btspp://localhost:$uuid;name=LynkrSPPServer"
    val streamConnNotifier: StreamConnectionNotifier = Connector.open(connectionString) as StreamConnectionNotifier

    println("\nServer Started. Waiting for clients to connect...")
    val connection: StreamConnection = streamConnNotifier.acceptAndOpen()

    val device: RemoteDevice = RemoteDevice.getRemoteDevice(connection)
    println("Remote Device Address: " + device.bluetoothAddress)
      println("Remote device name: " + device.getFriendlyName(true))

    val inStream: InputStream = connection.openInputStream()
    val bReader = BufferedReader(InputStreamReader(inStream))
    val lineRead :String!  = bReader.readLine()
    println("Message from mobile device: $lineRead")

    handlePayload(lineRead)

    val outStream: OutputStream = connection.openOutputStream()
    val pWriter = PrintWriter(OutputStreamWriter(outStream))
    pWriter.write( s: "received\r\n")
    pWriter.flush()
    pWriter.close()
    streamConnNotifier.close()
}
```

Windows - Action Routing

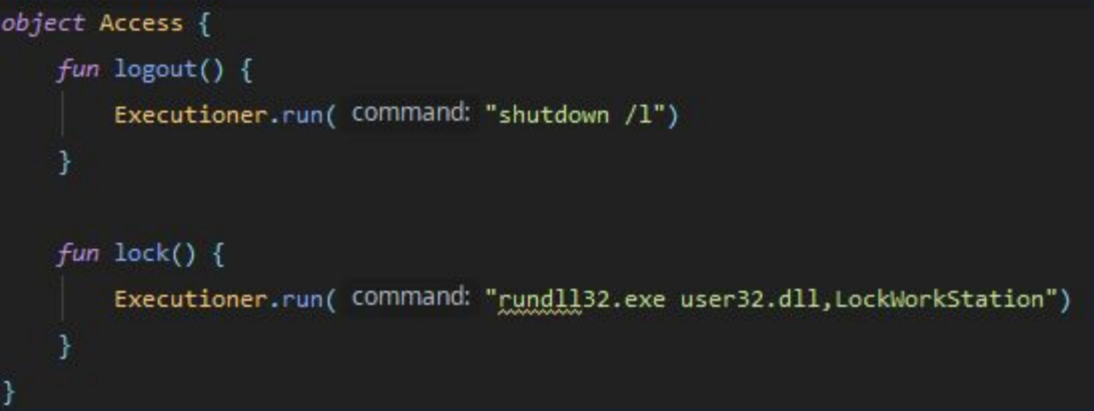


```
private fun handlePayload(payload: String) {
    val payloadTokens :List<String>  = payload.split( ...delimiters: "|")

    GlobalScope.launch { this: CoroutineScope
        when (payloadTokens[0]) {
            "access" -> ChannelManager.access.send(payloadTokens[1].trim())
            "power" -> ChannelManager.power.send(payloadTokens[1].trim())
            "volume" -> ChannelManager.volume.send(payloadTokens[1].trim())
            else -> println("Action Item ${payloadTokens[0]} Does Not Exist")
        }
    }
}
```

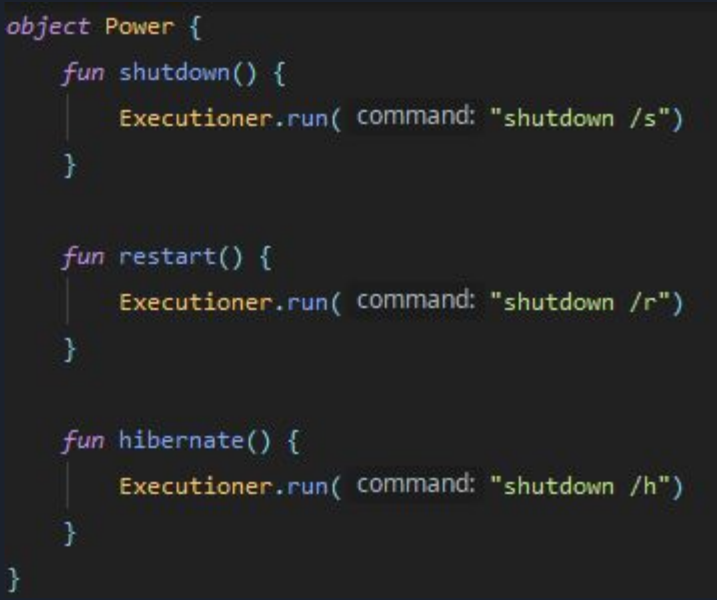


```
GlobalScope.launch { this: CoroutineScope
    while (true) {
        val action :String  = ChannelManager.access.receive()

        println("Access Received Action: $action")

        when (action) {
            "logout" -> Access.logout()
            "lock" -> Access.lock()
            else -> error("Access Cannot Handle: $action")
        }
    }

}
```

Windows - Action Handling

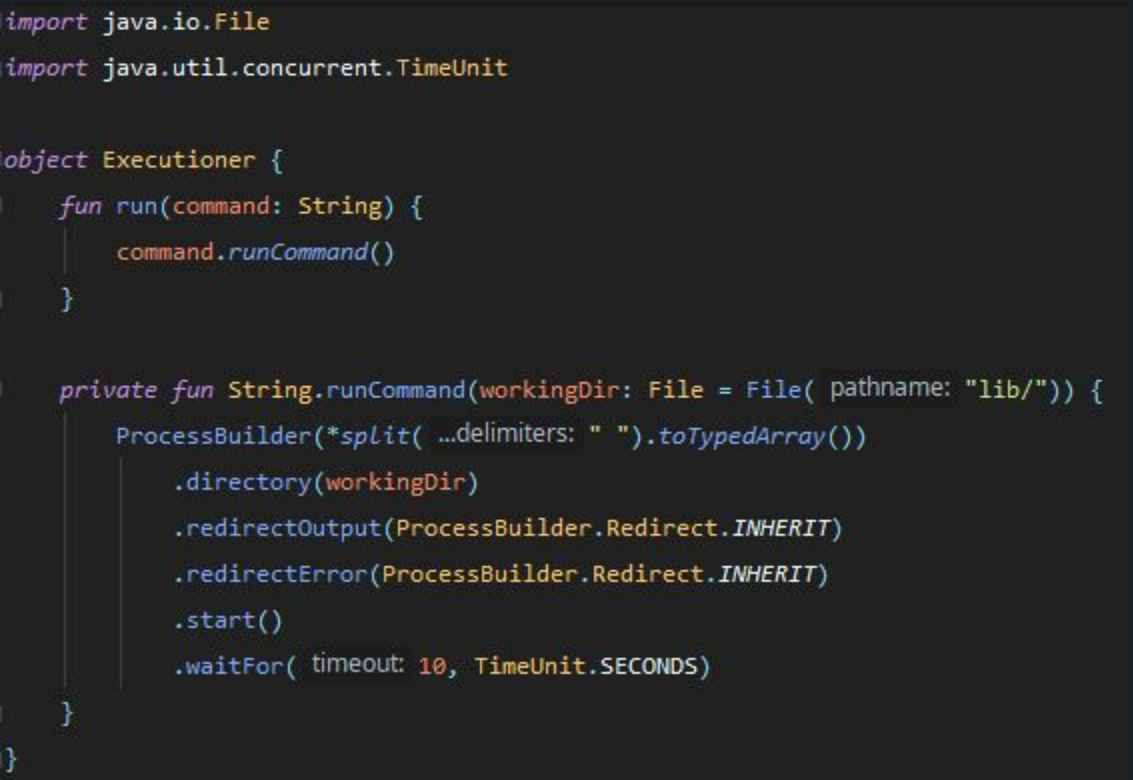


```
object Access {
    fun logout() {
        Executioner.run( command: "shutdown /l")
    }

    fun lock() {
        Executioner.run( command: "rundll32.exe user32.dll,LockWorkStation")
    }
}
```

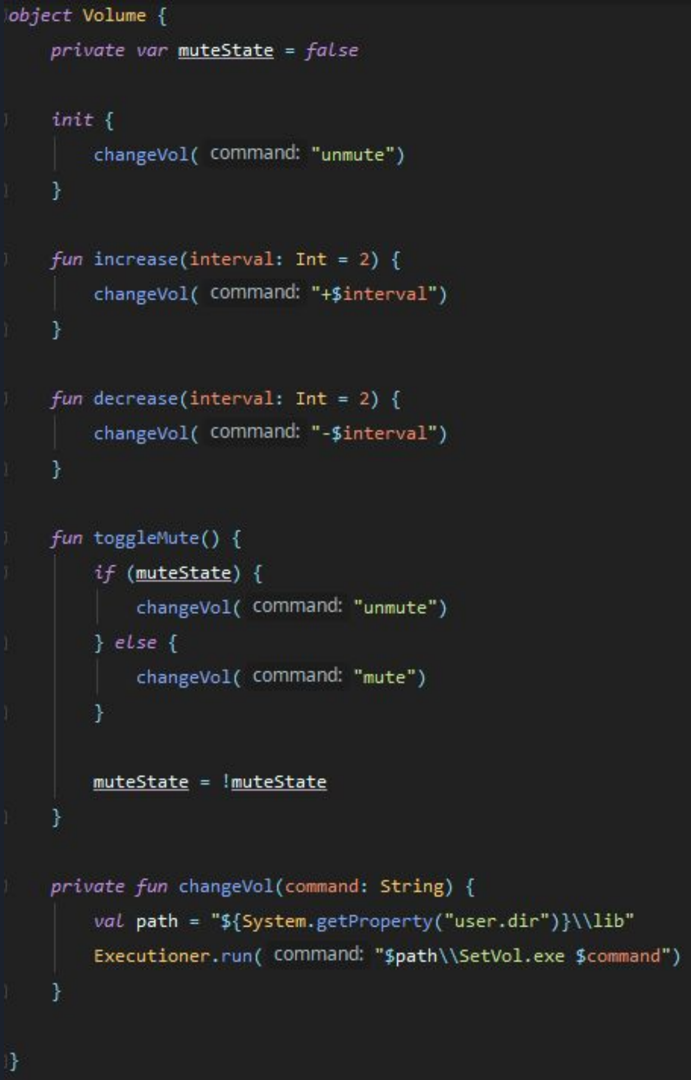


```
object Power {
    fun shutdown() {
        Executioner.run( command: "shutdown /s")
    }

    fun restart() {
        Executioner.run( command: "shutdown /r")
    }

    fun hibernate() {
        Executioner.run( command: "shutdown /h")
    }
}
```

Windows - Action Handling

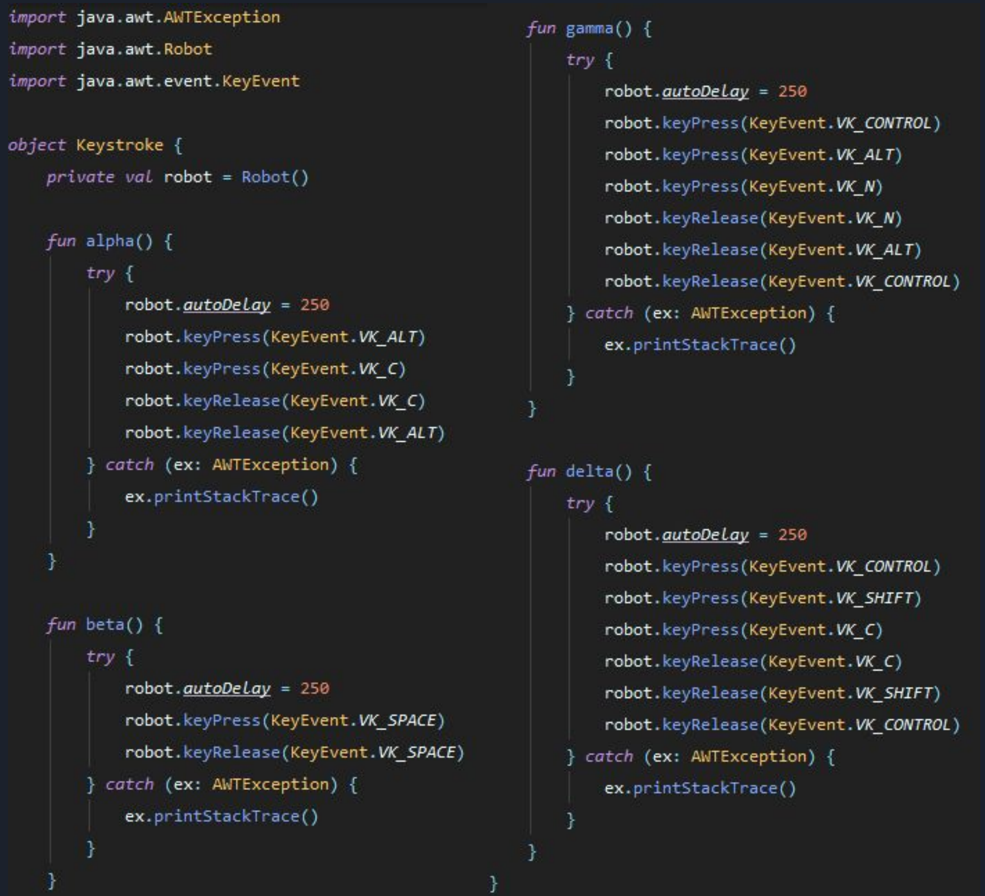


```kotlin
import java.io.File
import java.util.concurrent.TimeUnit

object Executioner {
    fun run(command: String) {
        command.runCommand()
    }

    private fun String.runCommand(workingDir: File = File( pathname: "lib/")) {
        ProcessBuilder(*split( ...delimiters: " ").toTypedArray())
            .directory(workingDir)
            .redirectOutput(ProcessBuilder.Redirect.INHERIT)
            .redirectError(ProcessBuilder.Redirect.INHERIT)
            .start()
            .waitFor( timeout: 10, TimeUnit.SECONDS)
    }
}
```



```kotlin
object Volume {
    private var muteState = false

    init {
        changeVol( command: "unmute")
    }

    fun increase(interval: Int = 2) {
        changeVol( command: "+$interval")
    }

    fun decrease(interval: Int = 2) {
        changeVol( command: "-$interval")
    }

    fun toggleMute() {
        if (muteState) {
            changeVol( command: "unmute")
        } else {
            changeVol( command: "mute")
        }

        muteState = !muteState
    }

    private fun changeVol(command: String) {
        val path = "${System.getProperty("user.dir")}\\lib"
        Executioner.run( command: "$path\\SetVol.exe $command")
    }

}
```

Windows - Keystrokes



```kotlin
import java.awt.AWTException
import java.awt.Robot
import java.awt.event.KeyEvent

object Keystroke {
    private val robot = Robot()

    fun alpha() {
        try {
            robot.autoDelay = 250
            robot.keyPress(KeyEvent.VK_ALT)
            robot.keyPress(KeyEvent.VK_C)
            robot.keyRelease(KeyEvent.VK_C)
            robot.keyRelease(KeyEvent.VK_ALT)
        } catch (ex: AWTException) {
            ex.printStackTrace()
        }
    }

    fun beta() {
        try {
            robot.autoDelay = 250
            robot.keyPress(KeyEvent.VK_SPACE)
            robot.keyRelease(KeyEvent.VK_SPACE)
        } catch (ex: AWTException) {
            ex.printStackTrace()
        }
    }

    fun gamma() {
        try {
            robot.autoDelay = 250
            robot.keyPress(KeyEvent.VK_CONTROL)
            robot.keyPress(KeyEvent.VK_ALT)
            robot.keyPress(KeyEvent.VK_N)
            robot.keyRelease(KeyEvent.VK_N)
            robot.keyRelease(KeyEvent.VK_ALT)
            robot.keyRelease(KeyEvent.VK_CONTROL)
        } catch (ex: AWTException) {
            ex.printStackTrace()
        }
    }

    fun delta() {
        try {
            robot.autoDelay = 250
            robot.keyPress(KeyEvent.VK_CONTROL)
            robot.keyPress(KeyEvent.VK_SHIFT)
            robot.keyPress(KeyEvent.VK_C)
            robot.keyRelease(KeyEvent.VK_C)
            robot.keyRelease(KeyEvent.VK_SHIFT)
            robot.keyRelease(KeyEvent.VK_CONTROL)
        } catch (ex: AWTException) {
            ex.printStackTrace()
        }
    }
}
```