Idiotlin Cheatsheet

1. Ktor

1. create `object App` with `main` method println => ✅
2. create ktor test:
```
@Test class Test {
  fun `When get root path Then return 200 OK`() {
    withTestApplication {
      with(handleRequest(HttpMethod.Get, "/")) {
        assertThat(response.status()).isEqualTo(HttpStatusCode.OK)
```
3. run test => ❗❗
4. implement ktor infrastructure:
```
embeddedServer(factory = Netty, port = 8080) {
   ktor()
 }.start(wait = true)

 fun Application.ktor() {
   routing {
     route("") {
       get {
         call.respondText("hi")
```
5. run test => ❗❗
6. also **test setup** in `withTestApplication({ ktor() }) { ... }`
7. run test => ✅

2. Kodein

1. create **Service** interface + **ServiceImpl** + data class **Model**
2. create **kodein** configuration:
```
fun kodein() = Kodein {
    bind<Service>() with singleton { InMemoryService() }
```
3. pass to **startup** function: `ktor(kodein())`
4. receive service: `val service by kodein.instance<Service>()`
5. return response: `call.respond(service.all())`
6. fix JSON **serialization**:
   1. run test => ❗❗
   2. configure JSON serialization:
      `install(ContentNegotiation) { serialization() }`
   3. run test => ❗❗
   4. add `@Serializable` annotation to model
   5. run test => ✅

3. Test JSON Content

1. extend existing test to assert `isEqualJsonArray()`
2. use some dummy JSON; run test => ❗❗
3. fix to proper JSON; run test => ✅
4. **override** service bean
   1. override Kodein (use **stub** instead of mockk):

```
ktor(Kodein {
  extend(kodein())
  bind<Service>(overrides = true) with
    instance(TestableService(listOf(model)))
})
```

   2. implement `TestableService`
   3. add a `Model` instance

4. Exposed

1. create new **interface** `ModelRepository`
2. create new **implementation** `ExposedModelRepository`
3. bind **kodein** instance, wire into service and delegate
4. implement exposed **table** and DB **connection** method:

```
fun connectToDatabase(dbUrl: String =
"jdbc:h2:mem:idiotlinDb;DB_CLOSE_DELAY=-1"): Database {
  val db = Database.connect(url = dbUrl, driver = "org.h2.Driver")
    transaction {
      SchemaUtils.create(ModelTable)
    }
  return db
}

object ModelTable : Table() {
  val name = varchar("name", length = 50)
}

// ad ExposedModelRepository:
transactional {
 ModelTable.selectAll().map {
    Model(name = it[ModelTable.name]) }}
```

5. connect to DB in `main()` method
6. run test => ✅

```
object App {
    @JvmStatic
    fun main(args: Array<String>) {
        connectToDatabase()
        embeddedServer(factory = Netty, port = 8080) {
            ktor()
        }.start(wait = true)
    }
}
fun Application.ktor(kodein: Kodein = kodein()) {
    install(ContentNegotiation) {
        serialization()
    }
    val service by kodein.instance<Service>()
    routing {
        route("") {
            get {
                call.respond(service.readAll())
}}}}
@Serializable
data class Model(
    val name: String
)
interface Service {
    fun readAll(): List<Model>
}
class ServiceImpl(
    private val repo: ModelRepository
) : Service {
    override fun readAll() = repo.fetchAll()
}
fun kodein() = Kodein {
    bind<ModelRepository>() with singleton { ExposedModelRepository() }
    bind<Service>() with singleton { ServiceImpl(instance()) }
}
fun connectToDatabase(dbUrl: String = "jdbc:h2:mem:idiotlinDb;DB_CLOSE_DELAY=-1"): Database {
    log.info { "Connecting to database: $dbUrl" }
    val db = Database.connect(url = dbUrl, driver = "org.h2.Driver")
    transaction {
        SchemaUtils.create(ModelTable)
    }
    return db
}
object ModelTable : Table() {
    val name = varchar("name", length = 50)
}
interface ModelRepository {
    fun fetchAll(): List<Model>
}
class ExposedModelRepository : ModelRepository {
    override fun fetchAll() = transaction {
        ModelTable.selectAll().map {
            Model(name = it[ModelTable.name])
}}}
```

```
@Test
class KtorTest {

    private val model = Model("test model")

    fun `When get root endpoint Then return 200 ok`() {
        withTestApplication({
            ktor(Kodein {
                extend(kodein())
                bind<Service>(overrides = true) with instance(TestableService(listOf(model)))
            })
        }) {
            with(handleRequest(HttpMethod.Get, "/")) {
                assertThat(response.status()).isEqualTo(HttpStatusCode.OK)
                assertThat(response.content).isEqualJsonArray("""[{"name":"${model.name}"}]""")
            }
        }
    }
}

class TestableService(
    private val models: List<Model>
) : Service {
    override fun readAll() = models
}



@Test
class ExposedModelRepositoryTest {
    private val model = Model("test model")
    private lateinit var db: Database
    @BeforeMethod
    fun `init db`() {
        db = connectToDatabase("jdbc:h2:mem:testDb;DB_CLOSE_DELAY=-1")
    }
    @AfterMethod
    fun `reset db`() {
        db.connector.invoke().close()
    }
    fun `When fetch all Then empty`() {
        assertThat(ExposedModelRepository().fetchAll()).isEmpty()
    }
    fun `Given model inserted When fetch all Then return that model`() {
        transaction {
            ModelTable.insert {
                it[name] = model.name
            }
        }

        assertThat(ExposedModelRepository().fetchAll()).all {
            hasSize(1)
            contains(model)
        }
    }
}
```