Systematic Testing of a Doujinsoft Application

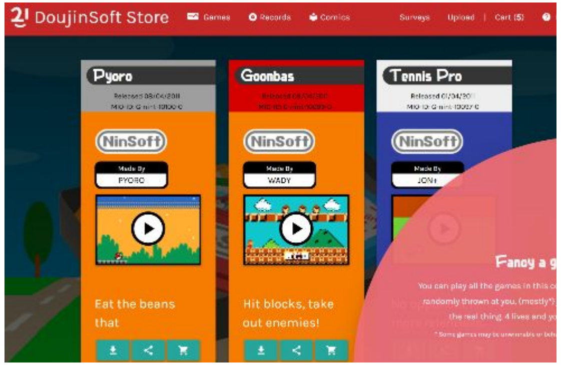
Xuemeng Wei, Yihang Wang, Jishuo Yang

Introduction

- **What is DoujinSoft?**

DoujinSoft is a Java-based web application designed to manage and distribute WarioWare DIY (WWDIY) fan-made content, including games, comics, and music. It acts as a digital archive and shop, allowing users to browse, search, and interact with user-created content.

- **what platform(s) is it run?**
  - Apache Tomcat (Servlet container).
  - Docker .
  - GitHub Codespaces (Cloud-based development/testing).
- **Active users?**
  - Creators who want to upload .mio game files
  - Players browsing the archive and content
  - Admins managing content (developers)



Introduction(cont.)

- Technologies

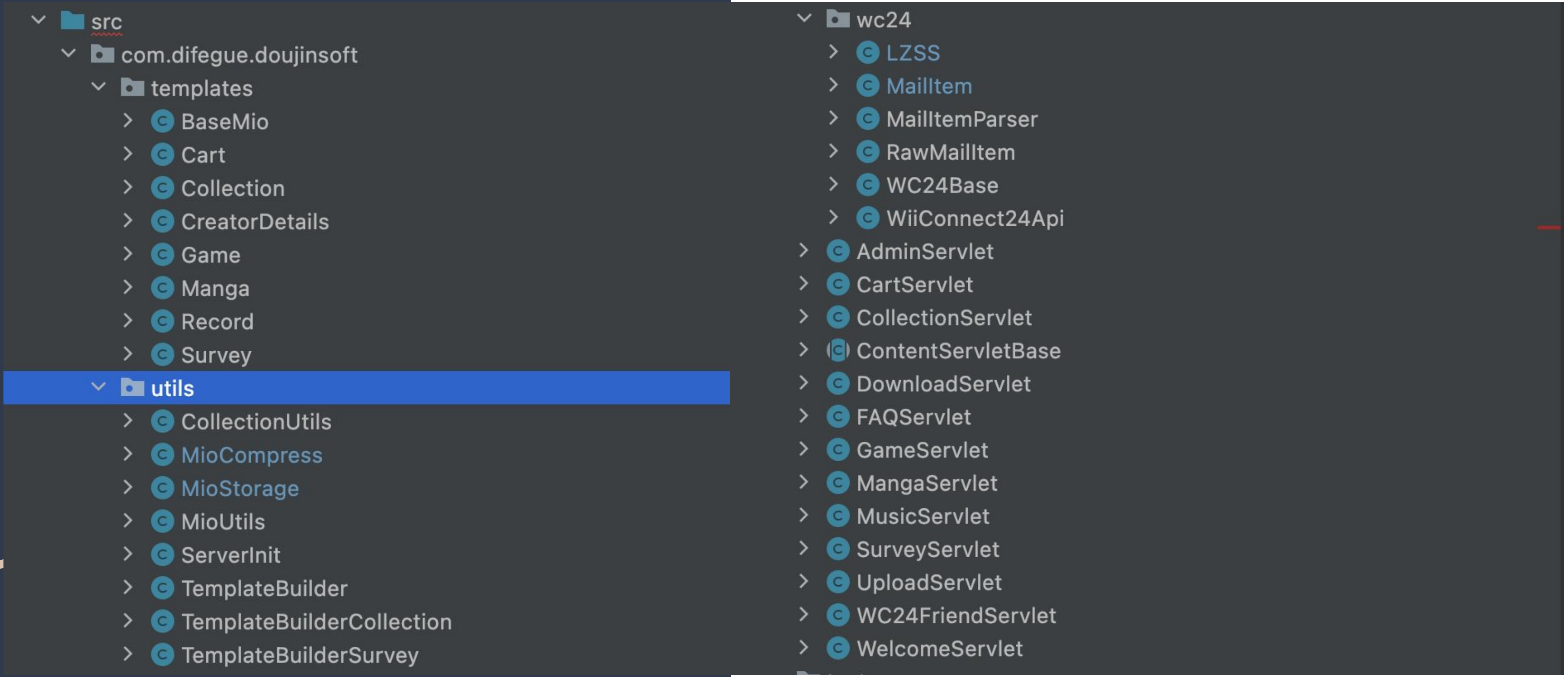
Backend

- Language: Java (Servlet/JSP)
- Framework: Apache Tomcat (Servlet container)
- Database: SQLite
- Build Tool: Maven
- API: JSON API

Frontend

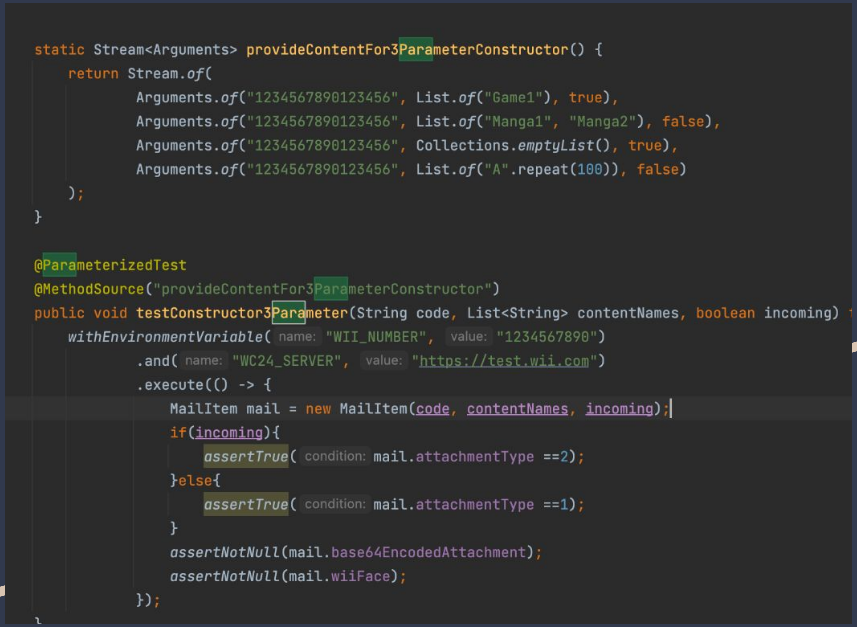
- HTML
- MaterializeCSS and JQuery

Code structure



```
src
  com.difegue.doujinsoft
    templates
      BaseMio
      Cart
      Collection
      CreatorDetails
      Game
      Manga
      Record
      Survey
    utils
      CollectionUtils
      MioCompress
      MioStorage
      MioUtils
      ServerInit
      TemplateBuilder
      TemplateBuilderCollection
      TemplateBuilderSurvey
    wc24
      LZSS
      MailItem
      MailItemParser
      RawMailItem
      WC24Base
      WiiConnect24Api
    AdminServlet
    CartServlet
    CollectionServlet
    ContentServletBase
    DownloadServlet
    FAQServlet
    GameServlet
    MangaServlet
    MusicServlet
    SurveyServlet
    UploadServlet
    WC24FriendServlet
    WelcomeServlet
```

Blackbox testing



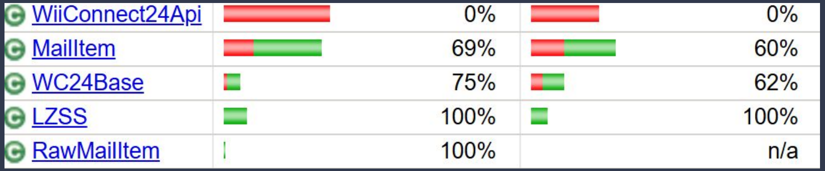
```
static Stream<Arguments> provideContentFor3ParameterConstructor() {
    return Stream.of(
            Arguments.of("1234567890123456", List.of("Game1"), true),
            Arguments.of("1234567890123456", List.of("Manga1", "Manga2"), false),
            Arguments.of("1234567890123456", Collections.emptyList(), true),
            Arguments.of("1234567890123456", List.of("A".repeat(100)), false)
    );
}

@ParameterizedTest
@MethodSource("provideContentFor3ParameterConstructor")
public void testConstructor3Parameter(String code, List<String> contentNames, boolean incoming)
    withEnvironmentVariable( name: "WII_NUMBER",  value: "1234567890")
            .and( name: "WC24_SERVER",  value: "https://test.wii.com")
            .execute(() -> {
                MailItem mail = new MailItem(code, contentNames, incoming);
                if(incoming){
                    assertTrue( condition: mail.attachmentType ==2);
                }else{
                    assertTrue( condition: mail.attachmentType ==1);
                }
                assertNotNull(mail.base64EncodedAttachment);
                assertNotNull(mail.wiiFace);
            });
}
```

Goal: Test based on expected behavior

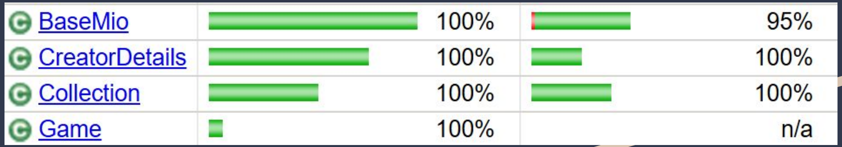
What we did:

- Simulated black-box testing
- Equivalence partitioning
- Edge cases(null, empty..)
- Combined equivalence blocks for ACoC and ECC coverage
- Use **@TempDir** to safely handle temporary files

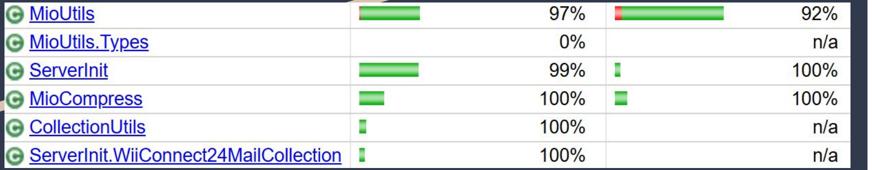
Whitebox testing



WiiConnect24Api	0%	0%
MailItem	69%	60%
WC24Base	75%	62%
LZSS	100%	100%
RawMailItem	100%	n/a



BaseMio	100%	95%
CreatorDetails	100%	100%
Collection	100%	100%
Game	100%	n/a



MioUtils	97%	92%
MioUtils.Types	0%	n/a
ServerInit	99%	100%
MioCompress	100%	100%
CollectionUtils	100%	n/a
ServerInit.WiiConnect24MailCollection	100%	n/a

Goal: 95% statement coverage & branch coverage

We have complex classes that we are still working on.

What we did:

In this process, mocking is heavily used since many classes need to interact with http/SQLite connection, or have branches that needs an exception to be thrown(where we mock one accordingly, especially when one happens rarely like IOException).

Testing Details – Utils

- **CollectionUtils class**
  1. GetCollectionFromFile(String path)

     EP_1: null EP_2: path points to a non-existent file EP_3: path points to a valid JSON file matching the Collection

  2. SaveCollectionToFile(Collection c, String path)

     C1: c == null C_2: c != null

     P_1: null P_2: path invalid or unwritable (e.g. parent directory missing) P_3: path valid

- **MioCompress class**
  1. compressMio(File orig, File dest, String desiredName)

     EO_1: null EO_2: path not exist EO_3: directory not file EO4: file unreadable EO5: valid

     ED_1: dest == null ED_2: path not exist ED_3: directory not file ED4: read only ED5: valid

     En_1: name = null EO_2: length=0 EO_3: length >=1

Testing Details – wc24

- **LZSS class : It doesn't implement compression itself—it relies on an external C++ tool (gbalzss.exe or gbalzss) to do the actual work.**
  1. **LZS_Encode(String filename, String output): Compresses a file using LZSS**
  2. **LZS_Decode(String filename, String output): Decompresses an LZSS-encoded file**

  Mock testing: Mocked **Runtime** and **ServletContext** to verify **OS-specific** command generation. (Use **System.setProperty**("os.name", "Windows 11");

- **MailItem class:**

  Simulate presence/absence of required env vars

Integration + GUI testing (Servlet)

- **Tool: selenium**
- **What we did:**
  - The admin page to add collections
  - Download buttons under /games, /records, /comics to download the .mio file (they are .miozip on server)
  - Sharing buttons under /games, /records, /comics to get an unique URL to access one work
  - Cart buttons to add works to cart
  - Next button to get next page
  - Search button to search
  - Basic access to other endpoints(/home, /about…)

- We are still working on this…

API testing

Content list API

Endpoint: /games, /records, /comics

More specially:

GET with no parameters, with or without”format=json”,

“/games?page=2”, “/games?id=some_hash”, “/games?name=Test”, “/games?creator=Author”, “/games?name=Test&creator=Author”, “/games?cartridge_id=some_id”, “sort_by=date”, “sort_by=name”...

API testing

Download API

- Endpoint: /download
- More specially:

GET /download?type=game&id=some_hash

GET
/download?type=game&id=some_hash&preview
=true

API testing

- (POST)
- Upload API
- Cart API
- Manage API

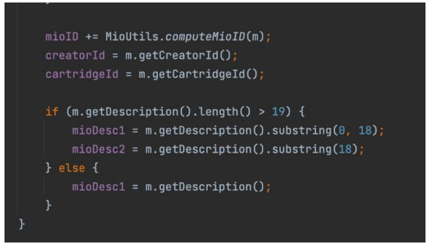
Still working on it!

Live demo

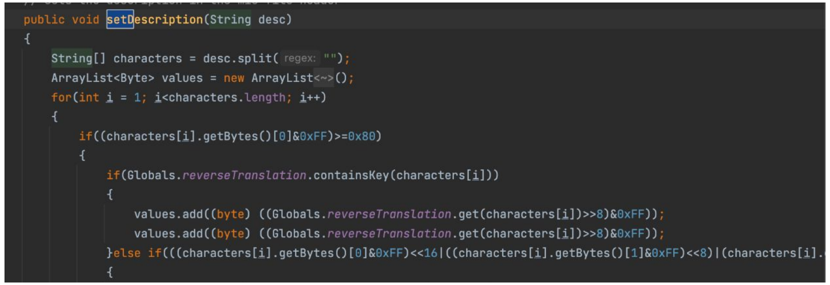
Issues or Errors (BaseMio)

Failure that comes from external package(com.xperia64.diyedit: Metadata)

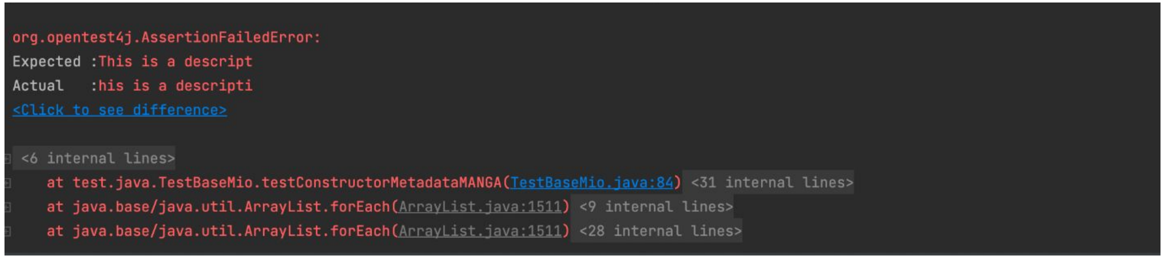
Highlight the importance of testing, even the program logic looks absolutely right



```
    mioID += MioUtils.computeMioID(m);
    creatorId = m.getCreatorId();
    cartridgeId = m.getCartridgeId();

    if (m.getDescription().length() > 19) {
        mioDesc1 = m.getDescription().substring(0, 18);
        mioDesc2 = m.getDescription().substring(18);
    } else {
        mioDesc1 = m.getDescription();
    }
}
```



```
public void setDescription(String desc)
{
    String[] characters = desc.split( regex: "");
    ArrayList<Byte> values = new ArrayList<~>();
    for(int i = 1; i<characters.length; i++)
    {
        if((characters[i].getBytes()[0]&0xFF)>=0x80)
        {
            if(Globals.reverseTranslation.containsKey(characters[i]))
            {
                values.add((byte) ((Globals.reverseTranslation.get(characters[i])>>8)&0xFF));
                values.add((byte) ((Globals.reverseTranslation.get(characters[i])>>8)&0xFF));
            }else if(((characters[i].getBytes()[0]&0xFF)<<16|((characters[i].getBytes()[1]&0xFF)<<8)|(characters[i].
            {
```

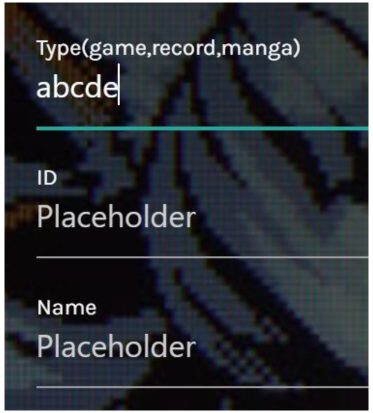


```
org.opentest4j.AssertionFailedError:
Expected :This is a descript
Actual   :his is a descripti
<Click to see difference>

<6 internal lines>
    at test.java.TestBaseMio.testConstructorMetadataMANGA(TestBaseMio.java:84) <31 internal lines>
    at java.base/java.util.ArrayList.forEach(ArrayList.java:1511) <9 internal lines>
    at java.base/java.util.ArrayList.forEach(ArrayList.java:1511) <28 internal lines>
```

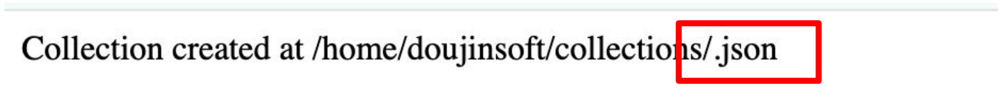
Issues or Errors (AdminServlet)

In Admin page, the website have no restriction on the type of collection. The collection should only be (GAME, RECORD, MANGA). But admin could type anything into the input box.

It should be replaced with a drop-down list to prevent unexpected user input at the client's side. Although the backend has its logic to return a default value, it's still not a good practice.



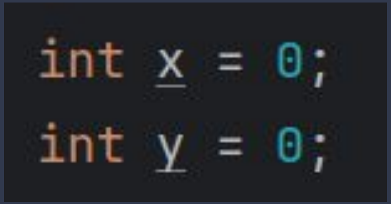
Type(game,record,manga)
abcde
ID
Placeholder
Name
Placeholder



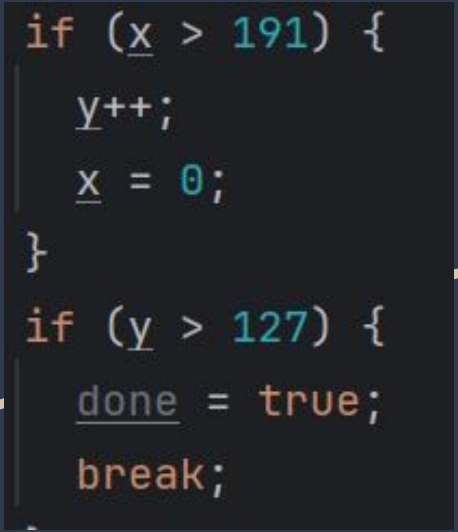
Collection created at /home/doujinsoft/collections/.json

id.json

Issues or Errors



```
int x = 0;
int y = 0;
```



```
if (x > 191) {
  y++;
  x = 0;
}
if (y > 127) {
  done = true;
  break;
```

getPixel() and setColor()

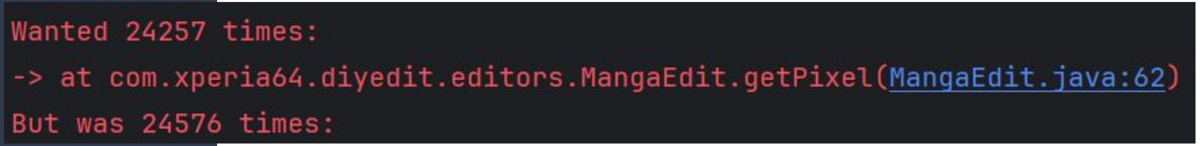
should run **24527(191*127) times**

but actually run **24576(192*128) times**

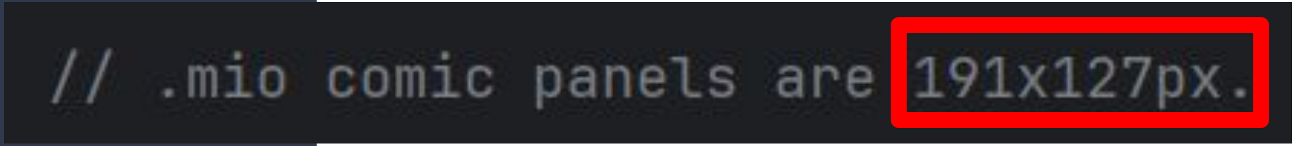
Fault: should be ”x>=191” and “y>=127”

It’s an example showing state error does not necessarily lead to failure:

It reads pixel from a binary file that contains more information than image; the drawing is wrapped using Graphics2D and will not throw during the process and the extra pixel will be cropped out automatically



```
Wanted 24257 times:
-> at com.xperia64.diyedit.editors.MangaEdit.getPixel(MangaEdit.java:62)
But was 24576 times:
```



```
// .mio comic panels are 191x127px.
```

Issues or Errors

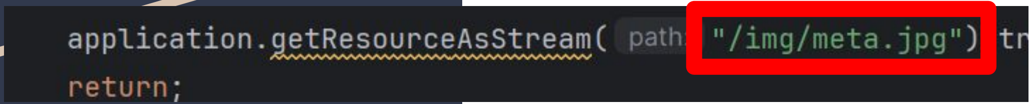
Download API (download preview picture)

(/download?type=game&id=some_hash&preview=true
)

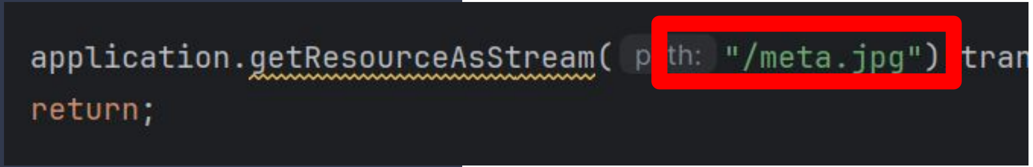
Expected: return a default jpg(meta.jpg) when id does not exists or type is not defined

Actually: successfully return for record/other undefined types, fail to do that for game/manga

Faults: write wrong path for the block that deals with the game/manga

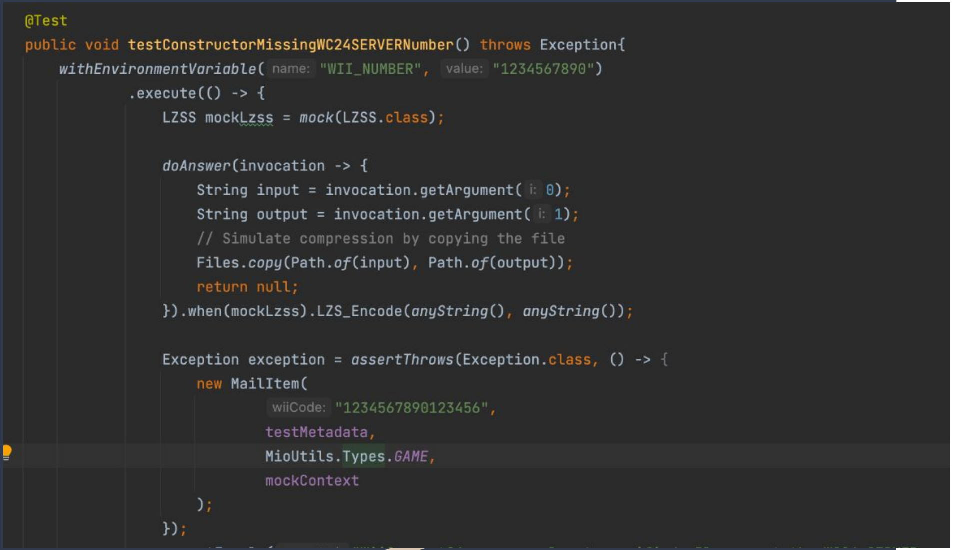


```
application.getResourceAsStream( path: "/img/meta.jpg") tr
return;
```



```
application.getResourceAsStream( path: "/meta.jpg") tran
return;
```

Challenges



```java
@Test
public void testConstructorMissingWC24SERVERNumber() throws Exception{
    withEnvironmentVariable( name: "WII_NUMBER",  value: "1234567890")
            .execute(() -> {
                LZSS mockLzss = mock(LZSS.class);

                doAnswer(invocation -> {
                    String input = invocation.getArgument( i: 0);
                    String output = invocation.getArgument( i: 1);
                    // Simulate compression by copying the file
                    Files.copy(Path.of(input), Path.of(output));
                    return null;
                }).when(mockLzss).LZS_Encode(anyString(), anyString());

                Exception exception = assertThrows(Exception.class, () -> {
                    new MailItem(
                            wiiCode: "1234567890123456",
                            testMetadata,
                            MioUtils.Types.GAME,
                            mockContext
                    );
                });
```

- MailItem

java.lang.reflect.InaccessibleObjectException: Unable to make field private static final java.util.HashMap java.lang.ProcessEnvironment.theEnvironment accessible: module java.base does not "opens java.lang" to unnamed module @548ad73b

How to mock different environment variables? → use junit-pioneer or system-lambda

Mock static method and constructor? → Mockito-inline (it becomes the default mock maker after Mockito 5.0)

a method is passed asynchronously (like a Runnable to a scheduler) and can't be called directly → use ArgumentCaptor to capture the method

What we learnt

1. Strength skills on black box, whitebox, Integration test, GUI, API testing
2. Environment variable management (SystemLambda for safe env-var mocking in tests)
3. External Process Integration (Runtime.exec(), System.setProperty)
4. Advanced mocking techniques (static mock)
5. TempDirectory for testing files compression safely