這次的講座主要是介紹最終的作業——一個多用途的套接字伺服器遊戲引擎 (socket server game engine)。這個作業類似於之前的資料庫 (DB) 作業，但這次的目標是實作一個文字冒險遊戲的引擎，而不只是處理 SQL 語句。

**關鍵概念**

1. **伺服器架構**
   * 你的遊戲引擎會運行在 **port 8888** 上，並透過 socket 通訊來接收來自用戶端的指令。
   * 每當接收到指令，伺服器會**處理指令、修改遊戲狀態、回傳結果**，然後關閉連線，等待下一個指令。
   * **不同於資料庫作業**，這次的伺服器需要支援**多用戶**，但仍然是單線程處理（每個請求都會阻塞直到完成）。
2. **遊戲指令**
   * 遊戲中有一些**經典的文字冒險遊戲指令**：
     + inventory (或 iv): 顯示當前持有的物品。
     + get [item]: 從目前位置撿起物品，加入物品欄 (inventory)。
     + drop [item]: 丟棄持有的物品，放回當前位置。
     + goto [location]: 移動到另一個位置。
     + look (或 l): 顯示當前位置的資訊、可互動的物品、其他玩家、通往的路線。
3. **遊戲世界的結構**
   * 遊戲世界透過兩個**配置檔案**來描述：
     + **Entities 檔案 (使用 DOT 語言)**：
       - 這是一種圖結構語言 (Graph Representation)，定義遊戲中的**位置 (locations)**、**物品 (artefacts)**、**傢俱 (furniture)**、**角色 (characters)** 之間的關係。
       - artefacts 是可撿起的物品，而 furniture 則是固定的物體，無法帶走。
       - 你可以透過 **Graphviz** 等工具視覺化這個圖結構。
     + **Actions 檔案 (使用 XML 語言)**：
       - 這個檔案描述**遊戲中的動作 (actions)**，每個動作包含：
         * **Triggers** (觸發詞)：用戶輸入的關鍵詞，例如 "chop", "cut", "cut down" 都能觸發砍樹的動作。
         * **Subjects** (主體)：執行動作時需要的條件，例如砍樹需要有 axe 和 tree。
         * **Consumed** (被消耗的物品)：執行動作後會消失的物品，例如 tree 會消失。
         * **Produced** (產生的物品)：動作執行後產生的東西，例如 log（木頭）會出現在場景中。
4. **指令解析器 (Parser)**
   * 這次不需要自己手寫 Parser，而是使用 **兩個已經提供的解析器**：
     + **JPGD** (Alexander Metz) 用於解析 .dot 文件（遊戲世界地圖）。
     + **JAXP** (Java API for XML Processing) 用於解析 .xml 文件（遊戲動作）。
   * 你的工作是**使用這些 API 讀取並解析遊戲資料**，然後將其轉換為適當的 Java 類別。
5. **語言處理 (Command Parsing)**
   * **這次沒有 BNF (巴科斯範式)，而是自然語言**，所以需要處理：
     + **大小寫不敏感** (Case-insensitive)
     + **修飾詞 (Decorated Commands)**：使用者可能會輸入 please chop down the lovely tree using the shiny axe 這樣的句子，但你要從中解析出關鍵詞 chop tree with axe。
     + **不同詞序 (Varying Word Order)**：chop tree with axe 和 use axe to chop tree 本質上是同一個指令。
     + **省略詞 (Partial Commands)**：如果玩家只有一把斧頭，輸入 chop tree 應該要自動推斷 axe 是使用的工具。
     + **多餘的物品 (Extraneous Entities)**：指令 open trapdoor with key and axe 如果 axe 不是必要條件，就應該拒絕執行。
     + **模糊指令 (Ambiguous Commands)**：如果場景中有 red door 和 blue door，輸入 open door with key 會導致歧義，因此應該要求使用者進一步指定。
6. **玩家身份識別**
   * 每個指令都必須以 username: 開頭，例如：

simon: open trapdoor with key

* + 不需要驗證用戶名，只要讀取並解析即可。

1. **特別的 Java 限制**
   * 這次作業**禁止使用以下 Java 特性**：
     + **Lambda 表達式** (->)
     + **陣列 (Array) 和 ArrayList**
     + **三元運算子 (?:)**
     + **非限定方法調用 (Unqualified Method Calls)**，必須寫成 this.methodName()。
     + **使用 + 來串接字串**（改用 StringBuilder 或 printf）。
   * 這些限制的目的是讓你學習 Java 更廣泛的功能，也能避免直接抄襲舊的作業。
2. **自動化測試**
   * 你可以使用 **Strange 檢查工具** 來檢測你的代碼是否違反上述 Java 限制。
   * 測試檔案包含：
     + example-stag-tests
     + actions-test
     + entities-test
   * 這些測試會幫助你確保 .dot 和 .xml 解析器能夠正常運作。
3. **評分標準**
   * 這次作業佔 **60% 的分數**，評分依據：
     + **是否成功解析並執行遊戲指令**
     + **是否正確讀取 .dot 和 .xml 配置文件**
     + **程式的彈性與錯誤處理能力**
     + **程式碼品質（清晰、易讀、符合 Java 規範）**
     + **不得使用違規的 Java 語法**
   * **自動測試** 會用隱藏的測試案例來評分，因此你的程式**必須遵守指定的命令格式**，不能隨意修改指令名稱。

**建議的第一步**

1. **讀取並解析 entities.dot 文件**，建立遊戲世界的物件結構（Location、Artefact、Furniture）。
2. **讀取 actions.xml 文件**，將動作對應到遊戲世界的變化。
3. **處理基本的指令解析**，例如 inventory, get item, drop item, goto location, look。
4. **擴展語言解析能力**，支援修飾詞、省略詞等變化。
5. **實作錯誤處理**，確保程式對於無效指令能適當回應錯誤訊息。

**🧠 作業目的與背景**

* 建立一個**文字冒險遊戲的遊戲引擎**，而非特定遊戲。
* 架設在 **Socket Server** 上，與使用者進行通訊（使用 port 8888）。
* 重點在於處理指令、維護遊戲狀態，並回傳回應。
* 會提供兩個設定檔（entities.dot 與 actions.xml）來定義遊戲世界與動作。

**🛑 禁止使用的 Java 語法（Illegal Constructs）**

**⚠️ 若使用下列任一項，直接 0 分處理：**

1. **Lambda 表達式** (->)
2. **Array / ArrayList**
3. **三元運算子** (condition ? x : y)
4. **非限定方法呼叫**（必須使用 this.methodName()）
5. **字串相加** (+)（改用 StringBuilder 或 printf）

✅ 使用 Set, Map, LinkedList, Deque 等 Collection 類別是允許的。

📌 使用 mvnw exec:java@strange -Dexec.args=檔案路徑 可檢查是否有違規。

**🧱 遊戲引擎結構與建構重點**

* 主類別：GameServer
* 入口建構子必須為：

public GameServer(File entitiesFile, File actionsFile)

* 主要功能實作於 handleCommand(String command) 方法中，處理並回應用戶端指令。
* 每次啟動伺服器，會重新載入設定檔（不保留遊戲進度），不可將遊戲狀態寫入設定檔。

**🧭 遊戲核心指令（內建）**

1. inventory 或 inv：顯示玩家目前持有的物品。
2. get [item]：從場景中撿起物品。
3. drop [item]：將物品放回場景中。
4. goto [location]：移動到其他位置。
5. look：檢視當前場景的詳細資訊（物品、人物、可通往的位置）。

🛑 指令名稱為**保留字**，不可用作其他實體名稱。

**🧩 遊戲元素 (Entities)**

儲存在 .dot 檔案中，使用 **DOT Graph 語言** 表示。

* **角色（Character）**：NPC、動物、機器人等。
* **玩家（Player）**：玩家本身。
* **位置（Location）**：遊戲中的場景，具有通往其他位置的「路徑」。
* **物品（Artefact）**：可撿起放入背包。
* **傢俱（Furniture）**：固定在場景中，無法移動。

📍 特殊位置：

* 第一個位置為遊戲起始點。
* storeroom：存放尚未出現在場景中的物品或角色，若未定義要自動建立。

使用 **JPGD Library** 來解析 .dot 檔案，並建立相對應的資料結構（建議使用繼承自 GameEntity 的子類別）。

**⚙️ 遊戲動作 (Actions)**

定義在 .xml 檔案中，用來描述特定動作，例如：

<triggers>

<keyphrase>chop</keyphrase>

<keyphrase>cut</keyphrase>

</triggers>

<subjects>

<entity>axe</entity>

<entity>tree</entity>

</subjects>

<consumed>

<entity>tree</entity>

</consumed>

<produced>

<entity>log</entity>

</produced>

<narration>You chopped down the tree!</narration>

使用 **JAXP (Java XML API)** 解析 .xml，轉換成內部資料結構。

**🤖 指令解析彈性（Command Parsing）**

你的 handleCommand() 必須具備自然語言處理的彈性，包括：

|  |  |
| --- | --- |
| **類型** | **解釋** |
| **大小寫無關** | GoTo, goto, GOTO 等效 |
| **裝飾詞容忍** | please get the axe 應解析為 get axe |
| **詞序彈性** | chop tree with axe 和 use axe to chop tree 皆可 |
| **部分指令支持** | unlock with key（只提供一個 subject）應能正確執行 |
| ❗ **禁止多餘實體** | open trapdoor with axe and key 若 axe 無效，則要拒絕執行 |
| ❗ **禁止含糊不清** | 若 open door with key 可對多個門執行，應中止並提示使用者更明確 |
| ❗ **禁止複合指令** | get key and open door 應被視為錯誤 |

**👥 多人玩家支援（Optional Extension）**

* 用戶端每次傳送指令都會附上玩家名稱，例如：

simon: get axe

* 每位玩家有自己的位置與物品欄。
* look 指令應列出同場景內的其他玩家。
* 無需登入或註冊流程，只要根據名稱建立新玩家即可。

**❤️ 玩家生命值（Optional Extension）**

* 玩家初始生命值為 3。
* 吃下毒物 / 被攻擊：生命值 -1。
* 吃補品 / 獲得幫助：生命值 +1（但最多為 3）。
* 死亡時掉落所有物品，生命值重置為 3，回到起始位置。

新增 health 指令顯示目前生命值。

**✅ 評分與檢查**

* 需符合：
  + GameServer 類別名稱與建構子格式。
  + 禁用語法規則（使用 strange 工具檢查）。
  + 測試檔案需能通過（不可修改 client 或測試程式碼）。
* 評分依據：
  + 功能正確性
  + 彈性與錯誤處理
  + 程式碼品質（結構、命名、可讀性）
  + 自主編寫（AI 或他人代碼將被視為 0 分）

**📌 小結建議順序**

1. ✅ 解析 entities.dot：建立遊戲場景與物品模型
2. ✅ 解析 actions.xml：建立動作清單與規則
3. ✅ 實作基本指令（look, get, drop, inv, goto）
4. ✅ 實作動作匹配邏輯（含觸發詞與條件）
5. ✅ 加入語言彈性處理（大小寫、裝飾詞、順序變化等）
6. ✅ 實作錯誤處理（模糊、多餘實體等）
7. 🔁 加強測試（JUnit 與範例檔）
8. ⚙️ 加入擴展功能（多玩家 / 生命值）👉 可留到最後再做

如果你想，我也可以幫你設計第一步驟：用 JPGD 解析 .dot 實體檔，建立 Location / Artefact 等 Java 類別。想開始的話告訴我！我們可以一步一步來 😎