CSI142 – Mini-Project Assignment

Project Title: OOP Sorting & Searching with Exceptions and Version Control

Start Date: 10th April

Duration: 3 Weeks (groups of 4–5 students)

---

1. Objective

Combine the **CSI142** concepts you've learned so far:

1. **Object-Oriented** programming (classes, inheritance/polymorphism, encapsulation)
2. **Sorting & Searching** (selection/insertion sorts, linear/binary searches)
3. **Exception Handling** (include at least one custom exception)
4. **Version Control** (GitHub or GitLab for collaborative development)

By the end, your group will create a **small console-based app** reflecting a real or fictional scenario, showcasing how these OOP elements come together in practice.

2. Requirements

1. **Domain & OOP Design**
   - Pick a **domain** relevant to Botswana, Southern Africa, or contemporary life (e.g., local produce store, NGO project, festival tickets, student roster, mobile data, etc.).
   - Create **at least two** domain classes (e.g., `Product`, `Student`, `DataPlan`), each with meaningful fields.
   - Demonstrate **encapsulation** (private fields, getters/setters) and **polymorphism** (an interface, abstract class, or small inheritance chain).

2. **Sorting & Searching**
   - Implement **two** sorts in a utility class (e.g., `Sorter`):
     - **Selection Sort**
     - **Insertion Sort**
     - Sort by a numeric field (score, price, etc.)
   - Implement **two** searches:
     - **Linear Search** for a string field (e.g., `name`), returning an index or `-1`.
     - **Binary Search** (iterative or recursive) for a numeric field, which requires data to be sorted first.

3. **Exception Handling**
   - Include **at least one custom exception** (e.g., `InvalidScoreException`, `NegativePriceException`), handling domain-specific errors.
   - Use `try-catch` or re-throw logic where appropriate (e.g., negative price, out-of-range score).

4. **Version Control & Group Collaboration**
   - Groups of **4–5** must collaborate via **GitHub** or **GitLab**.
   - Each member should commit code/documentation regularly, demonstrating genuine involvement.
   - Submit the repository link in your final Moodle submission.

5. **Running the Application**
    - Provide a **Main** class (`MainApp`) demonstrating how you:
        - Sort data (show before/after states).
        - Perform at least one linear and one binary search.
    - Measure performance for small vs. larger arrays to illustrate $O(n^2)$ vs. $O(\log n)$.

---

3. Structure & Timeline

- **Week 1** (10th April–16th April):
    1. Choose **domain** (e.g., a store, NGO tracker, sports teams).
    2. Draft domain classes with fields, getters, setters.
    3. Initialize **Git** repo, assign tasks, begin minimal code.

- **Week 2** (17th April–23rd April):
    1. Implement **selection** & **insertion** sorts.
    2. Implement **linear** & **binary** search.
    3. Integrate in an OOP style, commit frequently, test with small data sets.
    4. Add **custom exception** usage if not yet done.

- **Week 3** (24th April–30th April):
    1. Finalize code & thorough testing.
    2. Write a short **documentation** (1–2 pages).
    3. Submit code & repository link on Moodle by end of Week 3 (30th April).

---

4. Submission Details

1. **Deliverables**
    - **GitHub/GitLab Repo Link**: Show commit logs from all members.
    - **Source Code**: .java files (zipped or direct repository link).
    - **Short Documentation**:
        - Domain scenario & class structure
        - Sorting & searching approach
        - Custom exception usage
        - Instructions for running `MainApp`

2. **Evaluation Criteria**
    - **OOP** design (encapsulation, inheritance/polymorphism) – 20%
    - **Sorting** correctness (selection & insertion) – 20%
    - **Searching** correctness (linear & binary) – 20%
    - **Exception** handling (custom logic) – 15%
    - **Version Control** usage (collaboration, commit logs) – 15%
    - **Creativity & Clarity** (domain scenario, docs) – 10%

5. Tips & Encouragement

- Each group's **unique domain** ensures no two solutions are identical.
- You can store data in arrays or `ArrayList`, as long as you implement required sorts & searches.
- **Custom exceptions** vary widely (e.g., negative input, duplicates, out-of-range).
- Keep code **clean & modular**: domain classes separated from sorting/search utility.
- Communicate and commit often—**don't** finalize everything at the last minute.

Good luck—this mini-project helps you integrate **CSI142** concepts in a real or simulated environment.