HANOI UNIVERSITY OF SCIENCE AND TECHNOLOGY
SCHOOL OF INFORMATION AND COMMUNICATION TECHNOLOGY

OOP PROJECT REPORT

OBJECT ORIENTED PROGRAMMING
(IT3100E)

Vietnam History

Team members	*Email*
Vu Duc An 20215174	an.vd215174@sis.hust.edu.vn
Nguyen Tieu Phuong 20210692	phuong.nt210692@sis.hust.edu.vn
Nguyen Dac Tam 20210763	tam.nd210763@sis.hust.edu.vn
Nguyen Tran Nhat Quoc 20210726	quoc.ntn210726@sis.hust.edu.vn

Class
139407 - IT3100E
Lecturer
Trinh Tuan Dat
dat.trinhtuan@hust.edu.vn

July 21, 2023

Time period: 23^{nd}June - 10^{th}July

OOP PROJECT REPORT

Vietnam History

Class: 139407

Lecturer: Trinh Tuan Dat

Contents

I Statistics of Data Collected

We deploy a web crawler to gather data from a diverse range of websites, including notable sources like Wikipedia, Nguoikesu, and Google. Initially, the data we amass is vast in quantity. However, through meticulous cleaning and filtering processes, only a number of usable and refined data remained.

Data	Count	Attribute	Source	Description
Dynasty	14	id, name, existedTime, capital, kingName/listofKing		The dynasties that existed throughout Vietnam history
Event	71	id, name, time, destination, description, image, relatedPerson		Wars, resignation, and other historical occurrences
Festival	51	id, name, time, destination, description		Special celebrations of Vietnam
Person	1397	id, name, givenName, father, dateOfBirth, dateOfDeath, description, dynasty		Public figures, historical heroes, and person related to history
Relic	22	id, title, content, address, relatedPerson		The historical monuments and places alike

II UML Diagrams Design Explanation:

Some developers prefer to create UML diagrams before coding as part of the planning and design phase. However, we prefer to create UML diagrams after coding. This approach involves analyzing the existing codebase and generating UML diagrams to document the system's structure. It can be useful for understanding complex or legacy codebases and identifying patterns. It also helps us to come up with improvements and suggest refactoring if needed.

II.1 Package Diagram:

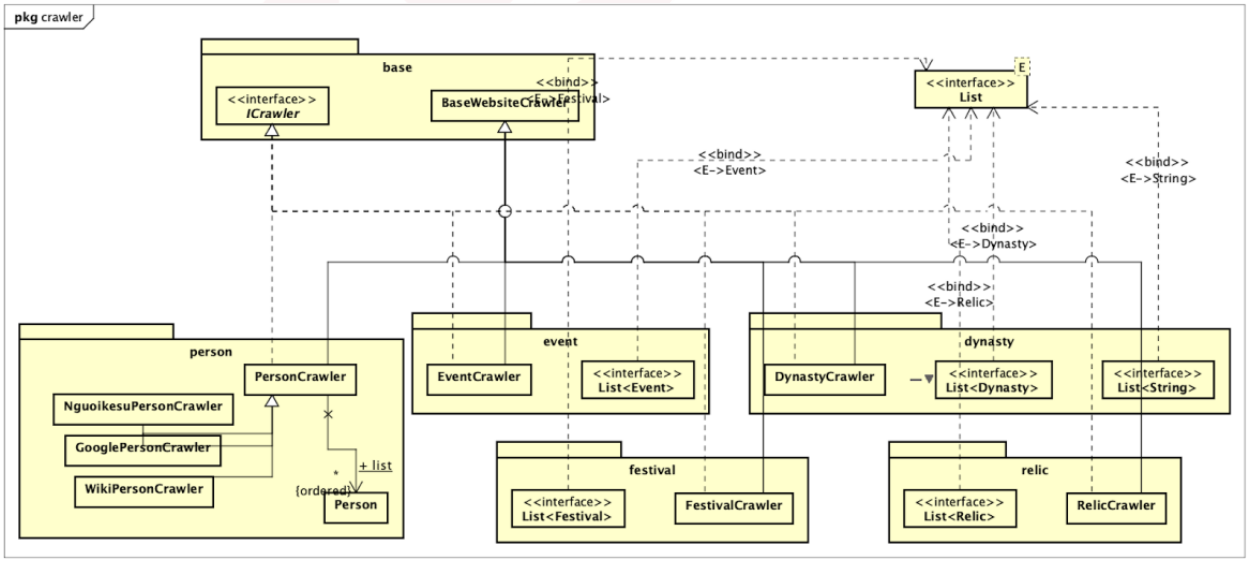

Figure 1: The package of crawlers. Here a certain crawler for an entity is grouped into its own package. The List interface in Java is also used.

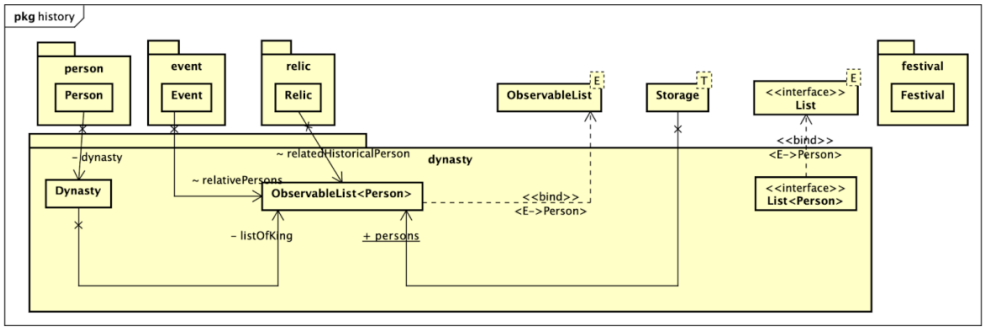

Figure 2: The package of classes for the entities. Each entity is in a package.

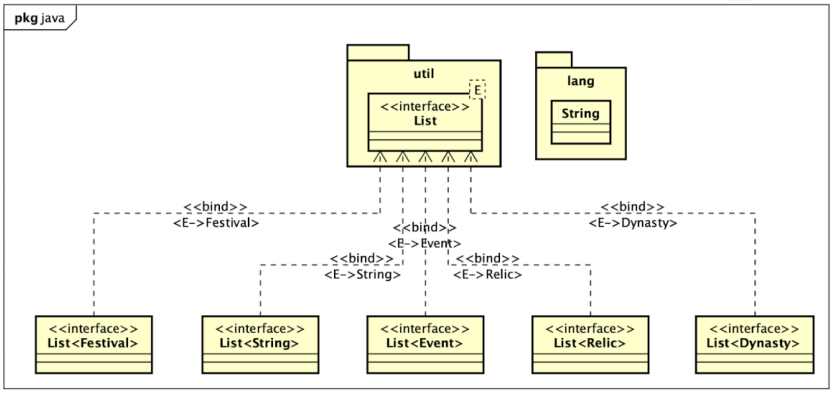

Figure 3: The package of the List Java built-in interface and class used.

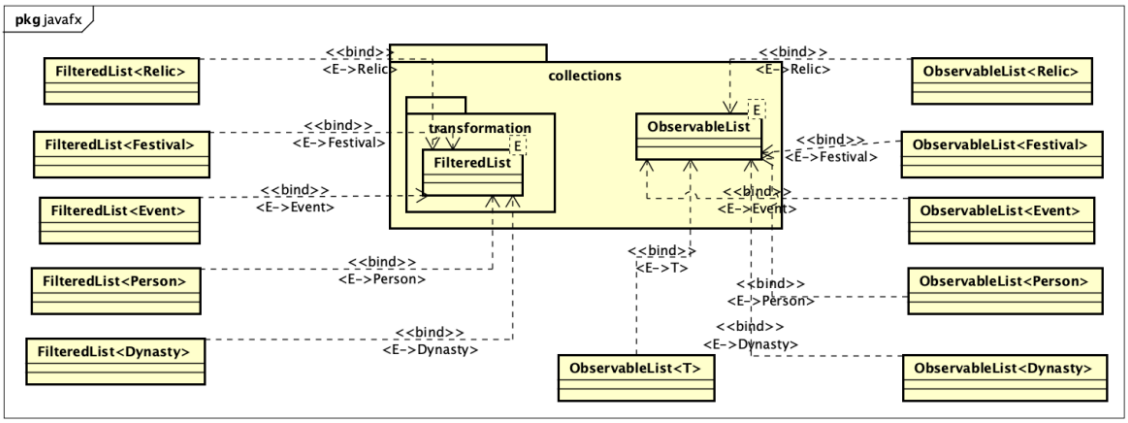

Figure 4: The package of JavaFX collections. The data, now encapsulated in objects, are passed to the ObservableLists. The FilteredList provides the wrapper around the source list.

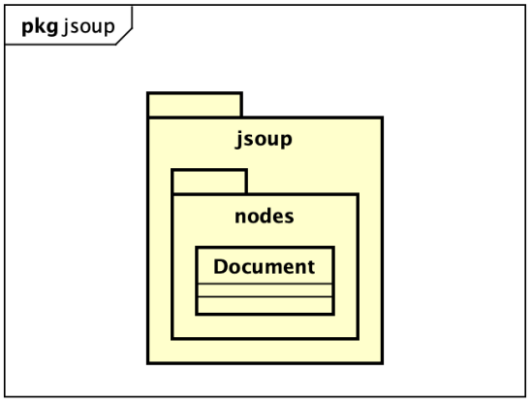

Figure 5: The package of Jsoup

II.2 Class Diagram

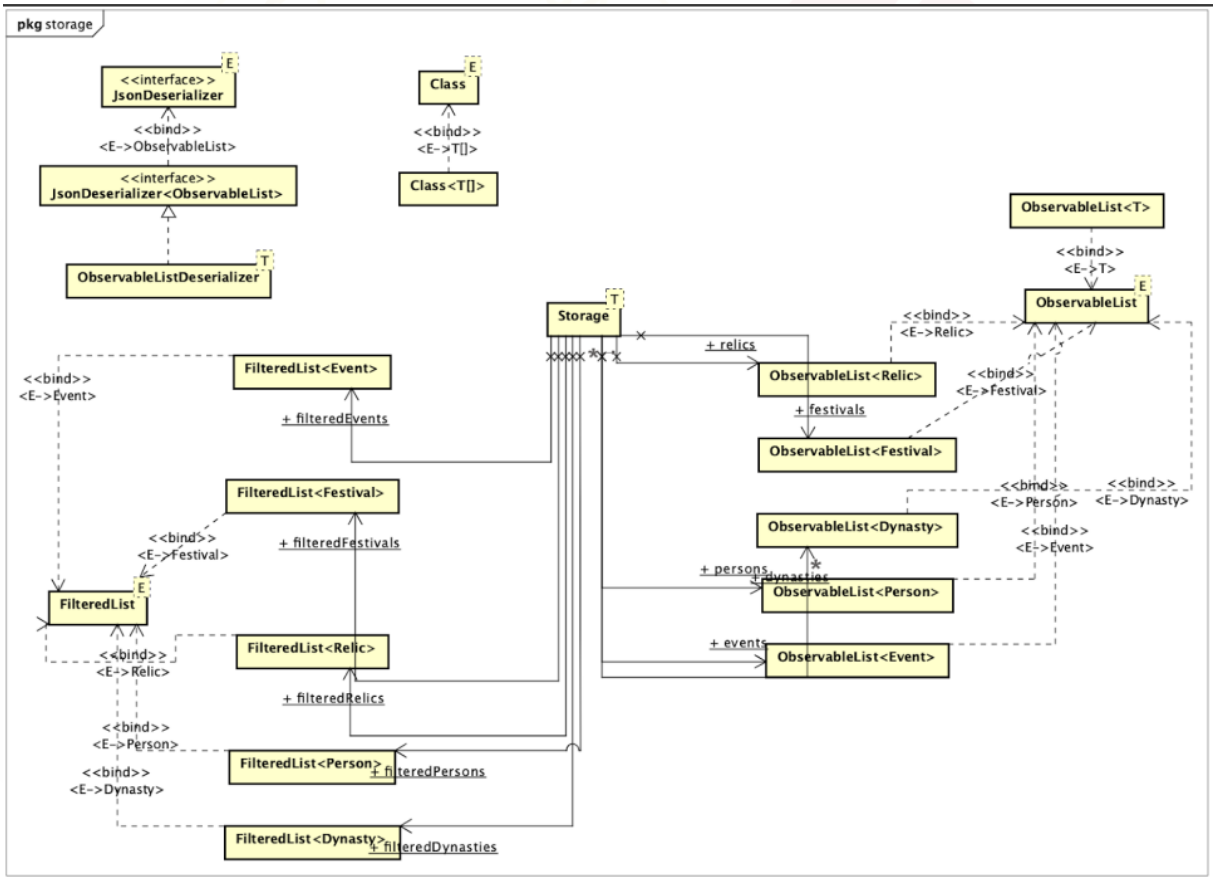

Figure 6: The class diagram of storage objects

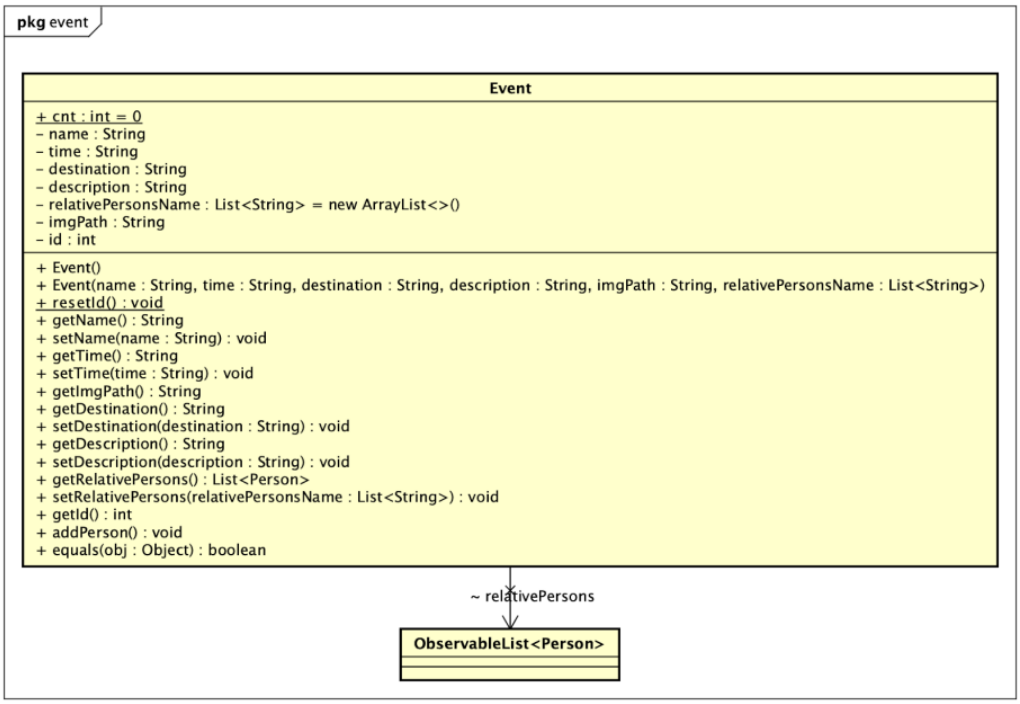

Figure 7: The class diagram of entity Event. This is a class that has only one relation to another entity (Person)

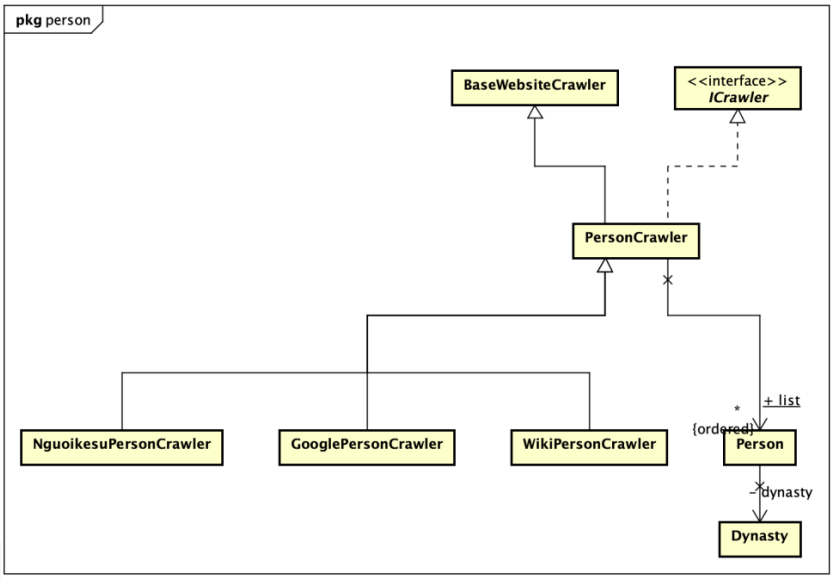

Figure 8: The class diagram of the Person Crawler. This is an abstract class that inherits from the BaseCrawler, and there are source-specific classes that inherit from it to implement different scraping methods for each source.

III Applied OOP Techniques:

Throughout the development of the project, we tried our best to apply the OOP techniques learned in class. Notably the 4 main principles of OOP are used:

III.1 Encapsulation:

This principle emphasizes the bundling of data and methods into a single unit called an object. What we do

- Assign each entity to a class of its own (Dynasty, Event, Relic,...);
- Assign the crawler of each entity to a class of its own (DynastyCrawler, 7EventCrawler, RelicCrawler...);
- Grouping the UI controllers for each entity. For each entity, the controllers for the detailed information and the list interface are also broken down into separate ones. (DynastyDetailController, EventDetailController, RelicDetailController...) and (DynastyListController, EventListController,...)

The benefits: By encapsulating related data and behavior together, we can achieve data hiding and protect the internal state of an object. This enhances security, modularity, and code reusability.

III.2 Inheritance

Inheritance allows us to create new classes based on existing classes, inheriting their attributes and behaviors. What we do:

- Implement an abstract class BaseCrawler that serves as the template for other crawlers. This class has attribute URL, crawl, write to json method that other crawlers based on. The crawlers for each entity inherits this base class.

The benefits: It promotes code reuse and enables the creation of hierarchical relationships between classes. With inheritance, we can build specialized classes that inherit common characteristics from a base class, reducing code duplication and improving maintainability.

III.3 Polymorphism

Polymorphism allows objects of dierent classes to be treated as objects of a common superclass. What we do:

- The crawlers override the method speciĺed in the abstract base class;
- In each entity class, constructor overloading is used extensively to allow for dierent cases of data (i.e. when only some data ĺeld/ attribute is available)

The benefits: It enables us to write code that can work with objects of multiple types, providing flexibility and extensibility. Polymorphism allows for method overriding, where a subclass can provide its own implementation of a method deĺned in its superclass, further enhancing code modularity and flexibility.

III.4 Abstraction:

Abstraction involves simplifying complex systems by breaking them down into manageable and understandable components. What we do

- Break down the essentials of the project:
  - An abstract class named "PersonCrawler" deĺnes an abstract method named "crawl". This method is implemented by other types of crawler, depending on web structure and base URL of other classes
  - “DetalBaseController” define method “initMainContent” that other classes(EventBaseController, RelicBaseController, FestivalBaseController, RelicBaseController, PersonBaseController) show detail content
  - The app, which contains the backend working of:
    * The crawlers
    * The objects
    * The data crawled
    * The storage structure
  - The GUI, which contains the frontend working of:
    * The controllers
    * CSS
    * The JavaFX and FXML Scene Builder

The benefits:

It allows us to focus on the essential features of an object or system while hiding unnecessary details. Abstraction helps in designing modular and maintainable code, as well as in creating reusable components.

When the user interacts with our app, they do not need to know the underlying structure of the program or understand how everything works.

Furthermore, other notable characteristics of our project are the application of some common design patterns, as well as the tech stack used

IV Design Pattern:

IV.1 Factory Pattern

BaseWebsiteCrawler works as a factory for other types of constructor. Benefits:

The Factory Pattern encapsulates object creation, provides an abstraction layer, promotes flexibility and code reuse. It separates the object creation process from the client code and allows for changes in the creation process without aecting other classes.

IV.2 Strategy Pattern:

In class Storage, to handle search functionality for dierent categories of data or dierent algorithms. In our project, we implement 5 methods of search(searchDynasty, searchFestival, searchPerson, searchRelic, searchEvent) based on string matching.

IV.3 Singleton Pattern:

To ensure that only one instance of each crawler is running at a time, we implement the Singleton Pattern by making each crawler class a singleton.

The beneĺt of the Singleton Pattern to prevent conflicts between multiple instances of the same crawler running simultaneously, reduces the memory footprint of the application, as there is only one instance of each crawler class in memory at any given time.

V Tech Stack:

1. **Maven:** The library classes added through Maven help to synchronize code (avoid conflicts between versions or IDEs).
2. **Java Coding Convention:** Classes, packages, variable names, and functions are named according to the standard.
3. Build Class and package diagrams using a consistent UML modeling language in **Astah**.
4. **Jsoup**: A specialized library for data collection through web pages, added through Maven.
5. Build a GUI interface with **JavaFX Scene Builder**.
6. **Gson**: A Google library for Java used for reading or writing data with .json files to objects.
7. **Github**: Managing source code through Github.

VI User Guide

To get started, clone the source code from this GitHub repository . Make sure you have Java and associated softwares mentioned in the Tech Stack V.

Navigate to the src folder of the code. From the App.java Íle, run the main() method. Now you can interact with the app via GUI.