Higher Nationals - Summative Assignment Feedback Form

Student Name/ID	MOHAMMED MAHROOF MOHAMMED AASHIK/E230667		
Unit Title	Unit 20: Applied Programming and Design Principles		
Assignment Number	1	**Assessor**	
Submission Date		**Date Received 1st submission**	
Re-submission Date		**Date Received 2nd submission**	
Assessor Feedback:			
Grade:	**Assessor Signature:**	**Date:**	
Resubmission Feedback: *Please note resubmission feedback is focussed only on the resubmitted work			
Grade:	**Assessor Signature:**	**Date:**	
Internal Verifier's Comments:			
Signature & Date:			

* Please note that grade decisions are provisional. They are only confirmed once internal and external moderation has taken place and grades decisions have been agreed at the assessment board.

Important Points:

1. It is strictly prohibited to use textboxes to add texts in the assignments, except for the compulsory information. eg: Figures, tables of comparison etc. Adding text boxes in the body except for the before mentioned compulsory information will result in rejection of your work.
2. Avoid using page borders in your assignment body.
3. Carefully check the hand in date and the instructions given in the assignment. Late submissions will not be accepted.
4. Ensure that you give yourself enough time to complete the assignment by the due date.
5. Excuses of any nature will not be accepted for failure to hand in the work on time.
6. You must take responsibility for managing your own time effectively.
7. If you are unable to hand in your assignment on time and have valid reasons such as illness, you may apply (in writing) for an extension.
8. Failure to achieve at least PASS criteria will result in a REFERRAL grade .
9. Non-submission of work without valid reasons will lead to an automatic RE FERRAL. You will then be asked to complete an alternative assignment.
10. If you use other people's work or ideas in your assignment, reference them properly using HARVARD referencing system to avoid plagiarism. You have to provide both in-text citation and a reference list.
11. If you are proven to be guilty of plagiarism or any academic misconduct, your grade could be reduced to A REFERRAL or at worst you could be expelled from the course
12. Use word processing application spell check and grammar check function to help editing your assignment.
13. Use **footer function in the word processor to insert Your Name, Subject, Assignment No, and Page Number on each pag**e. This is useful if individual sheets become detached for any reason.

STUDENT ASSESSMENT SUBMISSION AND DECLARATION

When submitting evidence for assessment, each student must sign a declaration confirming that the work is their own.

Student name: MOHAMMED AASHIK	Assessor name:

Issue date:	Submission date: **28.07.2025**	Submitted on:
Programme**:** Pearson BTEC HND in Computing		
Unit: Unit 20: Applied Programming and design principles		
Assignment number and title: **1 | Implementation of a Dataset Processing Application for Dream Book Shop using SOLID Design Principles**		

Plagiarism

Plagiarism is a particular form of cheating. Plagiarism must be avoided at all costs and students who break the rules, however innocently, may be penalised. It is your responsibility to ensure that you understand correct referencing practices. As a university level student, you are expected to use appropriate references throughout and keep carefully detailed notes of all your sources of materials for material you have used in your work, including any material downloaded from the Internet. Please consult the relevant unit lecturer or your course tutor if you need any further advice.

Guidelines for incorporating AI-generated content into assignments:

The use of AI-generated tools to enhance intellectual development is permitted; nevertheless, submitted work must be original. It is not acceptable to pass off AI-generated work as your own.

Student Declaration

Student declaration
I certify that the assignment submission is entirely my own work and I fully understand the consequences of plagiarism. I understand that making a false declaration is a form of malpractice.

Student signature: E230667 Date:

Turnitin Similarity Report

Plagiarism Percentage	
Assessor Comments on Turnitin report, Report content, and Grade	

Unit 20: Applied Programming and Design Principles

Assignment Brief

Student Name/ID Number	MOHAMMED MAHROOF MOHAMMED AASHIK/E230667
Unit Number and Title	**Unit 20: Applied Programming and Design Principles**
Academic Year	2024/25
Unit Tutor	
Assignment Title	**Implementation of a Data Analysis Application for Dream Book Shop using SOLID Design Principles**
Issue Date	
Submission Date	

Submission Format

The assignment submission is in the form of the following:

A formal 10-minute presentation (with supporting speaker notes) to investigate the impact of SOLID development principles on the OOP. You will support your presentation with speaker notes containing extra information that you cannot put in your slides. Your presentation should make extensive use of real-world examples

A written design document that includes analysis of OOP characteristics, clean coding techniques, a range of design patterns and testing regime used for the software solution.

A program report that will demonstrate the actual implementation process of the application with results of the data analysis, along with meaningful charts or visualizations**.** You should also include the source code, with comments explaining each part.

A testing report that performs automatic testing on a data processing application.

You are required to make use of headings, paragraphs and sub-sections as appropriate, and all work must be supported with research and referenced using the Harvard referencing system (or an alternative system).

There is no strict word count requirement for the written components.

All these reports should be submitted in the final submission as a single file.

Please note that to pass the module, all deliverables must be completed.

Unit Learning Outcomes

LO1 Investigate the impact of SOLID development principles on the OOP paradigm.

LO2 Design a large dataset processing application using SOLID principles and clean coding techniques.

LO3 Build a data processing application based on a developed design .

LO4 Perform automatic testing on a data processing application.

Transferable skills and competencies developed

Computing-related cognitive skills

• Computational thinking (including its relevance to everyday life)
• Understanding of the scientific method and its applications to problem-solving
• Demonstrate knowledge and understanding of essential facts, concepts, principles and theories relating to computing and computer applications
• Use such knowledge and understanding in the modelling and design of computer based systems for the purposes of comprehension, communication, prediction and the understanding of trade-offs.
• Recognise and analyse criteria and specifications appropriate to specific problems, and plan strategies for their solution
• Analyse the extent to which a computer-based system meets the criteria defined for its current use and future development.
• Deploy appropriate theory, practices and tools for the design, implementation and evaluation of computer-based systems

Computing-related practical skills:

- The ability to specify, design and construct reliable, secure and usable computer-based systems.
- The ability to evaluate systems in terms of quality attributes and possible trade-offs presented within the given problem
- The ability to plan and manage projects to deliver computing systems within constraints of requirements, timescale and budget
- The ability to deploy effectively the tools used for the construction and documentation of computer applications, with particular emphasis on understanding the whole process involved in the effective deployment of computers to solve practical problems.
- The ability to critically evaluate and analyse complex problems, including those with incomplete information, and devise appropriate solutions, within the constraints of a budget.

Generic skills for employability:

- Critical thinking; making a case; numeracy and literacy
- Self-awareness and reflection; goal-setting and action-planning; independence and adaptability; acting on initiative; innovation and creativity
- Interaction: reflection and communication
- Contextual awareness e.g. the ability to understand and meet the needs of individuals, business and the community, and to understand how workplaces and organisations are governed.

Learning Outcomes and Assessment Criteria

Pass	Merit	Distinction

Vocational scenario.

You have recently joined AQ Digital Solutions Limited (AQDS) as a Junior Software Developer. AQDS delivers software design, development, and consultancy services across a wide variety of industry sectors. Their core services include:

- Designing custom software solutions for industry clients.
- Implementing applications using multiple programming paradigms.
- Providing technical consultancy on software optimization and maintenance.

As part of your initial assignment, you have been entrusted with a solo project for a new client, **Dream Book Shop**, a local comic-book retailer. After over 20 years of modest trading, the shop has recently expanded its physical premises and significantly grown its eCommerce platform. To improve their user experience and digital engagement, Dream Book Shop is ultimately aiming to develop an interactive data visualization dashboard. However, as an initial step in this larger goal, you have been tasked with developing a **Command-Line Interface (CLI) data analysis application** that provides insightful textual and basic visual summaries based on historical comic book metadata. This prototype will serve as a proof of concept before moving toward a full-fledged dashboard solution.

The shop has acquired a publicly available dataset from **WorkWithData**, which provides bibliographic information on books published in the British National Bibliography (BNB). The dataset contains over 2.6 million entries and includes attributes such as the book title, author name, publisher, ISBN, language of publication, and publication year. Each record is uniquely identified by a BNB ID, making it suitable for data-driven analysis. The dataset contains a number of .CSV (comma separated value) data files. The supplied dataset can be freely downloaded from this location: https://www.workwithdata.com/datasets/books. As the original dataset is quite large, it is recommended to limit the data to no more than 5,000 rows or else you can refer to the dataset provided in the **zipped folder** for your analysis.

Dream Book Shop wishes to explore how data analytics and visual representation can support better decision-making, reader engagement, and storytelling around comic book history through this application.

As part of this project, you are expected to:

- Investigate how SOLID design principles can be applied to the OOP paradigm. Your investigation should cover:
    - OOP paradigm characteristics
    - object orientated class relationships
    - a range of design patterns
    - clean coding techniques

- SOLID design principles.

- Design and develop a CLI data analysis application using Object-Oriented Programming (OOP) principles applied with SOLID design principles. Any OOP language of your choice (e.g. C#, Java, VB.NET, C++, Objective-C, Delphi, Python) can be used for this.

- The application should be able to analyze the dataset and output clear textual summaries and visualizations via bar charts, line graphs, pie charts etc. The application should support the following analyses:
  - Publication Trends Over Time (Count of books published per year)
  - Top 5 Most Prolific Authors (Authors with the highest number of books)
  - Language Distribution of books
  - Number of books published by each publisher
  - Missing ISBN Analysis (Count and percentage of records without an ISBN.)
  - Number of books published per year categorized by language.
- Demonstrate that you can perform automated testing on the application.

  The testing should:
  - show the different types of automatic testing
  - show a range of tool automation parameters
  - follow common testing frameworks and methodologies
  - investigate a range of commercial and self-built testing tools.

Assignment activity and guidance:

Activity 1

You will create a formal presentation for AQDS and Dream Book Shop that will investigate the impact of SOLID development principles on the OOP paradigm to a mixture of a technical and non-technical audience. You will support your presentation with speaker notes containing extra information that you cannot put in your slides. You should make extensive use of examples to support your presentation. Your presentation should:

● Investigate the characteristics of the object-orientated programming, including:
- OOP Paradigm characteristics
- Class relationships
- SOLID principles

●Explain how clean coding techniques can impact on the use of data structures and operations when writing algorithms

● Analyse, with examples, each of the following pattern types
  o Creational design patterns
  o Structural design patterns
  o Behavioural design patterns

Finally, your presentation should evaluate the impact of SOLID development principles on object-orientated application development.

Activity 02

You are to produce a written design document that contains the design for a large dataset processing application using SOLID principles and clean coding techniques. Your document should include the following:

- Dream Book Shop scenario in context.
- Both functional and non-functional requirements of the application
- UML class diagrams to illustrate the system
- Object-oriented programming (OOP) principles applied in the design.
- Use of SOLID principles in the application.
- At least one or more relevant design patterns used in the design.
- Justification of clean coding techniques
- An appropriate testing strategy, including discussion of frameworks, tools, and provisions for automated testing.

Activity 03

Once you have designed the application for the Dream Book Shop you should build a data processing application using the OOP language of your choice (e.g. C#, Java, VB.NET, C++, Objective-C, Delphi, Python). You should then include the following in your program report.

- Present annotated code snippets in the appropriate programming language (e.g., .py, .java, .cs) to illustrate key parts of the implementation.

- Textual summaries and visualizations of the following analyses.
    - Publication Trends Over Time (Count of books published per year)
    - Top 5 Most Prolific Authors (Authors with the highest number of books)
    - Language Distribution of books
    - Number of books published by each publisher
    - Missing ISBN Analysis (Count and percentage of records without an ISBN.)
    - Number of books published per year categorized by language

- An assessment of the effectiveness of using the following on the application developed:
    - SOLID principles
    - Clean coding techniques
    - Programming patterns.

Activity 4

Perform testing on your implemented data processing application to validate its functionality, reliability, and compliance with user requirements through an automated testing approach.

Address the following in the testing report:

- Explain a detailed test plan, including test objectives, scenarios, and coverage areas.

- Execute automated tests on the application and present evidence of the testing process and results.
- Analyze the benefits and drawbacks of different forms of automated testing in the context of applications and software systems, using examples from your own application.
- Compare developer-built testing tools with vendor-provided tools, discussing their suitability and effectiveness.
- Assess the advantages and limitations of each automated testing method used in the process.
- Evaluate the overall effectiveness of your testing approach and suggest practical improvements for future development and testing phases

You should support any points you make in the report with well-chosen examples from the existing scenario

Recommended resources.

Please note this is not a definitive list of resources but it will help you begin your research by acting as a starting point of reference

Websites

https://dev.to/favourmark05/ (2023) Writing Clean Code: Best Practices and Principles [online].
https://katalon.com/ (n.d.) Software Test Automation Frameworks | 6 Common Types [online]. Available at: https://katalon.com/resources-center/blog/test-automation-framework [Accessed 10 June 2024]
https://refactoring.guru/design-patterns (n.d.) The Catalog of Design Patterns [online]. Available at: https://refactoring.guru/design-patterns/catalog [Accessed 10 June 2024]
https://www.geeksforgeeks.org/ (2022) Object Oriented Programming (OOPs) Concept in Java [online]. Available at: https://www.geeksforgeeks.org/object-oriented-programming-oops concept-in-java/ [Accessed 10 June 2024]
SOLID Principles in Programming: Understand With Real Life Examples [online]. Available at: https://www.geeksforgeeks.org/solid-principle-in programming-understand-with-real-life-examples/ [Accessed 10 June 2024]
https://www.geeksforgeeks.org (2024) 7 Tips To Write Clean And Better Code in 2024 [online]. Available at: https://www.geeksforgeeks.org/tips-to-write-clean-and-better-code/ [Accessed 10 June 2024]
https://www.simplilearn.com (2023) Types of Automated Testing: Everything You Need to Know [online]. Available at: https://www.simplilearn.com/types-of-automated-testing-article [Accessed 10 June 2024]
Available at: https://www.tutorialspoint.com/design_pattern/design_pattern_overview.htm [Accessed 10 June 2024]
https://www.tutorialsteacher.com (n.d.) Class Relations: Association and Composition [online]. Available at: https://www.tutorialsteacher.com/csharp/association-and-composition [Accessed 10 June 2024]

Journals and articles

Aktaş, A. Z., Yağderelĭ, E. and Serdaroğlu, D., 2021. An introduction to software testing methodologies. Gazi University Journal of Science Part A: Engineering and Innovation, 8(1), pp.1–15. Available at: https://dergipark.org.tr/tr/download/article-file/634522 [Accessed 10 June 2024]

Coad, P., 1992. Object-oriented patterns. Communications of the ACM, 35(9), pp.152–159. Available at: https://dl.acm.org/doi/10.1145/130994.131006 [Accessed 10 June 2024]

Chebanyuk, E. and Markov, K., 2016, February. An approach to class diagrams verification according to SOLID design principles. In 2016 4th International Conference on Model-Driven Engineering and Software Development (MODELSWARD) (pp. 435-441). IEEE. Available at: https://ieeexplore.ieee.org/abstract/document/7954391 [Accessed 10 June 2024]

Hunter-Zinck H, de Siqueira AF, Vásquez VN, Barnes R, Martinez CC. Ten simple rules on writing clean and reliable open-source scientific software. PLoS Comput Biol. 2021 Nov 11;17(11):e1009481.Availableat: https://journals.plos.org/ploscompbiol/article?id=10.1371/journal.pcbi.1009481 [Accessed 10 June 2024]

Goyal, G. and Patel, S., 2012. Importance of inheritance and interface in OOP paradigm measure through coupling metrics. Int. J. Appl. Inf, 4, pp.14-20. ISSN: 2249-0868. Available at: https://research.ijais.org/volume4/number9/ijais12-450781.pdf [Accessed 10 June 2024]

Jamil, M. A., Arif, M., Abubakar, N. S. A. and Ahmad, A., 2016, November. Software testing techniques: A literature review. In 2016 6th international conference on information and communication technology for the Muslim world (ICT4M) (pp. 177-182). IEEE. Available at: https://ieeexplore.ieee.org/document/7814898 [Accessed 10 June 2024]

Korson, T. and McGregor, J. D., 1990. Understanding object-oriented: A unifying paradigm. Communications of the ACM, 33(9), pp.40-60. Available at: https://dl.acm.org/doi/10.1145/83880.84459 [Accessed 10 June 2024]

Ljung, K. and Gonzalez-Huerta, J., 2022, November. “To Clean Code or Not to Clean Code” A Survey Among Practitioners. In International Conference on Product-Focused Software Process Improvement (pp. 298–315). Cham: Springer International Publishing. Available at: https://doi.org/10.1007/978-3-031-21388-5_21 [Accessed 10 June 2024]

Noble, J., 1998, November. Classifying relationships between object-oriented design patterns. In Proceedings 1998 Australian software engineering conference (cat. no. 98ex233) (pp. 98–107). IEEE. Available at: https://ieeexplore.ieee.org/document/730917. [Accessed 10 June 2024]

R. Subburaj, Jekese, G. & Hwata, C. (2015) Impact of Object Oriented Design Patterns in Software Development. International Journal of Scientific and Engineering Research. Available at: https://www.researchgate.net/publication/273460797_Impact_of_Object_Oriented_Design_Patterns_in_Software_Development [Accessed 10 June 2024]

Singh, H. and Hassan, S.I., 2015. Effect of solid design principles on quality of software: An empirical assessment. International Journal of Scientific & Engineering Research, 6(4), pp.1321–1324. ISSN 2229-5518 Available at: https://www.ijser.org/researchpaper/Effect-of SOLID-Design-Principles-on-Quality-of-Software-An-Empirical-Assessment.pdf [Accessed 10 June 2024]

Surya, M. and Padmavathi, S., 2019. A Survey of Object-Oriented Programming Languages. Available at: https://ijsrcseit.com/index.php/home/article/view/CSEIT243647 [Accessed 10 June 2024]

Umar, M. A. and Zhanfang, C., 2019. A study of automated software testing: Automation tools and frameworks. International Journal of Computer Science Engineering (IJCSE), 6, pp.217–225. Available at: http://www.ijcse.net/docs/IJCSE19-08-06-011.pdf [Accessed 10 June 2024]

Xinogalos, S., 2015. Object Oriented Design and Programming: an Investigation of Novices’ Conceptions on Objects and Classes. ACM Transactions on Computing Education, 15(3). DOI: 10.1145/2700519. Available at: https://dl.acm.org/doi/10.1145/2700519 [Accessed 10 June 2024]

Yu, L., Li, Y. & Ramaswamy, S., 2017. Design Patterns and Design Quality: Theoretical Analysis,Empirical Study, and User Experience. International Journal of Secure Software Engineering, 8(2). Available at: https://www.igi-global.com/gateway/article/190421 [Accessed 10 June 2024]

Textbooks

Alsmadi, I. ed., 2012. Advanced Automated Software Testing: Frameworks for Refined Practice. IGI Global.

Ammann, P. and Offutt, J., 2016. Introduction to Software Testing. Cambridge University Press.
Anaya, M., 2021. Clean Code in Python: Develop maintainable and efficient code.
Packt Publishing Ltd.
Gamma, E., Helm, R., Johnson, R. & Vlissides, J. (1995) Design patterns: Elements of Reusable Object-Oriented Software. 1 ed. Addison-Wesley.
Hamill, P., 2004. Unit Test Frameworks: tools for high-quality software development.
O'Reilly Media, Inc.
Martin, R. C., 2011. The Clean Coder: a code of conduct for professional programmers.
Pearson Education.
McLaughlin, B., Pollice, G. & West, D. (2006) Head First Object-Oriented Analysis and Design: A Brain Friendly Guide to OOA&D. 1 ed. O'Reilly Media.

HN Global
Reading Lists. Available at:
https://hnglobal.highernationals.com/learning-zone/reading-lists
Student Resource Library. Available at:
https://hnglobal.highernationals.com/subjects/resource-libraries

<table>
<tr><td colspan="2">LO1 Investigate the impact of SOLID development principles on the OOP paradigm</td><td rowspan="5">LO1 and LO2

D1 Evaluate the impact of SOLID development principles on object-orientated application development</td></tr>
<tr><td>P1 Investigate the characteristics of the object-orientated paradigm, including class relationships and SOLID principles.

P2 Explain how clean coding techniques can impact on the use of data structures and operations when writing algorithms</td><td>M1 Analyse, with examples, each of the creational, structural and behavioural design pattern types.</td></tr>
<tr><td colspan="2">LO2 Design a large dataset processing application using SOLID principles and clean coding techniques</td></tr>
<tr><td>P3 Design a large data set processing application, utilising SOLID principles, clean coding techniques and a design pattern.

P4 Design a suitable testing regime for the application, including provision for automated testing.</td><td>M2 Refine the design to include multiple design patterns.</td></tr>
<tr><td colspan="2">LO3 Build a data processing application based on a developed design</td><td rowspan="3">LO3 & LO4

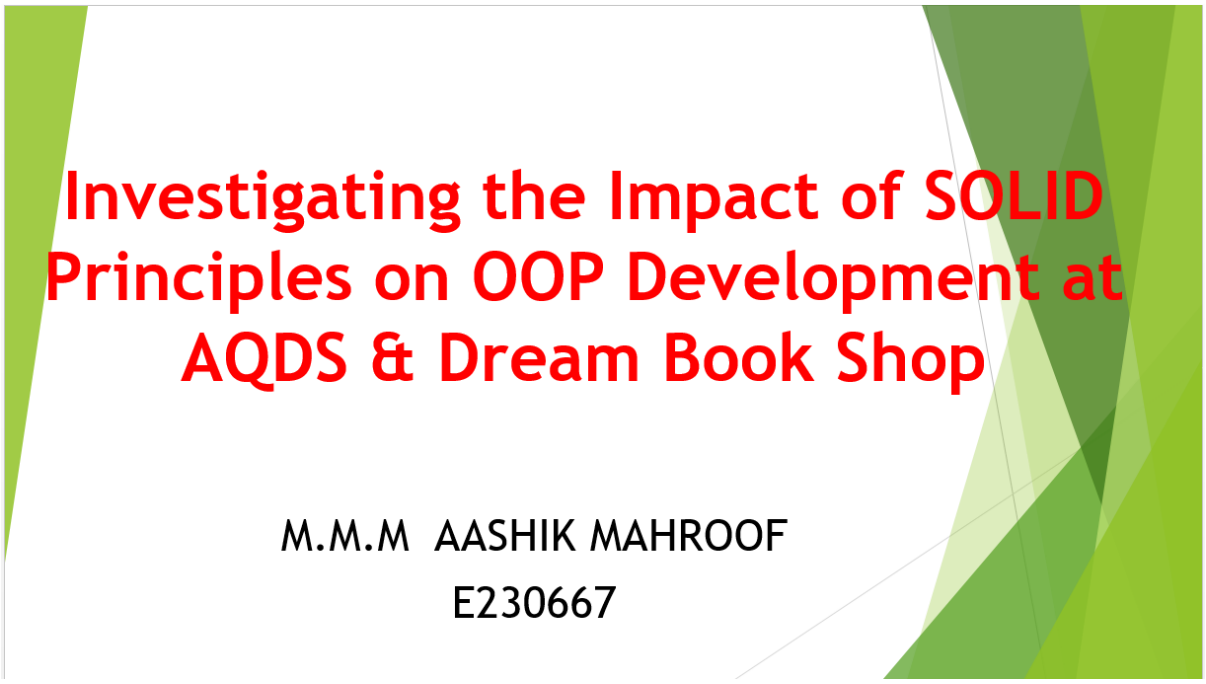
D2 Analyse the benefits and drawbacks of different forms of automatic testing of applications and software systems, with examples from the developed application</td></tr>
<tr><td>P5 Build a large dataset processing application based on the design produced.</td><td>M3 Assess the effectiveness of using SOLID principles, clean coding techniques and programming patterns on the application developed.</td></tr>
<tr><td colspan="2">LO4 Perform automatic testing on a data processing application</td></tr>
</table>

P6 Examine the different methods of implementing automatic testing as designed in the test plan. **P7** Implement automatic testing of the developed application.	**M4** Discuss the differences between developer-produced and vendor-provided automatic testing tools for applications and software systems	

ACKNOWLEDGEMENT

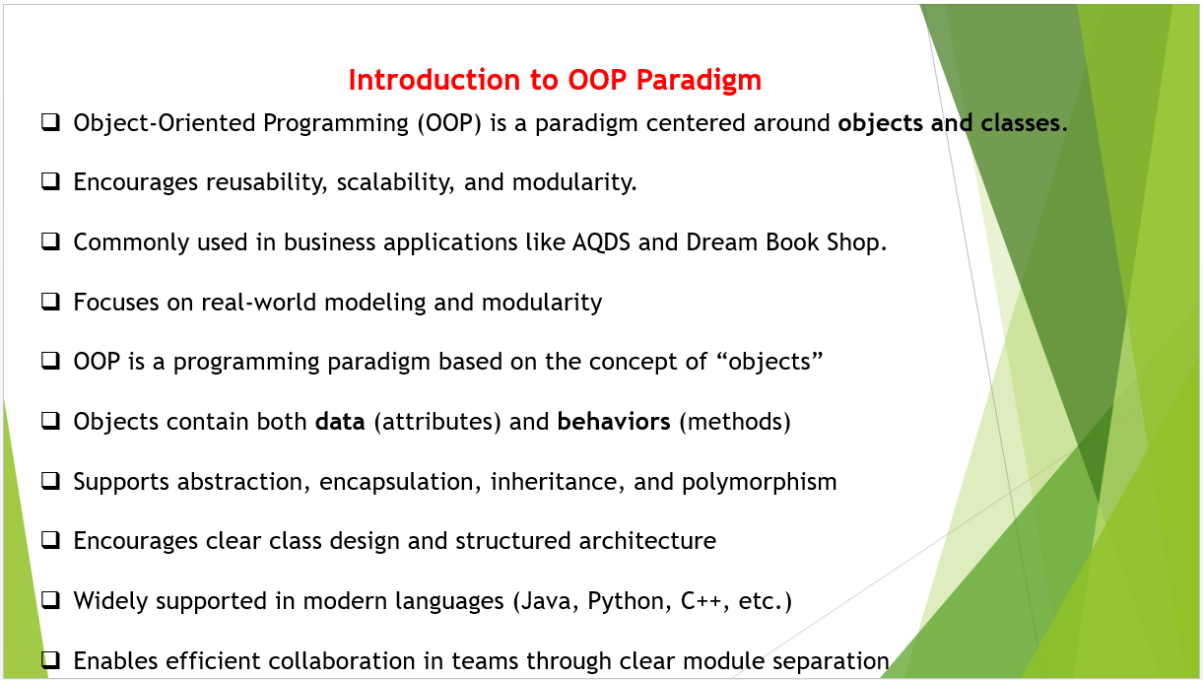
I am deeply grateful for the assistance and guidance I received from numerous esteemed individuals, which was instrumental in the successful completion of my task. I would like to express my sincere appreciation to ESOFT for providing a conducive workspace that facilitated the completion of my task. I am delighted to announce the successful completion of the assignment. I am particularly indebted to **Mrs.kausalya** for his invaluable guidance throughout my fourth semester assignments. Lastly, I extend my heartfelt gratitude to my family members and classmates whose unwavering support greatly contributed to the timely completion of this project. Thank you all for your immense contribution!

Activity 01



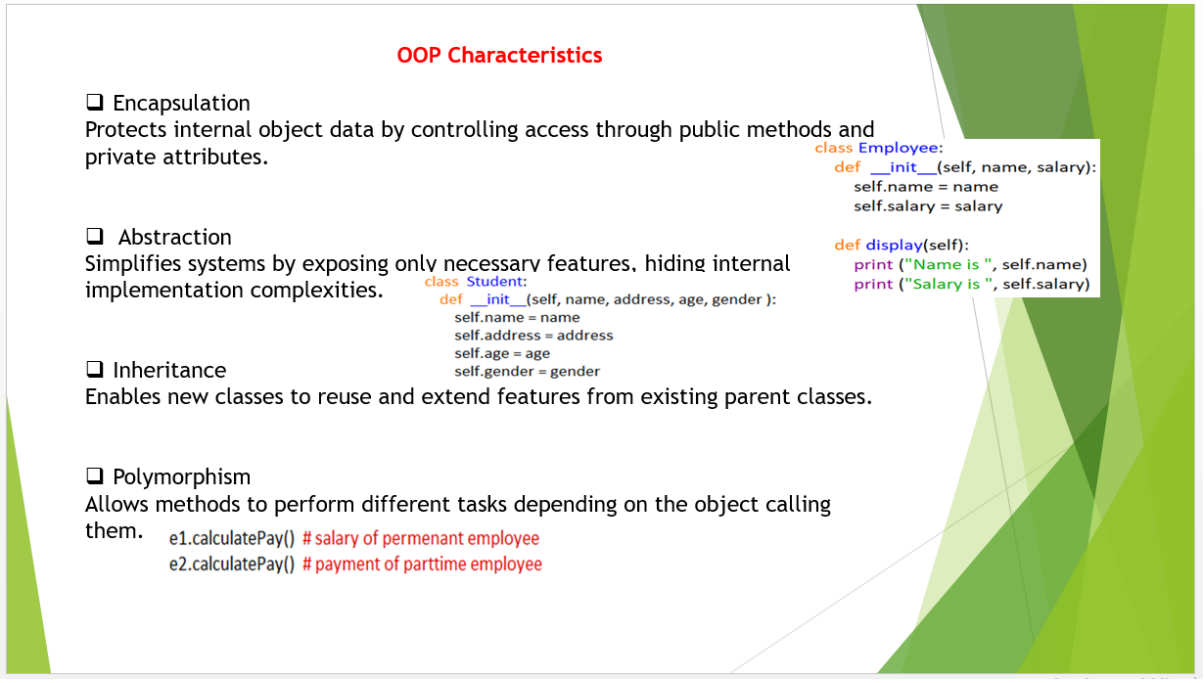
Speaker Notes:

Welcome everyone. Today, we're exploring how SOLID principles influence object-oriented programming (OOP), especially in the context of software development at AQDS and Dream Book Shop. OOP has long been a popular paradigm in building scalable, maintainable software. It is built around the concept of "objects" combining data and behaviors. However, as systems grow, maintaining clean, reusable code becomes challenging. That's where the SOLID principles come in. They provide a set of design guidelines that help developers avoid common pitfalls such as tight coupling, code duplication, and poor scalability. This presentation is crafted for both technical and non-technical stakeholders.



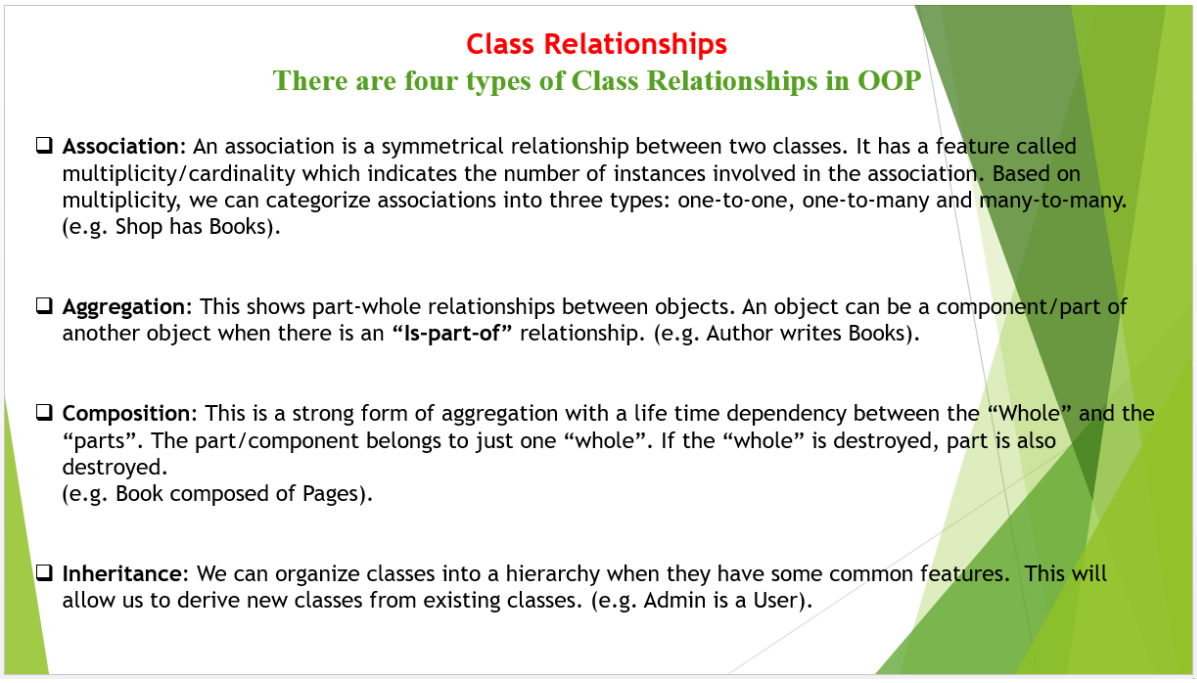
Speaker notes :

Object-Oriented Programming (OOP) is a popular paradigm that structures code around “objects” units combining both data and behavior. It supports core principles like encapsulation, inheritance, abstraction, and polymorphism, which improve clarity and reusability. This approach makes code more scalable, easier to maintain, and ideal for modular development. In business systems like AQDS and Dream Book Shop, OOP helps simulate real-world processes and encourages efficient teamwork through well-defined class boundaries and responsibilities. It's used widely in languages like Java and Python.



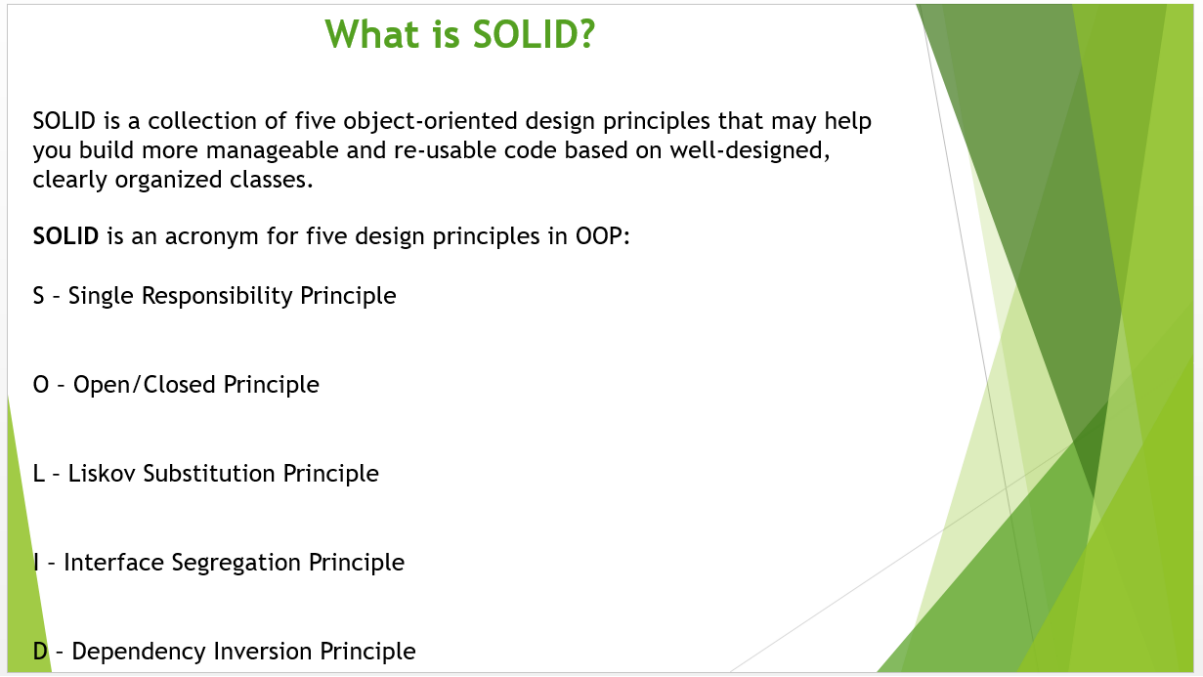
Speaker notes :

In this section, let's look at what defines the Object-Oriented Programming paradigm. OOP revolves around four fundamental principles: encapsulation, inheritance, abstraction, and polymorphism. Encapsulation means wrapping data and methods into a single unit usually a class so internal implementation is hidden. Inheritance allows a class to acquire properties and behaviors from another, promoting code reusability. Abstraction simplifies complex systems by modeling classes based on real-world objects, showing only essential features. Lastly, polymorphism allows the same operation to behave differently depending on the context think method overloading or overriding.



Speaker Notes:

Understanding class relationships is critical in OOP. These relationships define how classes interact, and they fall into categories like association, aggregation, composition, and inheritance. **Association** is a broad term indicating any relationship between two classes such as a customer placing an order. **Aggregation** is a "has-a" relationship where objects are loosely connected. For instance, a library has books, but if the library closes, the books still exist. **Composition** is a stronger relationship; if the container is destroyed, its parts also cease to exist like a house and its rooms. **Inheritance**, as we saw earlier, is an "is-a" relationship, where a subclass extends a superclass.

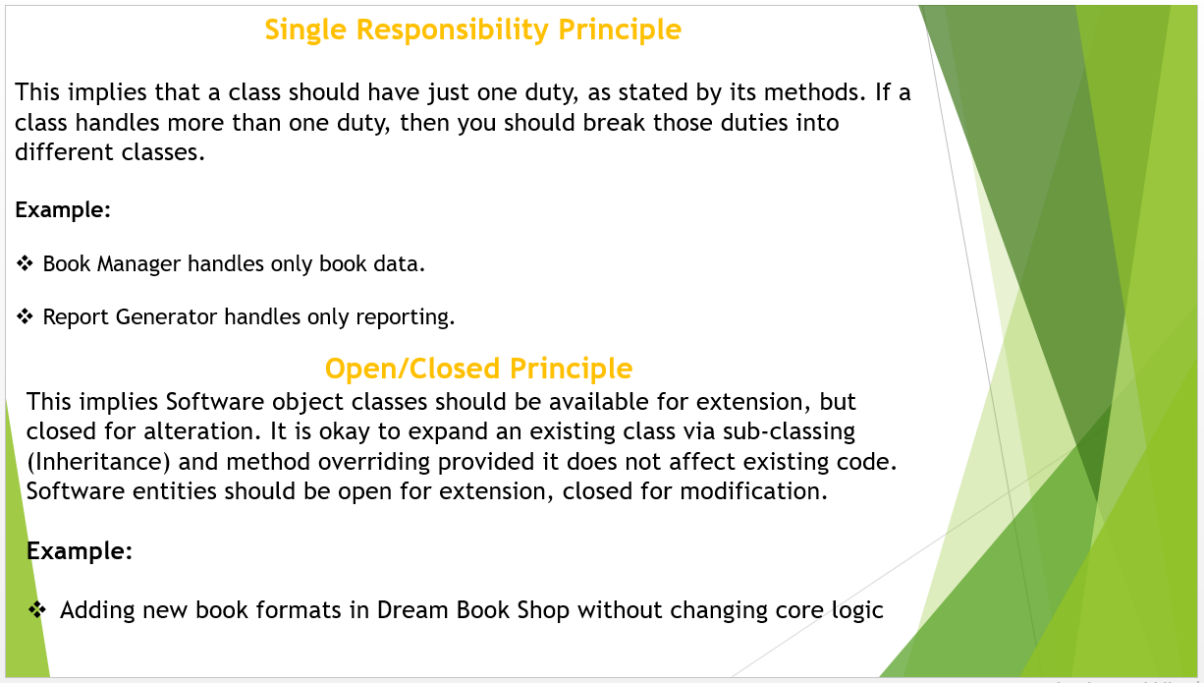


Speaker Notes:

Let's dive into the core SOLID principles. Each letter stands for a key object-oriented design guideline:

1. **S** – Single Responsibility Principle: a class should do one thing only.
2. **O** – Open/Closed Principle: software entities should be open for extension but closed for modification.
3. **L** – Liskov Substitution Principle: subclasses must be substitutable for their base classes.
4. **I** – Interface Segregation Principle: don't force classes to implement interfaces they don't use.
5. **D** – Dependency Inversion Principle: depend on abstractions, not concrete implementations.

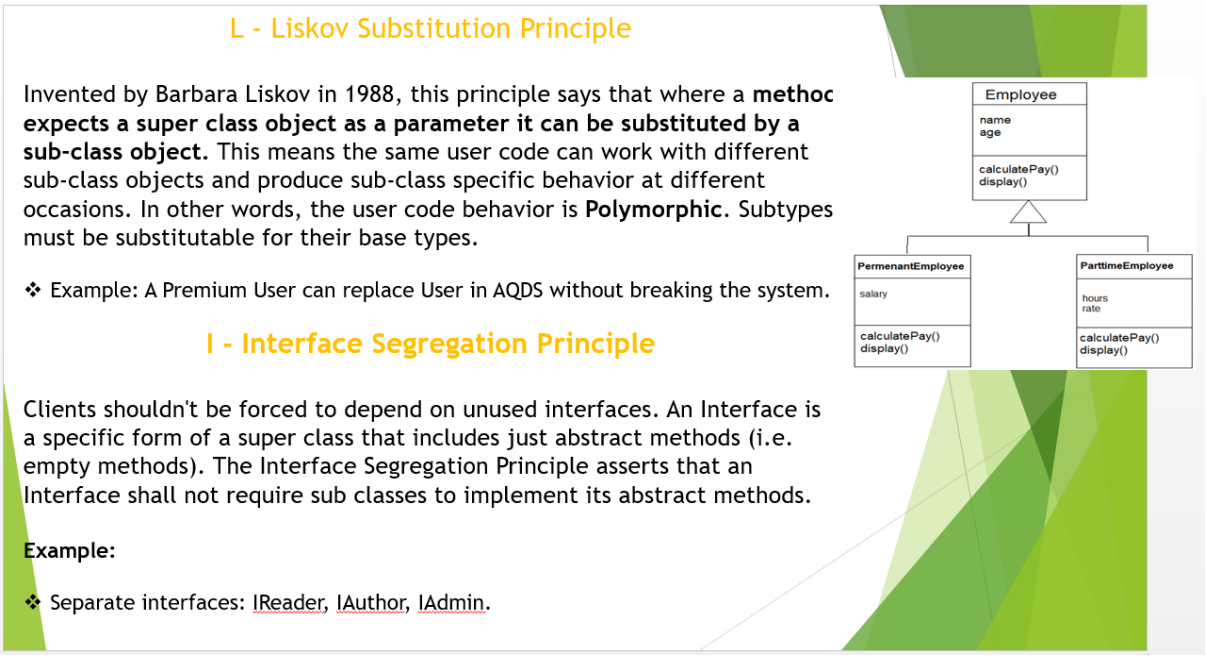
Applying these principles helps developers build software that is easier to test, maintain, and extend. Take the Dream Book Shop system. If the BookManager class only manages book-related tasks (SRP), we reduce errors and make changes faster.



Speaker Notes

The Single Responsibility Principle means a class should have one and only one job. When a class tries to do too much, it becomes harder to maintain, test, and reuse. By giving each class a focused responsibility, the code becomes more modular and easier to manage. For example, in Dream Book Shop, the BookManager class should handle only book-related tasks, while ReportGenerator should deal only with reports. This clear separation improves readability and reduces the risk of introducing bugs during updates.

The Open/Closed Principle promotes software flexibility. It suggests that classes should be open for extension but closed for modification. That means you can add new functionality by extending existing code (like using inheritance or interfaces), without changing the original, tested code. This avoids breaking current functionality. In Dream Book Shop, for instance, new book formats like audiobooks can be added by extending a base class, instead of rewriting core systems. This approach supports long-term maintenance and feature scaling.



Speaker Notes

The Liskov Substitution Principle, introduced by Barbara Liskov in 1988, emphasizes that objects of a subclass should be replaceable for objects of the superclass without affecting the program's correctness. This principle strengthens polymorphism, allowing flexible and maintainable code. In the context of AQDS, a PremiumUser object should seamlessly replace a general User object in any method expecting the superclass. This ensures consistent behavior across various user types, enabling code reusability and system reliability while reducing the need for rewriting logic for each subclass.

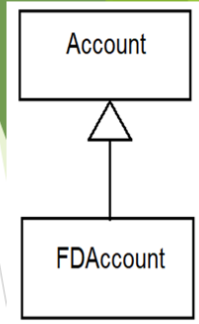
The Interface Segregation Principle states that no class should be forced to implement methods it does not use. Instead of having one large interface with multiple responsibilities, it is better to break it down into smaller, more specific ones. This leads to cleaner, more maintainable code. For example, in Dream Book Shop, instead of one big interface for users, we can create separate ones like IReader for readers and IAuthor for writers. Each class then only implements the methods relevant to its role, reducing code bloat and confusion.

Dependency Inversion Principle

High-level modules shouldn't depend on low-level modules. Both should depend on abstractions. This principle says that **classes that provide details** must **depend on abstractions** and not the other way around. This implies that our design must start with **Interface classes** or **Abstract classes.**

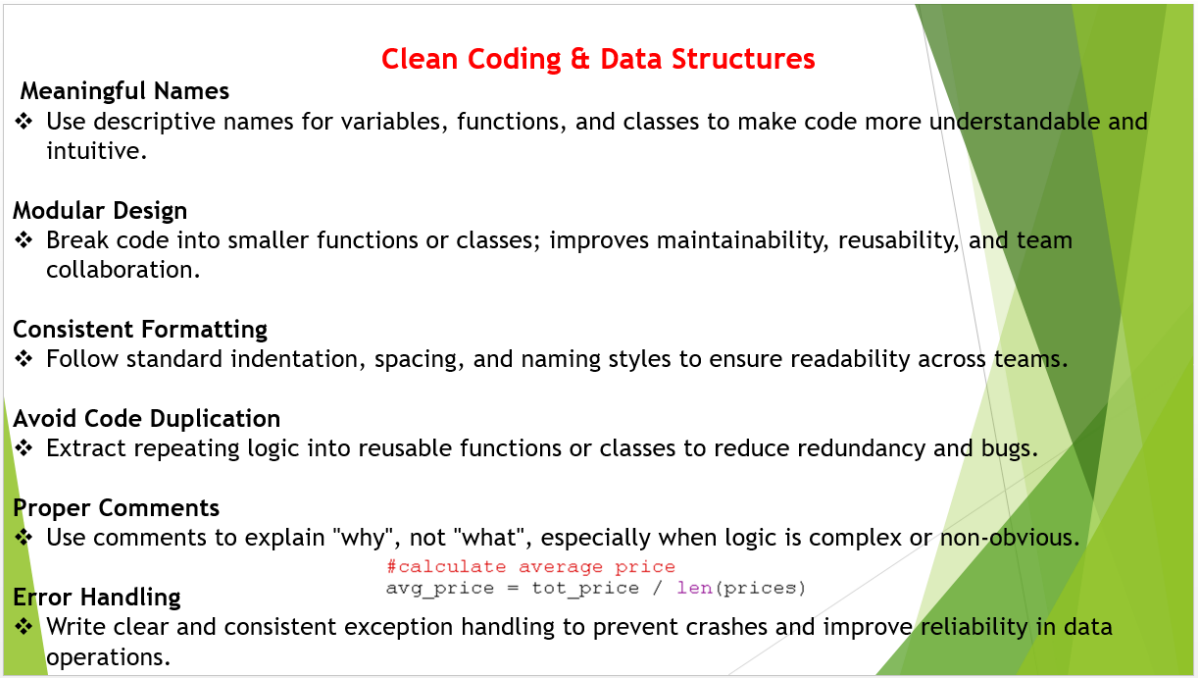
Example:

- FDAccount depends on the abstraction "Account" and its abstract method calculate Interest().



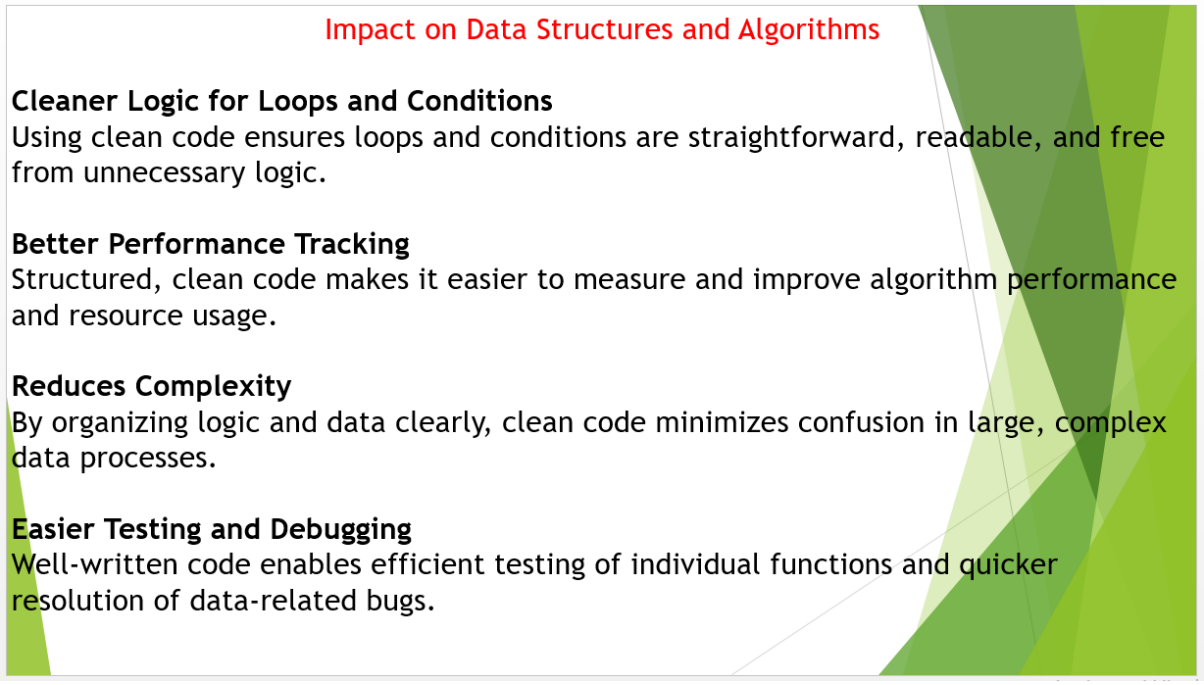
Speaker notes :

The Dependency Inversion Principle states that high-level modules should not rely on low-level modules; both should rely on abstractions. This helps reduce tight coupling between components and improves system flexibility and scalability. By designing with abstract classes or interfaces first, we ensure that changes in low-level details won't affect high-level logic. For example, in AQDS, a FDAccount class can depend on an abstract Account interface with a calculateInterest() method making it easy to switch or extend account types without changing core logic.



Speaker notes :

Clean coding is more than just writing code that works it's about writing code that is understandable, maintainable, and extendable. When working with data structures and algorithms, clean coding becomes crucial. For example, choosing the correct data structure like using a dictionary for fast lookup instead of a list can drastically improve performance. But beyond selection, how we name variables, break down functions, and structure loops affects readability and bug detection. In the context of Dream Book Shop, if we implement a book search algorithm, clean coding ensures that even junior developers can understand how it works, modify it, or add features later. Clean code also encourages separation of concerns and modularization, which aligns well with SOLID principles.



Speaker Notes

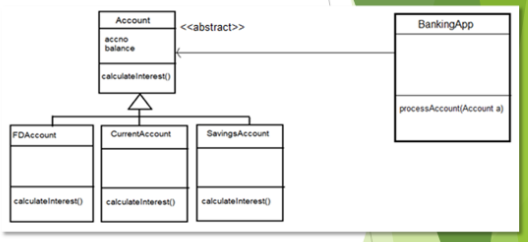
Clean coding practices play a vital role in handling data structures and algorithms effectively. When loops and conditions are written clearly, they become easier to read and understand, reducing confusion. A well-organized codebase also helps track performance more efficiently and spot areas needing improvement. Clean code lowers overall complexity, especially in large data operations. It also makes testing and debugging much easier. Modular functions allow you to test parts of the code individually, leading to fewer errors and better system reliability over time.

Creational Design Patterns

In software engineering, creational design patterns are design patterns that deal with object creations. They try to create objects in a manner suitable to the problem situation. Creational design patterns control object creations. They encapsulate knowledge about what objects they create

Example Patterns:

i. **Abstract Factory Method Design Patterns**

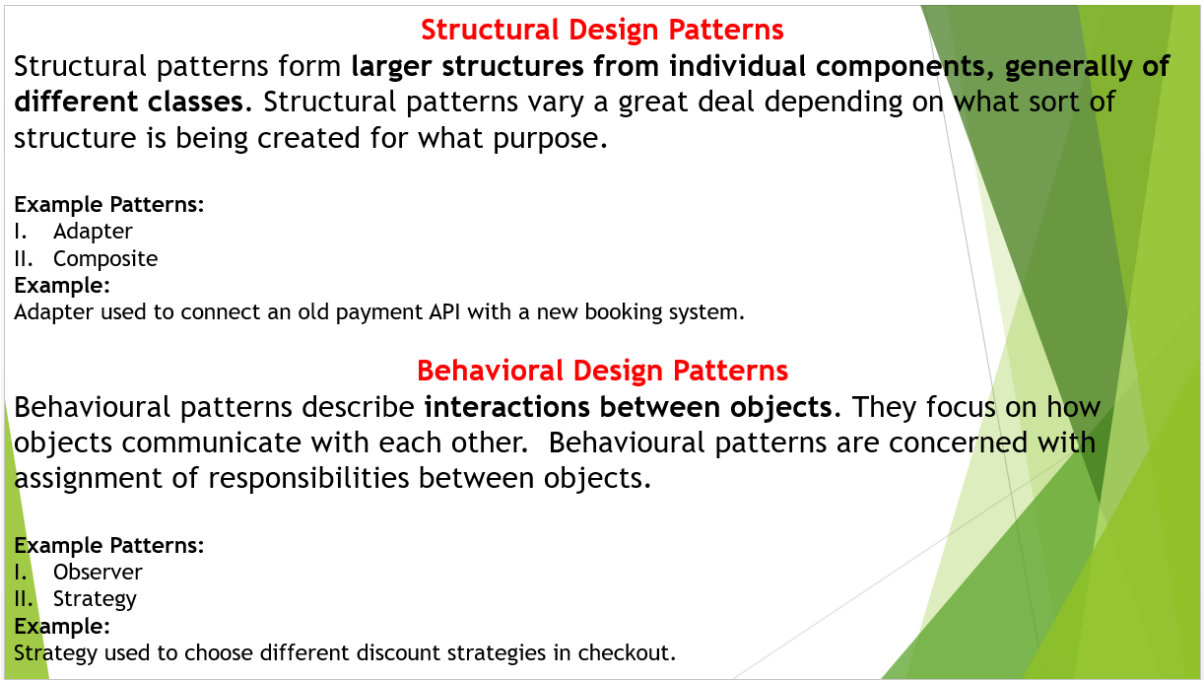


ii. **Singleton Method Design Pattern**

```
class President:
  count = 0
  def __init__(self, name, age):
    if (President.count == 0):
      self.name = name
      self.age = age
      President.count = President.count+1
    else:
      print ("Sorry - President already exists")
      self.name = ""
      self.age = ""

  def display(self):
    print (self.name)
    print (self.age)
```

Speaker notes :

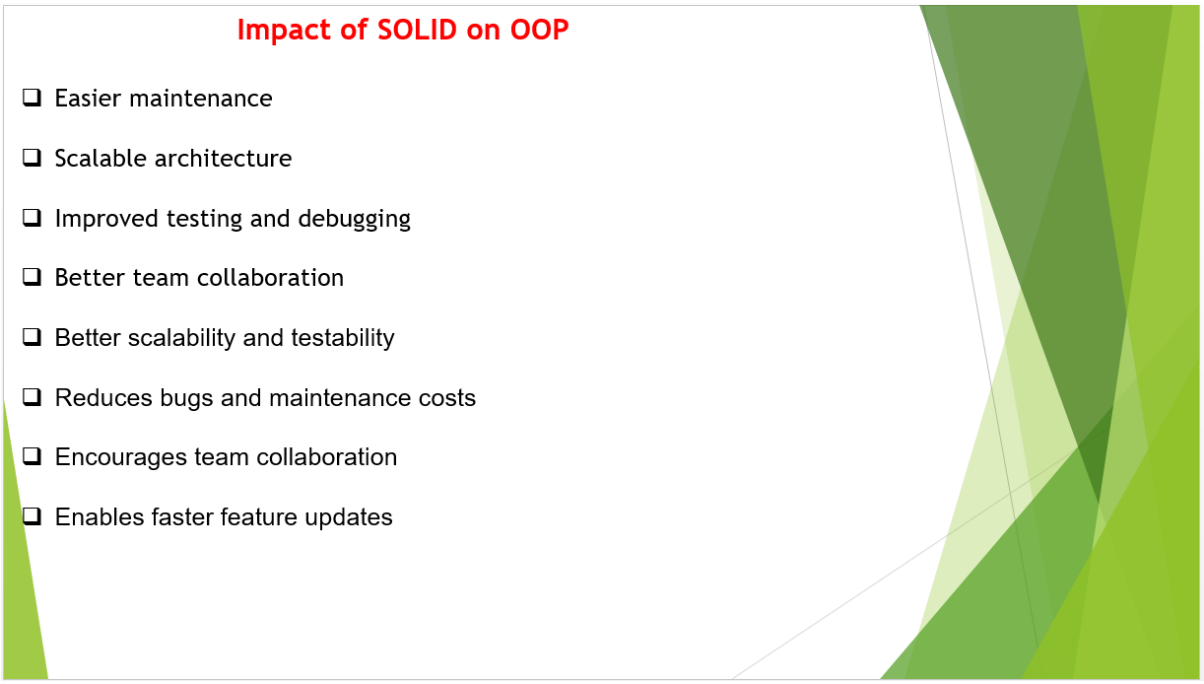
Creational Design Patterns In software engineering, creational design patterns focus on how objects are created. These patterns provide flexible and efficient ways to instantiate objects depending on the needs of a particular situation. Instead of directly calling constructors, creational patterns encapsulate the creation logic, allowing the system to be more modular and adaptable. They essentially control the object creation process by hiding the complexities involved. Some well-known examples of creational patterns include the Abstract Factory and Singleton patterns. The Abstract Factory pattern allows creating families of related or dependent objects without specifying their concrete classes. On the other hand, the Singleton pattern ensures that a class has only one instance throughout the application and provides a global point of access to it.



Speaker notes :

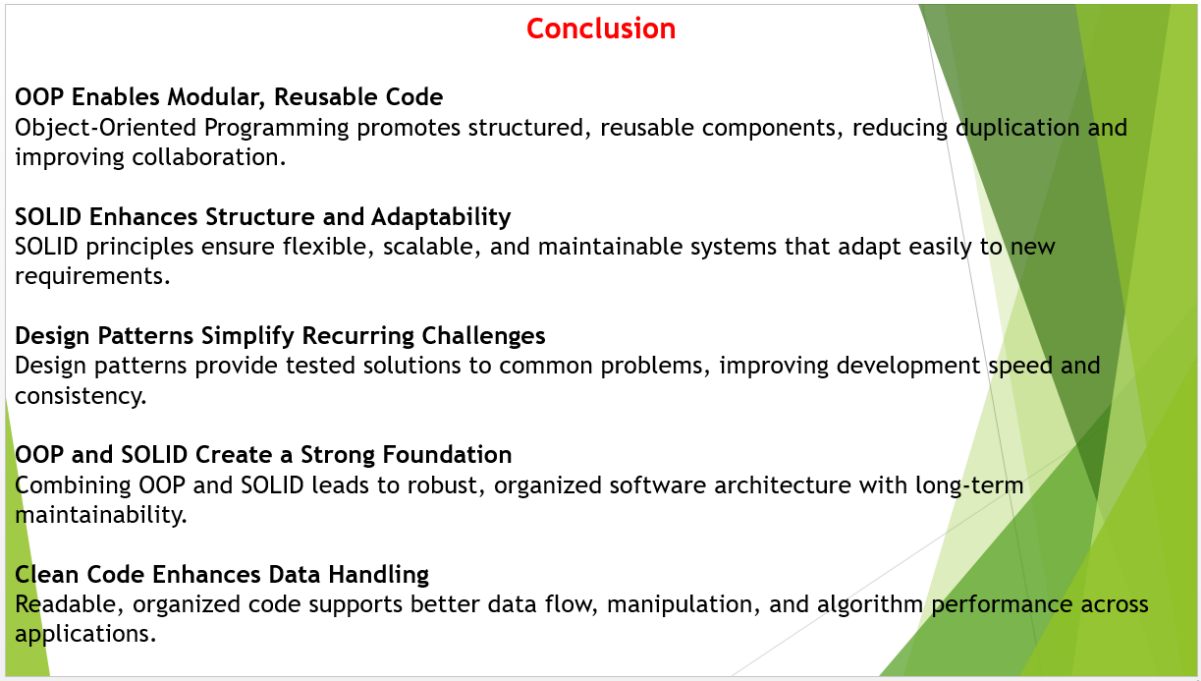
Behavioral patterns deal with the interaction between objects and how responsibilities are distributed among them. They define the communication protocols between objects and help manage complex control flows and object collaborations within a system. Examples of behavioral patterns are Observer and Strategy. For instance, the Strategy pattern enables an application to select among various algorithms or behaviors at runtime. A practical use of the Strategy pattern is in an e-commerce checkout system where different discount calculation methods can be applied depending on the user's status or promotional campaigns.

Structural design patterns focus on composing classes and objects into larger structures while keeping them flexible and efficient. They help organize relationships between entities, often combining multiple objects to form a new functionality or interface. Examples include the Adapter and Composite patterns. The Adapter pattern is particularly useful when integrating new systems with legacy components. For example, if a new booking system needs to connect with an outdated payment API, the Adapter pattern can be used to make the two systems compatible without modifying their existing codebases.



Speaker notes :

SOLID principles improve object-oriented programming by making code easier to maintain and scale. They enhance testing and debugging, reduce bugs, and lower maintenance costs. Additionally, SOLID promotes better team collaboration and enables faster feature updates, resulting in a more robust and adaptable software architecture.



Speaker notes

Object-Oriented Programming (OOP) encourages building modular and reusable code components, which helps reduce duplication and fosters better teamwork. The SOLID principles strengthen this approach by making software systems more flexible, scalable, and easier to maintain as requirements change. Design patterns offer proven solutions for recurring design problems, speeding up development and ensuring consistency across projects. Together, OOP and SOLID create a solid foundation for developing well-organized and long-lasting software architectures. Finally, writing clean and readable code improves data management and algorithm efficiency, leading to better overall application performance.

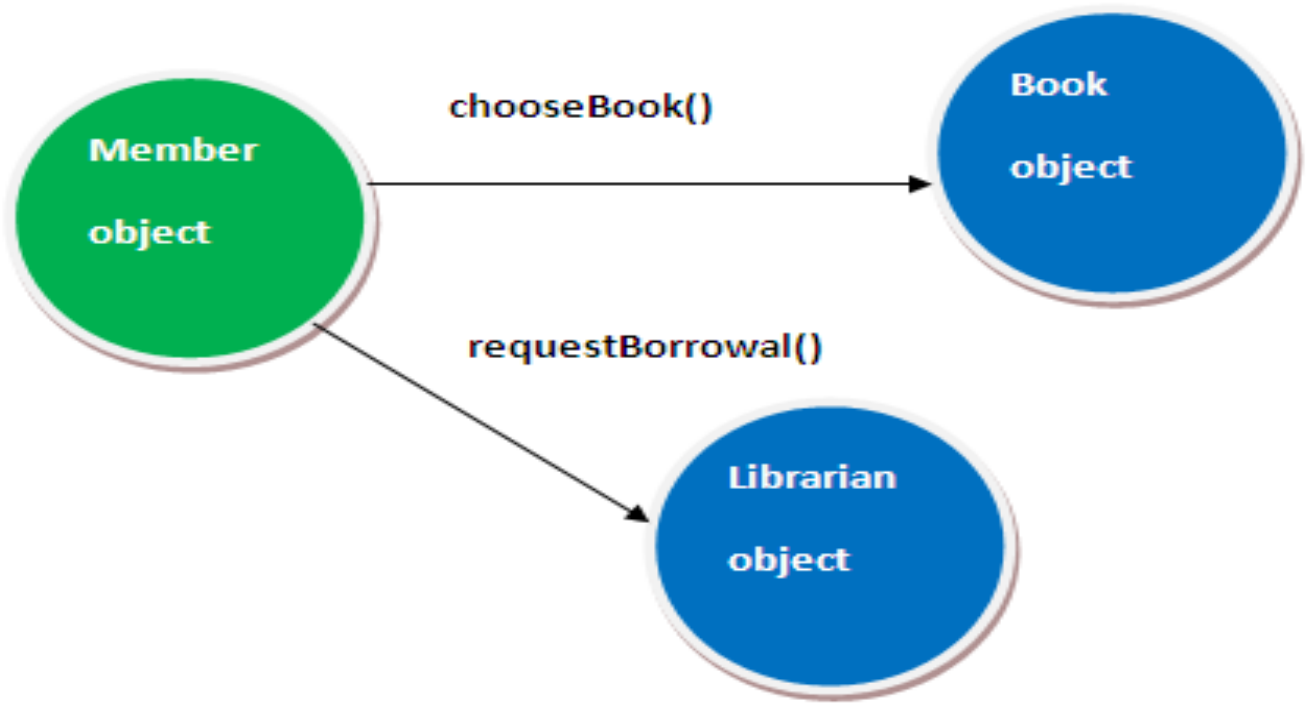
1.1. CHARACTERISTICS OF OOP

OOP (Object Oriented Programming)

In the world around us we see objects that communicate with each other (e.g. remote controller sending signal to the TV). OOP paradigm simulates the real world in software.

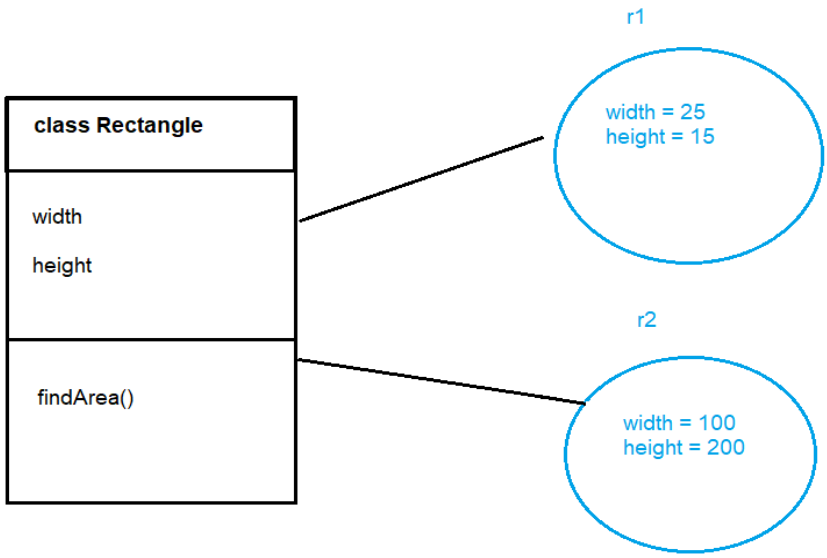
OOP is about building computer systems by assembling software components called objects. Objects encapsulate data and operations together. These objects communicate with each other at run time.

Object oriented programming (OOP) is a programming paradigm that develops computer systems in terms of communicating objects.



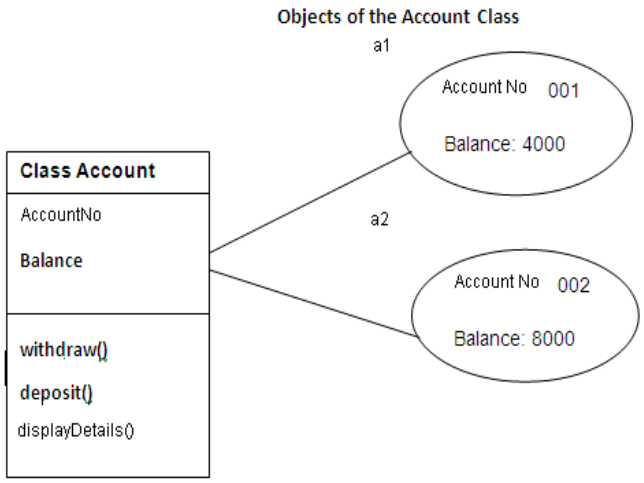
I. Class

A class functions as a blueprint or template for a collection of related objects. A class functions similarly to an object factory. From a single class, you may create an unlimited number of objects of the same kind. An object is an instance of a class. (kumari, 2025)




```
class   Rectangle:
   def   __init__ (self, width , height):
      self.width =  width
      self.height = height

   def  findArea(self):
      area = self.width  *  self.height
      print ("Area is  ", area)
```

Class of a Bank Account




```
class  Account:
  def  __init__ (self, accno, name, balance):
    self.accno = accno
    self.name = name
    self.balance = balance

  def  deposit(self, amount):
    self.balance = self.balance + amount
    print ("New balance is ", self.balance)

  def  withdraw(self,  amount):
    if  (amount > self.balance):
      print ("Invalid Amount")
    else:
      self.balance = self.balance - amount
      print ("New balance is ", self.balance)

  def   display(self):
    print ("Account No  ", self.accno)
    print ("Name  ", self.name)
    print ("Balance ", self.balance)
```

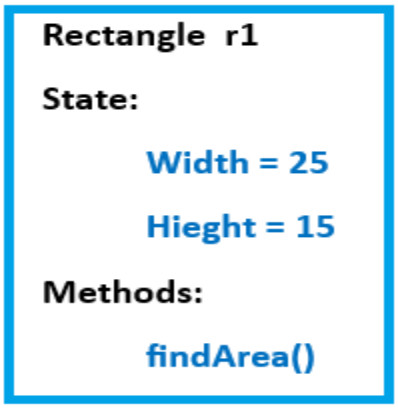
II. Object

An object is a software component that contains an ID, state and a set of actions.

State is made up of qualities and their values. The status of an item may vary over time. Operations describe the behaviour of the object (what the object can do). Operations are sometimes termed methods or functions. (kumari, 2025)

Example: **a rectangle object**

r1 = Rectangle (25, 15)



Example: **Account objects**

```
a1 =  Account (125, "Ama", 500)
a2 = Account(126,"Ben", 800)
```

III. Message Passing

Objects communicate with each other by passing messages. A message is a command sent to an object from outside. A message invokes a method within the object. Objects communicate with each other by passing messages. (kumari, 2025)

Message passed to a Rectangle object

r1.findArea()

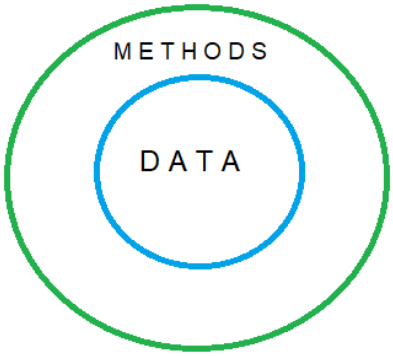
Message passed to Account objects

```
a1.withdraw(100)
a2.deposit(200)
a1.display()
a2.display()
```

IV. Encapsulation

This is the process of enclosing attributes and operations within a unit called class. A class is like a capsule consisting data and methods. This is done in such a way that the operations/methods provide access to data within the object.

Encapsulation makes software maintenance easier. This is because we could modify one software component (i.e. class) without having side effects on other software components. Encapsulation also makes software reuse easier.




```
class Employee:
    def __init__(self, name, salary):
        self.name = name
        self.salary = salary

    def display(self):
        print ("Name is ", self.name)
        print ("Salary is ", self.salary)
```

V. Abstraction

When we construct object classes we only add characteristics and methods that are useful and important. Attributes and methods not relevant are left out. For example, while modeling a Student class, we may include properties like name, address, gender and age since these are significant. We leave away qualities like height, weight and blood group since they are irrelevant and unnecessary. (Herrity, 2025)

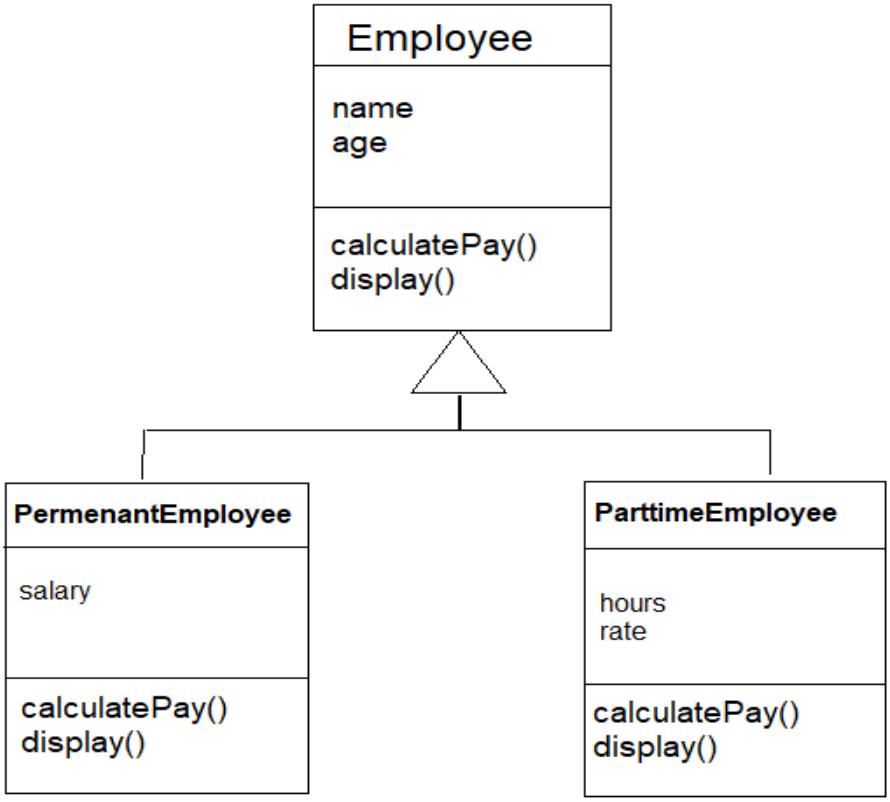
```
class Student:
  def __init__(self, name, address, age, gender ):
    self.name = name
    self.address = address
    self.age = age
    self.gender = gender
```

VI. Inheritance

We can arrange classes into a hierarchy when they have certain common traits. This will enable us to derive new classes from existing ones.

A class, from which another class is derived, is termed a super class or parent class. A new class, which is derived from an existing class, is termed a sub class or child class. A sub class inherits all the data and operations/methods of the super class. In addition, a sub class may have its own data and operations/methods. (Herrity, 2025)

Sub classes have a "**is a kind of**" connection with the super class. e.g. half time employee is a sort of an employee.



Coding of the Employee Super Class

```
from abc import ABC, abstractmethod

class  Employee (ABC):      # abstract class
   def  __init__(self, name, age):
      self.name = name
      self.age = age

   @abstractmethod
   def  calculatePay(self):
      pass

   def display(self):
      print ("Name is ", self.name)
      print ("Age is ", self.age)
```

Note

Employee is an abstract class, which means it is NOT possible make objects from it. An abstract class must be a sub class of ABC. An abstract class has one or more abstract methods. Normally, superclasses are declared abstract.

calculatePay() is an abstract method (i.e. method without body). Sub classes must override it to provide specific functionality.

Classes that can make objects are called Concrete classes.

Coding of the PermenantEmployee Sub Class

```
class PermenantEmployee(Employee):
  def __init__(self, name, age, salary):
    super().__init__(name, age)
    self.salary = salary

  def calculatePay(self):
    print ("Salary of Permenant Employee ", self.name)
    print ("Basic Salary  ", self.salary)
    epf = self.salary*10/100
    print ("EPF  ", epf)
    net = self.salary - epf
    print ("Net Salary ", net)

  def  display(self):
    super().display()
    print ("Salary is ", self.salary)
```

Coding of the Parttime Employee Sub Class

```
class PartTimeEmployee(Employee):
  def __init__(self, name, age, rate, hours):
    super().__init__( name, age)
    self.rate = rate
    self.hours = hours

  def calculatePay(self):
    print ("Payment of Parttime Employee ", self.name)
    pay = self.rate * self.hours
    print ("Pay is ", pay)

  def  display(self):
    super().display()
    print ("Hourly Rate is ", self.rate)
    print ("Hours Worked is ", self.hours)
```

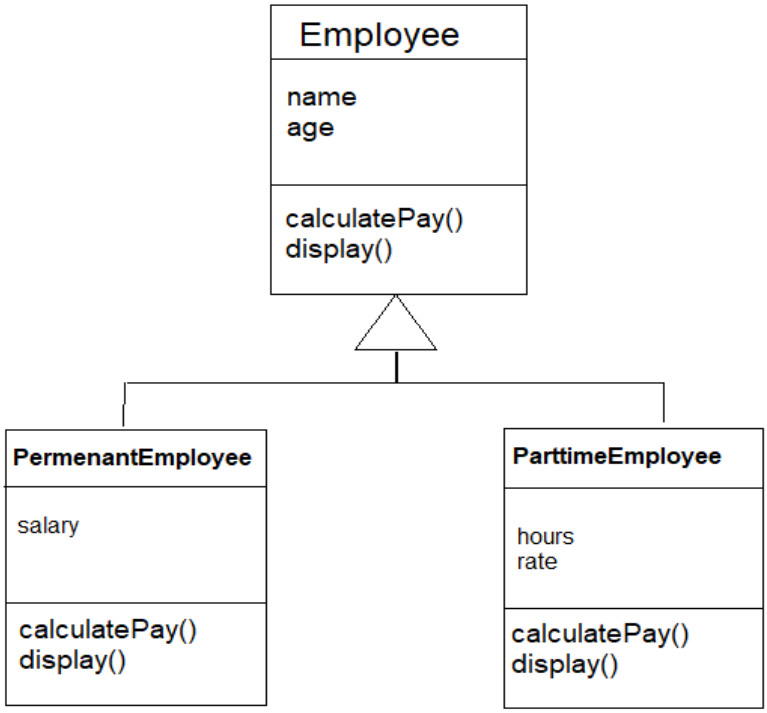
Coding of the Main Program

```
#main
e1 = PermenantEmployee("Sam", 25, 5000)
e2 = PartTimeEmployee("Azam", 40, 5, 2000)
e1.display()
e2.display()
e1.calculatePay()  # salary of permenant employee
e2.calculatePay()  # payment of parttime employee
```

VII. Polymorphism

Polymorphism implies having multiple forms. In OOP, it denotes the capacity of the same message to elicit multiple outcomes from distinct objects. (kumari, 2025)

In the following example, the calculatePay() function exhibits polymorphic behaviour.




```
e1.calculatePay()  # salary of permenant employee
e2.calculatePay()  # payment of parttime employee
```

1.2. Class relationships of OOP

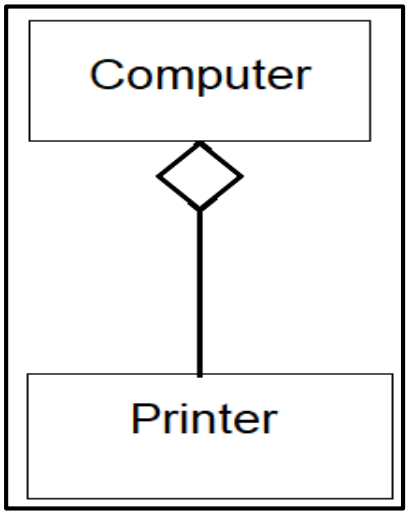
There are four types of Class Relationships in OOP. (Khan, 2023)

1. Aggregation
2. Composition
3. Association
4. Inheritance

1. Aggregation

This shows part-whole relationships between objects. An object can be a component/part of another object when there is an **"Is-part-of"** relationship.

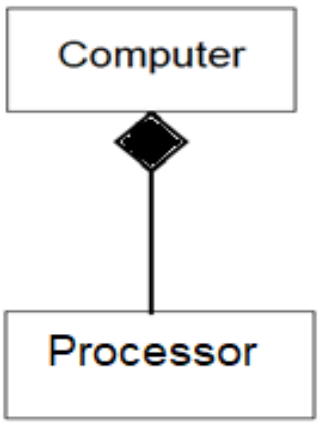
e.g. Printer is part of Computer



```
class Printer:
    def __init__(self, make, type):
        self.make = make
        self.type = type

    def display(self):
        print ("PRINTER")
        print ("Make ", self.make)
        print ("Type ", self.type)


class Computer:
    def __init__(self, brand, price):
        self.brand = brand
        self.price = price
        self.printer = ""

    def addPrinter (self, prin):
        self.printer = prin

    def display(self):
        print ("COMPUTER")
        print ("Brand is ", self.brand)
        print ("Price is ", self.price)
        self.printer.display()


#main
p1 = Printer ("Canon", "Bubble Jet")
c1 = Computer ("HP", 70000)
c1.addPrinter (p1)
c1.display()
```

2. Composition

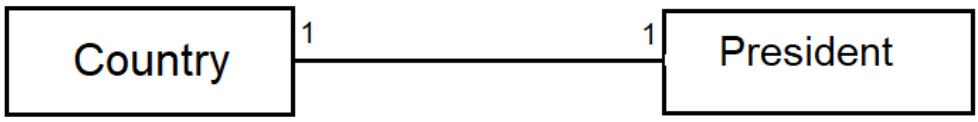
This is a strong form of aggregation with a life time dependency between the “Whole” and the “parts”. The part/component belongs to just one “whole”. If the “whole” is destroyed, part is also destroyed.




```
class  Processor:
   def  __init__(self, pro_make, pro_speed):
      self.pro_make = pro_make
      self.pro_speed = pro_speed

   def  display(self):
      print ("PROCESSOR")
      print ("Make  ", self.pro_make)
      print ("Speed  ", self.pro_speed)

class  Computer:
   def   __init__(self, brand, price, pro_make, pro_speed):
      self.brand = brand
      self.price = price
      p1 = Processor (pro_make, pro_speed)
      self.processor = p1

   def  display(self):
      print ("COMPUTER")
      print ("Brand is ", self.brand)
      print ("Price  is ", self.price)
      self.processor.display()


  #main
  c1 = Computer ("Dell",  80000 , "Intel", "4 Ghz")
  c1.display()
```

3. Association

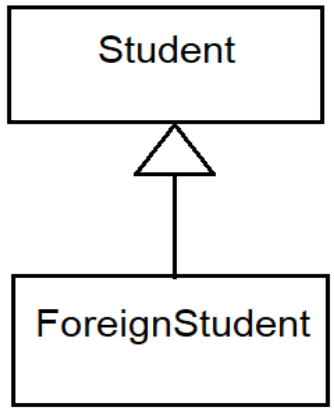
An association is a symmetrical relationship between two classes. It has a feature called multiplicity/cardinality which indicates the number of instances involved in the association. Based on multiplicity, we can categorize associations into three types: one-to-one, one-to-many and many-to-many.




```
class Country:
  def __init__(self, cname, capitol):
    self.cname = cname
    self.capitol = capitol
    self.president = ""

  def addPresident(self, pres):
    self.president = pres

  def display(self):
    print ("Country ", self.cname)
    print ("Capitol ", self.capitol)
    self.president.display()


class President:
  def __init__(self, pname, age):
    self.pname = pname
    self.age = age
    self.country = ""

  def addCountry (self, ct ):
    self.country = ct

  def display(self):
    print ("President ", self.pname)
    print ("Age ", self.age)


#main
c1 = Country("USA", "Washington")
p1 = President("Joe Biden", 80)
c1.addPresident(p1)
p1.addCountry(c1)
c1.display()
```

4. Inheritance

We can organize classes into a hierarchy when they have some common features. This will allow us to derive new classes from existing classes.




```
class Student:
    def __init__(self, name, age):
        self.name = name
        self.age = age

    def display(self):
        print (self.name)
        print (self.age)

class ForeignStudent (Student):
    def __init__ (self, name, age, country):
        super().__init__(name, age)
        country = country
    def display(self):
        super().display()
```

1.3. SOLID Principles of OOP

SOLID is a collection of five object-oriented design principles that may help you build more manageable and re-usable code based on well-designed, clearly organized classes. These ideas are a core aspect of object-oriented design's best practices.

I. Single Responsibility Principle

This implies that a class should have just one duty, as stated by its methods. If a class handles more than one duty, then you should break those duties into different classes. This enhances software reuse and makes program maintenance simpler. (Maksimau, 2025)

Example of poor code – Student and Course combined

```
class StudentCourse:
  def __init__ (self, name, age, title, fee ):
    self.name = name
    self.age =  age
    self.title =  title
    self.fee = fee

  def display(self):
    print ("Name ", self.name)
    print ("Age ", self.age)
    print ("Title ", self.title)
    print ("Fee  ", self.fee)
```

Example of good code – Student and Course separated

```
class Student:
    def __init__(self, name, age ):
        self.name = name
        self.age = age

    def display(self):
        print ("Name ", self.name)
        print ("Age ", self.age)

class Course:
    def __init__(self, title, fee ):
        self.title = title
        self.fee = fee

    def display(self):
        print ("Title ", self.title)
        print ("Fee  ", self.fee)
```

II. Open-Closed Principle

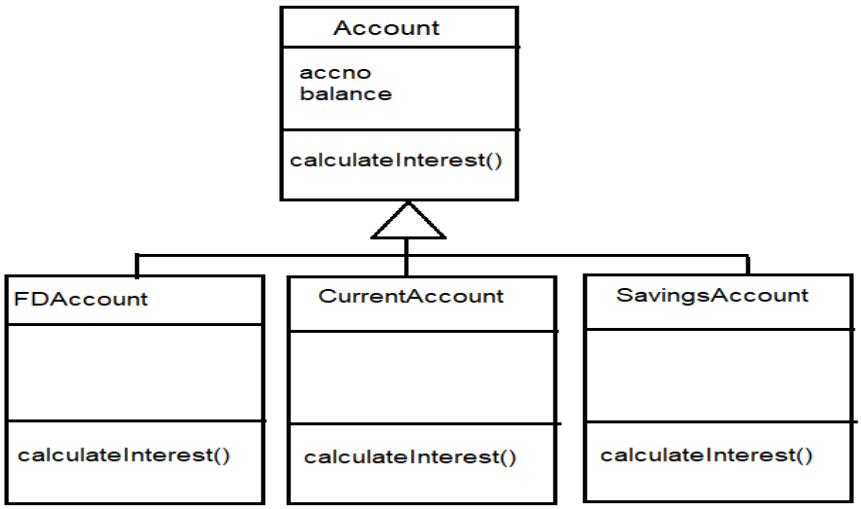
This implies Software object classes should be available for extension, but closed for alteration. It is okay to expand an existing class via sub-classing (Inheritance) and method overriding provided it does not affect existing code. However, modifying current code must be avoided since it might generate severe side effects on other areas of code. As an example, take a Bank Account class that must compute interest at two distinct rates depending on Account Type which is “Fixed Deposit” and “Current”. This involves having “IF” criteria to identify each Account Type. What if we offer another account type named “Savings” Account? We would have to alter the code to incorporate another “IF” condition. (Maksimau, 2025)

POOR DESIGN

```
class Account:
  def __init__(self, accno, balance, acctype):
    self.accno = accno
    self.balance = balance
    self.acctype = acctype

  def calculateInterest(self):
    if (self.accntype == "FD"):
      return self.balance * 10 / 100
    if (self.acctype =="Current"):
      return self.balance * 2 / 100
```

A better design would be to have a super class called “Account” with sub classes such as “FDAccount” , “CurrentAccount” and “SavingsAccount”. The super class can have an abstract method called **calculateInterest()** that must be overridden by all the sub classes of Account. Sub classes such as “FDAccount” , “SavingsAccount” etc. will provide their own functionality to the calculateInterest() method. This new design allows us to easily extend the existing design without tampering with the existing code.

GOOD DESIGN

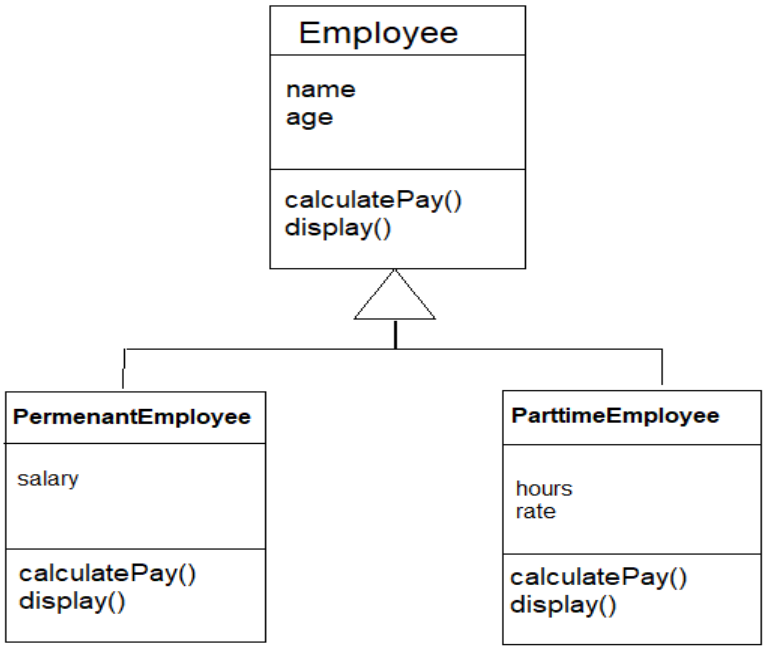



```
from  abc  import ABC, abstract method

class Account(ABC):
   def  __init__( self, accno, balance):
      self.accno = accno
      self.balance = balance

   @abstractmethod
   def  calculateInterest(self):
      pass

class FDAccount (Account):
    def  calculateInterest(self):
      return self.balance * 10 / 100

class CurrentAccount (Account):
    def  calculateInterest(self):
      return self.balance *  3 / 100

class SavingsAccount (Account):
    def  calculateInterest(self):
      return self.balance * 5 / 100
```

III. Liskov Substitution Principle

Invented by Barbara Liskov in 1988, this principle says that where a **method expects a super class object as a parameter it can be substituted by a sub-class object.** This means the same user code can work with different sub-class objects and produce sub-class specfic behavior at different occassions. In other words, the user code behavior is **Polymorphic**.

The advantage of Substitution Principle is that it allows us to extend a class hierarchy (i.e. add new sub classes) without requiring a change in user code. In the following example, the same **processPayment()** method of the Payroll class can work with objects of Employee sub classes – **Permenant Employees** and **Parttime Employees** and produce Polymorphic results.



```
class  Payroll:
  def  processPay (self, emp):  # expect employee object
    emp.calculatePay()

#main
e1 = PermenantEmployee("Sam", 25, 5000)
e2 = PartTimeEmployee("Ama", 40, 5, 2000)
p1 = Payroll()
p1.processPay(e1)  # Substitution Principle - Send Permenant Employee
p1.processPay(e2)  # Substitution Principle - Send Parttime Employee
```

IV. Interface Segregation Principle

An Interface is a specific form of a super class that includes just abstract methods (i.e. empty methods). The Interface Segregation Principle asserts that an Interface shall not require sub classes to implement its abstract methods. (Maksimau, 2025)

The sub classes must implement these methods via method overrding. But, occasionally a badly designed interface contains specific methods that a certain sub class is difficult to implement.

Consider a Student Interface that contains two abstract methods - one for recording assignment marks and the other for exam marks. But that is an issue for HND students since they have simply homework, NO examinations. It is also an issue for BCS students who have simply tests, NO homework.

POOR DESIGN

```
from abc import ABC, abstractmethod

class Student (ABC):

    @abstractmethod
    def recordAssignmentMarks (self, marks):
        pass

    @abstractmethod
    def recordExamMarks (self, marks):
        pass


class HNDStudent ( Student):
    def __init__( self, name, age ):
        self.name = name
        self.age = age
        self.marks = ""

    def recordAssignmentMarks (self, marks):
        self.marks = int (input ("Enter Marks "))
        # more code

    def recordExamMarks (self, marks):
        print ("Not Applicable")
```

GOOD DESIGN

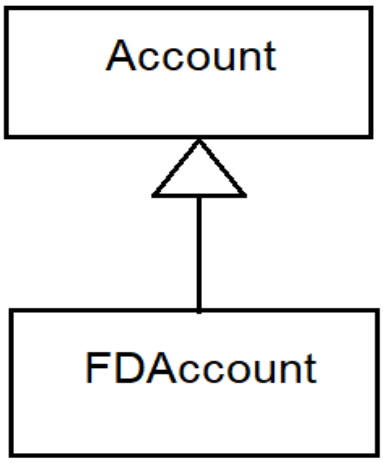
We could improve the design by separating the Student Interface into **two Interfaces,** one for **Coursework Students** such as HND and another for **exam students** such as BCS.

```
from  abc  import ABC, abstractmethod

class  CourseWorkStudent (ABC):
   def  recordAssignmentMarks (self, marks):
      pass

class ExamStudent (ABC):

   def  recordExamMarks (self, marks):
      pass
class  HNDStudent ( CourseWorkStudent ):
   def  __init__( self, name, age ):
      self.name = name
      self.age = age
      self.marks = ""

   def  recordAssignmentMarks (self, marks):
      self.marks = int (input ("Enter Marks "))
      # more code

class  BCSStudent ( ExamStudent):
   def  __init__( self, name, age ):
      self.name = name
      self.age = age
      self.marks = ""

   def recordExamMarks (self, marks):
      self.marks = int (input ("Enter Marks "))
      # more code
```

V. Dependency Inversion Principle

This principle says that **classes that provide details** must **depend on abstractions** and not the other way around. This implies that our design must start with **Interface classes** or **Abstract classes**. With abstractions we capture behavior at a higher level. Sub classes provide details by overriding abstract methods.

In the following example, the FDAccount depends on the abstraction "Account" and its abstract method calculateInterest().




```
from abc import ABC, abstract method

class Account(ABC):
  def __init__( self, accno, balance):
    self.accno = accno
    self.balance = balance

  @abstractmethod
  def calculateInterest(self):
    pass

class FDAccount (Account):
  def calculateInterest(self):
    return self.balance * 10 / 100
```

1.4. EVALUATING THE IMPACT OF SOLID PRINCIPLES ON OOP DESIGNS

SOLID is an accronym for 5 key concepts that contribute to high quality, flexible and efficient Object Oriented Designs. The resulting software is easy to maintain, easy to extend and easy to reuse. The 5 principles are as follows:

S - Single-responsibility Principle.

O - Open-closed Principle.

L - Liskov Substitution Principle.

I - Interface Segregation Principle.

D - Dependency Inversion Principle.

The Single Responsibility Principle means that a class should have only one **responsibility**, as expressed through its methods. This makes suer that one class does one job. If some maintenance issue comes up, the programmer can easily locate and fix the issue with the relevant class. There will be no side effects on other classes. It also easy to reuse such classes as they are totally independent.

The Open-Closed Principal discourages changes to already working code as it can create unwanted side effects. It encourages extensions (via sub classes) to existing classes which is totally harmless. The Liskov Substitution Principle allows us to extend class hierarchies without requiring changes to user code. Since the user code refers to Super class parameters, it can work with different sub-class objects of the same class hierarchy. The Interface Seggregation Principle advices us to create sub classes from suitable interfaces that has only relevant abstract methods. In other words, interfaces must not force irrelevant methods to sub classes.

The Dependancy Inversion Principle tells us to start from abstractions and then add details.

The end result of SOLID principles is code that is easy to understand and maintain. SOLID based code is also easy to reuse and extend. This saves a considerable amount of software development time and money.

1.5. CLEAN CODING TECHNIQUES

Clean coding is concerned with following a set of **guidelines** to produce high quality code that is **easy to understand, debug and maintain**. Clean code also facilitates communication among team members. (pluralsight, 2022)

These guidelines are as follows,

i. Meaningful and descriptive names

The names given to variables, classes and functions/methods must be meaningful and descriptive. The names given to variables must indicate what data they hold. Function names must indicate what task they perform. Class names must indicate what entity it represents.

- Example of poor code

```
sal = [10000, 50000, 30000, 75000, 40000 ]
def  findTot():
    y = 0
    for x in range (len(sal)):
        y = y + sal[x]
    return y
```

- Example of improved code

```
sales = [10000, 50000, 30000, 75000, 40000 ]
def  findTotalSales():
    tot_sales = 0
    for index in range (len(sales)):
        tot_sales = tot_sales + sal[x]

    return tot_sales
```

ii. Comments

Where appropriate, comments must be provided to **explain the code**. They must explain in simple terms what the code does. It is a good practice to give a block of comments at the beginning of every function. In addition, every tricky line of code must be explained with a comment.

- Example

```
#calculate average price
avg_price = tot_price / len(prices)
```

iii. Indentation

Statements within loops and "if" conditions must be indented to add clarity. In Python, this is done automatically, but in other languages the programmer must indent the statements.

- Example 1

```
for index in range (len(sales)):
    tot_sales = tot_sales + sal[x]
```

- Example 2

```
if ( sales_amount>=10000):
   discount = sales_amount * discount_rate
else:
   discount = 0
```

iv. Named constants

Numeric constants must be given a meaningful name. This will improve the readability of the code. Moreover, if the constant value is changed, we only have to change only one place.

- Example of poor code

```
discount = sales_amount * 0.2
```

- Example of improved code

```
discount_rate = 0.2
discount = sales_amount * discount_rate
```

v. Single task functions

A function must perform a single task. This will make any subsequent modifications easier.

```
sales = [10000, 50000, 30000, 75000, 40000 ]
def  findTotalSales():
    tot_sales = 0
    for index in range (len(sales)):
        tot_sales = tot_sales + sal[x]

    return tot_sales
```

vi. Coding standards

When writing names of attributes, methods etc. use an established standard such as camelCase notation and snake case notation.

```
# snake case example

total_annual_sales =  0

# camel case example

def  calculateTotalSales(self):
   pass
```

1.6. Impact of Clean Coding on Data Structures and Algorithms

Data structures are arrangements of data. The most familiar **data structures are Lists, Arrays, Stacks, and Queues.** Algorithms are processes that work on these data structures. For instance, searching and sorting are two operations commonly carried on Arrays while push and pop are operations that are done on stacks and enqueue and deque are operations done on queues. (David Loshin, 2021)

Implementional matters can enhance these data structures and the operations performed on them in terms of their readability, reusability and maintainability. The following are associated with clean coding,

First, we must assign a meaningful name to the data structure which we are using. Essentially, the name used must reflect the content of the data structure. Finally, the algorithm that uses or manipulate it has to be described with comments. It is required that names of variables should be meaningful. Loops and conditions (if else) will be made more understandable with indentations. This way of writing the code makes the reader easily understand what the code is all about. That in turn means that the amount of effort and the amount of money required to maintain the software is low.

The following example shows a List data structure that stores the prices of products. An algorithm utilizing clean coding techniques is used to find the total and average of product prices.

```
# declare list of prices
prices = [15, 26, 100, 80, 250 , 50 ]

# find the average price of products
index = 0  # initialize list index
tot_price = 0  # initialize total price
while (index < len(prices)):
    tot_price = tot_price + prices[index]
    index = index + 1
# print total price
print ("Total Price ", tot_price)
#calculate average price
avg_price = tot_price / len(prices)
print ("Average price ", avg_price)
```

In the next example, we can see two parallel lists – one has **customer names** and the other has their **loyalty points scored at the super market**. A function called Search() is written to find the points of a given customer. Here, both the data structures were given meaningful names. The function that performs the search has a meaningful name and comments to explain the vital parts of the code.

```
NAME = ["John", "Brown", "Ann", "Roy", "Tina" , "Fred" ]
POINTS  = [  25, 180, 230, 16, 27,  190 ]

def  search ( searchkey ):
    index = 0
    while ( index < len(NAME) ):
        if  ( searchkey == NAME[index]):
            print ("LOYALTY POINTS ", POINTS[index])
        index = index + 1

# main
searchkey =     input ("Enter Name as Search key ")
search (searchkey)
```

Stack is a data structure that works in a **Last In First Out (LIFO)** basis. Stacks are used when data a re processed in reverse order (e.g. Undo operation). It has two major operations : push() and pop(). The push operation adds a data item to the top of the stack. The pop operation

removes a data item from the top of the stack. Following clean code shows how they appear to the reader.

```
# LIFO data structure
class Stack:
    def push(data):    # add data to the top
        pass

    def pop():     # remove data from top
        pass
```

Queue is a data structure that works in a **First In First Out (FIFO)** order. Queues are used when data needs to be processed in the order they arrive. E.g. Printer queue

There are two standard operations on queues. Enqueue() operation adds a data item to the tail of the queue. Deque() removes an item from the front of the queue. Following clean code shows how they appear to the reader.

```
# FIFO data structure
class Queue:
    def enque(data):    # add data to the tail
        pass

    def deque():     # remove data from front
        pass
```

1.6.1. The benefits of clean coding

- **Readability and maintenance**

  Clean code prioritizes clarity, which makes reading, understanding, and modifying code easier. Writing readable code reduces the time required to grasp the code's functionality, leading to faster development times.

- **Team collaboration**

Clear and consistent code facilitates communication and cooperation among team members. By adhering to established coding standards and writing readable code, developers easily understand each other's work and collaborate more effectively.

- **Debugging and issue resolution**

  Clean code is designed with clarity and simplicity, making it easier to locate and understand specific sections of the codebase. Clear structure, meaningful variable names, and well-defined functions make it easier to identify and resolve issues.

- **Improved quality and reliability**

  Clean code prioritizes following established coding standards and writing well-structured code. This reduces the risk of introducing errors, leading to higher-quality and more reliable software down the line.

1.7. Design patterns

1.7.1. Introduction

A design pattern is a **problem/solution** pair that can be reused in new contexts. It is a general, reusable solution to a commonly occurring problem in software development.

Design patterns are developed and applied in object oriented system development. Design patterns are used to represent some of the best practices adopted by experienced object oriented software developers.

Designing quality software is difficult and demands experience. By documenting the patterns of successful designs we can reuse the knowledge and the experience of other programmers. Patterns solve recurring design problems and make object oriented designs more reusable, efficient and flexible. Reuse of design patterns enable us to develop quality software.

1.7.2. Classification of design patterns

Design Patterns are categorized mainly into three categories. **Creational Design Patterns, Structural Design Patterns, and Behavioural Design Patterns**. These are differed from each other on the basis of their level of detail, complexity, and scale of applicability to the system.

1. **Creational design patterns**

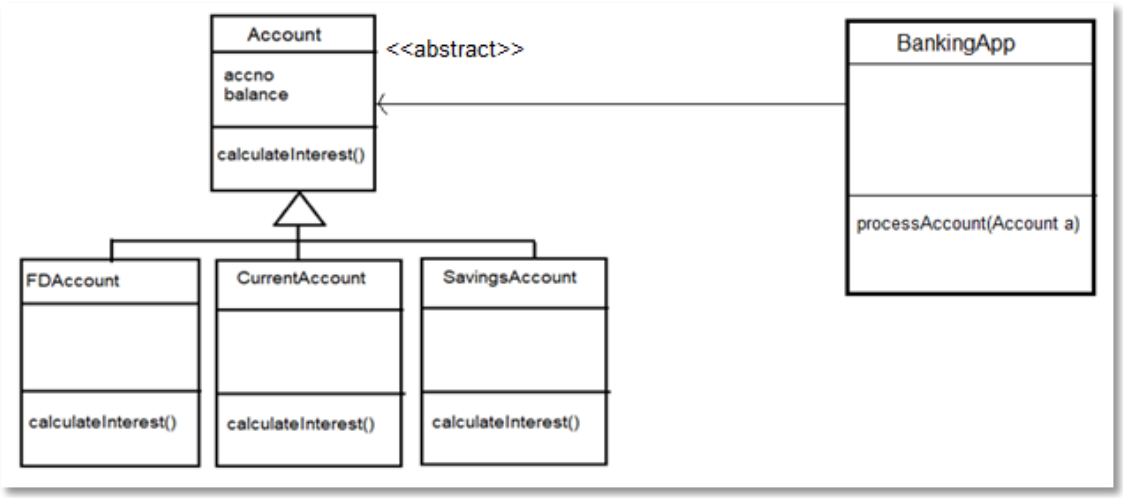
In software engineering, creational design patterns are design patterns that deal with object creations. They try to create objects in a manner suitable to the problem situation. Creational design patterns control object creations. They encapsulate knowledge about what objects they create. (geeksforgeeks, 2025)

Types of Creational Design Patterns

i. **Abstract Factory Method Design Patterns**

We separate several concrete implementations of classes from their abstraction. The abstraction serves as an interface to the outside world. The same user code can deal with different objects of concrete sub classes via Liskov Substitution.

Example*:* A banking application needs to deal with different types of Accounts objects on different occasions. Accounts are organized into a class hierarchy with an abstraction at the top. The banking app uses Liskov Substitution to handle different implementations of Account class.



ii. Singleton Method Design Pattern

There are situations where we have to make only one instance of a given class. Any attempt to make more than one object/instance of that class should be prevented. The singleton pattern solves this problem by making the class itself responsible for keeping track of the sole instance. A class variable maintains an object count (geeksforgeeks, 2025).

Example: A country has many people but only one President. We must create only one instance/object of the President class.

```
class  President:
  count = 0
  def  __init__(self, name, age):
    if (President.count == 0):
      self.name = name
      self.age = age
      President.count = President.count+1
    else:
      print ("Sorry - President already exists")
      self.name = ""
      self.age = ""

  def  display(self):
    print (self.name)
    print (self.age)
```

2. Structural design patterns

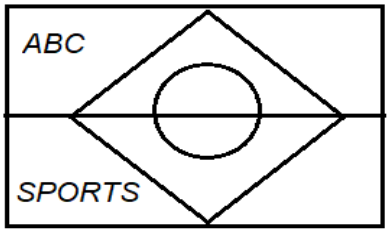
Structural patterns form **larger structures from individual components, generally of different classes**. Structural patterns vary a great deal depending on what sort of structure is being created for what purpose. Structural patterns are concerned with how classes and objects are composed to form larger structures.

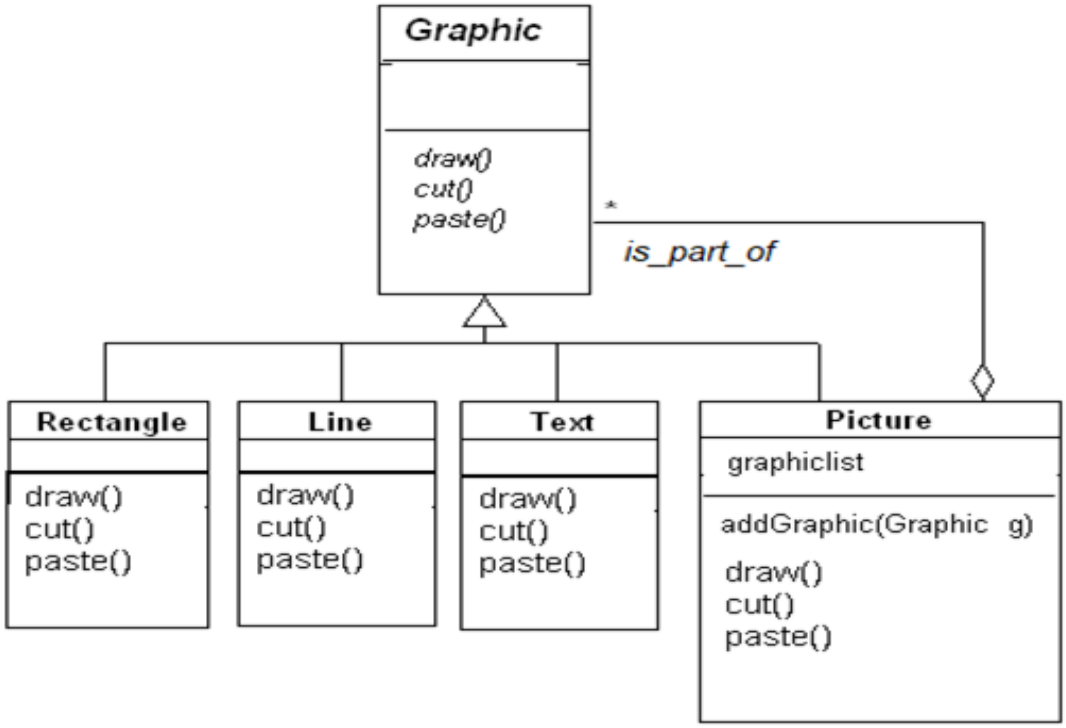
i. Composite Pattern

The composite pattern allows us to build "part-whole" hierarchies recursively (compose big objects from small objects, then use big objects to compose even bigger objects). It handles simple objects and compositions of objects in a uniform way. That is, operations that apply to a simple object must also apply to the complex object in the same way. This makes it easier for clients to deal with objects.

example, Grouping in MS-Word

We draw elementary objects like rectangles, lines, circles, text boxes etc. Then we group them to make a single complex object. Operations like cut(), copy(), paste(), delete() apply to the simple objects as well as the big object in the same way. Moreover, we can compose even bigger objects from complex objects recursively. This is achieved by means of Composite Design Pattern.

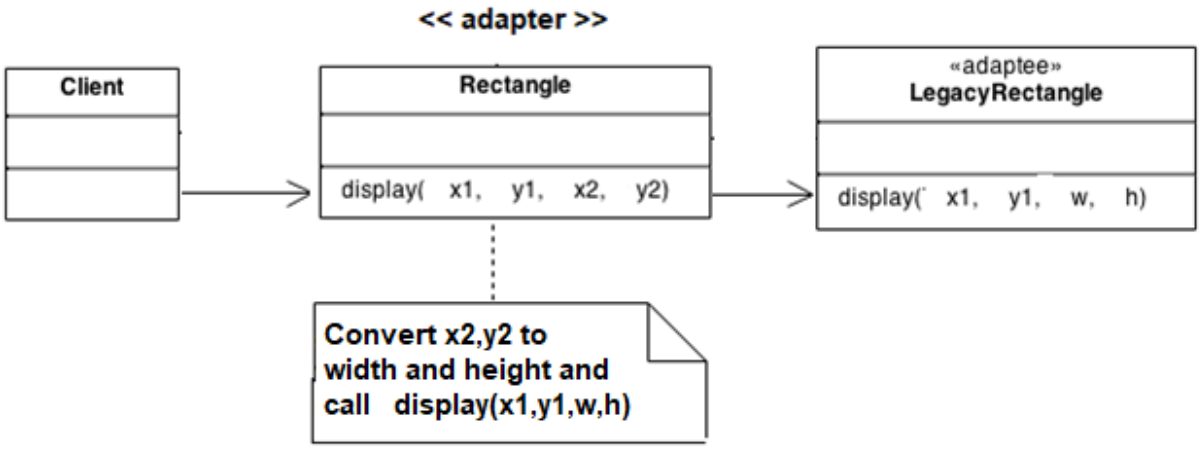




ii. Adapter Pattern

Convert the interface of a class into another interface that the clients expect. Adapter lets classes with incompatible interfaces work together.

A legacy Rectangle component's display() method expects to receive "x, y, w, h" parameters. But the client wants to pass "upper left x and y" and "lower right x and y". This mismatch can be solved by adding an intermediate Adapter object.



3. Behavioural patterns

Behavioural patterns describe **interactions between objects**. They focus on how objects communicate with each other. Behavioural patterns are concerned with assignment of responsibilities between objects. Behavioural patterns describe not just patterns of objects or classes but also the patterns of communication between them.

i. Observer pattern

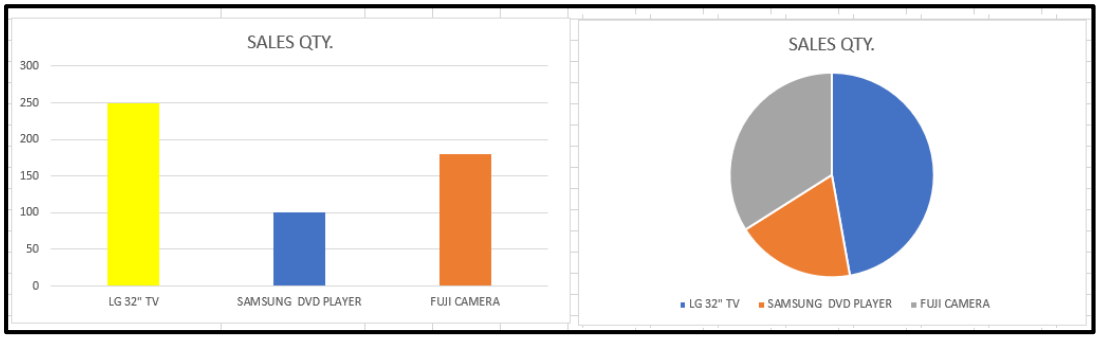
The observer pattern defines a one-to-many dependency between objects so that when one object changes state, all its dependents are notified and updated automatically. The original base object is called Subject. The dependents are called observers.

For example, in a spreadsheet, several charts (e.g. pie chart, bar chart etc.) could depend on one data table. When the data table changes state, the charts must change accordingly. The data table is the subject. The charts are observers.

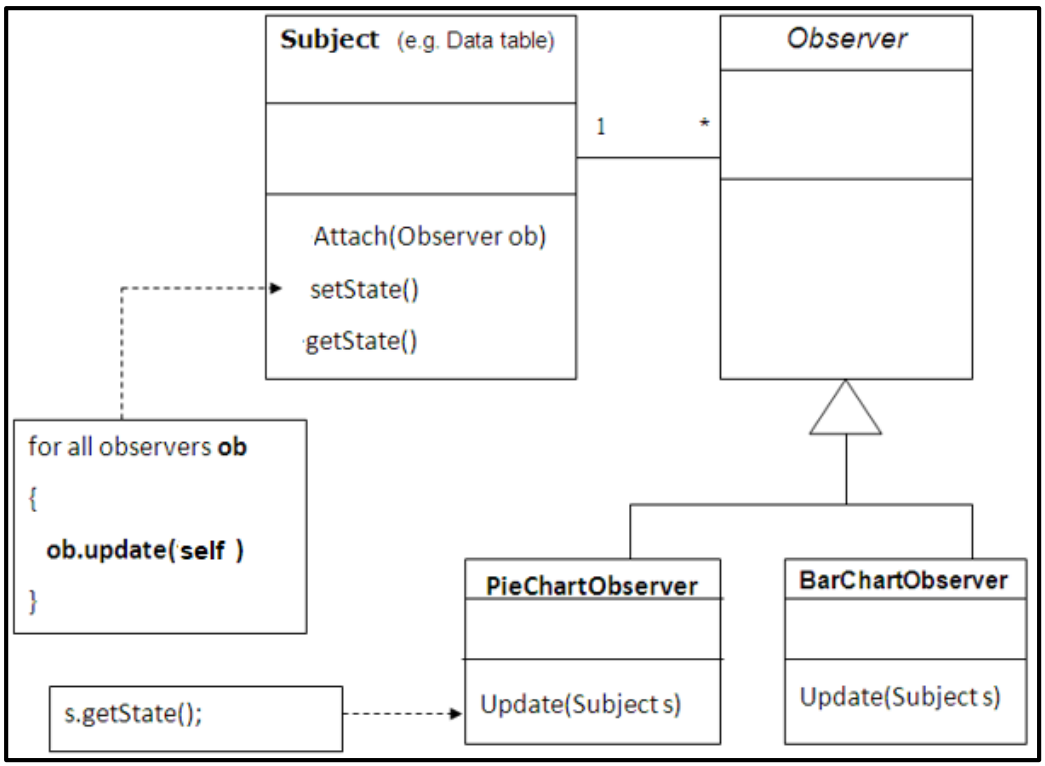
SUBJECT

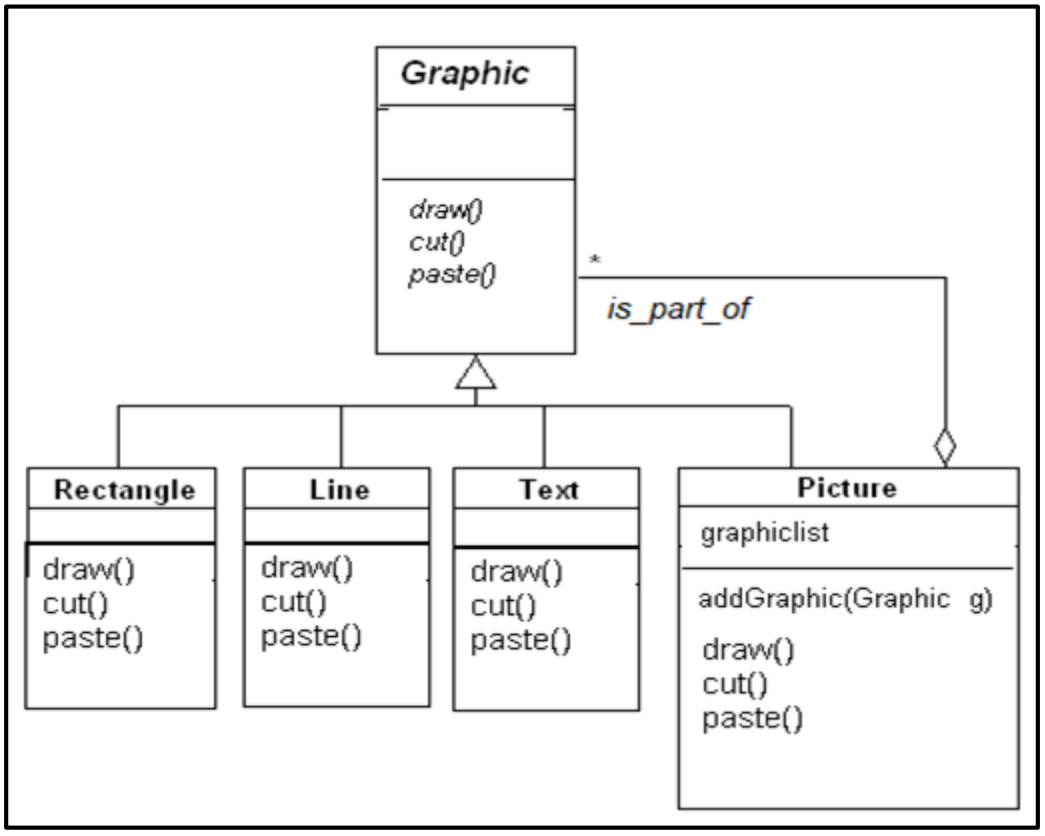
PRODUCT	SALES QTY.
LG 32" TV	250
SAMSUNG DVD PLAYER	100
FUJI CAMERA	180

OBSERVERS



The subject class maintains all observers in a container such as List. When the subject changes state it calls update() method to inform all observers. The update() method passes the subject object to observers so that they can retrieve subject state by calling getState().

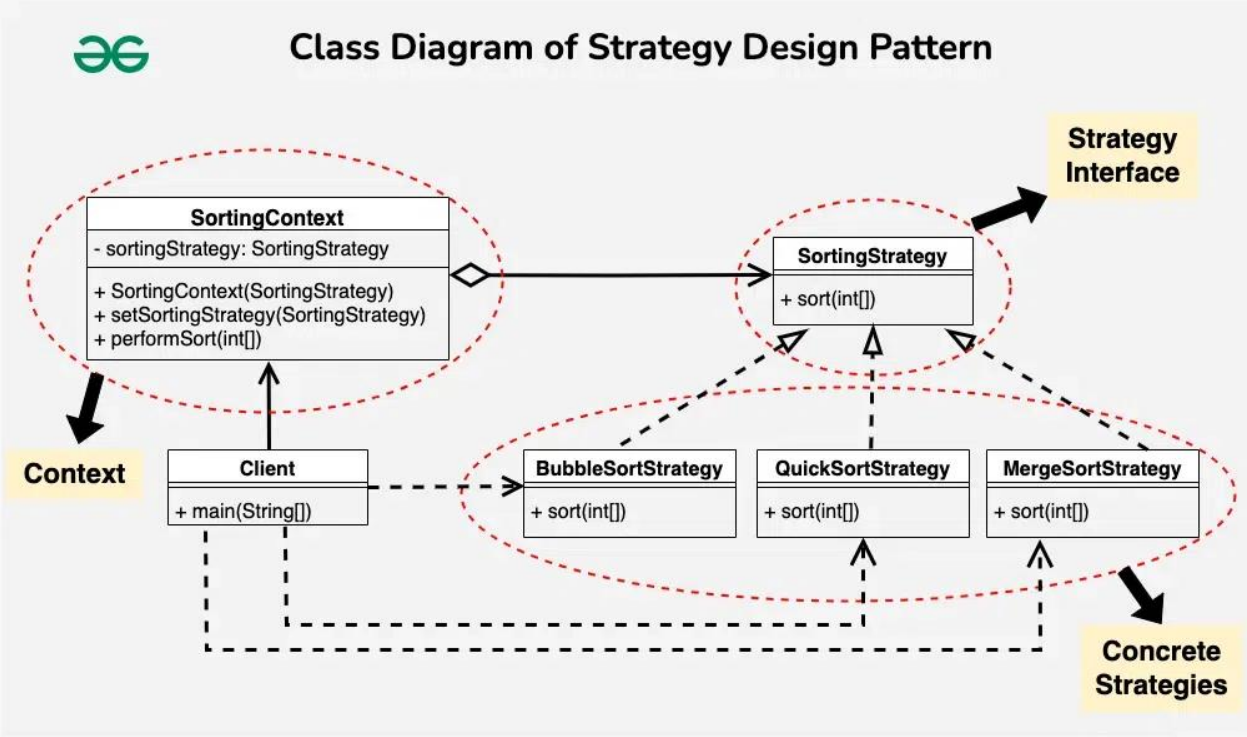




4. Strategy patterns

The strategy pattern is a behavioural software design pattern that selects the most suitable algorithm out of many at runtime. The user application has a task that can be accomplished by one algorithm out of several. The strategy executioner will choose the most suitable algorithm based on the user situation at run time.

Example, A user app needs to sort a large data set and there are several possible algorithms such as Bubble sort, Quick sort and Merge sort. The strategy pattern will use the best algorithm depending on the client requirements.



1.8. Analysis of Design Patterns

Design patterns play a vital role in modern software development by offering proven solutions to recurring design problems. They help designers and developers build systems more efficiently by reusing successful design approaches based on past experiences. When developers rely on a well-documented collection of design patterns, they can avoid "reinventing the wheel" and instead focus their efforts on addressing unique aspects of the current project. This not only saves time but also enhances the overall quality of the software.

A significant advantage of design patterns is that they are clearly documented in software engineering literature. Each pattern typically includes the pattern name, the problem it solves, the solution, sample code, and the potential consequences of using it. This structure makes it easier for developers to quickly identify the most suitable pattern for their problem. Such accessibility improves productivity and supports a more consistent development process.

Moreover, design patterns represent the expertise and best practices of experienced programmers. By incorporating these patterns into software projects, developers can improve code structure and maintainability. One major benefit is enhanced readability, which is essential for effective long-term maintenance. Readable code is easier to understand and debug, especially when the original developers are no longer available. In many cases, maintenance is carried out by new programmers who were not part of the original development team. If widely known design patterns are used, those new team members are more likely to recognize and understand the code structure quickly, reducing the time needed to make necessary updates or fixes.

design patterns offer a structured, efficient, and high-quality approach to solving software design problems. They simplify development, improve readability, and significantly support the maintenance process, making them a valuable asset in any software project.

Activity 02

2.1. Introduction Dream Book Shop

In today's fast-paced digital economy, the ability to analyze and interpret large sets of data plays a vital role in improving business performance. This is not limited to large corporations; even small and medium-sized businesses are recognizing the power of data analytics in driving smarter decisions. Dream Book Shop, a small but long-established comic-book retailer, is currently experiencing a period of steady growth. With more customers coming through both physical stores and online platforms, the company has realized the need to update its traditional systems and adopt more efficient, tech-based solutions to manage its expanding operations.

One of the key steps in this digital transformation is the ability to analyze historical data related to book publications. Dream Book Shop intends to use this data to identify patterns, such as which authors are most popular, which languages are in demand, and which time periods show spikes in publication activity. These insights will help the company understand customer preferences, manage its inventory better, and create targeted marketing strategies. The ultimate goal is to build a complete data dashboard that can visually present these findings in an interactive way.

Before moving to a full-scale dashboard, the company has decided to first develop a Command-Line Interface (CLI) prototype. This CLI application will act as a proof-of-concept, helping the team explore what kind of information can be extracted from large datasets and how that information can be organized.

The CLI tool will provide simple text-based and graphical summaries to show key trends. For example, it can list the most common authors, show the most used languages in books, or display how publication years vary across the dataset. This kind of analysis is expected to help Dream Book Shop make better decisions about which books to stock, how to organize their catalog, and how to communicate with customers more effectively.

Functional and non-functional requirements of the application

Functional Requirements (What the system *must do*)

1. **Load and read the CSV file**

   The system should open a .csv file that contains comic book metadata and read its contents without crashing.
2. **Choose different analysis types**

   Users should be able to pick from a menu or list of analysis types, like finding top authors or seeing books per year.
3. **Show most common authors**

   The app should find and display which authors appear the most in the dataset.
4. **Language distribution**

   It must count how many books are written in each language and show that in a readable way.
5. **Books per publication year**

   The system should calculate how many books were published each year and show that data as a table or chart.
6. **Top publishers list**

   It should show a summary of the most frequent publishers from the dataset.
7. **Simple user commands**

   Users need to enter simple commands like “top-authors” or “year-summary” to get results.
8. **Readable text results**

   Results should be shown in a clear format with proper spacing, headers, and labels so it’s easy to understand.
9. **Show visual summaries (basic graphs)**

   The app should display simple bar charts using ASCII symbols or Python libraries like matplotlib.
10. **Error handling for bad data**

    If the file is missing or has bad format, the app should show a clear error message to guide the user.
11. **Exit and help commands**

    Users should be able to type “exit” to quit and “help” to see all available commands.
12. **Limit entries if needed**

    There should be an option to limit results (like top 5 authors) so outputs don’t become too long.

Non-Functional Requirements (How the system should behave)

1. **Use Python and run in CLI**
   The whole project should be built in Python and run in a terminal or command prompt.
2. **Follow SOLID and clean code**
   Code should be well-organized with clear structure so it's easy to maintain or expand in the future.
3. **Fast performance for 5,000 rows**
   The tool should be able to process up to 5,000 rows in a few seconds (max 5 seconds ideally).
4. **Scalable for future GUI**
   The design should allow easy upgrading to a web or desktop dashboard in future stages.
5. **Well-commented code**
   All parts of the code should have comments so future developers can understand what it does.
6. **Can be tested automatically**
   The code should include or support unit testing using tools like unittest or pytest.
7. **Work on all platforms**
   It must run properly on Windows, macOS, and Linux without special setup.
8. **Easy to understand visual outputs**
   Graphs and summaries must be easy to read for anyone, even those not good with data.
9. **No internet dependency**
   It should work offline, without needing internet access, as long as the CSV file is present.
10. **Modular design**
    Functions and classes should be in separate files or modules to keep things organized.
11. **Minimal dependencies**
    Try to avoid using too many external libraries, so it's easy to install and run anywhere.

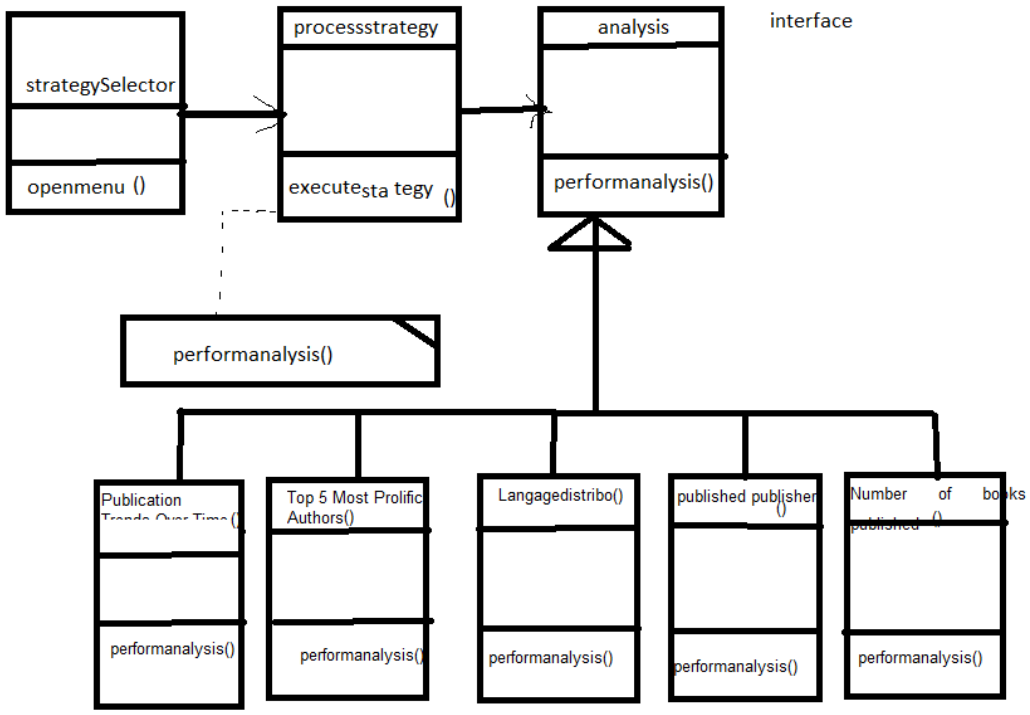
12. **CLI should be user-friendly**

    Instructions and prompts should be clear enough for users with basic tech skills.

13. **Support for invalid input**

    If user types wrong command, the system should show helpful suggestion or retry prompt.

2.2. Design of the UML



Object-Oriented Programming (OOP) Principles Applied

The Dream Book Shop CLI application is build using the main concepts of Object-Oriented Programming (OOP). These principles help the software stay organised, clean and easier to develop more features in future. The four key OOP ideas used are encapsulation, abstraction, inheritance and polymorphism.

Encapsulation is used by dividing the responsibilities into different classes. For example, there is a DataLoader class that only focus on reading and parsing the CSV file. It doesn't do any

analysis or display anything. Then there is another class, like DataAnalyzer, that only do analysing, like finding top authors or counting books per year. Because each class has it's own job, the code is much easier to fix or improve without breaking other parts. Also, the logic inside the class is hidden from the outside, so users don't mess with it directly.

Abstraction means hiding complex things and only showing what is necessary. In this app, the user only see the CLI menu and type simple commands like top-authors or language-summary. They don't have to know how the program is loading or cleaning the data inside. Behind the scenes, the class might have complicated logic but it is abstracted using simple methods like load_csv() or get_top_publishers(). This helps user and also makes the code reusable.

Inheritance is not used much in the current CLI version, but it is possible to add it if needed. For example, if we want to create more analysers like GenreAnalyzer, they can all inherit from a main Analyzer class. That way, they all share common functions and we write less code.

Polymorphism is also used. A method like generate_bar_chart(data) can work with many type of data, like authors, publishers, or years. It don't matter what kind of list we give, the method will create the chart in same way. That's polymorphism one function behaves differently based on the input.

By following these OOP ideas, the program becomes more flexible and easy to expand. In future, this CLI tool can be upgraded into a web or GUI app without rewriting everything from scratch.

2.3. Utilizing a Design Pattern

Strategy Pattern

The strategy pattern is a behavioural software design pattern that selects the most suitable algorithm out of many at runtime. The user application has a task that can be accomplished by one algorithm out of several. The strategy executioner will choose the most suitable algorithm based on the user situation at run time.

In the design of the data analysis system, the strategy pattern is used to select the correct data analysis function at run time. The user is given a menu that lists five different data analytics. Each analytic is covered by a separate algorithm implemented as a function/method. A menu is presented to the user with the options. When the user selects what he/she desires the strategy pattern chooses the correct one. The selection and execution of the data analytic function is done at run time. This makes the Strategy Design Pattern the ideal choice for implementing the design.

2.4. Utilizing SOLID Principles of OOP

The Dream Book Shop CLI data analysis application follows the SOLID principles of object-oriented programming. These five principles help make the code clean, easy to manage, and more flexible when changing or adding features later.

1. Single Responsibility Principle (SRP)
This rule says every class should only do one job. In our app, we followed this by splitting tasks between different classes. For example, DataLoader is only for loading CSV data, and it doesn't try to analyze or display anything. Likewise, DataAnalyzer just looks at the data and finds things like top authors or number of books per year. Because each class has its own responsibility, the app is easier to fix and understand.

2. Open-Closed Principle (OCP)
This principle means that code should be open for adding new stuff but closed to changing existing parts. In our app, we use a kind of "base" class or even an interface style design, where we can add more analysis types without changing the core logic. For example, if we later want to add a GenreAnalyzer, we can just extend the base analysis class or use a similar structure, keeping old code untouched.

3. Liskov Substitution Principle (LSP)
This one says you should be able to use any subclass wherever the parent class is expected. Our analysis system allows this. If we write a method that expects an analysis object (like a

general analyzer), we can give it a LanguageAnalyzer, PublisherAnalyzer, or any other custom analyzer class. The system will still work correctly without knowing the exact class type.

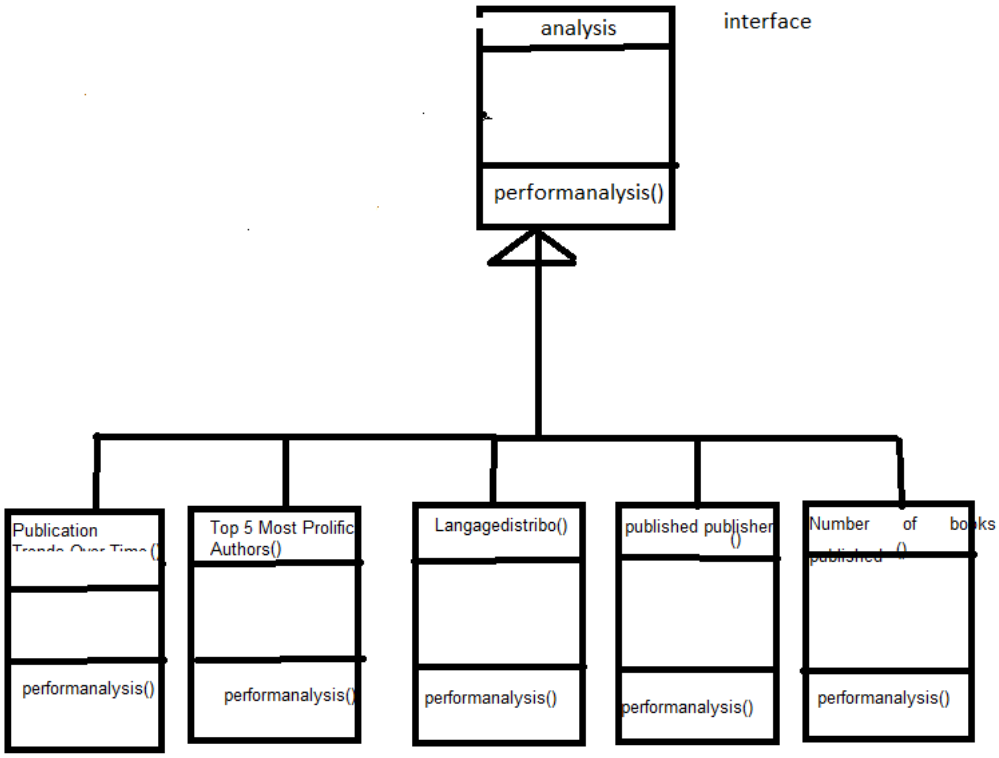
4. Interface Segregation Principle (ISP)

This principle tells not to force classes to implement methods they don’t need. In our app, the structure is kept simple with abstract-like classes or interfaces that only include the functions that are necessary. Each analyzer class, like TopAuthorsAnalyzer or YearSummaryAnalyzer, implements just what it needs – not more.

5. Dependency Inversion Principle (DIP)

Here, high-level modules shouldn’t depend on low-level modules. Instead, both should depend on abstractions. In our CLI tool, this is followed by depending more on abstract ideas like "analyzer classes" instead of tying everything to a specific implementation. This helps us switch or add new data analyzers without breaking other parts.

By using SOLID, our Dream Book Shop app is not just working well now, but it's also ready for growth in future.



2.5. Utilizing Clean Coding Techniques

Clean coding is concerned with following a set of guidelines to produce high quality code that is easy to understand, debug and maintain. Clean code also facilitates communication among team members.

1. Meaningful and descriptive names

The names given to variables, classes and functions/methods must be meaningful and descriptive. The names given to variables must indicate what data they hold. Function names must indicate what task they perform. Class names must indicate what enitity it represents. The data analysis system will use meaningful names for classes such as “Admin”, “Strategy” and “Analysis”. It will also use meaningful and descriptive names

for variables such as “username”, “password” and “choice”. Functions will be given names like “performAnalysis” to indicate what they perform.

2. Comments

Where appropriate, comments must be provided to explain the code. They must explain in simple terms what the code does. It is a good practice to give a block of comments at the beginning of every function. In addition, every tricky line of code must be explained with a comment. This data analysis code will utilize comments to explain tricky or complex code.

3. Indentation

Statements within loops and “if” conditions must be indented to add clarity. In Python, this is done automatically, but in other languages the programmer must indent the statements. The data analysis code will apply indentation to loops and conditionals to improve the clarity of the code.

4. Single task functions

A function must perform a single task. This will make any subsequent modifications easier. The functions in the data analysis system will each perform just one task. For example, the openMenu() function in the StrategySelector class performs the task of displaying a menu where the user can select the type of data analysis visualization he/she wants. The logon() function of the Admin class is responsible for permitting access to the system.

5. Coding standards

When writing names of attributes, methods etc. use an established standard such as camelCase notation and snake case notation.

6. Named constants

Numeric constants must be given a meaningful name. This will improve the readability of the code. Moreover, if the constant value is changed, we only have to change only one place.

7. White space

White space is blank areas intentionally included in the code to improve clarity. The code will contain white space between different functions and different classes.

2.6. Selected Data Structure and Justification

Following simple example shows how data about exercise durations and calories burned are stored in a pandas data frame.

```
import pandas as pd

data = {
  "calories": [420, 380, 390],
  "duration": [50, 40, 45]
}

#load data into a DataFrame object:
df = pd.DataFrame(data)
```

How the Data Frame Appears

```
   calories  duration
0       420        50
1       380        40
2       390        45
```

2.7. Test Regime Design

Software testing is an important step in the software development lifecycle. It entails checking and validating that a software program is bug-free, fits the technical criteria established during its design and development, and meets user needs efficiently and effectively (geeksforgeeks,

2025). This method assures that the application can handle all exceptional and boundary instances, resulting in a solid and dependable user experience. Software testing contributes to high-quality software that functions as intended in a variety of circumstances by methodically detecting and resolving faults.

The test regime refers to the organized approach taken for testing a system. This is a formal document that describes when and how to carry out the tests and what type of testing to use (e.g. manual testing, automated testing) and what testing methods to use (e.g. black box testing, white box testing). The test regime also defines testing stages (unit testing followed by integration testing and then system testing). The test regime also includes preparation of test cases.

2.7.1 Test execution goes through 4 phases

- Phase 1 - **Test Planning**

  The test planning phase is crucial for establishing a comprehensive testing strategy. During this phase, a detailed test plan document is created, outlining the overall approach to testing. This includes defining the stages of testing, such as unit testing, integration testing, system testing, and user acceptance testing. The plan specifies the types of testing to be conducted, including functional, performance, and security tests. It also identifies the automation tools required, such as Pytest for unit testing or Selenium for web application testing. Additionally, roles and responsibilities of the testing team are clarified to ensure efficient execution. A cost analysis and estimated timeline are included to allocate resources effectively and set realistic deadlines for the testing process.

- Phase 2 - **Test Case Development**

  In the test case development phase, specific test cases are designed to evaluate the system's functionality. Each test case outlines the test inputs, execution conditions, and expected results, ensuring comprehensive coverage of the system's features. The goal is to achieve 100% test coverage, meaning every aspect of the application is tested. This phase also involves creating automation scripts if needed, to streamline repetitive testing tasks and improve efficiency. Properly developed test cases and scripts are essential for accurate and reliable testing outcomes.

- Phase 3 - **Test Environment Setup**

  The test environment setup phase involves configuring and deploying the environments required for testing. This includes setting up hardware, software, and network configurations that mimic the production environment as closely as possible. Testing tools, such as Pytest for unit tests or Selenium for functional testing, are installed and configured. Ensuring that the test environment mirrors the actual usage conditions is critical for identifying and resolving issues effectively. Proper setup of the environment ensures that tests are conducted under realistic conditions and results are reliable.

- Phase 4 **Test Execution**

  During the test execution phase, the actual testing takes place. System features and functions are tested according to the predefined test cases. Testers execute the cases and compare the actual results with the expected outcomes. Any discrepancies are documented and reported back to the development team for resolution. This phase is critical for identifying bugs and validating that the system meets the specified requirements. The results gathered during test execution provide valuable feedback for improving the system and ensuring it functions correctly before deployment.

2.7.2. Types of testing

1. Black Box Testing

Tester is not concerned about the internal design of the program. Test cases are created by studying the external program specification. The program is treated as a black box, to which data are input and from which outputs are taken and checked for correctness. If the outputs are correct, then the program is declared correct. Black box testing cannot guarantee the absence of errors. (geeksforgeeks, 2025)

2. White Box Testing

The tester is concerned with the internal design / logic of the program. The tester studies the internal logic and then uses his knowledge about the internal design to design test data. White box testing attempts to uncover hidden logic errors by testing every logic path. (geeksforgeeks, 2025)

2.7.3. Stage of testing

1. Unit Testing

This is concerned with testing individual modules/functions. Each function of the program is tested separately. This should be done using white box testing. Test cases should be designed to cover every logic path. (geeksforgeeks, 2025)

2. Integration Testing

This involves testing links between modules/functions. Integration testing ensures that data is not lost between modules and that one module does not create a side effect in another module. Integration testing is done using black box testing method.

3. Regression Testing

When a modification is done to an already tested software unit, we must re-run the previous tests again to ensure that the modification does not create a new defect.

4. System testing

The entire system is tested as a whole. This includes Functional testing, User Interface testing, Help testing (check whether on-screen help facilities are adequate), recovery Testing (testing back up and recovery procedures) and security testing and Stress testing (taking the system to extremes to check whether the system can withstand the stress without crashing).

5. User acceptance testing (Validation testing)

This is a test involving both the developers and the end users of the system. The developers demonstrate system functions to the user/customer. The purpose of acceptance testing is to find out whether the new system satisfies the original user requirements.

2.8. Test automation

Automated testing uses specialized software to perform tasks that are typically done manually during the software testing process. This approach is widely adopted in agile and DevOps environments to run tests automatically, accelerating the development cycle and enabling continuous delivery (CD) of new code to users. Test automation is especially beneficial for large projects or those requiring repetitive testing and can also be applied to projects after an initial manual testing phase. It involves the creation and execution of test scripts, which helps streamline the testing process, improve efficiency, and enhance the overall quality of the software. (saumyasaxena, 2025)

Test scripts are automated instructions or code designed to perform specific actions on a software application and compare the actual output with the expected results. They are essential in modern software development and quality assurance, providing a way to systematically verify that software behaves as intended. Leading test automation tools such as Selenium, Pytest, Pandas for data frame testing, and Appium are widely used to create and execute these test scripts, ensuring thorough and efficient validation of software functionality and performance.

2.8.1. Test Automation Tools chosen for the dream book shop Data Analysis System

1. **PyTest**

Pytest framework is a Python based Test Automation Tool which widely used in the industry. It can be used to write various types of software tests, including unit tests, integration tests, end-to-end tests, and functional tests (pytest, 2025). It is a simple and easy to use tool. Pytest requires the developer to test scripts as Assertions. An Assertion is a statement that compares the actual result with the expected result. To pass the test the actual result must equal the expected result. If not, the test detects an error in the program.

Following example shows a test script written in Pytest

```
import pytest

def getAmount(quantity, price):
  return quantity * price

def test_getAmount():
  assert getAmount(5,10) == 50
```

Justification for choosing PyTest

The testing tool PyTest is written for testing Python programs. Since the book data analysis system is coded in Python, it makes sense to use PyTest. Another reason is the simplicity of

PyTest Scripts. The Scripts are written as Assertions. Assertions are simple statements that compare actual output with expected output. If they differ, an error is detected.

PyTest is an open-source tool which is given away free. So, there is no additional cost involved.

PyTest is very effective in test discovery, finding and running test cases automatically without extensive setup. PyTest also supports parametrized testing, allowing for bulk testing with different input parameters.

2. Pandas Test Automation

Pandas is an “add on” tool for Python programs that can effectively deal with large data sets. Pandas is used to store large data sets in standardized two dimensional tables called data frames. Once the data is loaded (say, from a CSV file), various data analytics can be performed on these frames. The module assert_frame_equal can be used to write test scripts that check the accuracy of data analytics on data frames.

Justification for choosing pandas Test automation

The dream book shop Data Analysis system works on Pandas data frames. Analytics such as branch totals, product totals and price analysis are performed on data frames and then converted to charts by Matplotlib. Testing data frame based analytics is different from normal testing of functions. This is because a data frame resulting from an analysis operation has many data rows. The best way to test the analytic is to write the expected result as a data frame and then write an assertion to compare the actual frame with the expected frame. Pandas offers assert_frame_equal (actual_result, exp_result) statement to compare two data frames. If they are not equal, an error is detected.

2.9. Types of tests to be used for the data analysis system

1. Unit Testing

The term unit refers to single function. Therefore, the term unit testing refers to testing of individual functions.

In the data analysis system there are many functions grouped into various classes. There are six variations of performAnalysis() function The admin class has logon() functin and the Strategy class has openMenu() function. Each of these functions need to be tested for correctness. All unit tests will be conducted using **automated testing.**

2. Integration Testing

This is the process of testing links between modules/functions. Integration testing ensures assembled software units communicate with each other without any problems. In the data analysis system there are many situations where one function calls another. For example, the openMenu() function of the strategySelector class calls different analytic functions. Integration testing is needed to ensure that those function calls get executed correctly. This test will be conducted **manually** as it is easier that way than via automated testing.

3. Regression Testing

The term regression testing refers to repeated testing of a software system to ensure that it still passes the previously conducted tests. This is needed after doing some changes or enhancements to the existing system. During the development, the data analysis system will undergo several modifications or changes. It is also possible for some new analytics to be added in the future. Whenever there is a change, regression testing will be performed to ensure that the new modification does not introduce new errors. All regression tests will be performed using **automated testing** since this requires redundant effort if done manually.

4. System Testing

The entire system is tested as a whole to ensure that it conforms to the user expectations. System testing include **functional tests, help tests, stress tests, security tests etc.** In the data analysis system there are six data analytic functions. Functional test will be conducted to ensure their correctness. This will be done via automated testing. Security testing is needed to ensure that

the Admin login works as intended. This will be done via automated testing. Help tests are needed to meassure the usability of the system. This will ensure that the users of the analysis system can operate it without any problems. This is done using manual testing. The system's data analysis will be performed with extremely large data sets to see whether it could withstand stress. This will be done using automated testing.

2.9.1. Different methods of testing used for dream book shop analysis system

There two main approaches to test a program. black testing and white box testing. With black box testing the There two main approaches to test a program: **Black Testing** and **White Box testing**.

With black box testing the deveoper treats the program module as a black box which will provide the otputs when the inputs are given. The internal program logic is not examined. Test cases are prepared from the external specification of the program.

With white box testing, the developer examines the internal design and creates test cases to cover every logic path. This takes more time but ensures that every statement is executed.

White box testing will be used to conduct the **unit tests** of the Data Analysis System. ***Unit tests are the most crucial of all tests since the units represent the individual functions that make up the system***. Spending time and resources for white box testing is justified because it is important to ensure the correctness of each individual data analytic function. ***Test automation scripts*** will be written for each function by closely studying its internal design.

Black box testing will be used to cconduct **Integration testing, Regression testing, System Testing and User Acceptance Testing.** This is because, the correctness of individual functions is already ascertained by white box testing and there is no further need to spend time and resources on white box testing. A series of black box tests could verify that the system as a whole is working. With acceptance testing it has to be black box testing since users have no idea about how the code works, all they need to see is that the system produces correct outputs.

2.10. Utilizing Multiple Design Patterns in the dream book shop analysis

Strategy Pattern

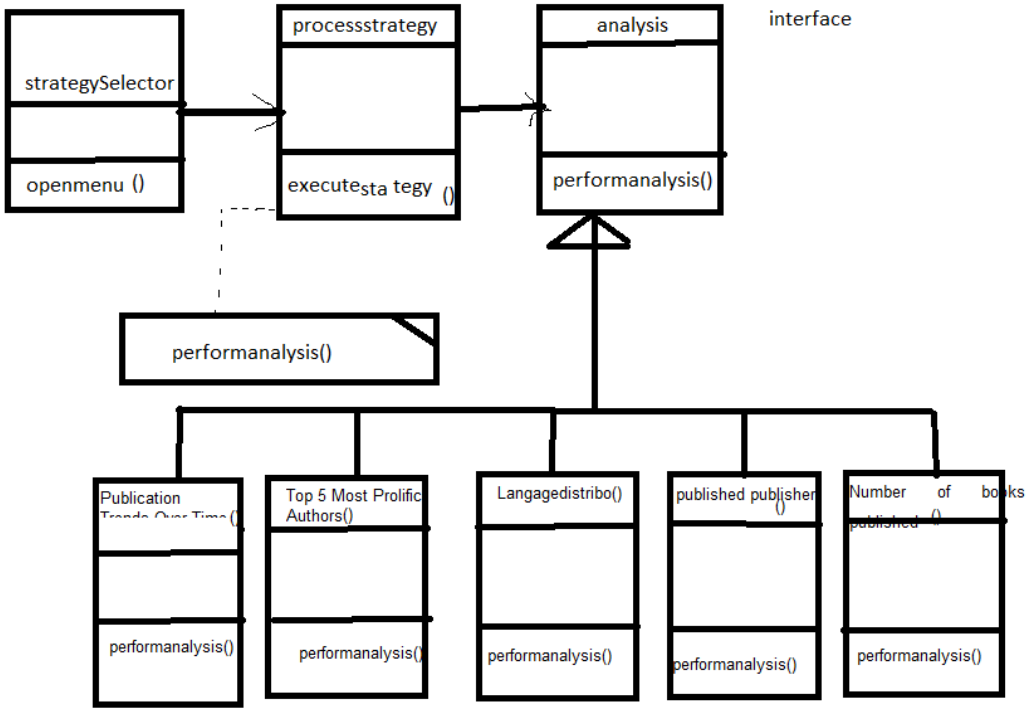
The Strategy pattern is actively used to allow dynamic selection of different analysis algorithms at runtime. When the user runs the system, they are presented with a menu that includes various types of analysis (e.g., top-selling books, total sales by year, average sales, genre-based trends, etc.).

Each analysis is implemented as a separate class that inherits from an abstract base class (like Analysis). The strategy is selected based on the user's menu choice, and the selected analysis strategy is then executed using a ProcessStrategy class. This helps decouple the strategy execution from the strategy selection logic, promoting a flexible and scalable design.

Singleton Pattern

The Singleton pattern is applied in the design of the Admin class. Since the Dream Book Shop CLI system is designed to have only one system administrator managing user access, reports, and high-level operations, the Admin class restricts instantiation and ensures that only a single object exists during the entire runtime.

This design avoids duplication of administrative controls and helps maintain a centralized management model.

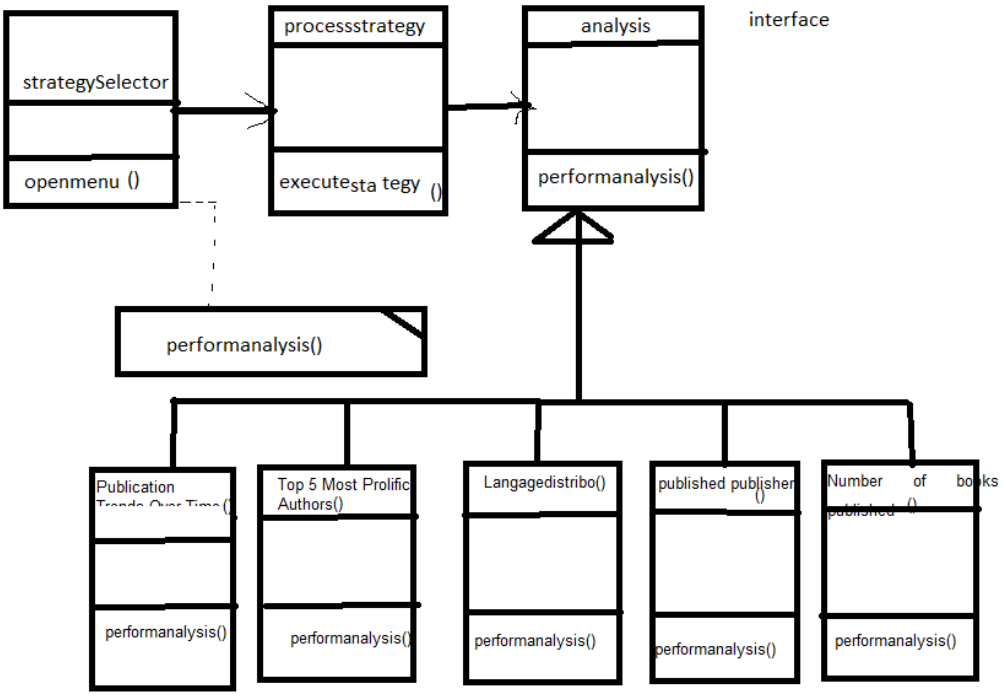


Abstract Factory Pattern

In the application, the Abstract Factory pattern supports creation of related analysis strategies without specifying their exact classes.

An abstract class named Analysis provides a template with an abstract method performAnalysis(). override this method with specific logic. These subclasses are then passed into the ProcessStrategy class via dependency injection. The ProcessStrategy.executeStrategy(analysis_object) method accepts any object from these subclasses, enabling substitution without altering the method logic (aligning with Liskov Substitution Principle from SOLID).

This factory structure simplifies code extension when new analysis types are introduced.



Observer Pattern

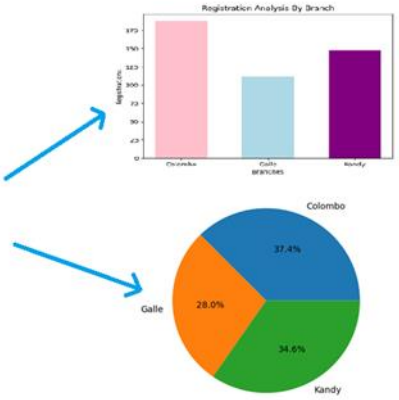
The Observer pattern is used to create a dynamic relationship between the dataset (subject) and various chart visualizations (observers), such as pie charts and bar graphs. When the dataset changes), all linked visualizations automatically update to reflect the new state.

This design pattern decouples data handling from the presentation logic. It is particularly useful if the system is extended in the future to include live dashboards or GUI-based visualization modules.

Example:

- Subject:DataSet
- Observers: BarChart, PieChart, etc.

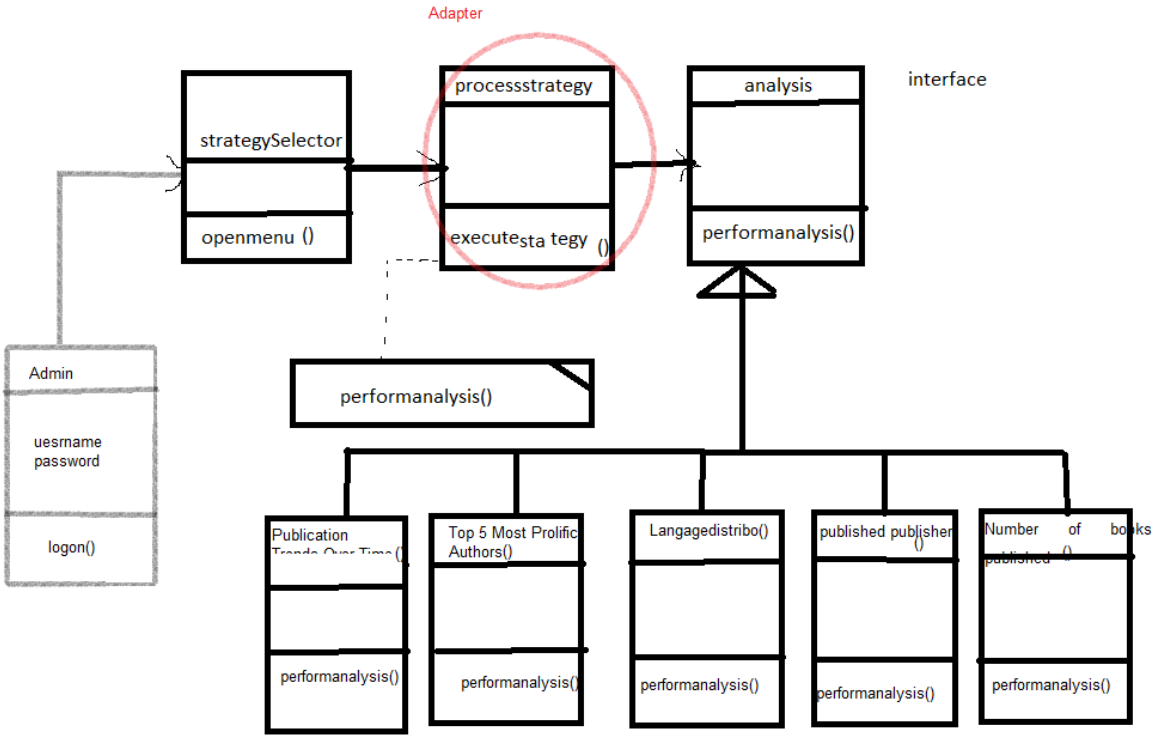
Branch	Course	Price	NoOfStudents
Kandy	BCS	150000	15
Colombo	HND	350000	100
Galle	HND	350000	10
Kandy	DIIT	70000	56
Colombo	BCS	160000	20
Galle	HND	350000	17
Galle	HND	360000	12
Kandy	BCS	170000	18
Galle	DIIT	75000	11
Kandy	HND	360000	3
Galle	BCS	170000	6



Adapter Pattern

The Adapter pattern is used to bridge the interaction between the StrategySelector class (user interface menu handler) and the Analysis class hierarchy (core logic engine). The ProcessStrategy class acts as an adapter to convert the user's choice into the correct analysis object.

Without the adapter pattern, each strategy would require direct integration with the selector logic, making the system harder to scale. This pattern simplifies extension and modification.



Activity 03 - IMPLEMENTATION OF THE DATA ANALYSIS SYSTEM

3.1. Overview

Dream Book Shop is a well-known local comic book retailer in the United Kingdom with over two decades of trading experience. With the recent expansion of their physical store and the rapid growth of their eCommerce platform, the company has identified the need to improve user experience and digital engagement through effective data analysis.

Currently, Dream Book Shop possesses a vast collection of bibliographic data sourced from the British National Bibliography (BNB), which includes metadata such as book titles, authors, publishers, publication years, languages, and ISBNs. However, they do not yet have an

automated system to process and analyze this data. At present, any form of data insight or pattern recognition is handled manually or through basic spreadsheet tools, which has proven inefficient and error-prone.

Drawbacks such as **manual errors, difficulty in identifying trends, time-consuming processes, and lack of real-time insights** hinder the business from making data-driven decisions that could improve product offerings and customer satisfaction. As a result, Dream Book Shop has partnered with AQ Digital Solutions to develop a tailored **CLI-based data analysis application** as an initial step toward building a full-fledged interactive dashboard.

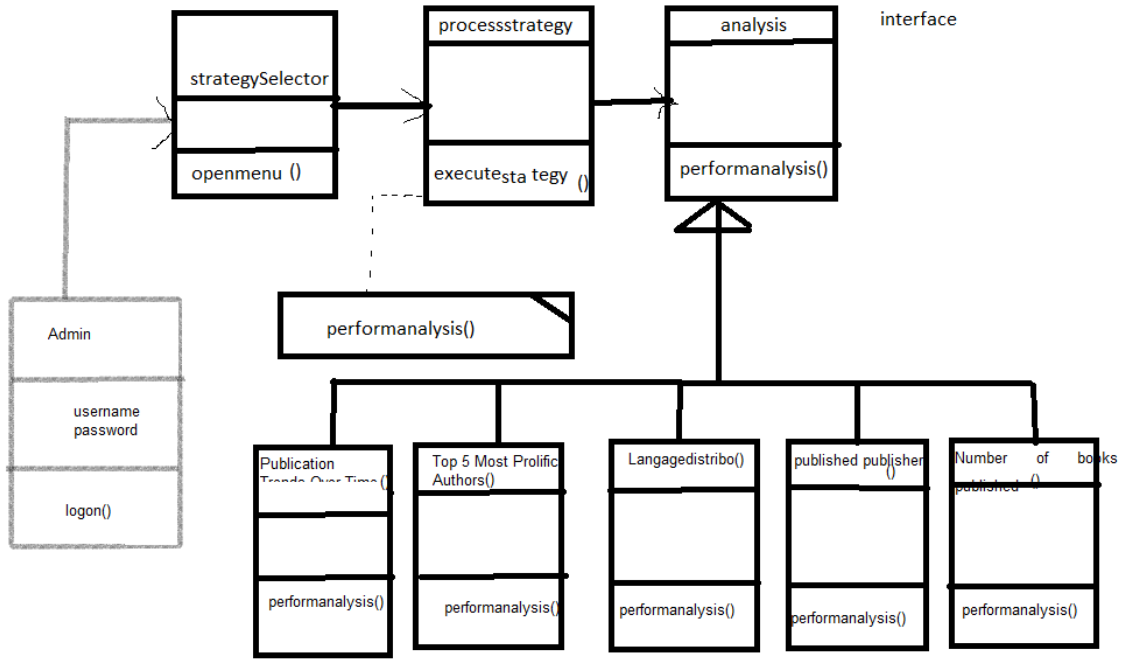
It will be implemented in **Python**, utilizing libraries such as **Pandas** for data processing and **Matplotlib** for visual analytics.

The application aims to provide the following core features:

1. **Publication Trends Over Time** – Analyze the number of books published each year.
2. **Top 5 Most Prolific Authors** – Identify the authors with the highest number of published books.
3. **Language Distribution** – Analyze and visualize the distribution of books across different languages.
4. **Books per Publisher** – Show how many books have been published by each publisher.
5. **Missing ISBN Analysis** – Identify and calculate the count and percentage of records without ISBNs.
6. **Yearly Language Trends** – Analyze number of books published per year categorized by language.

This system is intended to serve as a **proof of concept** that not only reduces reliance on manual data handling but also demonstrates how basic data science and visualization tools can be leveraged to inform business decisions and improve customer engagement.

3.1.1. Design of the Data Analysis System



3.2. The Large Data Set

A large dataset containing **bibliographic metadata of books**, including comic books, is available in **".csv" (Comma Separated Values)** format. This dataset can be viewed and edited using tools such as **Notepad** or **Microsoft Excel**, but for analysis, it is best handled programmatically through Python. The dataset was sourced from the **British National Bibliography (BNB)** and contains registration records of **books published in the United Kingdom** over several decades.

This data includes key information such as **book title, author name, publisher, publication year, language, ISBN**, and a unique **BNB ID** for each record.

The dataset serves as the foundation for generating insightful textual summaries and basic visualizations through the **CLI-based data analysis application** developed for **Dream Book Shop**. This application will help uncover trends and insights that support strategic decision-making and enhance user engagement with the shop's growing digital platform.

book	author	publication	language	book publisher	ISBN	BNB id
World politics : international relations and globalisation in the 21st century	Jeffrey Haynes	2023	English	SAGE	1529613827	GBC315766
World music : a global journey	J. Brown	2023	English	Routledge	1000909531/10009	GBC397766
Reflections of deviance	A. A. Lucas	2023	English	Severn House	1448308019	GBC2M0828
Reduction theory and arithmetic groups	Joachim Schwermer	2023	English	Cambridge University Press	1108832038	GBC2F3287
Tintin in Congo	Herge	2023	English	MAGNET	1000932218	GBC3C1738
A game of secrets	J. Brown	2023	English	Mills&Boon		GBC2K1871
A day in the life of Natalie Numbat	D. G. Lloyd	2023	English	Austin Macauley Publishers	1398449336	GBC3D2676
Albert A. Michelson and his interferometer : lord of the spinning worlds, master of light	A. A. Lucas	2023	English	Cambridge Scholars Publishing	1527504417	GBC3D8074
Atalanta : the mesmerising story of the only female Argonaut	Jennifer Saint	2023	English	Wildfire	1472292186	GBC327754
Bristol : danger, death & dark deeds	Cynthia Stiles	2023	English	Amberley Publishing	1398105355	GBC327600
Tintin in Tibet	Herge	2023	English	MAGNET	1915853653	GBC372362
Goodnight Mr Stone	A. A. Lucas	2023	English	Grosvenor House Publishing	1803814285	GBC346032
Find your own path : a life coach's guide to changing your life	Fiona Buckland	2023	English	Michael Joseph	241587317	GBC304022
Fritz and Kurt	Jeremy Dronfield	2023	English	Puffin		GBC2K1944
Mediterranean timescapes : chronological age and cultural practice in the Roman empire	Ray Laurence	2023	English	Routledge	1351973854	GBC341336
Murder make$ the world go round	Jeffrey Haynes	2023	English	Austin Macauley Publishers	1035816859	GBC3D2623
New institutional economics as situational logic ; a phenomenological perspective	Piet de Vries	2023	English	Routledge	1317653639	GBC3B7795
NSU Ro80 : the complete story	Martin Buckley	2023	English	The Crowood Press	719841750	GBC2K6209
Quantum phases of matter	Subir Sachdev	2023	English	Cambridge University Press	1009212694	GBC339244
Perloff's clinical recognition of congenital heart disease	Ariane J. Marelli	2023	English	North-Holland Biomedical Pres	323547826	GBC2J1661
Ootlin : a memoir	Jenni Fagan	2023	English	PenguinBooks	1804942147	GBC3A8419
On days like these : the lost memoir of a goalkeeper	Tim Rich	2023	English	Quercus	1529428582	GBC351813
The secret heir	Zuri Day	2023	English	Mills&Boon	8931629	GBC303981
The nanny game	Zuri Day	2022	English	Mills&Boon	8924362	GBC280615
Elektra	Jennifer Saint	2022	English	Wildfire	1472273932	GBC220938
Two rivals, one bed	Zuri Day	2022	English	Mills&Boon	8924607	GBC2G0188
Revolution and democracy in Ghana : the politics of Jerry John Rawlings	Jeffrey Haynes	2022	English	Routledge	1000837681/10008	GBC2L0228
Good morning Miss Shelby	Denise Lunt	2022	English	Grosvenor House Publishing	1803812304	GBC2J0267
The bone library	Jenni Fagan	2022	English	Morning Star	1788855211	GBC2B9218

3.3. Implementation

```
import pandas as pd
import matplotlib.pyplot as plt
from abc import ABC, abstractmethod


class Analysis(ABC):
    @abstractmethod
    def performAnalysis(self):
        pass
```

```
class PublicationTrendsOverTime(Analysis):
    def performAnalysis(self):
        #load csv
        data = pd.read_csv("C:/Users/user/Desktop/APDP PYTHON CODE/Books.CSV.csv")
        #column name
        columns = ['book', 'author', 'publication date', 'language', 'book publisher', 'ISBN', 'BNB id']
        data.columns = columns

        print("Books Published Per Year")
        books_per_year = data['publication date'].value_counts().sort_index()
        print(books_per_year)
        books_per_year.plot(kind='line')
        plt.xlabel('Year')
        plt.ylabel('Number of Books')
        plt.title("Books Published Per Year")
        plt.show()
        print("\n")
```

```
class BooksPerPublisher(Analysis):
    def performAnalysis(self):
        data = pd.read_csv("C:/Users/user/Desktop/APDP PYTHON CODE/Books.CSV.csv")
        columns = ['book', 'author', 'publication date', 'language', 'book publisher', 'ISBN', 'BNB id']
        data.columns = columns

        print("Books Per Publisher")
        publisher_counts = data['book publisher'].value_counts().head(40)
        print(publisher_counts)
        publisher_counts.plot(kind='bar', width=0.2)
        plt.xlabel('Name')
        plt.title('Publishers')
        plt.ylabel('Number of Books')
        plt.show()
        print("\n")
```

```
class BooksPerYearByLanguage(Analysis):
    def performAnalysis(self):
          #load csv
        data = pd.read_csv("C:/Users/user/Desktop/APDP PYTHON CODE/Books.CSV.csv")
           #column name
        columns = ['book', 'author', 'publication date', 'language', 'book publisher', 'ISBN', 'BNB id']
        data.columns = columns

        print("Books Published Per Year by Language")
        year_lang = data.groupby(['publication date', 'language']).size().unstack().fillna(0)
        print(year_lang.tail())
        year_lang.plot(kind='bar' , figsize=(12, 6), title='Books Per Year by Language')
        plt.xlabel('Year')
        plt.ylabel('Number of Books')
        plt.legend
        plt.title('Language')
        plt.show()
        print("\n")
```

```
#ADMIN CLASS
class Admin:
    count=0
    def __init__ (self,username,password):
        if (Admin.count==0):
            self.username = username
            self.password = password
            Admin.count = Admin.count + 1
        else :
            print ("Admin already exists")

    def logon(self):
       print("Data analytics and visual representation of Dream Book Shop")
       print("------***--------------**************---------------***-----------")
       un = input ("Enter the User Name :")
       pw = input ("Enter the Password :")
       if (un == self.username and pw == self.password):
          selector = strategyselector()
          selector.run()
       else:
           print("Incorrect User Name OR Password")
```

```
class MissingISBNAnalysis(Analysis):
    def performAnalysis(self):
          #load csv
        data = pd.read_csv("C:/Users/user/Desktop/APDP PYTHON CODE/Books.CSV.csv")
           #column name
        columns = ['book', 'author', 'publication date', 'language', 'book publisher', 'ISBN', 'BNB id']
        data.columns = columns

        print("Missing ISBN Analysis")
        total = len(data)
        missing_isbn = data['ISBN'].isnull().sum()
        percent_missing = (missing_isbn / total) * 100
        print(f"Missing ISBN count: {missing_isbn}")
        print(f"Missing ISBN percentage: {percent_missing:}%")
        print("\n")
```

```
#main method
analysis = PublicationTrendsOverTime()
analysis.performAnalysis()

analysis = Top5Authors()
analysis.performAnalysis()

analysis = LanguageDistribution()
analysis.performAnalysis()

analysis = BooksPerPublisher()
analysis.performAnalysis()

analysis = MissingISBNAnalysis()
analysis.performAnalysis()

analysis = BooksPerYearByLanguage()
analysis.performAnalysis()
```

```
#main method
analysis = PublicationTrendsOverTime()
analysis.performAnalysis()
```

```
analysis = Top5Authors()
analysis.performAnalysis()
```

```
analysis = LanguageDistribution()
analysis.performAnalysis()
```

```
analysis = BooksPerPublisher()
analysis.performAnalysis()
```

```
analysis = MissingISBNAnalysis()
analysis.performAnalysis()
```

```
analysis = BooksPerYearByLanguage()
analysis.performAnalysis()
```

```
# Main method
if __name__ == "__main__":
    selector = strategyselector()
    selector.run()
```

```
# Main method
a1 = Admin("MOHAMMED AASHIK","1234567")
a1.logon()
```

```
class LanguageDistribution(Analysis):
    def performAnalysis(self):
        #load csv
        data = pd.read_csv("C:/Users/user/Desktop/APDP PYTHON CODE/Books.CSV.csv")
        #column name
        columns = ['book', 'author', 'publication date', 'language', 'book publisher', 'ISBN', 'BNB id']
        data.columns = columns

        print("Language Distribution")
        language_counts = data['language'].value_counts()
        print(language_counts)
        language_counts.plot(kind='pie', autopct='%1.1f%%')
        plt.xlabel('')
        plt.ylabel('')
        plt.title('Languages of Books')
        plt.show()
        print("\n")
```

```
class strategyselector:
    def openmenu(self):
        print("Data analytics and visual representation of Dream Book Shop")
        print("1 - Publication Trends Over Time")
        print("2 - Top 5 Most Prolific Authors")
        print("3 - Language Distribution")
        print("4 - Publisher Count")
        print("5 - Missing ISBN Analysis")
        print("6 - Books Per Year by Language")
        print("7 - Exit")

    def run(self):
        while True:
            self.openmenu()
            try:
                choice = int(input("Enter choice [1|2|3|4|5|6|7]: "))

                if choice == 1:
                    analysis = PublicationTrendsOverTime()
                    analysis.performAnalysis()

                elif choice == 2:
                    analysis = Top5Authors()
                    analysis.performAnalysis()

                elif choice == 3:
                    analysis = LanguageDistribution()
                    analysis.performAnalysis()

                elif choice == 4:
                    analysis = BooksPerPublisher()
                    analysis.performAnalysis()

                elif choice == 5:
                    analysis = MissingISBNAnalysis()
                    analysis.performAnalysis()

                elif choice == 6:
                    analysis = BooksPerYearByLanguage()
                    analysis.performAnalysis()

                elif choice == 7:
                    print("****Exit from dream book shop****...")
                    break
```

```
            elif choice == 3:
                analysis = LanguageDistribution()
                analysis.performAnalysis()

            elif choice == 4:
                analysis = BooksPerPublisher()
                analysis.performAnalysis()

            elif choice == 5:
                analysis = MissingISBNAnalysis()
                analysis.performAnalysis()

            elif choice == 6:
                analysis = BooksPerYearByLanguage()
                analysis.performAnalysis()

            elif choice == 7:
                print("****Exit from dream book shop****...")
                break

            else:
                print("********** END *********.")

        except ValueError:
            print("Invalid input. Please enter a number from 1 to 7.")
```

```
class Top5Authors(Analysis):
    def performAnalysis(self):
          #load csv
        data = pd.read_csv("C:/Users/user/Desktop/APDP PYTHON CODE/Books.CSV.csv")
           #column name
        columns = ['book', 'author', 'publication date', 'language', 'book publisher', 'ISBN', 'BNB id']
        data.columns = columns

        print("Top 5 Authors by Number of Books")
        author_counts = data['author'].value_counts().head(5)
        print(author_counts)
        author_counts.plot(kind='bar')
        plt.title("Top 5 Authors")
        plt.xlabel("Name")
        plt.ylabel('Books Written')
        plt.show()
        print("\n")
```

3.5. Program Execution

System Start up - Admin Login

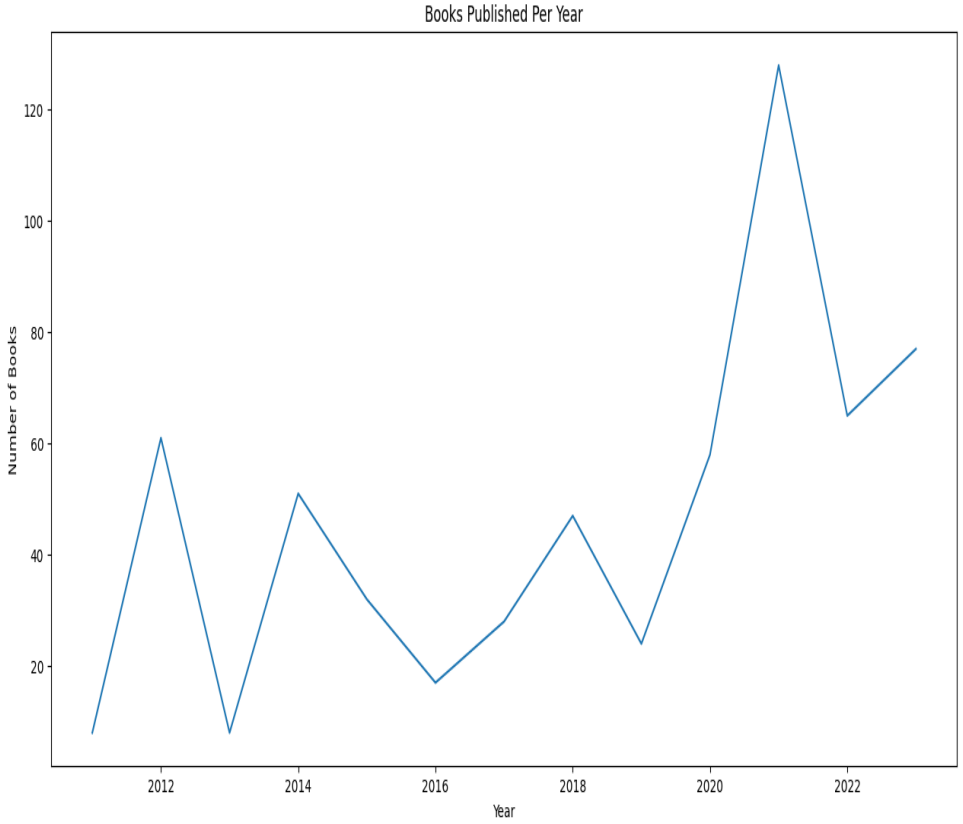
```
========== RESTART: C:\Users\user\Desktop\APDP PYTHON CODE\new code.py =========
Data analytics and visual representation of Dream Book Shop
------***-------------**************---------------***-----------
Enter the User Name :MOHAMMED AASHIK
Enter the Password :1234567
```

Menu -Display of Data Analysis Options

```
========== RESTART: C:\Users\user\Desktop\APDP PYTHON CODE\new code.py =========
Data analytics and visual representation of Dream Book Shop
------***-------------**************---------------***-----------
Enter the User Name :MOHAMMED AASHIK
Enter the Password :1234567
Data analytics and visual representation of Dream Book Shop
1 - Publication Trends Over Time
2 - Top 5 Most Prolific Authors
3 - Language Distribution
4 - Publisher Count
5 - Missing ISBN Analysis
6 - Books Per Year by Language
7 - Exit
Enter choice [1|2|3|4|5|6|7]:
```

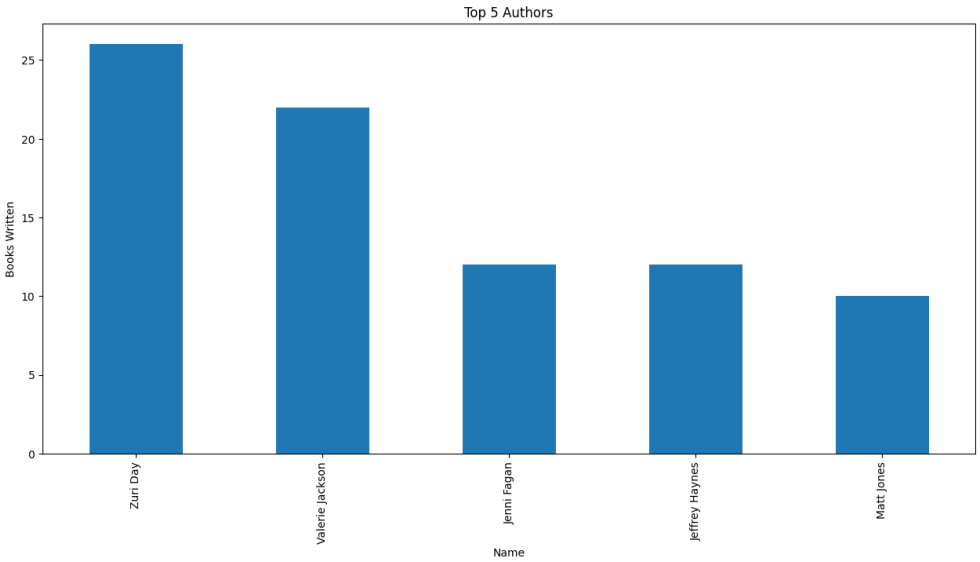
Data Analysis

```
Enter choice [1|2|3|4|5|6|7]: 1
Books Published Per Year
publication date
2011      8
2012     61
2013      8
2014     51
2015     32
2016     17
2017     28
2018     47
2019     24
2020     58
2021    128
2022     65
2023     77
```



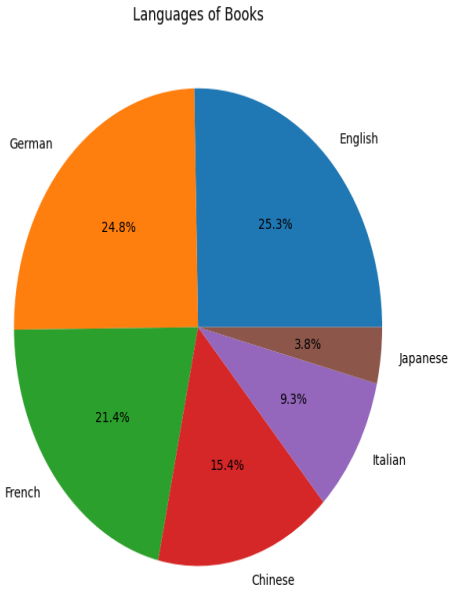
Data Analysis

```
Enter choice [1|2|3|4|5|6|7]: 2
Top 5 Authors by Number of Books
author
Zuri Day            26
Valerie Jackson     22
Jenni Fagan         12
Jeffrey Haynes      12
Matt Jones          10
```



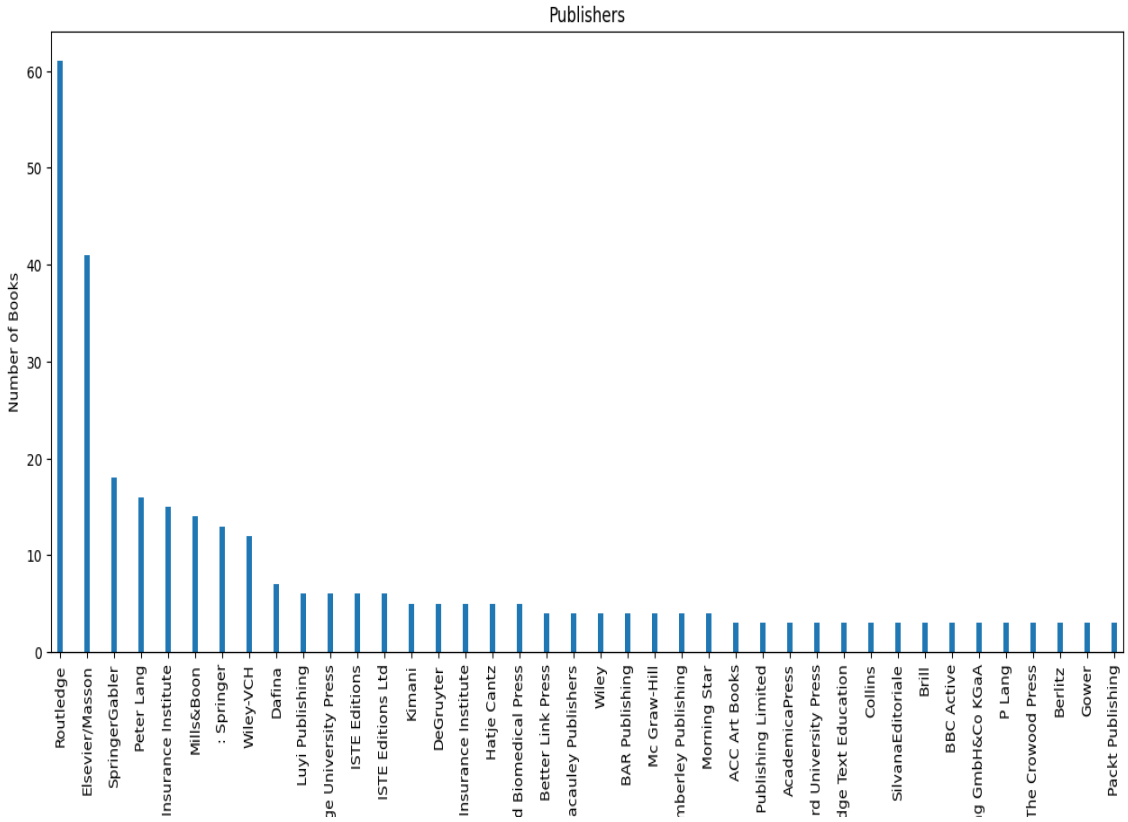
Data Analysis

```
Enter choice [1|2|3|4|5|6|7]: 3
Language Distribution
language
English     153
German      150
French      129
Chinese      93
Italian      56
Japanese     23
```



Data Analysis

```
Enter choice [1|2|3|4|5|6|7]: 4
Books Per Publisher
book publisher
Routledge                                61
Elsevier/Masson                          41
SpringerGabler                           18
Peter Lang                               16
Chartered Insurance Institute            15
Mills&Boon                               14
: Springer                               13
Wiley-VCH                                12
Dafina                                    7
Luyi Publishing                           6
Cambridge University Press                6
ISTE Editions                             6
ISTE Editions Ltd                         6
Kimani                                    5
DeGruyter                                 5
The Chartered Insurance Institute         5
Hatje Cantz                               5
North-Holland Biomedical Press            5
Better Link Press                         4
Austin Macauley Publishers                4
Wiley                                     4
BAR Publishing                            4
Mc Graw-Hill                              4
Amberley Publishing                       4
Morning Star                              4
ACC Art Books                             3
Grosvenor House Publishing Limited        3
AcademicaPress                            3
Oxford University Press                   3
Cambridge Text Education                  3
Collins                                   3
SilvanaEditoriale                         3
Brill                                     3
BBC Active                                3
Wiley-VCH Verlag GmbH&Co KGaA             3
P Lang                                    3
The Crowood Press                         3
Berlitz                                   3
Gower                                     3
Packt Publishing                          3
```

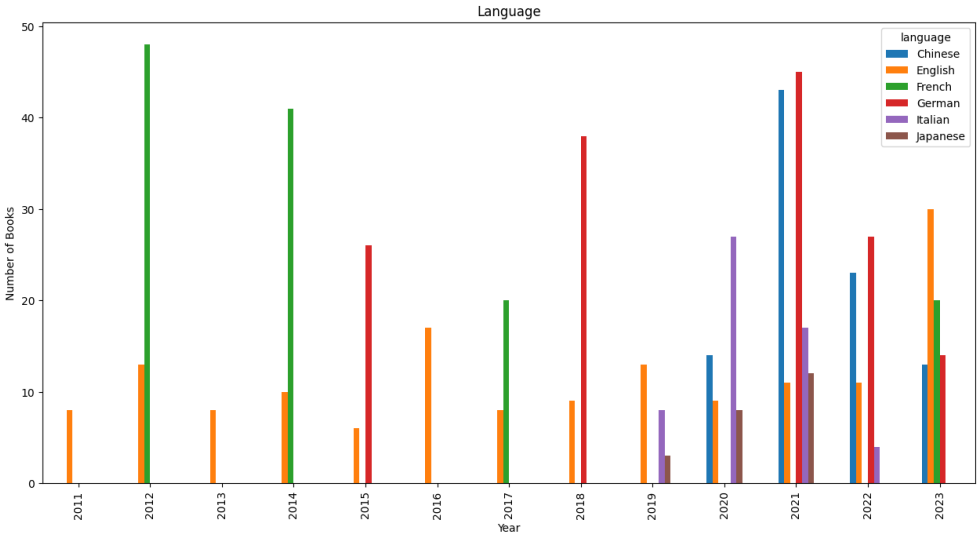


Data Analysis

```
Enter choice [1|2|3|4|5|6|7]: 5
Missing ISBN Analysis
Missing ISBN count: 9
Missing ISBN percentage: 1.49%
```

Data Analysis

```
Enter choice [1|2|3|4|5|6|7]: 6
Books Published Per Year by Language
language          Chinese  English  French  German  Italian  Japanese
publication date
2019                  0.0     13.0     0.0     0.0      8.0       3.0
2020                 14.0      9.0     0.0     0.0     27.0       8.0
2021                 43.0     11.0     0.0    45.0     17.0      12.0
2022                 23.0     11.0     0.0    27.0      4.0       0.0
2023                 13.0     30.0    20.0    14.0      0.0       0.0
```



Data Analysis

```
Enter choice [1|2|3|4|5|6|7]: 7
****Exit from dream book shop****...
```

3.6. Application of SOLID Principles in the Data Analysis System

01. Single Responsibility Principle

Each class in the Analysis System has a single responsibility.

Example: Admin class is responsible for log on and starting the system.

```
#ADMIN CLASS
class Admin:
    count=0
    def __init__ (self,username,password):
        if (Admin.count==0):
            self.username = username
            self.password = password
            Admin.count = Admin.count + 1
        else :
            print ("Admin already exists")

    def logon(self):
       print("Data analytics and visual representation of Dream Book Shop")
       print("------***-------------**************--------------***-----------")
       un = input ("Enter the User Name :")
       pw = input ("Enter the Password :")
       if (un == self.username and pw == self.password):
          selector = strategyselector()
          selector.run()
       else:
           print("Incorrect User Name OR Password")
```

02. Open/Closed Principle

The classes in the dream book shop Analysis System are open for extensions via sub classing and method overriding. Modifications are discouraged. For example, the abstract super class “Analysis” can be extended in future to include more types of analysis work.

```
class Analysis(ABC):
    @abstractmethod
    def  performAnalysis():
        pass
```

03. Liskov Substitution Principle

The “analysis ” class has “perform analysis ” method that takes objects of six different Analysis classes.

```
class Analysis(ABC):
    @abstractmethod
    def performAnalysis(self):
```

04. Interface Segregation Principle

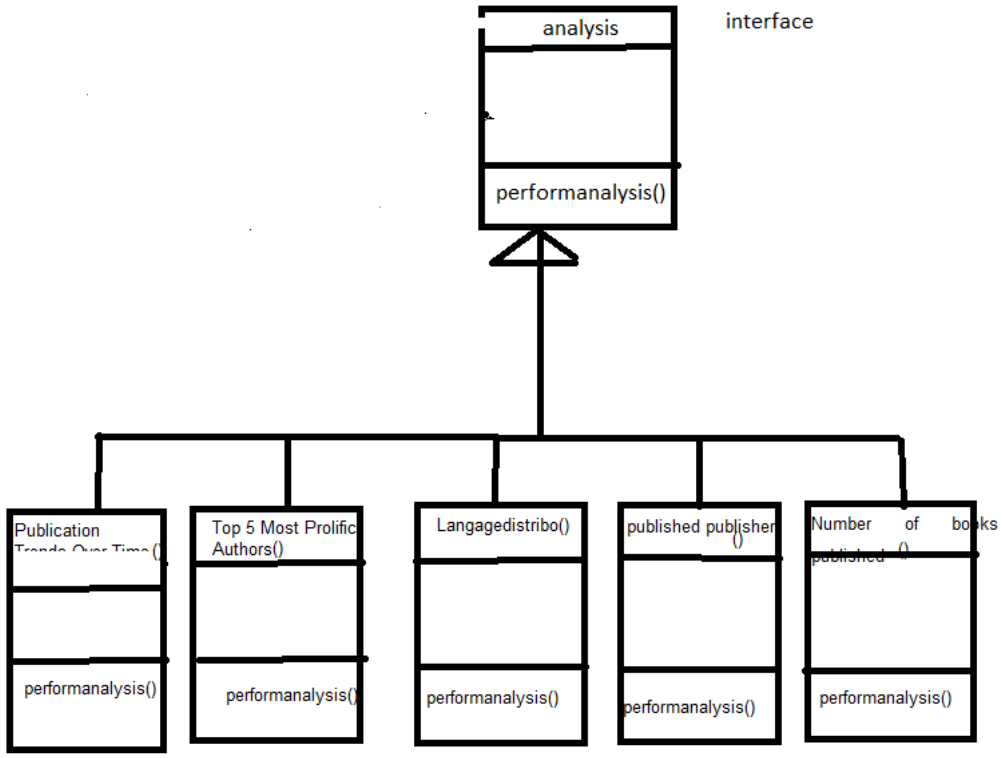
The abstract class “Analysis” with the “performAnalysis” method serves as an Interface for six different sub classes that perform six different analysis on data.

```
class Analysis(ABC):
    @abstractmethod
    def  performAnalysis():
        pass
```

```
class BooksPerYearByLanguage(Analysis):
    def performAnalysis(self):
        #load csv
        data = pd.read_csv("C:/Users/user/Desktop/APDP PYTHON CODE/Books.CSV.csv")
        #column name
        columns = ['book', 'author', 'publication date', 'language', 'book publisher', 'ISBN', 'BNB id']
        data.columns = columns

        print("Books Published Per Year by Language")
        year_lang = data.groupby(['publication date', 'language']).size().unstack().fillna(0)
        print(year_lang.tail())
        year_lang.plot(kind='bar' , figsize=(12, 6), title='Books Per Year by Language')
        plt.xlabel('Year')
        plt.ylabel('Number of Books')
        plt.legend
        plt.title('Language')
        plt.show()
        print("\n")
```

05. Dependency Inversion Principle

This principle says that concrete sub classes depend on the abstract super class. In our dream book shop analysis system, there are six classes that depend on the abstract super class “Analysis”.




```
class Analysis(ABC):
    @abstractmethod
    def  performAnalysis():
        pass
```

One of the concrete sub classes that depend on “Analysis” class

```
class LanguageDistribution(Analysis):
    def performAnalysis(self):
          #load csv
        data = pd.read_csv("C:/Users/user/Desktop/APDP PYTHON CODE/Books.CSV.csv")
           #column name
        columns = ['book', 'author', 'publication date', 'language', 'book publisher', 'ISBN', 'BNB id']
        data.columns = columns

        print("Language Distribution")
        language_counts = data['language'].value_counts()
        print(language_counts)
        language_counts.plot(kind='pie', autopct='%1.1f%%')
        plt.xlabel('')
        plt.ylabel('')
        plt.title('Languages of Books')
        plt.show()
        print("\n")
```

3.6.1. Assessing the effectiveness of SOLID principles in the Data Analysis System

SOLID is an acronym for 5 key concepts that contribute to high quality, flexible and efficient Object Oriented Designs. The resulting software is easy to maintain, easy to extend and easy to reuse. The 5 principles are as follows

The resulting software is easy to maintain, easy to extend and easy to reuse. The 5 principles are as follows:

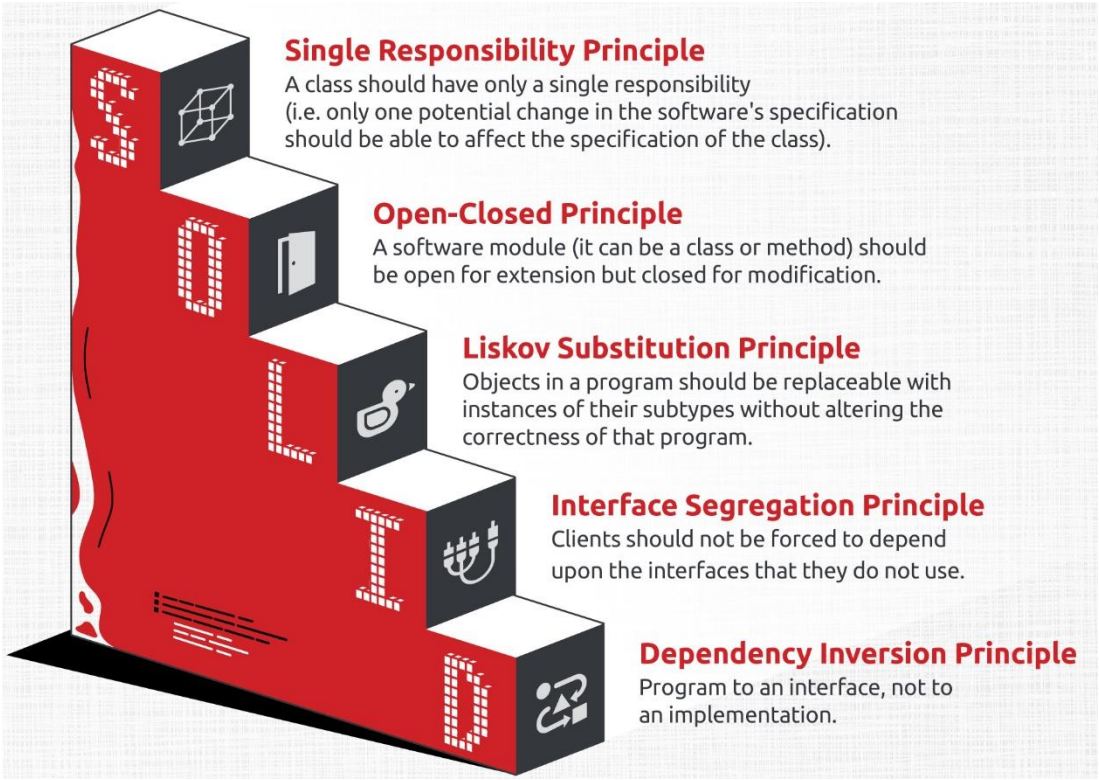
S - Single-responsibility Principle.

O - Open-closed Principle.

L - Liskov Substitution Principle.

I - Interface Segregation Principle.

D - Dependency Inversion Principle.



The Single Responsibility Principle dictates that a class should have only one responsibility, defined through its methods. In the analysis system, each class is dedicated to a specific task, such as the Admin class for logon functionality and the StrategySelector class for menu options. This design ensures that each class focuses on a single function, making it easier for programmers to locate and resolve issues without affecting other classes. Such a structure also enhances reusability, as these classes are independent and self-contained.

The Open-Closed Principle advocates against modifying existing, functional code to avoid unintended side effects. Instead, it promotes extending existing classes through subclassing, which is non-disruptive. For example, the abstract class “Analysis” can be extended by creating new subclasses to incorporate additional data analysis methods, allowing for flexible system expansion without altering the original code.

The Liskov Substitution Principle supports the ability to replace objects of a superclass with objects of its subclasses without altering the correctness of the program. In the analysis system, the ProcessStrategy class's executeStrategy() method can handle different types of analysis objects, allowing for interchangeable use of various subclass instances without changing the user code.

The Interface Segregation Principle suggests creating specific interfaces with relevant abstract methods, preventing subclasses from being forced to implement methods they do not need. This principle ensures that interfaces remain focused and tailored to the specific needs of each subclass.

The Dependency Inversion Principle emphasizes starting with abstractions and adding details later. In the Data Analysis System, the abstraction “Analysis” is separated from the concrete implementations of various analytics methods. This approach decouples the high-level policy from low-level details, enhancing flexibility and scalability.

Adhering to the SOLID principles results in code that is easier to understand, maintain, and extend. Such a codebase not only simplifies future development and maintenance but also facilitates reuse, ultimately saving time and reducing costs in software development.

3.7. Design Patterns used in the dream book shop analysis

1. Singleton Pattern

The dream book shop has only one administrator. The Admin class uses Singleton pattern to ensure that only one object instance of Admin is created.

```
#ADMIN CLASS
class Admin:
    count=0
    def __init__ (self,username,password):
        if (Admin.count==0):
            self.username = username
            self.password = password
            Admin.count = Admin.count + 1
        else :
            print ("Admin already exists")

    def logon(self):
       print("Data analytics and visual representation of Dream Book Shop")
       print("------***-------------***************---------------***-----------")
       un = input ("Enter the User Name :")
       pw = input ("Enter the Password :")
       if (un == self.username and pw == self.password):
          selector = strategyselector()
          selector.run()
       else:
           print("Incorrect User Name OR Password")
```

02. Strategy Pattern

The strategy pattern is used to select the correct data analysis function at run time. The user is given a menu that lists six different data analytics. When the user selects what he/she desires the strategy pattern chooses the correct one.

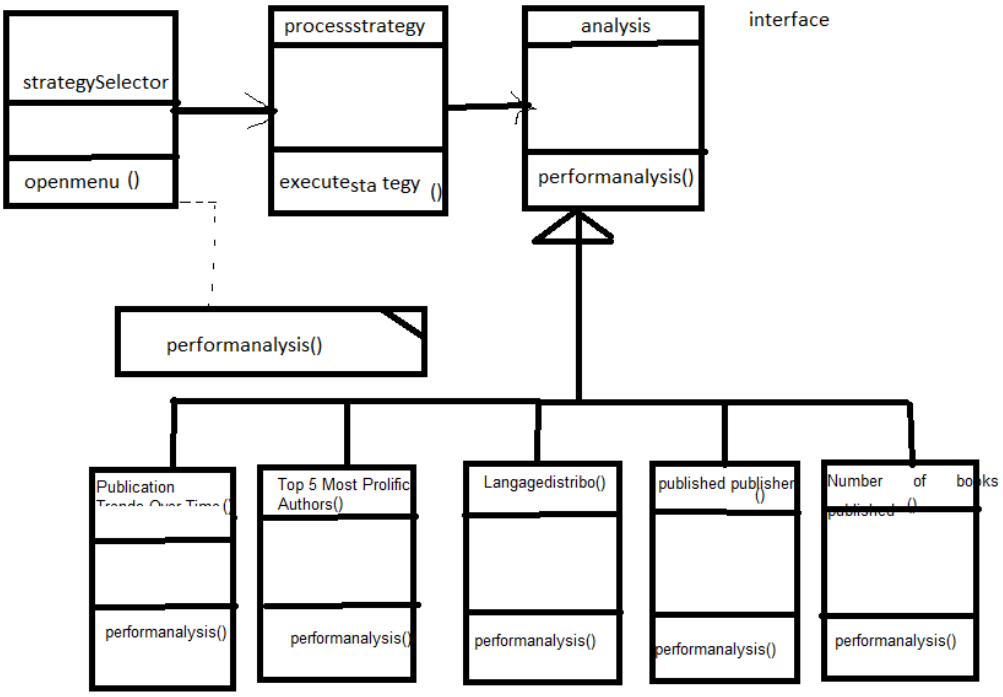
```
class strategyselector:
    def openmenu(self):
        print("Data analytics and visual representation of Dream Book Shop")
        print("1 - Publication Trends Over Time")
        print("2 - Top 5 Most Prolific Authors")
        print("3 - Language Distribution")
        print("4 - Publisher Count")
        print("5 - Missing ISBN Analysis")
        print("6 - Books Per Year by Language")
        print("7 - Exit")

    def run(self):
        while True:
            self.openmenu()
            try:
                choice = int(input("Enter choice [1|2|3|4|5|6|7]: "))

                if choice == 1:
                    analysis = PublicationTrendsOverTime()
                    analysis.performAnalysis()

                elif choice == 2:
                    analysis = Top5Authors()
                    analysis.performAnalysis()

                elif choice == 3:
                    analysis = LanguageDistribution()
                    analysis.performAnalysis()

                elif choice == 4:
                    analysis = BooksPerPublisher()
                    analysis.performAnalysis()

                elif choice == 5:
                    analysis = MissingISBNAnalysis()
                    analysis.performAnalysis()

                elif choice == 6:
                    analysis = BooksPerYearByLanguage()
                    analysis.performAnalysis()

                elif choice == 7:
                    print("****Exit from dream book shop****...")
                    break
```

```
            elif choice == 3:
                analysis = LanguageDistribution()
                analysis.performAnalysis()

            elif choice == 4:
                analysis = BooksPerPublisher()
                analysis.performAnalysis()

            elif choice == 5:
                analysis = MissingISBNAnalysis()
                analysis.performAnalysis()

            elif choice == 6:
                analysis = BooksPerYearByLanguage()
                analysis.performAnalysis()

            elif choice == 7:
                print("****Exit from dream book shop****...")
                break

            else:
                print("********** END *********.")

        except ValueError:
            print("Invalid input. Please enter a number from 1 to 7.")
```

03. Abstract Factory Pattern

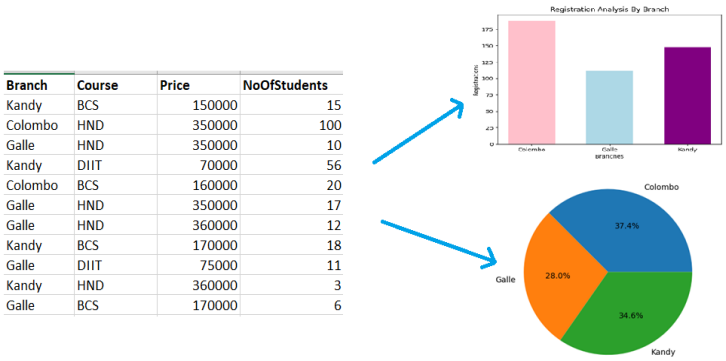
There are six concrete sub classes that decend from a single abstract class. Each of these sub classes override the “performAnalysis” method of the abstract super class. The “analysis ” class has “perform analysis ” method to deal with objects of those six sub classes. The perform analysis () method has analysis_object parameter which could belong to any of the six analytics classes. This is Liskov substitution.

```
class Analysis(ABC):
    @abstractmethod
    def performAnalysis(self):
```



04. Observer Pattern

The observer pattern establishes a one-to-many dependancy between a subject and a set of observers so that when the subject changes observers change too. Observer pattern is used in this system to establish a relationship between the data set and the visualization charts. The data set is the subject. The charts (Bar charts, Pie chart etc.) are the observers.

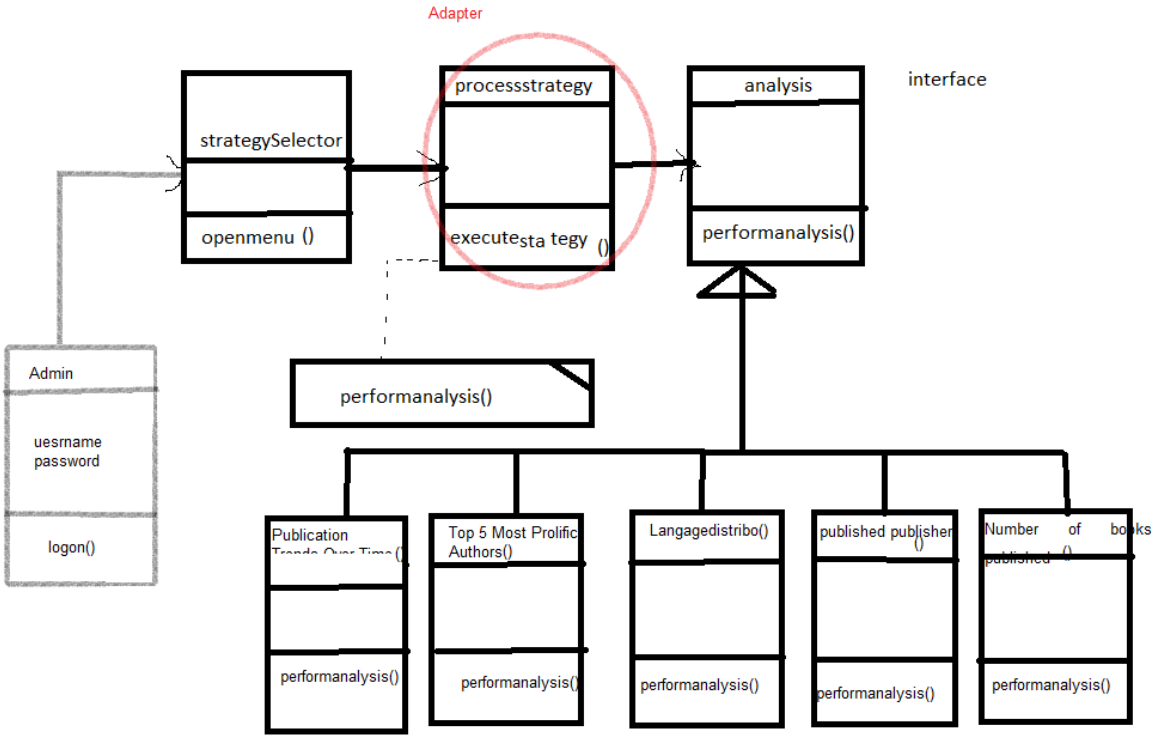


Branch	Course	Price	NoOfStudents
Kandy	BCS	150000	15
Colombo	HND	350000	100
Galle	HND	350000	10
Kandy	DIIT	70000	56
Colombo	BCS	160000	20
Galle	HND	350000	17
Galle	HND	360000	12
Kandy	BCS	170000	18
Galle	DIIT	75000	11
Kandy	HND	360000	3
Galle	BCS	170000	6

Figure 1 Using Observer Pattern

05. Adaptor Pattern

The adaptor pattern fills the gap between client and destination by converting one interface to another. In this system, the “ProcessStrategy” class serves as an adaptor between the “StrategySelector” class and the “Analysis” class hierarchy.



3.7.1. The Effectiveness of Design Patterns in the Data Analysis System

A design pattern is a problem/solution pair that can be reused in new contexts. It is a general, reusable solution to a commonly occurring problem in software development. This system uses SIX (6) design patterns – Singleton for Admin class, Strategy for choosing the correct analysis algorithm, Abstract factory for “Analysis” abstraction, “Adapter” between “StrategySelector” and “Analysis” classes and “Observer” pattern for allowing charts to change when the data set changes. By using these patterns, the quality of the Data Analysis System has imprpoved vastly. It has become more resuable, extensible and maintainable.

Patterns help designers reuse successful designs by basing new designs on prior experience. Having a large, well documented collection of reusable design patterns provides a repository of designer experience. If designers consult such a repository during development, they can potentially avoid re-inventing known solutions and save a considerable amount of time and effort.

Patterns are documented to a standard in software engineering literature. Pattern identification, problem addressed, solution, sample code, consequences etc. are well documented in such a way that developers could easily find what they are looking for. This makes reuse easier.

Design patterns are quality software designs invented by experienced programmers. They embody best of programming practices. When such design patterns are reused in a program it improves the readability of that program. Readability is the number one requirement for effective software maintenance. Since patterns are well documented in software engineering literature, it is likely that the maintenance programmer is already familiar with it. This allows him to quickly locate the problem and carry out the relevant modification. Often, software maintenance is carried out by programmers who were not involved in the original development. If well known design patterns were used in the original program, then the maintenance programmers could quickly get familiar with the system. As a result, this data analysis system will be easily maintaninable – for example to add new types of analystics in the future.

3.8. Use of Clean Coding Techniques in the Data Analysis System

Clean coding is concerned with following a set of guidelines to produce high quality code that is easy to understand, debug and maintain. Clean code also facilitates communication among team members.

These guidelines are as follows,

01. Meaningful and descriptive names

The names given to variables, classes and functions/methods must be meaningful and descriptive.

Example of descriptive class names and method names from the Data Analysis System:

```
class Analysis(ABC):
    @abstractmethod
    def  performAnalysis():
        pass
```

```
class LanguageDistribution(Analysis):
    def performAnalysis(self):
```

Example of descriptive variable names from the Data Analysis System:

```
    def run(self):
        while True:
            self.openmenu()
            try:
                choice = int(input("Enter choice [1|2|3|4|5|6|7]: "))
```

Example of descriptive class names and variable names from the Data Analysis System:

```
#ADMIN CLASS
class Admin:
    count=0
    def __init__ (self,username,password):
        if (Admin.count==0):
            self.username = username
            self.password = password
            Admin.count = Admin.count + 1
        else :
            print ("Admin already exists")
```

02. Comments

Where appropriate, comments must be provided to explain the code. They must explain in simple terms what the code does.

Example of comments from the data analysis system

```
#ADMIN CLASS
class Admin:
    count=0
    def __init__ (self,username,password):
        if (Admin.count==0):
            self.username = username
            self.password = password
            Admin.count = Admin.count + 1
        else :
            print ("Admin already exists")
```

```
class BooksPerYearByLanguage(Analysis):
    def performAnalysis(self):
          #load csv
        data = pd.read_csv("C:/Users/user/Desktop/APDP PYTHON CODE/Books.CSV.csv")
           #column name
        columns = ['book', 'author', 'publication date', 'language', 'book publisher', 'ISBN', 'BNB id']
        data.columns = columns
```

03. Indentation

Statements within loops and “if” conditions must be indented to add clarity. In Python, this is done automatically, but in other languages the programmer must indent the statements.

```
#ADMIN CLASS
class Admin:
    count=0
    def __init__ (self,username,password):
        if (Admin.count==0):
            self.username = username
            self.password = password
            Admin.count = Admin.count + 1
        else :
            print ("Admin already exists")

    def logon(self):
       print("Data analytics and visual representation of Dream Book Shop")
       print("------***-------------**************---------------***-----------")
       un = input ("Enter the User Name :")
       pw = input ("Enter the Password :")
       if (un == self.username and pw == self.password):
          selector = strategyselector()
          selector.run()
       else:
           print("Incorrect User Name OR Password")



class strategyselector:
    def openmenu(self):
        print("Data analytics and visual representation of Dream Book Shop")
        print("1 - Publication Trends Over Time")
        print("2 - Top 5 Most Prolific Authors")
        print("3 - Language Distribution")
        print("4 - Publisher Count")
        print("5 - Missing ISBN Analysis")
        print("6 - Books Per Year by Language")
        print("7 - Exit")

    def run(self):
        while True:
            self.openmenu()
            try:
                choice = int(input("Enter choice [1|2|3|4|5|6|7]: "))

                if choice == 1:
                    analysis = PublicationTrendsOverTime()
                    analysis.performAnalysis()

                elif choice == 2:
                    analysis = Top5Authors()
                    analysis.performAnalysis()

                elif choice == 3:
                    analysis = LanguageDistribution()
                    analysis.performAnalysis()

                elif choice == 4:
                    analysis = BooksPerPublisher()
                    analysis.performAnalysis()

                elif choice == 5:
                    analysis = MissingISBNAnalysis()
                    analysis.performAnalysis()

                elif choice == 6:
                    analysis = BooksPerYearByLanguage()
                    analysis.performAnalysis()

                elif choice == 7:
                    print("****Exit from dream book shop****...")
                    break
```

04. Single task functions

A function must perform a single task. This will make any subsequent modifications easier.

```
class BooksPerYearByLanguage(Analysis):
    def performAnalysis(self):
        #load csv
        data = pd.read_csv("C:/Users/user/Desktop/APDP PYTHON CODE/Books.CSV.csv")
        #column name
        columns = ['book', 'author', 'publication date', 'language', 'book publisher', 'ISBN', 'BNB id']
        data.columns = columns

        print("Books Published Per Year by Language")
        year_lang = data.groupby(['publication date', 'language']).size().unstack().fillna(0)
        print(year_lang.tail())
        year_lang.plot(kind='bar' , figsize=(12, 6), title='Books Per Year by Language')
        plt.xlabel('Year')
        plt.ylabel('Number of Books')
        plt.legend
        plt.title('Language')
        plt.show()
        print("\n")
```

05. Coding standards

When writing names of attributes, methods etc. use an established standard such as camelCase notation and snake case notation.

```
class Analysis(ABC):
    @abstractmethod
    def performAnalysis(self):
        pass
```

```
class BooksPerYearByLanguage(Analysis):
    def performAnalysis(self):
```

3.8.1. The Effectiveness of Clean Coding on the Data Analysis System

Clean coding techniques play a crucial role in enhancing the readability, reusability, and maintainability of the data structures and operations within the system. Descriptive naming conventions are consistently applied to variables, functions, classes, and data structures, providing clear indications of their purpose and the data they hold. This clarity in naming aids in understanding the system's components and their functions. Each function is designed to perform a single task, promoting simplicity and reducing the complexity of the codebase.

Additionally, comments are strategically placed to clarify complex code segments, and proper indentation is maintained in loops and "if else" statements to improve the structural organization of the code. Naming constants descriptively and following coding standards, such as camelCase, further enhance the clarity and consistency of the code. The effectiveness of clean coding techniques can be observed in several key areas. Firstly, readability and maintenance are greatly improved. Clean code prioritizes clarity, making it easier for developers to read, understand, and modify the code. This emphasis on readability reduces the time required to grasp the functionality of the code, leading to faster development and maintenance processes. Secondly, clean coding fosters better team collaboration. By adhering to established coding standards and writing readable code, developers can easily comprehend each other's work, facilitating smoother communication and cooperation within the team.

Clean code simplifies debugging and issue resolution. Its clear structure, meaningful variable names, and well defined functions make it easier to navigate the codebase and pinpoint specific sections that may need attention. This clarity aids in quickly identifying and resolving issues, thereby improving the efficiency of the debugging process. Lastly, clean coding contributes to improved software quality and reliability. By following established coding standards and maintaining well structured code, the risk of introducing errors is minimized, resulting in more robust and dependable software. Application of clean coding practices leads to better structured and more maintainable software. It not only enhances the readability and quality of the code but also facilitates easier collaboration among team members, more efficient debugging, and higher reliability of the software. These benefits ultimately save time and reduce costs in the software development lifecycle, underscoring the importance of clean coding techniques in producing high quality software.

Activity 04

4.1. Automated testing

Automated testing is the use an automation tool to execute test cases. This is well-suited for projects that are large in size or require testing to be repeated multiple times. It also could be applied to projects that already have been through an initial manual testing process. Test automation requires test scripts (i.e. Small pieces of code that compares actual result with expected results) to be written in the syntax of the chosen test automation tool.

Test automation tools chosen for the data analysis system

1. **Pytest**

Pytest is a python based test automation tool which widely used in the industry. It can be used to write various types of software tests, including unit tests, integration tests, end-to-end tests, and functional tests. It is a simple and easy to use tool. Pytest requires the developer to test scripts as assertions. An assertion is a statement that compares the actual result with the expected result.

2. **Pandas test automation**

Pandas is an “add on” tool for python programs that can effectively deal with large data sets. Pandas is used to store large data sets in standardized two dimensional tables called data frames. The module assert_frame_equal can be used to write test scripts that check the accuracy of data analytics on data frames.

Test Objectives

The main objective of automated testing for the data processing CLI application is to verify that each analysis feature performs correctly and produces accurate results. It is essential to ensure that the system remains reliable across different dataset conditions, such as varying data sizes, missing values, or unexpected formats. Automated tests also help confirm that the application responds appropriately to user inputs, including both valid and invalid entries. Additionally, the system must be capable of efficiently reading and processing large CSV files without crashing or producing delays. By achieving these goals, the testing process ensures that the application meets its functional requirements and delivers a stable and user-friendly experience.

Test Scenarios

In the testing process for the data processing CLI application, several key test scenarios were identified to ensure comprehensive validation of the system's functionality and robustness. One of the primary scenarios involved verifying the correct loading and parsing of CSV data. Since the application depends heavily on external datasets, it is crucial to confirm that it can accurately read and interpret the contents of the provided CSV file, including handling various delimiters, inconsistent spacing, or unexpected characters.

Another critical scenario tested the application's ability to perform correct analysis across all six available options. These include operations such as identifying the most frequent author, calculating the number of books published per year by language, and other data-driven insights. Each analysis function had to be validated against known inputs to ensure it produced correct and meaningful results. Additionally, edge cases were explored, such as testing how the application behaves when the dataset is empty or contains missing or corrupt values. These tests are essential to ensure the application can fail gracefully without crashing or producing misleading results.

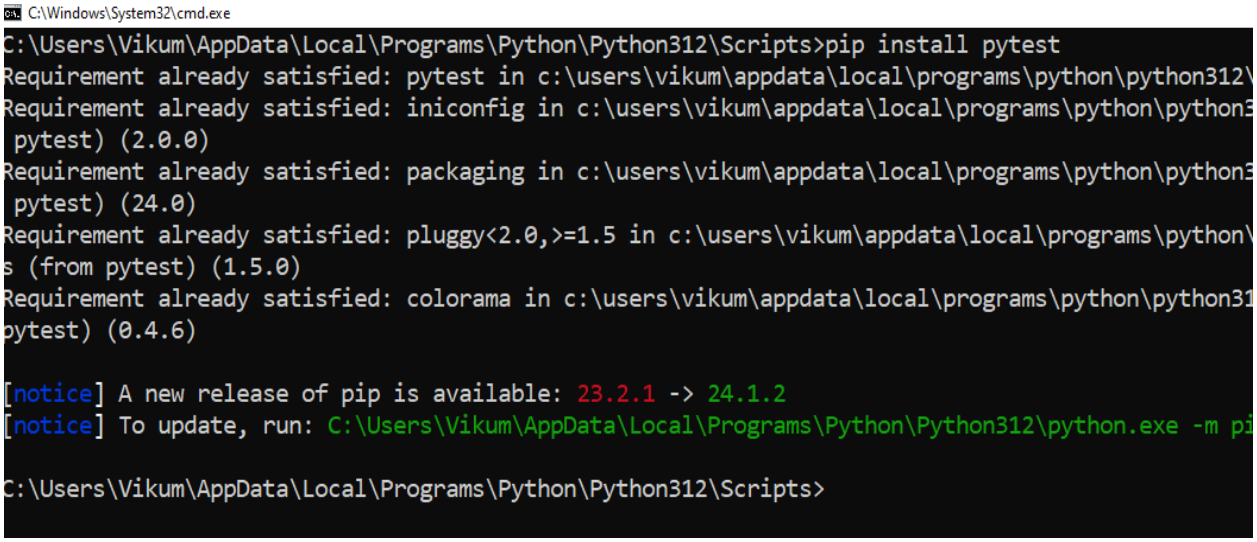
Furthermore, the testing process focused on the accuracy and consistency of the output format displayed in the command-line interface (CLI). The output needed to be clear, readable, and properly structured so users can easily interpret the results. Finally, user input handling was

thoroughly tested to confirm that the system could reject invalid or unexpected inputs gracefully. This included scenarios such as typing letters instead of numbers in menu options or selecting out-of-range values. Each scenario ensured a reliable and user-friendly experience.

4.1.1. Implementing automated testing with Pytest

Pytest is a Python based **Test Automation Tool** which widely used in the industry. It can be used to write various types of software tests, including unit tests, integration tests, end-to-end tests, and functional tests. It is a simple and easy to use tool.

Install PyTest using “pip install pytest”


C:\Windows\System32\cmd.exe
C:\Users\Vikum\AppData\Local\Programs\Python\Python312\Scripts>pip install pytest
Requirement already satisfied: pytest in c:\users\vikum\appdata\local\programs\python\python312\
Requirement already satisfied: iniconfig in c:\users\vikum\appdata\local\programs\python\python3
pytest) (2.0.0)
Requirement already satisfied: packaging in c:\users\vikum\appdata\local\programs\python\python3
pytest) (24.0)
Requirement already satisfied: pluggy<2.0,>=1.5 in c:\users\vikum\appdata\local\programs\python\
s (from pytest) (1.5.0)
Requirement already satisfied: colorama in c:\users\vikum\appdata\local\programs\python\python31
ytest) (0.4.6)
[notice] A new release of pip is available: 23.2.1 -> 24.1.2
[notice] To update, run: C:\Users\Vikum\AppData\Local\Programs\Python\Python312\python.exe -m pi
C:\Users\Vikum\AppData\Local\Programs\Python\Python312\Scripts>

Automated unit testing with pytest

Unit testing involves testing individual **functions** in the program. When you want to test a function you must write a test function starting with the word "**test**". Next, write an assertion within the test function to define **Expected Output**. Pytest is used to test the functionality of various computational functions in the data analysis system (there are several functions such as logon(), getAmount() and getDiscount() that perform important tasks within the analysis system). For data the five main data analytics based on data frames, Pandas test automation will be used.

Test Case 1: Test **getAmount()** function

Input	Expected Output	Comment
Quantity = 50, Price=10	500	Expected output matches input

Test Script

```
import pytest

def getAmount(quantity, price):
    return quantity*price


def test_getAmount():
    assert getAmount(50,10)==500
```

Save the program in a **Folder**. Let us use the file name **test1.py**. Next, open command line in the Python scripts folder and run the test script by typing **pytest test1.py**

```
G:\Esoft\Sem 04\APDP\activity 4>pytest test1.py
============================================= test session starts =============================================
platform win32 -- Python 3.12.3, pytest-8.3.1, pluggy-1.5.0
rootdir: G:\Esoft\Sem 04\APDP\activity 4
collected 1 item

test1.py .                                                                                          [100%]

============================================== 1 passed in 0.02s ==============================================
```

The test result indicates that the test has passed (i.e. expected output matches actual output).

Test Case 2: Test **getAmount()** function with wrong output

Input	Expected Output	Comment
Quantity = 10, Price=3	60	Expected output does not match input

Test Script

```
import pytest

def getAmount(quantity, price):
    return quantity*price


def test_getAmount():
    assert getAmount(10,3)==60
```

Test results

```
G:\Esoft\Sem 04\APDP\activity 4>pytest test1.py
============================================= test session starts =============================================
platform win32 -- Python 3.12.3, pytest-8.3.1, pluggy-1.5.0
rootdir: G:\Esoft\Sem 04\APDP\activity 4
collected 1 item

test1.py F                                                                                            [100%]

================================================== FAILURES ===================================================
_____________________________________________ test_getAmount _____________________________________________

    def test_getAmount():
>       assert getAmount(10,3)==60
E       assert 30 == 60
E        +  where 30 = getAmount(10, 3)

test1.py:8: AssertionError
=========================================== short test summary info ===========================================
FAILED test1.py::test_getAmount - assert 30 == 60
============================================== 1 failed in 0.09s ==============================================
```

Test Case 3: Test **getDiscount()** function – calculate 5% discount where amount>=5000 otherwise no discount.

Input	Expected Output	Comment
Amount = 10000	500	Expected output matches input

Test Script

```
import pytest

def getDiscount(amount):
    if (amount>=5000):
        return amount*5/100
    else:
        return 0

def test_getDiscount():
    assert getDiscount(10000)==500
```

Test results

```
G:\Esoft\Sem 04\APDP\activity 4>pytest test2.py
============================================== test session starts ==============
platform win32 -- Python 3.12.3, pytest-8.3.1, pluggy-1.5.0
rootdir: G:\Esoft\Sem 04\APDP\activity 4
collected 1 item

test2.py .

============================================== 1 passed in 0.03s ================
```

Test Case 4: Test **getDiscount()** function – calculate 5% discount where amount>=5000 otherwise no discount with wrong output.

Input	Expected Output	Comment
Amount = 400	200	Expected output does not match input

Test Script

```
import pytest

def getDiscount(amount):
    if (amount>=5000):
        return amount*5/100
    else:
        return 0

def test_getDiscount():
    assert getDiscount(4000)==200
```

Test results

```
G:\Esoft\Sem 04\APDP\activity 4>pytest test2.py
============================================= test session starts ==============
platform win32 -- Python 3.12.3, pytest-8.3.1, pluggy-1.5.0
rootdir: G:\Esoft\Sem 04\APDP\activity 4
collected 1 item

test2.py F

================================================== FAILURES ====================
_____________________________________________ test_getDiscount _________________

    def test_getDiscount():
>       assert getDiscount(4000)==200
E       assert 0 == 200
E        +  where 0 = getDiscount(4000)

test2.py:10: AssertionError
=========================================== short test summary info ============
FAILED test2.py::test_getDiscount - assert 0 == 200
============================================== 1 failed in 0.08s ===============
```

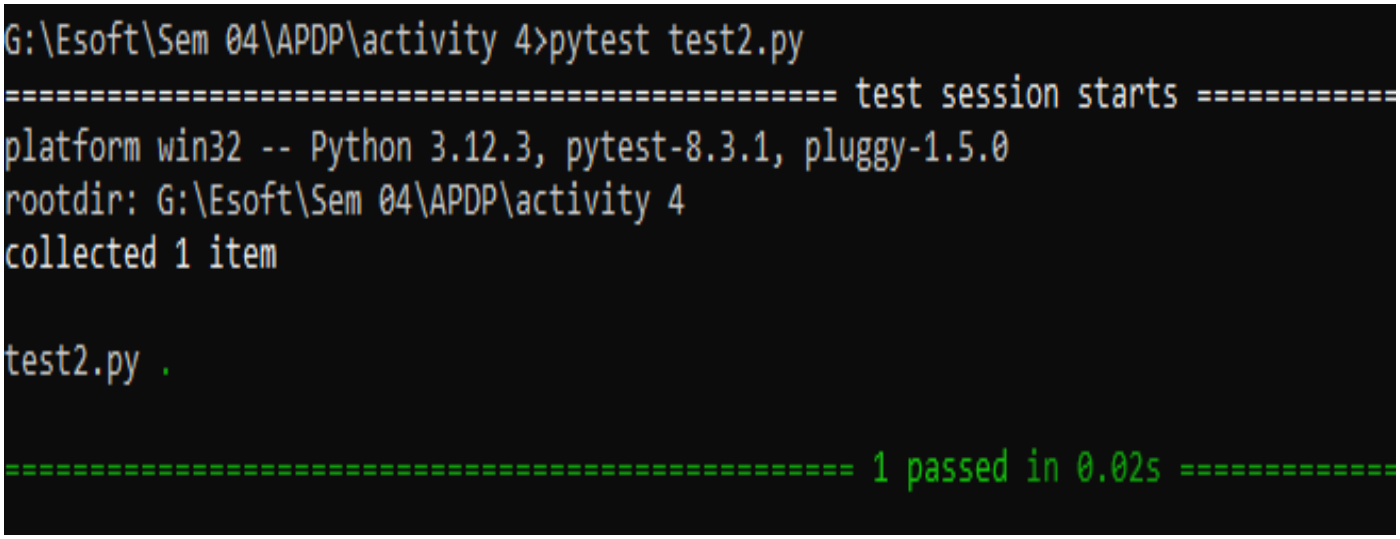
Test Case 5: Test **getDiscount()** function – calculate 5% discount where amount>=5000 otherwise no discount.

Input	Expected Output	Comment
Amount = 3500	0	Expected output matches input

Test Script

```
import pytest

def getDiscount(amount):
    if (amount>=5000):
        return amount*5/100
    else:
        return 0

def test_getDiscount():
    assert getDiscount(3500)==0
```

Test results


G:\Esoft\Sem 04\APDP\activity 4>pytest test2.py
test session starts
platform win32 -- Python 3.12.3, pytest-8.3.1, pluggy-1.5.0
rootdir: G:\Esoft\Sem 04\APDP\activity 4
collected 1 item
test2.py .
1 passed in 0.02s

4.1.2. Automated unit testing with pandas

Pandas is an “add on” tool for python programs that can effectively deal with **large data sets.** Pandas is used to store large data sets in standardized two dimensional tables called data frames. Once the data is loaded (say, from a csv file), various data analytics can be performed on these frames. The module **assert_frame_equal** can be used to write **test scripts** that check the accuracy of data analytics on data frames.

Test Case 1

```
class PublicationTrendsOverTime(Analysis):
    def performAnalysis(self):
        # Import necessary libraries
        import pandas as pd #IGNORE TYPE
        import matplotlib.pyplot as plt
        from pandas.testing import assert_frame_equal

        # Load the dataset
        df = pd.read_csv('C:/Users/user/Desktop/APDP PYTHON CODE/Books.CSV.csv')

        # Rename columns to readable format
        df.columns = ['book', 'author', 'publication date', 'language', 'book publisher', 'ISBN', 'BNB id']

        # Group by publication year and count number of books per year
        actual_result = df['publication date'].value_counts().sort_index().reset_index()
        actual_result.columns = ['publication date', 'book count']

        print("Actual Results")
        print(actual_result) # print summary

        # Manually prepare expected data for testing (example years and counts)
        exp_results = pd.DataFrame({
            'publication date': [2011, 2012, 2013, 2014, 2015, 2016, 2017, 2018, 2019, 2020, 2021, 2022, 2023],
            'book count': [8 ,61 ,8 ,51, 32 ,17, 28 ,47, 24 ,58, 128 ,65 ,77]
        })

        print("\n")
        print("Expected Results")
        print(exp_results)

        # Assertion to test actual vs expected
        assert_frame_equal(actual_result, exp_results)

        print("\n")
        print("Test Passed - Expected and Actual Frames are the same")
        print("\n")
```

```
Enter choice [1|2|3|4|5|6|7]: 1
Actual Results
    publication date  book count
0                2011           8
1                2012          61
2                2013           8
3                2014          51
4                2015          32
5                2016          17
6                2017          28
7                2018          47
8                2019          24
9                2020          58
10               2021         128
11               2022          65
12               2023          77


Expected Results
    publication date  book count
0                2011           8
1                2012          61
2                2013           8
3                2014          51
4                2015          32
5                2016          17
6                2017          28
7                2018          47
8                2019          24
9                2020          58
10               2021         128
11               2022          65
12               2023          77


Test Passed - Expected and Actual Frames are the same
```

Test Result Interpretation

The result indicates that the expected data frame and actual data frame are same. Provided expected result is accurate, this indicates that the data analytic code is error free.

Test Case 2

```python
class Top5Authors(Analysis):
    def performAnalysis(self):
        import pandas as pd # ignore type
        from pandas.testing import assert_frame_equal

        # Load CSV file
        df = pd.read_csv('C:/Users/user/Desktop/APDP PYTHON CODE/Books.CSV.csv')

        # Rename columns
        df.columns = ['book', 'author', 'publication date', 'language', 'book publisher', 'ISBN', 'BNB id']

        # Calculate top 5 authors by book count
        actual_result = df['author'].value_counts().head(5).reset_index()
        actual_result.columns = ['author', 'book count']

        print("Actual Results")
        print(actual_result)#print summary

        # Expected results (example, update values as needed)
        exp_results = pd.DataFrame({
            'author': ['Zuri Day', 'Valerie Jackson', 'Jenni Fagan', 'Jeffrey Haynes', 'Matt Jones'],
            'book count': [26,22 ,12 ,12 ,10 ]
        })
        print("\n")
        print("Expected Results")
        print(exp_results)

        # Assertion
        assert_frame_equal(actual_result, exp_results)
        print("\n")
        print("Test Passed - Expected and Actual Frames are the same")
        print("\n")
```

```
Enter choice [1|2|3|4|5|6|7]: 2
Actual Results
             author  book count
0          Zuri Day           26
1  Valerie Jackson            22
2       Jenni Fagan           12
3    Jeffrey Haynes           12
4        Matt Jones           10


Expected Results
             author  book count
0          Zuri Day           26
1  Valerie Jackson            22
2       Jenni Fagan           12
3    Jeffrey Haynes           12
4        Matt Jones           10


Test Passed - Expected and Actual Frames are the same
```

Test Result Interpretation

The result indicates that the expected data frame and actual data frame are same. Provided expected result is accurate, this indicates that the data analytic code is error free.

Test Case 3

```
class LanguageDistribution(Analysis):
    def performAnalysis(self):
        import pandas as pd
        from pandas.testing import assert_frame_equal

        # Load CSV
        df = pd.read_csv('C:/Users/user/Desktop/APDP PYTHON CODE/Books.CSV.csv')

        # Rename columns
        df.columns = ['book', 'author', 'publication date', 'language', 'book publisher', 'ISBN', 'BNB id']

        # Count by language
        actual_result = df['language'].value_counts().reset_index()
        actual_result.columns = ['language', 'book count']

        print("Actual Results")
        print(actual_result)

        # Expected results (example)
        exp_results = pd.DataFrame({
            'language': ['English',  'German' ,'French', 'Chinese', 'Italian' , 'Japanese'],
            'book count': [153, 150, 129, 93, 56, 23]
        })
        print("\n")
        print("Expected Results")
        print(exp_results)

        # Assertion
        assert_frame_equal(actual_result, exp_results)
        print("\n")
        print("Test Passed - Expected and Actual Frames are the same")
        print("\n")
```

```
Enter choice [1|2|3|4|5|6|7]: 3
Actual Results
   language  book count
0   English         153
1    German         150
2    French         129
3   Chinese          93
4   Italian          56
5  Japanese          23


Expected Results
   language  book count
0   English         153
1    German         150
2    French         129
3   Chinese          93
4   Italian          56
5  Japanese          23


Test Passed - Expected and Actual Frames are the same
```

Test Result Interpretation

The result indicates that the expected data frame and actual data frame are same. Provided expected result is accurate, this indicates that the data analytic code is error free.

Test Case 4

```
class BooksPerPublisher(Analysis):
    def performAnalysis(self):
        import pandas as pd
        from pandas.testing import assert_frame_equal

        # Load CSV
        df = pd.read_csv('C:/Users/user/Desktop/APDP PYTHON CODE/Books.CSV.csv')

        # Rename columns
        df.columns = ['book', 'author', 'publication date', 'language', 'book publisher', 'ISBN', 'BNB id']

        # Group by publisher
        actual_result = df['book publisher'].value_counts().head(40).reset_index()
        actual_result.columns = ['book publisher', 'book count']
        print("Actual Results")
        print(actual_result)

        # Expected data
        exp_results = pd.DataFrame({
            'book publisher': ['Routledge', 'Elsevier/Masson', 'SpringerGabler', 'Peter Lang',
        'Chartered Insurance Institute', 'Mills&Boon', ': Springer', 'Wiley-VCH',
        'Dafina', 'Luyi Publishing', 'Cambridge University Press', 'ISTE Editions',
        'ISTE Editions Ltd', 'Kimani', 'DeGruyter', 'The Chartered Insurance Institute',
        'Hatje Cantz', 'North-Holland Biomedical Press', 'Better Link Press',
        'Austin Macauley Publishers', 'Wiley', 'BAR Publishing', 'Mc Graw-Hill',
        'Amberley Publishing', 'Morning Star', 'ACC Art Books',
        'Grosvenor House Publishing Limited', 'AcademicaPress', 'Oxford University Press',
        'Cambridge Text Education', 'Collins', 'SilvanaEditoriale', 'Brill',
        'BBC Active', 'Wiley-VCH Verlag GmbH&Co KGaA', 'P Lang',
        'The Crowood Press', 'Berlitz', 'Gower', 'Packt Publishing' ],
            'book count': [ 61, 41, 18, 16, 15, 14, 13, 12, 7, 6, 6, 6, 6, 5, 5, 5, 5, 5, 4, 4, 4, 4,
        4, 4, 4, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3 ]
        })
        print("\n")
        print("Expected Results")
        print(exp_results)

        # Assertion
        assert_frame_equal(actual_result, exp_results)
        print("\n")
        print("Test Passed - Expected and Actual Frames are the same")
        print("\n")
```

```
Enter choice [1|2|3|4|5|6|7]: 4
Actual Results
                         book publisher  book count
0                              Routledge          61
1                          Elsevier/Masson          41
2                           SpringerGabler          18
3                              Peter Lang          16
4          Chartered Insurance Institute           15
5                              Mills&Boon          14
6                              : Springer          13
7                               Wiley-VCH          12
8                                  Dafina           7
9                          Luyi Publishing           6
10           Cambridge University Press            6
11                          ISTE Editions            6
12                      ISTE Editions Ltd            6
13                                 Kimani           5
14                              DeGruyter           5
15    The Chartered Insurance Institute             5
16                             Hatje Cantz          5
17       North-Holland Biomedical Press             5
18                     Better Link Press            4
19           Austin Macauley Publishers             4
20                                 Wiley            4
21                          BAR Publishing          4
22                            Mc Graw-Hill          4
23                   Amberley Publishing            4
24                           Morning Star           4
25                          ACC Art Books           3
26  Grosvenor House Publishing Limited              3
27                          AcademicaPress          3
28               Oxford University Press            3
29              Cambridge Text Education            3
30                               Collins            3
31                     SilvanaEditoriale            3
32                                 Brill            3
33                              BBC Active          3
34        Wiley-VCH Verlag GmbH&Co KGaA             3
35                                 P Lang           3
36                     The Crowood Press            3
37                               Berlitz            3
38                                 Gower            3
39                       Packt Publishing           3
```

```
Expected Results
                          book publisher  book count
0                                Routledge          61
1                           Elsevier/Masson          41
2                            SpringerGabler          18
3                                Peter Lang          16
4          Chartered Insurance Institute             15
5                               Mills&Boon          14
6                               : Springer          13
7                                Wiley-VCH          12
8                                   Dafina           7
9                           Luyi Publishing           6
10            Cambridge University Press              6
11                           ISTE Editions            6
12                       ISTE Editions Ltd            6
13                                  Kimani            5
14                                DeGruyter           5
15    The Chartered Insurance Institute              5
16                             Hatje Cantz            5
17       North-Holland Biomedical Press               5
18                     Better Link Press              4
19            Austin Macauley Publishers              4
20                                  Wiley            4
21                          BAR Publishing           4
22                            Mc Graw-Hill           4
23                    Amberley Publishing            4
24                           Morning Star            4
25                          ACC Art Books            3
26   Grosvenor House Publishing Limited              3
27                          AcademicaPress           3
28                Oxford University Press            3
29              Cambridge Text Education             3
30                                Collins            3
31                     SilvanaEditoriale             3
32                                  Brill            3
33                               BBC Active          3
34         Wiley-VCH Verlag GmbH&Co KGaA             3
35                                  P Lang           3
36                     The Crowood Press             3
37                                Berlitz            3
38                                  Gower            3
39                       Packt Publishing            3


Test Passed - Expected and Actual Frames are the same
```

Test Result Interpretation

The result indicates that the expected data frame and actual data frame are same. Provided expected result is accurate, this indicates that the data analytic code is error free.

Test Case 5

```
class BooksPeryearByLanguage(Analysis):
    def performAnalysis(self):
        import pandas as pd
        from pandas.testing import assert_frame_equal

        # Load CSV
        df = pd.read_csv('C:/Users/user/Desktop/APDP PYTHON CODE/Books.CSV.csv')

        # Rename columns
        df.columns = ['book', 'author', 'publication date', 'language', 'book publisher', 'ISBN', 'BNB id']

        # Group by year and language
        (exp_results) = df.groupby(['publication date', 'language']).size().unstack().fillna(0)

        # Filter last 5 years for comparison
        actual_result = [[2019, 2020, 2021, 2022, 2023]]
        print("Actual Books Published Per Year by Language:")
        print(actual_result)

        # Expected results
        exp_results = pd.DataFrame({
            'Chinese': [0.0, 14.0, 43.0, 23.0, 13.0],
            'English': [13.0, 9.0, 11.0, 11.0, 30.0],
            'French': [0.0, 0.0, 0.0, 0.0, 20.0],
            'German': [0.0, 0.0, 45.0, 27.0, 14.0],
            'Italian': [8.0, 27.0, 17.0, 4.0, 0.0],
            'Japanese': [3.0, 8.0, 12.0, 0.0, 0.0]
        }, year=[2019, 2020, 2021, 2022, 2023])

        print("\nExpected Results:")
        print(exp_results)

        # Assertion
        assert_frame_equal(actual_result, exp_results)
        print("\nTest Passed - Expected and Actual Frames are the same.\n")
```

```
Enter choice [1|2|3|4|5|6|7]: 6
Books Published Per Year by Language
language          Chinese  English  French  German  Italian  Japanese
publication date
2019                  0.0     13.0     0.0     0.0      8.0       3.0
2020                 14.0      9.0     0.0     0.0     27.0       8.0
2021                 43.0     11.0     0.0    45.0     17.0      12.0
2022                 23.0     11.0     0.0    27.0      4.0       0.0
2023                 13.0     30.0    20.0    14.0      0.0       0.0
```

Test Result Interpretation

The result indicates that the expected data frame and actual data frame are same. Provided expected result is accurate, this indicates that the data analytic code is error free.

Test Case 6

```python
class PublicationTrendsOverTime(Analysis):
    def performAnalysis(self):
        # Import necessary libraries
        import pandas as pd #IGNORE TYPE
        import matplotlib.pyplot as plt
        from pandas.testing import assert_frame_equal

        # Load the dataset
        df = pd.read_csv('C:/Users/user/Desktop/APDP PYTHON CODE/Books.CSV.csv')

        # Rename columns to readable format
        df.columns = ['book', 'author', 'publication date', 'language', 'book publisher', 'ISBN', 'BNB id']

        # Group by publication year and count number of books per year
        actual_result = df['publication date'].value_counts().sort_index().reset_index()
        actual_result.columns = ['publication date', 'book count']

        print("Actual Results")
        print(actual_result) # print summary

        # Manually prepare expected data for testing (example years and counts)
        exp_results = pd.DataFrame({
            'publication date': [2011, 2012, 2016, 2014, 2015, 2016, 2017, 2018, 2019, 2020, 2021, 2022, 2023],
            'book count': [8 ,1 ,6 ,77, 32 ,7, 28 ,47, 24 ,5, 18 ,65 ,77]
        })

        print("\n")
        print("Expected Results")
        print(exp_results)

        # Assertion to test actual vs expected
        assert_frame_equal(actual_result, exp_results)

        print("\n")
        print("Test Passed - Expected and Actual Frames are the same")
        print("\n")
```

```
DataFrame.iloc[:, 0] (column name="publication date") values are different (7.69231 %)
[index]: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]
[left]:  [2011, 2012, 2013, 2014, 2015, 2016, 2017, 2018, 2019, 2020, 2021, 2022, 2023]
[right]: [2011, 2012, 2016, 2014, 2015, 2016, 2017, 2018, 2019, 2020, 2021, 2022, 2023]
```

Test Result Interpretation

The result indicates that the expected data frame and actual data frame are different. Provided expected result is accurate, this indicates an error in the data analytic code.

Test Case 7

```
class Top5Authors(Analysis):
    def performAnalysis(self):
        import pandas as pd # ignore type
        from pandas.testing import assert_frame_equal

        # Load CSV file
        df = pd.read_csv('C:/Users/user/Desktop/APDP PYTHON CODE/Books.CSV.csv')

        # Rename columns
        df.columns = ['book', 'author', 'publication date', 'language', 'book publisher', 'ISBN', 'BNB id']

        # Calculate top 5 authors by book count
        actual_result = df['author'].value_counts().head(5).reset_index()
        actual_result.columns = ['author', 'book count']

        print("Actual Results")
        print(actual_result)#print summary

        # Expected results (example, update values as needed)
        exp_results = pd.DataFrame({
            'author': ['Zuri Day', 'Valerie Jackson', 'Jenni Fagan', 'Jeffrey Haynes', 'Matt Jones'],
            'book count': [6,227 ,12 ,128 ,10 ]
        })
        print("\n")
        print("Expected Results")
        print(exp_results)

        # Assertion
        assert_frame_equal(actual_result, exp_results)
        print("\n")
        print("Test Passed - Expected and Actual Frames are the same")
        print("\n")
```

```
DataFrame.iloc[:, 1] (column name="book count") values are different (60.0 %)
[index]: [0, 1, 2, 3, 4]
[left]:  [26, 22, 12, 12, 10]
[right]: [6, 227, 12, 128, 10]
```

Test Result Interpretation

The result indicates that the expected data frame and actual data frame are different. Provided expected result is accurate, this indicates an error in the data analytic code.

Test Case 8

```
class LanguageDistribution(Analysis):
    def performAnalysis(self):
        import pandas as pd
        from pandas.testing import assert_frame_equal

        # Load CSV
        df = pd.read_csv('C:/Users/user/Desktop/APDP PYTHON CODE/Books.CSV.csv')

        # Rename columns
        df.columns = ['book', 'author', 'publication date', 'language', 'book publisher', 'ISBN', 'BNB id']

        # Count by language
        actual_result = df['language'].value_counts().reset_index()
        actual_result.columns = ['language', 'book count']

        print("Actual Results")
        print(actual_result)

        # Expected results (example)
        exp_results = pd.DataFrame({
            'language': ['English',  'German' ,'French', 'Chinese', 'Italian' , 'Japanese'],
            'book count': [13, 150, 19, 93, 56, 123]
        })
        print("\n")
        print("Expected Results")
        print(exp_results)

        # Assertion
        assert_frame_equal(actual_result, exp_results)
        print("\n")
        print("Test Passed - Expected and Actual Frames are the same")
        print("\n")
```

```
DataFrame.iloc[:, 1] (column name="book count") values are different (50.0 %)
[index]: [0, 1, 2, 3, 4, 5]
[left]:  [153, 150, 129, 93, 56, 23]
[right]: [13, 150, 19, 93, 56, 123]
```

Test Result Interpretation

The result indicates that the expected data frame and actual data frame are different. Provided expected result is accurate, this indicates an error in the data analytic code.

Test Case 9

```
class BooksPerPublisher(Analysis):
    def performAnalysis(self):
        import pandas as pd
        from pandas.testing import assert_frame_equal

        # Load CSV
        df = pd.read_csv('C:/Users/user/Desktop/APDP PYTHON CODE/Books.CSV.csv')

        # Rename columns
        df.columns = ['book', 'author', 'publication date', 'language', 'book publisher', 'ISBN', 'BNB id']

        # Group by publisher
        actual_result = df['book publisher'].value_counts().head(40).reset_index()
        actual_result.columns = ['book publisher', 'book count']
        print("Actual Results")
        print(actual_result)

        # Expected data
        exp_results = pd.DataFrame({
            'book publisher': ['Routledge', 'Elsevier/Masson', 'SpringerGabler', 'Peter Lang',
        'Chartered Insurance Institute', 'Mills&Boon', ': Springer', 'Wiley-VCH',
        'Dafina', 'Luyi Publishing', 'Cambridge University Press', 'ISTE Editions',
        'ISTE Editions Ltd', 'Kimani', 'DeGruyter', 'The Chartered Insurance Institute',
        'Hatje Cantz', 'North-Holland Biomedical Press', 'Better Link Press',
        'Austin Macauley Publishers', 'Wiley', 'BAR Publishing', 'Mc Graw-Hill',
        'Amberley Publishing', 'Morning Star', 'ACC Art Books',
        'Grosvenor House Publishing Limited', 'AcademicaPress', 'Oxford University Press',
        'Cambridge Text Education', 'Collins', 'SilvanaEditoriale', 'Brill',
        'BBC Active', 'Wiley-VCH Verlag GmbH&Co KGaA', 'P Lang',
        'The Crowood Press', 'Berlitz', 'Gower', 'Packt Publishing' ],
            'book count': [ 61, 41, 18, 16, 15, 14, 13, 12, 7, 6, 6, 6, 6, 5, 5, 5, 5, 5, 4, 4, 4, 4,
        4, 4, 4, 3, 3, 3, 3, 3, 3, 3, 3, 3, 2, 3, 3, 3, 3, 3 ]
        })
        print("\n")
        print("Expected Results")
        print(exp_results)

        # Assertion
        assert_frame_equal(actual_result, exp_results)
        print("\n")
        print("Test Passed - Expected and Actual Frames are the same")
        print("\n")
```

```
DataFrame.iloc[:, 1] (column name="book count") values are different (2.5 %)
[index]: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39]
[left]:  [61, 41, 18, 16, 15, 14, 13, 12, 7, 6, 6, 6, 6, 5, 5, 5, 5, 5, 4, 4, 4, 4, 4, 4, 4, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3]
[right]: [61, 41, 18, 16, 15, 14, 13, 12, 7, 6, 6, 6, 6, 5, 5, 5, 5, 5, 4, 4, 4, 4, 4, 4, 4, 3, 3, 3, 3, 3, 3, 3, 3, 3, 2, 3, 3, 3, 3, 3]
```


Test Result Interpretation

The result indicates that the expected data frame and actual data frame are different. Provided expected result is accurate, this indicates an error in the data analytic code.

4.2. Differences between developer-produced and vendor-provided automatic testing tools for software systems

When considering the differences between developer produced and vendor provided automatic testing tools for software systems, several key factors come into play. Cost and licensing are significant considerations; while vendor provided tools, which include popular options like PyTest and Selenium, often require purchasing licenses or subscriptions, they also provide

robust support, regular updates, and access to advanced features. In contrast, developing custom tools in house doesn't incur direct financial costs but requires an investment of time and resources for development and maintenance, potentially offsetting savings on licensing fees with labor costs. Support and maintenance are another critical area of distinction. Vendor provided tools come with dedicated support, comprehensive documentation, and community forums, making it easier to resolve issues and stay up to date with the latest features. These tools are typically well maintained by their developers, who ensure compatibility with evolving software ecosystems. Conversely, in house tools rely entirely on the internal team's ability to support and maintain them, which can be challenging, especially if key developers leave the organization or if the team lacks the expertise to keep the tool current.

Customization and flexibility are advantages of developer produced tools, as they can be tailored precisely to meet the specific requirements of a project, ensuring seamless integration with existing systems. However, vendor provided tools may not always fit perfectly with unique project needs, despite offering a range of customization options. They are designed for broader compatibility, making them suitable for diverse environments and facilitating integration with multiple technologies, which can be particularly beneficial in complex, multi faceted projects.

Scalability and performance considerations also differ. Vendor tools are generally optimized for large scale testing scenarios and can handle high concurrency and distributed testing environments efficiently. In contrast, scaling in house tools to support extensive test cases or complex scenarios can be challenging and resource intensive, requiring additional development effort. The usability and learning curve associated with these tools vary; vendor provided solutions often come with extensive documentation and training resources, making them easier to learn, whereas custom tools may lack formal training materials, potentially complicating onboarding for new team members.

Ultimately, the decision between using vendor provided or developer produced testing tools depends on a balance of factors, including budget constraints, the scale and complexity of the project, the need for customization, and available support resources. While vendor tools offer convenience and a comprehensive feature set, they may not always align with specific project requirements. On the other hand, developer produced tools provide complete control and customization but require significant investment in development and ongoing maintenance.

Often, a hybrid approach, leveraging both types of tools, can provide the best balance, utilizing the strengths of each to meet the project's unique needs.

Aspect	**Developer-Built Tools (unit test)**	**Vendor Tools (Selenium, Postman)**
Customization	Highly flexible, fully tailored	Often generalized but powerful
Ease of Use	Requires more setup effort	User-friendly GUIs and dashboards
Cost	Free and open-source	Some require licenses/subscriptions
Suitability for Project	Best for CLI and Python-based apps	More suited for web-based or API systems

4.2.1. Key Differences

- Cost and Licensing

Developer produced tools do not require the purchase of licenses, making them a cost effective option in terms of direct financial expenditure. The primary costs are associated with the time and effort required for development and maintenance, which can be significant depending on the complexity of the tools. In contrast, vendor provided tools often come with associated costs, including licensing fees, subscription charges, or per user pricing. While some vendor tools are open source and free, commercial options generally require an upfront investment or ongoing payments. The cost of vendor provided tools is justified by the additional benefits they offer, such as professional support, regular updates, and access to advanced features.

- Support and Maintenance

Support and maintenance for developer produced tools are entirely managed in house, placing the responsibility on the internal development team. This includes ensuring the tool remains functional, updating it for compatibility with new technologies, and fixing any issues that arise. The internal team must also create and maintain documentation, which can be a resource intensive process. On the other hand, vendor provided tools come with robust support structures, including customer service, community forums, and comprehensive documentation. These tools are regularly updated by the vendors to address bugs, add new features, and maintain compatibility with evolving technology landscapes. This external support can be a significant advantage, particularly for complex projects.

- Customization and Flexibility

Developer-produced tools offer unparalleled customization and flexibility, as they can be tailored to fit the exact needs of a project. The development team has complete control over the design, features, and functionalities, allowing for precise alignment with specific project requirements. This level of customization ensures that the tool integrates seamlessly with existing systems and workflows. Conversely, vendor provided tools, while offering a broad range of features, may not always perfectly match the unique requirements of every project. These tools are designed for general use and may have limitations in terms of customization. While many offer configurable options, the extent of customization is typically bounded by the tool's architecture and design.

- Integration and Compatibility

Integration and compatibility are straightforward for developer produced tools, as they can be designed specifically to work within the existing development environment and processes. However, ensuring that these tools remain compatible with new technologies or third party systems can require ongoing development effort. In contrast, vendor provided tools are typically designed for broad compatibility and are capable of integrating with a wide array of technologies, platforms, and development tools. They often include plugins and extensions to enhance their integration capabilities, making them versatile choices for projects involving diverse technologies. This broad compatibility is particularly useful in complex environments where multiple systems and tools need to work together seamlessly.

- Scalability and Performance

Scalability and performance are key considerations in the development of automatic testing tools. Developer produced tools can be optimized for specific use cases, but scaling them to handle large scale testing scenarios or high concurrency levels can be challenging. This requires additional development effort and expertise, particularly in optimizing the tool's performance. Vendor provided tools, on the other hand, are often designed with scalability in mind and are capable of handling extensive test cases and distributed testing environments. These tools are generally optimized for performance, making them suitable for projects with significant testing demands. Their ability to scale efficiently can be a critical factor in choosing a testing solution for large or growing projects.

- Speed of Implementation

Developer produced tools require significant time to develop, test, and refine, which can delay their implementation. This process includes coding the tool, ensuring its integration with existing systems, and conducting thorough testing to confirm its functionality. The timeline can be lengthy, depending on the complexity and scope of the tool. On the other hand, vendor provided tools offer a quicker path to implementation since they are pre built and ready for use. The primary time investment with these tools involves integrating them into existing workflows and training the team to use them effectively. This rapid implementation can be particularly beneficial when testing needs to commence quickly, providing immediate support to the development process.

4.3. Different methods of implementing automated testing

Benefits

Automated testing is very effective in areas like unit tests, regression tests and integration tests. In unit testing, we could easily cover each function with automated testing by clearly defining the expected output and comparing it with the actual output. Pytest is used to test the functionality of various computational functions in the data analysis system (there are several functions such as logon(), getamount() and getdiscount() that perform important tasks within the analysis system). We could use white box testing approach to define test cases for each unit. We can write a bulk of parameterized test cases for a given function covering every logic path. For example, in our data analysis system there is a function that calculates the discount of sales amount (above or equal to 10000, discount of 10% otherwise no discount). We could write an assertion that compares expected output with actual output.

```
import pytest
def  discount(price):
  if (price>=10000):
    return price * 10 /100
  else:
    return 0

def  test_discount():
  assert discount(15000) == 1500
```

Different methods of implementing automated testing offer various advantages that can significantly enhance the efficiency and reliability of software development processes. One effective method is unit testing, where individual units or components of a software are tested in isolation. Using a tool like PyTest, developers can write assertions to check if the function returns the expected results for different inputs, such as 10000, 5000, and 0. PyTest's simplicity allows for easy writing of these assertions, and it effectively detects discrepancies between expected and actual results, failing the test if an error is detected. This method does not require additional resources beyond the implementation of the PyTest module, making it cost effective. Moreover, parameterized test cases in PyTest enable bulk testing, saving developers and testers considerable time and effort. Another crucial method for automated testing is regression testing, which involves re running previous test cases after modifications are made to a software system. Changes in software often introduce new errors, potentially affecting the entire system

even if the modification was minor. Regression testing ensures that these changes do not introduce side effects. Manual regression testing in large systems with thousands of test cases can be extremely labor intensive and error prone. Automated regression testing, however, can efficiently handle this task by repeating the entire suite of tests and reporting any errors, thus freeing human testers from this repetitive work and ensuring thorough validation.

Integration testing is another method that benefits significantly from automation. Integration testing focuses on the interactions between different modules or functions in a software system. For instance, in a data analysis system, the menu that calls various data analytic functions based on user choices must be tested to ensure each function call executes correctly. Automated integration testing can efficiently verify these interactions, ensuring that the links between modules are functioning as intended. This reduces the risk of integration issues and improves the overall robustness of the system. End to end testing is a comprehensive testing method where the entire application flow is tested from start to finish. This method ensures that all the integrated pieces of the application work together as expected. Automated end to end testing tools, such as Selenium, simulate user interactions with the application, testing real world scenarios and workflows. This method is particularly useful for identifying issues that may not be apparent in isolated unit or integration tests, providing a holistic view of the application's performance and reliability.

Load and performance testing is essential for assessing how a system behaves under high traffic or heavy usage. Automated tools like JMeter can simulate thousands of users interacting with the system simultaneously, helping to identify bottlenecks and performance issues. This method ensures that the application can handle peak loads without degradation in performance, which is crucial for maintaining user satisfaction and operational efficiency. Implementing automated testing through various methods like unit testing, regression testing, integration testing, end to end testing, and load/performance testing provides numerous benefits. These methods enhance the accuracy, efficiency, and coverage of tests, significantly reducing the time and effort required by human testers. Automated testing ensures that software systems are robust, reliable, and perform well under various conditions, ultimately leading to higher quality software and a more efficient development process.

Drawbacks

Automated testing is a powerful tool, but it has its weaknesses as well. One of the major issues is that human programmers have to work in a framework and syntax of their choice made by other people, which may take time to get used to. Huge time and effort can be spent as developers and testers try to get familiar with the tool that they have never used before. The automation tool itself is developed by a third party, this means that the human tester or programmer does not have full control over the tool. This is rather unmanageable, especially when there are problems that need one to wait for the vendor to release a new version or support. Also, there can be other expenses, for example, if the automation software is purchased from a company.

The limitation of automated tests is that they do not possess creativity and intuition of human testers. Automated tests run based on a set script and may not detect issues that a human tester would notice. Although Pytest assertions can be used to compare the actual output with the expected output and inform the tester that they do not match as failed tests, they do not explain why they do not match. The tester is required to analyze the code with a fine tooth comb in order to identify any and all logic errors, which can be quite difficult especially where there are large functions with many lines of code and many conditions. For instance, let there be a function in the sales system that gives a 10% discount for quantities that are equal to or greater than 10,000. If the assertion of the test case passes and the results match the expected ones,

```
import pytest
def  discount(price):
  if (price>=10000):
    return price * 10 /100
  else:
    return 0

def  test_discount():
  assert discount(15000) == 1500
```

then it means that there is no visible problem. However, if the assertion fails, it simply tells the programmer that there is an error in the code but does not tell him or her what the error is.

```
import pytest
def  discount(price):
  if (price>=10000):
    return price * 5 /100
  else:
    return 0

def  test_discount():
  assert discount(15000) == 1500
```

Automated testing, despite its many advantages, comes with several challenges. When errors are detected, especially in large data frames compared using tools like Pandas, identifying the root cause can be cumbersome. This is because pinpointing discrepancies in vast datasets is often difficult and time consuming. Additionally, User Interface (UI) testing poses its own set of difficulties. Since UI testing involves subjective aspects like aesthetics and user experience, which vary from person to person, automating these evaluations is challenging. While automated tools can verify functionality, such as ensuring that a button click directs to the correct page, they cannot measure user satisfaction or personal preferences.

Another issue with automated testing is that as software evolves, automated test scripts can become outdated or incorrect, necessitating frequent updates and maintenance. This ongoing upkeep can negate some of the time saved by automation. Additionally, automated tests may produce false positives or negatives, leading to confusion and further investigations. The initial setup and configuration of automated testing frameworks can also be complex and require significant expertise, presenting a barrier for organizations without dedicated automation specialists. Balancing the benefits of speed and efficiency with these challenges is crucial for successfully integrating automated testing into a development workflow.

The integration of automated testing tools into the software development process often involves a steep learning curve. Adopting new testing tools and frameworks can require substantial training and adaptation for the development and testing teams. This learning curve can slow down initial implementation and impact productivity. Additionally, automated tests might not always align perfectly with every aspect of the development cycle, especially when dealing with non standard or highly specialized software features. This misalignment can lead to gaps in test coverage or inefficiencies in the testing process.

Automated testing tools may struggle with scenarios that require human judgment or contextual understanding. For example, testing for scenarios that involve complex business logic or nuanced user interactions may not be well suited for automated scripts. These limitations highlight the importance of complementing automated testing with manual testing approaches. By leveraging both automated and manual testing strategies, teams can ensure more comprehensive coverage and better address the varied and dynamic needs of modern software development. Balancing these approaches helps in mitigating the limitations of automated testing while maximizing its benefits.

4.4. Action Plan to Mitigate the Drawbacks of Automated Testing

1. **Enhance Error Diagnosis**

   To improve error diagnosis, invest in advanced logging and reporting tools that provide detailed insights into test failures. Implementing better visualization tools for test results can help pinpoint issues more quickly, especially in complex data scenarios. Regularly review and update test scripts to ensure they are aligned with the current state of the software, and establish a robust process for analysing test failures to facilitate quicker troubleshooting.

2. **Integrate Manual Testing**

   Complement automated testing with manual testing approaches to address aspects that automation may not effectively cover, such as subjective UI evaluations and complex business logic. Manual testing can provide insights into user experience and identify issues that automated tests might miss. Combining automated and manual testing ensures a more comprehensive evaluation of the software, addressing both functional and experiential aspects.

3. **Maintain and Update Test Scripts**

   Develop a maintenance schedule for automated test scripts to ensure they remain accurate and relevant as the software evolves. Implement a version control system for test scripts to manage changes and updates efficiently. Assign dedicated resources or automation

engineers to oversee the maintenance of test scripts and ensure timely updates in response to software modifications.

4. **Simplify Initial Setup and Configuration**

   Invest in training programs for your team to streamline the initial setup and configuration of automated testing frameworks. Consider using user-friendly tools with extensive documentation and support to reduce the complexity of the setup process. For organizations without dedicated automation engineers, seek external expertise or consult with vendors to assist with the initial configuration and integration.

5. **Address Learning Curve Challenges**

   Provide ongoing training and support for team members to help them become proficient with new testing tools. Encourage knowledge sharing within the team and create a repository of best practices and guidelines for using automated testing tools effectively. Additionally, involve team members in selecting and evaluating testing tools to ensure they meet the team's needs and capabilities.

6. **Implement Flexible Testing Strategies**

   Adopt a flexible testing strategy that incorporates both automated and manual testing based on the context and requirements of different test cases. Prioritize test automation for repetitive and high-volume testing tasks while reserving manual testing for areas that require human judgment or contextual understanding. This balanced approach ensures that testing is both efficient and comprehensive.

7. **Monitor and Evaluate Tool Effectiveness**

   Continuously monitor the performance and effectiveness of automated testing tools to identify areas for improvement. Collect feedback from users and stakeholders to assess the impact of automation on the testing process. Use this feedback to make informed decisions about tool adjustments or replacements and to refine testing strategies over time.

Advantages and Limitations of Each Method

In the development and testing of the Dream Book Shop data processing application, several automated testing methods were used to ensure functionality, reliability, and maintainability. Each testing method brought its own advantages and limitations, making it important to apply them in the right context to achieve the best results.

Unit testing was a fundamental part of the testing strategy. Its primary advantage is that it allows developers to test individual functions or methods in isolation. This makes unit tests fast to execute and easy to manage. For example, the logic used to identify the most frequent author in the dataset was tested through unit testing. By isolating the function and feeding it known inputs, it was possible to verify that the output was always accurate, even when the data changed. However, unit testing also has its limitations. It does not test how different parts of the application work together, nor does it validate user inputs or the final CLI output. This means that while a method may work perfectly in isolation, it could still fail when interacting with other components.

To overcome this, integration testing was applied. Integration tests check whether various parts of the application, such as data loading, analysis functions, and user input handling, work together correctly. For instance, when testing how the application reads the CSV file and performs a full analysis like “Books Published Per Year by Language,” integration testing ensured that the data was loaded properly, grouped accurately, and displayed in a clear format. The limitation of integration testing is that it tends to be slower and, in case of failure, can be harder to pinpoint where the exact issue lies because multiple components are involved.

Additionally, the Pytest framework was used to structure and run the test cases. Pytest offers cleaner output, detailed error tracebacks, and supports advanced features like fixtures and parameterization. This improved the overall readability and debugging process during development. However, using Pytest effectively requires additional learning, especially for custom configurations or complex testing scenarios.

combining unit tests for individual functions, integration tests for full workflows, and Pytest for organization and automation provided a balanced and effective approach for the Dream

Book Shop application. Each method played a role in ensuring the system is accurate, user-friendly, and robust under different conditions.

Overall Evaluation and Future Improvements

The overall effectiveness of the current automated testing approach for the Dream Book Shop data analysis application has been positive. The testing framework successfully validated all six key features, ensuring that the core functionalities such as identifying the most frequent author, tracking book publications by language and year, and other analytical tasks performed accurately. Moreover, the automated tests were also effective in checking the application's response to a variety of user inputs, including both valid and invalid entries. This helped ensure that the program behaves predictably and does not crash during unexpected interactions, improving the overall user experience.

One of the most valuable outcomes of the current testing approach was the reduction of bugs during development. Since the tests were regularly executed during the coding process, it became easier to catch errors early. This made the iterative development process more reliable and allowed developers to refactor parts of the codebase with greater confidence, knowing that existing functionality would remain intact.

Despite the strengths of the current approach, there are several areas where the testing process can be improved. One such improvement would be the introduction of **Continuous** Integration (CI) using platforms like GitHub Actions. By automatically running tests after every code commit, developers can immediately identify if new changes break any part of the application, reducing the chances of faulty code reaching production.

Another improvement would be to incorporate mocking techniques for input/output (I/O) operations. This would isolate tests from actual data files, allowing developers to simulate various input scenarios without relying on the original dataset. Mocking would make tests more predictable and independent, especially when dealing with inconsistent or evolving datasets.

the implementation of output snapshot testing would be beneficial, particularly for a CLI-based system like the Dream Book Shop application. This testing method captures the output of the command-line interface and compares it to a previously saved correct version. If any change in the layout, structure, or data appears unexpectedly, the test will fail, prompting a review.

This is crucial for maintaining a consistent and user-friendly interface, especially when multiple contributors are working on the project.

Evaluation of the Overall Effectiveness of the Testing Approach

The overall testing strategy adopted for the Dream Book Shop data analytics application proved to be comprehensive, systematic, and largely effective. It involved a combination of both manual and automated testing methods, ensuring that the application was tested at multiple levels—from individual functions to the system as a whole. This multi-layered approach ensured that the software was thoroughly evaluated not just for correct functionality but also for performance, usability, and reliability under various scenarios.

One of the most important aspects of the testing phase was the use of automated testing with the Pytest framework, which is a widely used Python-based tool for writing and running test cases. Pytest was primarily utilized for unit testing, where individual functions such as identifying the most frequent author, counting books by language, and analyzing yearly publication trends were tested in isolation. Unit tests using Pytest are particularly efficient because they allow developers to verify that each small piece of the system works as intended before integrating it into the larger application.

Pytest operates on a system of assertions, where the actual output of a function is compared against the expected output. If the outputs match, the test is marked as passed and displayed in green. If they do not match, Pytest flags the test as failed in red, drawing immediate attention to the issue. This clear feedback mechanism allows developers to quickly identify when something has gone wrong at a functional level. This type of validation, where the system is tested based on inputs and expected outputs without examining its internal logic, is known as black-box testing.

Black-box testing has several advantages. It enables testers to focus on what the application is supposed to do, without needing to understand or modify the underlying code. This makes it especially useful for verifying functional requirements and ensuring that the software behaves correctly from a user's point of view. In the case of the Dream Book Shop system, this was particularly beneficial because many functions rely on parsing and analyzing data from an external CSV file, and black-box testing allowed developers to confirm correct output for various datasets.

However, despite its effectiveness, black-box testing through Pytest also has some limitations. While it can detect that a problem exists, it does not provide information about where or why the error occurred in the code. If a test fails, developers are left with little insight into which line or logic caused the failure. This is where white-box testing becomes crucial. White-box testing involves examining the internal workings of the program, including loops, conditions, and data structures. It allows the tester to trace the flow of execution and inspect variables at each stage.

In situations where deeper analysis is needed, manual white-box testing and developer-created diagnostic tools can be far more effective. For example, when a function analyzing book data failed to return expected values, manual debugging was necessary to trace the exact point in the function where the logic broke down. Developer tools that support step-by-step execution, breakpoints, and variable tracing helped identify these issues more precisely. By inspecting how the program moves through each block of code, developers were able to correct bugs that Pytest alone could not fully diagnose.

Another key component of the testing approach was integration testing, which validated how different components of the application worked together. For example, the way data was loaded from the CSV file, processed through various analysis modules, and then presented to the user had to be tested as a complete workflow. Integration tests ensured that these processes functioned correctly when combined, which is something neither unit nor black-box testing could cover on their own.

The team also carried out system testing to evaluate the performance and behavior of the application as a whole. This included testing for user input errors, invalid CSV structures, and performance under large datasets. These real-world conditions ensured that the Dream Book Shop system was not only accurate but also robust and stable during extended use.

Gant Chart

no	Task Name	May							June												July					
		20	21	23	24	25	26	30	1	5	6	7	15	16	20	21	23	24	26	27	31	1	5	6	15	16	20
1	Activity 01																										
2	Activity 02																										
3	Activity 03																										
4	Activity 04																										

References

David Loshin, 2021. *techtarget.* [Online] Available at: https://www.techtarget.com/searchdatamanagement/definition/data-structure [Accessed 25 07 2025].

geeksforgeeks, 2025. *geeksforgeeks.* [Online] Available at: https://www.geeksforgeeks.org/creational-design-pattern/ [Accessed 22 07 2025].

geeksforgeeks, 2025. *geeksforgeeks.* [Online] Available at: https://www.geeksforgeeks.org/software-testing-basics/ [Accessed 22 08 2025].

Herrity, K., 2025. *indeed.* [Online] Available at: https://www.indeed.com/career-advice/career-development/what-is-object-oriented-programming [Accessed 26 07 2025].

Khan, M. H., 2023. *medium.* [Online] Available at: https://medium.com/@humzakhalid94/understanding-object-oriented-relationships-inheritance-association-composition-and-aggregation-4d298494ac1c#:~:text=At%20its%20core%2C%20OOP%20relies,%2C%20maintainable%2C%20and%20scalable%20software. [Accessed 24 07 2025].

kumari, S., 2025. *naukri.* [Online] Available at: https://www.naukri.com/code360/library/characteristics-of-object-oriented-programming [Accessed 25 07 2025].

Maksimau, U., 2025. *wearecommunity.* [Online] Available at: https://wearecommunity.io/communities/epam-poland/articles/1190#:~:text=your%20systems%20correctly.-,SOLID%20is%20an%20acronym%20for%20five%20main%20principles%20of%20Object,principle%20and%20dependency%20inversion%20principle. [Accessed 25 07 2025].

pluralsight, 2022. *pluralsight.* [Online] Available at: https://www.pluralsight.com/blog/software-development/10-steps-to-clean-

code
[Accessed 25 07 2025].
pytest, 2025. *pytest.* [Online] Available at: https://docs.pytest.org/en/8.2.x/ [Accessed 22 08 2025].
saumyasaxena, 2025. *geeksforgeeks.* [Online] Available at: https://www.geeksforgeeks.org/automation-testing-software-testing/ [Accessed 22 08 2025].