**Implementation Comparison Python and Java: Expense Tracker Application**

Nitish Dhinaharan, Nicolas Ferradas Bertoli, Alexa R. Fisher, Arun Bhaskar Gaddle

Department of Information Technology, University of the Cumberlands

MSCS-632-B01 Advanced Programming Languages

Dr. Dax Bradley

April 6, 2025

**Implementation Comparison Python and Java: Expense Tracker Application**

Python and Java are a few of the most popular programming languages in recent years. The members selected the languages due to specific expertise. The design of the Expense Tracker applications highlighted the key differences between the programming languages. Feature differences underscored in this report include data structures, memory management, and error handling. Each language's characteristics affected the application's design, performance, and readability.

**Java**

**Data Structures**

Data structures are ways to organize, manage, and store data efficiently for access and modification. In Java, Data structures are supported both through built-in classes (like Lists and Maps, etc.) and user-defined structures (like trees, stacks, and queues) implemented via classes (Weiss, 2014).The data structures we used for the purpose of this application are AVLTree and a custom DoublyLinkedList. AVLTree is a type of self-balancing binary search tree (BST). This balancing characteristic helps maintain O(log n) time complexity for insertion, deletion, and search operations in our application (Goodrich et al., 2014). Doubly Linked List is a linear data structure that allows for forward and backward directions (Weiss, 2014). This is used for the filter operation in our operation, which lets us retrieve all the data with matching conditions in a single traversal. Hence, these two data structures are optimal for our application.

**Memory Management**

Java employs automatic memory management through automatic garbage collection (GC) which helps to abstract away memory management from the developers. This makes it easier for the developers since there’s no need for manual memory management. The automatic GC helps eliminate dangling pointers and prevent memory leaks. This helps improve application performance (Gosling et al., 2014).

**Error Handling**

Error handling in Java is managed through a robust exception handling mechanism that allows developers to anticipate, catch, and respond to runtime errors gracefully without crashing the program (Eckel, 2006). Errors in Java are categorized into two broad types – errors and exceptions. Java uses try-catch blocks to handle these errors and exceptions. The try block is where we define a piece code to monitor for errors or exceptions, and the catch block handles the exception and executes the necessary operation when running into errors (Oracle, n.d.)

**Python**

**Data Structures**

Data structures within Python are built-in (Phillips et al., 2016). The data structures emphasize Python’s ease of use and writability (Phillips et al., 2016). A few highlighted data structures include dictionaries and lists. Lists allow developers to store items dynamically (Phillips et al., 2016). Dictionaries focus on key-value pairing for storage and retrieval (Phillips et al., 2016). The dictionary containers are ideal for mapping objects (Phillips et al., 2016). Developers often utilize dictionaries for searching (Phillips et al., 2016).

**Memory Management**

Python takes a hassle-free approach to memory management (Yegulalp, 2022). Developers do not have to allocate, manage, or dispose of memory manually (Yegulalp, 2022). Within Python, memory management is automatic and completed during runtime (Yegulalp, 2022). It utilizes reference counting, commonly noted as refcount, in which each object tracks the number of other objects referencing it (Yegulalp, 2022). Once the object is no longer referenced and the count reaches zero, Python automatically deallocates its memory (Yegulalp, 2022).

**Error Handling**

Error handling in Python is crucial. Due to Python’s dynamic typing, runtime interpretation can potentially yield unexpected inputs (Lott & Phillips, 2021). Developers can mitigate these inputs by handling exceptions (Lott & Phillips, 2021). A common way to handle the exceptions is a “try…except” clause (Lott & Phillips, 2021). The clause tries the reference code; if invalid input occurs, an exception halts the processing (Lott & Phillips, 2021). Frequently, developers include a print statement describing the error.

**Comparison**

The comparison of Python and Java reveals distinct design characteristics. Python features dynamic typing, which allows for high flexibility. The types will be resolved in runtime. Whereas, Java features static typing, which is less flexible and the types will be resolved in compile time. Data structures are built-in, simplifying development (Phillips et al., 2016). Examples of data structures within Python include lists and dictionaries, which allow for effective performance and readability. Dictionaries allow rapid searching based on key pairs, and lists focus on sequences. The application used dictionaries and lists to create methods for filtering objects. The “deleteExpense()” method removed a transaction from the transaction list within the Python application. Meanwhile, the Java application used the AVLTree data structure for insert and search operations. Specifically, the application uses the “insert()” method of the AVLTree to log expenses and income. The application uses a custom “rangeSearch()” method of the AVLTree to filter expense and income by a given condition. The Java application uses a custom “filterByCategory()” method, of a DoublyLinkedList structure, to filter expenses and income by category. This design is optimal since it helps maintain O(log n) time complexity for insertion, and range search operations in our application (Goodrich et al., 2014).

Memory management in Python is automatic (Yegulalp, 2022). It highlights the ease of use and emphasizes simplistic design at the potential cost of performance overhead (Yegulalp, 2022). The memory used by the deleted transaction’s dictionary was eventually freed automatically by Python’s garbage collector (Yegulalp, 2022). Similar to the Python application, our Java application also utilizes an automatic memory management mechanism using the Java JVM’s garbage collector (Gosling et al., 2014).

In Python, developers utilize “try…except” code blocks to implement error handling (Lott & Phillips, 2021). It improves debugging and maintainability, contributing to the code’s readability (Lott & Phillips, 2021). Both of our Python and Java applications utilize error handling mechanisms to handle errors with processing the input date while logging, deleting and searching for transactions with a given input date. The difference is in the methods used to accomplish this. The Python application utilizes a “try… except” code block, whereas the Java application uses a “try… catch” code block to handle errors.

The figures below visualize the key differences between each language regarding the main features of data structures, and error handling.

**Figure 1:** *Comparison of Data Structures*

| **Data Structures** | |
| --- | --- |
| Java | Python |
| public class Account {  private String name;  private double totalBalance;  private List<Expense> expenses;  private List<Income> credits;  // storage for optimized search  private AVLTree<Expense> amonutExpenseSearchTree;  private AVLTree<Income> amountIncomeSearchTree;  private AVLTree<Expense> dateExpenseSearchTree;  private AVLTree<Income> dateIncomeSearchTree;  private CategoryDoublyLinkedList expenseLinkedList;  private CategoryDoublyLinkedList incomeLinkedList;  } | class Account:  def \_\_init\_\_(self) – > None:  self.transactions: List[Transaction] = []  self. transaction\_by\_type = defaultdict(list) # Caches transactions by type  self.lock = threading.Lock() # Ensures thread-safe operations |

*Note:*Java code uses List, AVLTree and a custom DoublyLinkedList. The Python code uses Lists, notated with brackets “[ ]” to hold transaction dictionaries.

**Figure 2:** *Comparison of Error Handling*

| **Error Handling** | |
| --- | --- |
| Java | Python |
| private static Date convertToDateFormat(String dateString){  try {  SimpleDateFormat formatter = new SimpleDateFormat("dd-MM-yyyy");  Date date = formatter.parse(dateString);  return date;  } catch (Exception e){  System.out.println("Incorrect date format");  return null;  }  } | def \_parse\_date(self, date\_str: str) - > Optional[datetime.date]:  try:  return datetime.datetime.strptime(date, "%Y-%m-%d").date()  except ValueError:  print("Error: Incorrect date format. Use YYYY-MM-DD.")  return None |

*Note:*The Java code highlights an example of error handling utilizing the “try…catch” code block to handle errors in a date format conversion. The Python code highlights an example of error handling utilizing the “try…except” code block to a date format Value Error.

**Conclusion**

The implementation of the Expense Tracker application in both Python and Java demonstrated how language-specific features significantly influence software design, performance, and readability. Python’s dynamic typing, built-in data structures, and concise syntax facilitated faster development and greater code readability, especially for rapid prototyping and smaller-scale applications. In contrast, Java’s strong static typing, robust data structure libraries like AVL Trees and custom DoublyLinkedLists, and explicit memory and error handling mechanisms contributed to greater control, efficiency, and scalability, particularly for complex or performance-sensitive applications. While both languages provide automatic memory management and structured exception handling, the differences in syntax and verbosity shaped how these features were implemented. Overall, the comparison highlights that language choice impacts not only coding style but also the structural and functional aspects of the application.

**References**

Goodrich, M. T., Tamassia, R., & Goldwasser, M. H. (2014). *Data structures and algorithms in Java* (6th ed.). Wiley.

Weiss, M. A. (2014). *Data structures and algorithm analysis in Java* (3rd ed.). Pearson.

Eckel, B. (2006). *Thinking in Java* (4th ed.). Prentice Hall.

Oracle. (n.d.). *Exceptions (The Java™ Tutorials)*.<https://docs.oracle.com/javase/tutorial/essential/exceptions/>

Gosling, J., Joy, B., Steele, G., Bracha, G., & Buckley, A. (2014). *The Java language specification* (Java SE 8 ed.). Addison-Wesley.

Lott, S. F., & Phillips, D. (2021). *Python Object-Oriented Programming : Build Robust and Maintainable Object-oriented Python Applications and Libraries*. (pp. 125–162). Packt Publishing, Limited. https://research.ebsco.com/plink/91471866-7c84-3156-baf4-e96080663282

Phillips, D., Romano, F., Vo.T.H, P., Layton, R., Czygan, M., & Raschka, S. (2016). Python Data Structures. In *Python: Real-World* *Data Science*. (pp. 199–232). Packt Publishing, Limited. https://research.ebsco.com/plink/38bbd789-a4c7-32f3-921a-1544dac25d41

Yegulalp, S. (2022). Python garbage collection and the gc module. *InfoWorld.Com,* <https://www.proquest.com/trade-journals/python-garbage-collection-gc-module/docview/2714107855/se-2>