# TASK 1:

# 1.1

# 1.2

Algorithm: Linear Search

Start the linear search algorithm.

Receive the list of vehicle years and the target search year as input.

Traverse through the list of vehicle years, starting from the first element:

a. For each year in the list, compare it with the target search year.

b. If the current year matches the target search year, return the current index and exit the loop.

c. If the current year does not match the target search year, continue to the next year in the list.

If the loop completes without finding a match, return -1 to indicate that no vehicle was found for the target search year.

End the linear search algorithm.

This algorithm aligns with the searchForVehicleYear method in the code, which iterates through the list of vehicle years and returns the index of the first occurrence of the target search year or -1 if it's not found.  
  
--  
  
Algorithms play a fundamental role in computer science and programming. An algorithm is a step-by-step procedure or a set of rules for solving a specific problem or accomplishing a particular task. They provide a structured approach to problem-solving and enable us to efficiently solve complex problems by breaking them down into smaller, manageable steps.

Reasons why we need algorithms:

Efficiency: By using algorithms, we can address issues in the most effective and efficient way possible. We may reduce the amount of time and resources needed to complete computations or complete activities by carefully developing algorithms. This is important in many fields, including big data processing, optimization issues, and real-time systems.

Reproducibility: An algorithm's description of how to solve a problem is clear-cut and exact. Since they may be written in a programming language, others can execute them and get the same outcomes. Building on current solutions and collaborating effectively depend on this repeatability.

Scalability: As issues and datasets expand, algorithms enable us to manage the complexity that comes with it. We can effectively handle more inputs without noticeably degrading performance by using scalable algorithms. This is especially important in fields like network optimization, machine learning, and data analysis.

Innovation is fuelled by algorithms, which make it possible to develop new products and technology. They serve as the basis for a number of developments in computer science, including as artificial intelligence, cryptography, image processing, and optimization methods. We may push the limits of what is computationally possible by comprehending and creating algorithms.  
  
In short, algorithms are crucial tools in programming and computer science. It enables us to address scaling problems, deliver repeatable solutions, address problems quickly, establish standardized practices, and drive innovation across a range of industries. Any programmer or computer scientist should be able to understand and use algorithms efficiently.  
  
https://www.simplilearn.com/tutorials/data-structure-tutorial/what-is-an-algorithm

**1.3**

**Problem Statement**: Clearly define the problem that needs to be solved. In this case, the problem was to find a specific value (target year) in a given list of values (vehicle years).  
  
**Plan**: Create a plan to solve the problem. This involves choosing an appropriate algorithmic approach. In this case, the plan was to use the linear search algorithm, which sequentially checks each element in the list until a match is found or the end of the list is reached.  
  
**Algorithm Development**: Develop the algorithm based on the chosen plan. This includes defining the steps to be executed to solve the problem. The linear search algorithm involves iterating through each element in the list and comparing it with the target value.  
  
**Code Implementation**: Translate the algorithm into code using a specific programming language. This step involves writing the necessary syntax and constructs to implement the algorithm. In this case, the provided code in Java implements the linear search algorithm.  
  
**Testing**: Execute the code with different test cases to ensure it produces the expected output and behaves correctly in various scenarios. This helps identify and fix any errors or issues in the code.  
  
**Debugging**: If any errors or unexpected behavior is encountered during testing, the code needs to be debugged. This involves identifying and resolving the issues causing the incorrect output or runtime errors. Debugging may include reviewing the code, analyzing the algorithm, and using debugging tools to track and fix the problems.  
  
By following these stages, the development process of the linear search algorithm involves a systematic approach to problem-solving, algorithm design, and code implementation. It also emphasizes the importance of testing and debugging to ensure the correctness and reliability of the implemented solution. **1.4**The illustration of the linear search algorithm was very helpful in the development process. It helped me understand the steps involved in the algorithm and how to implement them in code. The illustration also helped me identify potential areas where the algorithm could be improved or optimized.  
  
The relationship between the written code and the algorithm is that the code is the implementation of the algorithm. The algorithm provides the steps to be taken to solve the problem, while the code is the actual implementation of those steps. In other words, the algorithm is the blueprint or plan, while the code is the physical manifestation of that plan.  
  
In the case of the linear search algorithm, the written code was a direct implementation of the steps outlined in the algorithm. The code iterated through the list of elements and compared each element with the target value until a match was found or the end of the list was reached. The algorithm helped me understand the logic behind this implementation and identify potential areas for optimization, such as the use of a break statement to exit the loop once a match is found.  
  
Overall, the illustration of the linear search algorithm was crucial in helping me understand the problem-solving process and translating that understanding into a working implementation.

# TASK 2:

# 2.1

**Data Structure:** A more complex information structure would be needed to store many data types for each vehicle. For car names and years, distinct Array Lists were used in the before approach. Maintaining data integrity and ensuring consistent access to and change of the data may be difficult with the inclusion of additional attributes like VIN, assessment, and problems.

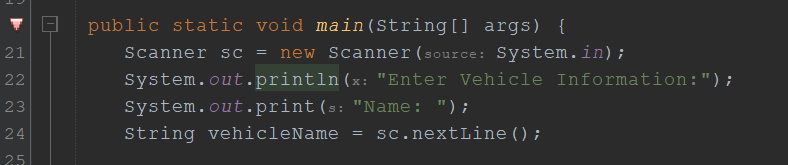
**Data relationships**: Presentation between data when additional auto attributes are added. For example, the VIN must be unique to each vehicle and verified to match specific requirements. The order of other data properties may change if the list of years is sorted ascending, which can lead to data errors or misalignment.

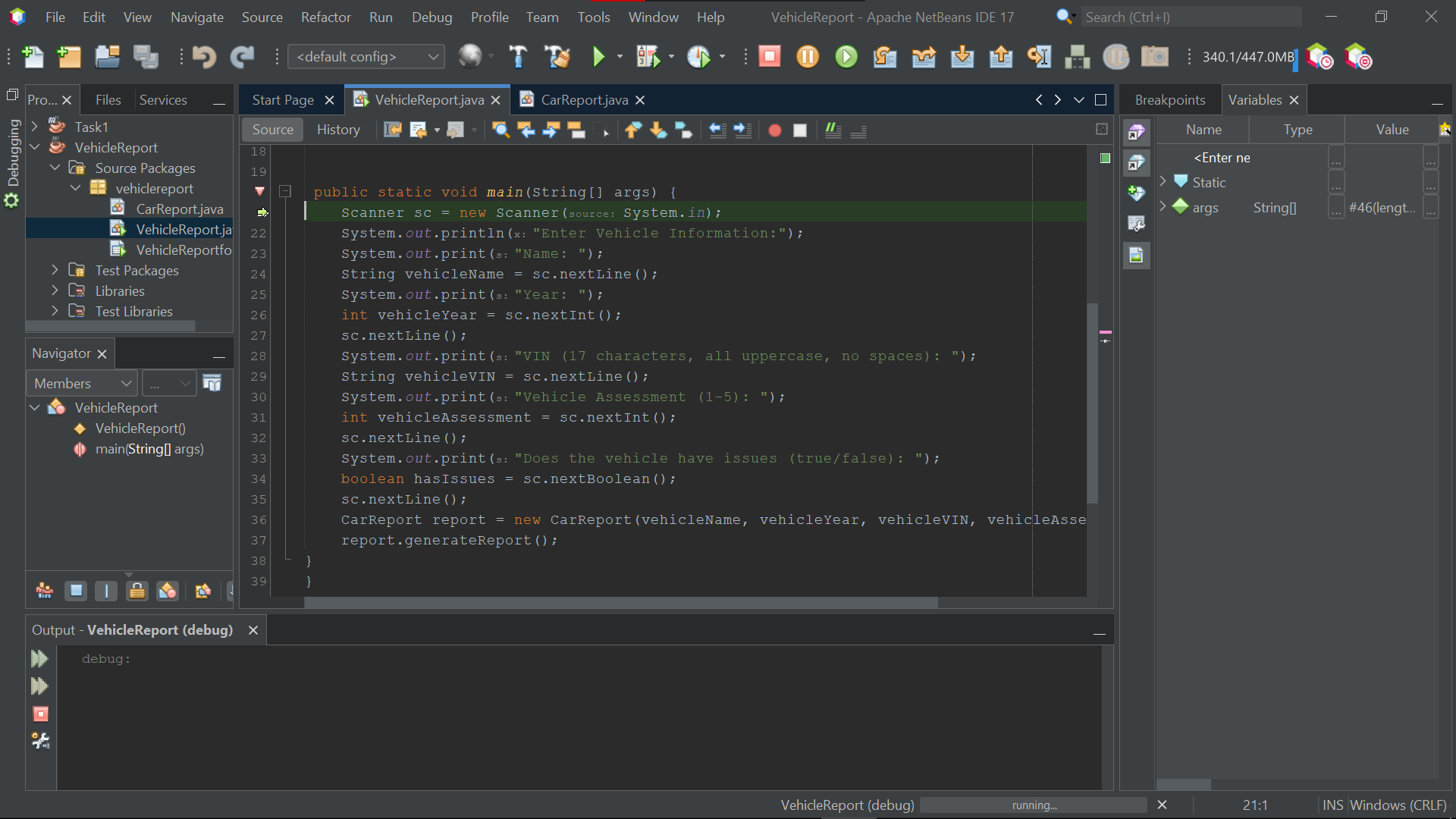
**Data validation:** As the number of characteristics increases, the validation procedure gets more difficult. Every feature must be verified in line with a set of guidelines and limitations. The VIN, for example, must have a specific length and format, and the assessment must be within an authorized rating range. To preserve the system's honesty, effective validation and handling of incorrect information become necessary.

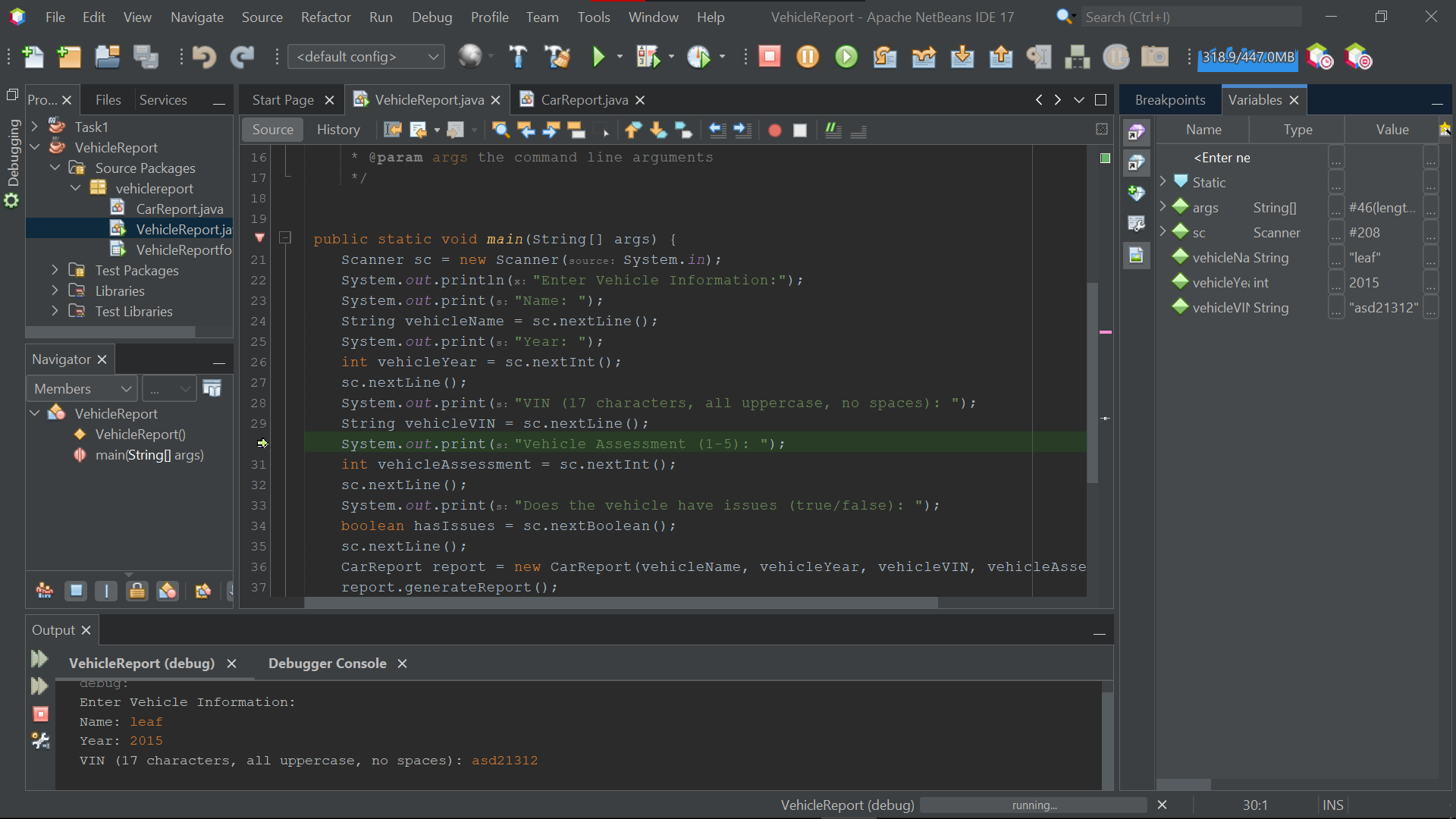
**2.2 / 2.3 / 2.4**

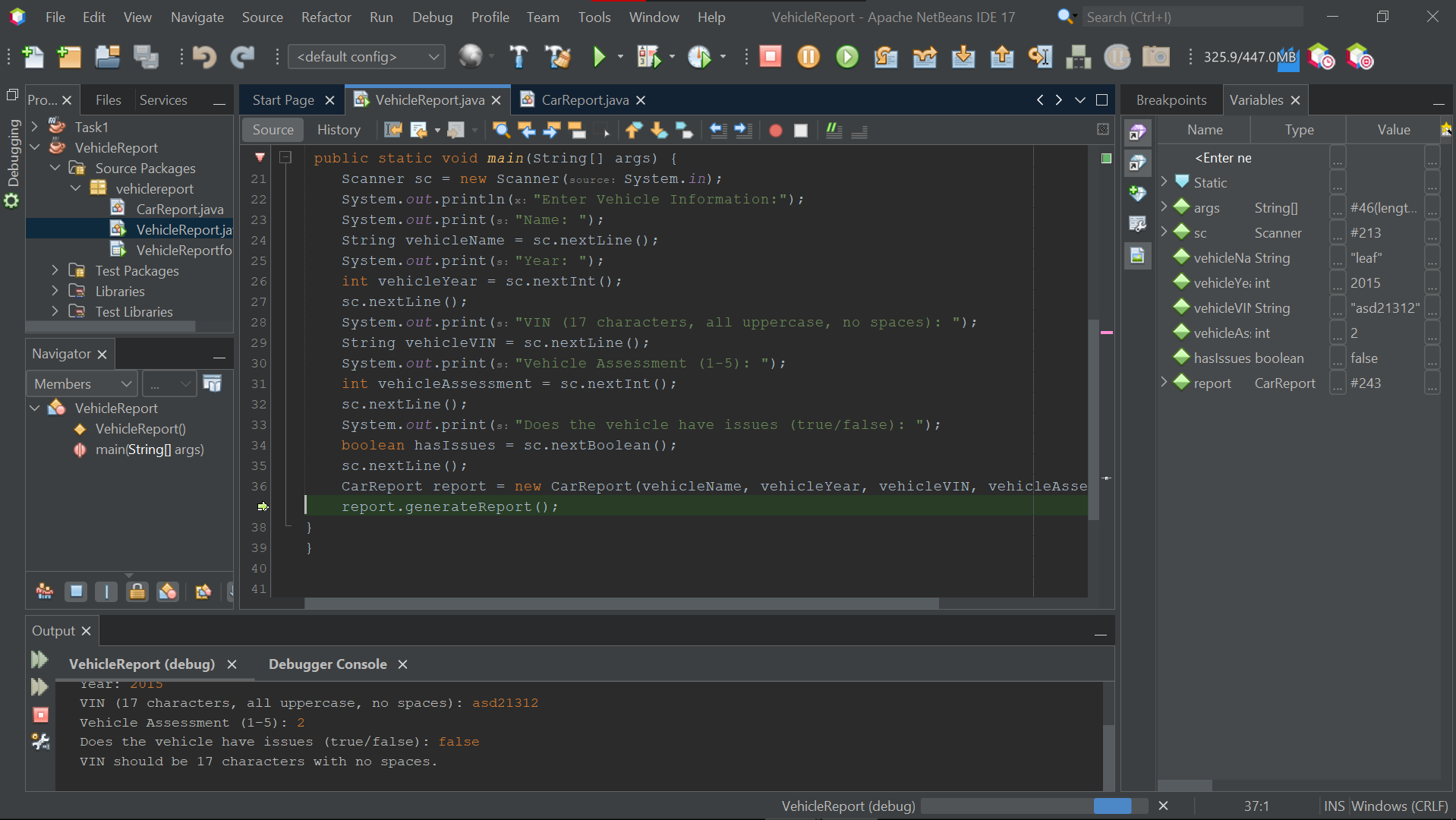
****

**2.5**break point

****

****  
I’ll do step over on all of them

The error will be here

When I get to report.generateReport I click Step into  
  
A screenshot of a computer program

Description automatically generated with medium confidenceThen I got the error message

# definition of the debugging process:

Debugging is the process of identifying and resolving errors, defects, or faults in software code. It involves locating and fixing issues that prevent the code from functioning as intended. Debugging is an essential part of the software development cycle and is typically performed after initial testing to address any problems discovered during testing. The process may involve analyzing error messages, inspecting the code, setting breakpoints, and stepping through the code to trace the flow and identify the root cause of the issue. Debugging can be time-consuming, and experienced programmers recognize it as a significant challenge in the development process. While debugging is important, it is also beneficial to focus on preventing bugs by adopting best coding practices, using testing methodologies, and employing techniques like code reviews and automated testing to catch issues early in the development lifecycle.

# TASK 3:

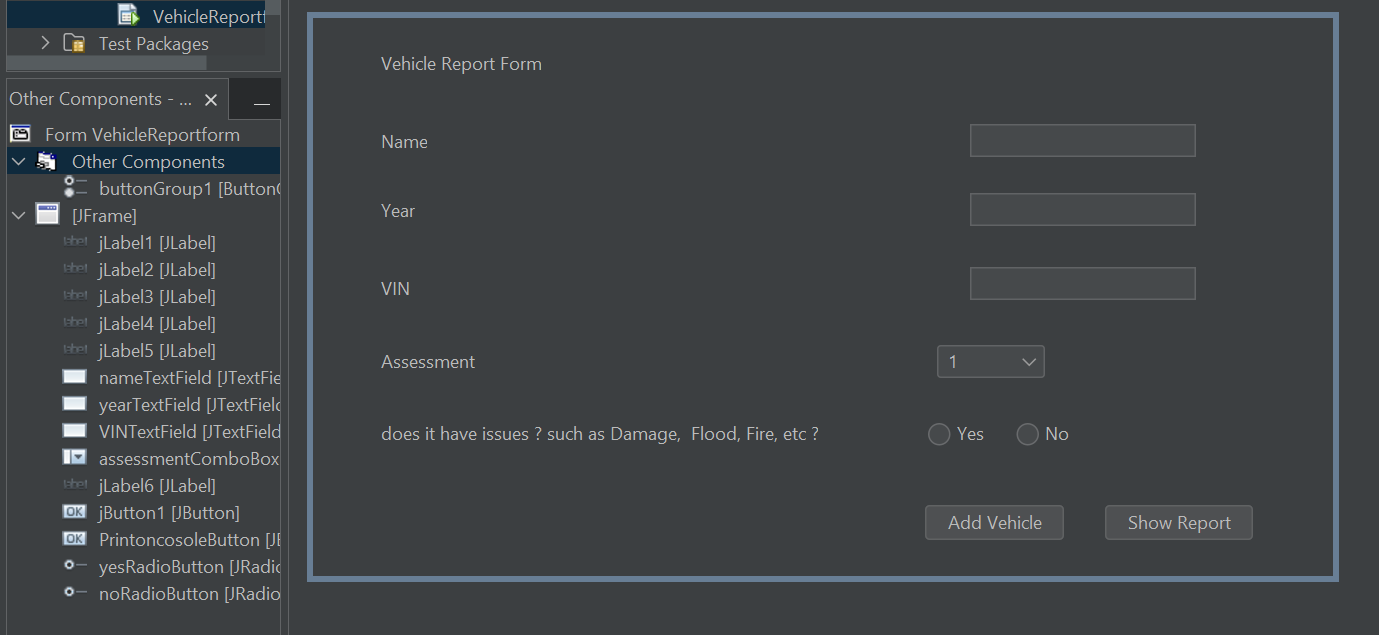
**You have faced a difficuilty with entering large number odetailsis for every vechile in Task 2, in this task, you have been asked to ehance user entry process by building a user-friendly GUI that will help in adding vechile details and generating reports.**

**The GUI should allow the user to enter all necessary details, and looks something similar to *Figure 7.***

**Consider that “*Add Vechile*” button will fetch different data from the GUI, validate it, and continue the implementation as in Task 2. “*show report*” button, will trigger the generateReport() method for each entered vehicle and show results on console. The implementation will not be different from in 2.4.**

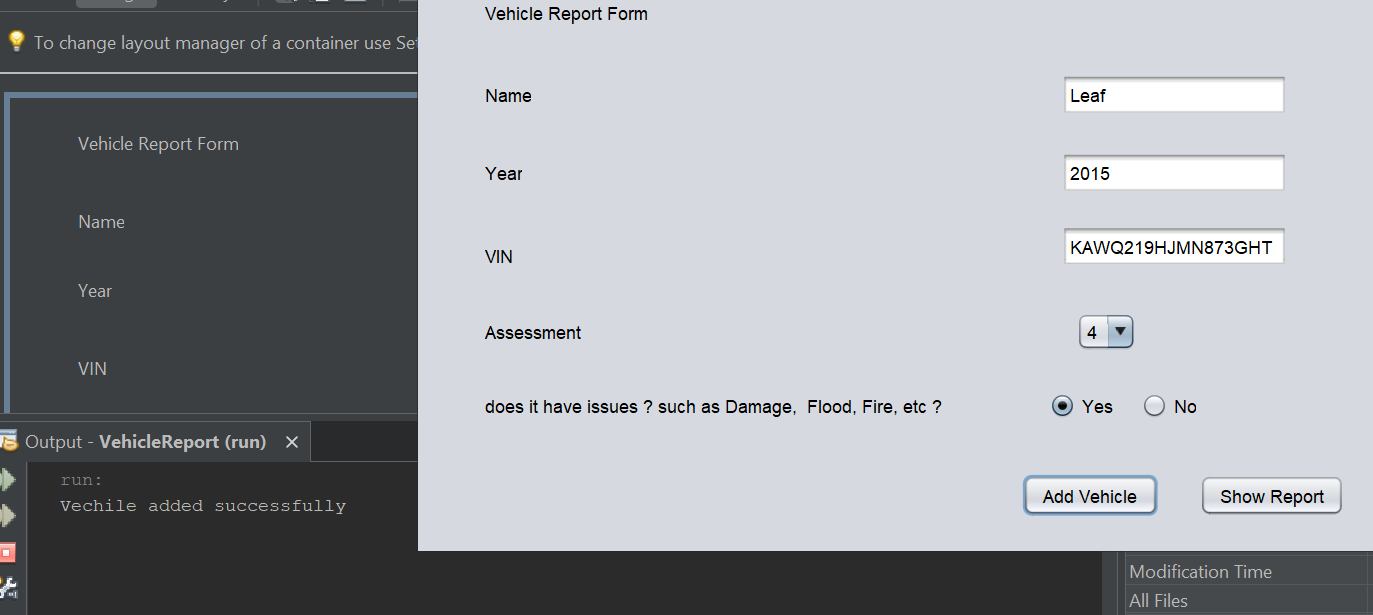
# Source

****



A screenshot of a computer

Description automatically generated



A screenshot of a computer

Description automatically generated

# Task 4

4.1

# Task 1: Procedural Programming

**Main characteristics:**

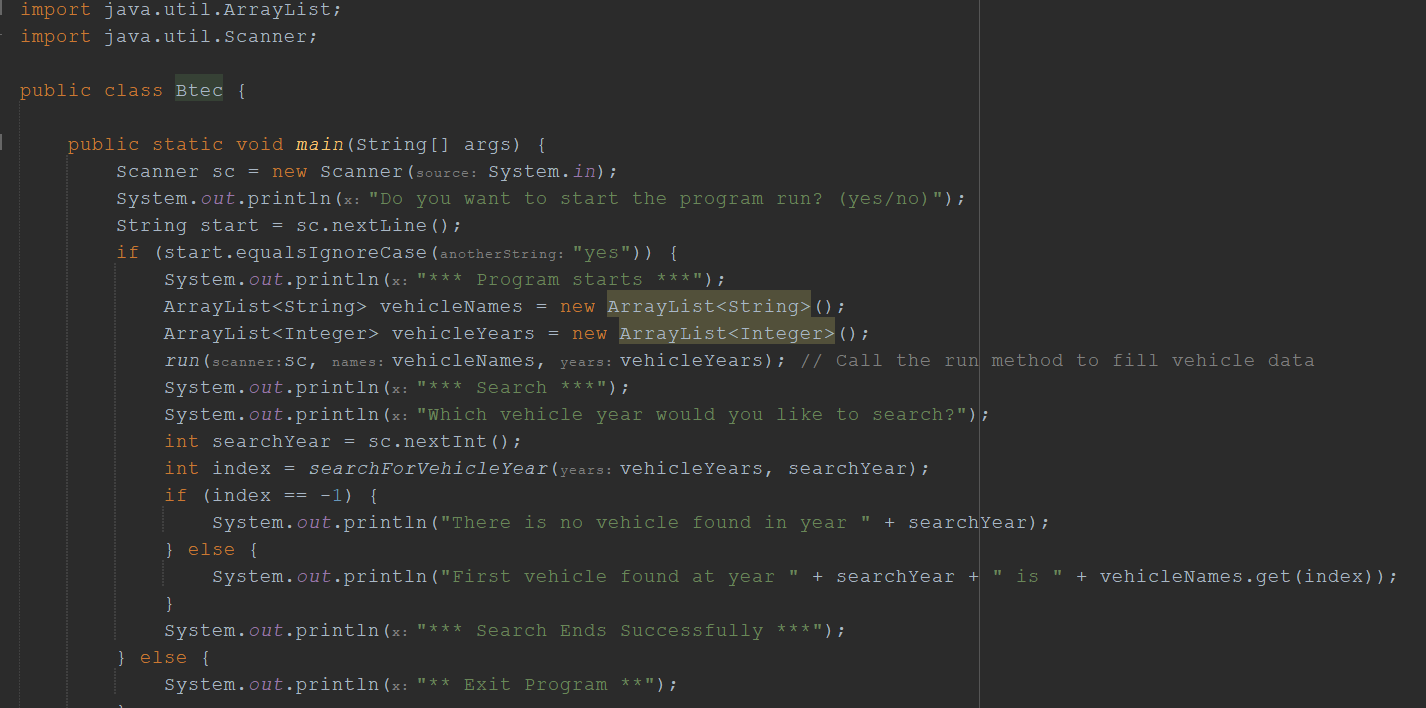
**Procedures:** Procedural programming focuses on procedures or functions that operate on data. The code is organized into a series of procedures that perform specific tasks.

**Top-down design:** The code follows a top-down design approach, where the problem is broken down into smaller sub-tasks, and each sub-task is implemented as a procedure.

**Sequential execution:** The code is executed sequentially, one instruction after another, and the control flows through the procedures.

**Relationships:**

**Data sharing:** In procedural programming, data is typically shared between procedures through parameters and return values. For example, in Task 1, the run method accepts Scanner, ArrayList<String>, and ArrayList<Integer> as parameters and modifies the lists accordingly.

**Modularity:** Procedures provide modularization, allowing different parts of the code to be developed and tested independently. In Task 1, the run method encapsulates the logic for adding vehicles and stopping vehicle addition.  


# Task 2: Object-Oriented Programming (OOP)

**Main characteristics:**

**Objects and classes:** OOP organizes code into objects that encapsulate data and behavior. Objects are instances of classes, which define the attributes (data) and methods (behavior) that objects can have.

**Encapsulation:** OOP promotes encapsulation, where data and methods are bundled together within objects. This helps in data hiding and abstraction**.  
  
Inheritance:** OOP supports inheritance, allowing classes to inherit attributes and methods from other classes. This promotes code reuse and allows for hierarchical organization of classes. **Polymorphism:** OOP enables polymorphism, where objects of different classes can be treated uniformly through inheritance and method overriding.  
 **Relationships:  
  
Class-object relationship:** Classes define the blueprint for creating objects. In Task 2, the JavaPP class represents the main program, and objects of Scanner and ArrayList are created within the class.  
 **Object interaction**: Objects interact with each other by invoking methods and accessing properties. For example, in Task 2, the run method interacts with the Scanner and ArrayList objects to add vehicle data.

**Inheritance relationship:** Inheritance allows for the reuse of code and the creation of specialized classes. Although not explicitly demonstrated in Task 2, inheritance can be used to create subclasses with additional attributes and behaviors based on a superclass**.**A screen shot of a computer screen

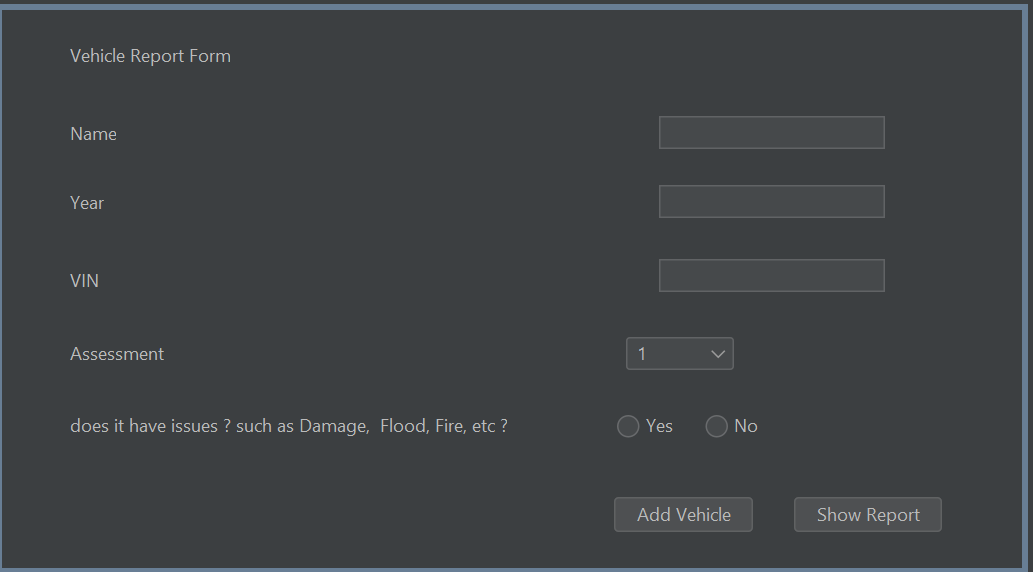
Description automatically generated with low confidence

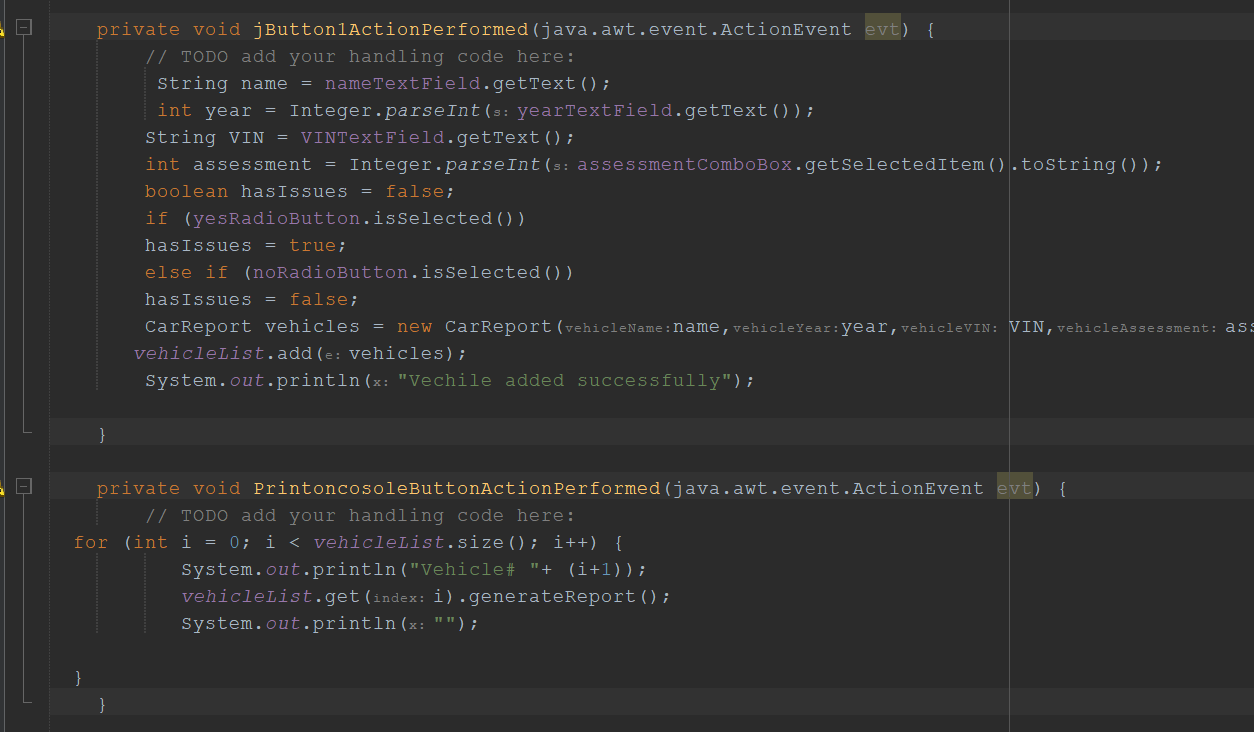
Task 3: Event-Driven Programming **Main characteristics:**

**Event handling:** Event-driven programming focuses on responding to user actions or system events. The code is structured around handling specific events or triggers.

**Event loop:** The program typically runs an event loop that waits for events to occur and dispatches them to appropriate event handlers.

**Asynchronous execution:** Event-driven programs often run asynchronously, allowing multiple events to be handled concurrently. **Relationships:  
Event handling**: In Task 3, the code responds to events such as user input or button clicks. Event handlers or listeners are used to handle these events and perform the desired actions.

**Event-driven architecture:** The program structure revolves around events and their associated event handlers. The flow of control is determined by the occurrence of events, rather than following a linear execution path.



|  |  |  |  |
| --- | --- | --- | --- |
| **Paradigm** | **Pros** | **Cons** | **Domain of Applicability** |
| Procedural programming | - Easy to understand and learn - Good for tasks that require explicit control flow - Can be used to create efficient programs | - Can be error-prone - Can be difficult to maintain - Not as expressive as other paradigms | - General-purpose - Good for system programming, game development, and other tasks that require explicit control flow |
| Object-oriented programming | - Good for tasks that involve objects and their interactions - Easy to create reusable code - Can be used to create modular and well-organized programs | - Can be complex and difficult to learn - Can be difficult to debug - Not as well-suited for tasks that do not involve objects | - Software development, game development, and other tasks that involve objects and their interactions |
| Event-driven programming | - Good for tasks that respond to events - Easy to create responsive and user-friendly interfaces - Can be used to create scalable and maintainable applications | - Can be complex and difficult to learn - Can be difficult to debug - Not as well-suited for tasks that do not involve events | - GUI development, web development, and other tasks that respond to events |

4.2

**4.3**  
the structure and characteristics of each paradigm--procedural programming (PP), object-oriented programming (OOP), and event-driven programming (EDP):

1. Procedural Programming (PP):

**Structure:** PP organizes code into procedures or functions that operate on data, following a  
top-down approach.  
  
**Characteristics:**  
- Focus on algorithms and step-by-step execution.  
- Modularity through smaller procedures.  
- Use of global data and sequential execution.  
  
PP is suitable for algorithmic tasks but can become challenging to manage as programs grow larger.

2. Object-Oriented Programming (OOP):

**Structure:** OOP organizes code into objects with data and behavior, interacting through message passing and inheritance.  
  
**Characteristics:**  
- Emphasis on objects and classes.  
- Encapsulation of data and behavior.  
- Inheritance for code reuse and polymorphism for flexibility.

OOP promotes code reusability and modular design but can be complex to understand and manage class hierarchies.

3. Event-Driven Programming (EDP):  
**Structure:** EDP revolves around events and event handlers that respond to user actions or  
system notifications.  
  
**Characteristics:**  
- Event-driven systems driven by events and event handlers.  
- Continuous event loops and asynchronous programming.  
- Commonly used in GUI applications for user interaction.

EDP excels in event-driven systems and GUI applications but can be challenging to coordinate event handlers and their dependencies.

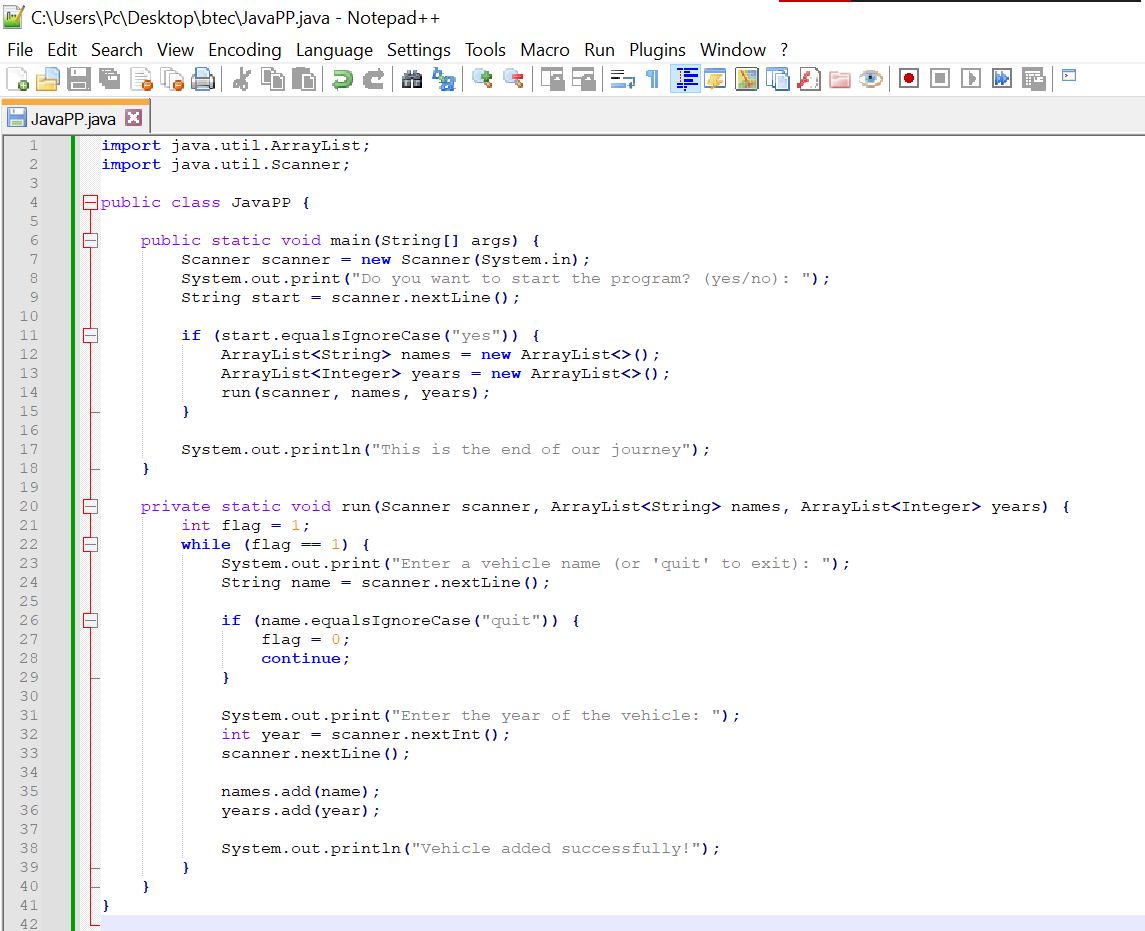
**4.4**  
NetBeans IDE offers a range of features that aid developers in effectively managing the development process. These include:

**1. Code Editing and Navigation:** NetBeans IDE provides a powerful editor that supports syntax highlighting, code completion, refactoring, and formatting for various programming languages. It also allows developers to easily navigate through the code structure, find usages, jump to declarations, and view documentation.  
  
**2. Project Management:** NetBeans IDE facilitates project creation and management by providing templates and wizards for different project types. These tools simplify the setup of new projects. The IDE also offers a project explorer view, allowing developers to navigate project files in a hierarchical structure. This feature assists in locating and organizing project resources.

**3. Version Control Integration:** NetBeans IDE integrates with popular version control systems, such as Git, Subversion, Mercurial, and CVS. It allows developers to perform common version control operations, such as commit, update, merge, diff, revert, and branch. It also shows the version history and status of files and folders in the project view.

**4. Debugger and Profiler:** NetBeans IDE offers a comprehensive debugger and profiler that helps developers find and fix errors and performance issues in their code. The debugger supports breakpoints, watches, expressions, variables, call stack, threads, and remote debugging. The profiler supports CPU profiling, memory profiling, thread profiling, and SQL queries profiling.

**5. Testing Support:** NetBeans IDE offers comprehensive testing support with features like integration with JUnit. Developers can write and execute unit tests within the IDE, ensuring code correctness. The IDE provides a test results window and integrates with popular testing frameworks, simplifying test execution and result analysis.

**4.5**  
  
A picture containing text, electronics, screenshot, software

Description automatically generated

A screenshot of a computer

Description automatically generated with low confidence  
  
The differences between using NetBeans IDE and a simple text editor like Notepad are significant. Here are the key distinctions:  
  
**1. Feature Set:**  
**(IDE):** IDEs are feature-rich software tools designed for software development. They offer a wide range of features such as code editing, debugging tools, project management, version control integration, and more. IDEs provide a complete development environment tailored to specific programming languages or frameworks.  
  
**(Text Editor):** Text editors focus on basic text editing functions and lack advanced features specific to software development, such as debugging tools or project management capabilities.  
  
  
**2. Code Productivity and Efficiency:  
(IDE):** IDEs enhance developer productivity with features like code completion, refactoring tools, and debugging capabilities, streamlining the coding process and reducing errors.  
  
**(Text Editor):** Text editors lack advanced productivity features, requiring more manual coding and potentially resulting in increased time and errors.

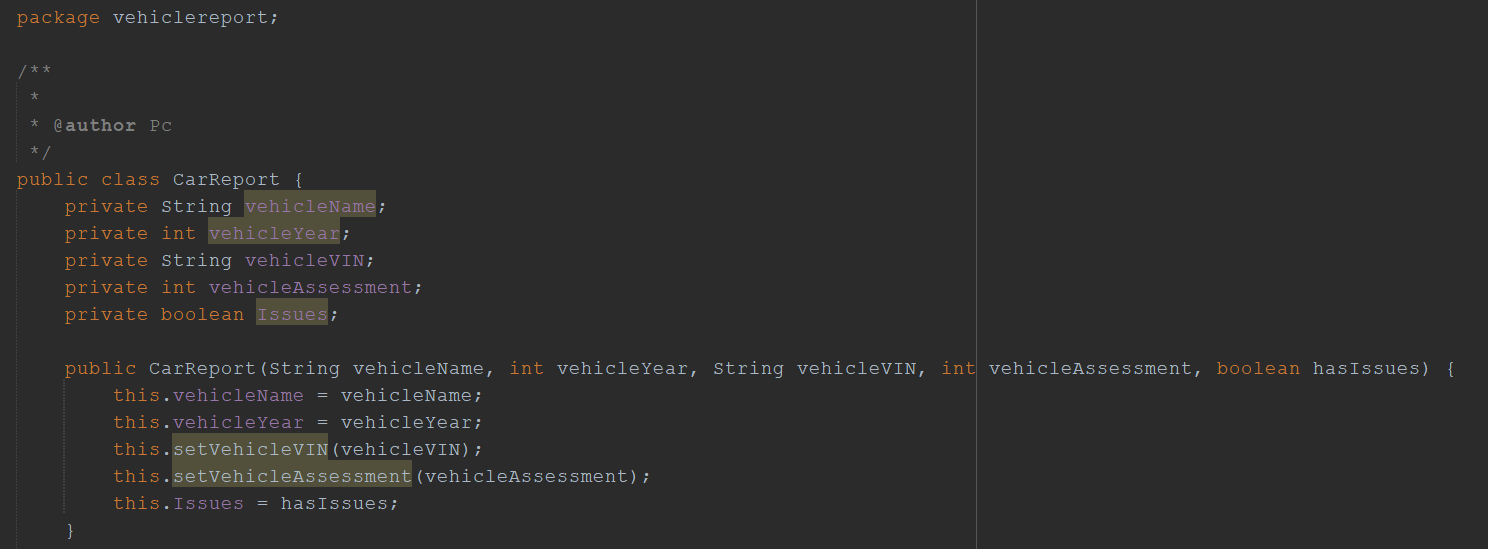
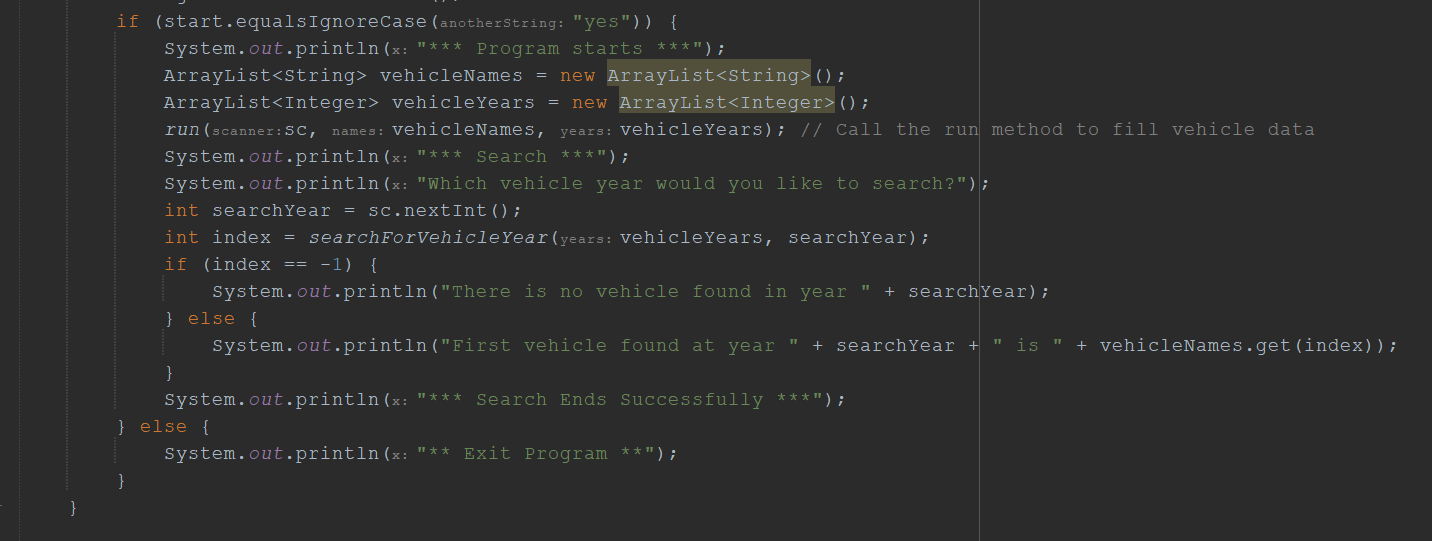
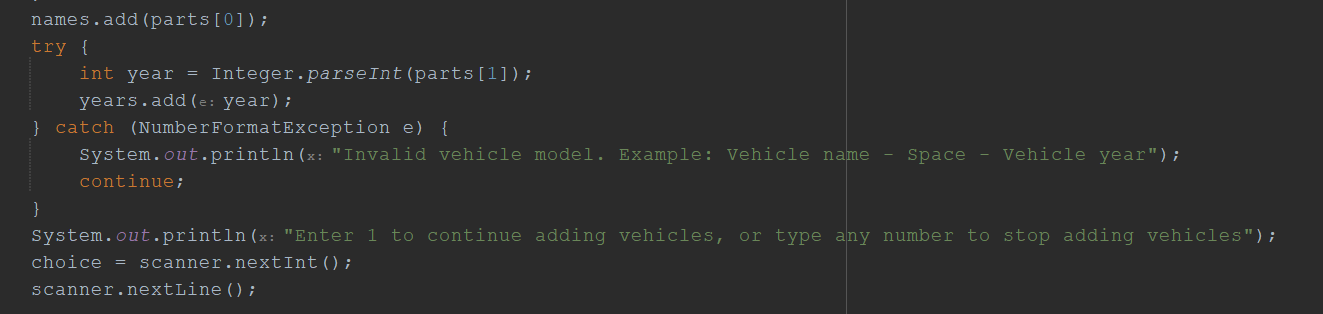
**3. Project Management and Collaboration:**  
 **(IDE):** IDEs provide comprehensive project management features, allowing developers to organize code, manage dependencies, and collaborate efficiently. They often integrate with version control systems for seamless team development.  
  
**(Text Editor):** Text editors lack built-in project management capabilities, requiring manual code organization and external tools for collaboration.

**4. Language and Framework Support:**  
**(IDE):** IDEs offer extensive support for specific programming languages or frameworks, providing language-specific features, tools, and integrations to enhance development efficiency.  
  
**(Text Editor):** Text editors have broader language support but lack specialized features for specific languages or frameworks.  
  
4.6  
The debugging process plays a critical role in developing secure and reliable applications by assisting in data validation and error handling for different types of input. Here's how debugging contributes to the security and stability of applications:

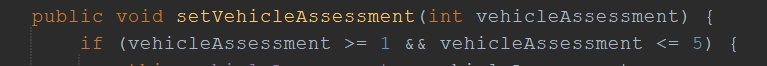
1. Data Validation:  
  
**1. Identifying Input Errors:** Debugging helps detect unexpected or incorrect behavior, indicating potential issues with data validation. It enables developers to locate and address input errors or invalid data that may compromise system security and integrity.  
  
**2. Resolving Validation Vulnerabilities:** Debugging identifies gaps in data validation, such as inadequate input sanitization or improper handling of user input. By addressing these vulnerabilities, developers can ensure proper validation and sanitization, guarding against security threats like SQL injection, cross-site scripting (XSS), or command injection attacks.

# 2. Error Handling:

**1. Identifying Exceptional Scenarios:** Debugging reveals errors, exceptions, or unexpected behaviors caused by incorrect input or faulty logic. This helps developers understand potential errors that may occur during runtime.  
  
**2. Enhancing Error Handling:** Debugging provides insights into error messages, stack traces, and program states during errors. Developers can use this information to improve error handling, such as providing informative messages, logging errors for troubleshooting, and implementing effective exception handling strategies.  
 **3.Graceful Error Handling:** Debugging identifies potential failure points, enabling developers to implement robust error handling routines. This includes gracefully managing errors, preventing application crashes, and safeguarding sensitive information from being exposed through error messages.

3. Testing and Verification:  
  
1. Debugging facilitates testing and verification by allowing developers to step through code, examine variables, and validate application behavior with different inputs. This helps identify potential vulnerabilities, logic flaws, or data validation issues that could lead to security breaches or application failures.  
  
2. Actively debugging and testing the application ensures the effectiveness of data validation mechanisms, the resilience of error handling routines, and the expected behavior of the application across various scenarios.  
  
**4.7**   
  
A coding standard, also referred to as a coding style guide or coding conventions, is a collection of guidelines and recommended practices that establish the rules and conventions for writing code consistently and in a readable manner. The main objective of a coding standard is to ensure that code is written in a manner that is easily comprehensible, maintainable, and conducive to collaboration, regardless of the number of developers involved. Below are some important elements of coding standards:  
  
**1. Naming Conventions:** - Use CamelCase for class and interface names, starting with an uppercase letter.  
-Use camelCase for method and variable names, starting with a lowercase letter.  
-Write constants in uppercase with underscores between words.  
-Package names should be in lowercase, following a reverse domain name format.  
**  
2. Indentation and Formatting:**  
- Maintain consistent indentation (4 spaces or tabs) for code blocks.  
- Place opening braces "{" on the same line as the statement and closing braces "}" on a new line.  
- Use a single space around binary operators, excluding unary operators.  
  
  
**3. Comments and Documentation:**  
- Add descriptive comments to explain code purpose and functionality.  
- Comment code blocks, classes, methods, and important statements.  
- Use Javadoc comments for generating API documentation  
  
  
**4. Error and Exception Handling:**  
- Handle exceptions using try-catch blocks or declare them in method signatures.  
- Avoid catching general exceptions unless necessary.  
- Provide meaningful error messages and consider logging exceptions for debugging.  
  
  
**5. Code Organization and Structure:**  
- Organize code into packages based on functionality or module.  
- Apply appropriate access modifiers (private, public) to classes, methods, and variables.  
- Keep methods and classes focused and adhere to the Single Responsibility Principle.  
- Utilize design patterns and follow established architectural principles.  
A picture containing text, screenshot, font

Description automatically generated

A picture containing screenshot, font, text, line

Description automatically generated  
  
  
  
  
  
  
  
  
  
**6. Code Reusability and Maintainability:**- Avoid code duplication by extracting reusable code into methods or separate classes.  
- Use meaningful names for variables and methods to enhance code readability.  
- Follow the "Don't Repeat Yourself" principle by eliminating redundant code.  
