Sudoku Game Project

JAVA PROGRAMMING LANGUAGE

Atish Kumar Sahu
SOFTWARE DEVELOPER | WEB DEVELOPER

09-Aug-2023

Content

ABOUT:

Greetings! I am Atish Kumar Sahu, a dynamic and dedicated individual hailing from the enchanting city of Berhampur, Odisha, India. With a relentless passion for technology and innovation, I have made significant strides in the world of software development. During my tenure as a Junior Application Developer at Pantheon Inc, from June 2022 to October 2022, I immersed myself in the realm of application development, honing my skills and contributing to th e creation of cutting edge solutions.

My pursuit of knowledge led me to acquire a B.Tech degree in Computer Science Engineering (CSE) from Parala Maharaja Engineering College, graduating in the year 2022. Throughout my academic journey, I demonstrated a keen aptitude for learning and consistently showcased an exemplary work ethic. As a professional, I pride myself on my adeptness in team management, fostering collaboration, and driving projects to successful completion. My unwavering focus and determination enable me to tackle challenges head on, delivering results that exceed expectations.

In terms of technical expertise, I possess proficiency in an array of programming languages, including C, Java, and MySQL, and my knowledge extends to the realm of web development. Additionally, I am well versed in utilizing tools such as MS Office and Google Suite to streamline operations and boost productivity.

In summary, I am an enthusiastic and adaptable individual, committed to delivering exceptional outcomes in the realm of software development. With a solid foundation in technology and a penchant for hard work, I eagerly embrace opportunities to contribute meaningfully to projects and organizations. Thank you for considering my profile, and I look forward to making a valuable impact wherever I embark on my professional journey.

Introduction:

Sudoku is a popular logic-based number placement puzzle game. The objective is to fill a 9x9 grid with digits so that each column, each row, and each of the nine 3x3 subgrids (also called boxes) that compose the grid contain all of the digits from 1 to 9. This project aims to implement a Sudoku game using Java programming, leveraging concepts like AWT and Swing for the graphical user interface.

Introduction To JAVA Programming:

Java is a general purpose high level programming language. It means using java we can develop verities of applications like desktop application web application enterprise application device application. Java is a technology because java has huge library support for simplifying the code complexity due to support or readymade method. The extension of java is (.java).

Java is a platform. A platform is an environment where we can execute our java program. java has its own JRE that’s why can say java is itself a platform. JAVA is compiled as well as an interpreted language. Java supports object oriented programming. Java is used to develop an internet base of application [applet, servlet, JSP]. Java is used to create dynamic web pages. Java provides facilities to program electronic consumable devices such as mobile, laptops, palmtops, using J2ME.

Java supports multithreading. Java Standard Edition(JSE) provide the basic core functionality of the java programming language. The core java concept is called JSE which is especially used to develop a standalone application or desktop application. Java Enterprise Edition(JEE) especially used to develop web and enterprise applications. On enterprise platforms, JEE is widely used for developing enterprise level applications. Java Micro Edition(JME) is used to develop mobile device applications or we can say that android development is the JME of java language. Java Development Kit JDK this provides an environment to develop and run a java application.

So we have to install JDK first. Java Runtime Environment(JRE) provides an environment only to run a java application. Once you ins tall the JDK automatically JRE will create. Java Virtual Machine(JVM) is an interpreter who is responsible to run a java program one statement at a time. JVM provides a java execution engine that executes the java source code.

Import Statement In JAVA

In Java, the import statement is used to access classes, interfaces, and other types that are defined in different packages. When you want to use a class or type from a package other than the current package, you need to import it into your code using the import statement. The import statement is placed at the beginning of a Java source file (before the class declaration) and allows you to specify the package and class you want to use in the code. Once the import statement is used, you can refer to the imported class directly by its simple name (without the package name).

The import statement in Java is used to make classes and interfaces from other packages available in your current source code file. It allows you to reference those classes and interfaces by their simple names (without fully qualified package names), making your code more concise and readable. The import statement essentially helps resolve naming conflicts and eases the process of using classes from external packages by creating a shortcut to their declarations.

Main() Function In Java

In Java, the main() function is a crucial entry point for executing a Java program. It serves as the starting point for the program's execution when it is run. The main() method must have a specific signature (public static void main(String[] args)) defined within a class. The public access modifier allows it to be accessible from outside the class, and static indicates that the method is associated with the class itself rather than an instance of the class. The void return type signifies that the method does not return any value. The String[] args parameter allows you to pass command-line arguments to the program. When the Java Virtual Machine (JVM) runs the program, it starts execution from the main() method's code block, making it an essential component for initializing the application, setting up variables, and controlling its overall flow.

If-Else Condition In JAVA

In Java, the if-else condition is a fundamental control structure that enables the execution of different code blocks based on the evaluation of a specified condition. The if statement allows you to execute a certain block of code if the provided condition is true. Alternatively, the else statement provides an alternative code block to be executed when the condition is false. This construct empowers programmers to create decision-making logic in their programs. Multiple else if clauses can be added to check additional conditions in sequence, allowing for more complex branching behavior. The if-else condition plays a pivotal role in guiding program flow, enabling different code paths to be taken based on the values of variables, user input, or other criteria, thus enhancing the program's flexibility and functionality.

For Loop In JAVA

The for loop in Java is a versatile iterative construct that simplifies repetitive tasks by allowing a specified block of code to be executed multiple times. It consists of three components within its parentheses: an initialization statement, a termination condition, and an increment or decrement expression. These components work in harmony to control the loop's execution. The loop begins by executing the initialization statement, followed by evaluating the termination condition.

If the condition is true, the code block enclosed within the loop is executed. After each iteration, the increment or decrement expression is evaluated, and the process repeats until the termination condition becomes false. The for loop is particularly effective for tasks that require a fixed number of iterations or when iterating through arrays, collections, and ranges of values. Its concise syntax and systematic approach make it an essential tool for managing repetitive operations in Java programs.

Class & Constructor In Java

In Java, a class serves as a blueprint for creating objects with specific attributes (variables) and behaviors (methods). It encapsulates data and functionality within a single entity, promoting code organization and reusability. A class defines the structure and behavior of objects, acting as a template that can be instantiated multiple times.

Constructors are special methods within a class responsible for initializing the object's state when it's created. They have the same name as the class and can take parameters to set initial values to the object's variables. Constructors play a crucial role in ensuring that objects are created in a consistent and valid state. By combining classes and constructors, Java enables the creation of structured, organized, and efficient code, facilitating object-oriented programming principles like encapsulation and abstraction.

Function In JAVA & Types Of Function

In Java, functions are known as methods, and they encapsulate a set of instructions to perform a specific task. Methods provide modularity and reusability to code by allowing the encapsulation of logic within a single unit. They can accept input parameters and return values, making them versatile tools for processing data and performing operations. Java supports several types of methods, including instance methods, which operate on specific instances of a class, and static methods, which belong to the class itself rather than instances.

Instance methods can access instance variables, while static methods can only access static variables. Java also has built-in methods provided by libraries, as well as user-defined methods created by developers. This categorization of methods enhances code organization, improves readability, and promotes the creation of efficient and maintainable Java programs.

Try & Catch In JAVA

In Java, the try-catch block is an essential mechanism for managing exceptions, which are unexpected or erroneous conditions that can disrupt the normal flow of a program. The `try` block encapsulates the code that might raise an exception. If an exception occurs within the `try` block, the program's control is transferred to the corresponding `catch` block. The `catch` block contains code to handle the exception gracefully, preventing the program from crashing.

By catching exceptions, developers can provide meaningful error messages, take corrective actions, or log the issues for debugging purposes. Multiple `catch` blocks can be used to handle different types of exceptions, allowing for precise error handling. Additionally, the `finally` block can be employed to contain code that should execute regardless of whether an exception occurred or not. The try-catch mechanism in Java enhances the robustness and reliability of programs by enabling systematic handling of unexpected situations.

Collection In Java

In Java, the Collection framework is a comprehensive set of classes and interfaces that provide a structured and efficient way to manage groups of objects. It enables developers to work with data structures like lists, sets, maps, queues, and more, simplifying the process of storing, manipulating, and retrieving elements. The Collection framework promotes code reuse and enhances efficiency by offering reusable implementations of common data structures.

It also supports features such as searching, sorting, and iterating through elements. The framework comprises interfaces like List, Set, and Map, along with their corresponding implementations such as ArrayList, HashSet, and HashMap. This organized hierarchy ensures consistency and compatibility when working with collections. Java's Collection framework is a vital component in creating flexible and scalable applications, as it provides powerful tools for handling complex data structures efficiently.

ArrayList In Java

In Java, an ArrayList is a dynamic and resizable implementation of the List interface within the Collection framework. It provides a versatile way to store and manipulate collections of objects, allowing elements to be added or removed without the need for manual resizing. ArrayList automatically adjust their size as elements are added or removed, making them suitable for scenarios where the number of elements is not known in advance. Elements within an ArrayList are stored in a contiguous memory block, enabling efficient access and traversal.

The ArrayList class offers various methods for adding, removing, accessing, and manipulating elements, such as add(), remove(), get(), and size(). By leveraging the ArrayList class, developers can easily manage and work with dynamic lists of objects, contributing to the flexibility and adaptability of Java applications.

ArrayList Methods In JAVA

The ArrayList class in Java offers a rich assortment of methods that facilitate seamless manipulation of dynamic lists of objects. The add() method is used to insert elements at specified positions, while remove() removes elements based on their indices or values. The get() method retrieves elements by index, enabling access to specific items. The size() method returns the current number of elements in the ArrayList.

Other methods like clear() remove all elements, isEmpty() checks if the list is devoid of elements, and contains() determines whether a certain element exists. Furthermore, indexOf() finds the index of a specified element, and set() replaces an element at a given index. Collectively, these methods empower developers to efficiently manage and manipulate the content of ArrayList, tailoring them to various programming needs and scenarios.

AWT Concept In JAVA

Abstract Window Toolkit (AWT) in Java is a foundational framework that enables the creation of graphical user interfaces (GUIs) for applications. AWT provides a comprehensive set of classes and components to design interactive and visually appealing user interfaces across different platforms. It consists of various GUI components like buttons, labels, text fields, checkboxes, and more, allowing developers to create windows, dialogs, and complex layouts.

AWT components encapsulate both appearance and behavior, facilitating the creation of rich user experiences. The event-handling mechanism in AWT manages user interactions such as button clicks, mouse movements, and keyboard inputs. By associating listeners with components, developers can define responses to these events, enabling dynamic and responsive interactions.

AWT utilizes the native windowing system of the underlying platform, ensuring consistent look and feel across different operating systems. While AWT is known for its simplicity and platform compatibility, it lacks some advanced features and components available in Swing, another Java GUI toolkit. Nevertheless, AWT remains a vital building block for graphical programming in Java, offering a straightforward way to craft user interfaces for a wide range of applications.

Swing Concept In Java

Swing, a prominent GUI toolkit in Java, extends the capabilities of AWT and presents a comprehensive set of components that enable the creation of sophisticated and platform-independent graphical user interfaces. Unlike AWT, Swing components are rendered entirely in Java, ensuring a consistent appearance across various operating systems.

Swing provides an extensive collection of GUI components, including buttons, text fields, tables, trees, and more, all of which can be customized extensively to match specific design requirements. Moreover, Swing's pluggable look-and-feel architecture empowers developers to modify the visual style of their applications without altering the code. Swing also offers features like double-buffering to enhance graphics rendering performance and supports advanced functionalities such as drag-and-drop, internationalization, and accessibility features.

The event handling in Swing utilizes the Observer pattern, where components generate events and listeners respond accordingly. The Swing toolkit promotes the development of feature-rich, interactive, and responsive graphical applications in Java, making it a popular choice for building desktop software that boasts a polished and user-friendly interface.

Different Classes and Methods In AWT

Java's Abstract Window Toolkit (AWT) encompasses a variety of classes, each offering distinct graphical components and methods for creating graphical user interfaces (GUIs).

1. Frame Class: The Frame class serves as the basic window container in AWT. It provides a platform-independent window with a title bar, maximize/minimize buttons, and resizing capabilities. Methods like setTitle(), setSize(), and setVisible() allow developers to customize the frame's appearance and behavior. The add() method is used to add other components to the frame's content pane.

2. Button Class: The Button class represents a push-button component. It offers methods like setLabel() to set the button's text label and addActionListener() to register an action listener for button clicks.

3. Label Class: The Label class is used to display text or images. Methods like setText() and setIcon() allow setting the label's content, while setAlignment() controls the text alignment within the label.

4. TextField Class: The TextField class provides a single-line text input field. Methods like getText() retrieve the entered text, and setText() modifies the field's content.

5. TextArea Class: For multiline text input, the TextArea class is used. It offers methods to set and retrieve text content, as well as to control the layout and scrolling behavior.

6. Checkbox Class: The Checkbox class represents a checkbox component that can be selected or deselected. Methods like getState() and setState() manage the checkbox's state.

7. Choice Class: A dropdown menu is created using the Choice class. It includes methods to add and remove items from the menu, as well as to get the selected item.

8. Panel Class: The Panel class acts as a container to group other components. It helps manage layout by allowing components to be added and organized within it.

9. Layout Managers: AWT offers layout manager classes like FlowLayout, BorderLayout, and GridLayout to control the positioning and sizing of components within containers.

10. Event Handling: AWT provides event handling methods such as addActionListener(), addMouseListener(), and addKeyListener() to register listeners for various types of events like button clicks, mouse actions, and keyboard inputs.

11. Scrollbar Class: The Scrollbar class creates horizontal or vertical scrollbars that allow users to navigate through content that doesn't fit within a component. Methods like setValue() and getValue() control the scrollbar's position.

12. List Class: The List class implements a scrolling list of items. It offers methods like add() and remove() to manage the items in the list, and getSelectedIndex() to get the index of the selected item.

13. CheckboxGroup Class: The CheckboxGroup class is used to group related checkboxes together so that only one can be selected at a time. Checkboxes can be associated with a group using the setCheckboxGroup() method.

14. FileDialog Class: The FileDialog class displays a file dialog that allows users to select files or directories. Methods like setMode() control the mode (open or save), and getFile() retrieves the selected file.

15. Font Class: The Font class represents fonts and provides methods to control font style, size, and attributes. It's used in components like Label and Button to set text fonts.

16. Canvas Class: The Canvas class is a blank area where custom graphics can be drawn. Methods like paint() allow developers to implement custom rendering.

17. Choice Class: The Choice class creates a dropdown menu from which users can select an item. Methods like addItem() and getSelectedIndex() manage the items and retrieve the selected index.

18. Dialog Class: The Dialog class creates a modal dialog box that overlays the main window. Methods like setTitle() and setModal() control the dialog's appearance and modality.

19. MenuBar and Menu Class: The MenuBar class creates a menu bar, while the Menu class creates a dropdown menu within the menu bar. Methods like add() add menus and menu items to the bar.

20. FileDialog Class: The FileDialog class displays a file dialog box for choosing files. Methods like setMode() set the dialog mode (open or save), and getFile() retrieves the selected file.

These classes and methods in AWT collectively enable developers to design interactive and functional graphical user interfaces, ranging from basic windows to complex layouts, providing the foundation for Java desktop applications.

Different Classes and Methods in Swing

1. JButton Class: Creates a button component. Methods like setText() and addActionListener() handle button text and actions.

2. JLabel Class: Displays text or images. Methods include setText(), setIcon(), and setHorizontalAlignment() for content display.

3. JTextField Class: Provides a single-line text input field. Methods like getText() and setText() manage input content.

4. JTextArea Class: Enables multiline text input. Methods like getText() and setText() work with larger text content.

5. JCheckBox Class: Represents a checkbox component. Methods like isSelected() and addItemListener() handle checkbox state and events.

6. JRadioButton Class: Creates a radio button component. Methods like isSelected() and addItemListener() manage state and events.

7. JComboBox Class: Creates a dropdown combo box. Methods like addItem() and getSelectedItem() handle items and selections.

8. JList Class: Represents a list component. Methods like setListData() and getSelectedValue() manage items and selections.

9. JScrollPane Class: Adds scrollbars to other components. Methods like setViewportView() and setVerticalScrollBarPolicy() control scrolling behavior.

10. JPanel Class: Serves as a container for other components. Methods like add() and setLayout() manage component arrangement.

11. JDialog Class: Creates a dialog box. Methods like setTitle(), setSize(), and setVisible() handle dialog properties.

12. JFileChooser Class: Presents a file dialog for file selection. Methods like showOpenDialog() and showSaveDialog() display the dialog.

13. JMenuBar and JMenu Classes: Create menu bars and menus. Methods like add() and addSeparator() manage menu items.

14. JPopupMenu Class: Generates popup menus. Methods like add() and show() manage popup content and display.

15. JSlider Class: Creates a slider component. Methods like setValue() and addChangeListener() control slider values and events.

16. JProgressBar Class: Represents a progress bar. Methods like setValue() and setString() manage progress and text display.

17. JTabbedPane Class: Provides a tabbed pane component. Methods like addTab() and setSelectedIndex() manage tabs and selections.

18. JTable Class: Creates a table component. Methods like setModel() and getValueAt() manage table data and cell values.

19. JFileChooser Class: Displays a file dialog for selecting files or directories. Methods like showOpenDialog() and showSaveDialog() manage dialog behavior.

20. JFrame Class: Represents a top-level window with decorations. Methods include setTitle(), setSize(), and setVisible() to manage window properties.

The Required Functions In Sudoku Game

1. startGame(): The `startGame()` function initiates the Sudoku game by setting up the user interface and initializing the game environment. It prepares the initial state of the grid and allows users to interact with the puzzle.

2. initialiseArrays(): The `initialiseArrays()` function creates the necessary data structures to hold the Sudoku grid and its solution. It allocates memory for arrays or lists that store the numbers entered by the user and the solution generated by the program.

3. initialiseInputs(): The `initialiseInputs()` function sets up the input fields where players can enter numbers into the Sudoku grid. It ensures that the grid is ready to accept user input and prepares the interface for gameplay.

4. isValid(): The `isValid()` function checks whether the number entered by the user in a specific cell is valid according to the rules of Sudoku. It verifies that the number does not violate any row, column, or box constraints.

5. addToGrid(): The `addToGrid()` function adds a valid number to the Sudoku grid based on the user's input. It updates the grid's state and visual representation to reflect the new entry.

6. createGrid(): The `createGrid()` function generates the initial state of the Sudoku grid, creating a puzzle with empty cells that players can fill in. It ensures that the generated puzzle is solvable and follows Sudoku rules.

7. printInitialGrid(): The `printInitialGrid()` function displays the initially generated Sudoku grid with some empty cells on the user interface. This provides players with the starting point for solving the puzzle.

8. generateSolution(): The `generateSolution()` function calculates and generates a valid solution for the Sudoku puzzle. It uses backtracking or other algorithms to find a complete solution that adheres to the rules.

9. setValueInput(): The `setValueInput()` function sets the value of a selected cell in the Sudoku grid based on the user's input. It updates the grid's state and UI display accordingly.

10. selectNumber(): The `selectNumber()` function handles the process of selecting a number from a list of available numbers for input into the grid. It allows players to choose a number to place in the selected cell.

11. initComponents(): The `initComponents()` function initializes various graphical user interface components required for the Sudoku game, such as buttons, labels, and input fields. It sets up the visual layout of the game interface.

Conclusion

In conclusion, the implementation of a Sudoku game in Java involves a comprehensive integration of various programming concepts and techniques. From the fundamental principles of object-oriented programming to the intricate mechanics of graphical user interface design, the project encompasses an array of skills.

By leveraging Java's versatile features, such as classes, methods, and event handling mechanisms, a Sudoku game can be developed that not only adheres to the game's logical rules but also provides an interactive and visually engaging experience for players.

The project delves into the intricacies of user input validation, grid generation, and solution algorithms, offering a well-rounded learning opportunity for developers. Furthermore, the utilization of Java's AWT and Swing libraries allows for the creation of an intuitive and user-friendly interface.

By crafting a range of functions, from grid initialization to validation and solution display, the Sudoku game showcases the practical application of programming concepts in a context that challenges both logic and creativity. Overall, the Sudoku game project in Java serves as a platform for honing coding skills while delivering a rewarding and entertaining user experience.

----- THE END -----