Basic Java Programming Structure

1. ***Has basic knowledge of the relationships of a Java package, class, object, and method.***
   - A package is a namespace for organizing classes, a class is a blueprint for objects, an object is an instance of a class, and a method is a function defined within a class.
2. ***Has a clear understanding of what JDK, JRE, and JVM mean to the Java language.***
   - JDK (Java Development Kit) is a software development environment for building applications, JRE (Java Runtime Environment) provides libraries and other components to run applications, and JVM (Java Virtual Machine) executes Java bytecode.
3. ***Able to distinguish the difference between .java and .class extensions.***
   - A .java file contains the source code written in Java, while a .class file contains the compiled bytecode that the JVM executes.
4. ***Can name a few of the Java keywords, special characters, and their importance.***
   - Keywords like class, public, void, and special characters like {} are essential for defining the structure and behavior of Java programs.
5. ***Able to explain the importance of syntax and semantics in a Java program.***
   - Syntax ensures that the code is written correctly according to the language rules, while semantics ensures that the code behaves as intended.
6. ***Can enumerate different Java comments and expound their significance.***
   - Java comments include single-line (//), multi-line (/* ... */), and Javadoc (/** ... */), which are used to explain code, making it more readable and maintainable.
7. ***Has a clear understanding of the difference between local and global variables.***
   - Local variables are declared within a method and accessible only within that method, whereas global variables (class fields) are declared within a class but outside any method and can be accessed by any method in the class.
8. ***Able to explain how the main method works in every Java program.***
   - The main method is the entry point of a Java program where the JVM starts execution.

Data Types

9. ***Knows the importance of data types in programming.***
   - Data types define the kind of data a variable can hold, ensuring proper memory allocation and data manipulation.
10. ***Can name/enumerate three different primitive data types in Java.***
    - Common primitive data types in Java include int, double, and char.
11. ***Able to discriminate different primitive data types and give an example of each data type.***
    - int for integers, double for floating-point numbers, and char for single characters.
12. ***Has a basic idea of how reference data types work in Java.***
    - Reference data types, like objects and arrays, store references (memory addresses) to the actual data.
13. ***Explain the difference between primitive data types and reference data types.***
    - Primitive data types store actual values, whereas reference data types store addresses of objects or arrays in memory.

Variables

14. ***Able to expound on the idea and importance of identifiers in Java programming.***
    - Identifiers are names given to elements like classes, variables, and methods, allowing for meaningful and readable code.
15. ***Knows and explains the rules in naming identifiers (class, variable, methods, etc.).***
    - Identifiers must start with a letter, underscore, or dollar sign and cannot be a Java keyword or contain spaces.
16. ***Able to describe the significance of variables in Java programming.***
    - Variables store data values that can be used and manipulated throughout a program.
17. ***Knows how to declare a variable and initialize it.***
    - A variable is declared by specifying its data type and name, and initialized by assigning it a value, e.g., int num = 5;.
18. ***Can explain the idea of constant variables (using the final keyword).***
    - Constant variables, declared with the final keyword, cannot have their value changed once assigned.

Basic Input/Output

19. ***Able to explain input/output statements (System.in and System.out).***
    - System.in is used for input from the keyboard, while System.out is used to output data to the console.
20. ***Describe how print(), println(), and printf() methods work in the program.***
    - print() outputs data without a newline, println() outputs data with a newline, and printf() allows formatted output.
21. ***Able to explain the purpose of the Scanner class in the program.***
    - The Scanner class is used to read input from various sources, including user input from the console.
22. ***Explain the difference between nextInt(), nextDouble(), next(), nextLine(), and so on.***
    - Methods like nextInt(), nextDouble(), next(), and nextLine() are used to read different types of input, such as integers, doubles, single words, and entire lines, respectively.
23. ***Able to write a simple program using input/output statements and the Scanner class.***
    - A program that takes user input using the Scanner class and prints it out using System.out.println().
24. ***Explain and describe how JOptionPane class works in the program.***
    - JOptionPane provides standard dialog boxes such as message, input, and confirmation dialogs for graphical user input/output.
25. ***Differentiate showInputDialog() from showMessageDialog().***
    - showInputDialog() prompts the user for input, while showMessageDialog() displays a message to the user.
26. ***Able to write a simple program using the JOptionPane class.***
    - A program that uses JOptionPane.showInputDialog() to get user input and JOptionPane.showMessageDialog() to display it.
27. ***Knows the idea of typecasting and parsing a value and its importance to Java programming.***
    - Typecasting and parsing convert data from one type to another, ensuring proper data manipulation and compatibility.

Control Structures (Selection)

28. ***Can name relational/equality operators and evaluate their expressions.***
    - Relational operators include ==, !=, <, >, <=, and >=, used to compare values in expressions.
29. ***Can name the three logical operators and evaluate their expressions.***
    - The three logical operators are && (and), || (or), and ! (not), used to combine or invert boolean expressions.
30. ***Explain the importance of control structures (selection) in the program.***

- Control structures allow the program to make decisions and execute different code paths based on conditions.

31. ***Write sample codes to perform conditional statement (?:).***
    - An example of the ternary operator: int result = (a > b) ? a : b;.
32. ***Explain the four selection structures: one-way, two-way, compound, and multiple.***
    - One-way (if), two-way (if-else), compound (if-else if-else), and multiple (switch) selection structures control the flow based on conditions.
33. ***Able to write a sample code using a one-way selection structure (Single-If).***
    - if (x > 0) { System.out.println("Positive"); }
34. ***Able to write a sample code using a two-way selection structure (If-else).***
    - if (x > 0) { System.out.println("Positive"); } else { System.out.println("Non-positive"); }
35. ***Write a sample code using a compound selection structure (If-else with multiple statements).***
    - if (x > 0) { System.out.println("Positive"); } else if (x < 0) { System.out.println("Negative"); } else { System.out.println("Zero"); }
36. ***Able to write a sample code using multiple selection structures (Nested-if).***
    - if (x > 0) { if (x < 10) { System.out.println("Single digit positive"); } }
37. ***Describe the idea of switch structure and write a sample code of it.***
    - A switch structure evaluates an expression and executes the matching case block: switch (day) { case 1: System.out.println("Monday"); break; ... }

Control Structures (Iteration)

38. ***Able to clearly explain what an iteration is and its importance in every program.***
    - Iteration repeatedly executes a block of code as long as a condition is met, essential for tasks like traversing arrays or processing input.
39. ***Enumerate the four different looping mechanisms.***
    - The four looping mechanisms are for, while, do-while, and enhanced for loops.
40. ***Describe and write a sample application of a counter-controlled loop.***
    - A counter-controlled loop uses a counter variable: for (int i = 0; i < 10; i++) { System.out.println(i); }
41. ***Describe and write a sample application of a sentinel-controlled loop.***
    - A sentinel-controlled loop continues until a sentinel value is encountered: while (input != -1) { input = scanner.nextInt(); }
42. ***Describe and write a sample application of a flag-controlled loop.***
    - A flag-controlled loop uses a boolean flag to control iteration: boolean flag = true; while (flag) { ... flag = false; }
43. ***Explain and write a sample code using a while loop.***
    - while (x > 0) { System.out.println(x); x--; }

Control Structures (Iteration)

46. ***Know the difference between break and continue.***
    - break terminates the loop entirely, while continue skips the current iteration and proceeds to the next one.

Methods

47. ***Has a deep understanding of what a method is and how important it is in software development.***
    - A method is a block of code designed to perform a specific task, promoting code reusability and modularity in software development.
48. ***Can explain the relationship between a class and a method in a program.***
    - A class contains methods, which define the behaviors and actions that objects created from the class can perform.
49. ***Able to define what a predefined method is and name a few of them.***
    - Predefined methods are built-in functions provided by Java, such as System.out.println(), Math.sqrt(), and String.length().
50. ***Explain clearly what a user-defined method is and what its significance is in software development.***
    - A user-defined method is a custom method created by the programmer to perform specific tasks, enhancing code organization and reusability.
51. ***Able to explain the difference between formal and actual parameters.***
    - Formal parameters are variables defined in the method signature, while actual parameters are the values passed to the method when it is called.
52. ***Explain how the non-value returning (void), the non-parameterized method works and be able to write sample codes.***
    - A void, non-parameterized method performs a task without returning a value or requiring input, e.g., void printMessage()
{ System.out.println("Hello"); }.
53. ***Explain how the non-value returning (void), the parameterized method works and be able to write sample codes.***
    - A void, parameterized method performs a task without returning a value but requires input, e.g., void printMessage(String message)
{ System.out.println(message); }.
54. ***Explain how value returning, the non-parameterized method works and be able to write sample codes.***
    - A value-returning, non-parameterized method performs a task and returns a value without requiring input, e.g., int getNumber() { return 5; }.
55. ***Explain how value returning parameterized method works and be able to write sample codes.***
    - A value-returning, parameterized method performs a task, returns a value, and requires input, e.g., int add(int a, int b) { return a + b; }.

Arrays and Dynamic Arrays

56. ***Explain the importance of arrays and dynamic arrays.***
    - Arrays and dynamic arrays are essential for storing and managing collections of data efficiently.
57. ***Utilized arrays in the program to manage and maintain the data properly.***
    - Arrays allow for efficient data management and manipulation through indexed access to elements.
58. ***Knows how to manipulate records in an array.***
    - Records in an array can be manipulated by accessing and modifying elements using their indices.
59. ***Able to differentiate arrays and dynamic arrays.***
    - Arrays have a fixed size, whereas dynamic arrays can grow or shrink in size as needed.
60. ***Use arrays properly in the program.***
    - Proper use of arrays involves initializing, accessing, and modifying elements correctly.

File Handling Techniques

61. ***The use of the file is fully implemented in maintaining the data.***
    - File handling is crucial for data persistence, allowing programs to read from and write to files.

62. ***Able to create multiple text files for data organization and record-keeping.***
   - Creating text files helps in organizing and maintaining records systematically.
63. ***Has exceptional knowledge regarding how to properly handle files to keep all the records secure, stable, consistent, and well-managed.***
   - Proper file handling involves techniques to ensure data security, stability, and consistency.
64. ***Knows when to use File, FileReader, and FileWriter classes in the program.***
   - File is used to create and access file attributes, FileReader to read data from files, and FileWriter to write data to files.
65. ***Create multiple file structures that are logically related to enforcing data integrity and security of the data.***
   - Logical file structures help in maintaining data integrity and ensuring data security.

Encapsulation
66. ***Can define and explain the importance of Encapsulation.***
   - Encapsulation is the principle of bundling data and methods within a class and restricting access to certain components, ensuring data integrity and security.
67. ***Able to describe each component of a class Encapsulated (property and behavior).***
   - An encapsulated class contains properties (fields) and behaviors (methods) that define its state and actions.
68. ***Evaluate the difference between private, public, and protected modifiers of a class.***
   - Private members are accessible only within the class, public members are accessible from anywhere, and protected members are accessible within the package and subclasses.
69. ***Able to recommend the use of static and non-static field members of a class.***
   - Static members belong to the class itself, while non-static members belong to instances of the class, allowing for flexible and efficient code design.
70. ***Has sufficient understanding of the importance of a constructor.***
   - Constructors initialize new objects, setting up initial states and performing necessary setup tasks.
71. ***Able to name different types of constructors and explain each.***
   - Default constructors take no arguments, parameterized constructors take arguments to set initial values, and copy constructors create a new object as a copy of an existing object.
72. ***Explain the idea of mutators (setters) and accessors (getters) of a certain class.***
   - Mutators (setters) modify the value of private fields, while accessors (getters) retrieve the value, ensuring controlled access to the class's properties.
73. ***Able to create single or multiple instances of an encapsulated class.***
   - Instances of an encapsulated class can be created using the new keyword, allowing for multiple objects with independent states.
74. ***Knows how to call or access the property and behavior of a class using an Object and a Class itself.***
   - Properties and behaviors of a class can be accessed through object references or class names for static members.
75. ***Has a deeper understanding of how instance variables and class variables are created.***
   - Instance variables are created with each object instance, while class variables are shared among all instances of the class.

Inheritance
76. ***Has an explicit understanding of the concept of code-reuse and code-recycle through derivation.***
   - Inheritance allows for code reuse by creating new classes based on existing ones, enhancing maintainability and reducing redundancy.
77. ***Able to describe how an is-A relationship works and the purpose of the extends keyword in such a concept.***
   - An is-A relationship indicates inheritance, where a subclass extends a superclass, inheriting its properties and behaviors.
78. ***Has sufficient knowledge of the underlying terms, which are superclass and subclass (Parent and child, base and derived class).***
   - The superclass (parent or base class) provides common properties and behaviors, while the subclass (child or derived class) inherits and can extend or override them.
79. ***Able to describe each Inheritance type and its hierarchy (i.e., single, hierarchical, multiple, etc.).***
   - Single inheritance involves one superclass, hierarchical inheritance involves multiple subclasses from one superclass, and multiple inheritance (through interfaces) involves a class implementing multiple interfaces.
80. ***Knowledgeable in extending other user-defined and built-in classes.***
   - Extending classes involves creating new classes based on existing ones, adding or modifying functionality as needed.

Polymorphism
81. ***Has explicit knowledge of Polymorphism as an OOP concept.***
   - Polymorphism allows objects to be treated as instances of their parent class, enabling dynamic method binding and code flexibility.
82. ***Can explain the difference between method overriding and method overloading.***
   - Method overriding occurs when a subclass provides a specific implementation of a method already defined in its superclass, while method overloading involves multiple methods with the same name but different parameters within the same class.
83. ***Able to expound the ideas of the essential terms as follows: Static Binding versus Dynamic Binding, Compile-time versus Runtime Polymorphism.***
   - Static binding (compile-time) occurs during compilation for method overloading, while dynamic binding (runtime) occurs during execution for method overriding.
84. ***Able to explain the key terms extends, super, and instanceOf.***
   - extends is used for inheritance, super refers to the superclass, and instanceOf checks if an object is an instance of a specific class.
85. ***Knows the idea of typecasting: upcasting and downcasting an instance of a class.***
   - Upcasting converts a subclass reference to a superclass reference, while downcasting converts a superclass reference back to a subclass reference.

Windows Programming and Event Handling
86. ***Can name different components containing Java swing components.***
   - Swing components include JButton, JLabel, JTextField, JPanel, and more for building GUI applications.
87. ***Able to apply swing components layout in designing the application properly.***
   - Proper layout management in Swing involves using layout managers like BorderLayout, FlowLayout, and GridLayout.
88. ***Knowledgeable in applying how OOP underlying concepts are successfully used to each of these swing components.***
   - OOP concepts like inheritance, encapsulation, and polymorphism are used to create and manage Swing components effectively.