Scope of a variable

The scope of a vriable is the part of the program for which the declaration is in effect.

In Java, the scope of such a variable is from its declaration to the end of the method.

Lifetime of a variable

The lifetime of a variable is the time period in which the variable has valid memory.

In Java, the lifetime of a variable is the period of time beginning when the method is entered and ending when execution of the method terminates.

Types of Scope of a varibale:

- Local scope: “visible” within function or statement block from point of declaration until the end of the block.
- Global scope: visible everywhere unless “hidden”.
- Class scope: “seen” by class members.
- Namespace scope: visible within namespace block.
- File scope: visible within current text file.

Type with lifetime of variable:

- Static: A static variable is stored in the data segment of the “object file” of a program. Its lifetime is the entire duration of the program’s execution.
- Automatic: An automatic variable has a lifetime that begins when program execution enters the function or statement block or compound and ends when execution leaves

the block. Automatic variables are stored in a “function call stack”.

- Dynamic: The lifetime of a dynamic object begins when memory is allocated for the object (e.g., by a call to malloc() or using new) and ends when memory is deallocated (e.g., by a call to free() or using delete). Dynamic objects are stored in “the heap”.

Related posts:

1. Sequence Control & Expression | PPL
2. PPL:Named Constants
3. Parse Tree | PPL | Prof. Jayesh Umre
4. Basic elements of Prolog
5. Loops | PPL | Prof. Jayesh Umre
6. Subprograms Parameter passing methods | PPL | Prof. Jayesh Umre
7. Programming Paradigms | PPL | Prof. Jayesh Umre
8. Subprograms Introduction | PPL | Prof. Jayesh Umre
9. Phases of Compiler | PPL | Prof. Jayesh Umre
10. Parse Tree | PPL
11. Influences on Language design | PPL | Prof. Jayesh Umre
12. Fundamentals of Subprograms | PPL | Prof. Jayesh Umre
13. Programming Paradigm
14. Influences on Language Design
15. Language Evaluation Criteria
16. OOP in C++ | PPL
17. OOP in C# | PPL
18. OOP in Java | PPL
19. PPL: Abstraction & Encapsulation
20. PPL: Semaphores

21. PPL: Introduction to 4GL
22. PPL: Variable Initialization
23. PPL: Conditional Statements
24. PPL: Array
25. PPL: Strong Typing
26. PPL: Coroutines
27. PPL: Exception Handler in C++
28. PPL: OOP in PHP
29. PPL: Character Data Type
30. PPL: Exceptions
31. PPL: Heap based storage management
32. PPL: Primitive Data Type
33. PPL: Data types
34. Programming Environments | PPL
35. Virtual Machine | PPL
36. PPL: Local referencing environments
37. Generic Subprograms
38. Local referencing environments | PPL | Prof. Jayesh Umre
39. Generic Subprograms | PPL | Prof. Jayesh Umre
40. PPL: Java Threads
41. PPL: Loops
42. PPL: Exception Handling
43. PPL: C# Threads
44. Pointer & Reference Type | PPL
45. Design issues for functions
46. Parameter passing methods
47. Fundamentals of sub-programs

48. Subprograms
49. Design issues of subprogram
50. Garbage Collection
51. Issues in Language Translation
52. PPL Previous years solved papers
53. Type Checking | PPL | Prof. Jayesh Umre
54. PPL RGPV May 2018 solved paper discussion| Prof. Jayesh Umre
55. PPL Viva Voce
56. PPL RGPV June 2017 Solved paper | Prof. Jayesh Umre
57. Concurrency
58. Basic elements of Prolog
59. Introduction and overview of Logic programming
60. Application of Logic programming
61. PPL: Influences on Language Design
62. Language Evaluation Criteria PPL
63. PPL: Sequence Control & Expression
64. PPL: Programming Environments
65. PPL: Virtual Machine
66. PPL: Programming Paradigm
67. PPL: Pointer & Reference Type
68. try-catch block in C++