1. Chapter 1
   1. Orthogonality
      1. A relative small set of primitive constructs can be combined in a relatively small number of ways to build the control and data structures of the language
   2. **Design trade off**
      1. **Readability**
      2. **Efficiency (writability)**
      3. **Safety (reliability)**
      4. **Checking the index out of bounds at runtime improves efficiency because the compiler does not need to check each array index. However, this causes reliability issues especially in languages like C because the program can access an array index outside of the array bounds which returns a garbage value, or causes the program to segmentation fault at run time if memory allocation is attempted on an index that is out of bounds. To handle this java implements exception handling which improve readability and reliability but reduces writability.**
   3. Interpreter vs Compiler
      1. Compiler
         1. Translates high-level programming languages into machine code
         2. Inputs the entire program generates machine code file
         3. Executed conditional statements faster because the entire machine code is passed and executed by the cpu
         4. Requires more space because comp has to store the entire object code in memory
         5. errors are reported after the entire program is checked
      2. Interpreter
         1. Code is executed by the interpreter line by line
         2. Execution stops if an error is found
         3. Takes longer to execute conditionals than compiler
         4. Less memory needed the line is simply being moved into the cpu then deleted after execution is complete
      3. Javac to get java bytecode
      4. Java bytecode is used by the java interpreter
2. Chapter 3
   1. Syntax Grammar Rules
   2. BNF, **EBNF**
      1. Ambiguous grammar
   3. Attributes
      1. Device used to describe more of the structure of a programming language it is an extension of context-free grammar
      2. **Inherited attribute**
      3. **Synthesized attribute**
   4. Static Semantics
      1. The legal forms of programs (example type constraints) checked at compile time
   5. Operational Semantics
      1. Describe the meaning of a statement or program by specifying the effects of running it on a machine
   6. Denotational Semantics
      1. Look at notes for this do not need the error checking from the textbook
      2. How to use denotational semantics to explain the grammar rule
3. Chapter 4
   1. Lexical analyzer
      1. Small language constructs like names and literals
   2. Syntax analyzer
      1. Deals with large scale constructs- expressions, statements, program units
   3. Reason for separation
      1. Simplicity
         1. Lexical analyzer is simple separating reduces syntax analyzer’s size
      2. Efficiency
         1. Lexical analyzer is optimizable syntax analyzer is not
      3. Portability
         1. Syntax analyzer can be machine independent lex can not
   4. Top down recursive Descent
   5. Know what a bottom up parser is do need to know the implementation
4. Chapter 5
   1. Variable 6 tuples
      1. Name
         1. Syntax used to create a valid string to abstract the address location where the variable is declared
      2. Address (L-Value)
         1. When a name of a variable appears in the left side of an assignment represents the address where the variable is pointing too
      3. Type
         1. Determines the range of values and the set of operations that are defined for the values of the type
      4. Value (R-Value)
         1. The contents of the memory cell or cells associated with the variable
            1. To access the r value of a variable the l value must be determined first
      5. Lifetime
         1. Def birth when the binding is needed death when the binding is not needed and can be deleted
         2. The time beginning when a variable is allocated a memory cell location and ending when a variable is deallocated a memory cell location
         3. Memory diagram
      6. Scope
         1. Static
            1. If there is no local reference to the variable in the current method it extends to the next larger container checks if there is a reference to it if there is not continues to the next larger container if none is found it errors out
            2. If the binding occurs before runtime and remains unchanged throughout program execution. Binding occurs during declaration
            3. Explicit Declaration

Statement in a program that lists variable names and specifies their type

* + - * 1. Implicit Declaration

Associating variables with types through default conventions rather than declaration statements. Aka the first appearance of a variable name constitutes its declaration type

* + - * 1. Type Inference

Implicit declaration using context type is determined by the type of the initial value

* + - 1. Dynamic
         1. The last assignment is traced back through the method calls regardless of the size of the called method
         2. Binding first occurs during run time or binding can change in the course of program execution AKA BINDING OCCURS WHEN IT IS ASSIGNED A VALUE
         3. Technically no variable actually has an assigned type its just associated with a typed value
         4. Provides more flexibility but more costly and more errors

1. **ML**