Lexical Structure (Chapter 3, JLS)

- A Java source code file is viewed as a string of unicode characters, including line separators.
- A Java source code file is segregated into recognizable substrings known as input elements.
- Some input elements—white space and comments—are ignored.
- Some input elements—identifiers, literals, separators and operators—are relevant.

- White Space
  \[ \text{(HT | LF | VT | FF | CR | \(\alpha\))}^+ \]
  - ignored by the compiler
  - serves to delimit other input elements
  - subject to conventions on indentation, spacing, empty lines

- Comment
  
  /* ... */
  
  // ... (CR | LF | CR LF)
  
  /** ... */
  
  - ignored by the compiler
  - traditional, end-of-line and documentation comments

- Identifier
  
  \( (\text{Letter} | \_ | \$)(\text{Letter} | \_ | \text{Digit} | \_ | \$)^* \)
  
  - names a variable or method (beginning conventionally with a lower-case letter)
  - names a class (beginning conventionally with an upper-case letter)
  - some identifiers are reserved, known as keywords
• Integer Literal
  
  \[0 \mid 1-9Digit^*[l \mid L]\]
  \[0(x \mid X)(Digit \mid a-f \mid A-F)^*[l \mid L]\]
  \[0(0-7)^*[l \mid L]\]

  - represents an integer number
  - decimal, hexadecimal and octal integer literals

• Floating-Point Literal
  
  \[Digit^+.Digit^*[f \mid F \mid d \mid D]\]
  \[Digit^+.Digit^*(e \mid E)[+ \mid -]Digit^+[f \mid F \mid d \mid D]\]
  \[.Digit^+[f \mid F \mid d \mid D]\]
  \[.Digit^+(e \mid E)[+ \mid -]Digit^+[f \mid F \mid d \mid D]\]
  \[Digit^+[f \mid F \mid d \mid D]\]
  \[Digit^+(e \mid E)[+ \mid -]Digit^+[f \mid F \mid d \mid D]\]

  - represents an (approximate) real number
  - decimal or scientific notation floating-point literals

• Boolean Literal
  
  \[true \mid false\]
• Character Literal

'Character'
- represents a single Unicode character
- characters ' and \ are represented in a string by \ and \\

• String Literal

"Character"
- represents a finite sequence of Unicode characters
- characters " and \ are represented in a string by \" and \\
- characters HT, LF, FF and CR are represented in a string by \t, \n, \f and \r

• Separator

( | ) | [ | ] | { | } | ; , .

• Operator

= | > | < | ! | ~ | - | ? | : |
== | <= | >= | != | && | || | ++ | -- |
+ | - | * | / | & | | ~ | % | << | >> | >>> |
+= | -= | *= | /= | &= | | -= | %= | <<= | >>= | >>>=
## Types, Values and Variables (Chapter 4, JLS)

<table>
<thead>
<tr>
<th>Primitive Types</th>
<th>Values</th>
<th>Representation</th>
</tr>
</thead>
<tbody>
<tr>
<td>boolean</td>
<td>{false, true}</td>
<td>1-bit (possibly padded to 1 byte)</td>
</tr>
</tbody>
</table>

### Numeric Types

#### Integral Types

- \( \subseteq \mathbb{Z} \)
- **byte**
  - \(-2^7 \ldots (2^7 - 1)\)
  - 8-bit signed two’s complement
- **short**
  - \(-2^{15} \ldots (2^{15} - 1)\)
  - 16-bit signed two’s complement
- **int**
  - \(-2^{31} \ldots (2^{31} - 1)\)
  - 32-bit signed two’s complement
- **long**
  - \(-2^{63} \ldots (2^{63} - 1)\)
  - 64-bit signed two’s complement
- **char**
  - \(0 \ldots (2^{16} - 1)\)
  - 16-bit Unicode (www.unicode.org)

#### Floating-Point Types

- \( \subseteq \mathbb{R} \)
- **float**
  - 32-bit ANSI/IEEE Standard 754
- **double**
  - 64-bit ANSI/IEEE Standard 754

### Reference Types

<table>
<thead>
<tr>
<th>Class Type</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>e.g. Object</td>
<td>user-defined objects</td>
</tr>
<tr>
<td>e.g. String</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Array Type</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>e.g. int[]</td>
<td>sequence of Type</td>
</tr>
<tr>
<td>e.g. String[]</td>
<td></td>
</tr>
</tbody>
</table>
Variables

Primitive Types
- Value stored in variable
  e.g. boolean x = true;
  e.g. int y = 5;
  e.g. char z = 'A';
- Default value stored is false or 0
  e.g. boolean x;
  e.g. int y;
- final variables can be assigned at most once
  e.g. final int SIZE = 10;
  e.g. final char EXCELLENT = 'A';
  e.g. final float PI = 3.14159;

Reference Types
- Reference (i.e., pointer) to value (i.e., object) stored in variable
  e.g. String x = "hello";
  e.g. Widget y = new Widget();
  e.g. int[] z = {3, 2, 4, 5};
- Default reference stored is null, which refers to nothing
  e.g. String x;
  e.g. int[] y;
- final variables can be assigned at most once; however, objects to which they refer can be modified
  e.g. final String GREETING = "hello";
  e.g. final int[] CUTOFFS = {90, 80, 70, 60};
Expressions (Chapter 15, JLS)

Literals

Variables

Operations:

<table>
<thead>
<tr>
<th>Prec./Assoc.</th>
<th>Category</th>
<th>Operator</th>
<th>Operand(s)</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>high n/a</td>
<td>Postfix</td>
<td>++ --</td>
<td>numeric variable</td>
<td>numeric value, before side effect</td>
</tr>
<tr>
<td>↓ n/a</td>
<td>Prefix</td>
<td>++ --</td>
<td>numeric variable</td>
<td>numeric value, after side effect</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>numeric (+ -)</td>
<td>≥ 32-bit numeric value</td>
</tr>
<tr>
<td>↓ ←</td>
<td>Unary</td>
<td>+ - ~ !</td>
<td>integral (-)</td>
<td>≥ 32-bit integral value</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>boolean (!)</td>
<td>boolean value</td>
</tr>
<tr>
<td>↓ →</td>
<td>Multiplicative</td>
<td>* / %</td>
<td>numeric, numeric</td>
<td>≥ 32-bit numeric value</td>
</tr>
<tr>
<td>↓ →</td>
<td>Additive</td>
<td>+ -</td>
<td>numeric, numeric</td>
<td>≥ 32-bit numeric value</td>
</tr>
<tr>
<td>↓ →</td>
<td>Shift</td>
<td>&lt;&lt; &gt;&gt; &gt;&gt;&gt;</td>
<td>integral, integral</td>
<td>≥ 32-bit integral value</td>
</tr>
<tr>
<td>↓ n/a</td>
<td>Relational</td>
<td>&lt; &gt; &lt;= &gt;=</td>
<td>numeric, numeric</td>
<td>boolean value</td>
</tr>
<tr>
<td>↓ →</td>
<td>Equality</td>
<td>== !=</td>
<td>boolean, boolean</td>
<td>boolean value</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>numeric, numeric</td>
<td>boolean value</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>reference, reference</td>
<td></td>
</tr>
<tr>
<td>↓ →</td>
<td>Bitwise</td>
<td>&amp; ^</td>
<td>integral, integral</td>
<td>≥ 32-bit integral value</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>boolean, boolean</td>
<td>boolean value</td>
</tr>
<tr>
<td>↓ →</td>
<td>Boolean</td>
<td>&amp;&amp;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>↓ ←</td>
<td>Conditional</td>
<td>:=?</td>
<td>boolean, any, any</td>
<td>value, short-circuit</td>
</tr>
<tr>
<td>low ←</td>
<td>Assignment</td>
<td>*= /= %= += -= &lt;&lt;= &gt;&gt;= &gt;&gt;&gt;= &amp;= ^=</td>
<td>=</td>
<td>variable, any (=)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>numeric variable, numeric (*= /= %= += -=)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>integral variable, integral (&lt;&lt; &gt;&gt;= &gt;&gt;&gt;= &amp;= ^=</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>boolean variable, boolean (&amp;&amp; ^=</td>
<td>=)</td>
</tr>
</tbody>
</table>

- Order of operations determined by
  1) parentheses
  2) precedence
  3) associativity

- Evaluation: result (and possible side effect) obtained by
  1) evaluating left operand
  2) evaluating right operand (unless short-circuit)
  3) applying operation
Statements (Chapter 14, JLS)

- Block
  
  \{ Statement* \}
  
  - used to execute a sequence of statements

- Empty statement
  
  ;
  
  - has no effect

- Labeled statement
  
  Identifier : Statement
  
  - used with labeled break and continue statements

- Local variable declaration
  
  [final] Type Identifier [= Expression][, Identifier [= Expression]]* ;
  
  - used to create local variable(s) and optionally assign initial value(s)
  
  - scope of variable is from point of declaration to end of block in which declared

- Expression
  
  Expression ;
  
  - Expression must have a side effect

- if statement
  
  if ( Expression ) Statement [else Statement]
  
  - condition Expression must be of type boolean

- switch statement
  
  switch ( Expression ) { ((case Expression : )+ Statement*)*[default : Statement+] }
  
  - condition Expression must be integral
  
  - case Expression(s) must be integral constant (literal or final variable)
• while statement
  while ( Expression ) Statement
  - condition Expression must be of type boolean

• do statement
  do Statement while ( Expression ) ;
  - condition Expression must be of type boolean

• for statement
  for ( [Expression[, Expression]*]; [Expression]; [Expression[, Expression]*) ] ) Statement
  for ( LocalVariableDeclaration; [Expression]; [Expression[, Expression]*) ) Statement
  - initial Expression(s) must have a side effect
  - condition Expression must be of type boolean
  - update Expression(s) must have a side effect

• break statement
  break [Identifier] ;
  - must occur inside a switch, while, do or for statement

• continue statement
  continue [Identifier] ;
  - must occur inside a while, do or for statement
Arrays (Chapter 10, Sections 15.10 and 15.13, JLS)

- Arrays are objects that contain zero or more component variables.
- Arrays are homogeneous—components have the same type.
- Arrays are created dynamically (as the program executes).
- The length of an array is the number of components therein; it does not change after creation.

- Array type
  \[Type\ [\ ]\]^+
  - component type may be any \(Type\) (including array types)
  - dimension is number of pairs of \[

- Array initializer
  \{ [Expression , Expression]^* ] , \} \\
  \{ [ArrayInitializer , ArrayInitializer]^* ] , \} \\
  - creates an array
  - components are assigned from expression values
  - length is number of expressions or array initializers

- Array creation expression
  new ArrayType ArrayInitializer
  - creates an array having a specific component type
  - dimension, length and compatible component values can be given

  new Type ([ Expression ]^+ ([ ])^* \\
  - dimension and lengths (in some dimensions) can be given by expression values

- Array type local variable declaration
  [final] ArrayType Identifier [= Expression]( , Identifier [= Expression]^* ;
  [final] ArrayType Identifier [= ArrayInitializer]( , Identifier [= ArrayInitializer]^* ;
  - used to create local array variable(s) and optionally assign initial component value(s)
  - expressions must be array creation expressions
  - scope of variable is from point of declaration to end of block in which declared

- Array access expression
  Expression [ Expression ]
  - array reference expression must be array type and have non-null value
  - index expression must be integral type and have value between 0 and length-1
  - first, second, third, \ldots\ components are accessed by index expression values 0, 1, 2, \ldots

- Array length expression
  Identifier . length