import java.lang.System;
public class Area {
    public static void main(String[] args) {
        int width;
        width = 8;
        int height = 3;
        int area = width * height;
        System.out.println(area);
    }
}
import java.lang.System;
public class Area
{
    public static void main(String[] args)
    {
        int width;
        width = 8;
        int height = 3;
        int area = width * height;
        System.out.println(area);
    }
}

import java.lang.*;
public class Area
{
    public static void main(String[] args)
    {
        int width;
        width = 8;
        int height = 3;
        int area = width * height;
        System.out.println(area);
    }
}

**Lexicon**

---

**Lexical Elements**

Without worrying about syntax or semantics, let us identify the elements of a program:

- **Keywords**
- **Identifiers**
- **Literals**
- **Operators**
- **Separators**

---

**Style**

- **Class naming convention**
  A noun. Use Pascal/Title case, e.g. Math, ArrayList.

- **Method naming convention**
  A verb. Use camel case, e.g. equals, toString, isLeapYear

- **Variable naming convention**
  A noun. Use camel case, e.g. length, interestRate, gender
  Applies also to attributes and parameters.

- **Block layout**
  Braces must align vertically and the all statements must be left justified and indented by one tab position.

---

**Class**

- **Header**

**Class Body, a Block**

**Method**

- **Header**

**Method Body, a Block**
Lexicon

Keywords
- abstract
- assert
- boolean
- break
- byte
- case
- catch
- char
- class
- const
- continue
- default
- do
- double
- else
- enum
- extends
- final
- finally
- float
- for
- goto
- if
- implements
- import
- instanceof
- int
- interface
- long
- native
- new
- package
- private
- protected
- public
- return
- short
- static
- strictfp
- super
-switch
- synchronized
- this
- throw
- throws
- transient
- try
- void
- volatile
- while

Identifiers
- abstrac
- abstract
- assert
- boolean
- break
- byte
- case
- catch
- char
- class
- const
- continue
- default
- do
- double
- else
- enum
- extends
- final
- finally
- float
- for
- goto
- if
- implements
- import
- instanceof
- int
- interface
- long
- native
- new
- package
- private
- protected
- public
- return
- short
- static
- strictfp
- super
- switch
- synchronized
- this
- throw
- throws
- transient
- try
- void
- volatile
- while

Literals
- true
- false
- null

Operators
- =
- >
- <
- !
- ~
- ?
- :
- &
- |
- +
- *
- /
- ^
- %

Separators
- ,
- ;
- {
- }
- [ ]
- ( )

Example
Identify the language elements in the following program...

```java
import java.lang.System;
public class Area
{
    public static void main(String[] args)
    {
        int width;
        width = 8;
        int height = 3;
        int area = width * height;
        System.out.println(area);
    }
}
```
import java.lang.System;

public class Area {
    public static void main(String[] args) {
        int width;
        width = 8;
        int height = 3;
        int area = width * height;
        System.out.println(area);
    }
}

Compile Time vs Run Time Errors

- Before program can run, it must be compiled to (translated) Java bytecode
- Studio does this as you enter/edit your code; it flags compile-time errors:
  - Syntax errors, e.g. missing {
  - Type errors, e.g. “abc” * 3

Compile Time vs Run Time Errors

- When you execute your program, you may get runtime errors:
  - ArithmeticException, e.g. 10 / 0
  - ArrayIndexOutOfBoundsException, etc.
- Logic errors: program appears to run normally but does not behave as required
The Declaration Statement

The name of a primitive or non-primitive type, e.g. int, double...

- An identifier to be associated with a memory block
- Scope of the variable = the enclosing block of the declaration.
- The variable is not known outside its scope.
- Declaration does not initialize. Not with 0 or null or anything else.

### Primitive & Non-Primitive

- **Primitive**
  - number
  - character
  - boolean
  - class
  - interface
  - array

- **Non-Primitive**

### Numeric Types

- **Integer**
  - int
  - long
  - Integer literals are int by default unless suffixed with L

- **Real**
  - float
  - double
  - Real literals are recognized by a decimal point or an exponent. They are double by default unless suffixed with F. For exponential notation, use E.
### INTEGER OR REAL?
- **Integer**
  - Use for integer data, e.g. count.
  - 100% exact
- **Real**
  - Use for real data, e.g. amount.
  - Inherently inaccurate

### The Type **boolean**
- Stores the result on a condition
- Has only two possible values
- `true` and `false` are reserved words
- Boolean variables are not integers
- The Boolean operators are:
  - `&&` (for **and**), `||` (for **or**), and `^` (for xor)

*Note: Boolean literals are the easiest to recognize!*

### The Character Type **char**
- A letter, digit, or symbol
- Digits versus Numbers
- Store the code, not the typeface
- The case of English: ASCII vs Unicode
- **char** is thus an (unsigned) integer type
- No **char** operators! They auto-promote to **int**.

*Character literals are recognized by single quotes surrounding the character, e.g. ‘A’*

### More on Characters

<table>
<thead>
<tr>
<th>Code</th>
<th>Character</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>...</td>
</tr>
<tr>
<td>32</td>
<td><code>space</code></td>
</tr>
<tr>
<td></td>
<td>...</td>
</tr>
<tr>
<td>48-57</td>
<td><code>'0'-'9'</code></td>
</tr>
<tr>
<td></td>
<td>...</td>
</tr>
<tr>
<td>65-90</td>
<td><code>'A'-'Z'</code></td>
</tr>
<tr>
<td></td>
<td>...</td>
</tr>
<tr>
<td>97-122</td>
<td><code>'a'-'z'</code></td>
</tr>
<tr>
<td></td>
<td>...</td>
</tr>
<tr>
<td>65535</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Escape</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>\uXXXX</td>
<td>The character whose code is (hex) XXXX</td>
</tr>
<tr>
<td>\’</td>
<td>Single quote</td>
</tr>
<tr>
<td>&quot;</td>
<td>Double quote</td>
</tr>
<tr>
<td>\</td>
<td>Backslash</td>
</tr>
<tr>
<td>\n</td>
<td>New line</td>
</tr>
<tr>
<td>\r</td>
<td>Carriage return</td>
</tr>
<tr>
<td>\t</td>
<td>Form Feed</td>
</tr>
<tr>
<td>\</td>
<td>Tab</td>
</tr>
<tr>
<td>\b</td>
<td>Backspace</td>
</tr>
</tbody>
</table>
### Java's Primitive Type

<table>
<thead>
<tr>
<th>PRIMITIVE TYPES</th>
<th>Type</th>
<th>Size (bytes)</th>
<th>Approximate Range min</th>
<th>Approximate Range max</th>
<th>S.D.</th>
</tr>
</thead>
<tbody>
<tr>
<td>SIGNED NUMBER</td>
<td>byte</td>
<td>1</td>
<td>-128</td>
<td>+127</td>
<td>N/A</td>
</tr>
<tr>
<td></td>
<td>short</td>
<td>2</td>
<td>-32,768</td>
<td>+32,767</td>
<td>N/A</td>
</tr>
<tr>
<td></td>
<td>int</td>
<td>4</td>
<td>-2×10^9</td>
<td>+2×10^9</td>
<td>N/A</td>
</tr>
<tr>
<td></td>
<td>long</td>
<td>8</td>
<td>-9×10^18</td>
<td>+9×10^18</td>
<td>N/A</td>
</tr>
<tr>
<td>UNSIGNED REAL</td>
<td>char</td>
<td>2</td>
<td>0</td>
<td>65,535</td>
<td>N/A</td>
</tr>
<tr>
<td>SINGLE REAL</td>
<td>float</td>
<td>4</td>
<td>+3.4×10^38</td>
<td>+3.4×10^38</td>
<td>7</td>
</tr>
<tr>
<td></td>
<td>double</td>
<td>8</td>
<td>-1.7×10^310</td>
<td>+1.7×10^310</td>
<td>15</td>
</tr>
<tr>
<td>BOOLEAN</td>
<td>boolean</td>
<td>1</td>
<td>true/false</td>
<td></td>
<td>N/A</td>
</tr>
</tbody>
</table>

- • Stores a sequence of characters
- • Optimized for speed → immutable
- • Optimized declaration → shortcut
- • Optimized concatenation → + operator
- • Rich API (e.g. indexOf, charAt, substring)

Note: String literals are surrounded with double quotes and can use the same escape sequences as chars.

### Class Type   **String**  (in java.lang)

- Stores an instance of time
- Captures both date and time
- Accurate to a millisecond
- Simple API (toString and getTime)

Note: Like all class types (except for String), Date has no literals and no operators.

### Class Type   **Date**  (in java.util)

- Stores an instance of a rectangle
- Captures the height and width as int
- API (getArea and getCircumference)

Like all class types (except for String), it has no literals and no operators.

### Class Type   **RectangleModel**  (in our package)
Declaration

Class Type: FractionModel
(in our package)

- Stores an instance of a fraction
- Numerator and denominator are long
- API (add, sub, multiply, divide, …)

Like all class types (except for String), it has no literals and no operators.

Class Type: TextView
(in android.widget)

- Stores a UI label
- Many attributes: text, layout, style, …
- API (getText, setText, setTypeFace, …)

Like all class types (except for String), it has no literals and no operators.

The Assignment Statement

An operator

name = value;

- Pre-declared and in-scope
- Its type can hold RHS
- Its content will be overwritten

A separator

- a Literal
- a Name, or
- an Expression