Using the following Java application as an illustration, let’s study Java’s lexicon – those words and symbols that make up a Java program.

```java
/* Compute automobile's fuel consumption using the formula: mpg = miles / gallons. */
import java.util.Scanner;
public class MPG {
    public static void main( String[] args ) {
        // declare data
        double miles;    // miles driven in car
        double gallons;  // number of gallons used
        double mpg;      // miles per gallon
        // input data
        Scanner in = new Scanner( System.in );
        System.out.println("Enter miles driven and gallons used:");
        miles = in.nextDouble( );
        gallons = in.nextDouble();
        // calculate mpg
        mpg = miles / gallons;
        // output results
        System.out.print( miles + " mi. / ");
        System.out.print( gallons + " gal." );
        System.out.println( " = " + mpg + " mpg" );
    }
}
```
Comments

Comments are messages to a human reader that are ignored by the computer. You, the programmer, use them to explain aspects of the program that may not be obvious to your fellow programmers.

Example
Comments that extend over more than one line can be enclosed in the delimiters /* and */.

```java
/* Compute automobile's fuel consumption using the formula: mpg = miles / gallons.
*/
```

Example
Comments that extend to the end of a line are preceded by the delimiter //.

```java
// declare data
double miles;   // miles driven in car
```

Keywords

Keywords are words defined by those persons who created the Java language. The meaning of a keyword cannot be altered by you, the programmer.

Examples

<table>
<thead>
<tr>
<th>public</th>
<th>class</th>
<th>static</th>
<th>void</th>
<th>double</th>
</tr>
</thead>
</table>

Predefined Identifiers

Predefined identifiers are words that were defined by those persons who implemented the Java language and its library of existing classes. A predefined identifier automatically has its preset meaning, which you, the programmer, can change if you wish to do so (it is recommended that you not do so).

Examples

<table>
<thead>
<tr>
<th>main</th>
<th>String</th>
<th>System</th>
<th>out</th>
<th>println</th>
</tr>
</thead>
</table>
User Identifiers

*User identifiers* are the words that you invent to name the various components of your own program.

<table>
<thead>
<tr>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPG</td>
</tr>
</tbody>
</table>

Symbols and Symbol Pairs

As a computer language, Java’s vocabulary extends to symbols that most languages consider to be punctuation. Sometimes they appear alone and sometimes they appear in pairs.

<table>
<thead>
<tr>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>{</td>
</tr>
</tbody>
</table>

The same symbol can represent different things according to the context in which it appears.

**Example**

In this statement the symbol `/` represents a division operator, telling the computer to divide the value of *miles* by that of *gallons*:

```java
mpg = miles / gallons;
```

Here the same symbol is paired with itself to represent the comment delimiter `//`:

```
// initialize data
```

And here it appears within a string literal telling the computer to display it in the program’s output:

```java
System.out.print( miles + " mi. / " );
```
Literals
A literal is an immutable program datum that stands for itself. The Java application **MPG** contains two types of literals:

- Numeric literals, which represent decimal numbers
  
<table>
<thead>
<tr>
<th>Examples</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>341</td>
<td>15.5</td>
</tr>
</tbody>
</table>

- String literals, which represent themselves. In Java, you must delimit a string literal with the double quotation mark (").
  
<table>
<thead>
<tr>
<th>Examples</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot; mi. / &quot;</td>
<td>&quot; gal.&quot;</td>
</tr>
<tr>
<td>&quot; = &quot;</td>
<td>&quot; mpg&quot;</td>
</tr>
</tbody>
</table>

White Space
White space in a Java program includes spaces, new lines and horizontal tabs. Java generally ignores white space so that you can space your program as you wish. You must, however, insert white space wherever you need to delimit two lexemes. Follow these guidelines:

- Separate word lexemes with at least one space so that they don’t run together.
- Don’t split a single lexeme (words, literals, symbols pairs) by white space.

<table>
<thead>
<tr>
<th>Examples</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>public class MPG {</td>
<td>OK.</td>
</tr>
<tr>
<td>public class MPG{}</td>
<td>OK. Line endings are white space.</td>
</tr>
<tr>
<td>public class MPG</td>
<td>OK. No white space needed between <strong>MPG</strong> and `{.</td>
</tr>
<tr>
<td>public class MPG</td>
<td><strong>Wrong!</strong> Missing white space separating <strong>public</strong> from <strong>class.</strong></td>
</tr>
</tbody>
</table>
### Examples

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>public class MPG</code></td>
<td>Wrong! Missing white space separating <code>class</code> from <code>MPG</code>.</td>
</tr>
<tr>
<td><code>public class M P G</code></td>
<td>Wrong! White space splits the user identifier <code>MPG</code> into three lexemes.</td>
</tr>
<tr>
<td><code>// declare data</code></td>
<td>OK. Recognized as a comment.</td>
</tr>
<tr>
<td><code>// declare data</code></td>
<td>Wrong! White space splits the comment delimiter so that it is no longer recognized as such.</td>
</tr>
<tr>
<td><code>miles = 341;</code></td>
<td>OK.</td>
</tr>
<tr>
<td><code>miles = 34 1;</code></td>
<td>Wrong! White space splits the literal <code>341</code> into two numeric literals.</td>
</tr>
</tbody>
</table>

### Letter Case

Java is case-sensitive, meaning that it treats upper-case letters as different from their lower-case equivalent. Java keywords are always spelled with lower-case letters.

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>public class MPG</code></td>
<td>OK.</td>
</tr>
<tr>
<td><code>Public class MPG</code></td>
<td>Wrong!</td>
</tr>
<tr>
<td><code>public Class MPG</code></td>
<td>Wrong!</td>
</tr>
</tbody>
</table>

Identifiers can be any letter case, but you must spell them consistently.

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>public static void Main ( String [] args )</code></td>
<td>Wrong! The predefined identifier <code>main</code> is all lower-case.</td>
</tr>
<tr>
<td><code>public static void main ( string [] args )</code></td>
<td>Wrong! The predefined identifier <code>String</code> starts with a capital S.</td>
</tr>
</tbody>
</table>
# Examples

```java
// declare data
double miles;    // miles driven in car
double gallons;  // number of gallons used

// input data
Miles = 341;
gallons = 15.5;

Wrong!
You must always spell your words the same. Here `miles` has a lower-case M in the first instance and a capital M in the second instance.
```

Besides upper and lower-case letters, you can use digits and the underscore character to spell an identifier but nothing else and the identifier cannot start with a digit.

## Examples

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Correct/Invalid</th>
</tr>
</thead>
<tbody>
<tr>
<td>double numberone;</td>
<td>Correct</td>
</tr>
<tr>
<td>double numberOne;</td>
<td>Correct</td>
</tr>
<tr>
<td>double NumberOne;</td>
<td>Correct</td>
</tr>
<tr>
<td>double NUMBER_ONE;</td>
<td>Correct</td>
</tr>
<tr>
<td>double NUMBERONE;</td>
<td>Correct</td>
</tr>
<tr>
<td>double number-one;</td>
<td>Wrong! Illegal character (−) in the identifier.</td>
</tr>
<tr>
<td>double number1;</td>
<td>Correct</td>
</tr>
<tr>
<td>double 1stNumber;</td>
<td>Wrong! Cannot start with a digit.</td>
</tr>
<tr>
<td>double Number One;</td>
<td>Wrong! Cannot split the identifier with a space.</td>
</tr>
</tbody>
</table>
### Exercises

The following problems refer to this Java application.

```java
/* Compute automobile's fuel consumption using the formula: mpg = miles / gallons. */
import java.util.Scanner;
public class MPG {
    public static void main( String[] args ) {
        // declare data
        double miles;    // miles driven in car
        double gallons;  // number of gallons used
        double mpg;      // miles per gallon
        // input data
        Scanner in = new Scanner( System.in );
        System.out.println("Enter miles driven and gallons used:");
        miles = in.nextDouble();
        gallons = in.nextDouble();
        // calculate mpg
        mpg = miles / gallons;
        // output results
        System.out.print( miles + " mi. / " );
        System.out.print( gallons + " gal." );
        System.out.println( " = " + mpg + " mpg" );
    }
}
```

1. Circle a multi-line comment within the application.
2. Circle a single-line comment within the application.
3. List five keywords used in the application.
4. List four predefined identifiers used in the application.
5. List four user identifiers used in the application.
6. Circle a symbol-pair lexeme within the application.

7. Circle a literal within the application.

8. Circle the keywords whose spelling is *syntactically* correct (whether or not they are *stilistically* correct).

<table>
<thead>
<tr>
<th>class</th>
<th>Class</th>
<th>CLASS</th>
<th>public</th>
</tr>
</thead>
<tbody>
<tr>
<td>Public</td>
<td>PUBLIC</td>
<td>void</td>
<td>Void</td>
</tr>
</tbody>
</table>

9. Circle the identifiers whose spelling is *syntactically* correct (whether or not they are *stilistically* correct).

<table>
<thead>
<tr>
<th>firstquartersales</th>
<th>FirstQuarterSales</th>
<th>first-quarter-sales</th>
</tr>
</thead>
<tbody>
<tr>
<td>first quarter sales</td>
<td>first_quarter_sales</td>
<td>FIRST_QUARTER_SALES</td>
</tr>
<tr>
<td>FIRSTQUARTERSALES</td>
<td>FIRST QUARTER SALES</td>
<td>FIRST-QUARTER-SALES</td>
</tr>
<tr>
<td>1stQuarterSales</td>
<td>salesForQuarter1</td>
<td></td>
</tr>
</tbody>
</table>

10. Circle each lexical error in this Java application.

    ```java
    public class Hello World{
        public static void main(String[ ]args){
            system.out.println("Welcome to Java Programming!");
        }
    }
    ```
Circle each lexical error in this Java application.

/* Calculate the height of a building by timing how long a ball takes to fall from the roof.*/

public class FindHeight{
    public static void main( String [] args )
    {
        // declare data
        double t; // number of seconds for fall
        double h; // height
        double g; // gravity
        // initialize data
        t=2.5; // seconds  G=9.81; // meters per seconds squared
        // calculate the height in feet then meters
        h=(G*t*t)/2; h*=3.281; // 1 m = 3.281 ft.
        System.out.println("A ball droops"+h+"feet in");
        System.out.println(t+" seconds");
    }
}