INTEGER DATA AND OPERATIONS

A variable that is one of the integer data types can hold only whole numbers. Four different built-in integer data types allow a variable to hold numbers in various ranges, as shown in this table:

<table>
<thead>
<tr>
<th>Java Integer Data Types</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
</tr>
<tr>
<td>int</td>
</tr>
<tr>
<td>long</td>
</tr>
<tr>
<td>short</td>
</tr>
<tr>
<td>byte</td>
</tr>
</tbody>
</table>

Integer literals are typed as a sequence of digits preceded by an optional sign (+ or –). The underscore (_) can be used as a group separator; that is, as Americans use commas (,) and Europeans use periods (.)

Example

```java
int a = 10;
int b = +10;
int c = -10;
int d = 1_024;
```

An integer literal is an `int` value unless you specify it to be `long` by using the suffix `L`.

Examples

```java
int a = 2_147_483_647; // OK.
int a = 2_147_483_648; // Wrong!
The literal 2_147_483_648 cannot be stored in 32 bits.
int a = 2_147_483_648L; // Wrong!
The 64-bit value 2_147_483_648L cannot be stored into the 32-bit int variable a.
```
Examples

long a = 2_147_483_648L;  OK.

You can specify an octal (base 8) value by using the prefix 0 or a hexadecimal (base 16) value by using the prefix 0x, where 0 is the decimal digit (not the English letter).

<table>
<thead>
<tr>
<th>Examples</th>
<th>Value shown in decimal</th>
</tr>
</thead>
</table>
| int a, b, c, d;  
a = 012;  
b = 0xA;  
c = 02000;  
d = 0x400; | a 10  
b 10  
c 1024  
d 1024 |

When both operands to the / are integers, the / performs integer division, meaning that it truncates any fractional part and yields an integer quotient.

Example

```
int a, b;  
a = 3 / 6;  
b = 7 / 2;
```

<table>
<thead>
<tr>
<th>Example</th>
<th>Value</th>
</tr>
</thead>
</table>
| double x;  
x = 3 / 6; | x 0.0 |

The data type of the destination variable doesn't affect the result because the truncation is part of the division operation.

Example

```
double x;  
x = 7.0 / 2.0;
```

<table>
<thead>
<tr>
<th>Example</th>
<th>Value</th>
</tr>
</thead>
</table>
| double x;  
x = 7.0 / 2.0; | x 3.5 |
If any of the arithmetic operators have one integer operand and one floating-point operand, the Java compiler converts the integer operand to floating-point and applies a floating-point operation.

**Example**

double x, y, z;
x = 7.0 / 2;
y = 7 / 2.0;
z = x * 2;

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>3.5</td>
</tr>
<tr>
<td>y</td>
<td>3.5</td>
</tr>
<tr>
<td>z</td>
<td>7.0</td>
</tr>
</tbody>
</table>

**Long Division**

In grade school you learned how to divide two integers using *long division* to yield a *quotient* and a *remainder*.

**Example**

3 friends want to split two 8-piece pizzas. Each person gets 5 pieces with 1 piece left over.

<table>
<thead>
<tr>
<th>divisor</th>
<th>dividend</th>
<th>quotient</th>
<th>remainder</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>16</td>
<td>5</td>
<td>1</td>
</tr>
</tbody>
</table>

In Java, the `/` and `%` operators team up to allow you to do long division of integers. `/` gives you the quotient; `%` gives you the remainder.

**Example**

How many yards and feet are equal to $x$ feet? For example, 17 feet equals $17/3 = 5$ yards and $17\%3 = 2$ feet.

```java
int yards = x / 3;
int feet = x % 3;
```
**Example**
How many pounds and ounces are equal to \( x \) ounces? For example, 100 ounces equals \( \frac{100}{16} = 6 \) pounds and \( 100 \mod 16 = 4 \) ounces.

```java
int pounds = x / 16;
int ounces = x % 16;
```

**Clock Arithmetic**

*Clock arithmetic* is a procedure for counting integers so that upon reaching a value the count starts over at zero. It is called “clock arithmetic” because of the way time is counted.

**Example**
Seconds are counted starting at zero and proceeding to 1, 2, 3, and so on until 59 seconds is reached. After 59, the number of seconds starts over at zero because 60 seconds is one minute.

The value that causes the count to start over is called the *modulus*; thus, clock arithmetic is also known as *modular arithmetic*.

**Example**
For counting seconds the modulus is 60.

To perform clock arithmetic in Java, you use the remainder operator (`%`) to divide the result by the modulus. If the modulus is \( n \), the numbers counted will never equal or exceed \( n \), but will range from 0 to \( n-1 \).

**Example**
This Java code prints 3, 0, 1, 2, 3, 0.

```java
int c = 2;
System.out.println( c = ( c + 1 ) % 4 );
System.out.println( c = ( c + 1 ) % 4 );
System.out.println( c = ( c + 1 ) % 4 );
System.out.println( c = ( c + 1 ) % 4 );
System.out.println( c = ( c + 1 ) % 4 );
System.out.println( c = ( c + 1 ) % 4 );
```
### Exercises

1. Declare `score` an `int` variable initialized to zero.
2. Declare `count` an `int` variable initialized to fifty million.
3. Declare `population` a `long` variable initialized to one-hundred billion.
4. Declare `salesPrice` a `short` variable initialized to thirty-two thousand.
5. Declare `score` a `byte` variable initialized to one hundred.
6. Declare `MAX_HEIGHT` an `int` constant identifier initialized to one thousand.
7. Declare `MAX_FREQUENCY` an `int` constant identifier initialized to 107.

Use octal literals for the following.

8. Declare an `int` variable initialized to 100₈.
9. Declare a `long` variable initialized to 1,000,000₈.

Use hexadecimal literals for the following.

10. Declare an `int` variable initialized to 100₁₆.
11. Declare a `long` variable initialized to 1,000,000₁₆.
12. Declare an `int` variable initialized to the binary number containing all ones.
13. Declare an `int` variable initialized to the binary number containing all zeroes.
14. Declare an `int` variable initialized to the binary number 0110 0110 …

For each of the following, give the final values of the `int` variables `q` and `r`.

15. \[ q = 100 / 12; \]
   \[ r = 100 \mod 12; \]
16. \[ q = 50 / 5; \]
   \[ r = 50 \mod 5; \]
17. \[ q = 50 / 11; \]
   \[ r = 50 \mod 11; \]
For each of the following, give the final value of the **int** variable `r`.

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>15.</td>
<td><code>r = 0 % 5;</code></td>
</tr>
<tr>
<td>16.</td>
<td><code>r = 1 % 5;</code></td>
</tr>
<tr>
<td>17.</td>
<td><code>r = 2 % 5;</code></td>
</tr>
<tr>
<td>18.</td>
<td><code>r = 3 % 5;</code></td>
</tr>
<tr>
<td>19.</td>
<td><code>r = 4 % 5;</code></td>
</tr>
<tr>
<td>20.</td>
<td><code>r = 5 % 5;</code></td>
</tr>
<tr>
<td>21.</td>
<td><code>r = 6 % 5;</code></td>
</tr>
</tbody>
</table>

For each of the following write the Java code to perform the calculation. Declare any variables needed. Use **int** variables and the operators discussed in this topic.

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>22.</td>
<td>Convert minutes to hours and minutes (e.g. 200 minutes is 3 hours and 20 minutes).</td>
</tr>
<tr>
<td>23.</td>
<td>Convert seconds to minutes and seconds (e.g. 150 seconds is 2 minutes and 30 seconds).</td>
</tr>
<tr>
<td>24.</td>
<td>Convert seconds to hours, minutes and seconds (e.g. 5,000 seconds is 1 hour, 23 minutes and 20 seconds).</td>
</tr>
</tbody>
</table>
| 25. | Given the hour o’clock on the west coast of the United States, determine the hour o’clock on the east coast.  

*Note:* There’s a 3-hour time difference. Your solution must work for any hour 1 through 12.
The following two problems require the use of a `DecimalFormat` object to format a number with leading 0s (e.g. to format the number 12 as 012). Refer to the topic [Using the Java API → java.text.DecimalFormat](#).

<table>
<thead>
<tr>
<th>Problem</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>26.</td>
<td>An <code>int</code> variable holds a 9-digit Social Security Number. Print the number in the typical 12-character format. For example, 123456789 must be printed as 123-45-6789.</td>
</tr>
<tr>
<td>27.</td>
<td>An <code>int</code> variable holds a 3-digit number. Split out its 3 digits and display it as shown to the right, which is the display for the original value of 234.</td>
</tr>
</tbody>
</table>