Datatypes, Variables, and Operations
Primitive Type Classification

PrimitiveType:
   NumericType
   boolean

NumericType:
   IntegralType
   FloatingPointType

IntegralType: one of
   byte short int long char

FloatingPointType: one of
   float double
## Numerical Data Types

<table>
<thead>
<tr>
<th>Name</th>
<th>Range</th>
<th>Storage Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>byte</td>
<td>$-2^7$ to $2^7 - 1$ (-128 to 127)</td>
<td>8-bit signed</td>
</tr>
<tr>
<td>short</td>
<td>$-2^{15}$ to $2^{15} - 1$ (-32768 to 32767)</td>
<td>16-bit signed</td>
</tr>
<tr>
<td>int</td>
<td>$-2^{31}$ to $2^{31} - 1$ (-2147483648 to 2147483647)</td>
<td>32-bit signed</td>
</tr>
<tr>
<td>long</td>
<td>$-2^{63}$ to $2^{63} - 1$ (i.e., -9223372036854775808 to 9223372036854775807)</td>
<td>64-bit signed</td>
</tr>
</tbody>
</table>
| float | Negative range: $-3.4028235E+38$ to $-1.4E-45$  
Positive range: $1.4E-45$ to $3.4028235E+38$ | 32-bit IEEE 754 |
| double | Negative range: $-1.7976931348623157E+308$ to $-4.9E-324$  
Positive range: $4.9E-324$ to $1.7976931348623157E+308$ | 64-bit IEEE 754 |
Declaring Variables

```java
int x; // Declare x to be an integer variable;

double radius; // Declare radius to be a double variable;

char a; // Declare a to be a character variable;
```
Identifiers

- An identifier is a sequence of characters that consist of letters, digits, underscores (_), and dollar signs ($).
- An identifier must start with a letter, an underscore (_), or a dollar sign ($). It cannot start with a digit.
  - An identifier cannot be a reserved word. (See Appendix A, “Java Keywords,” for a list of reserved words).
- An identifier can be of any length.
Assignment Statements

x = 1; // Assign 1 to x;

radius = 1.0; // Assign 1.0 to radius;

a = 'A'; // Assign 'A' to a;
Declaring and Initializing in One Step

- `int x = 1;`
- `double d = 1.4;`
Named Constants

```java
final datatype CONSTANTNAME = VALUE;

final double PI = 3.14159;
final int SIZE = 3;
```
Naming Conventions

- Choose meaningful and descriptive names.
- Variables and method names:
  - Use lowercase. If the name consists of several words, concatenate all in one, use lowercase for the first word, and capitalize the first letter of each subsequent word in the name. For example, the variables `radius` and `area`, and the method `computeArea`. 
Class names:
  – Capitalize the first letter of each word in the name. For example, the class name ComputeArea.

Constants:
  – Capitalize all letters in constants, and use underscores to connect words. For example, the constant PI and MAX_VALUE
# Numeric Operators

<table>
<thead>
<tr>
<th>Name</th>
<th>Meaning</th>
<th>Example</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>Addition</td>
<td>34 + 1</td>
<td>35</td>
</tr>
<tr>
<td>-</td>
<td>Subtraction</td>
<td>34.0 - 0.1</td>
<td>33.9</td>
</tr>
<tr>
<td>*</td>
<td>Multiplication</td>
<td>300 * 30</td>
<td>9000</td>
</tr>
<tr>
<td>/</td>
<td>Division</td>
<td>1.0 / 2.0</td>
<td>0.5</td>
</tr>
<tr>
<td>%</td>
<td>Remainder</td>
<td>20 % 3</td>
<td>2</td>
</tr>
</tbody>
</table>
Integer Division

5 / 2 yields an integer 2.

5.0 / 2 yields a double value 2.5

5 % 2 yields 1 (the remainder of the division)
Remainder Operator

Remainder is very useful in programming. For example, an even number % 2 is always 0 and an odd number % 2 is always 1. So you can use this property to determine whether a number is even or odd. Suppose today is Saturday and you and your friends are going to meet in 10 days. What day is in 10 days? You can find that day is Tuesday using the following expression:

$$ (6 + 10) \mod 7 = 2 $$

Saturday is the 6th day in a week

A week has 7 days

After 10 days

The 2nd day in a week is Tuesday
Floating Point Approximation

Calculations involving floating-point numbers are approximated because these numbers are not stored with complete accuracy. For example,

System.out.println(1.0 - 0.1 - 0.1 - 0.1 - 0.1 - 0.1 - 0.1);

displays 0.5000000000000001, not 0.5, and

System.out.println(1.0 - 0.9);

displays 0.09999999999999998, not 0.1. Integers are stored precisely. Therefore, calculations with integers yield a precise integer result.
Floating Point (cont’d)

- `double x1 = 0.3;`
- `double x2 = 0.1 + 0.1 + 0.1;`
- `System.out.println(x1 == x2); // evaluates to false`
- **Why?**
  - `1/10 = 1/16 + 1/32 + 1/256 + 1/512 + 1/4096 + 1/8192 + ...`
Floating Point cont’d

- double epsilon = 1e-15;
- double x1 = 0.3;
- double x2 = 0.1 + 0.1 + 0.1;
- System.out.println(Math.abs(x1 - x2) < epsilon) // evaluates to true
Exponent Operations

System.out.println(Math.pow(2, 3));  
// Displays 8.0  
System.out.println(Math.pow(4, 0.5));  
// Displays 2.0  
System.out.println(Math.pow(2.5, 2));  
// Displays 6.25  
System.out.println(Math.pow(2.5, -2));  
// Displays 0.16
Number Literals

A *literal* is a constant value that appears directly in the program.

```java
int i = 34;
long x = 1000000;
double d = 5.0;
int ssn = 342_44_9922;
byte b = 1000; // compile error
float f = 5.0f;
long l = 500000000000; // error
long l = 500000000000L;
long l = 500_000_000_000L; // _ seperator
```
Scientific Notation

1.23456e+2 is the same as 1.23456e2 and is equivalent to 123.456

1.23456e-2 is equivalent to 0.0123456.

E (or e) represents an exponent and it can be either in lowercase or uppercase.
## Shortcut Assignment Operators

<table>
<thead>
<tr>
<th>Operator</th>
<th>Example</th>
<th>Equivalent</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>+=</code></td>
<td><code>i += 8</code></td>
<td><code>i = i + 8</code></td>
</tr>
<tr>
<td><code>-=</code></td>
<td><code>f -= 8.0</code></td>
<td><code>f = f - 8.0</code></td>
</tr>
<tr>
<td><code>*=</code></td>
<td><code>i *= 8</code></td>
<td><code>i = i * 8</code></td>
</tr>
<tr>
<td><code>/=</code></td>
<td><code>i /= 8</code></td>
<td><code>i = i / 8</code></td>
</tr>
<tr>
<td><code>%=</code></td>
<td><code>i %= 8</code></td>
<td><code>i = i % 8</code></td>
</tr>
</tbody>
</table>