Computational Expression
Variables, Primitive Data Types, Expressions

Janyl Jumadinova

10-12 September, 2018
Variables

- **Variable** is a name for a memory’s location where a data value is stored.
Variables

- **Variable** is a name for a memory’s location where a data value is stored.
- **Variable Declaration** allows the compiler to reserve space in the main memory that is large enough for the specified type:
  
  ```java
  int count;
  ```
  
  Must give a value to the variable before using it in the main method.
**Variables**

- **Variable** is a name for a memory’s location where a data value is stored.
- **Variable Declaration** allows the compiler to reserve space in the main memory that is large enough for the specified type
  ```
  int count;
  ```
- **Variable Assignment** assigns a value to the variable
  ```
  count = 0;
  ```
Variables

- **Variable** is a name for a memory’s location where a data value is stored.
- **Variable Declaration** allows the compiler to reserve space in the main memory that is large enough for the specified type
  
  ```java
  int count;
  ```

- **Variable Assignment** assigns a value to the variable
  
  ```java
  count = 0;
  ```

- Must give a value to the variable before using it in the main method.
Java Identifiers

- reserved keywords (class, public, static, void)
- Java classes, methods, variables: words we chose or make up when writing a program
  System, println, main, args
Java Identifiers

- reserved keywords (class, public, static, void)
- Java classes, methods, variables: words we chose or make up when writing a program
  System, println, main, args

Identifier

a letter followed by zero or more letters (including $ and _) and digits
Identifiers
- An identifier may be any descriptive sequence of uppercase and lowercase letters, numbers, or the underscore and dollar-sign characters.
Identifiers

- An identifier may be any descriptive sequence of uppercase and lowercase letters, numbers, or the underscore and dollar-sign characters.
- Ex: Average, count, num1, $test, this_is_fine
Identifiers

- An identifier may be any descriptive sequence of uppercase and lowercase letters, numbers, or the underscore and dollar-sign characters.
- Ex: `Average`, `count`, `num1`, `$test`, `this_is_fine`
Identifiers

- Identifiers must start with a letter, a currency character ($), or a connecting character such as the underscore (_).

- Identifiers cannot start with a number.

- After the first character, identifiers can contain any combination of letters, currency characters, connecting characters, or numbers.

- There is no limit to the number of characters an identifier can contain.

- You can’t use a Java keyword as an identifier.

- Identifiers in Java are case-sensitive; foo and FOO are two different identifiers.
Identifiers

- Identifiers must start with a letter, a currency character ($), or a connecting character such as the underscore (_).
- Identifiers cannot start with a number.
Identifiers

- Identifiers must start with a letter, a currency character ($), or a connecting character such as the underscore (_).
- Identifiers cannot start with a number.
- After the first character, identifiers can contain any combination of letters, currency characters, connecting characters, or numbers.
Identifiers

- Identifiers must start with a letter, a currency character ($), or a connecting character such as the underscore (_).
- Identifiers cannot start with a number.
- After the first character, identifiers can contain any combination of letters, currency characters, connecting characters, or numbers.
- There is no limit to the number of characters an identifier can contain.
Identifiers

- Identifiers must start with a letter, a currency character ($), or a connecting character such as the underscore (_).
- Identifiers cannot start with a number.
- After the first character, identifiers can contain any combination of letters, currency characters, connecting characters, or numbers.
- There is no limit to the number of characters an identifier can contain.
- You can’t use a Java keyword as an identifier.
Identifiers

- Identifiers must start with a letter, a currency character ($), or a connecting character such as the underscore (_).
- Identifiers cannot start with a number.
- After the first character, identifiers can contain any combination of letters, currency characters, connecting characters, or numbers.
- There is no limit to the number of characters an identifier can contain.
- You can’t use a Java keyword as an identifier.
- Identifiers in Java are case-sensitive; foo and FOO are two different identifiers.
Java keywords

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

A constant value in Java is created by using a literal representation of it.

- 100 (integer literal)
- 98.6 (float literal)
- ‘X’ (character literal)
- ‘“This is a test”’ (String literal)
Constants

- Constants hold the same value during their existence.
- Can use a keyword `final` before the type and name of the variable:
  - always contains the same value.
- `final int MAX_BUDGET = 1000`
Data Types

- Data stored in memory is a string of bits (0 or 1)
Data Types

- Data stored in memory is a string of bits (0 or 1)
- How the computer interprets the string of bits depends on the context.
- In Java, we must make the context explicit by specifying the type of the data.
Data Types

- Java has two categories of data:
  
  - primitive data (e.g., number, character)
  - object data (programmer created types)

- There are 8 primitive data types: byte, short, int, long, float, double, char, boolean

- Primitive data are only single values; they have no special capabilities.
Primitive Data Types

- integers: `byte`, `short`, `int`, `long`
- floating point: `float`, `double`
- characters: `char`
- booleans: `boolean`
## Common Primitive Data Types

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
<th>Example of Literals</th>
</tr>
</thead>
<tbody>
<tr>
<td>int</td>
<td>integers (whole numbers)</td>
<td>42, 60634, -8</td>
</tr>
<tr>
<td>double</td>
<td>real numbers</td>
<td>0.039, -10.2</td>
</tr>
<tr>
<td>char</td>
<td>single characters</td>
<td>'a', 'B', '&amp;', '6'</td>
</tr>
<tr>
<td>boolean</td>
<td>logical values</td>
<td>true, false</td>
</tr>
</tbody>
</table>
## Range of Values

<table>
<thead>
<tr>
<th>Type</th>
<th>Storage</th>
<th>Range of Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>int</td>
<td>32 bits</td>
<td>-2,147,483,648 to 2,147,483,647</td>
</tr>
<tr>
<td>double</td>
<td>64 bits</td>
<td>$\pm 10^{-45}$ to $\pm 10^{38}$</td>
</tr>
<tr>
<td>char</td>
<td>16 bits = 2 bytes</td>
<td>0 to $2^{16}$ or \u0000 to \uFFFF</td>
</tr>
<tr>
<td>boolean</td>
<td>1 bit</td>
<td>NA</td>
</tr>
</tbody>
</table>
Expression

Expression is a combination of one or more operators (+, −, %, ...) and operands (literals, constants, variables, ...)

Operators are evaluated in an expression according to the rules of precedence.
Order of Precedence

- Operators are evaluated in an expression according to the rules of precedence.
- Operators within ( ) are evaluated first.
- *, /, % evaluated next (L to R).
- +, - evaluated last (L to R).
The Scanner class in the java.util package is a simple text scanner which can parse primitive types and strings. We can use the Scanner class to get the input from the terminal. We must create an instance of the Scanner as:

`Scanner name = new Scanner (System.in)`

where `name` is the name you choose for your instance of the Scanner.
Scanner Methods

- `next()` : get the next word (token) as a String
- `nextLine()` : get a line of input as a String
- `nextInt()` : get an integer
- `nextDouble()` : get a double value