Variables

Objectives

- Understand variables and parameters.
- Know the difference between how value and reference types behave.
- Understand scope rules.
Topics

- Fields, Parameters, and Local Variables.
- Variable allocation and deallocation.
- Value and Reference Types.
- Scope rules.

Fields, Local Variables, and Parameters

- Fields are allocated and initialized when objects are created. They last the entire life of the object.
- Local Variables are allocated each time a method is executed and deallocated when the method returns.
- Parameters are a special kind of local variable that are initialized with the values passed with a message.
**Scope Rules**

- Scope rules determine where the name of a variable may be used to access its contents.
- Fields are in scope from the point where they are declared to the end of the class definition.
  - They are also in scope in subclasses subject to their accessibility specification (public, protected, etc.)
- Local variables (declared inside a method) are in scope from the point they are declared to the end of the innermost block in which they are declared.
- Parameters are in scope throughout the body of the method where they are declared.
- It is illegal to have more than one local variable with the same name in scope at the same time, but it is legal to have a local variable and field with the same name.

**Method Execution**

- When a method is invoked by sending a message:
  1. Parameters and local variables are allocated (memory is reserved).
  2. Parameters are initialized with argument values sent with the message.
     - The values are matched with parameters positionally. The first value is assigned to the first parameter, the second value is assigned to the second parameter, etc.
- Initializers (if specified) are applied to local variables when the line of code where they are declared is executed.
- Parameters and local variables are deallocated when the method returns.
Example: What does this do?

```java
class A {
    private int x = 5;
    private int y = 20;
    public int m1(int x, int y) {
        x = x - y;
        return this.x - x;
    }
    public void m2(int i) {
        x = x + i;
    }
    public int m3() {
        if (y > 0) {
            int x = 100;
            y = x - y;
        }
        return x;
    }
    public void m4() {
        System.out.println(x + "," + y);
    }
}

class B {
    public static void main(String[] args) {
        A a1 = new A();
        A a2 = new A();
        int x = 7;
        int y = 10;
        x = a1.m1(y, x);
        a1.m2(y);
        System.out.println(x);
        x = a1.m3();
        System.out.println(x);
        a1.m4();
        a2.m4();
        a1.m2(a2.m3());
    }
}
```

Primitive Data Types

<table>
<thead>
<tr>
<th>Type</th>
<th>Bits</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>boolean</td>
<td>-</td>
<td>true or false</td>
</tr>
<tr>
<td>byte</td>
<td>8</td>
<td>-128 to +127</td>
</tr>
<tr>
<td>short</td>
<td>16</td>
<td>-32,768 to +32,767</td>
</tr>
<tr>
<td>char</td>
<td>16</td>
<td>0 to 65,535</td>
</tr>
<tr>
<td>int</td>
<td>32</td>
<td>± 2.1 billion (approximate)</td>
</tr>
<tr>
<td>long</td>
<td>64</td>
<td>± 9.2e18 (approx.)</td>
</tr>
<tr>
<td>float</td>
<td>32</td>
<td>± 3.4e38 (~7 digit precision)</td>
</tr>
<tr>
<td>double</td>
<td>64</td>
<td>± 1.8e308 (~10 digit precision)</td>
</tr>
</tbody>
</table>
Value and Reference Types

- Primitive types are value types.
  - Variables of value types hold values:
    
    ```java
    int n = 5;       n 5
    ```

- Other types (e.g. class and interface types) are reference types.
  - Variables of reference types hold references to (memory addresses of) values.
    
    ```java
    String s = “hello”;  s  hello
    ```

Value and Reference Types (2)

```java
int n1 = 0, n2 = 5;  n1 0 n2 5
```

```java
n1 = n2;  n1 5 n2 5
```

```java
String s1 = “hello”, s2 = “goodbye”;  s1  hello s2  goodbye
```

```java
s1 = s2;  s1  hello s2  goodbye
```
Example: Swapping (1)

```java
public class Swapper {
    public void swap(int x, int y) {
        int tmp = x;
        x = y;
        y = tmp;
    }
}

public class Main {
    public static void main(String[] args) {
        int x = 1;
        int y = 2;
        Swapper s = new Swapper();
        s.swap(x, y);
        System.out.println(x);
        System.out.println(y);
    }
}
```

Example: Swapping (2)

```java
public class Num {
    private int n;
    public Num(int n) {
        this.n = n;
    }
    public int getN() {
        return n;
    }
    public void setN(int n) {
        this.n = n;
    }
}

public class Swapper {
    public void swap(Num x, Num y) {
        Num tmp = x;
        x = y;
        y = tmp;
    }
}

public class Main {
    public static void main(String[] args) {
        Num x = new Num(1);
        Num y = new Num(2);
        Swapper s = new Swapper();
        s.swap(x, y);
        System.out.println(x.getN());
        System.out.println(y.getN());
    }
}
```
public class Num {
    private int n;
    public Num(int n) {
        this.n = n;
    }
    public int getN() {
        return n;
    }
    public void setN(int n) {
        this.n = n;
    }
}

public class Main {
    public static void main(String[] args) {
        Num x = new Num(1);
        Num y = new Num(2);
        Swapper s = new Swapper();
        s.swap(x, y);
        System.out.println(x.getN());
        System.out.println(y.getN());
    }
}

public class Swapper {
    public void swap(Num x, Num y) {
        int tmp = x.getN();
        x.setN(y.getN());
        y.setN(tmp);
    }
}