Array

An **array** is a structure that holds multiple values of the same type. The length of an array is established when the array is created. After creation, an array is a **fixed-length** structure. Array identifier is actually a reference to a true object that holds the references to the other objects.

```
double gears[] = new double[5];

gears[0] = 4.624;
gears[1] = 3.231;
gears[2] = 2.893;
gears[3] = 1.052;
gears[4] = 0.962;
```

Sometimes it is convenient to initialize an array immediately during its declaration.

```
double gears[] = {4.624, 3.231, 2.893, 1.052, 0.962};
```

Note: Although the **new** operator is not presented, the array is allocated dynamically (compiler does it for us).

Array (cont.)

The index of array elements start from 0.

When an array contains n elements, the elements have indexes from 0 to (n-1).

Special attribute `length` contains the number of elements in the array.

When for example

```
int[] a = new int[10];
int n = a.length;
```

The following form of a declaration of an array are possible, they are both equivalent:

```
int[] a;
int a[];
```

Array (cont.)

When we declare a variable such as

```
int[] a = new int[10];
```

it is a **reference** to an array. So after assignment

```
int[] b = a;
```

both a and b point to the same array and when we change a value of a[0], the value of b[0] is also changed.

Multidimentional Array

Multidimentional array is in fact **one-dimentional** array containing arrays as its elements.

```
final byte EMPTY = 0;
final byte CIRCLE = 1;
final byte CROSS = 2;
byte board[][] = {
   {EMPTY, CIRCLE, CROSS},
   {CIRCLE, EMPTY, CROSS},
   {EMPTY, CIRCLE, EMPTY}
};
for (int i = 0; i < board.length; i++)
   for (int j = 0; j < board[i].length; j++)
      System.out.println("board[" + i + "][" + j + "] = " +
         board[i][j]);
```

Note: The `length` is not a method. The `length` is a property provided by the Java platform for all arrays.

Manipulating Arrays

The `java.lang.System.arraycopy()` method provides efficient copy of data from one array into another.

```
char from[] = {'a', 'b', 'c', 'd', 'e', 'f'};
char to[] = new char[3];
System.arraycopy(from, 2, to, 0, to.length);
```

Note: Destination array must be allocated before `arraycopy()` is called and must be large enough to contain the data being copied.

Method Definition

A definition of a **method** begins with a **header** that if followed by a **body** of the method.

The header has the following format:

```
type name ( args )
```

where

- `name` is the name of the method
- `type` is a return type of the method
- `args` is a comma separated list of arguments, it may be empty

Each argument is of the form `type name`.

The **body** of a method is a block (i.e., statements enclosed between { and }).

Method Definition (cont.)

The return type may be **void** if the method does not return a value. When the return type is not **void** the method must return a value using the command

```
return expr;
```

An example:

```
int gcd(int a, int b)
{
    while (b != 0) {
        int c = a % b;
        a = b;
        b = c;
    }
    return a;
}
```

If the return type is **void** the following form of the return statement can be used:

```
return;
```

Method Invocation

A method is called using its name that is followed by a comma separated list of parameters between paranthesis.

Any expression can be used as a parameter.

All parameters are evaluated and assigned to the arguments of the method.

A method invocation can be used in expression. The value of the expression corresponding to the method invocation is the return value returned by the method.

An example:

```
int x, y;
x = gcd(24, 18);
y = gcd(x + 1, 36);
```