2. Java Variables and Arrays

Basic Java Constructs

- Everything is related to classes (objects).

  • Data declaration

    ```java
    int i;  // declare i to be an int
    Test t;  // declare t to be an object of type Test
    i = 0;  // set the int i equal to 0
    t = new Test();  // initialize t and allocate space using a constructor
    ```

  • Flow of control

    Traditional, but with object-oriented syntax for function calls and member functions.

    Where control starts in the Applet is important.

  • Executable statements

    Similar to C/C++ `while`, `for`, `if/else`, `switch`, etc.
Categories of Variables:

- **Instance variable**
  - created when object is instantiated
  - automatic duration unless tagged as "static"

- **Local variable**
  - method formal parameters, method vars, block vars
  - automatic duration related to block/context execution
Variable Duration (Lifetime):

- **Automatic**
  - variables exist when the context is active
  - values do not persist

- **Static**
  - variables exist once class is loaded into memory
  - values persist until explicit programmer deconstruction
Example of static variable:

```java
// Employee.java
public class Employee {
    public String firstName;
    public String lastName;
    public static int count; // # of objects instantiated

    public Employee( String fName, String lName ) {
        firstName = fName;
        lastName = lName;

        ++count; // increment static count of employees
    }
}
```

Notes:
- Static variables are shared among all objects of the same class
- Can be accessed by "class_name.static_variable_name"
Scope Rules

- Scope: where a variable can be used (referenced)
- **class scope**: can be referenced throughout class
- **block scope**: can be referenced only within local block

<table>
<thead>
<tr>
<th>Variable</th>
<th>Scope</th>
</tr>
</thead>
<tbody>
<tr>
<td>instance variables &amp; class method</td>
<td>class scope</td>
</tr>
<tr>
<td>local vars, block vars, formal parameters</td>
<td>block scope</td>
</tr>
</tbody>
</table>

Note: static methods cannot access non-static variables directly. Use

```
object_name.variable_name
```
Example:

// Testing static method
import java.io.*;

public class Test {
    public int t;
    public static int c;
    private int pt;
    private static int pc;

    public static void main ( String args[] ) {
        Test s;
        s = new Test();
        s.t = 5;
        Test ps;
        ps = new Test();
        ps.pt = 15;
        c = 10;
        pc = 20;
        System.out.println("Value of t = "+ s.t);
        System.out.println("Value of c = "+ c);
        System.out.println("Value of pt = "+ ps.pt);
        System.out.println("Value of pc = "+ pc);
    }
}
Methods
- Similar to functions of C/C++

Main Methods

Applet Methods

Method Overloading
- Methods with different signatures (data type of parameter list) can have the same name
- Signature which matches invocation is the method selected at run time for execution
- Why? consider "String.valueOf()" which can convert parameter of any type to string

Constructor Methods
- A method with the same name as the class
- Called automatically to Initializes the instance variables of an object
Parameter Passing

<table>
<thead>
<tr>
<th>primitive-data-type variables</th>
<th>call-by-value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Objects</td>
<td>call-by-reference</td>
</tr>
</tbody>
</table>

- Primitive data types (boolean, char, byte, short, int, long, float, double) are ALWAYS passed by **value**

- Objects are ALWAYS passed by **reference**

- Java does not allow programmer to choose between call-by-reference or call-by-value.

**value**: copy is made

**reference**: pointer to original location is passed
Java Arrays

- Arrays and strings are objects in Java, but can not be extended
- Indexes start at 0
- Components can be primitive types or objects (more precisely, references to objects), including arrays
- Any expression which evaluates to an integer can be used as the array subscript
- Arrays can be re-allocated to change the size; they are static once created

Declaring and Allocating Arrays

- Arrays must be declared and allocated

```
int myarray[ ] = new int[ 12 ];    //or
int[] myarray = new int[ 12 ];
```

// or

```
int array[ ];    //define array name
myarray = new int[ 12 ];      //allocate slots
```
• Elements are automatically initialized to zero for primitive type data (false for boolean), to null otherwise.

• Every array knows its own length

• Allocated size can be dynamically determined with an expression:

```
char chararray[ ];
chararray = new char[size*6 ];
```
Dynamic Arrays

- Re-allocation possible

```java
char myarray[ ] = new char[255];
char tmparray[ ] = new char[255];

for (int i=0; i < myarray.length; i++)
    tmparray[i] = myarray[i];

myarray = new char[255*2];
for (int i=0; i < 255; i++)
    myarray[i] = tmparray[i];
```
Passing arrays to Methods

- Java treats arrays as **objects**. When the actual parameter is an array, it is passed to the method by **reference**.
  (No need to pass the size of the array as a parameter)

- However, in the "signature" of a method, an array parameter has to be explicitly specified

```java
int c[ ] = new int[ 12 ];

modifyArray( c );

public void modifyArray( int b[ ] ) {

}
```
When elements in an array are primitive data types, the elements can be passed by value.

int c[ ] = new int[ 12 ];
...
modifyElement( c[4] );
...
public void modifyElement( int elem ) {
    ...
}

Multiple-Subscripted Arrays

- Java arrays support *arrays of arrays*

Declaration:

```java
int b[][ ];
b = new int[3][4];
```

b.length = ?  
b[3].length = ?

- The number of elements in a row can be different from other rows

Declaration:

```java
int b[][ ];
b = new int[3][];
b[0] = new int[5];
b[1] = new int[3];
```
Or,

```java
int b[][] = new int[3][];
b[0] = new int[5];
b[1] = new int[3];
```

However, the following is not an alternative

```java
int b[];
b = new int[3];
b[0] = new int[5];
b[1] = new int[3];
```
What is the output of the fourth line?

```java
int c[][ ] = new int[12][ ];
c[3][ ] = new int[6];
modifyArray( c[3][ ]); System.out.println( c[3][4] ); //what is the output?

public void modifyArray( int b[] ) {
    for (int j = 0; j < b.length; j++)
        b[j] += 5;
}
```

- To pass a row of a double-subscripted array to a method taking a single subscripted array, pass "array-name[row]"

- Java compiler does not check if array element references are valid
• Arrays of objects must have each slot allocated with the **new** command

```java
int b[][];
b = new int[4][6];
```

```java
JButton board[];
board = new JButton[4][6];
```

```java
for (int i = 0; i < board.length; i++)
    for (int j = 0; j < board[i].length; j++)
        board[i][j] = new JButton();
```