Language Basics

Variables

- Variables are containers / refers to addresses of memory for storing data values.
- In the Java programming language, the terms "field" and "variable" are both used.

The Java programming language defines the following kinds of variables:

- [ ] **Instance Variables (Non-Static Fields)**
  - fields declared without the `static` keyword.
  - Non-static fields are also known as *instance variables* because their values are unique to each *instance* of a class (to each object, in other words).
  - The current speed of one bicycle is independent from the `currentSpeed` of another.
- [ ] **Class Variables (Static Fields)**
  - A *class variable* is any field declared with the `static` modifier; this tells the compiler that there is exactly one copy of this variable in existence, regardless of how many times the class has been instantiated.
  - A field defining the number of gears for a particular kind of bicycle could be marked as `static` since conceptually the same number of gears will apply to all instances. The code `static int numGears = 6;` would create such a static field.
  - Additionally, the keyword `final` could be added to indicate that the number of gears will never change.
- [ ] **Local Variables**
  - A method will often store its temporary state in *local variables*.
  - The syntax for declaring a local variable is similar to declaring a field (for example, `int count = 0;`).
  - As such, local variables are only visible to the methods in which they are declared; they are not accessible from the rest of the class.
- [ ] **Parameters**
  - Recall that the signature for the `main` method is `public static void main(String[] args)`. Here, the `args` variable is the parameter to this method.
  - The important thing to remember is that parameters are always classified as "variables" not "fields".

Naming Convention For Variables

- Variable names are case-sensitive.
- A variable's name can be any legal identifier — an unlimited-length sequence of Unicode letters and digits, beginning with a letter, the dollar sign "`$`", or the underscore character "`_`".
- Also keep in mind that the name you choose must not be a keyword or reserved word.
- **The convention, however, is to always begin your variable names with a letter.**
- **If the name you choose consists of only one word, spell that word in all lowercase letters.**
- **If it consists of more than one word, capitalize the first letter of each subsequent word.**
- **By convention, the underscore character is never used elsewhere.**

Primitive DataType

- The Java programming language is statically-typed, which means that all variables must first be declared before they can be used.
- The Java programming language is statically-typed, which means that all variables must first be declared before they can be used
- Primitive values do not share state with other primitive values.
- **The eight primitive data types supported by the Java programming language are:**
    - **byte:**
      The `byte` data type is an 8-bit signed two's complement integer.
      (-128 - 127].
      The `byte` data type can be useful for saving memory in large arrays, where the memory savings actually matters.
    - **short**:
      The `short` data type is a 16-bit signed two's complement integer.
      (-32,768 - 32,767].
    - **int**:
      By default, the `int` data type is a 32-bit signed two's complement integer.
      (-2^{31} - 2^{31}-1].
    - **long**: The `long` data type is a 64-bit two's complement integer.
      (-2^{63} - 2^{63}-1].
    - **float**: The `float` data type is a single-precision 32-bit IEEE 754 floating point.
    - **double**: The `double` data type is a double-precision 64-bit IEEE 754 floating point.
    - **boolean**: The `boolean` data type has only two possible values: `true` and `false`.
    - **char**: The `char` data type is a single 16-bit Unicode character. It has a minimum value of `'\u0000'` (or 0) and a maximum value of `'\uffff'` (or 65,535 inclusive).

Default Values

Data Type	Default Value (for fields)
byte	0
short	0
int	0
long	0L
float	0.0f
double	0.0d
char	'\u0000'
String (or any object)	null
boolean	false

Literals

A *literal* is the source code representation of a fixed value;

```
boolean result = true;
char capitalC = 'C';
byte b = 100;
short s = 10000;
int i = 100000;
```

Integer Literals

An integer literal is of type `long` if it ends with the letter `L` or `l`; otherwise it is of type `int`. It is recommended that you use the upper case letter `L` because the lowercase letter `l` is hard to distinguish from the digit `1`.

```
// The number 26, in decimal
int decVal = 26;
//  The number 26, in hexadecimal
int hexVal = 0x1a;
// The number 26, in binary
int binVal = 0b11010;
```

Floating-Point Literals

A floating-point literal is of type `float` if it ends with the letter `F` or `f`; otherwise its type is `double` and it can optionally end with the letter `D` or `d`.

```
// same value as d1, but in scientific notation
double d2 = 1.234e2;
float f1  = 123.4f;
```

Character and String Literals

Literals of types `char` and `String` may contain any Unicode (UTF-16) characters.
you can use a "Unicode escape" such as `'\u0108'` (capital C with circumflex), or `"S\u00ED Se\u00F1or"` (Sí Señor in Spanish).
Always use 'single quotes' for `char` literals and "double quotes" for `String` literals.

The Java programming language also supports a few special escape sequences for `char` and `String` literals: `\b` (backspace), `\t` (tab), `\n` (line feed), `\f` (form feed), `\r` (carriage return), `\"` (double quote), `\'` (single quote), and `\\` (backslash).

There's also a special `null` literal that can be used as a value for any reference type. `null` may be assigned to any variable, except variables of primitive types.

Using Underscore Characters in Numeric Literals

This feature enables you, for example. to separate groups of digits in numeric literals, which can improve the readability of your code.

the following example shows other ways you can use the underscore in numeric literals:

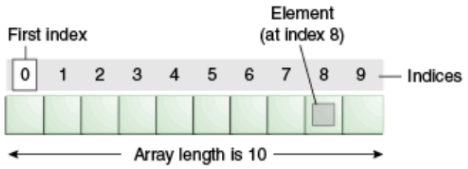
```
long creditCardNumber = 1234_5678_9012_3456L;
long socialSecurityNumber = 999_99_9999L;
float pi =  3.14_15F;
long hexBytes = 0xFF_EC_DE_5E;
long hexWords = 0xCAFE_BABE;
long maxLong = 0x7fff_ffff_ffff_ffffL;
byte nybbles = 0b0010_0101;
long bytes = 0b11010010_01101001_10010100_10010010;
```

You can place underscores only between digits; you cannot place underscores in the following places:

- At the beginning or end of a number
- Adjacent to a decimal point in a floating point literal
- Prior to an `F` or `L` suffix
- In positions where a string of digits is expected

Arrays

- An *array* is a container object that holds a fixed number of values of a single type.
- The length of an array is established when the array is created.



An array of 10 elements.

- Each item in an array is called an *element*, and each element is accessed by its numerical *index*.
- Index begins from 0.

Declaring a Variable to Refer to an Array

The preceding program declares an array (named `anArray`) with the following line of code:

```
// declares an array of integers
    int[] anArray;
```

Creating, Initializing, and Accessing an Array

One way to create an array is with the `new` operator

Int[] anArray; //just to let compiler know anArray will hold integer elements in array
// create an array of integers
anArray = new int[10];

The next few lines assign values to each element of the array:

```
anArray[0] = 100; // initialize first element
anArray[1] = 200; // initialize second element
    anArray[2] = 300; // and so forth
```

Each array element is accessed by its numerical index:

```
System.out.println("Element 1 at index 0: " + anArray[0]);
System.out.println("Element 2 at index 1: " + anArray[1]);
    System.out.println("Element 3 at index 2: " + anArray[2]);
```

Alternatively, you can use the shortcut syntax to create and initialize an array:

```
int[] anArray = {
    100, 200, 300,
    400, 500, 600,
    700, 800, 900, 1000
    };
```

2-D Arrays

You can also declare an array of arrays (also known as a *multidimensional* array) by using two or more sets of brackets, such as `String[][] names`.

In the Java programming language, a multidimensional array is an array whose components are themselves arrays.

A consequence of this is that the rows are allowed to vary in length, as shown in the following `MultiDimArrayDemo` program:

```
class MultiDimArrayDemo {
   public static void main(String[] args) {
      String[][] names = {
         {"Mr. ", "Mrs. ", "Ms. "},
         {"Smith", "Jones"}
      };
      // Mr. Smith
      System.out.println(names[0][0] + names[1][0]);
      // Ms. Jones
      System.out.println(names[0][2] + names[1][1]);
   }
}
```

Finally, you can use the built-in `length` property to determine the size of any array. The following code prints the array's size to standard output:

```
System.out.println(anArray.length);
```

Copying Arrays

The `System` class has an `arraycopy` method that you can use to efficiently copy data from one array into another:

```
public static void arraycopy(Object src, int srcPos,
                             Object dest, int destPos, int length)
```

```
class ArrayCopyDemo {
    public static void main(String[] args) {
        String[] copyFrom = {
            "Affogato", "Americano", "Cappuccino", "Corretto", "Cortado",
            "Doppio", "Espresso", "Frappuccino", "Freddo", "Lungo",
"Macchiato",
            "Marocchino", "Ristretto" };

        String[] copyTo = new String[7];
        System.arraycopy(copyFrom, 2, copyTo, 0, 7);
        for (String coffee : copyTo) {
            System.out.print(coffee + " ");
        }
    }
}
```

Some other useful operations provided by methods in the `java.util.Arrays` class are:

- Searching an array for a specific value to get the index at which it is placed (the `binarySearch(byte[] a, byte key)` method).
- Comparing two arrays to determine if they are equal or not (the `equals(byte[] a, byte[] a2)` method).
- Filling an array to place a specific value at each index (the `fill(boolean[] a, int fromIndex, int toIndex, boolean val)` method).
- Sorting an array into ascending order. (`parallelSort(byte[] a)` | `sort(byte[] a)`
- See Aggregate Operations for more information about streams.
- Converting an array to a string. The `toString` method converts each element of the array to a string, separates them with commas, then surrounds them with brackets. For example, the following statement converts the `copyTo` array to a string and prints it:
  `System.out.println(java.util.Arrays.toString(copyTo));`

https://docs.oracle.com/javase/8/docs/api/java/util/Arrays.html

Questions and Exercises: Variables

Questions

1. The term "instance variable" is another name for ___. NonStatic
2. The term "class variable" is another name for ___. Static
3. A local variable stores temporary state; it is declared inside a ___. method
4. A variable declared within the opening and closing parenthesis of a method signature is called a ____. Parameters
5. What are the eight primitive data types supported by the Java programming language?
   Int, long, char, double, float, boolean, byte, short.
6. Character strings are represented by the class ___. Strings
7. An ___ is a container object that holds a fixed number of values of a single type. Arrays

Exercises

1. Create a small program that defines some fields. Try creating some illegal field names and see what kind of error the compiler produces. Use the naming rules and conventions as a guide.
2. In the program you created in Exercise 1, try leaving the fields uninitialized and print out their values. Try the same with a local variable and see what kind of compiler errors you can produce. Becoming familiar with common compiler errors will make it easier to recognize bugs in your code.