

Data Types & Variables

Data Types

Java defines eight simple types:

- 1) byte – 8-bit integer type
- 2) short – 16-bit integer type
- 3) int – 32-bit integer type
- 4) long – 64-bit integer type
- 5) float – 32-bit floating-point type
- 6) double – 64-bit floating-point type
- 7) char – symbols in a character set
- 8) boolean – logical values true and false

Data Types

byte: 8-bit integer type.

Range: -128 to 127.

Example: `byte b = -15;`

Usage: particularly when working with data streams.

short: 16-bit integer type.

Range: -32768 to 32767.

Example: `short c = 1000;`

Usage: probably the least used simple type.

Data Types

`int`: 32-bit integer type.

Range: -2147483648 to 2147483647.

Example: `int b = -50000;` Usage:

- 1) Most common integer type.
- 2) Typically used to control loops and to index arrays.
- 3) Expressions involving the `byte`, `short` and `int` values are promoted to `int` before calculation.

Data Types

long: 64-bit integer type.

Range: -9223372036854775808 to 9223372036854775807.

Example: long l =
1000000000000000000;

Usage: useful when int type is not large enough to hold the desired value

float: 32-bit floating-point number.

Range: 1.4e-045 to 3.4e+038.

Example: float f = 1.5;

Usage:

- 1) fractional part is needed
- 2) large degree of precision is not required

Data Types

double: 64-bit floating-point number.

Range: 4.9e-324 to 1.8e+308. Example:

double pi = 3.1416; Usage:

- 1) accuracy over many iterative calculations
- 2) manipulation of large-valued numbers

Data Types

char: 16-bit data type used to store characters.

Range: 0 to 65536. Example: `char c = 'a';` Usage:

1) Represents both ASCII and Unicode character sets;

Unicode defines a character set with characters found in (almost) all human languages.

2) Not the same as in C/C++ where char is 8-bit and represents ASCII only

Data Types

`boolean`: Two-valued type of logical values.

Range: values `true` and `false`. Example: `boolean b =`

`(1<2);` Usage:

1) returned by relational operators, such as `1<2`

2) required by branching expressions such as `if` or `for`

Variables

Variables

declaration - how to assign a type to a variable

initialization - how to give an initial value to a variable

scope - how the variable is visible to other parts of the program

lifetime - how the variable is created, used and destroyed

type conversion - how Java handles automatic type conversion

type casting - how the type of a variable can be narrowed down

Variables

- Java uses variables to store data.
- To allocate memory space for a variable JVM requires:
 - 1) to specify the data type of the variable
 - 2) to associate an identifier with the variable
 - 3) optionally, the variable may be assigned an initial valueAll done as part of variable declaration.

Basic Variable Declaration

datatype identifier [=value];

- datatype must be
 - A simple datatype
 - User defined datatype (class type)
- Identifier is a recognizable name confirm to identifier rules
- Value is an optional initial value.

Identifier Rules

- If the name contains multiple words, start it with the lowercase letter followed by an uppercase letter such as `firstName`, `lastName`.
- It should not start with the special characters like `&` (ampersand), `$` (dollar), `_` (underscore).
- Avoid using one-character variables such as `x`, `y`, `z`.

Variable Declaration

We can declare several variables at the same time:

```
type identifier [=value][, identifier [=value] ...];
```

Examples:

```
int a, b, c;
```

```
int d = 3, e, f = 5;
```

```
byte g = 22;
```

```
double pi = 3.14159;
```

```
char ch = 'x';
```

Variable scope

- Scope determines the visibility of program elements with respect to other program elements.
- In Java, scope is defined separately for classes and methods:
 - 1) variables defined by a class have a global scope
 - 2) variables defined by a method have a local scope
- A scope is defined by a block: `{ ... }`
- A variable declared inside the scope is not visible outside:

```
{ int n; } n = 1; // this is illegal
```

Variable Lifetime

- A variable declared in a method will not hold its value between different invocations of this method.
- A variable declared in a block loses its value when the block is left.
- Initialized in a block, a variable will be reinitialized with every re-entry. Variables lifetime
- is confined to its scope!

ScopeExample.java

Arrays

Array

An array is a group of liked-typed variables referred to by a common name, with individual variables accessed by their index.

Arrays are:

- 1) declared
- 2) created
- 3) initialized
- 4) used

Also, arrays can have one or several dimensions.

Array Declaration

Array declaration involves:

- 1) declaring an array identifier
- 2) declaring the number of dimensions
- 3) declaring the data type of the array elements

Two styles of array declaration:

```
type array-variable[];
```

```
Type array[] variable;
```

Array Creation

- After declaration, no array actually exists. In
- order to create an array, we use the new
- operator:
 `type array-variable[];`
 `array-variable = new type[size];`
- This creates a new array to hold size elements of
- type type, which reference will be kept in the
- variable array-variable.

Array Indexing

- Later we can refer to the elements of this array through their indexes:
`array-variable[index]`
- The array index always starts with zero!
- The Java run-time system makes sure that all array indexes are in the correct range, otherwise raises a runtime error.

Array Initialization

- Arrays can be initialized when they are declared:

```
int monthDays[] =  
{31,28,31,30,31,30,31,31,30,31,30,31};
```

Note:

- 1) there is no need to use the new operator
- 2) the array is created large enough to hold all specified elements

Multidimensional Arrays

Multidimensional arrays are arrays of arrays:

1) declaration: `int array[][];`

2) creation: `int array = new int[2][3];`

3) initialization

```
int array[][] = { {1, 2, 3}, {4, 5, 6} };
```