Object Oriented Programming
Java: Data types, Variables and Arrays

Bikash Karmokar
Lecturer,
Department of CSE,
Bangladesh University

Primitive Data Types

Java defines eight primitive types of data:

- byte
- short
- int
- long
- char
- float
- double
- boolean

Primitive Data Types

These can be put in four groups:

Integers: byte, short, int, and long; Represent whole valued signed numbers.

Floating-point numbers: float and double; Represent fractional numbers.

Characters: char; Represents symbols in a character set, like letters and numbers.

Boolean: boolean; Special type for representing true or false values.

Primitive Data Types

Integers

- byte 8-bit integer.
- short 16-bit integer.
- int 32-bit signed integer.
- long 64-bit singed integer.

Floating-point numbers

- float 32-bit floating-point number.
- double 64-bit floating-point number.

Other types

- char **16-bit,Unicode 2.1 character.**
- boolean true or false, **false is not 0 in Java**

Variables: declaration and initialization

The variable is the basic unit of storage in a Java program. A variable is defined by the combination of an identifier, a type, and an optional initializer.

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

```
int a, b, c;                    // declares three ints, a, b, and c.
int d = 3, e, f = 5;            // declares three more ints, initializing
                                // d and f.
byte z = 22;                    // initializes z.
Double PI = 3.14159;            // declares an approximation of pi.
char x = 'x';                   // the variable x has the value 'x'.
```

Variables: declaration and initialization

```
count = 100; // oops!  cannot use count before it is declared!

int count;
```

A variable cannot be used prior to its declaration.

Variables are created when their scope is entered, and destroyed when their scope is left. Variable will not hold its value once it has gone out of scope.

Variables: Dynamic Initialization

```
//Demonstrate dynamic initialization.

class DynInit {
        public static void main(String args[]) {

                double a = 3.0, b = 4.0;

                // c is dynamically initialized
                double c = Math.sqrt(a * a + b * b);

                System.out.println(" Initialized value of c is " + c);
        }
}
```

Variables: Scope and Lifetime

Java allows variables to be declared within any block. A block is begun with an opening curly brace and ended by a closing curly brace.

```
{
        //a block
}
```

A block defines a scope. So, each time you start a new block, you are creating a new scope.

Variables: Scope and Lifetime

Traditional scopes like global and local do not fit well with Java's strict, object-oriented model.

In Java, the two major scopes are those defined by

- a class &
- a method.

If a method has parameters, they too are included within the method's scope.

Type Conversion and Casting

Java allows these types of conversion:

- Java's Automatic Conversion
  - Also known as widening conversion
  - For example, the **int** type is always large enough to hold all valid **byte** values.

- Casting Incompatible Types
  - Also known as narrowing conversion
  - For example, if the integer's value is larger than the range of a byte, it will be reduced modulo byte's range.

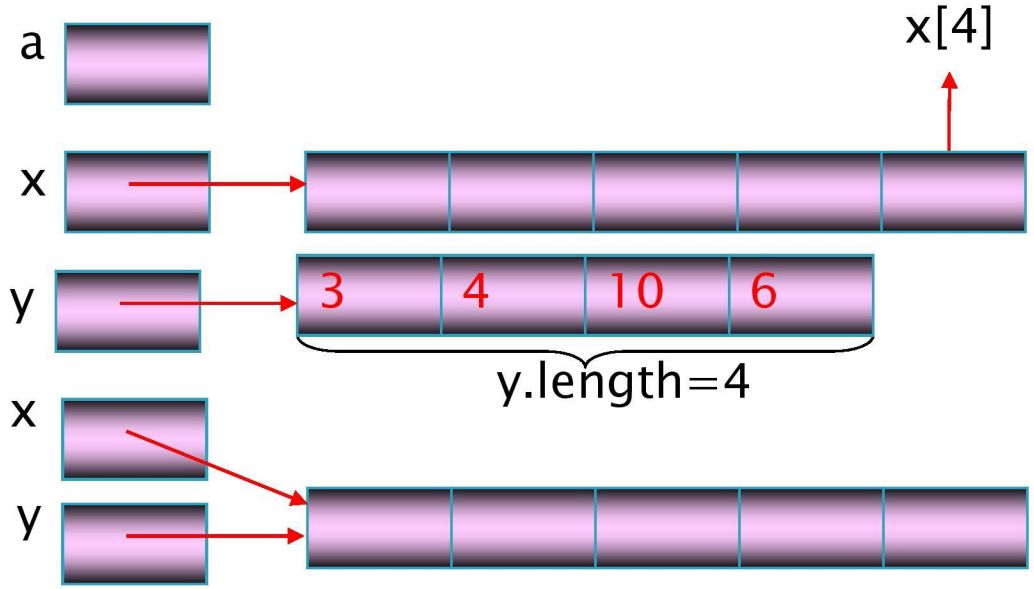
A different type of conversion will occur when a floating-point value is assigned to an integer type: **truncation**.

Arrays

- An array is a group of like typed variables that are referred to by a common name.
- Arrays are just like objects and manipulated by reference
- They are dynamically created with new
- They are automatically garbage collected when no longer referenced to.
- length is a property provided by the Java platform for all arrays.
- A specific element in an array is accessed by its index.

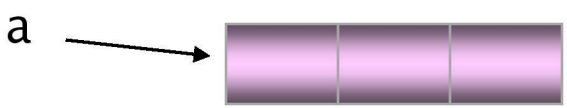
Arrays

```
int  a ;

int  [ ]  x = new int [ 5 ];

int  [ ] y  = { 3, 4, 10, 6};

y  = x;   // old object lost
```

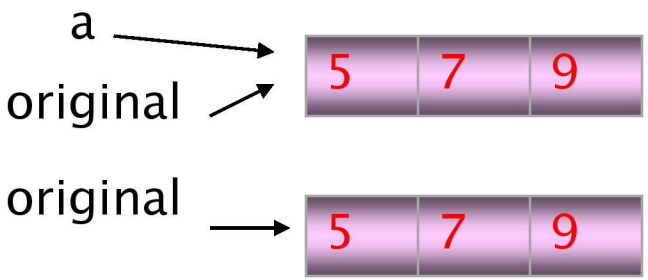


Dynamic Array Expansion

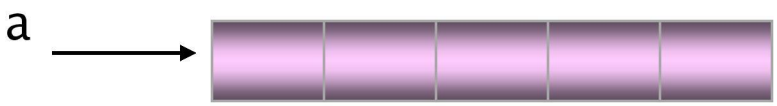
```
int [ ] a = new int [ 3 ];
```



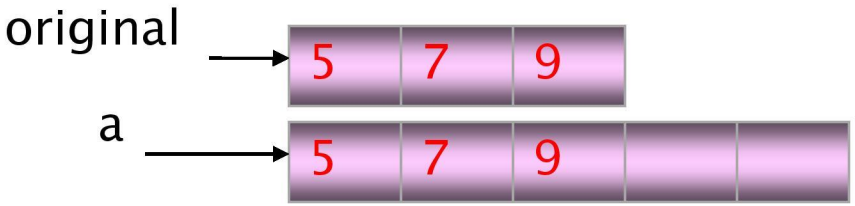

```
a=new int[ ] {5, 7, 9}
int [ ] original = a;
```




```
a = new int [ 5 ];
```

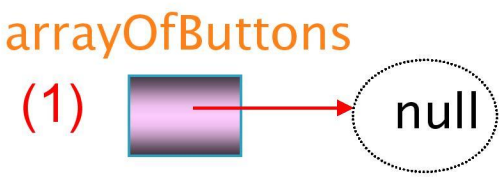


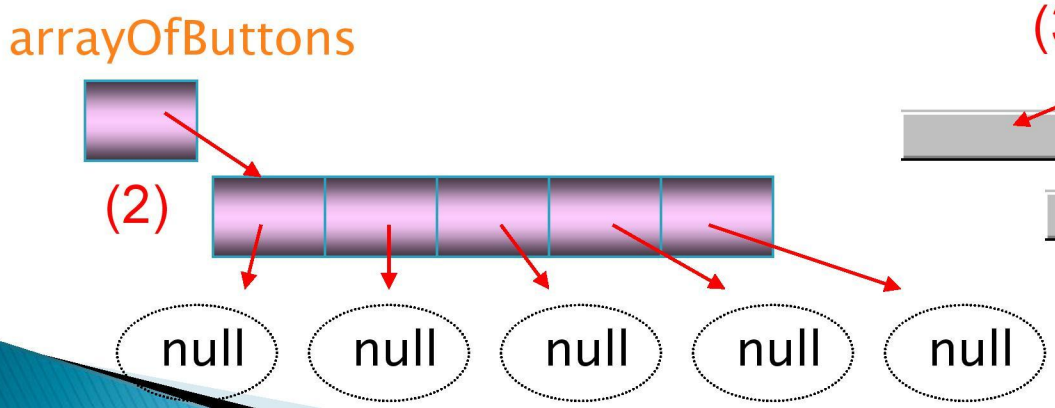

```
for( int I = 0; I < 3 ; I++)
  a[ I ] = original[ I ] ;
```

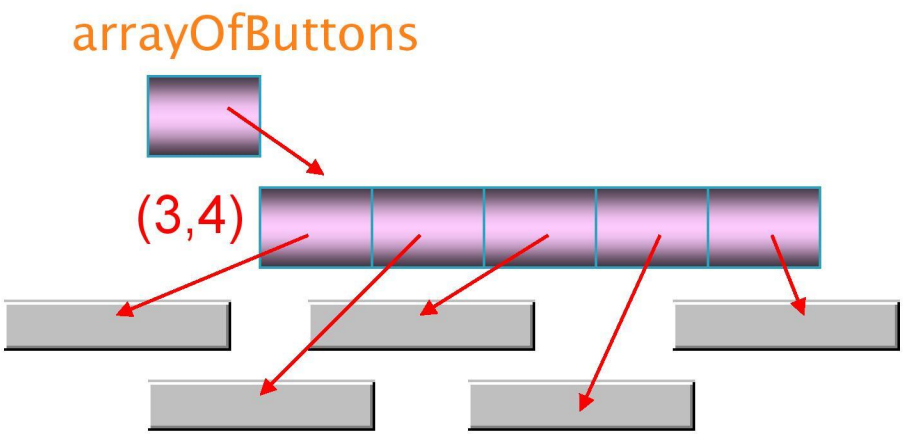


Array of Reference

```
Button [ ] arrayOfButtons;                                  //(1)
arrayOfButtons = new Button[ 5 ];                           //(2)
for ( int i=0 ; i < arrayofButtons.length; i++)             //(3)
    arrayOfButtons[ i ] = new Button( );                    //(4)
```





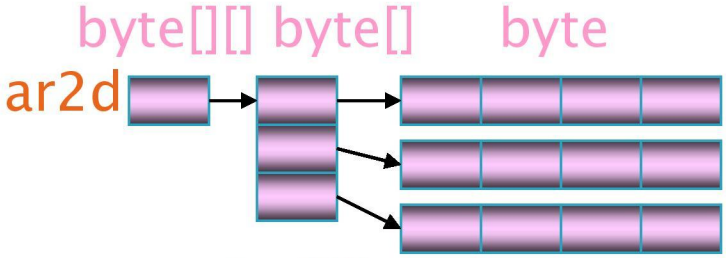


Multi-dimensional Arrays

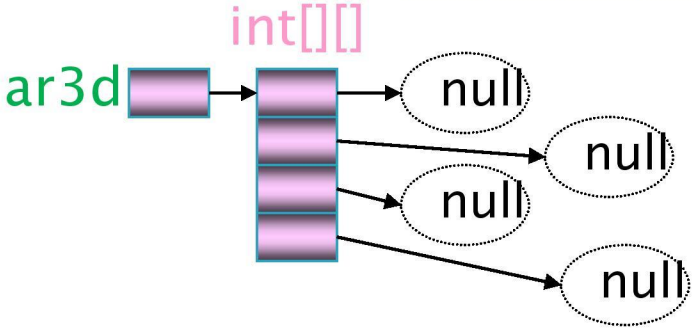
Multi-dimensional are implemented as arrays of arrays

Example:

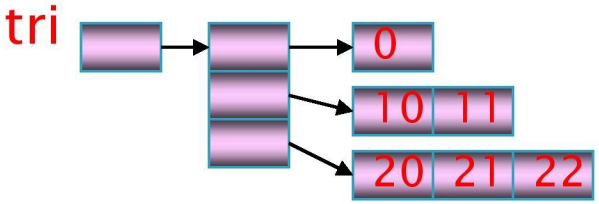
```
byte[ ][ ] ar2d = new byte [3][4];
```




```
int [ ][ ][ ] ar3d = new int [4][ ][ ];
```




```
for (int i = 0; i < tri.length; i++) {
    tri[i] = new int [i + 1];

    for (int j = 0; j < tri[i].length; j++) {
        tri[i][j] = i * 10 + j;
    }
}
```



Alternative Array Declaration Syntax

Array can be declared in both these ways:

```
int a2 [ ] = new int[3];
int [ ] a2 = new int[3];
```

```
char twod2 [ ][ ] = new char[3][4];
char [ ][ ] twod2 = new char[3][4];
```

```
int[ ] nums, nums2, nums3;        // create three arrays
int nums[], nums2[], nums3[];     // create three arrays
```

```
int num[ ], num2;          // one array, one int type variable
int [ ] num4,num5;         // both arrays
```

Arrays: java vs c++

In C++, when we declare an array, storage for the array is allocated. In Java, when we declare an array, we really only declare a pointer or reference to an array; storage for the array itself is not allocated until we use the "new" keyword. This difference is elaborated below:

C++

```
int A[10]; // A is an array of length 10
A[0] = 5; // set the 1st element of array A
```

JAVA

```
int [ ] A; // A is a reference / pointer to an array
A = new int [10]; // now A points to an array of length 10
A[0] = 5; // set the 1st element of the array pointed to by A
```

Arrays: java vs c++

In both C++ and Java we can initialize an array using values in curly braces. Here's the example

```
// myArray points to an array of length 3
// containing the values 13, 12, and 11
```

C++ code:

```
int myArray[ ] = {13, 12, 11};
```

Java code:

```
int [ ] myArray = {13, 12, 11};
```

Reference:

Java : The Complete Reference
Seventh Edition
Herbert Schildt

Thank You