x AND y. These local variables are swapped, having NO effect on a AND b.

Later we'll see how to correctly do a swap.

Ex // FcnExample
See website example.

Observe that functions can call other functions. In fact functions can even call themselves. This is called RECURSION.

Exercise
Change the initial values of a, b, and x in the above example, then try to trace it by hand. Check your work by running the program.

Prepare for the next exam by making up more complex examples.

64

```
// FcnExample.java

import java.util.*;

class FcnExample{
   public static void main( String[] args ){
      int a = 5, b = 10;
      double x = 1.5, y;

      a = fcn1(a, b);
      y = fcn2(x, a);
      b = fcn3(y, x);
      x = fcn2(y, b);
      System.out.println(a + " " + b + " " + x + " " + y);
   }

   static int fcn1(int i, int j){
      int k = 7;
      k += (i+j);
      return k;
   }

   static double fcn2(double t, int n){
      return (n-t);
   }

   static int fcn3(double u, double v){
      int m;
      m = (int)(2*u+v);
      return fcn1(2*m, 3);
   }

}

// Output:
// 22 94 73.5 20.5
```

Let $n \geq 0$ be an integer. The <u>factorial</u> of n, written $n!$ is the product of n and all the positive integers less than it. By convention $0! = 1$. Thus

$$n! = \begin{cases} n(n-1)(n-2)\cdots 3\cdot 2\cdot 1 & \text{if } n > 0 \\ 1 & \text{if } n = 0 \end{cases}$$

$n!$ can be easily calculated with a loop

<u>Ex.</u>

```
Int n=10, Product=1, i;

i = 1;
while (i <= 10) {
    Product *= i;
    i++;
}
System.out.Println(Product);

// This Prints 3628800
```

Exercise
Re-write this as a for loop.

We can write a static function called factorial and put this loop in it

Ex.

```
static int factorial(int n){
    int product=1, i=1;
    while(i<=n){
        product *= i;
        i++;
    }
    return product;
}
```

Then call from function main;

```
System.out.println(factorial(10));
```

It's also possible to do this using Recursion, i.e. write a factorial function that calls itself.

67

```
// Factorial.java
// Illustrates recursion

import java.util.*;

class Factorial{
   public static void main( String[] args ){
      int n = 10;
      System.out.println(factorial(n));
      System.out.println(factorial2(n));
   }

   static int factorial(int n){
      int product = 1, i = 1;
      while(i<=n){
         product *= i;
         i++;
      }
      return product;
   }

   static int factorial2(int n){
      if(n>0){
         return n*factorial2(n-1);
      }else{
         return 1;
      }
   }

}
```

We trace the call to factorial2( ) from main( ) where n = 5

main( ) — n: 5 — Print 120

---

fact2( ) — n: 5 — return 5 · 24 = 120

---

fact2( ) — n: 4 — return 4 · 6 = 24

---

fact2( ) — n: 3 — return 3 · 2 = 6

---

fact2( ) — n: 2 — return 2 · 1 = 2

---

fact2( ) — n: 1 — return 1 · 1 = 1

---

fact2( ) — n: 0 — return 1

IT IS OFTEN USEFUL TO HAVE TWO (OR MORE) METHODS WITH THE SAME NAME. WE CALL THIS METHOD OVERLOADING.

EX.

```
static int sum(int x, int y){
    return x+y;
}

static int sum(int x, int y, int z){
    return x+y+z;
}

static double sum(double x, double y){
    return x+y;
}
```

THE COMPILER CAN DISTINGUISH BETWEEN THE VARIOUS SUM() FUNCTIONS BY THEIR SIGNATURE, i.e. THE RETURN TYPE, TOGETHER WITH THE LENGTH AND TYPES IN THE ARGUMENT LIST.

THUS

```
System.out.println(sum(6,7));
```

Prints out 13, while

```
System.out.println(sum(6.0, 7.01));
```

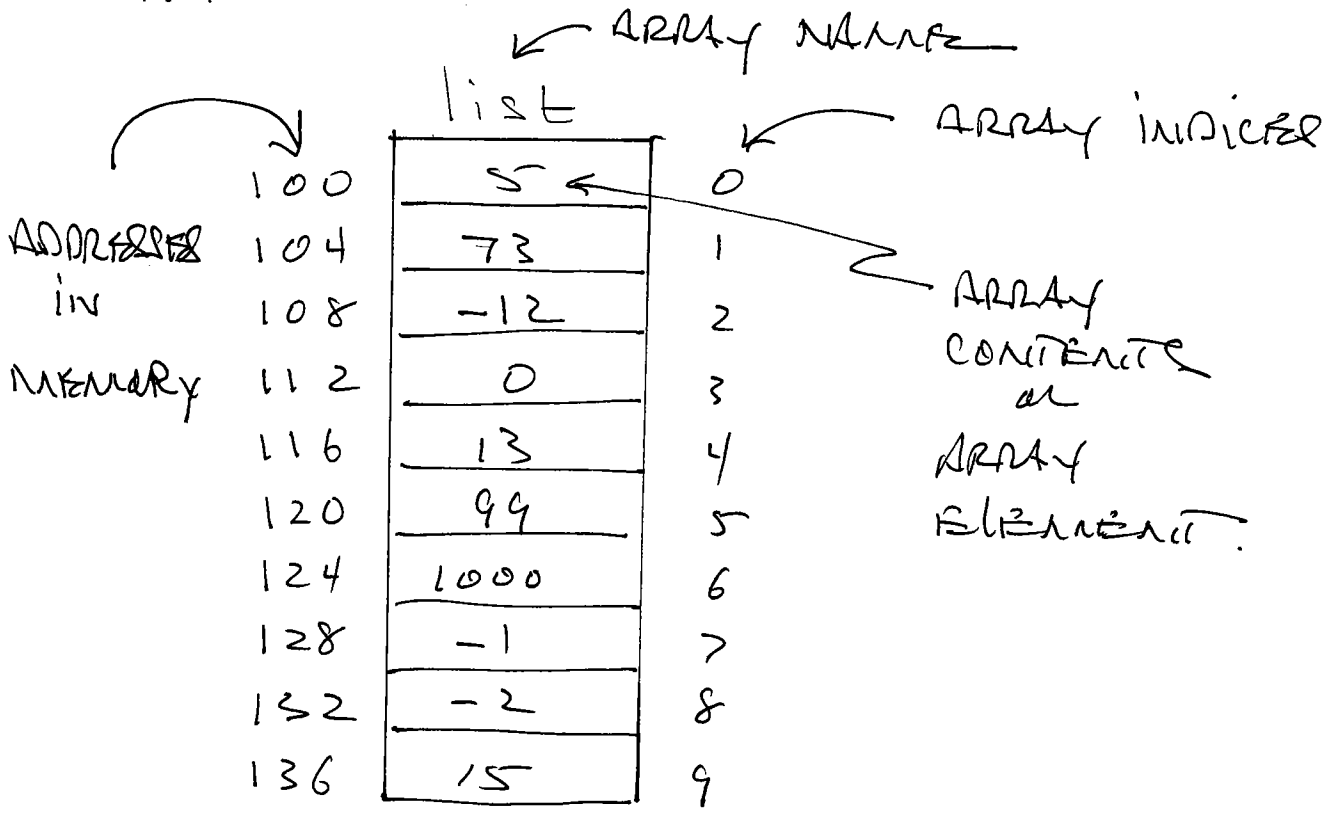
Prints 13.0.

ONE MUST BE CAREFUL NOT TO OVERLOAD METHODS IN AN AMBIGUOUS MANNER. SEE P.P. 110 - 111 FOR EXAMPLES.

CHAPTER 8

ARRAYS AND CONTAINERS

AN ARRAY IN JAVA IS A CONTIGUOUS SET OF MEMORY LOCATIONS, ALL STORING THE SAME TYPE OF DATA.



RECALL THE int DATA TYPE OCCUPIES 32 BITS (i.e. 4 BYTES) OF MEMORY. AN ARRAY OF 10 INTS MIGHT LOOK LIKE THE ABOVE PICTURE.

ARRAYS ARE NAMED USING VALID JAVA IDENTIFIERS, SUCH AS THE NAME 'list' ABOVE.

INDIVIDUAL ARRAY ELEMENTS ARE ACCESSED BY THEIR INDEX. FOR INSTANCE, GIVEN THE ABOVE PICTURE

```
System.out.println( list[2] + " " + list[5]);
```

PRINTS OUT : -12 99

WE DECLARE ARRAYS BY PLACING EMPTY BRACKETS AFTER THE CORRESPONDING DATA TYPE, AS IN:

```
int[] list;
```

HOWEVER THE ABOVE DECLARATION DOES NOT ALLOCATE ANY SPACE FOR THE ARRAY list. WE ALLOCATE SPACE FOR AN ARRAY BY USING THE new OPERATOR

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

THIS CREATES AN UNINITIALIZED ARRAY SIMILAR TO THE ONE ABOVE.

TO INITIALIZE THE ARRAY ARRAY WE MUST DO

```
list[0] = 5;
list[1] = 73;
list[2] = -12;
  :
list[8] = -2;
list[9] = 15;
```

NOTICE THAT ARRAY INDICES RANGE FROM 0 TO ONE LESS THAN THE ARRAY LENGTH. ANY REFERENCE, IN THIS EXAMPLE, TO AN INDEX GREATER THAN 9, OR A NEGATIVE INDEX WILL CAUSE AN ArrayIndexOutOfBoundsException TO BE THROWN.

WE CAN ALWAYS OBTAIN AN ARRAY'S LENGTH AS FOLLOWS

```
int n = list.length;
```

THEN USE n TO CONTROL A LOOP WHICH PROCESSES ARRAY list[]

Ex.

```
int n = 10, i;
int[] list = new int[n];

for(i=0; i<n; i++){
    list[i] = 3*i;
}

for(i=0; i<n; i++){
    System.out.print(list[i] + " ");
}
System.out.println();
```

CAN BE ANY INT EXPRESSION

CAN BE ANY INT EXPRESSION

OUTPUT:

0 3 6 9 12 15 18 21 24 27

OBSERVE THAT THE LOOP

```
for(i=0; i<=n; i++){
    System.out.print(list[i] + " ");
}
```

WOULD GENERATE THE SAME OUTPUT, THEN THROW AN EXCEPTION.

```
////////////////////////////////////////////////////////////////////////////////
///
//
//  ArrayExample.java
//
//  Reads in the length of a word list, reads in the list of words, then prints out
//  the list in reverse order.  Stores the list in an array of Strings.
//
////////////////////////////////////////////////////////////////////////////////
///

import java.util.Scanner;

class ArrayExample{
   public static void main( String[] args ){
      Scanner sc = new Scanner(System.in);
      int i, n;
      String[] list;

      // get list length
      System.out.print("Enter the list length: ");
      n = sc.nextInt();

      // allocate space for a String array of length n
      list = new String[n];

      // read in the n words
      System.out.println("Enter " + n + " words: ");
      for(i=0; i<n; i++){
         list[i] = sc.next();
      }

      // print the words in reverse order, all on one line
      System.out.println("In reverse order:");
      for(i=n-1; i>=0; i--){
         System.out.print(list[i]+" ");
      }
      System.out.println();
   }



}
```

ANOTHER WAY TO BOTH ALLOCATE AND INITIALIZE AN ARRAY IS

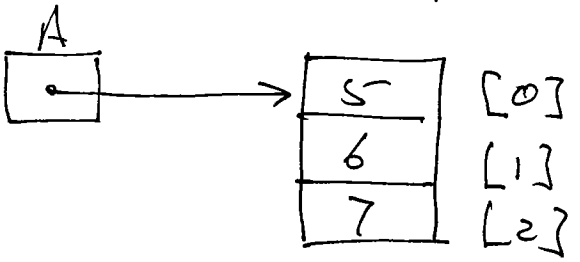
```
int[] list = {0, 3, 6, 9, 12, 15, 18, 21, 24, 27};
```

THE COMPILER INFERS THE LENGTH OF ARRAY list BY THE EXPRESSIONS IN BRACES.

NOTICE THAT ARRAY NAMES ARE FUNDAMENTALLY DIFFERENT FROM THE VARIABLE NAMES WE'VE ENCOUNTERED SO FAR

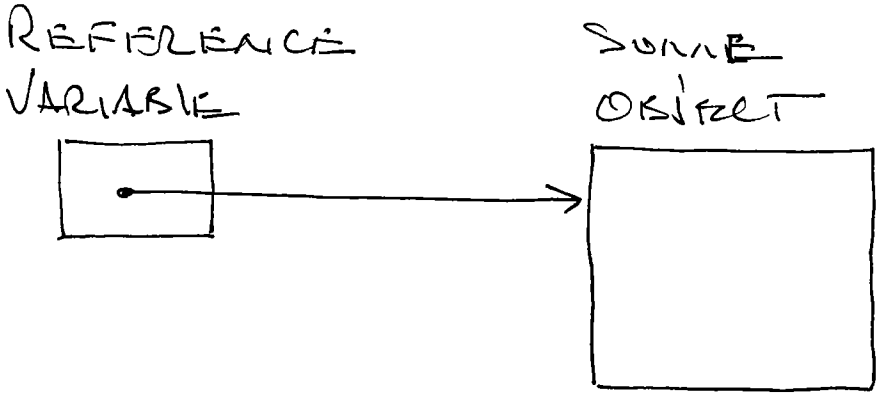
```
int a = 6;
```


```
int[] A = {5, 6, 7};
```



THE NAME OF AN ARRAY IS ITSELF A VARIABLE WHICH STORES THE ADDRESS OF THE FIRST ELEMENT IN THE ARRAY. SUCH VARIABLES ARE CALLED REFERENCE VARIABLES.

MORE PRECISELY, A REFERENCE VARIABLE LITERALLY STORES THE ADDRESS OF SOME OTHER OBJECT IN MEMORY

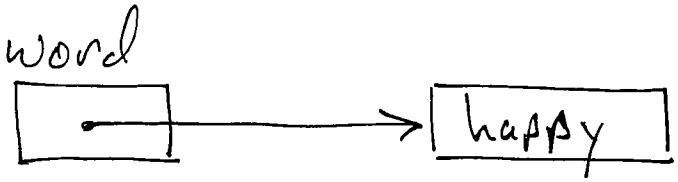


ALL ARRAY VARIABLES ARE REFERENCE VARIABLES. STRING VARIABLES ARE ALSO REFERENCE VARIABLES.

EX.

```
String word = "happy";
```

CREATES THE FOLLOWING PICTURE



REFERENCE VARIABLES ARE ALSO CALLED POINTERS IN SOME OTHER LANGUAGES, ALSO HANDLES IN OTHER CONTEXTS.

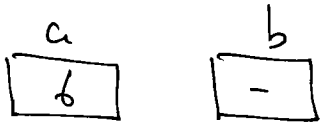
RECALL THAT Java DATA TYPES BELONG TO ONE OF TWO CATEGORIES.

- PRIMITIVE TYPES : 8
  byte, char, short, int, long
  float, double
  boolean

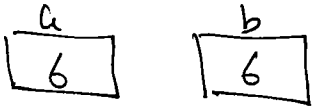
- REFERENCE TYPES : All others
  String, Scanner, ...
  ARRAY TYPES : int[], double[], ...

Ex. PRIMITIVE TYPES

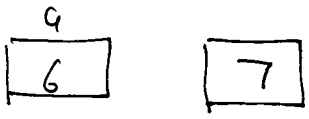
```
int a = 6, b;
```




```
b = a;
```



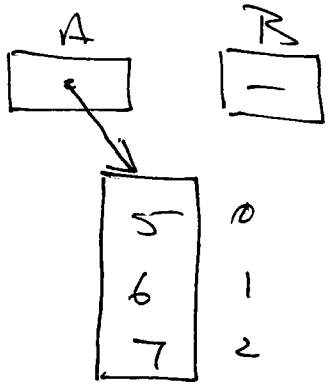

```
b = 7;
```



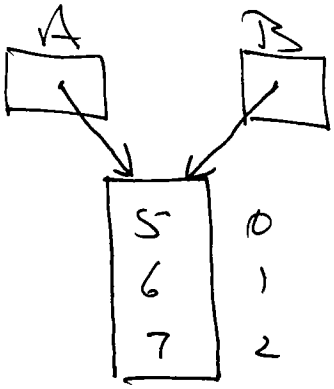

```
System.out.println(a); // Prints 6
```

Ex REFERENCE TYPES : ARRAYS

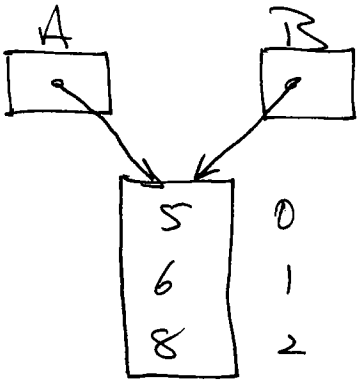
```
int[] A = { 5, 6, 7};
int[] B;
```




```
B = A;
```




```
B[2] = 8;
```




```
System.out.print("A = ( ");
for(int i = 0; i < 3; i++)
    System.out.print(A[i] + " ");
System.out.println(")");

// Prints A = ( 5 6 8 )
```

MUCH CONFUSION ARISES FROM THIS DISTINCTION BETWEEN PRIMITIVE VARIABLES AND REFERENCE VARIABLES. ONE SHOULD ALWAYS KEEP THE APPROPRIATE PICTURE (AS ABOVE) IN MIND. (SEE EX. ON P. 127.)

WHEN AN ARRAY PARAMETER IS PASSED TO A METHOD, IT IS JUST THE ADDRESS THAT IS PASSED, i.e. NO COPY OF THE ARRAY IS MADE.

Ex.

```
static void printArray(int[] A){
    System.out.Print("( ");
    for(int i=0; i<A.length; i++){
        System.out.Print(A[i]+" ");
    }
    System.out.println(")");
}

Public static void main(String[]args){
    int[] a={3, 2, 1};
    int[] b={4, 5, 6, 7, 8};
    printArray(a); // (3 2 1)
    printArray(b); // (4 5 6 7 8)
}
```