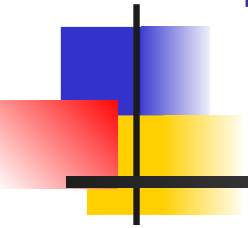


Passing the Value of a Parameter to a Method

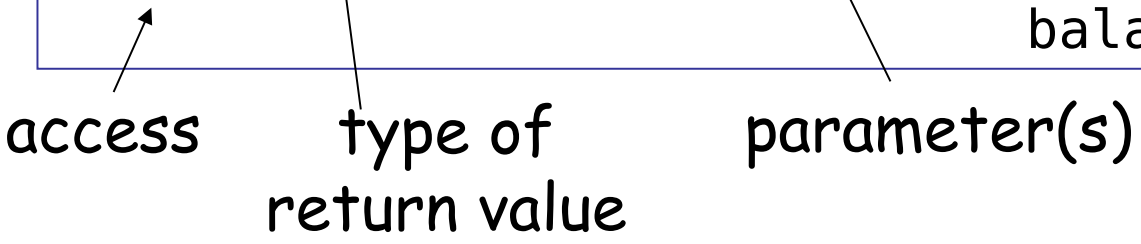
Method parameters are always pass-by-value in Java

Variable length parameters

Method Parameters

Pass data to methods using parameters.

```
public void deposit( long amt ) {
                                balance += amt; }
```



Actual argument type must be compatible with parameter:

```
BankAccount acct = new BankAccount();
int a = 1000;
long b = 1000000L;
acct.deposit( a );        // OK ... argument type matches parameter
acct.deposit( b );          // OK ...but why?
acct.deposit( 50.25 );    // ERROR ... incompatible type
```

Method Parameters, again

- Both the number and type of argument must match the method *signature*.

```
// overloaded method:
int max(int m, int n) {   . . .    }
float max(float x, float y) {   . . .   }
float max(float x, float y, float z) { . . . }
```

- Which "max" method will be called?

```
int   r = max( 20, 45 );
float q = max( 20, 33.F);  // mixed arguments
float z = max(1, 2, 3.5F); // mixed arguments
int   p = (int) max( 2 , -9.3F );
```

Arguments are Passed by Value

- In Java, arguments are always passed by value.
- The method cannot change the caller's argument.
- A method *can* change the object the an argument references!

```
public void swap( int a, int b ) { // swap args
      int temp = a;
      a = b;
      b = temp;
}
public static void main(String [] args) {
      int a = 10;  int b = 20;
      swap( a, b );
      System.out.println( "a = " + a );
            // prints "a = 10"
```

Passing objects as arguments (1)

- A method can not change the *value* of the caller's arguments. So this has no effect on date in main...

```
public void changeDate( Date date ) {
      date = new Date(105, // year 2005
                      Calendar.DECEMEBER, 31);
}

public static void main(String [] args) {
      Date date =
          new Date(100, Calendar.JANUARY, 1);
      changeDate( date );
      System.out.printf( "Date is %tF",
date );
```

```
Date is 2000-01-01
```

Passing objects as arguments (2)

- A method can change the *object* that the parameter refers to.

```
public void changeDate( Date date ) {
      date.setMonth( Calendar.DECEMEBER );
      date.setDate( 31 );
   date.setYear( 105 );
}
public static void main(String [] args) {
      Date date =
         new Date(100, Calendar.JANUARY, 1);
      changeDate( date );
      System.out.printf( "Date is %tF",
date );
```

```
Date is 2005-12-31
```

Passing array as argument

- The same rule applies to arrays:

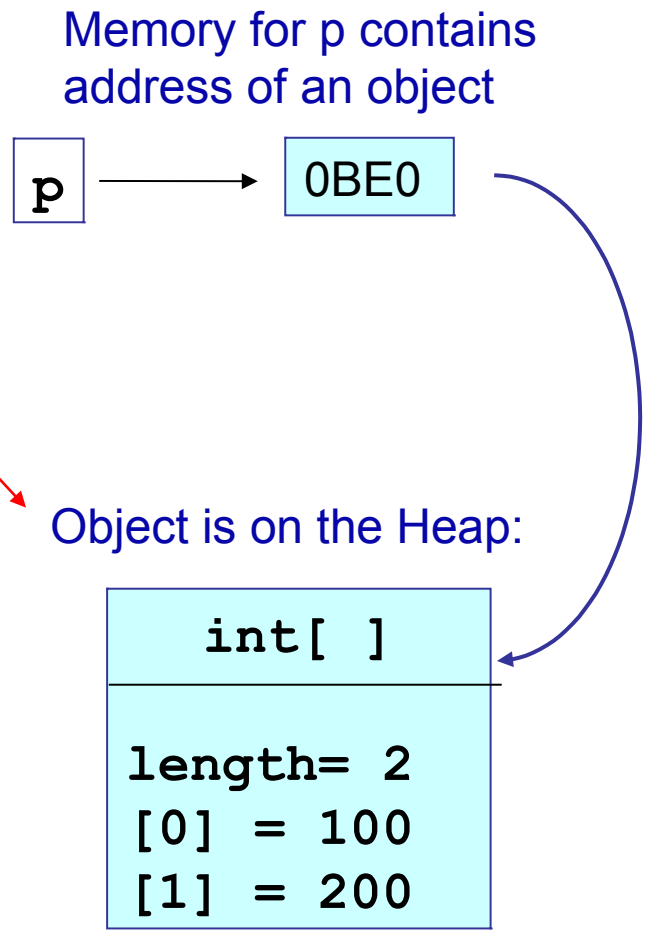
```
public void swap( int [ ] a ) {    // swap
first 2 elements
      int tmp = a[0];
      a[0] = a[1];
      a[1] = tmp;
}

public static void main(String [] args) {
      int [ ] a = new int[ ] { 10, 20 };
      swap( a );
      System.out.println(" a[0] = " + a[0] );
```

An array variable is a reference to an *array object*.

```
a[0] = 20
```

Object parameters (continued)

```
int [] p = { 100, 200 };
```



An array is an object.

The array name p refers to the storage area where the array object is stored.

Object parameters (continued)

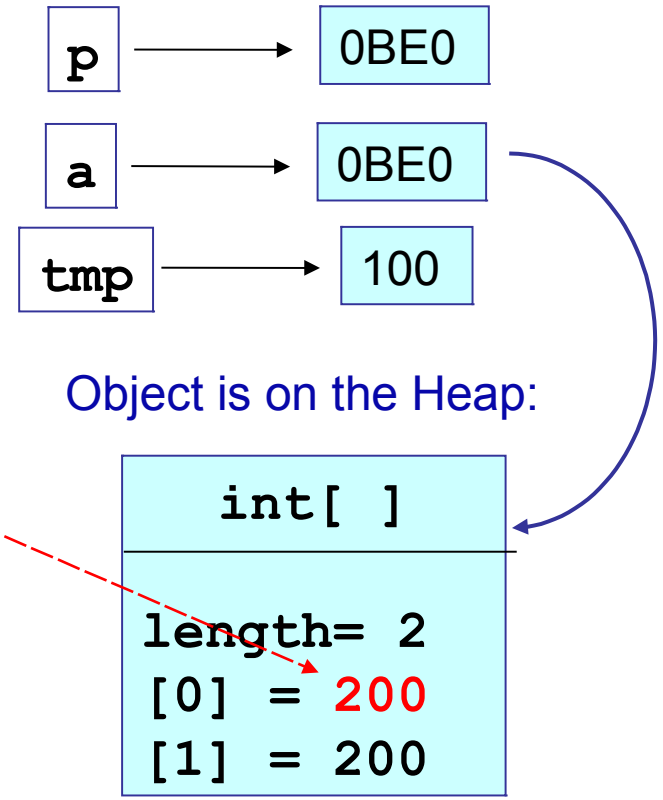
```
int [] p = { 100, 200 };
swap( p );
```

```
void swap(int [] a) {
    int tmp = a[0];
    a[0] = a[1];
```

The swap method gets a copy of p's address. It also has storage for a local variable (tmp).

swap( ) can use the address to change parts of the array object.

Memory for p contains address of an object



Parameter Passing

Pass by Value ("call by value") means a function gets a copy of the caller's arguments. Changes to the copy to not effect the caller.

Pass by Reference ("call by reference") means that the function parameters refer to the same storage used by the caller's arguments.

Java *always* uses "**pass by value**".

- a method cannot change values of the caller's arguments
- a method can change the object that a parameter refers to (this change effects the caller's data)

Parameter Passing in C#

- C# has both call by value and call by reference.
- Use "**`ref`**" to indicate call-by-reference parameters

```
/* this is call by value (can't change args) */
static void swap(int a, int b) {
  int temp = a;  a = b;   b = temp;
}
```

call by value

```
/* this is call by reference (can change args)*/
static void swap(ref int a, ref int b) {
  int temp = a;  a = b;  b = temp;
}
```

call by reference

How does C Pass Parameters?

- C always passes parameters by value (same as Java).
- To enable a function to change values of caller's arguments, you must use a pointer ("**`int *a`**" in C).

```
/* this doesn't work (pass by value) */
void swap(int a, int b) {
  int temp = a;  a = b;    b = temp;
}
```

```
/* this works: use pointers */
void swap(int *a, int *b) {
  int temp = *a;  *a = *b;  *b = temp;
}
```

Parameter Passing in C

- An array name is a pointer (reference) to an array. So, even using "call by value" a function can change the caller's array elements!

```
/* double the first element of the array */
void double(int a[ ]) {
  a[0] = 2*a[0];  // change the storage a points to
}
```

```
int main( ) {
  int p[1];
  p[0] = 100;
  double( p );
  printf("%d\n", p[0]);  // prints "200"
}
```

Parameter Passing in C++

- C++ has both "call by value" and "call by reference"
- Use "&" to indicate reference parameters

```
/* this does not change the caller's a or b */
void swap(int a, int b) {
  int temp = a;  a = b;   b = temp;
}
```

pass by value

```
/* this does change the caller's a and b values */
void swap(int &a, int &b) {
  int temp = a;  a = b;  b = temp;
}
```

pass by reference

You can write "int& a" or "int &a" or "int & a".

Variable Length Parameters

- A method can have a variable number of parameters.
- We can write one max method to do this:

```
max = MyMath.max( x1 ); // = x1
max = MyMath.max( x1, x2 );
max = MyMath.max( x1, x2, x3 );
max = MyMath.max( x1, x2, x3, x4 );
max = MyMath.max( x1, x2, x3, x4, x5 );
```

Variable Length Parameter Syntax

- Use "`... name`" for the variable length parameter.
- Use ***`name[k]`*** as array inside the method.

```
public static double max( double ... x )
{
      if (x.length == 0)
            throw new
IllegalArgumentException("duh!");
      double max = x[0];
      for (int k=1; k<x.length; k++)
            if (x[k] > max) max = x[k];
      return max;
}
```

Be Careful!

- The *actual* number of parameters may be zero.
- Be careful for zero-length array.

```
double max = MyMath.max( ); // stupid but legal
```

- To avoid empty parameter list, add a required param:

```
public static double max( double first,
                                    double ...
x )
{
      double max = first;
      for (int k=0; k<x.length; k++)
            if (x[k] > max) max = x[k];
```

Rules for Variable Length Parameter

- Can only have 1 variable length param per method.
- Must be *last parameter* in method signature.

```
double power( double ... x, int ... y ) // ERROR

void addMany( List list, String ... item) // OK

void addMany(String ... item, List list) // ERROR
```

How printf( ) works

How can **`printf( )`** print any number of items?

```
System.out.printf("hello\n");
System.out.printf("%s", s1);
System.out.printf("(x,y)=(%f,%f)", x, y);
System.out.printf("%s %f %s", s1, x, s3);
```

Format string

variable number of Objects