

Call by Value

```
void F(int x){x=5;}

int a=8;
F(a);
cout << a << endl;    // result: 8
```

Pointer-Based Call-by-Reference

```
#include <iostream>

using namespace std;

//Pointer based call-by-reference

void order(int*, int*);

int main()
{
    int i = 7, j = 3;
    cout << i << '\t' << j << endl;    // 7 3 is printed
    order(&i, &j);
    cout << i << '\t' << j << endl;    // 3 7 is printed
    int look; cin >> look;
}

void order(int* p, int* q)
{ int temp;

  if (*p > *q) {
      temp = *p;
      *p = *q;
      *q = temp;
  }
}
```

Not possible in
Java !!

Reference Type

Not in Java

```
int n;
int & nn = n;    // nn is an alternative name for n, an alias for n
```

```
double a[10];
double& last = a[9];    // last is an alias for a[9]
```

```
n = 10; cout << nn << endl;
last = 5.3; cout << a[9] << endl;
```

Prints:
10
5.3

```
cout << &last << &a[9]
```

Prints: 0x258fda8 0x258fda8
last is not a separate variable, but
• last is identical to a[9]

```
int a = 5;    // definition of a
int* p = &a;    // p is a separate pointer variable, its value is the address of a
int& aa = a;    // aa is an alias for a, aa is identical to a
*p = 7;    // the variable pointed to by p (i.e. a, aa) is assigned 7
a = *p + 1;    // ~ a = a + 1 (== 8)
cout << aa << endl;    // prints: 8
```

Initialisation is not assignment

```
int a, b = 2;

int& aa = a;    // initialisation,
                // aa initialised to be an alias for a; cannot be changed

aa = b;    // assignment, aa, a assigned the value of b

const int c = 7;    // initialisation, readonly variable initialised to 7

c = 8;    // assignment, illegal !!

int & bb;    // illegal, must be initialised
```

Call by Reference

not in Java !

```
#include <iostream>
using namespace std;
```

```
// call-by-reference
```

```
void order(int& p, int& q);
```

```
int main()
```

```
{ int i = 7, j = 3;
```

```
  cout << i << '\t' << j << endl;
```

```
    order(i, j);
```

```
  cout << i << '\t' << j << endl;
```

```
  int look; cin >> look;
```

```
}
```

7 3 is printed

Actual parameters must be
variables/ assignable

3 7 is printed

```
void order(int& p, int& q)
```

```
{ int temp;
```

```
  if (p > q) { temp = p; p = q; q = temp;}
```

```
}
```

p and q are aliases
for actual variables

Call by Reference

```
//Finding a minimum element in an array slice
```

```
void order(int& p, int& q)
```

```
{ int temp = p;
```

```
  if (p > q) { p = q; q = temp;}
```

```
}
```

```
int place_min( int a[ ], int size, int lb = 0)
```

```
{ for (int i = lb+1; i < lb + size; ++i) order(a[lb], a[i]);
```

```
  return a[lb];
```

```
}
```

```
int main()
```

```
{ int a[9] = { 6, -9, 99, 3, -14, 9, -33, 8, 11};
```

```
  cout << "Minimum = " << place_min(a, 6, 2) << endl;
```

```
}
```

Prints: -33

Call by Reference

```
class BIG
{ public:
    void F(){...}
private:
    ... // contains many data
}
```

```
class Client
{ public:
    void Val-Method(BIG a)
    { ...a.F()...
    }
    void Ref-Method(BIG& b)
    { ... b.F()...
    }
    void CRef-Method(const BIG& c)
    { ... c.F()...
    }
    ...
private:
    ...
}
```

The diagram illustrates the call-by-value and call-by-reference mechanisms for the Client class methods. Arrows point from the parameter types in the method signatures to explanatory text boxes:

- An arrow points from `BIG a` in `Val-Method` to a box stating: "call by value; a is a copy of a BIG object".
- An arrow points from `BIG& b` in `Ref-Method` to a box stating: "call by reference, aliasing Ref-Method may change the contents of b".
- An arrow points from `const BIG& c` in `CRef-Method` to a box stating: "call by reference, aliasing CRef-Method may **not** change the contents of b".