Chapter 6: Arrays

Objectives

- After studying this chapter, Stdent should be able to
  - Manipulate a collection of data values, using an array.
  - Declare and use an array of primitive data types in writing a program.
  - Declare and use an array of objects in writing a program
  - Define a method that accepts an array as its parameter and a method that returns an array
  - Describe how a two-dimensional array is implemented as an array of arrays

Array Basics

- An array is a collection of data values.
- If your program needs to deal with 100 integers, 500 Account objects, 365 real numbers, etc., you will use an array.
- In Java, an array is an indexed collection of data values of the same type.

Arrays of Primitive Data Types

- Array Declaration

```
<data type> [ ] <variable>
   //variation 1
<data type>   <variable>[ ]   //variation 2
```

- Array Creation

```
<variable> = new <data type> [ <size> ];
```

-

Variation 1

```
double[ ] rainfall;
rainfall
    = new double[12];
```

Variation 2

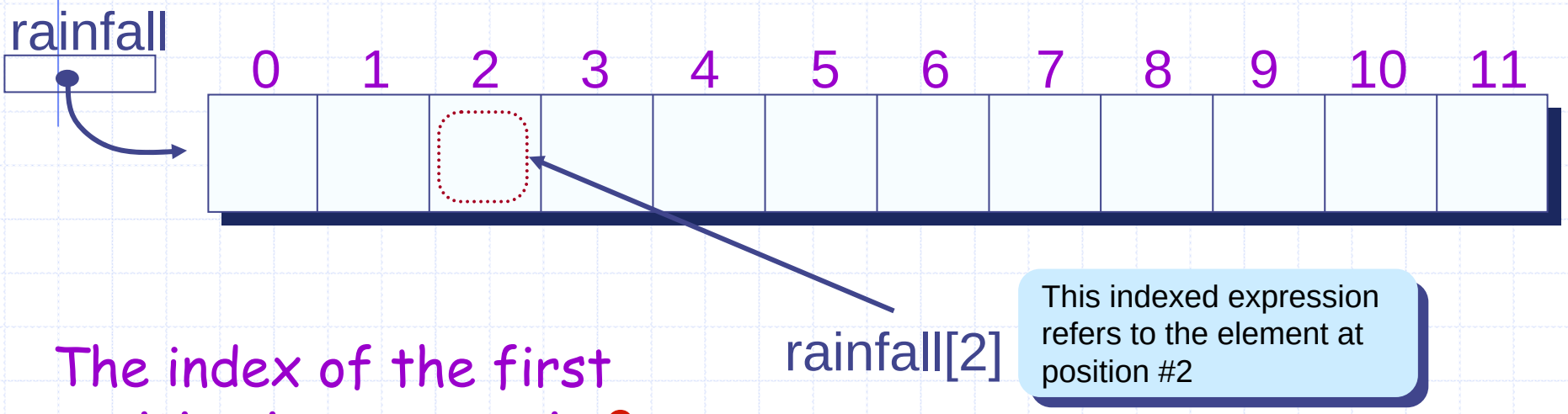
```
double rainfall [ ];
rainfall
    = new double[12];
```

An array is like an object!

Accessing Individual Elements

- Individual elements in an array accessed with the indexed expression.

```
double[] rainfall = new double[12];
```



The index of the first position in an array is 0.

Array Processing – Sample1

```
double[] rainfall = new double[12];

double   annualAverage,
          sum = 0.0;

for (int i = 0; i < rainfall.length; i++) {

    rainfall[i] = Double.parseDouble(
                      JOptionPane.showinputDialog(null,
                      "Rainfall for month " + (i+1) ) );
    sum += rainfall[i];
}

annualAverage = sum / rainfall.length;
```

The public constant length returns the capacity of an array.

Array Processing – Sample 2

```
double[] rainfall = new double[12];
String[] monthName = new String[12];
monthName[0] = "January";
monthName[1] = "February";
…
double   annualAverage, sum = 0.0;

for (int i = 0; i < rainfall.length; i++) {
    rainfall[i] = Double.parseDouble(
                        JOptionPane.showinputDialog(null,
                                "Rainfall for "
                                        + monthName[i] ));
    sum += rainfall[i];
}
annualAverage = sum / rainfall.length;
```

The same pattern for the remaining ten months.

The actual month name instead of a number.

Array Processing – Sample 3

- Compute the average rainfall for each quarter.

```
//assume rainfall is declared and initialized properly

double[] quarterAverage = new double[4];

for (int i = 0; i < 4; i++) {
    sum = 0;
    for (int j = 0; j < 3; j++) {
                                        //compute the sum of
        sum += rainfall[3*i + j];       //one quarter
    }
    quarterAverage[i] = sum / 3.0;   //Quarter (i+1) average
}
```

Array Initialization

- Like other data types, it is possible to declare and initialize an array at the same t

```
int[] number = { 2, 4, 6, 8 };

double[] samplingData = { 2.443, 8.99, 12.3, 45.009, 18.2,
                          9.00, 3.123, 22.084, 18.08 };

String[] monthName = {"January", "February", "March",
                      "April", "May", "June", "July",
                      "August", "September", "October",
                      "November", "December"  };
```

number.length → 4
samplingData.length → 9
monthName.length → 12

Variable-size Declaration

- In Java, we are not limited to fixed-size array declaration.
- The following code prompts the user for the size of an array and declares an array of

```
int size;

int[] number;

size= Integer.parseInt(JOptionPane.showInputDialog(null,
                                "Size of an array:"));

number = new int[size];
```

Arrays of Objects

- In Java, in addition to arrays of primitive data types, we can declare arrays of objects
- An array of primitive data is a powerful tool, but an array of objects is even more powerful.
- The use of an array of objects allows us to model the application more cleanly and logically.

```
public class Person
{
        private String name;
        private int age;
        private char gender;

        public Person()
    {age=0; name=" "; gender=' ';}

        public Person(String na, int ag, char gen)
  {setAge(ag); setName(na); setGender(gen); }

 public Person(Person pr)
   { setPerson(pr);}

 public void setPerson(Person p)
  { age=p.age; gender =p.gender;
  name=p.name. substring(0, p.name.length());
  }

        public void setAge (int a) {age=a;}
        public void setGender (char g) {gender=g;}
        public void setName(String na)
         {name=na.substring(0, na.length());}

        public int getAge(){return age;}

        public char getGender () {return gender;}

        public String getName () { return name;}

}
```

The Person Class

- We will use Person objects to illustrate the use of an array of objects.

```
public class Person
{
                private String name;
                private int age;
                private char gender;
                public Person()    {age=0; name=" "; gender=' ';}
                public Person(String na, int ag, char gen)  {setAge(ag); setName(na); setGender(gen); }
                public Person(Person pr)       { setPerson(pr);}
                public void setPerson(Person p)
                { age=p.age; gender =p.gender;
                  name=p.name. substring(0, p.name.length());      }
                public void setAge (int a) {age=a;}
                public void setGender (char g) {gender=g;}
                public void setName(String na)
                 {name=na.substring(0, na.length());}
                public int getAge(){return age;}
                public char getGender () {return gender;}
                public String getName () { return name;}
}
```

Creating an Object Array - 1

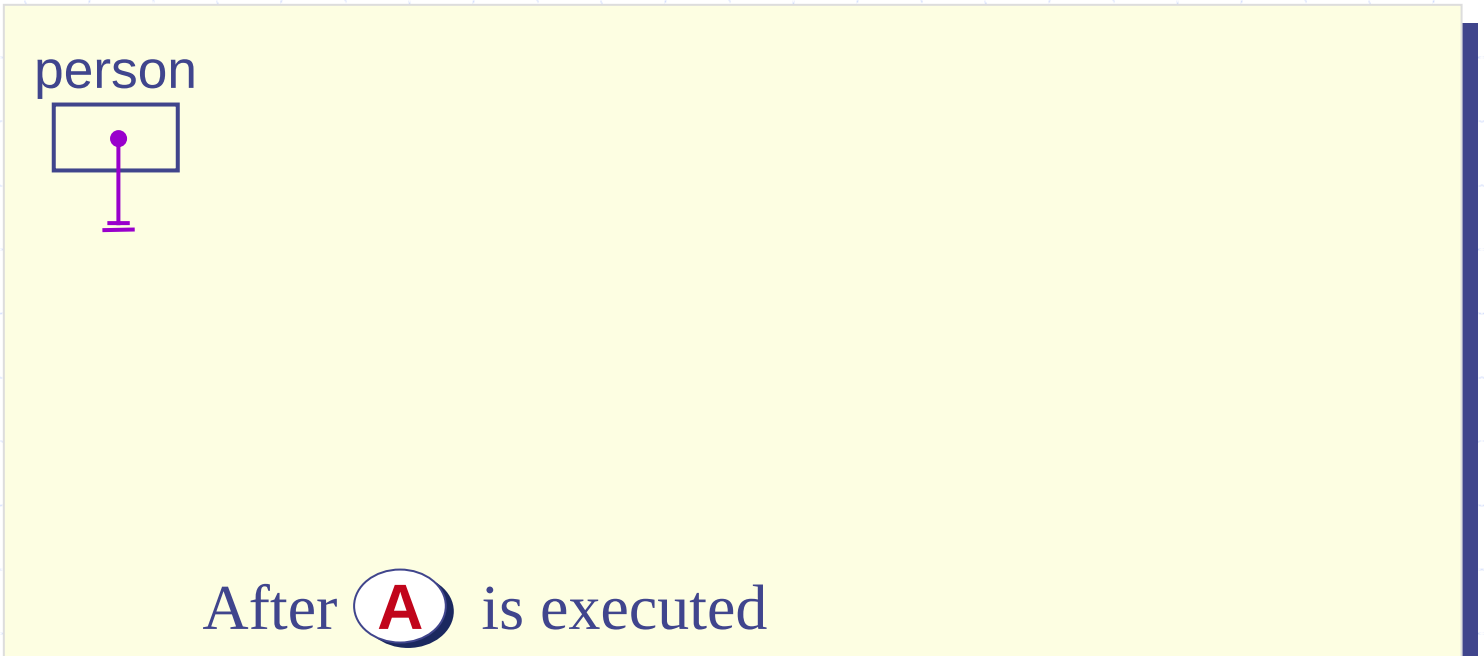
Code

A

```
Person[ ]  person;
person = new Person[20];
person[0] = new Person( );
```

Only the name person is declared, no array is allocated yet.

State of Memory



After A is executed

Creating an Object Array - 2

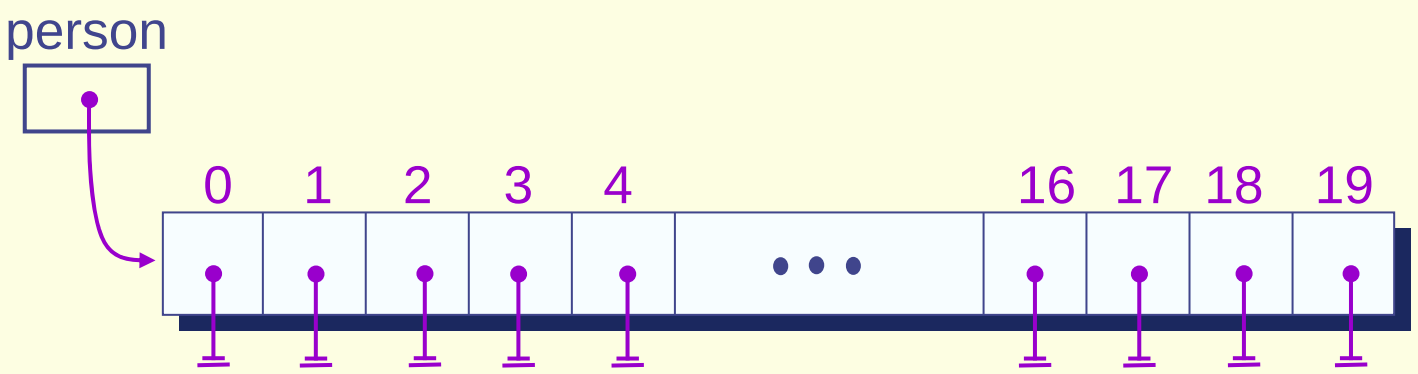
Code

B

```
Person[ ]  person;
person = new Person[20];
person[0] = new Person( );
```

Now the array for storing 20 Person objects is created, but the Person objects themselves are not yet created.

State of Memory



After B is executed

Creating an Object Array - 3

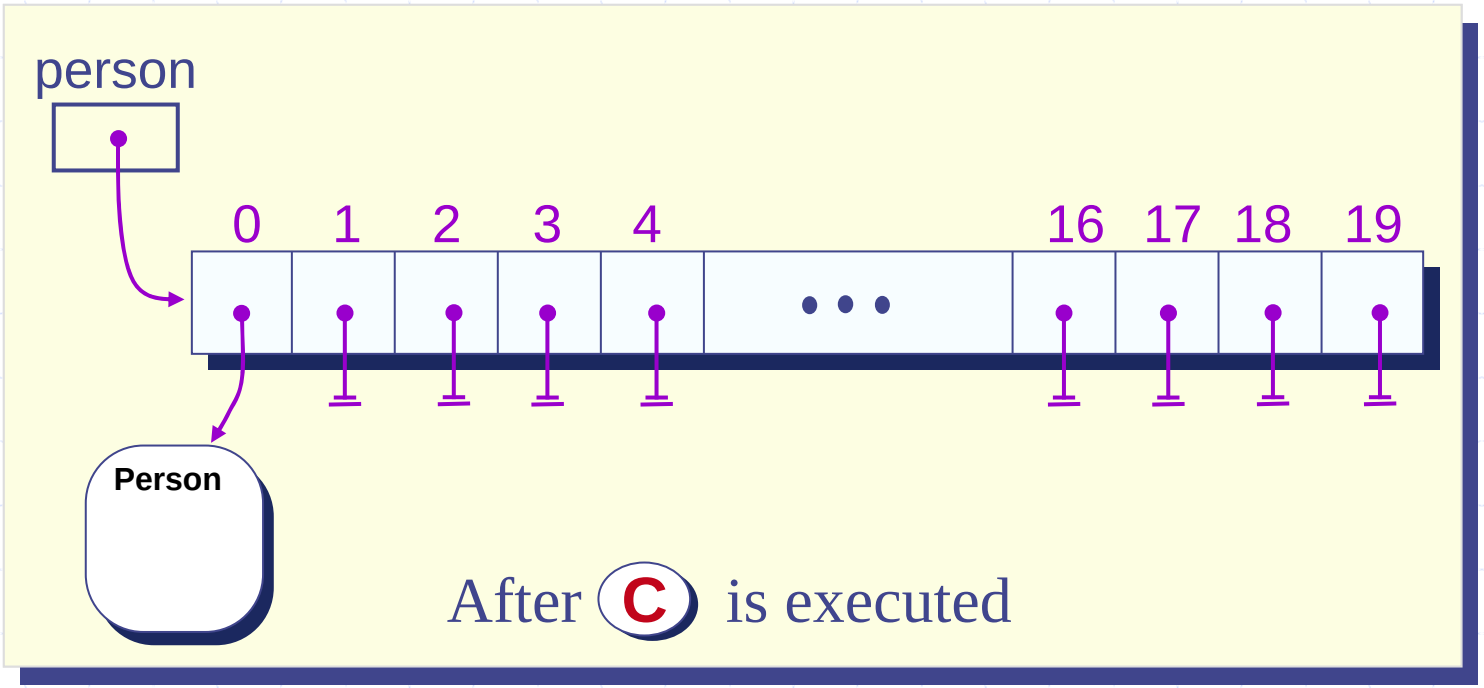
Code

```
Person[ ]  person;
person = new Person[20];
person[0] = new Person( );
```

C

One Person object is created and the reference to this object is placed in position 0.

State of Memory



Person Array Processing – Sample 2

- Find the youngest and oldest persons.

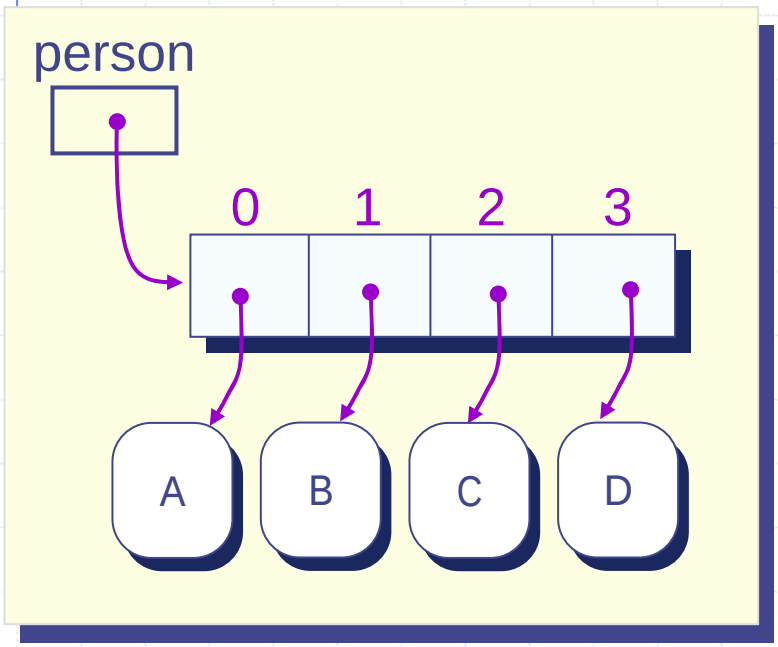
```
int      minIdx = 0;      //index to the youngest person
int      maxIdx = 0;      //index to the oldest person

for (int i = 1; i < person.length; i++) {

    if ( person[i].getAge() < person[minIdx].getAge() ) {
        minIdx       = i;         //found a younger person

    } else if (person[i].getAge() > person[maxIdx].getAge() ) {

        maxIdx        = i;        //found an older person
    }
}

//person[minIdx] is the youngest and person[maxIdx] is the oldest
```

Object Deletion – Approach 1

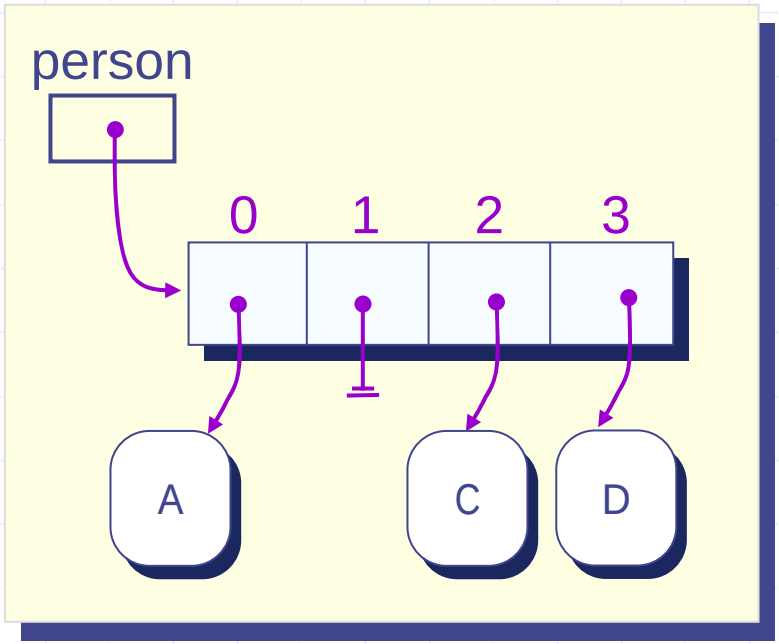
```
int delIdx = 1;
person[delIdx] = null;
```

A

Delete Person B by setting the reference in position 1 to null.



Before A is executed



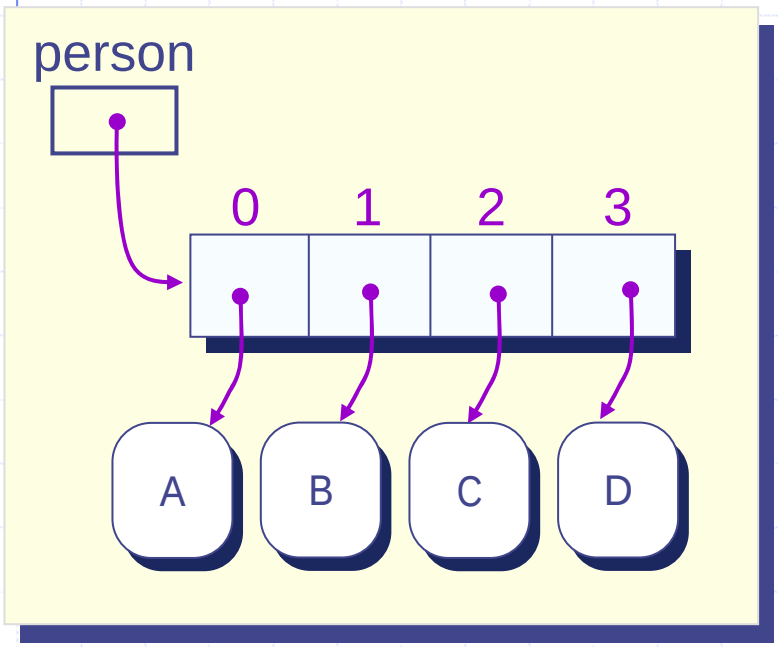
After A is executed

Object Deletion – Approach 2

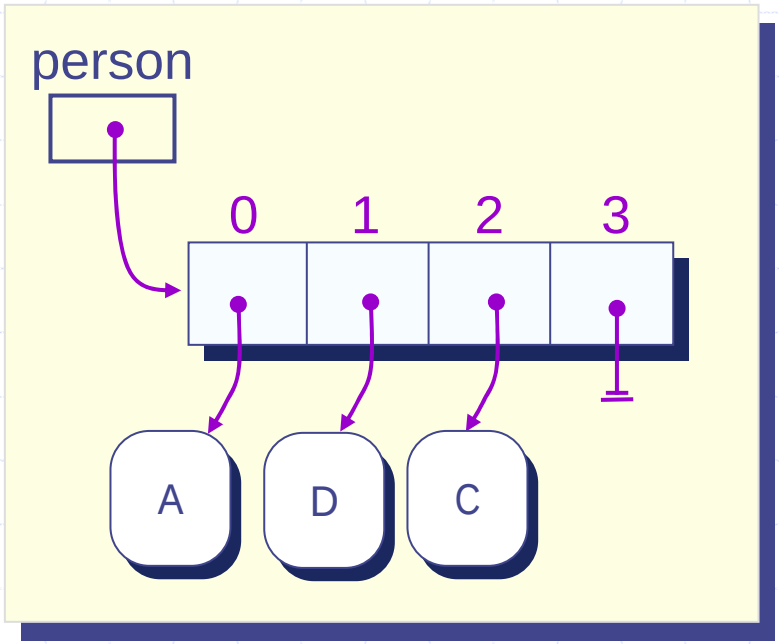
A

```
int delIdx = 1, last = 3;
person[delIndex] = person[last];
person[last]     = null;
```

Delete Person B by setting the reference in position 1 to the last person.



Before A is executed



After A is executed

Person Array Processing – Sample 3

- Searching for a particular person. Approach 2 Deletion is used.

```
int i = 0;

while ( person[i] != null && !person[i].getName().equals("Latte") ) {
    i++;
}

if ( person[i] == null ) {
    //not found - unsuccessful search
    System.out.println("Ms. Latte was not in the array");

} else {
    //found - successful search
     System.out.println("Found Ms. Latte at position " + i);
}
```

Passing Arrays to Methods - 1

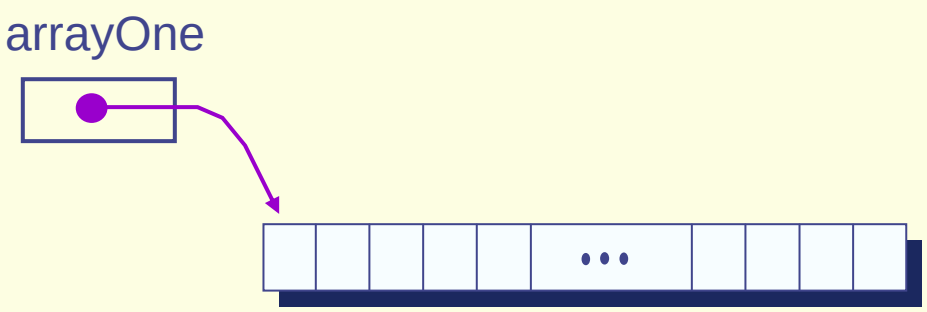
Code

A

```
minOne
= searchMinimum(arrayOne);
```

```
public int searchMinimum(float[]
number))
{

     …


}
```

At A before searchMinimum



State of Memory

A. Local variable number does not exist before the method execution

Passing Arrays to Methods - 2

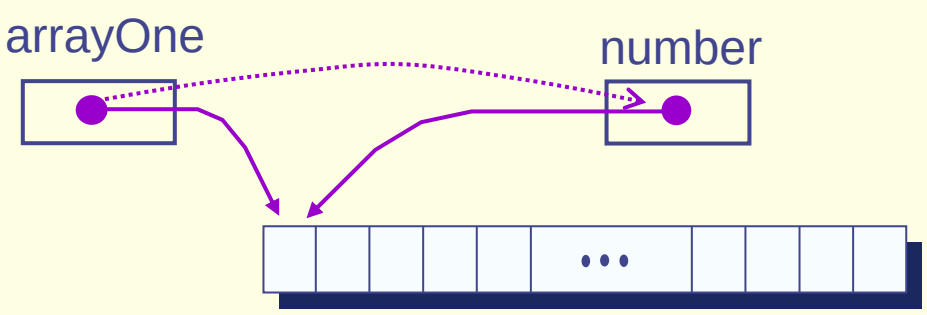
Code

```
minOne
= searchMinimum(arrayOne);
```

```
public int searchMinimum(float[]
number))
{

     …


}
```

B

The address is copied at B



State of Memory

B. The value of the argument, which is an address, is copied to the parameter.

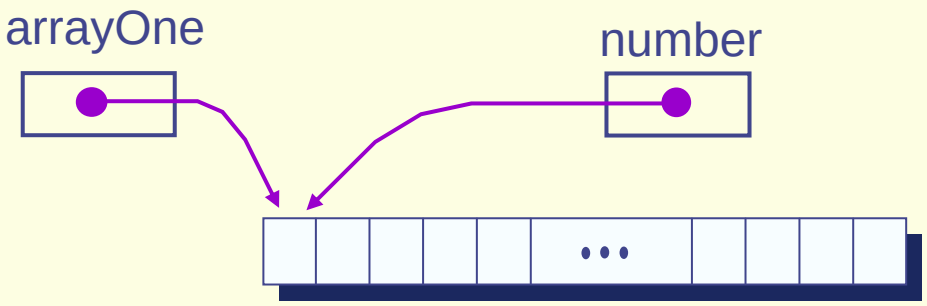
Passing Arrays to Methods - 3

Code

```
minOne
= searchMinimum(arrayOne);
```

```
public int searchMinimum(float[]
number))
{
     …
C


}
```

While at C inside the method



State of Memory

C. The array is accessed via number inside the method.

Passing Arrays to Methods - 4

Code

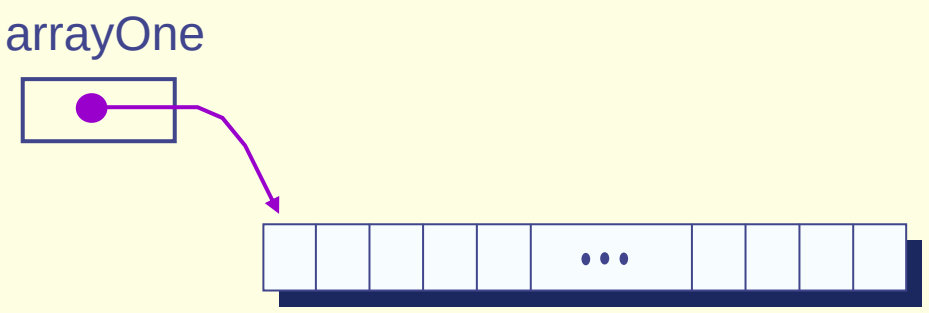
```
minOne
= searchMinimum(arrayOne);
```

D

```
public int searchMinimum(float[]
number))
{

     …


}
```

State of Memory

At D after searchMinimum



D. The parameter is erased. The argument still points to the same object.