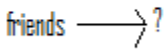
Arrays of Objects

After learning arrays, it's natural to want to make arrays of Objects. The good news is that this is possible and that there are very few, if any, new rules to learn for making arrays of objects. One declares an array of objects (perhaps an array of Contact objects) as follows:

```
Contact[] friends;
```

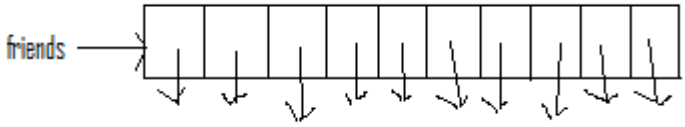
Picture wise, this ONLY allocates space for a single reference:



Next, let's allocate space for the array itself:

```
friends = new Contact[10];
```

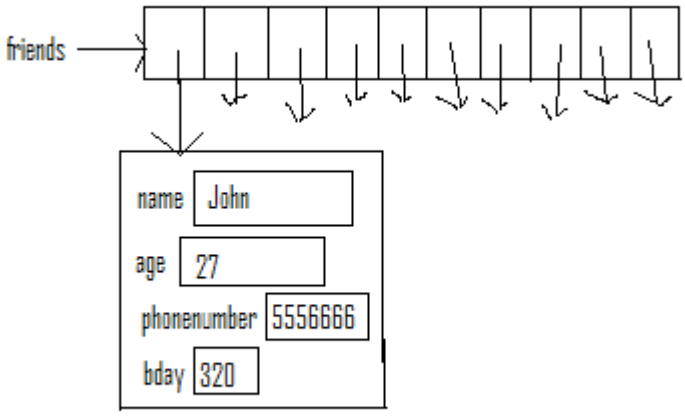
The corresponding picture is as follows:



Notice that the array slots themselves are NOT the objects, but rather references to objects. At this point in time however, NO objects have been created! Thus, each of these 10 references is uninitialized. If we create an object, we can directly use one of these references to refer to it:

```
friends[0] = new Contact("John", 27, 5556666, 3, 20);
```

The ensuing picture is as follows:



The key is to remember that each array element is just a reference to an Object of the specified type. All other rules we've learned are valid.

Creating a Class that Has an Array of Objects as an Instance Variable

It's a common practice to create a class that maintains a collection of some object. In this example, we'll use the Contact class, a small part of which, is shown here:

```
public class Contact {

    private String name; // Stores name of Contact
    private int age;  // Stores age of Contact
    private int phonenumber; // Stores phone number of contact
    private int bday;  // Stores birthday in an int

    // Creates Contact object based on parameters.
    public Contact(String n, int a, int p, int month, int day) {
          name = n;
          age = a;
          phonenumber = p;
          bday = 100*month + day;
    }
}
```

Now, imagine creating a class that manages a collection of Contact objects. This is very similar to an address book. The basic set up for this class would be as follows:

```
public class AddressBook {

    private Contact[] friends;
    private int numfriends; // Num of friends in AdrBook

    // Create an empty AddressBook
    public AddressBook() {
          friends = new Contact[10];
          numfriends = 0;
    }
}
```

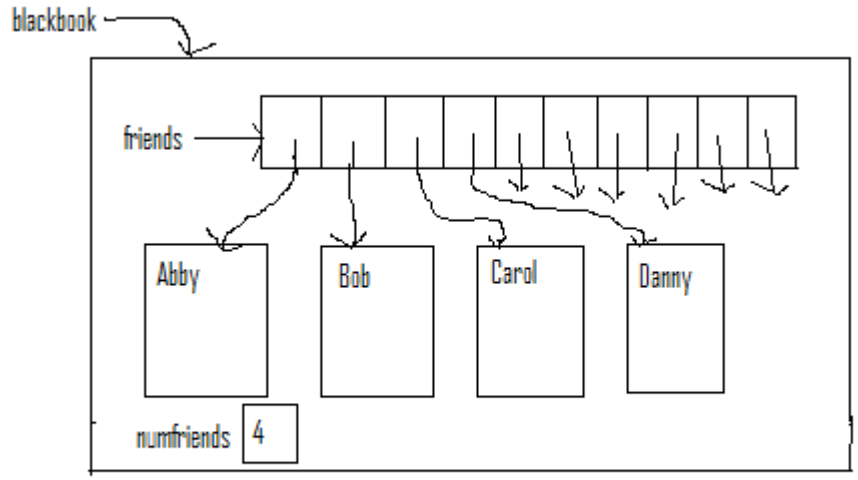
Thus, our AddressBook object is nothing but an array to store each Contact and an integer which indicates how many Contacts are currently store. If you carefully think about it, this number ALSO represents the minimum index into the array that is empty, so long as we always fill our array from smallest to largest index and maintain it in that fashion.

Let's take a look at one method from this class, in detail. You should be able to figure out the rest from this example:

```
// Deletes a contact with name s, if one exists.
public void deleteContact(String s) {

      int place = haveContact(s);
      if (place >= 0) {
            friends[place] = friends[numfriends-1];
            numfriends--;
      }
}
```

Let's trace through an example where the AddressBook object in question has 4 contacts and then attempts to delete one. Let this be the picture of the AddressBook object (let's call it blackbook) before the delete:



Now, consider running the line of code:

```
blackbook.delete("Bob");
```

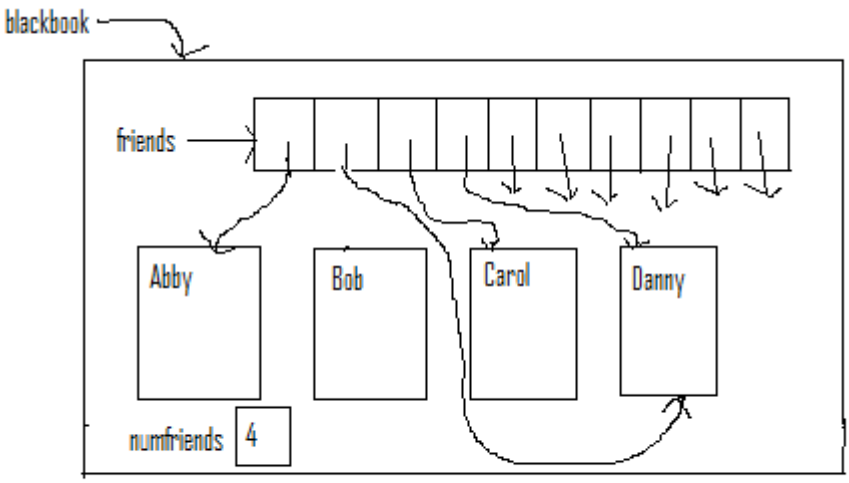
The first thing the delete code does is find which index Bob is located in. (This is what the haveContact method does. If haveContact returns -1, then the person being searched is NOT in the object.) This is then stored in the variable place. Thus, after this line of code the variable place equals 1. Now, we run the line:

```
friends[place] = friends[numfriends-1];
```

In this instance, this translates to

```
friends[1] = friends[3];
```

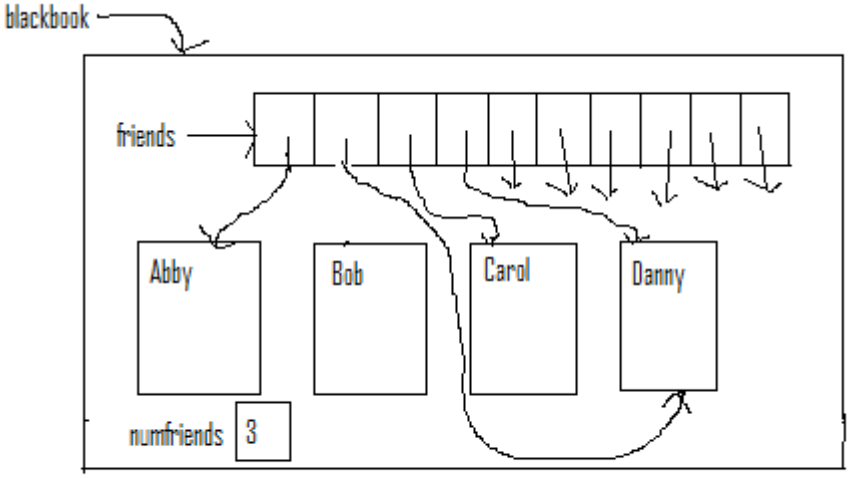
and the ensuing picture is:



Then, we run the line:

```
numfriends--;
```

Finally, our picture is:



Thus, even though we have two references pointing to Danny, only one of them (in index 1) will ever be used because the instance variable numfriends controls where we'll ever look in the array friends. A second thing to note is that no reference is pointing to Bob. Thus, Bob will eventually get garbage-collected. Finally, none of the Contact objects should really be depicted INSIDE the blackbook object. Technically, they are floating around, not in the object. Rather, the only parts of the object are the variable numfriends and the references in the friends array.