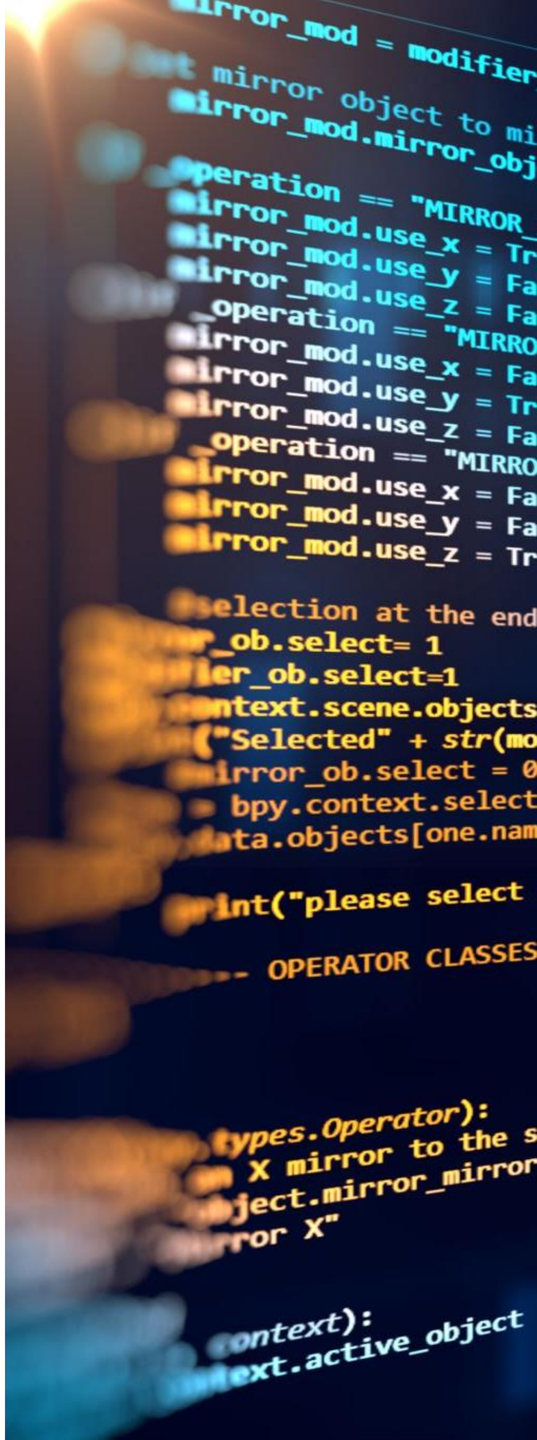
CS 1027
Fundamentals of Computer Science II

Algorithm Design & Arrays

Ahmed Ibrahim

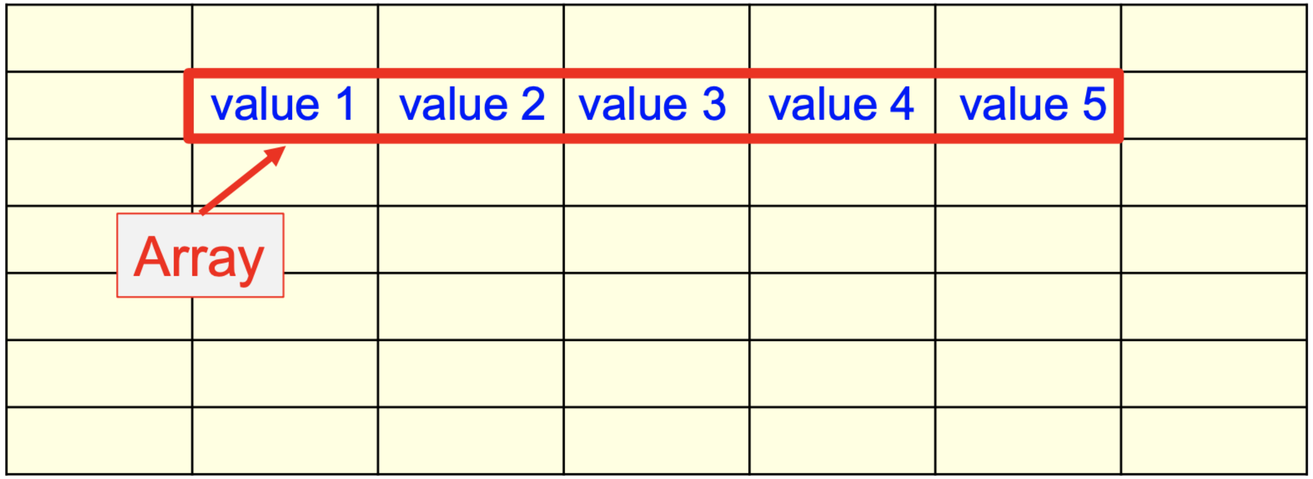


Recap

- **Object-Oriented Design**: Focused on using objects and classes to encapsulate data and behavior, applying modularity, encapsulation, and abstraction principles.
- **Address Book Project**: Demonstrated managing contacts using the Person class for individual contacts and AddressBook for storing the list.
- **Classes and Methods**: Explained declaring attributes and methods with visibility modifiers (public/private), highlighting private variables for data integrity, and the use of getters/setters.
- **Constructors**: Explained the role of constructors in initializing objects, highlighting that constructors have the same name as the class and how they're used with the new keyword to create instances.
- **Arrays**: Explained arrays as a data structure to store multiple values, their usage in Java, and how they are managed as objects with attributes like length.

AddressBook Class

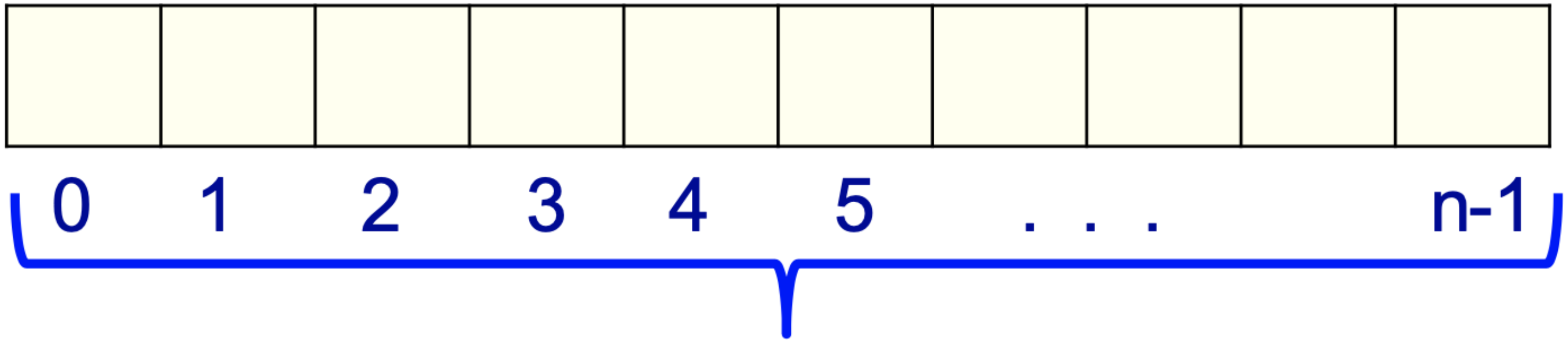
- We need a way to store a list of contacts in the AddressBook class.
- A data structure that can be used for this purpose is an **array**:
  - An array stores a collection of values in adjacent memory locations.



Conceptual 2D Representation of Memory

Arrays

- An array stores a collection of values in **adjacent** memory locations
- Each value stored in an array has a unique index
  - Array indices in Java start at Zero: 0, 1, 2, ..., n-1



Indices for an array storing n values

Arrays (cont.)

- In Java, arrays are objects, so they are **referenced** with **non-primitive variables**.
  - An array is declared using square brackets: int[] arr1;
    - arr1 is a **reference** variable (address location) to an array storing integer values.
- Example:
  - int[] numbers; // Declaration of an array of integers
  - numbers = new int[5]; // Creates an array of 5 integers

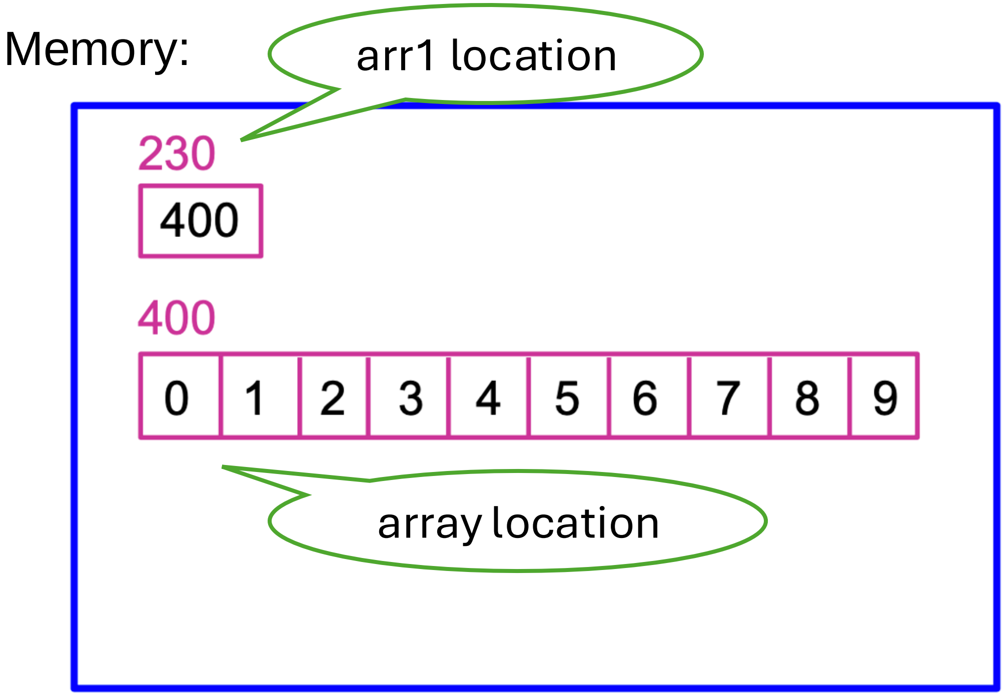
Note: In Java, arrays are treated as **objects**, meaning they are not just a data collection but also have <u>attributes</u> (like length) and <u>associated methods</u>.

Arrays (cont.)

- Example:

```
int[] arr1; // Declaration of an array of integers
arr1 = new int[10]; // Creates an array of 10 integers
for (int i = 0; i < 10; ++i) // looping
    arr1[i] = i;
```

- After executing this code, the memory of the computer and the symbol table will look like this



Variable	Type	Address
arr1	int[]	230

AddressBook Class - Instance Variables

- To store the list of contacts, we will use an array, so the first instance variable of this class will be

Type

```
private Person[] contactList;
```

- We will use a second instance variable to store the number of contacts that have been stored in the array:

```
private int numContacts;
```

- Note that the number of contacts and the length of the array do not need to be the same. The length of the array is the maximum number of contacts that we can store in it.

Keyword Final

- We will use a third instance variable that will be used to specify the length of the array:

```
private final int DEFAULT_MAX_CONTACTS = 10;
```

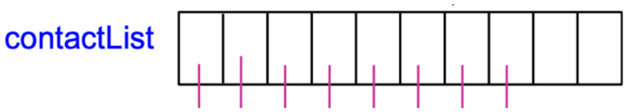
- The keyword final is used to specify a constant, i.e., a variable whose value cannot be modified.
- So, for example, the following code fragment is invalid:

```
private final int DEFAULT_MAX_CONTACTS = 10;
DEFAULT_MAX_CONTACTS = 5;
```

AddressBook Class - Methods

- We need a constructor and methods for **adding** a new contact and for **removing** a contact.
- We will define two different constructors for this class:

```
/* This constructor creates an array of a specified size */
public AddressBook(int maxNumber) {
 contactList = new Person[maxNumber];
 numContacts = 0;
}
```



```
/* This constructor creates an array of default size */
public AddressBook() {
 contactList = new Person[DEFAULT_MAX_CONTACTS];
 numContacts = 0;
}
```

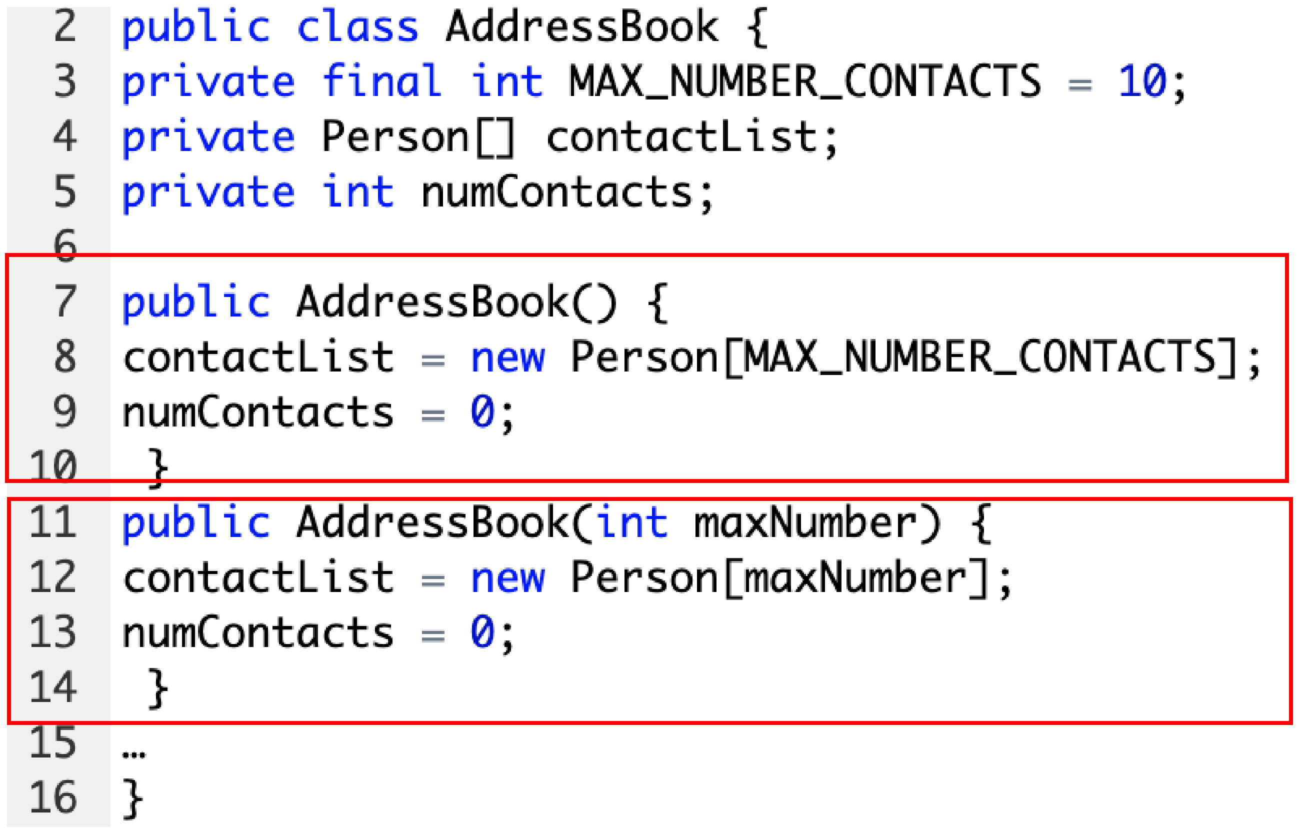
AddressBook Class - Methods

- Having two methods with the same name within a class is called overloading.
- Two methods can have the same name as long as they have different signatures.
- A signature consists of the name of a method + the number and types of its parameters.
- Note that the two presented constructors have different signatures:

AddressBook(int)	one int parameter
AddressBook()	no parameters

AddressBook Class

- Here is a partial code for the AddressBook Class:



```java
public class AddressBook {
private final int MAX_NUMBER_CONTACTS = 10;
private Person[] contactList;
private int numContacts;

public AddressBook() {
contactList = new Person[MAX_NUMBER_CONTACTS];
numContacts = 0;
 }
public AddressBook(int maxNumber) {
contactList = new Person[maxNumber];
numContacts = 0;
 }
…
}
```

Question!

What is the significance of using the private visibility modifier for the contactList and numContacts instance variables in the AddressBook class?

A) It prevents other classes from accessing or modifying them directly, ensuring data encapsulation and integrity.

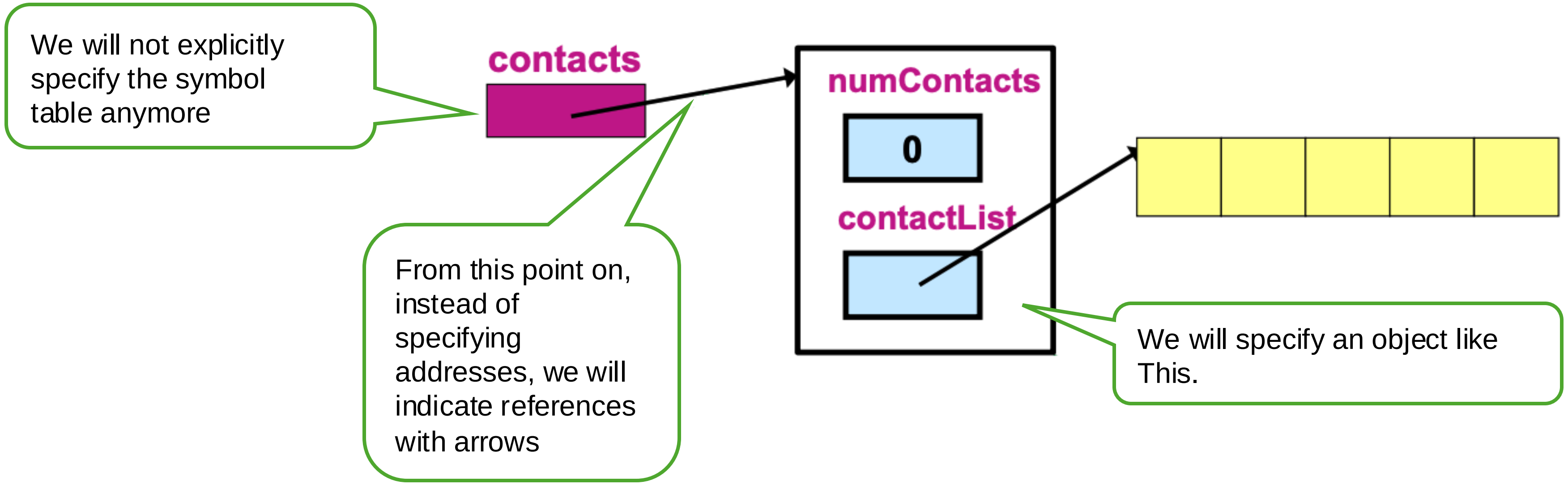
B) It allows the AddressBook class to access these variables from any package.

C) It makes these variables accessible only to methods in subclasses of AddressBook.

D) It restricts access to these variables to only the main method.

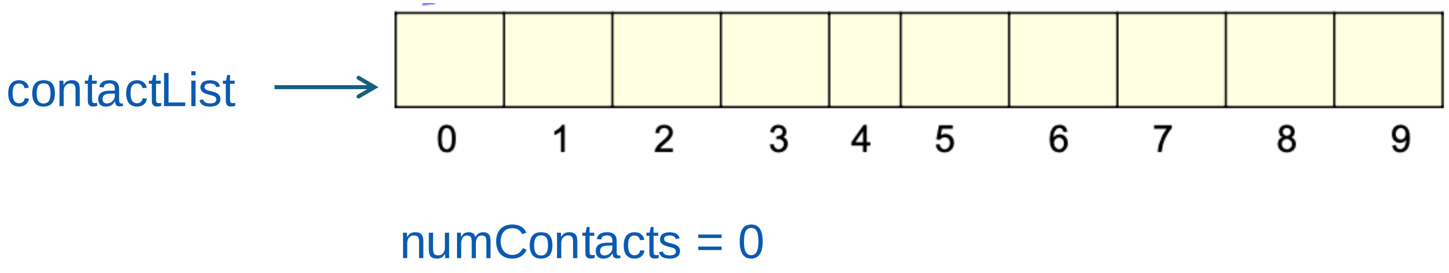
Example: AddressBook Object

- contacts = new AddressBook(5);
- After executing this code, the new object looks like this:


We will not explicitly specify the symbol table anymore
contacts
numContacts
0
contactList
From this point on, instead of specifying addresses, we will indicate references with arrows
We will specify an object like This.

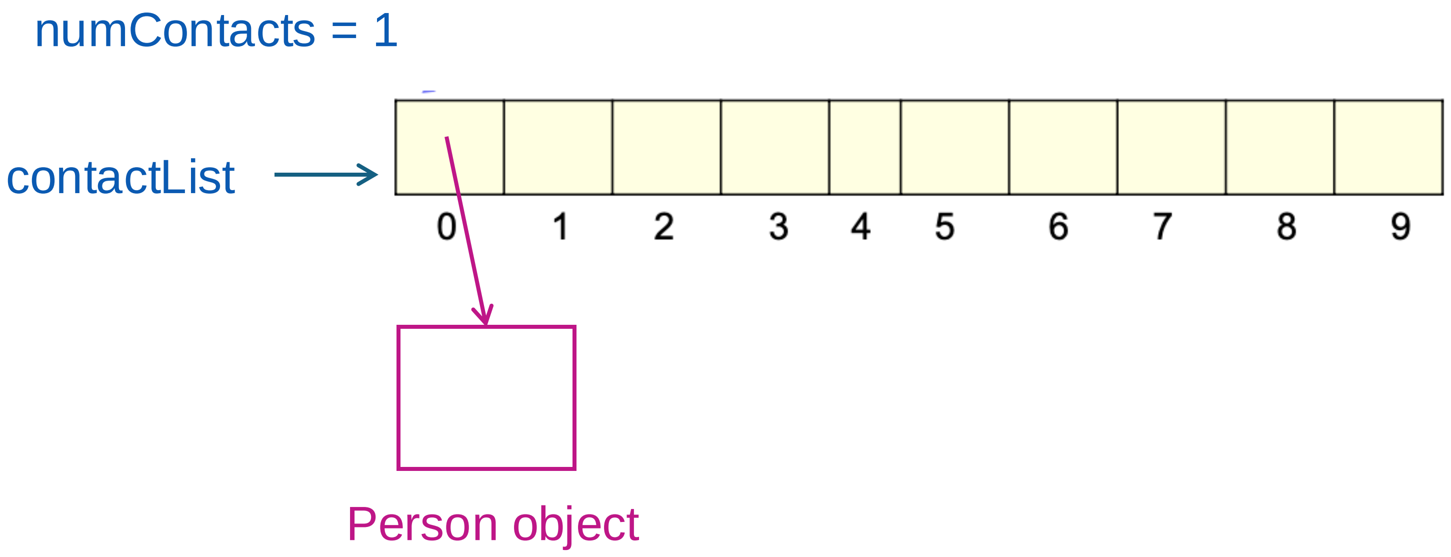
Adding a New Contact

- To add a new contact to contactList, we need to consider two cases:
    - If contactList is **not full**, i.e., numContacts < length of contactList , then we simply add the new contact to the **END** of the array and **increase** the value of numContacts by one.
    - If contactList is **full**, we will need to extend it to fit (will be addressed later).
- Note that numContacts is both the number of contacts in the array and the first index of the array that does not store a contact, ??.



Adding a New Contact (cont.)

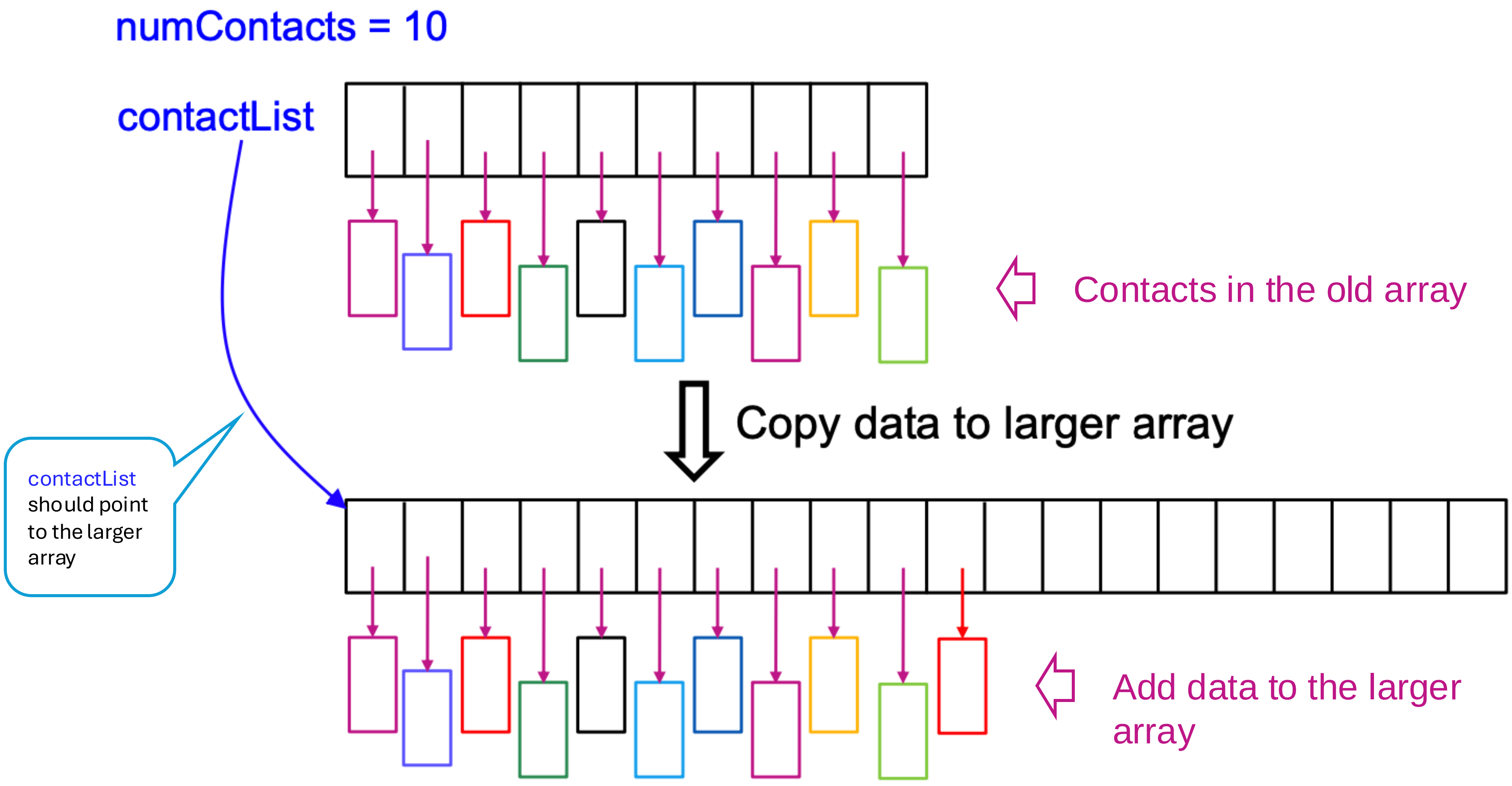
- After adding a NEW Person object to the initially empty array, the array will look like this:



- A new Person object can now be stored in index 1 (note that numContacts = 1), then on index 2, and so on.

Adding a New Contact (cont.)

- However, if contactList is **FULL**, i.e., numContacts = length of contactList, then we cannot add new contacts to the array, as arrays in Java have a fixed size. This is different from **Python** lists.
- In this situation, we need to create a new, larger array, **copy** the information from the old array to the **new** one, and then add new contacts to the larger array.



numContacts = 10

contactList

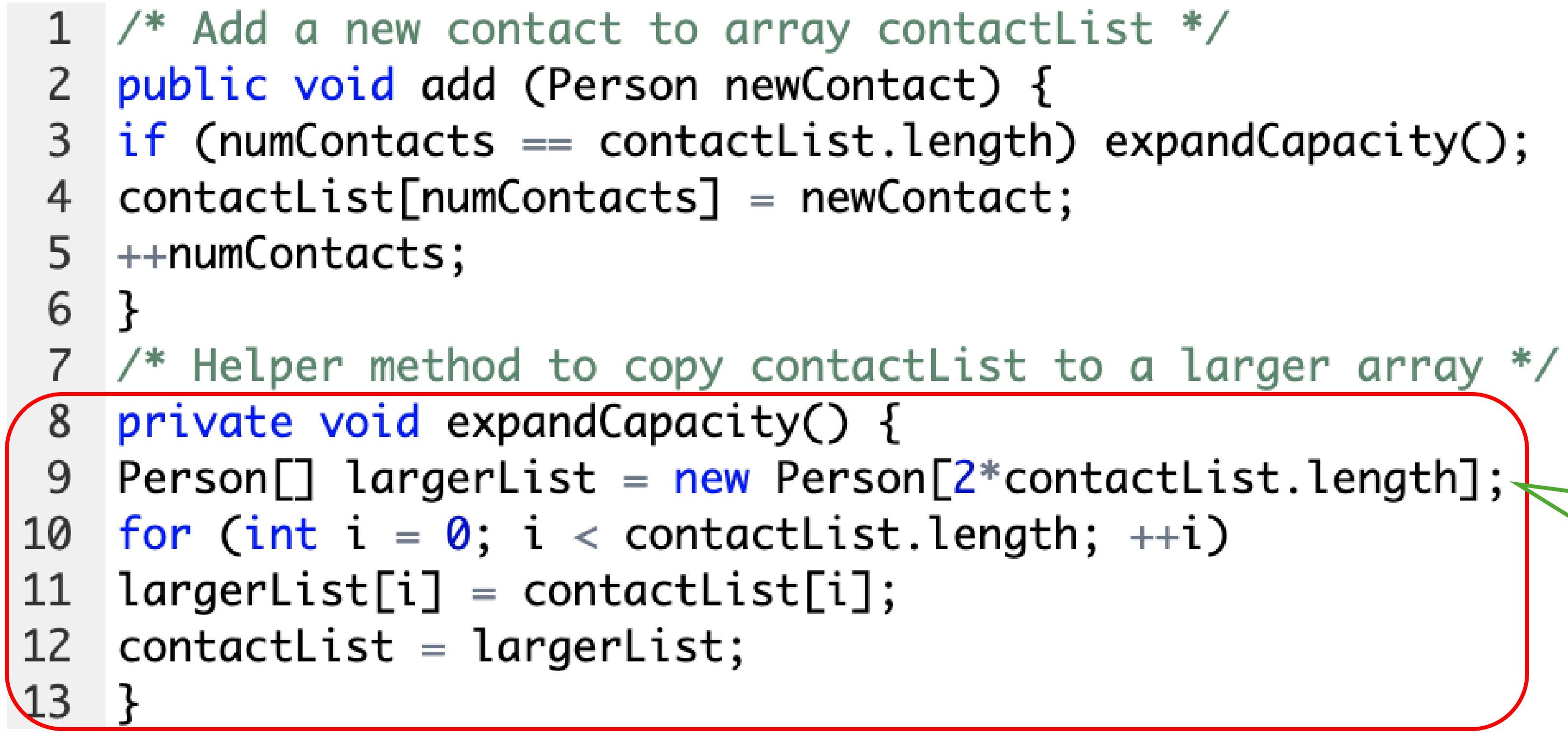
Contacts in the old array

Copy data to larger array

contactList should point to the larger array

Add data to the larger array

Expanding an Array



```java
/* Add a new contact to array contactList */
public void add (Person newContact) {
if (numContacts == contactList.length) expandCapacity();
contactList[numContacts] = newContact;
++numContacts;
}
/* Helper method to copy contactList to a larger array */
private void expandCapacity() {
Person[] largerList = new Person[2*contactList.length];
for (int i = 0; i < contactList.length; ++i)
largerList[i] = contactList[i];
contactList = largerList;
}
```

Double the space

Question!

What happens if you attempt to add a contact when the contactList array in the AddressBook class is full?

A) A runtime error occurs because Java doesn't allow adding elements beyond the array's fixed size.

B) The contact is automatically added, and the array's size is doubled.

C) A new, larger array is created, and the existing contacts are copied to this new array.

D) The numContacts variable decreases by one, and no new contact is added.

Removing a Contact

We now design a method that removes a given target contact from contactList. This method will work as follows:

- If target is in contactList, then it will be removed from the array and the method will return the value true so we know that the target was found and removed.
- If the target is not in contactList, the method will simply return the value FALSE as an indication that the target was not in the array.

We will use this problem as an example of how to design algorithms using **pseudocode**.

Pseudocode

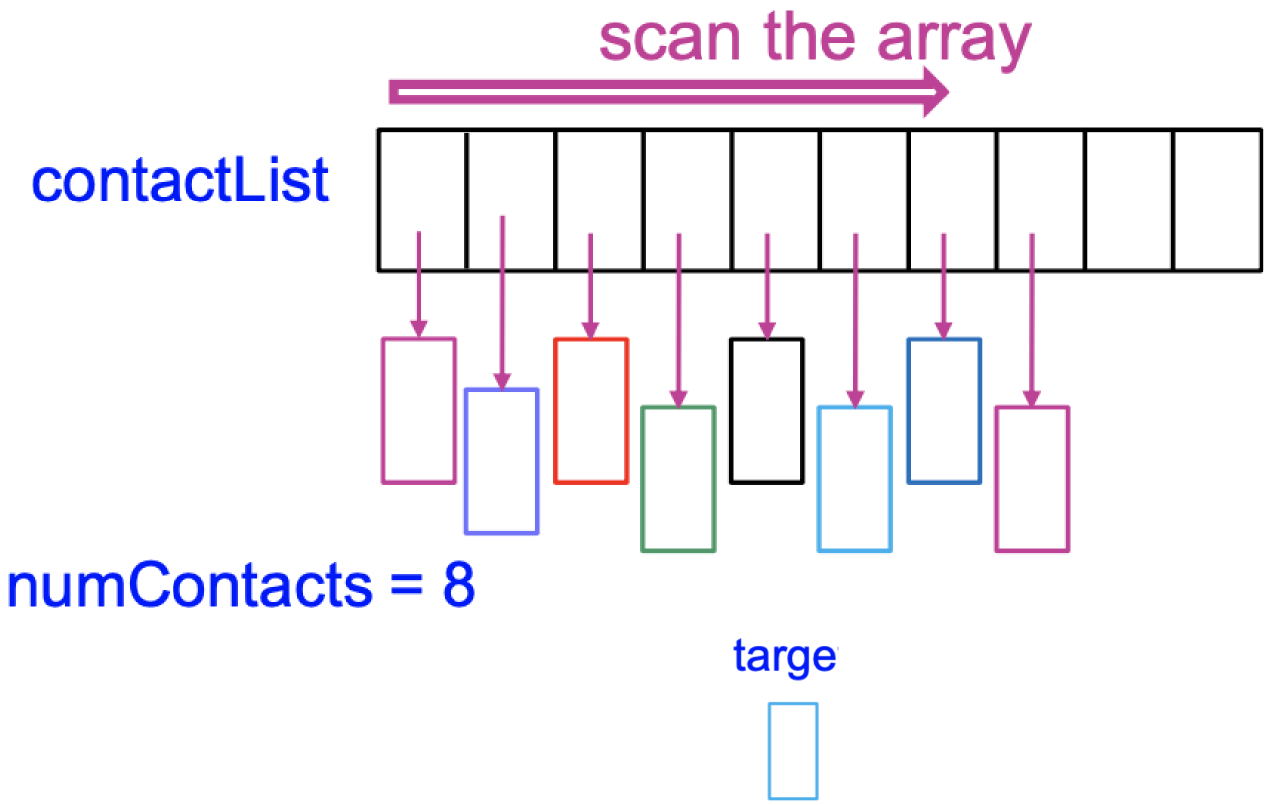
- The advantage of writing an algorithm in pseudocode is that we can concentrate on designing the steps that the algorithm needs to perform to achieve a desired task without having to think about how to express the algorithm in the correct Java syntax.
- Once we have designed a correct algorithm for a problem in pseudocode, translating it into Java is a somewhat mechanical process.
- Writing algorithms first in pseudocode and then translating them into Java makes it easier to design programs.

Pseudocode

- The beauty of pseudocode is that there is no fixed syntax or rigid rules for it.
- Pseudocode is a mixture of English and programming-like statements.
- Each programmer designs their own version of pseudocode.
- A programmer just needs to ensure that pseudocode is understandable to other people and that it is detailed enough that translation into Java or other programming language is simple.
- There should be an (almost) one-to-one correspondence between lines of pseudocode and lines of Java code.

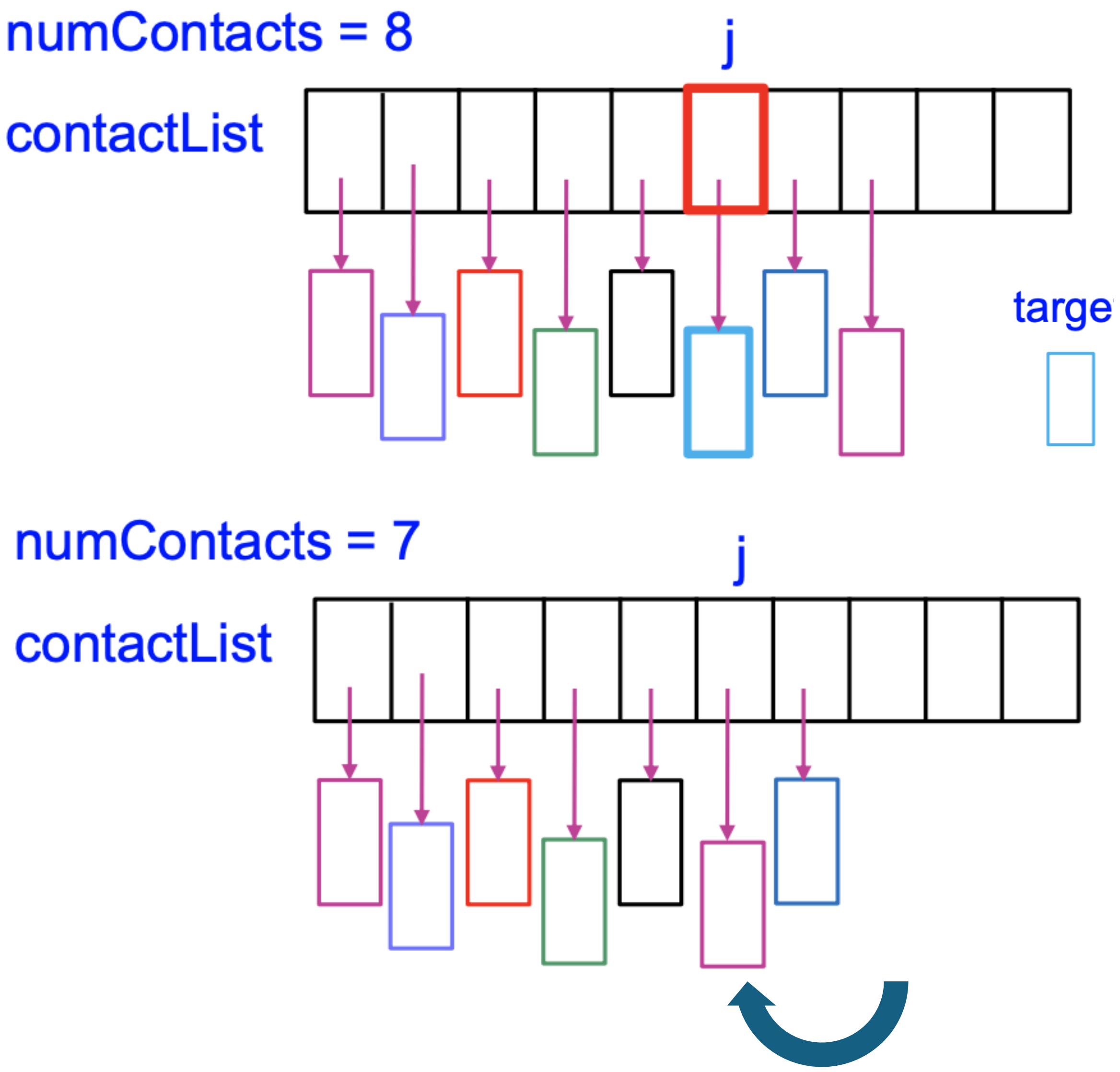
Removing a Contact (cont.)

- First, we need to look for the target in the array:
  - Scan the array starting at index 0 and compare each value stored in the array with the target.
  - If the end of the array is reached, then this means that the target is not in it.
  - Otherwise, we find the index of the target
- The above algorithm is **called linear search**.



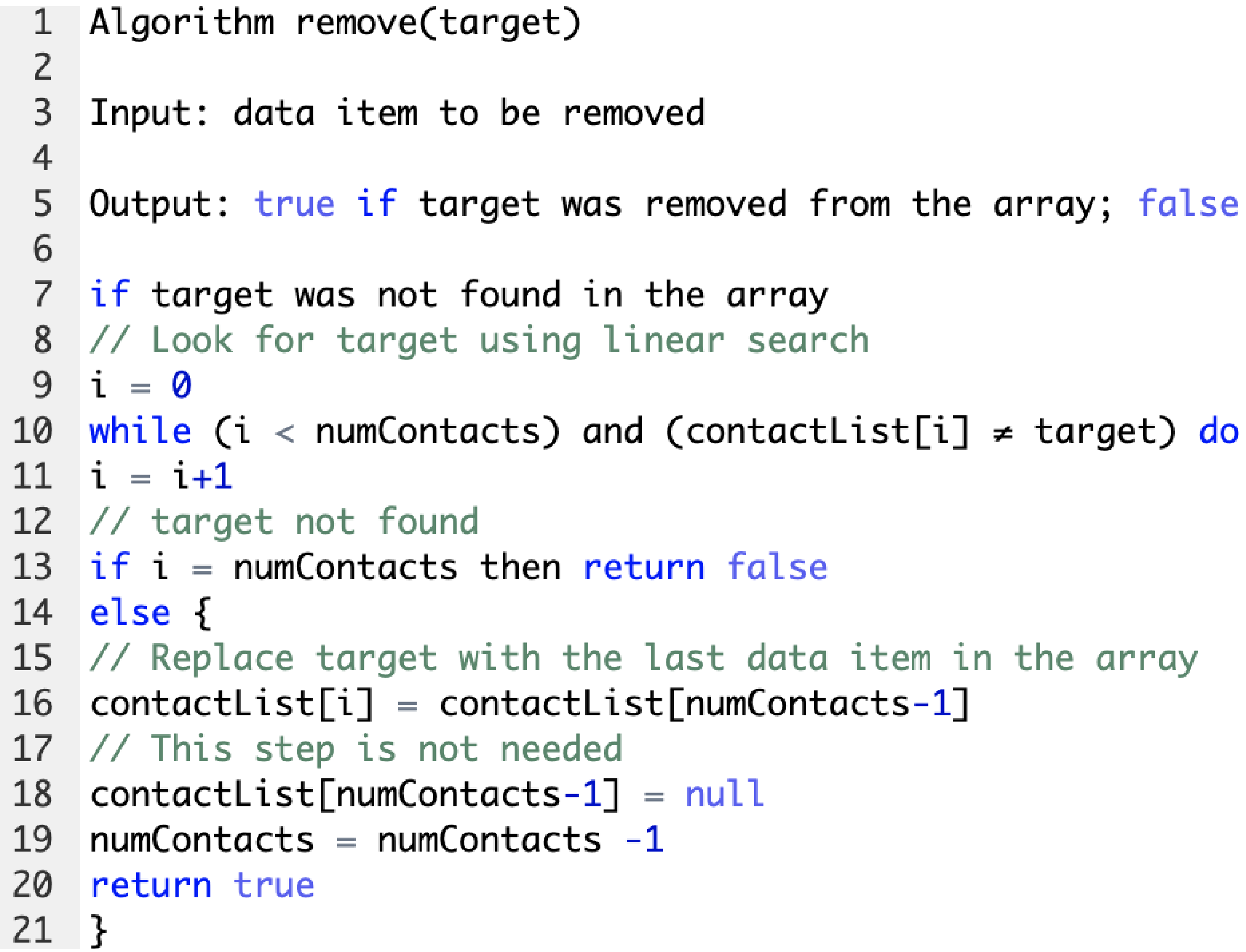
Removing a Contact (cont.)

- If the target was found at index j, we replace it with the last value in the array, as values in the array must appear in adjacent positions; finally, we decrease numContacts.
- We can also shift to the left all values that appear at indices > j. For any problem there are many ways to solve it.



Algorithm Remove

- This algorithm performs a **linear search** to find the target in the contactList array.
- If the target is found, it replaces that element with the last element in the array, sets the last element to NULL (though this step isn't strictly necessary), and decrements numContacts to maintain the array's size.
- If the target is not found, it returns FALSE.
- The algorithm efficiently removes the target without shifting all elements, reducing the **time complexity**.

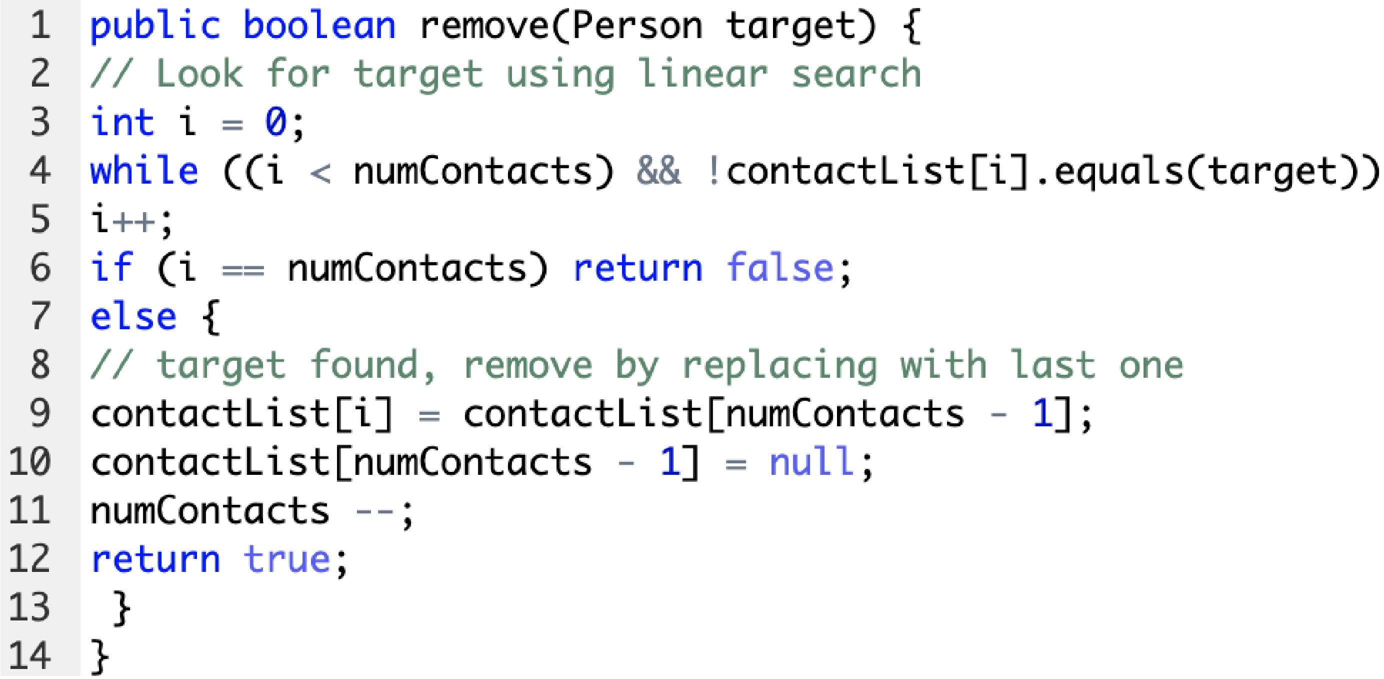


```
Algorithm remove(target)

Input: data item to be removed

Output: true if target was removed from the array; false

if target was not found in the array
// Look for target using linear search
i = 0
while (i < numContacts) and (contactList[i] ≠ target) do
i = i+1
// target not found
if i = numContacts then return false
else {
// Replace target with the last data item in the array
contactList[i] = contactList[numContacts-1]
// This step is not needed
contactList[numContacts-1] = null
numContacts = numContacts -1
return true
}
```

Algorithm Remove – Pseudocode

Algorithm Implementation

- Here is the translation of the pseudocode to Java:

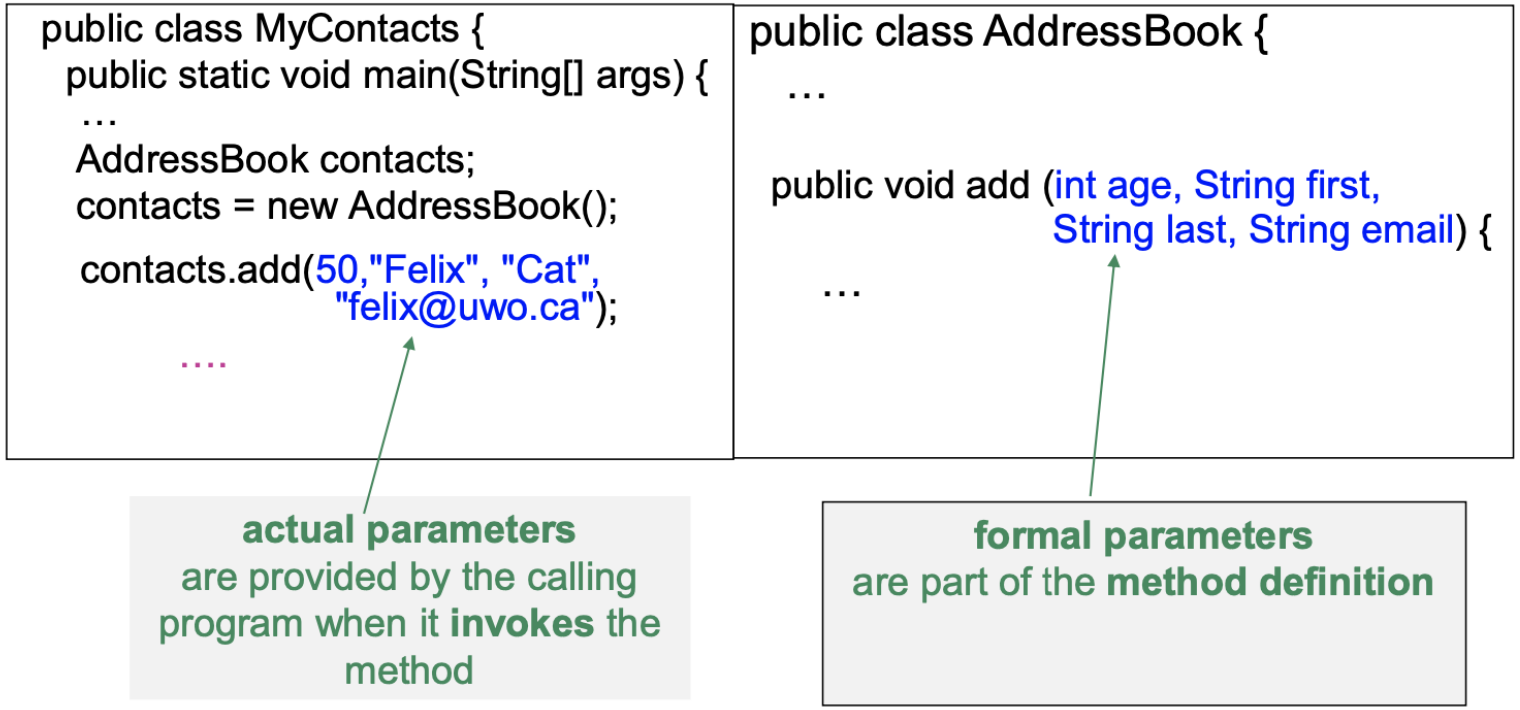


```java
public boolean remove(Person target) {
// Look for target using linear search
int i = 0;
while ((i < numContacts) && !contactList[i].equals(target))
i++;
if (i == numContacts) return false;
else {
// target found, remove by replacing with last one
contactList[i] = contactList[numContacts - 1];
contactList[numContacts - 1] = null;
numContacts --;
return true;
 }
}
```

Passing Parameters

- Why are algorithms and methods written with parameters?
- So that the methods can be more general. We can invoke methods with *different values* passed in as parameters
- The variable in the parameter list in the method definition is known as a formal parameter.
- When we invoke a method with a parameter, the value passed as the parameter is called the actual parameter.

Passing Parameters



```
public class MyContacts {
  public static void main(String[] args) {
   …
   AddressBook contacts;
   contacts = new AddressBook();
   contacts.add(50,"Felix", "Cat",
                  "felix@uwo.ca");
         ….
```

```
public class AddressBook {
  …

  public void add (int age, String first,
                   String last, String email) {
     …
```

actual parameters are provided by the calling program when it **invokes** the method

formal parameters are part of the **method definition**

- When the add method is executed, the value of each actual parameter is passed by value to the corresponding formal parameter variable.

Thank you