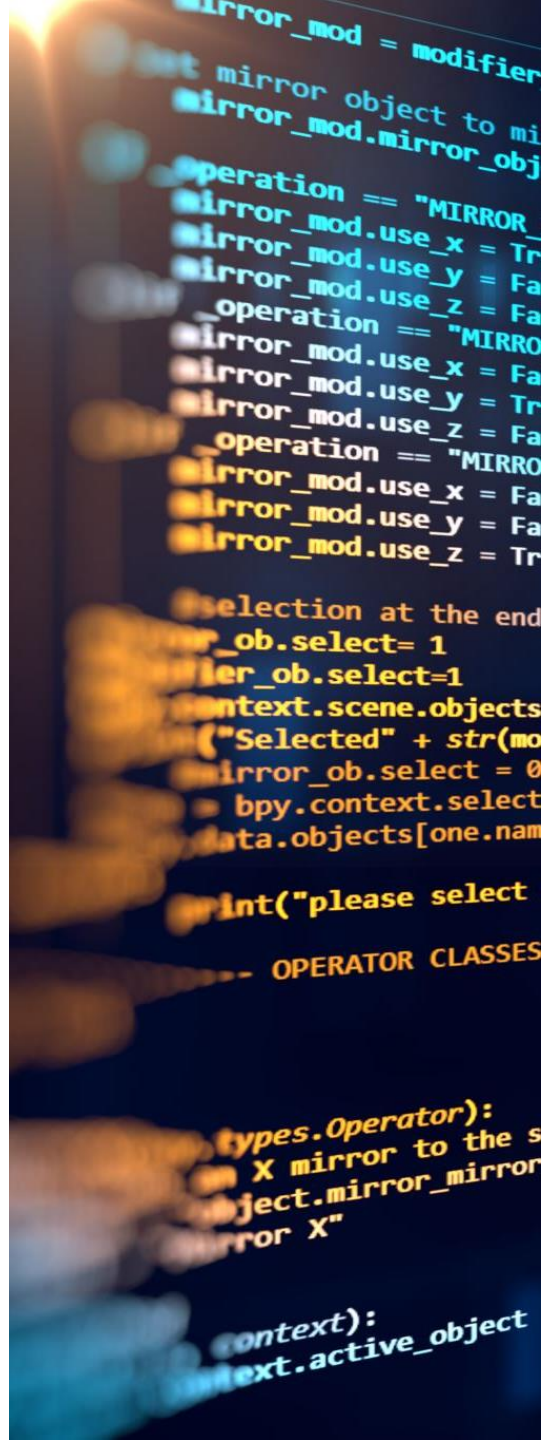
CS 1027
Fundamentals of Computer Science II

Object Oriented Design

Ahmed Ibrahim

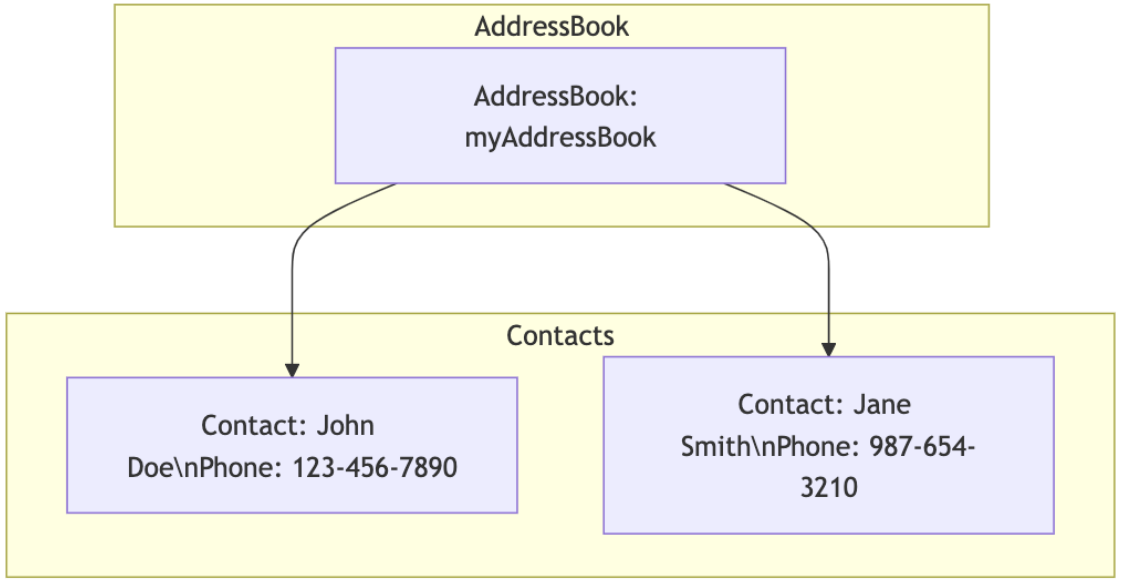


Recap

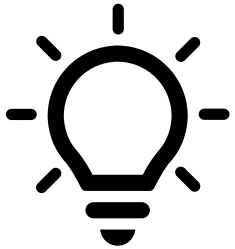
- **Object-Oriented Design**: Focus on designing programs using objects and classes.
- **Computer Model & Memory**: Programs and data are stored in memory; machine language is binary.
- **Programming Languages**: High-level languages (e.g., Java) are translated to machine code via compilers.
- **Program Design**: Follows steps: Specification, Design, Implementation, Testing, and Verification.
- **Good Design**: Plan solutions with pseudocode, breaking down complex problems into simpler tasks.
- **OOP Principles**: Use objects and classes for modular design, encapsulating data and behaviors.
- **Modularity & Encapsulation**: Keep modules independent; hide internal details, exposing only necessary parts.
- **Abstraction**: Modules interact through public methods while keeping internal data private.
- **Objects and Classes**: Classes define the structure and behavior of objects; use public methods to control access to private data.

Address Book Project

- Say we want to keep track of the contact information of friends and relatives.
- We wish to write a program that keeps a list of our contacts, allows us to add a new contact, removes a contact from the list, and prints information about all our contacts.



Address Book Project



- Say we want to keep track of the contact information of friends and relatives.
- We wish to write a program that keeps a list of our contacts, allows us to add a new contact, removes a contact from the list, and prints information about all our contacts.

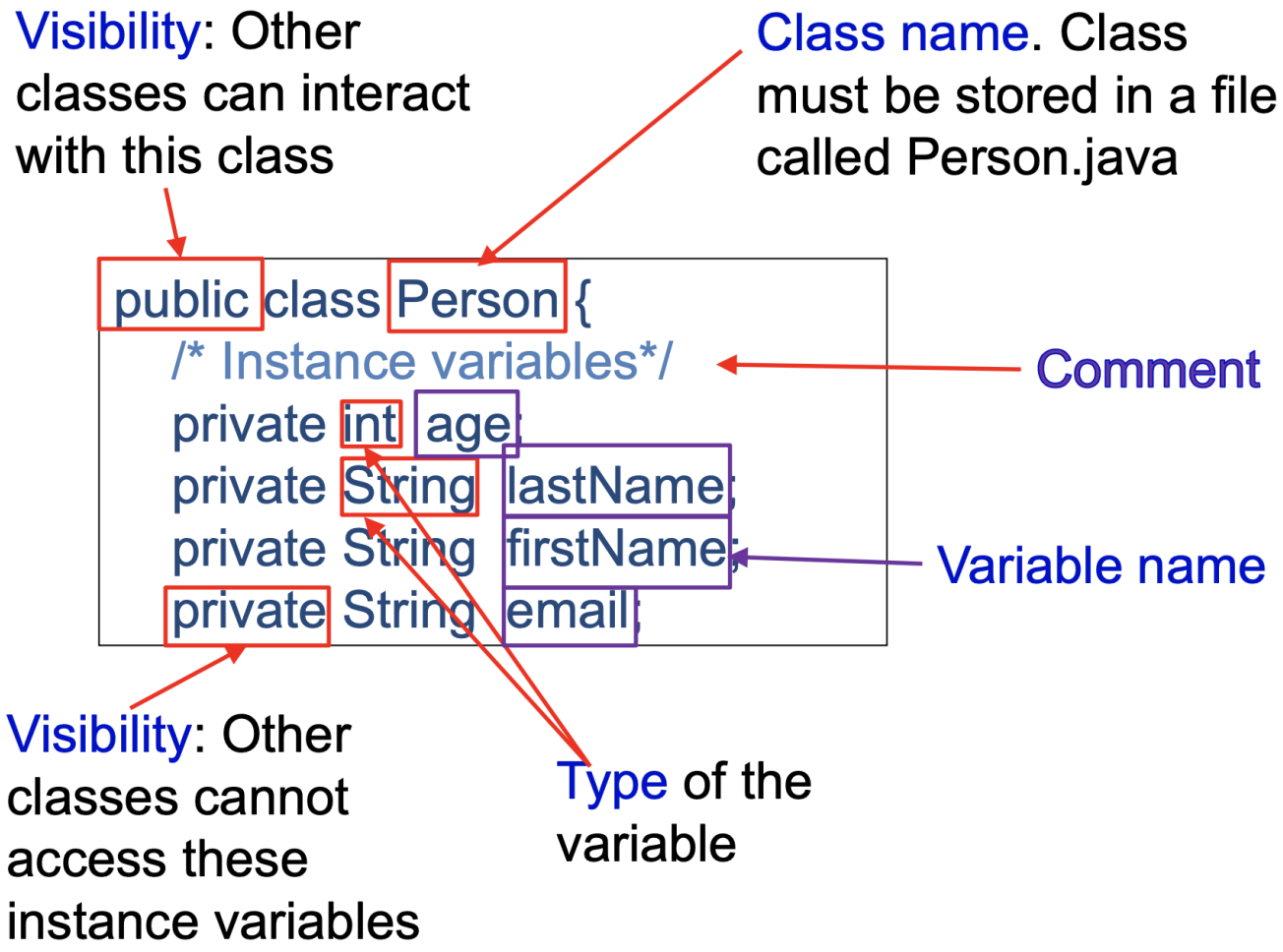
Features that need implementation (**functionality**)

Address Book Project (cont.)

- We need to determine how to **split the problem** into simpler ones, which means determining what modules or objects we need for the program.
- Since we need to keep track of contact information of individuals, for each individual we need an object to store that information.
- We use a class called **Person** that models the information about one individual in our address book.

Person Class

- Recall that we need to specify each class's attributes and methods.
- Attributes or instance variables
  - Let's say we're creating a class to represent a person. For each person, we'd want to store their age, email address, and first and last name. This is a practical example of how we specify attributes for a class.



Variable Declarations

- In Java, we need to declare variables before we can use them. (This is different from Python, where variables do not need to be declared)
- A variable declaration contains up to four parts (some are optional):
  - Visibility (public, private, protected – will talk about protected later)
  - Modifier (static, final – will talk about these later)
    - The static modifier makes a variable or method belong to the class
    - The final modifier prevents changes
  - Data type (primitive or non-primitive)
  - Name

Good programming practice:

Declare instance variables as private so they cannot be directly and arbitrarily modified by other classes. This avoids errors that are otherwise very difficult to find.

Method Declarations

- A method declaration contains several parts (some are optional):
  - visibility (public, private, protected)
  - modifier (static, abstract, we will talk about these later)
    - An abstract method is a method that is declared but not implemented in the abstract class.
  - return type
  - name
  - formal parameters ("arguments")
  - exception list (we will talk about this later)
    - An exception is an event that occurs during the execution of a program and disrupts the normal flow of instructions.
  - body: the code of the method

Methods

- Methods contain the code of the objects and
- implement the functionality of an object.
- Two important kinds of methods are:
  - accessor methods or getters: They allow retrieving the values of private instance variables
  - modifier methods or setters: They allow modifying the values of private instance variables

Methods

```
public class Person {
    /* Instance variables*/
    private int age;
    private String lastName;
    private String firstName;
    private String email;
    // Method declarations
    public String getEmail () {
        return email;
    }
    public void setEmail (String newMail) {
        email = newEmail;
    }
    . . .
}
```

getters

setters

Methods

- Methods contain the objects' code and implement the **object's functionality**.
- An object must be created before we can store information in its instance variables or execute the code in its methods.
- To create an object in Java, we use the **NEW** operator.

Constructor

- A constructor is a special method that is called automatically when an object is created with the new operator.
- Its purpose is to **initialize** the instance variables of an object when the object is created.
- In Python, we use the special method __init__ to do the job of a constructor.
- In Java, **a constructor has the same name as the class name**.

Here is the full code for the Person class:

```
public class Person {
    private int age;
    private String firstName, lastName, email;
    public Person (int theAge, String first, String last, String theEmail) {
        age = theAge;
        firstName = first;
        lastName = last;
        email = theEmail;
    }
    public String getEmail() {
        return email;
    }
    public void setEmail(String newEmail) {
        email = newEmail;
    }
}
```

Constructor

getters

setters

Address Book Project (cont.)

- Suppose that we wish to write a program that implements an address book to maintain the contact information of a group of people:
  - The program should keep a list of contacts storing information about any number of persons.
  - The program should allow us to add a new contact to the list.
  - The program should allow us to remove a contact from the list.

Address Book Project (cont.)

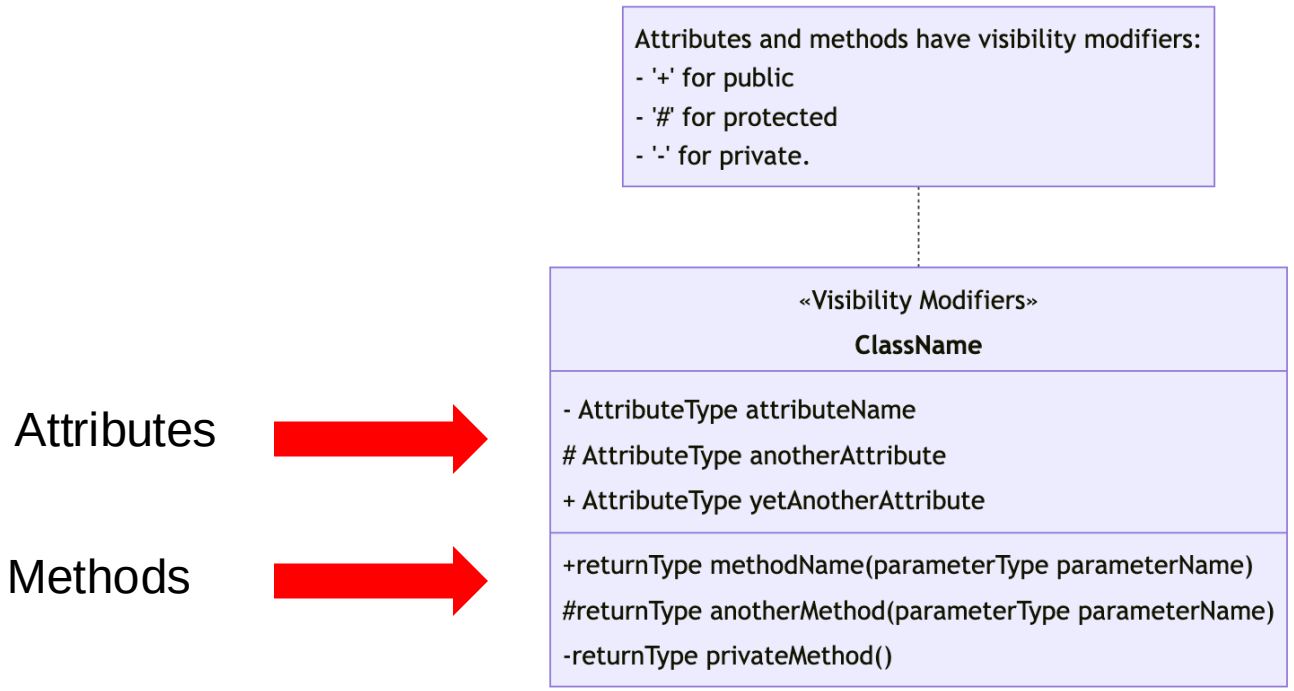
- In this application, we can use the Person class we designed.
- An object of this class stores one person's contact information.

Person Class

Attributes or instance variables:
private int age;
private String lastName
private String firstName
private String email

Methods:
public Person(int theAge, String first, String last, String TheEmail)
public void setEmail (String email)
public String getEmail()
public boolean equals(Person other)

"Specification"



Modifier	Same Class	Same Package	Subclasses	Other Packages
public (+)	Yes	Yes	Yes	Yes
protected (#)	Yes	Yes	Yes	No
private (-)	Yes	No	No	No

Question!

What is the main benefit of using private instance variables in a class?

A) To make the code more readable by other classes.

B) To ensure that only the methods within the class can access or modify the variables, maintaining data integrity.

C) To allow subclass methods to directly modify the variables.

D) To make the instance variables visible to the entire program.

Question!

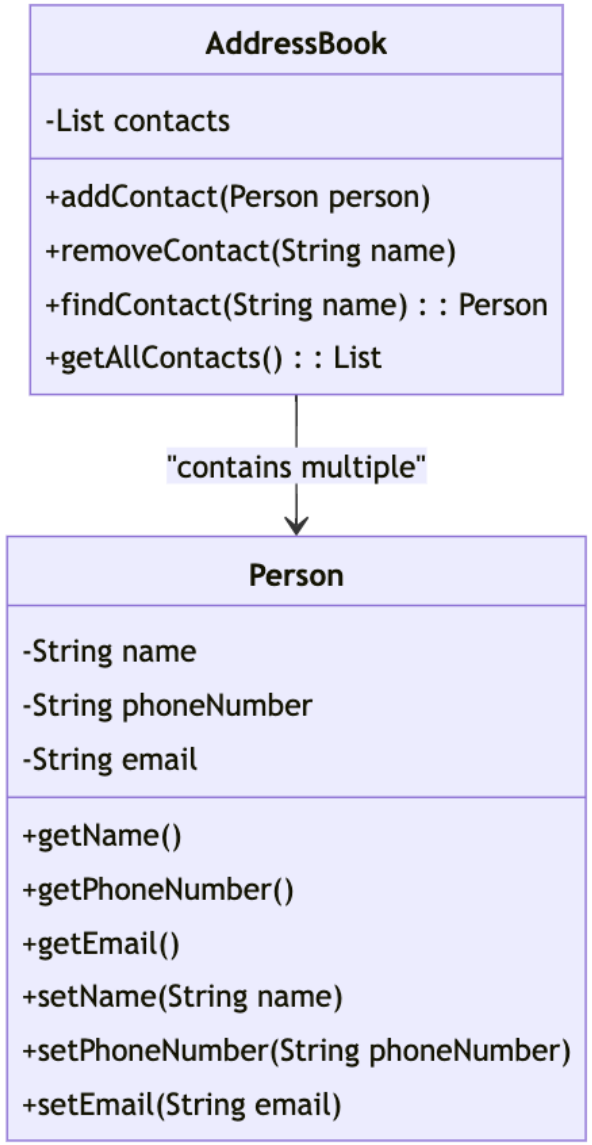
- **Which of the following is the correct way to declare a private integer instance variable in Java, as mentioned in the lecture?**

A) **private** final int numContacts;

B) int **private** numContacts;

C) **private** int numContacts;

D) final **private** int numContacts;



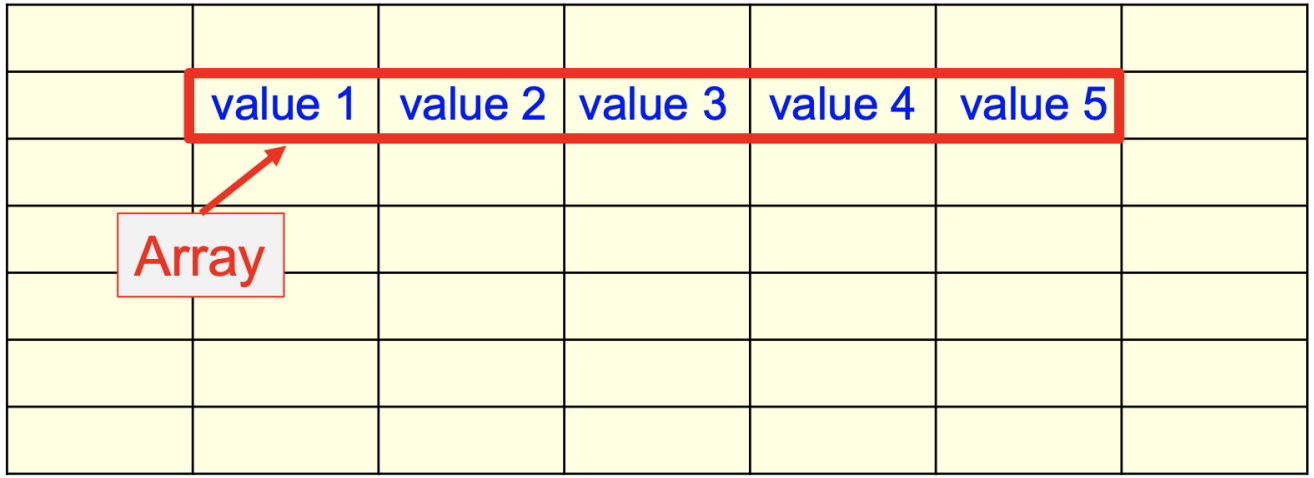
Address Book Project (cont.)

The program will then consist of two modules:

1. A module to store one person's contact information.
   - This will be implemented using the Person class.
2. A module to store the list of contacts.
   - For this module, we will design a new class called AddressBook.
   - To design this class, we need to determine two things:
     - The instance variables of the class
     - The methods of the class

AddressBook Class

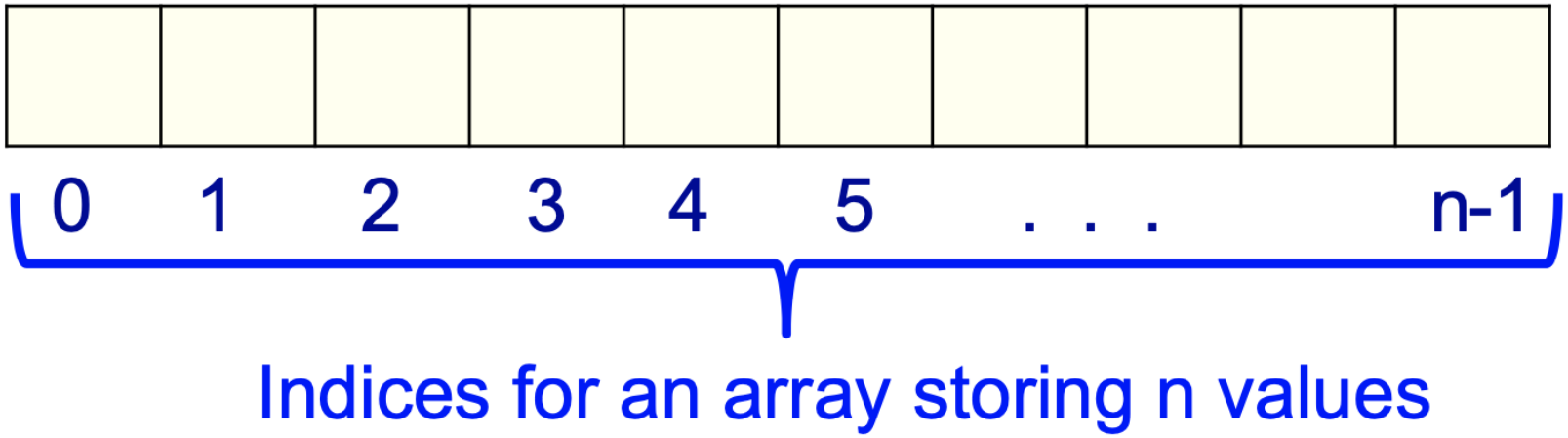
- We need a way to store a list of contacts in this class.
- A data structure that can be used for this purpose is an **array**:
  - An array stores a collection of values in adjacent memory locations.



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



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 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

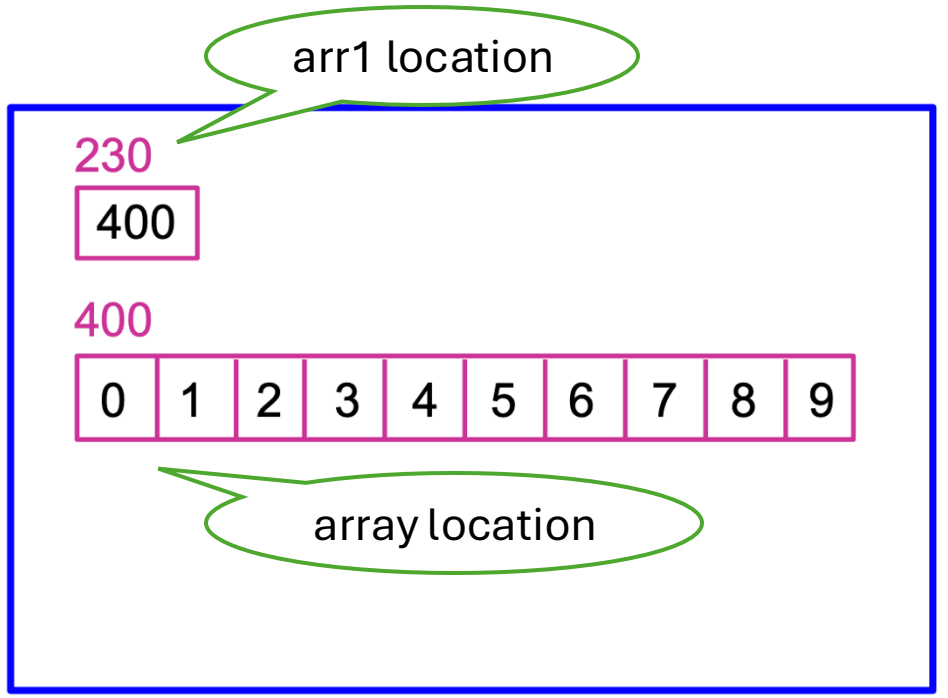
Note: In Java, arrays are treated as **objects**, meaning they are not just a data collection but also have attributes (like length) and associated methods.

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

  ```
  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


thank you