Object Oriented Programming with Java II

Dr. Mohamed K. Hussein

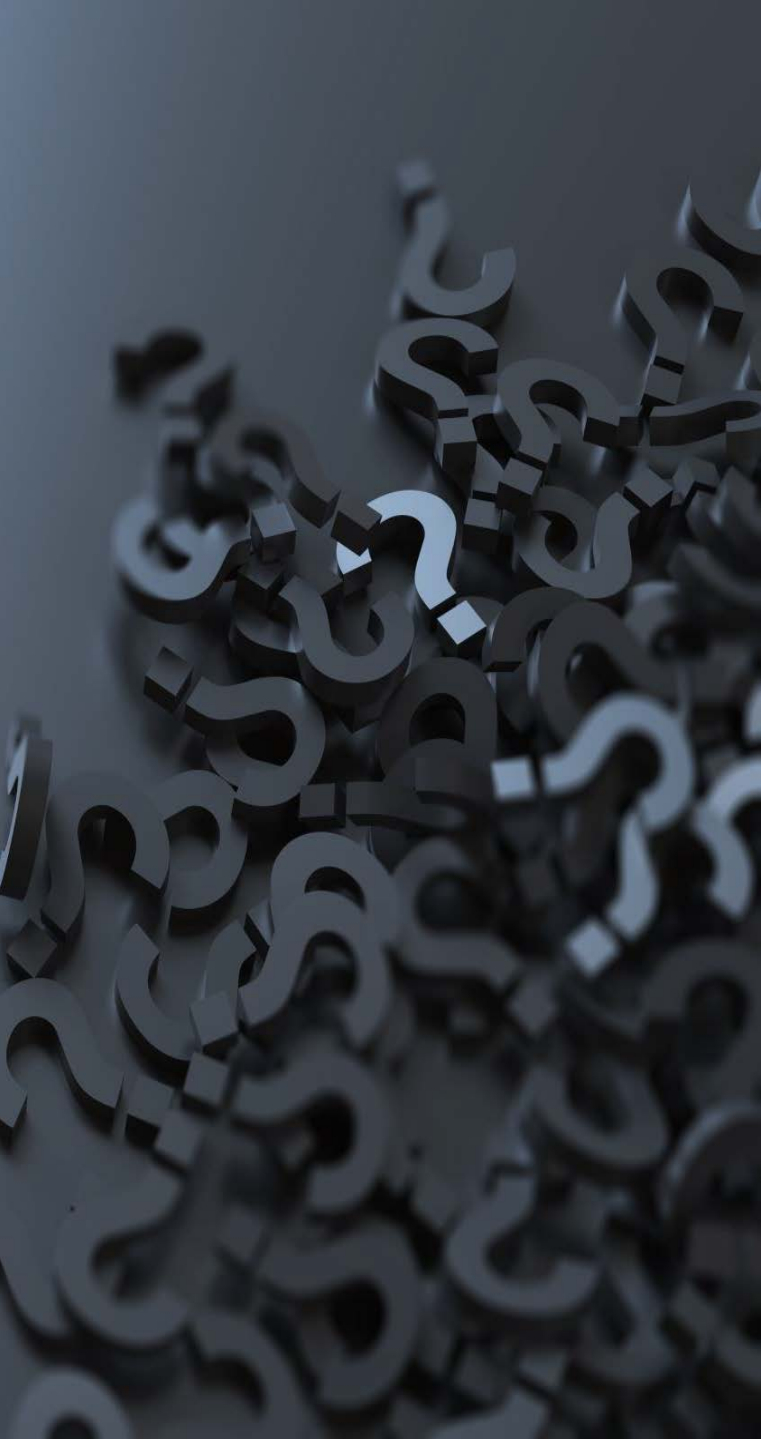
Lecture 2

Associate Prof., Computer Science department,

Faculty of Computers & Information Science,

Suez Canal University

Email: m_khamis@ci.suez.edu.eg



Lecture outcomes

- What are accessor and mutator methods.
  - How they can be used in conjunction with encapsulation.
- What is constructor overloading.
- How to represent a class using class diagrams.
  - Attributes, methods and access permissions.
- Scoping rules
- Addresses & References

Viewing & Modifying Attributes Accessor & Mutators

1) **Accessor methods**: 'get()' method

   - Used to determine the current value of an attribute

2) **Mutator methods**: 'set()' method

   - Used to change an attribute (set it to a new value):

```
public int getAge()    {
    return(age);
}

public void setAge(int anAge) {
    age = anAge;
}
```

Example

```
public class Person {
    private int age;
    public Person() {
        age = 0;
    }
    public int getAge() {

        return(age);
    }

    public void setAge(int anAge){
        age = anAge;
    }
}
```

```
public class MainClass {
    public static void main(String [] args){
        Person ahmed = new Person();
        System.out.println(ahmed.getAge());
        ahmed.setAge(21);
        System.out.println(ahmed.getAge());
    }
}
```

Constructors Overloading

- Constructors are used to initialize objects
  - Set the attributes as the object is created.
- Different versions of the constructor can be implemented with different initializations
  - e.g., one version sets all attributes to default values while another version sets some attributes to the value of parameters.

Constructor overloading

- Method overloading, same method name, different parameter list.

```
public Person(int anAge) {
    age = anAge;
}
```

```
public Person() {
    age = 0;
}
```

// Calling the versions (distinguished by parameter list)

```
Person p1 = new Person(100);
```

```
Person p2 = new Person();
```

Example

```
public class Person {
    private int age;
    private String name;
    public Person() {
        System.out.println("Person()");
        age = 0;
        name = "No-name";
    }
    public Person(int anAge) {
        System.out.println("Person(int)");
        age = anAge;
        name = "No-name";
    }
    public Person(String aName) {
        System.out.println("Person(String)");
        age = 0;
        name = aName;
    }
    public Person(int anAge, String aName) {
        System.out.println("Person(int,String)");
        age = anAge;
        name = aName;
    }
    public int getAge() {
        return(age);
    }
    public String getName() {
        return(name);
    }
    public void setAge(int anAge) {
        age = anAge;
    }
    public void setName(String aName) {
        name = aName;
    }
}
```

Example

```
public class MainClass {
    public static void main(String [] args) {
        Person p1 = new Person();                    // age, name default
        Person p2 = new Person(21);                  // age=21
        Person p3 = new Person(“person_3");          // name=“person_3”
        Person p44 = new Person(65,"person4");       // age=65, name = “person_4”
        System.out.println(p1.getAge() + " " + p1.getName());
        System.out.println(p2.getAge() + " " + p2.getName());
        System.out.println(p3.getAge() + " " + p3.getName());
        System.out.println(p4.getAge() + " " + p4.getName());
    }
}
```

Method signatures

- Method signatures consist of: the type, number and order of the parameters.
- The signature will determine the overloaded method called:

```
Person p1 = new Person();

Person p2 = new Person(25);
```

Private Keyword

- It syntactically means this part of the class cannot be accessed outside of the class definition.
- You should always do this for variable attributes, very rarely do this for methods.

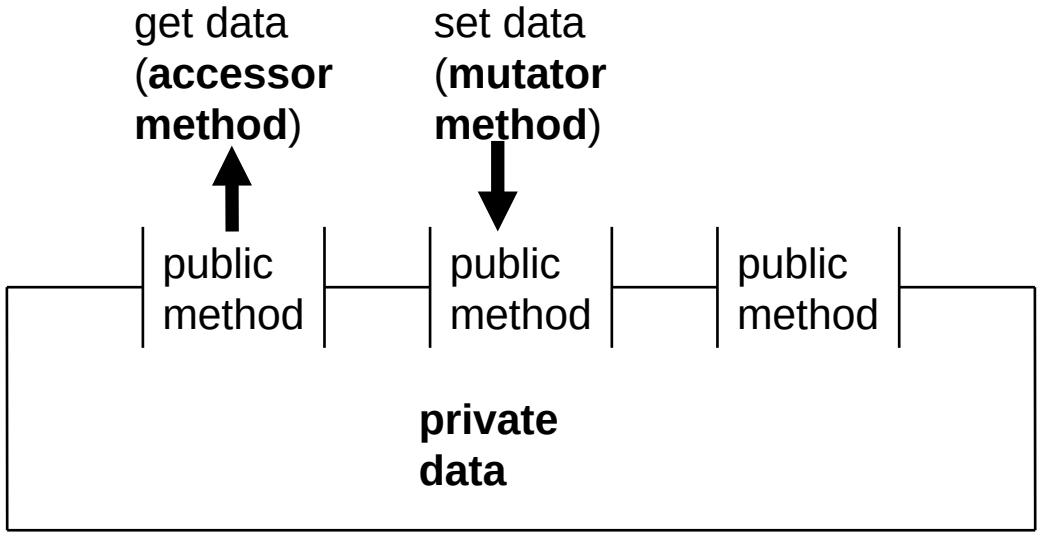
- **Example**

```
public class Person {
     private int age;
     public Person() {
       age = 12;
       //OK – access allowed here
     }
}
```

```
public class MainClass {
     public static void main(String [] args) {
          Person aPerson = new Person();
          aPerson.age = 12;
          // Syntax error: program won't compile!
     }
}
```

Encapsulation/Information Hiding

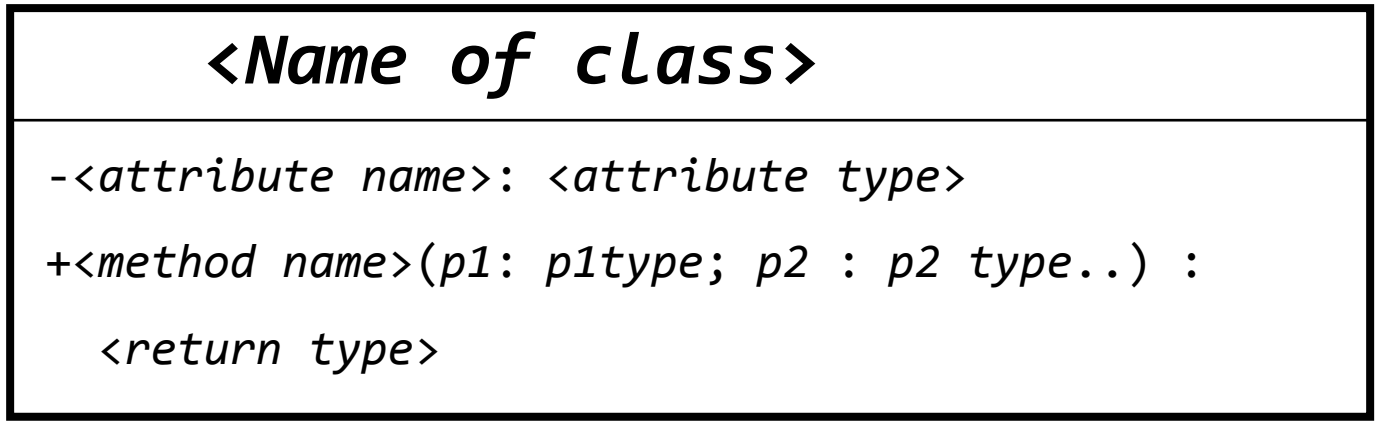
- Protects the inner-workings (data) of a class.
- Only allow access to the core of an object in a controlled fashion (use the *public* parts to access the *private* sections).
  - Typically it means public methods accessing private attributes via accessor and mutator methods.
  - Controlled access to attributes:
    - Can prevent invalid states
    - Reduce runtime errors

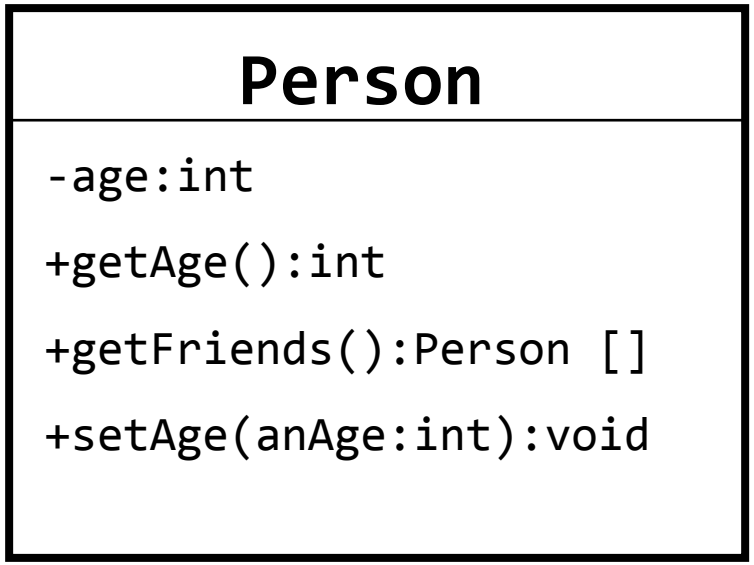


Graphical Summary Of Classes

- UML (Unified modeling language) class diagram
  - Source "*Fundamentals of Object-Oriented Design in UML*" by Booch, Jacobson, Rumbaugh (Dorset House Publishing: a division of Pearson), 2000.
  - UML class diagram provides a quick overview about a class
    - ater, you we'll talk about relationships between classes

UML Class Diagram





Attributes Vs. Locals

- **Attributes**
  - Declared inside a class definition but outside the body of a method

- **Locals**
  - Declared inside the body of a method

- inside a class definition but outside the body of a method

```
public class Person {
    private String [] childrenName = new String[10];
    private int age;
}
```

-
```
public class Person {
     public getAge() {
          int data;
          Scanner in = new Scanner(System.in);
}
```

Scope of Attributes Vs. Locals

- Scope is the location where an identifier (attribute, local, method) may be accessed
  - Scope of attributes (and methods): anywhere inside the class definition
  - Scope of locals: after the local has been declared until the end of closing brace (e.g., end of method body)
- Example:

```
public class Person {
    private String [] childrenName = new String[10];
    private int age;

    public nameFamily() {
        int i;
        for (i = 0; i < 10; i++) {
            childrenName[i] = in.nextLine();
        }
    }
}
```

Local (method scope)

Attribute (class scope)

When to Use: Attributes

- Typically there is a separate attribute for each instance of a class and it lasts for the life of the object.

```
class Person
{
    private String [] childrenName = new String[10];
    private int age;
    /*
      For each person it's logical to track the age and
      the names any offspring.

    */
}
```

Q: Life of an object?

When to Use: Locals

- Local variables: temporary information that will only be used inside a method

```
public nameFamily()
{
    int i;
    Scanner in = new Scanner(System.in);
    for (i = 0; i < 10; i++)
    {
        childrenName[i] = in.nextLine();
    }
}
```

Scope of 'i' (int)

Scope of 'in' (Scanner)

- Q: Does it make sense for every 'Person' to have an 'i' and 'in' attribute?

Scoping Rules

- Rules of access
  1. Look for a local (variable or constant)
  2. Look for an attribute

```
public class Person
{
    public void method()
    {
        x = 12;
    }
}
```

Second: look for the definition of an attribute e.g., “private int x;”

First: look for the definition of a local identifier e.g., “int x;”

Reference to an identifier

Shadowing

- The name of a local matches the name of an attribute.
- Because of scoping rules the local identifier will 'hide' (shadow) access to the attribute.
- This is a common logic error!

```
public class Person {
    private int age = -1;
    public Person(int newAge) {
        int age;  // Shadows/hides attribute
        age = newAge;
    }
    public void setAge(int age) { // Shadow/hide attribute
        age = age;
    }
}

Person aPerson = new Person(0);  // age is still -1
aPerson.setAge(18);              // age is still -1
```

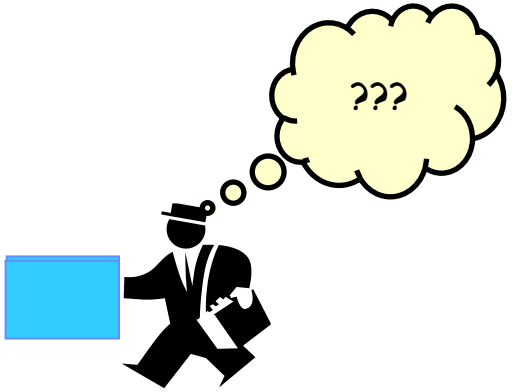
Addresses And References

- Real life metaphor: to determine the location that you need to reach the 'address' must be stored.

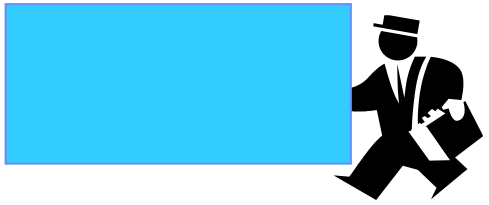

121


122


123



- Think of the delivery address as something that is a 'reference' to the location that you wish to reach.
  - Lose the reference and you can't 'access' (go to) the desired location.



Variables & Addresses

- Variables are a ‘slot’ in memory that contains ‘one piece’ of information.

  num = 123

- Normally a location is accessed via the name of the variable.

  - Note however that each location is also numbered!

  - This is the address of a memory location.

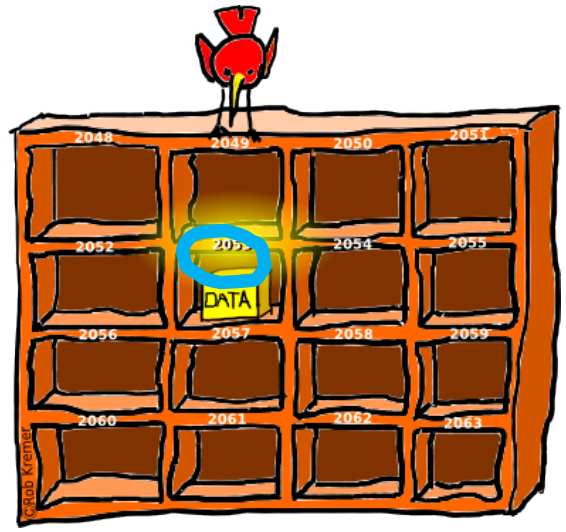


Image: Curtesy of Rob Kremer

References & Objects

```
public class Person
{
    private int age;
    public Person() {
            age = 0;
    }

    public Person(int data) {
            age = data;
    }

 public int getAge() {
            return(age);
    }
 public void setAge(int data) {
            age = data;
    }
}
```

- In main():

```
Person ahmed;
 Person mariam;

 ahmed = new Person(20);
 mariam = new Person(15);
 System.out.println("Ahmed object age: " + ahmed.getAge());

 mariam = ahmed;
 ahmed.setAge(21);
 System.out.println("ahmed object name: " + ahmed.getName());
 System.out.println("mariam object name: " + mariam.getName());
```

Summary

- Constructor overloading
- Accessor method (“get”)
- Mutator method (“set”)
- Method overloading
- Method signature
- Encapsulation/information hiding
- UML
- Scoping rules
- Addresses and references

Assignment 2

I. Create a class called time that has:

- separate int member data for hours, minutes, and seconds.
- One constructor should initialize this data to 0,
- Another constructor initialize it to fixed values.
- Accessor & mutators methods for the attributes.
- A method to display time in 11:59:59 format.
- A method to add two objects of type time passed as arguments.
- In the main() should create two initialized time objects and one that isn't initialized. Then it should add the two initialized values together, leaving the result in the third time variable. Finally, it should display the value of this third variable.

Assignment 2

II. Create an employee class.

- The member data should comprise an int for storing the employee number and a float for storing the employee's salary.
- Methods to allow the user to enter this data and display it.
- Apply constructors overloading.
- Accessor & mutators methods for the attributes.

Assignment 2

III. Create a date class.

- Its member data should consist of three ints: month, day, and year.
- Two member functions: getDate(), which allows the user to enter a date in 12/31/02 format, and showDate(), which displays the date.
- Apply constructors overloading.
- Accessor & mutators methods for the attributes.

Thank you