College of Computer and Information Systems Department of Computer Science

جامعة أم القرى
UMM AL-QURA UNIVERSITY

CS 1704: Object Oriented Programming

1st Trimester, 2025

Lecture 2

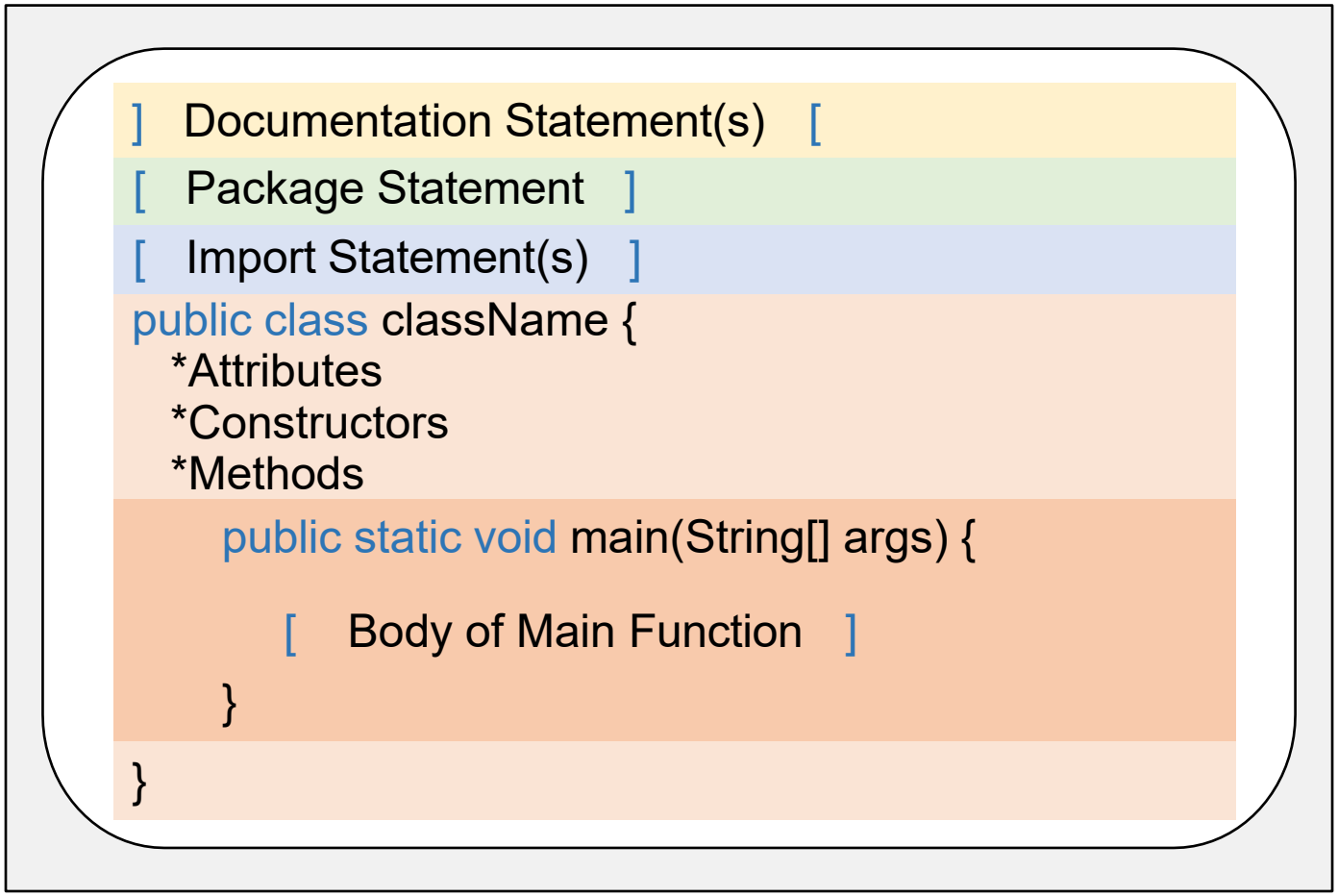
OOP - Part2

Prepared By:

Dr. Azhar Hasan Alhindi

Revision: Basic Java Program Structure

- There are 5 components that we usually include in a Java program, which are:



JAVA: An Introduction to Problem Solving & Programming, 8th Ed. By Walter Savitch
ISBN 0134462033

Documenting your Class

- Before you can go **public** with your new class ➔ prepare the **documentation** for the class
- **JavaDoc** is a **document generator tool** **in Java**
- Documenting with *javadoc* ➔ automatically create **HTML-based** documentation/file which provides information about classes and methods of the project.
- An HTML document created from **comments** in Java program
- Documentation comments **Ignored** by the Java compiler, but processed by ***javadoc***
- Begin with `/**` and end with `*/`
- Formatting **tags** to begin with the `@` symbol, *e.g.*,

```
@author text

@param parameter-name description

@return description
```

Documenting your Class (Cont.)

- Three different **locations** in the source file ➔

  Before the declaration of the **public** class, **public** field or **public**
  method or constructor.

- To generate the documentation:
  2. In command ➔ `javadoc filename.java`
  3. In Netbeans ➔ Run-> Generate JavaDoc

Documenting your Class (Cont.)

- Example:

1 Get the Meal class and save it in another file called ‘`MealForJavaDoc.java`’.

2 Place this before setName method (you can add more)

```
/**
* Set the name attribute
* @param n name of meal
*/

public void setName(String n){
name=n;
}
```

```java
/**
 *
 * @author Azhar
 */
public class Student {

    String name;

    /**
   * Changes the name of this Student.
   * This may involve a lengthy legal process.
   * @param newName This Student's new name.
   *                Should include both first
   *                and last name.
   * @return the new name in the parameter
   */
    public String setName(String newName)
    {
        name = newName;
        return name;
    }
```

All Methods	Static Methods	Instance Methods	Concrete Methods

Modifier and Type	Method and Description
static void	main(java.lang.String[] args)
java.lang.String	setName(java.lang.String newName) Changes the name of this Student.

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait,

Constructor Detail

Student

public Student()

Method Detail

setName

public java.lang.String setName(java.lang.String newName)

Changes the name of this Student. This may involve a lengthy legal process.

Parameters:

newName - This Student's new name. Should include both first and last name.

Returns:

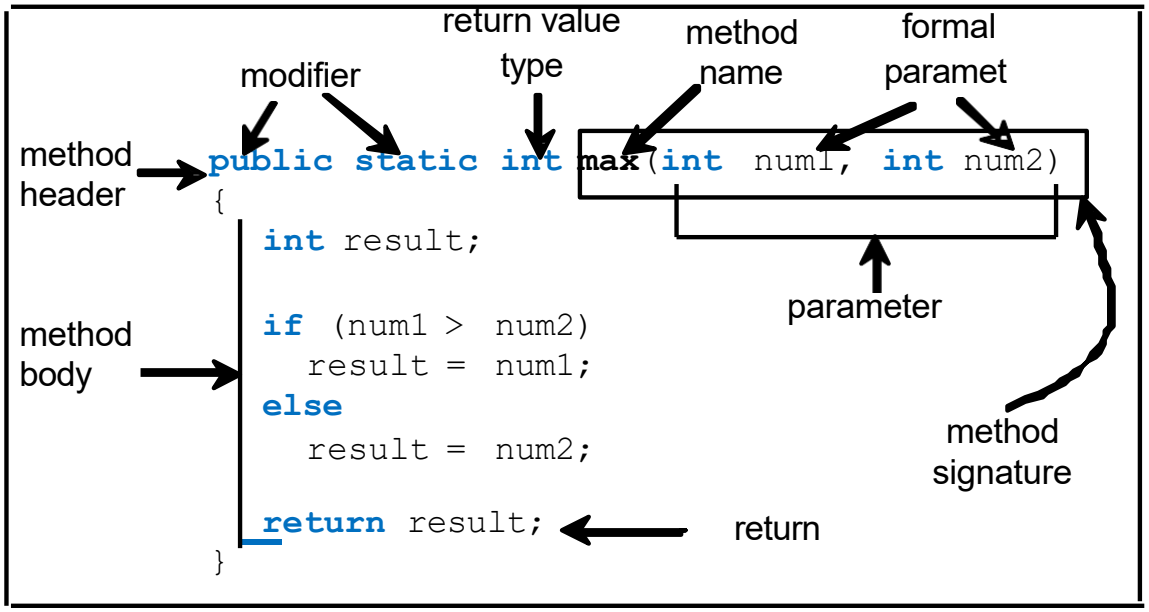
the new name in the parameter

main

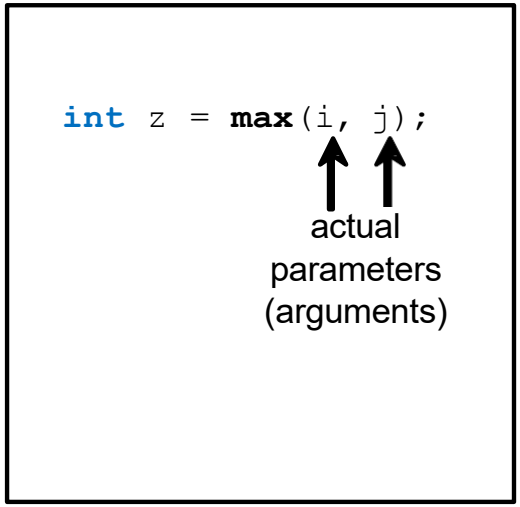
Method Structure

- A method consists of two parts: a method header and body

Define a method



Call a method



Calling a method is equivalent to executing the method body

Passing parameters

- when an **object/ reference** of the variable is passed into a method as a parameter, its reference is sent which means that any change to the object in the called method will be reflected in the original object in the calling methods.

- This is called **"call by reference"** while using **primitive data types** is called **"call by value"** because the actual value (copy of the variable) is sent to the called method and hence any changes does not affect the original variable

Passing parameters

- **Call-by-value:**
  - Actual parameters can be fields, literals or expressions.
  - Modifying the formal parameter does not change the actual parameter

```
double discount =1.00;
int cal= 400;
Meal salad = new Meal ("Salad",3.99-discount, cal, Meal.VEGETERIAN);

public void Meal (String n, double p, int c , int d){
        name = n;
        price = p;
        c = (int)(c*0.5);
        calories = c;
        dite = d;
}
```

Passing parameters: Example

```
public class Main {
   public static void main(String[] args) {
      A a = new A(10);
      int y = 5;
      changeObject(a,y);
      System.out.println(a.x);
      System.out.println(y);
   }

   public static void changeObject( A b, int z) {
     b.x =100
     Z=50
   }
}
```

```
public class A{
 public int  x;

 public A(int x){
  this.x = x;
}
}
```

Output:
100
5

Anonymous reference/ anonymous object

- Int x=9; //x is primitive variable
- A obj; //obj is reference variable
- we can not call the show method using obj.
- We need to create that object first ..

  A obj = new A();
- Now we can reach to the show method:

  obj.show();

```
Public class A{
  Show(){….}
}
```

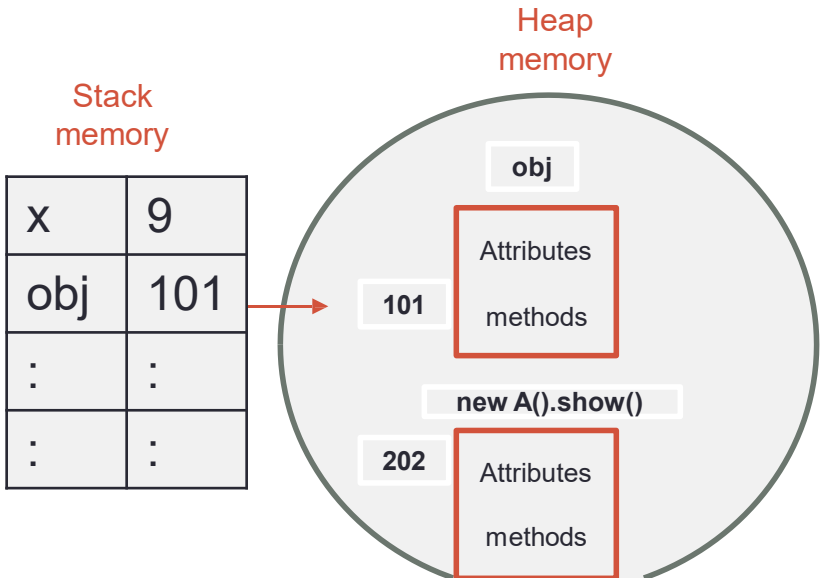
- **If we need to use this object once**, we do not need to create any reference object here:

  so we can call the method by this way:

  new A().show();

- This is will not consume any stack memory it will only consume heap memory
- So, this type of objects that does not have a reference/ a name is called anonymous object.

- **The advantages of that are:**
  - It will not use any memory inside stack (to save memory).
  - This type of objects is created and dies immediately.



Class Relationships

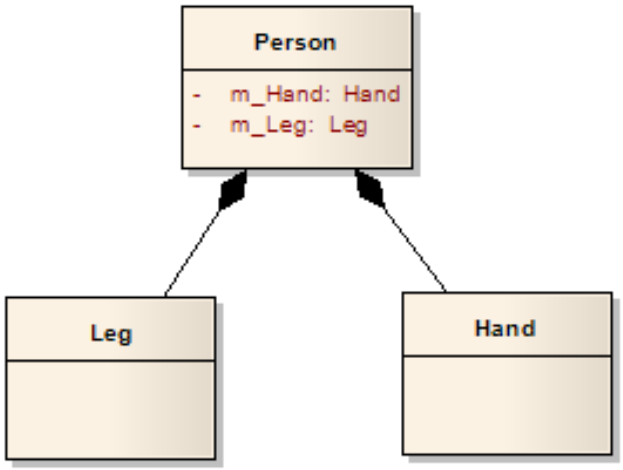
Association
Aggregation - Composition
Inheritance

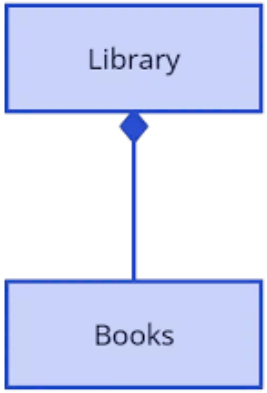
Class Relationships

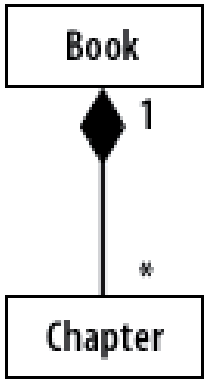
- To **design classes**, you need to explore the relationships among classes. The common relationships among classes are association, aggregation, composition, and inheritance.
- Association
- Aggregation, **Composition**
- Inheritance (Next Lecture)

Object Composition

- **The Composition** is a way to design or implement the **"has-a"** relationship.
- The **"has-a"** relationship is used to ensure the **code reusability** in our program.
- In Composition, we use an instance variable that refers to another object.
- The composition relationship of two objects is possible when one object contains another object, and that object is fully dependent on it.







The composition relations are implemented using data fields in classes.

Example

```
public class ADate {
  public static final int JANUARY   = 1;
  public static final int FEBRUARY  = 2;
  public static final int MARCH     = 3;
  public static final int APRIL     = 4;
  public static final int MAY       = 5;
  public static final int JUNE      = 6;
  public static final int JULY      = 7;
  public static final int AUGUST    = 8;
  public static final int SEPTEMBER = 9;
  public static final int OCTOBER   = 10;
  public static final int NOVEMBER  = 11;
  public static final int DECEMBER  = 12;
  private int day, month, year;

  public ADate() {
    day = 0; month = 0; year = 0;
  }                                                    continue
  public ADate(int day, int month, int year) {
    this.day = day; this.month = month; this.year = year;
  }
```

Example

```
  public ADate copy() {
    ADate d = new ADate();
    d.day = day;
    d.month = month;
    d.year = year;
    return(d);
  }

  public void setDay(int d) { day = d; }
  public void setMonth(int m) { month = m; }
  public void setYear(int y) { year = y; }
  public int getDay() { return(day); }
  public int getMonth() { return(month); }
  public int getYear() { return(year); }

  public String toString() {
    return("(" + day + ", " + month + ", " + year + ")");
  }
}
```

Object Composition

```
public class TestAPerson1 {
  public static void main (String[] args) {
    ADate date1 = new ADate(1, 7, 1964);
    Person person = new Person("Homer", 38, date1);
    System.out.println("person:  " + person);
    ADate date2 = person.getDateOfBirth();
    System.out.println("date2:  " + date2);
  }
}
```

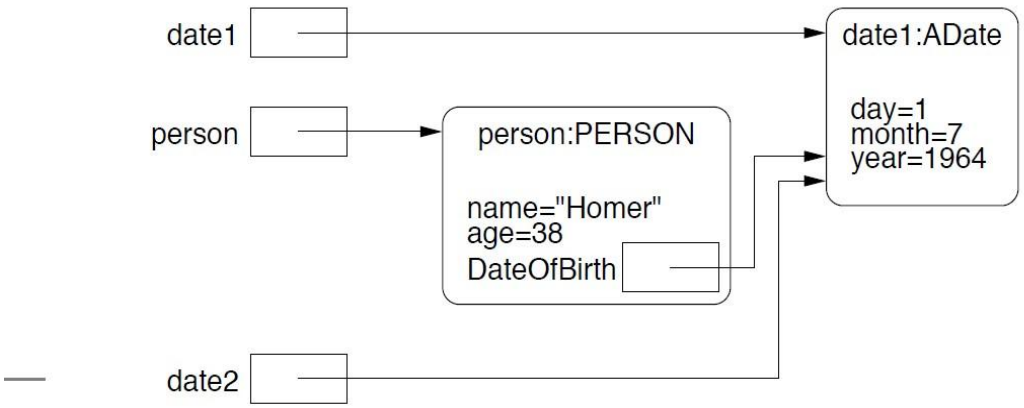
```
public class Person {
  private static final String NO_NAME = "NONAME";
  private String name;
  private int age;
  private ADate dateOfBirth;

  public Person() {
    name = NO_NAME; age = 0; dateOfBirth = new ADate();
  }
  public Person(String n, int a) {
    name = n; age = a; dateOfBirth = new ADate();
  }
  public Person(String n, int a, ADate d) {
    this(n, a); dateOfBirth = d;
  }

    code fragment 1        // mutator and accessor methods

  public String toString () {
    return("(" + name + ", " + age + ", " + dateOfBirth + ")");
  }
}
```


Object composition — objects within objects:



Object Composition

- **Creating Copy of objects:**

  - Shallow Copy:
    - A new object is created that has an exact copy of the values in the original object.
    - Just the reference addresses are copied

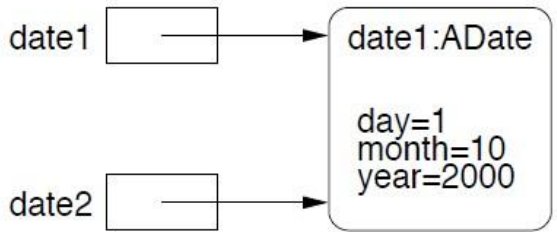
  - Deep Copy :
    - A deep copy copies all fields and makes copies of dynamically allocated memory pointed to by the fields.

Object Composition

```
public class TestADate2 {
  public static void main (String[] args) {
    ADate date1 = new ADate(1, 10, 2000);
    ADate date2 = date1;                    // copying the reference
    System.out.println("date1:   " + date1);
    System.out.println("date2:   " + date2);

    System.out.println("Statement:  date2.setDay(3);");
    date2.setDay(3);
    System.out.println("date1:   " + date1);
    System.out.println("date2:   " + date2);
  }
}
```

Shallow copy:



The shallow copy of an object will have exact copy of all the fields of original object. It is coping the reference.

Object Composition

```
public class TestADate3 {
  public static void main (String[] args) {
    ADate date1 = new ADate(1, 10, 2000);
    ADate date2 = date1.copy();              // copying the object
    System.out.println("date1:   " + date1);
    System.out.println("date2:   " + date2);

    System.out.println("Statement:  date2.setDay(3);");
    date2.setDay(3);
    System.out.println("date1:   " + date1);
    System.out.println("date2:   " + date2);
  }
}
```

Deep copy:

Object Composition

- Avoid a shallow copy — prone to error
- Use a deep copy.

```
public Person(String n, int a, Date d) {
    this(n, a);
    dateOfBirth = d.copy();
}
public void setDateOfBirth(Date d) {
    dateOfBirth = d.copy();
}
public Date getDateOfBirth() {
    return(dateOfBirth.copy());
}
```

- Can also use an anonymous reference

```
Person person = new Person("Homer", 38, new Date(1, 7, 1964));
```

Object Composition

- Can hide the Date class within the Person class

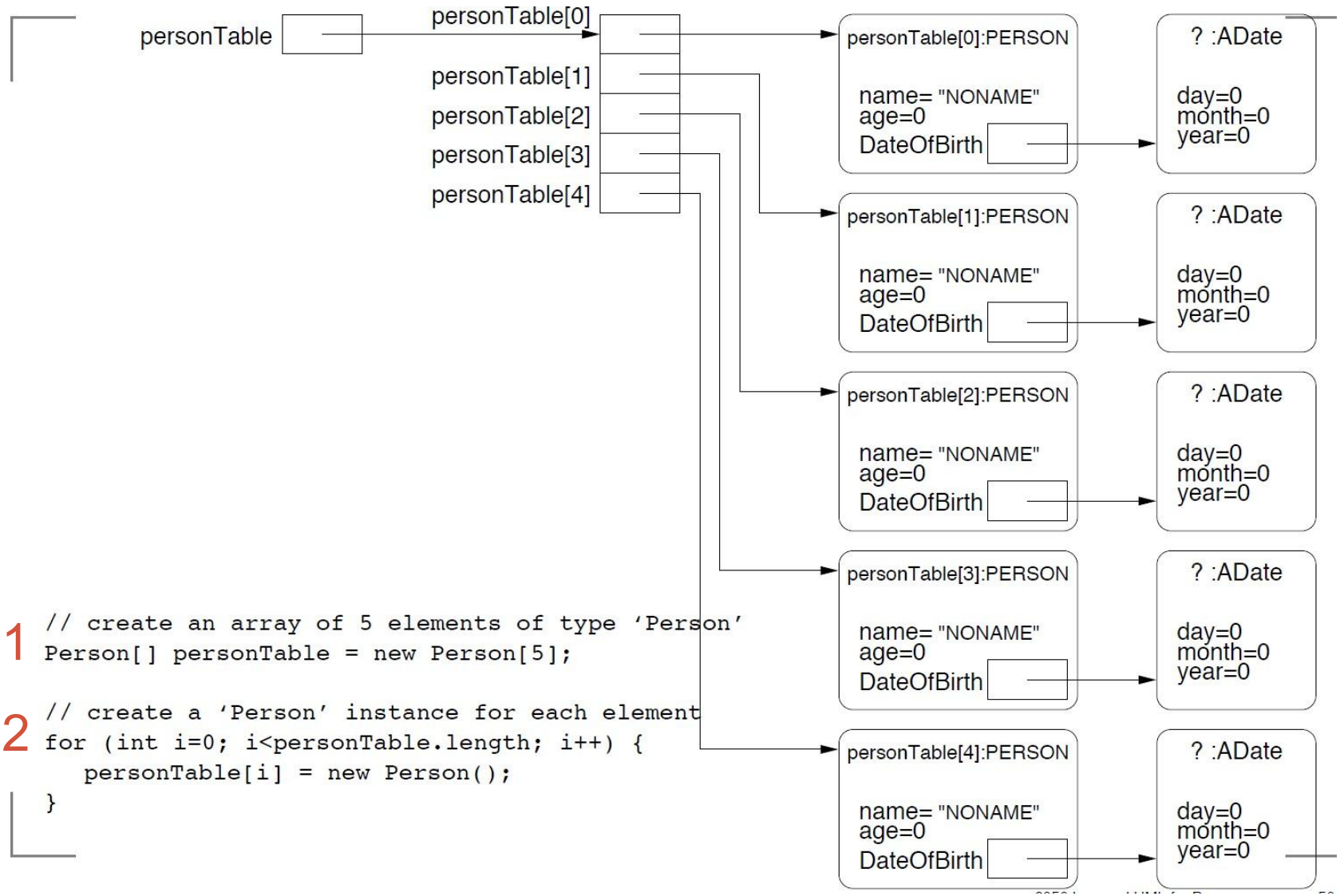
```
Public Person(String n, int a, int d, int m, int y) {
    name= n;
    age = a;
    dateOfBirth = new Date(d,m,y);
}
public void setDateOfBirth(int d, int m, int y) {
    dateOfBirth = new Date(d,m,y);
}
```

Object Composition: Arrays of Objects

arrays of objects can be created in the same way as arrays of Java basic types

```
int[] number = new int[5];              // array of 5 integers
Person[] personTable = new Person[5]; // array of 5 people
```

Object Composition: Arrays of Objects



1
```
// create an array of 5 elements of type 'Person'
Person[] personTable = new Person[5];
```

2
```
// create a 'Person' instance for each element
for (int i=0; i<personTable.length; i++) {
   personTable[i] = new Person();
}
```

References

- Slides are modified from: Dr. Atif Alhejali's lectures, Dr.seereen's lectures & Dr.Manal's lectures and other references
- https://www.youtube.com/watch?v=qP9-3LnMZsE
- https://www.youtube.com/watch?v=pTB0EiLXUC8&t=75s