Object Oriented Programming (JAVA)



## Semester: Fall 2024

**Software Engineering**

**Faculty of Information Technology UCP Lahore, Pakistan**

|  |  |
| --- | --- |
| **Week 5** | |
| **Topic** | **Passing and returning objects, array of objects, Copy Constructors, Object Referencing and Garbage Collection** |
| **Objective** | * Understanding how to pass and return objects in Java through member functions. * Learn how to initialize, access, and modify arrays of objects in Java. * Differentiate between deep copy and shallow copy in Java objects. * Explore object referencing and understand Java’s garbage collection mechanism. |

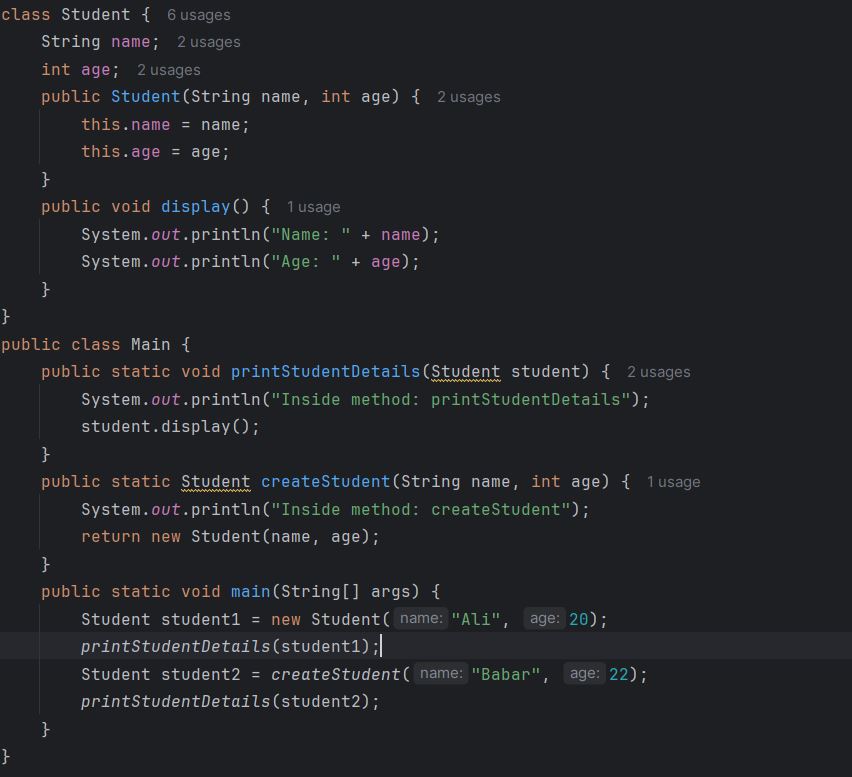
**Introduction to passing and retuning objects:**

Although Java is strictly passed[by value](https://www.geeksforgeeks.org/g-fact-31-java-is-strictly-pass-by-value/), the precise effect differs between whether a [primitive type](https://www.geeksforgeeks.org/data-types-in-java/) or a reference type is passed. When we pass a primitive type to a method, it is passed by value. But when we pass an object to a method, the situation changes dramatically, because objects are passed by what is effectively call-by-reference.

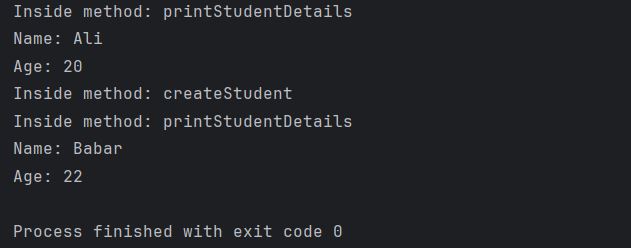
Java does this interesting thing that’s sort of a hybrid between pass-by-value and pass-by-reference.

**Passing and returning an object to a member function:**

**In java we can pass the objects to a method and also return the objects from a method as it can be perceived from the example below:**

****

**Output:**

****

**Using an array of objects:**

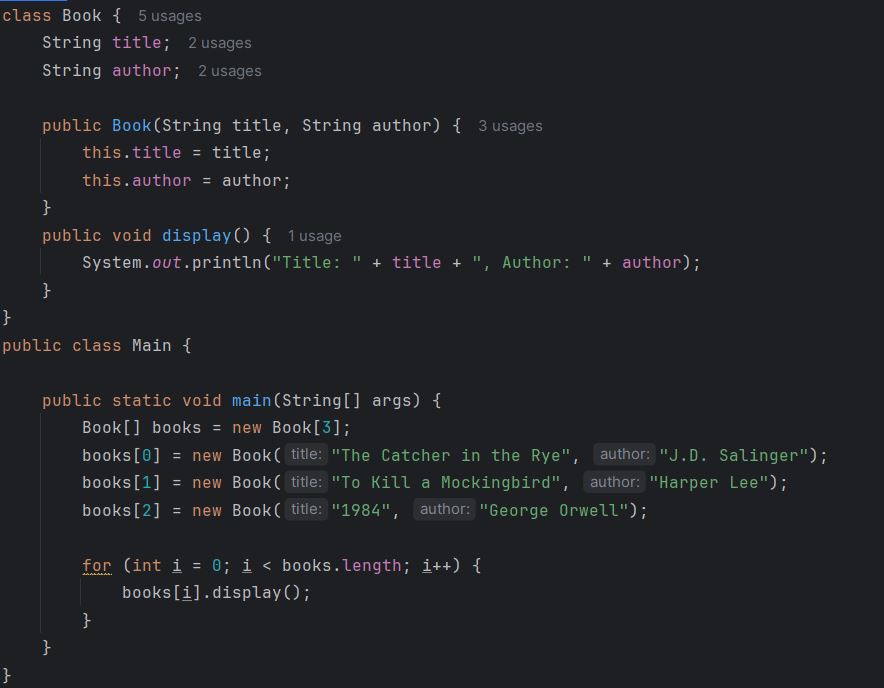
An Array of Objects is created using the [Object class](https://www.geeksforgeeks.org/object-class-in-java/), and we know Object class is the root class of all Classes.We use the*Class\_Name* followed by a square bracket*[]* then object reference name to create an Array of Objects.

Class\_Name[ ] objectArrayReference;

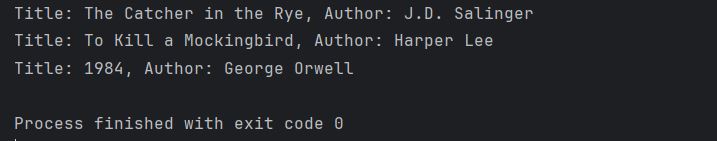
**Alternatively, we can also declare an Array of Objects as :**

Class\_Name objectArrayReference[ ];

Both the above declarations imply that *objectArrayReference* is an array of objects.

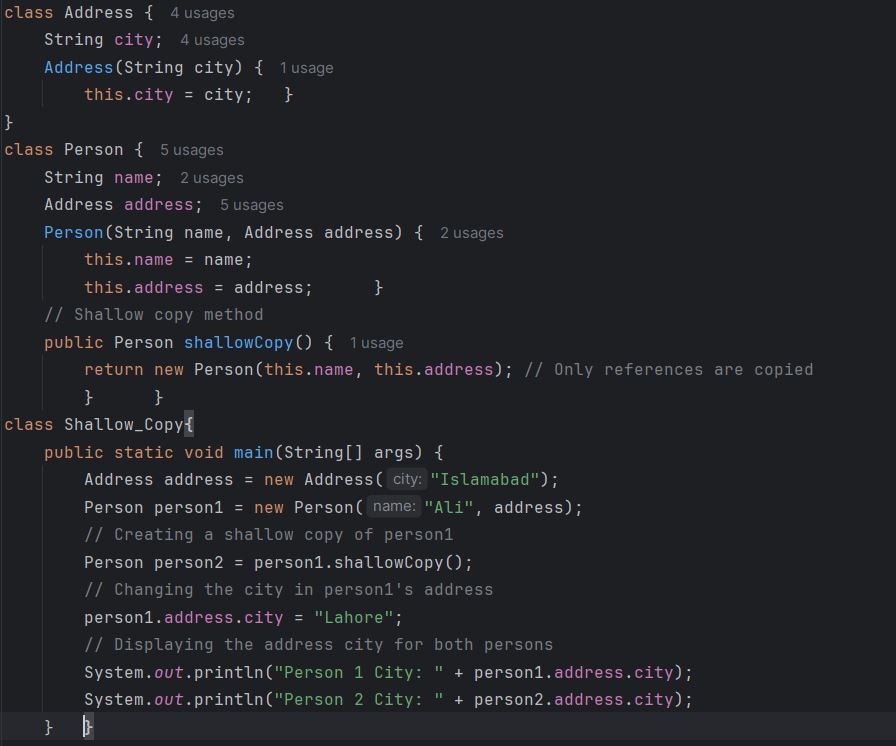
****

**Output:**

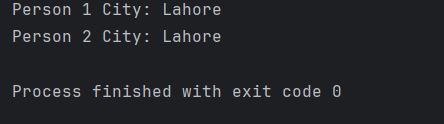
****

**Copy Constructor:** Like C++, Java also supports a copy constructor. But, unlike C++, Java doesn’t create a default copy constructor if you don’t write your own.

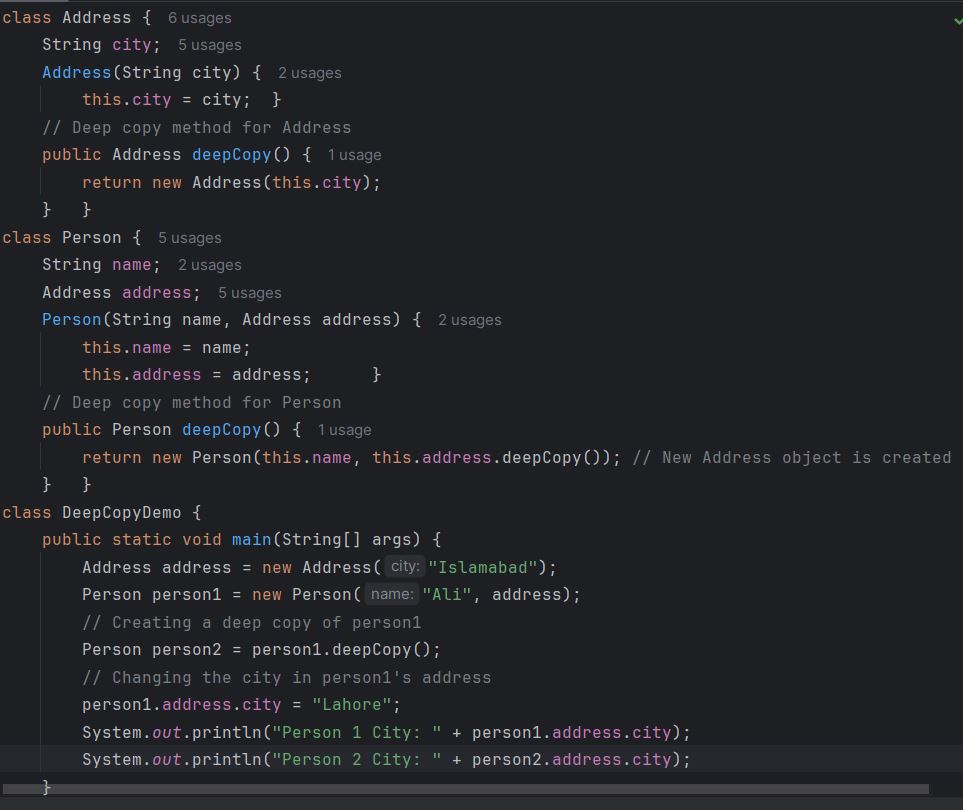
1. **Shallow Copy:** In shallow copy, the object’s references are copied, so changes to the referenced object affect both the original and the copied object.



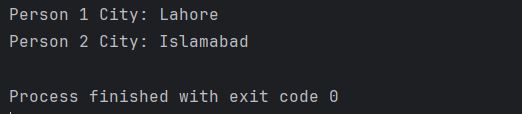
**Output:**



1. **Deep Copy:** Deep repetition truly clones the underlying data. It is not shared between the first and therefore the copy.



**Output:**



**Object Referencing and Garbage Collection:**

In Java, object references can be assigned from one object to another, and when you set the original reference to null, the object may become eligible for garbage collection if there are no more active references to it. Java's garbage collector automatically reclaims memory used by objects that are no longer referenced.

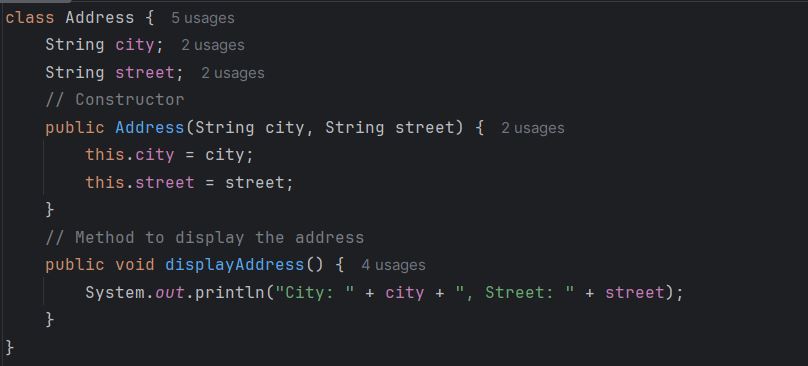
**Example:**

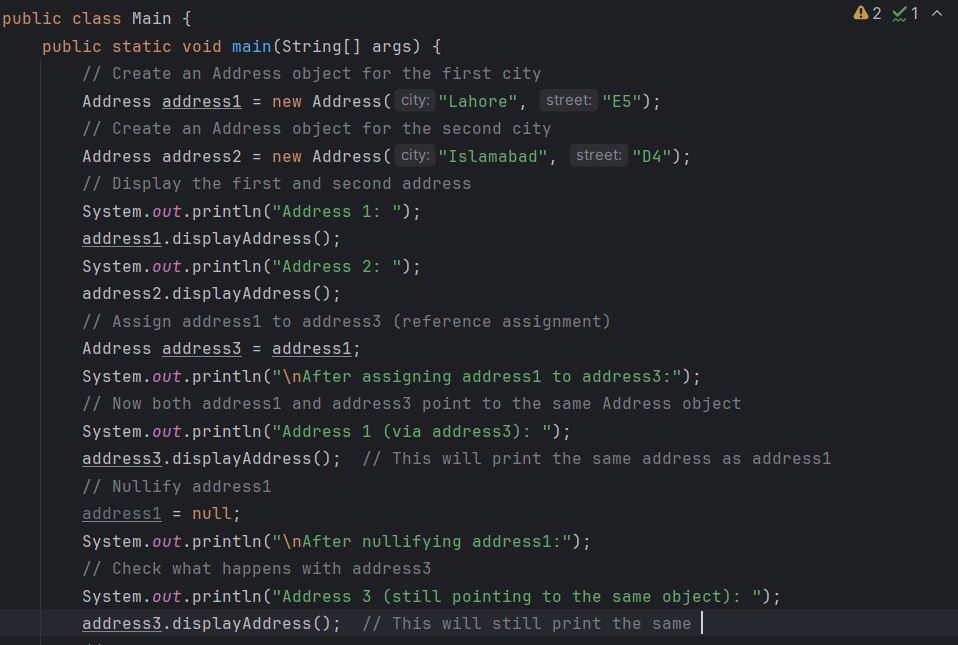
Create Address objects: Two address objects have been defined each associated with different cities.

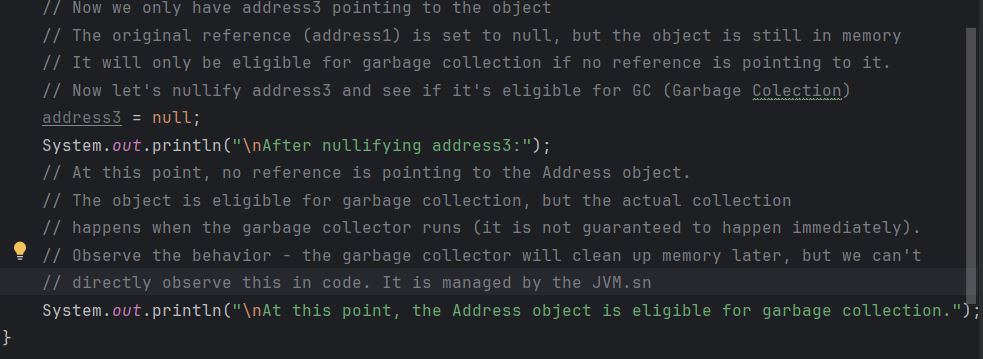
Assign object reference: One object reference will be assigned to another, so both references point to the same object.

Nullify original reference: The original reference will be set to null, and we’ll observe the effect on the object.

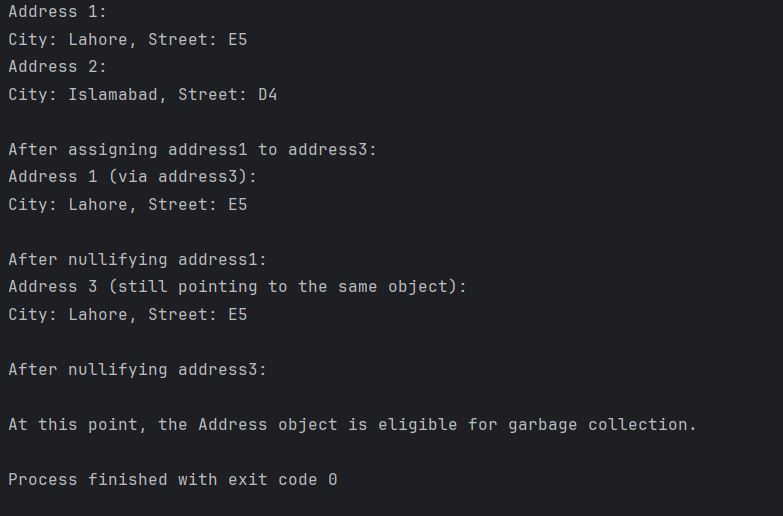
Garbage collection: When there are no more references to the object, it becomes eligible for garbage collection.







**Output:**

****

**Scenario Based Task**

**Task 1: Define the SmartFan Class**

**Objective**: Create a class that represents a smart fan.

**Instructions**:

1. Create a SmartFan class with the following attributes:

* String fanName (e.g., "Bedroom Fan")
* boolean isOn (initially false)

2. Add the following methods:

* void turnOn(): turns the fan on.
* void turnOff(): turns the fan off.
* String getStatus(): returns "On" or "Off" based on fan status.

3. Create a class SmartHomeController where:

* You pass a SmartFan object to a method operateFan(SmartFan fan) that turns ON the fan and returns the modified object.
* Print the status before and after passing the object to the method.

|  |
| --- |
| **Expected Output**:  Initial Fan status: Off  After operation: On |

**Task 2: Using Arrays of Objects in Java**

**Instructions**:

1. Extend the SmartFan class from Task 1.
2. In SmartHomeController, create an array of 5 SmartFan objects, each representing a different fan in the house.
3. Write methods to:

* Initialize the array with default fan names, such as:
  + "Bedroom Fan", "Kitchen Fan", "Living Room Fan", "Bathroom Fan", "Garage Fan"
* Turn ON all the fans.
* Print their statuses.

|  |
| --- |
| **Expected Output**:  Bedroom Fan status: On  Kitchen Fan status: On  Living Room Fan status: On  Bathroom Fan status: On  Garage Fan status: On |

**Task 3: Deep Copy vs Shallow Copy**

1. Modify the SmartFan class by adding an attribute:

* int speed (range 0–5)

1. Write methods for:

* **Shallow copy**: create a new reference pointing to the same SmartFan object.
* **Deep copy**: create a new object with the same values but a different reference.

1. In SmartHomeController, demonstrate shallow and deep copy:

* Create a SmartFan object.
* Make a shallow and deep copy.
* Modify the original fan and observe how both copies behave.

|  |
| --- |
| **Expected Output**:  Original speed: 2  Shallow copy speed: 2  Deep copy speed: 2  After changing original speed to 4:  Original speed: 4  Shallow copy speed: 4  Deep copy speed: 2 |

**Task 4: Object Referencing and Garbage Collection**

**Objective:**

Demonstrate how Java handles object references and garbage collection.

**Instructions:**

1. Extend the SmartFan class.
2. In SmartHomeController:

* Assign one object reference to another.
* Nullify the original reference.
* Request garbage collection using System.gc().

1. Show how both references point to the same object before nullifying.

|  |
| --- |
| **Expected Output**:  Fan status: On  AnotherRef status: On |