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. |

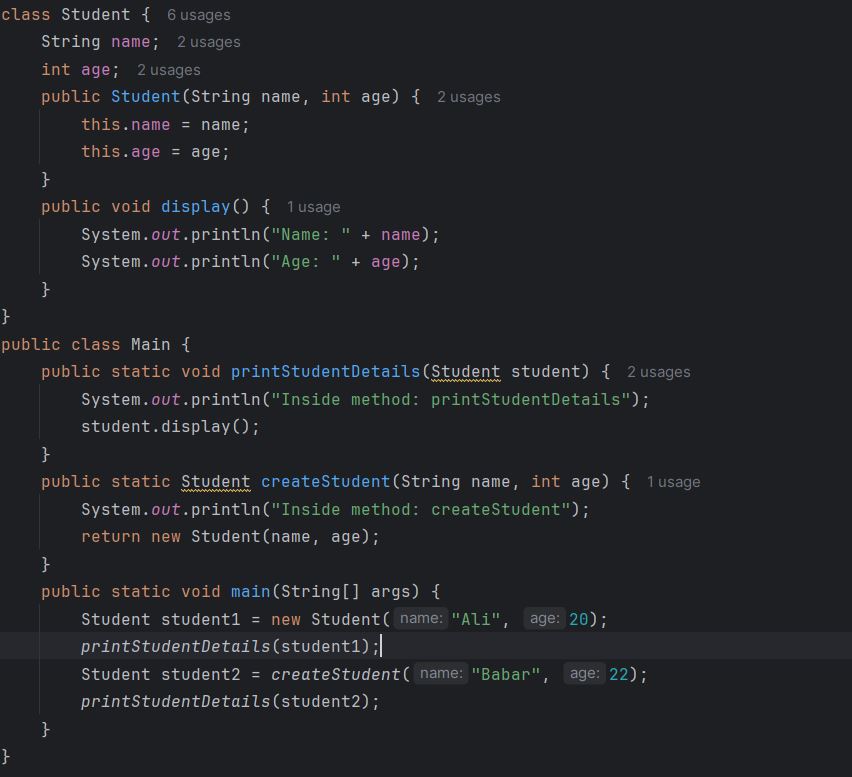
**Introduction to passing and retuning objects:**

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.

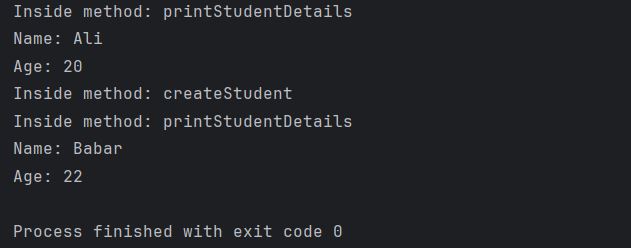
Java does this interesting thing that’s sort of a hybrid between pass-by-value and pass-by-reference.

**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:**

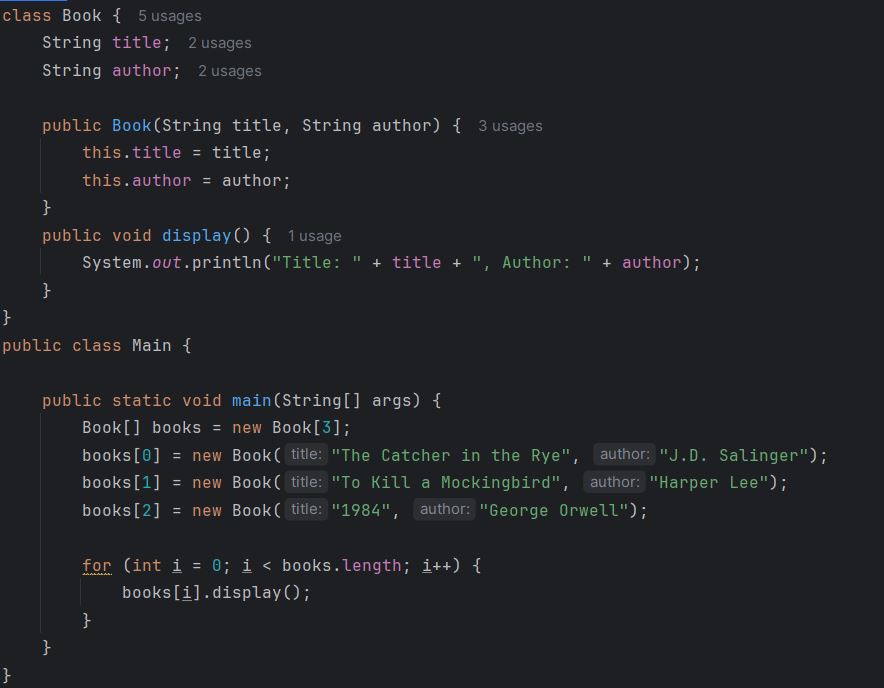
An Array of Objects is created using the [Object class](https://www.geeksforgeeks.org/object-class-in-java/), and we know Object class is the root class of all Classes.We use the*Class\_Name* followed by a square bracket*[]* then object reference name to create an Array of Objects.

Class\_Name[ ] objectArrayReference;

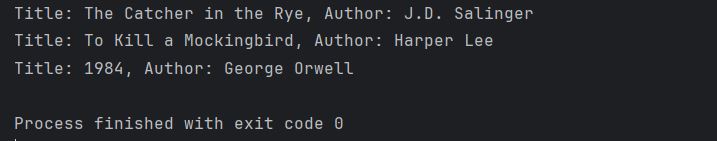
**Alternatively, we can also declare an Array of Objects as :**

Class\_Name objectArrayReference[ ];

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

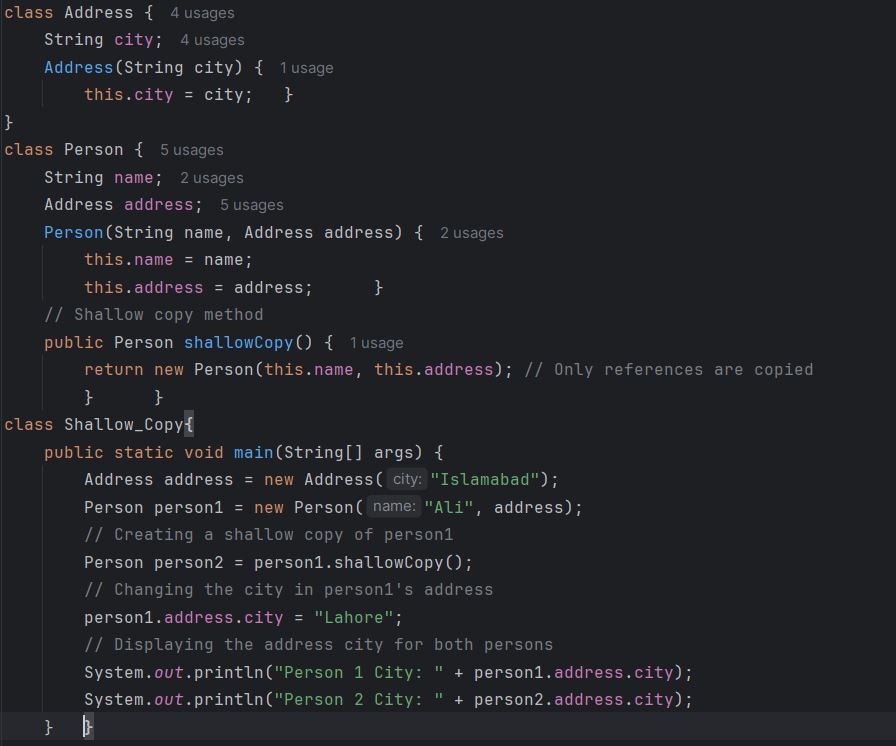
****

**Output:**

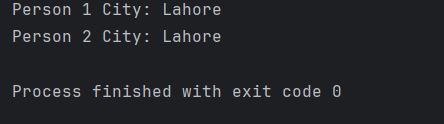
****

**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.

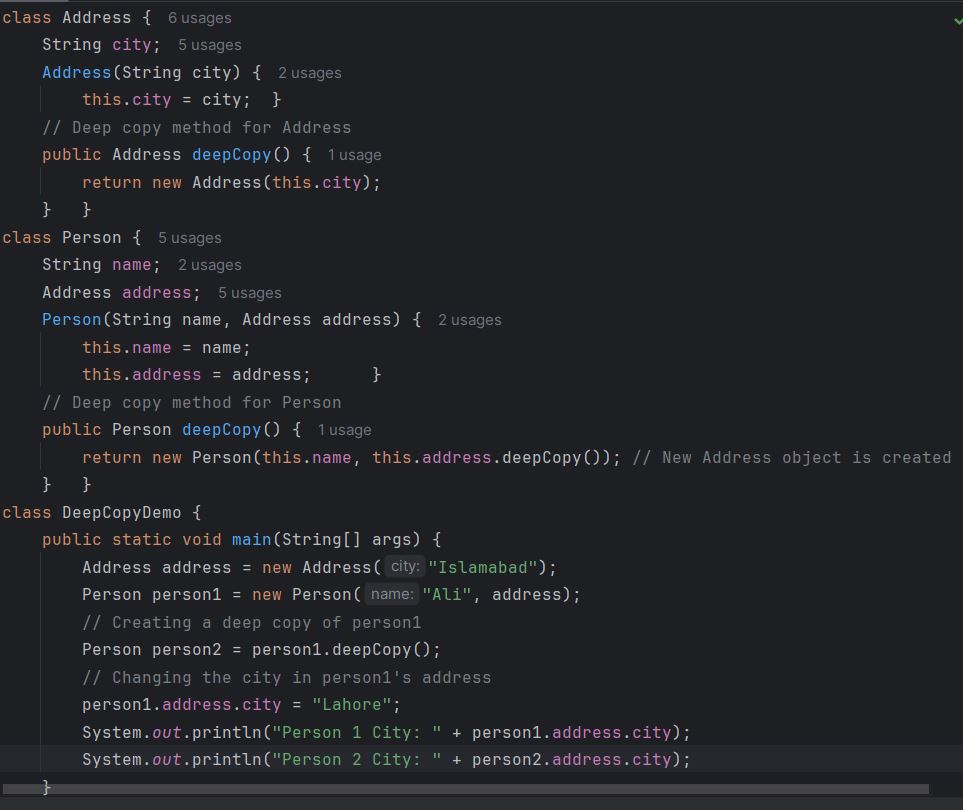
1. **Shallow Copy:** In shallow copy, the object’s references are copied, so changes to the referenced object affect both the original and the copied object.



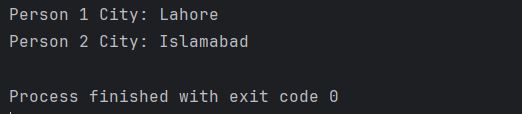
**Output:**



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.

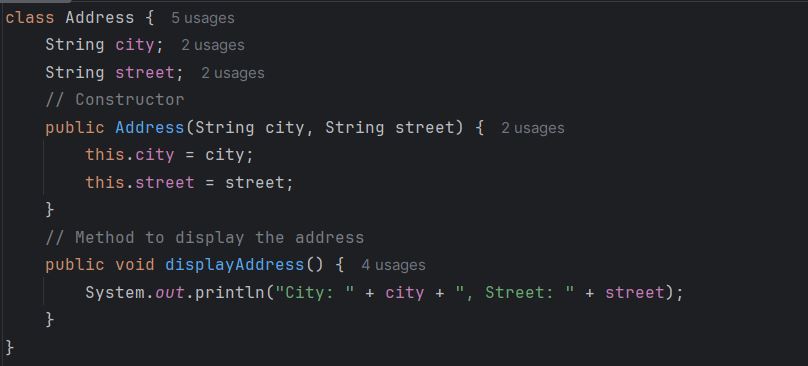
**Example:**

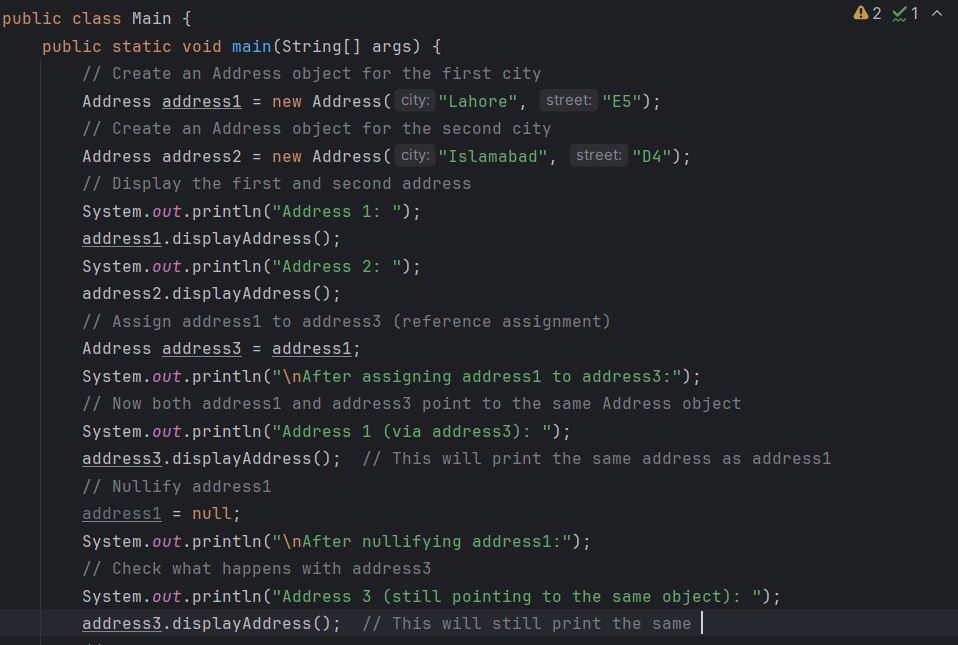
Create Address objects: Two address objects have been defined each associated with different cities.

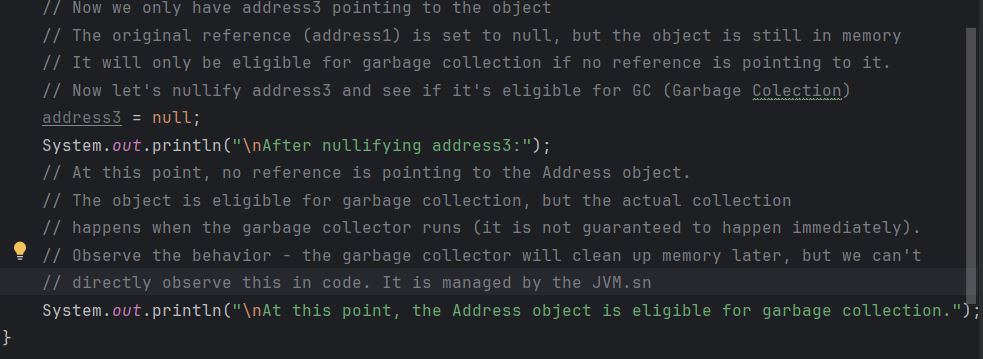
Assign object reference: One object reference will be assigned to another, so both references point to the same object.

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

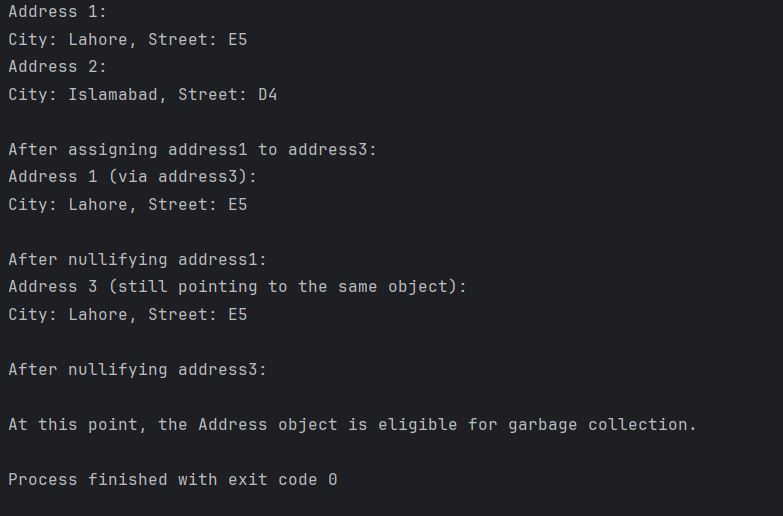
Garbage collection: When there are no more references to the object, it becomes eligible for garbage collection.







**Output:**

****

**Basic Task**

### Question 1: Library Book Management System

Design a **Book** class in Java to manage a library’s book inventory. Each Book object should store details of a book and provide functionalities for copying, comparing, and tracking copies available in the library. Additionally, implement a method to identify the book with the highest average rating from an array of Book objects.

#### Requirements:

1. **Attributes**:
   * isbn (String): Unique identifier for the book (International Standard Book Number).
   * title (String): Title of the book.
   * author (String): Author of the book.
   * availableCopies (int): The number of copies of this book currently available in the library.
   * ratings (double[]): An array to store ratings (up to 5 ratings) given by readers on a scale of 0.0 to 5.0.
2. **Constructors**:
   * **Default Constructor**: Initializes default values.
   * **Parameterized Constructor**: Initializes isbn, title, author, availableCopies, and ratings. Use deep copy where necessary.
   * **Copy Constructor**: Accepts another Book object and creates a deep copy.
3. **Getter and Setter Methods**:
   * Provide **getters** and **setters** for each attribute.
   * **Input Validation**:
     + availableCopies should not be negative.
     + Each value in ratings should be within 0.0 and 5.0.
4. **Rating Calculation Method**:
   * Implement a calculateAverageRating() method that:
     + Adds up all ratings in the ratings array.
     + Calculates the average rating.
     + Returns the average rating as a double.
5. **Deep Copy Method**:
   * Implement a public Book copy() method that returns a new Book object, which is a deep copy of the current instance.
6. **Comparison Method**:
   * Implement a public boolean equals(Book otherBook) method that takes another Book object as a parameter and returns true if both books have the same isbn and title, and false otherwise.
7. **Find Highest Rated Book Method**:
   * Implement a public static Book findHighestRated(Book[] books) method that takes an array of Book objects as a parameter and returns the Book object with the highest average rating. If two books have the same average rating, return the first one found.
8. **Testing the Class**:
   * In the main method:
     + Create a Book object using the parameterized constructor.
     + Use setters to modify some attributes.
     + Add ratings using the ratings array.
     + Use copy() to create a deep copy of the original Book.
     + Use equals() to compare the copied Book with the original Book.
     + Create an array of Book objects and use findHighestRated to find and display the book with the highest average rating.
     + Display details of the original and copied books.

### Question 2: Student Record Management System

Design a **Student** class in Java to manage student records. Each Student object should store details of a student and allow for copying, comparing, and calculating the grade based on the student’s marks.

#### Requirements:

1. **Attributes**:
   * studentId (int): Unique identifier for the student.
   * name (String): Name of the student.
   * age (int): Age of the student.
   * marks (float[]): An array to store the marks for 3 subjects.
   * grade (char): The calculated grade of the student based on the total marks.
2. **Constructors**:
   * **Default Constructor**: Initializes default values.
   * **Parameterized Constructor**: Initializes studentId, name, age, and marks. Use deep copy where necessary.
   * **Copy Constructor**: Accepts another Student object and creates a deep copy.
3. **Getter and Setter Methods**:
   * Provide **getters** and **setters** for each attribute.
   * **Input Validation**:
     + age should be positive and within a reasonable range (e.g., 5 to 100).
     + Each value in marks should be within 0.0 and 100.0.
   * When setting marks, recalculate the grade.
4. **Grade Calculation Method**:
   * Implement a calculateGrade() method that:
     + Adds up the values in the marks array.
     + Calculates the average score.
     + Assigns grade based on the average:
       - 'A' for 90-100
       - 'B' for 80-89
       - 'C' for 70-79
       - 'D' for 60-69
       - 'F' for below 60
5. **Deep Copy Method**:
   * Implement a public Student copy() method that returns a new Student object, which is a deep copy of the current instance.
6. **Comparison Method**:
   * Implement a public boolean equals(Student otherStudent) method that takes another Student object as a parameter and returns true if both students have the same studentId and name, and false otherwise.
7. **Testing the Class**:
   * In the main method:
     + Create a Student object using the parameterized constructor.
     + Use setters to modify some attributes.
     + Use copy() to create a deep copy of the original Student.
     + Use equals() to compare the copied Student with the original Student.
     + Display details of the original and copied students.

**Scenario Based Task**

**Task 1: Define the SmartLight Class**

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

**Instructions**:

* Create a SmartLight class with the following attributes:
  + String name (e.g., "Living Room Light")
  + boolean isOn (initially false)
* Write methods in the SmartLight class:
  + turnOn(): Turns the light on.
  + turnOff(): Turns the light off.
  + getStatus(): Returns the status (on/off) of the light as a string.
* Create a class SmartHomeController where:
  + You pass a SmartLight object to a method operateLight(SmartLight light) that turns on the light and returns the modified object.
  + Print the status before and after passing the object.

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

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

**Instructions**:

* Extend the SmartLight class from Task 1.
* In SmartHomeController, create an array of 5 SmartLight objects, each representing a different light in the house.
* Write methods to:
  + Initialize the array with default light names (e.g., "Kitchen Light", "Bedroom Light").
  + Access and modify the status of each light.
* Turn all the lights on and print their statuses.

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

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

**Instructions**:

* Modify the SmartLight class by adding an attribute int brightness (range 0-100).
* Write methods for performing a shallow copy and a deep copy of the SmartLight object.
  + Shallow Copy: Create a new reference pointing to the same object.
  + Deep Copy: Create a new object with the same values but different references.
* Create a shallow and deep copy of a SmartLight object, modify the original, and observe how the copies behave.

|  |
| --- |
| **Expected Output**:  Original brightness: 50  Shallow copy brightness: 50  Deep copy brightness: 50  After changing original brightness to 80:  Original brightness: 80  Shallow copy brightness: 80  Deep copy brightness: 50 |

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

**Objective**:

* Extend the SmartLight class.
* In SmartHomeController, demonstrate object referencing by assigning one object reference to another.
* Nullify the original reference and observe how garbage collection works in Java when no references to the object exist.
* Call System.gc() to request garbage collection explicitly.

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