# National University of Computer & Emerging Sciences, Karachi

**Software Engineering Department**

**Spring 2024, Lab Manual – 03**

|  |  |
| --- | --- |
| **Course Code: CL-217** | **Course: Object Oriented Programming Lab** |
| **Instructor(s):** | **Ali Fatmi** |

LAB - 3

Classes & Objects in Java

**CONTENTS:**

**Class:**

A class is a group of objects which have common properties. It is a template or blueprint from which objects are created. It is a logical entity. It can't be physical. A class in Java can contain:

* Fields
* Methods
* Constructors
* Blocks
* Nested class and interface The syntax to declare the class is:

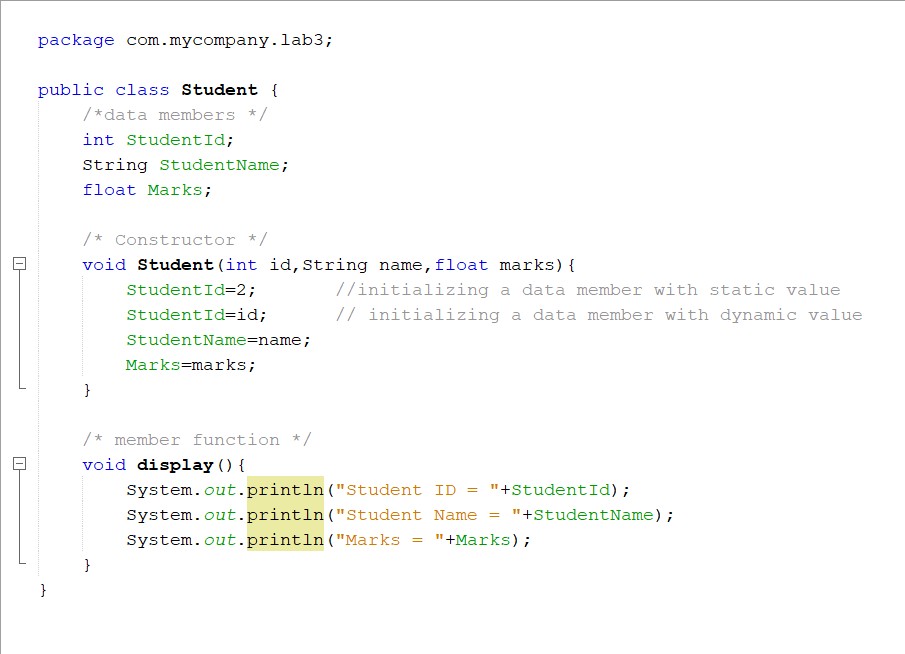
class <class\_name>{

//class body

}



Class declaration is enclosed within code blocks. In other words, the body of the class is enclosed between the area between the curly braces. In the class body, you can declare data members (also called as fields or instance variable), member functions (also called as behaviors or instance methods) and constructors or destructors.



*Figure 1: structure of a class*

Methods or member functions in a class can be of any type given below:

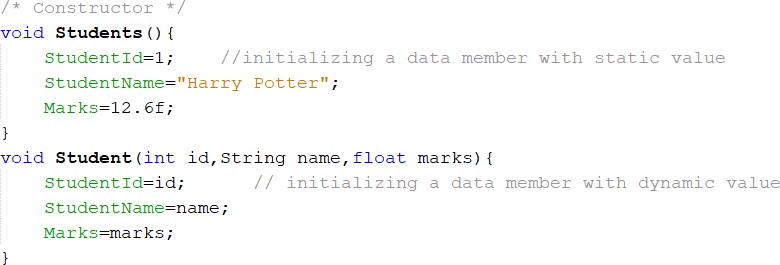
<return\_type> function\_name (<argument 1, argument2, ……., argumentN>) {}

<return\_type> function\_name (void) {}

void function\_name (<argument 1, argument2, ……., argumentN>) {} void function\_name (void) {}

**function\_names must not be a Java keyword.**

A class can have different methods and constructors. Constructors are specialized methods which are called only when an object is created. Constructors do not have a return type and they are of multiple types like default constructors (that do not accept any arguments) and parameterized constructors (that accepts arguments). We will discuss them further in details in lab 4. If a class does not have a constructor then Java invokes a builtin default constructor for object creation.



# Object:

*Figure 2: Constructors*

An entity is a real-world entity that has state and behavior e.g., chair, bike, marker, pen, table, car, etc. It can be physical or logical (tangible and intangible). The example of an intangible object is the banking system. An object has three characteristics:

* State: represents the data (value) of an object.
* Behavior: represents the behavior (functionality) of an object such as deposit, withdraw, etc.
* Identity: An object identity is typically implemented via a unique ID. The value of the ID is not visible to the external user. However, it is used internally by the JVM to identify each object uniquely.

An object is created using the new operator. On encountering the new operator, JVM allocates memory for the object and returns a reference or memory address of the allocated object. The reference or memory address is then stored in a variable. This variable is also called as reference variable. The syntax for creating an object is as follows:

<class\_name> <object\_name> = new <classname>();

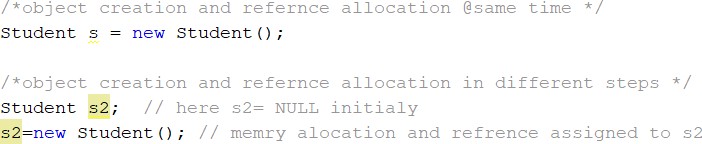
Where,

**new:** Is an operator that allocates the memory for an object at runtime.

**object\_name(or reference variable):** Is the variable that stores the reference of the object

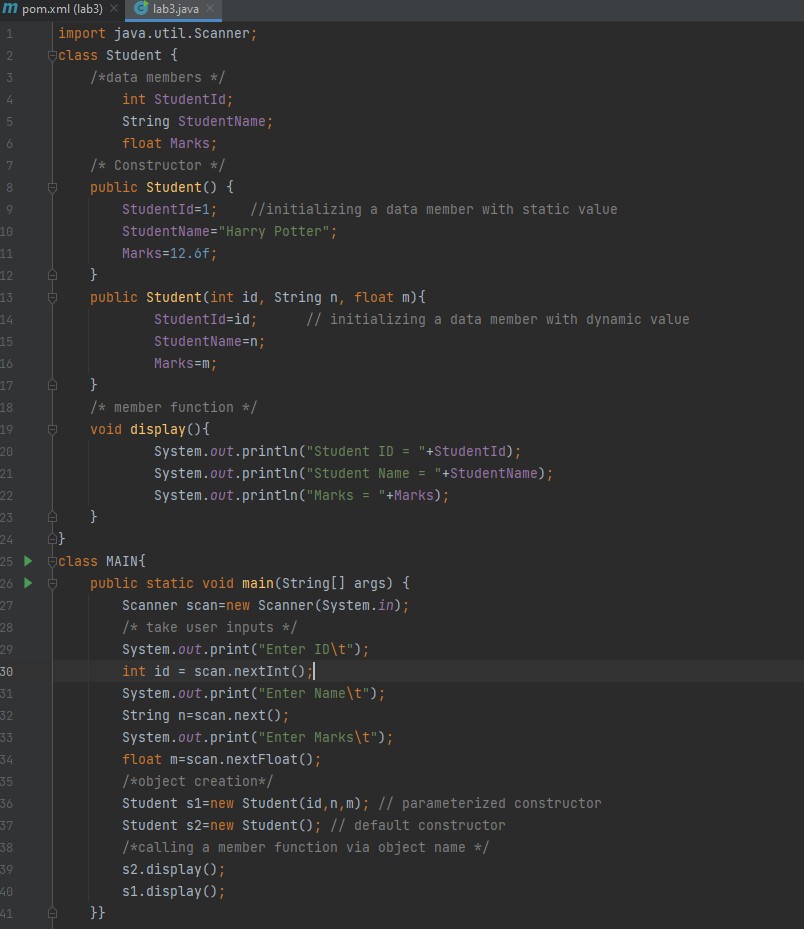
Creation of an object involves:

1. Declaration of reference variable
2. Creation of object and assigning its reference value to reference variable



# Example 1:

*Figure 3: object creation*



*Figure 4: student class*

# Access Modifiers in Java:

There are four types of Java access modifiers:

**Private:** The access level of a private modifier is only within the class. It cannot be accessed from outside the class.

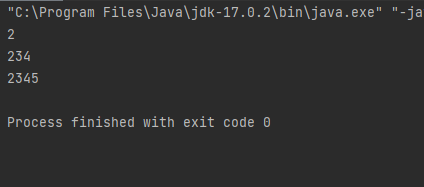
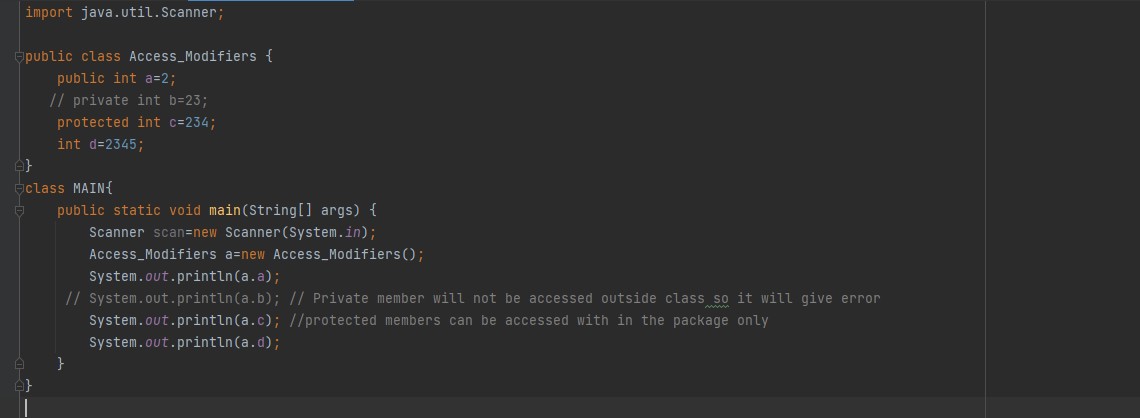
**Default:** The access level of a default modifier is only within the package. It cannot be accessed from outside the package. If you do not specify any access level, it will be the default.

**Protected:** The access level of a protected modifier is within the package and outside the package through child class. If you do not make the child class, it cannot be accessed from outside the package.

**Public:** The access level of a public modifier is everywhere. It can be accessed from within the class, outside the class, within the package and outside the package.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Access Modifier** | **Within class** | **Within package** | **Outside class** | **Outside package** |
| **Public** | Yes | Yes | Yes | Yes |
| **Private** | Yes | No | No | No |
| **Protected** | Yes | Yes | Yes | No |
| **default** | Yes | Yes | No | No |

# Example 2:



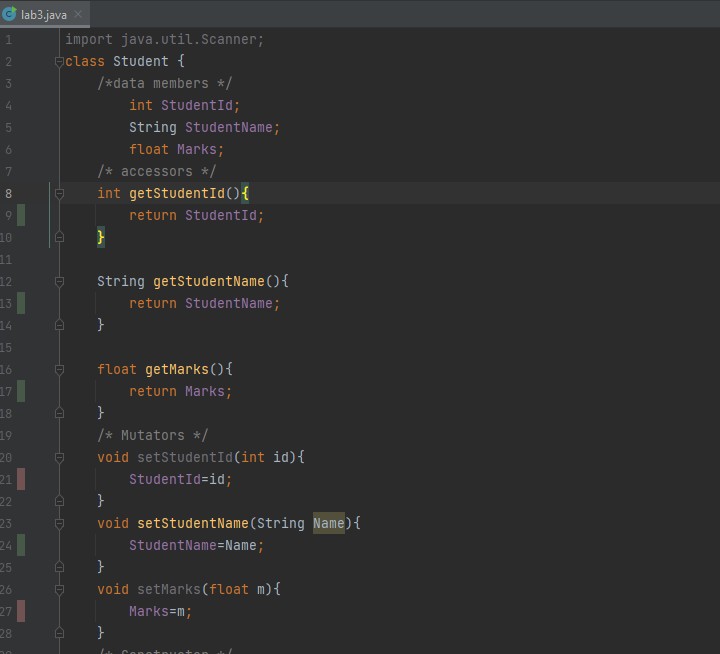
**Accessors & Mutators in Java:**

An **Accessor** method is commonly known as a get method or simply a getter. A property of the object is returned by the accessor method. They are declared as public. A naming scheme is followed by accessors, in other words they add a word to get in the start of the method name. They are used to return the value of a private field. The same data type is returned by these methods depending on their private field.

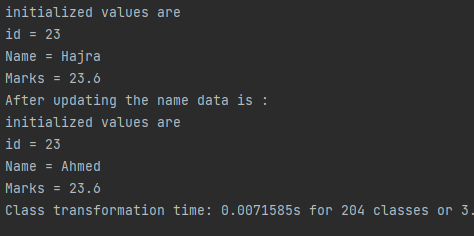
<return\_datatype> <function\_name>() {}

A **Mutator** method is commonly known as a set method or simply a setter. A Mutator method mutates things, in other words change things. It shows us the principle of encapsulation. They are also known as modifiers. They are easily spotted because they started with the word set. They are declared as public. Mutator methods do not have any return type and they also accept a parameter of the same data type depending on their private field. After that it is used to set the value of the private field.

Void <function\_name>(arguments){}



*Figure 6: accessors & mutators in Java*



*Figure 7: output*

## TASK – 01:

Write a class named Car that has the following data members:

* yearModel – an int field that hold the car’s year model.
* make – a String field that holds the make of the car.
* speed – an int field that holds the car’s current speed.

The class also should have the following constructor and other methods:

* constructor – that accepts the car’s year model and make as arguments. These values should be assigned to the object’s yearModel and make fields. The constructor also should assign 0 to the speed field.
* Accessors. Appropriate accessor methods should get the values stored in an object’s yearModel, make and speed fields.
* accelerate. The accelerate method should add 5 to the speed field each time it is called.
* brake. The brake method should subtract 5 from the speed field each time it is called. Demonstrate the class in a program that creates a Car object, and then calls the accelerate method five times. After each call to the accelerate method, get the current speed of the car

and display it. Then call the brake method five times. After each call to the brake method, get the current speed of the car and display it.

## TASK – 02:

Write a Java class Book with following features:

* + Instance variables:
    - **title** for the title of book of type String.
    - **author** for the author’s name of type String.
    - **price** for the book price of type double.
  + Constructor:
    - **public Book (String title, Author name, double price)**: A constructor with parameters, it creates the Author object by setting the the fields to the

passed values.

* + Instance methods:
    - **public void setTitle(String title)**: Used to set the title of book.
    - **public void setAuthor(String author)**: Used to set the name of author of book.
    - **public void setPrice(double price)**: Used to set the price of book.
    - **public double getTitle()**: This method returns the title of book.
    - **public double getAuthor()**: This method returns the author’s name of book.
    - **public String toString()**: This method printed out book’s details to the screen

Write a separate class **BookDemo** with a main () method creates a Book titled “Great Expectations” with author Charles Dickens and price 79.75.

## TASK – 03 Flight Reservation System :

## Implement a flight reservation system where users can book seats on flights. Create classes for flights, passengers, and seats. Design the system such that it handles multiple flights, each with different seat configurations. Ensure that the system handles cases like seat availability, double booking prevention, and passenger information management.

## Data Members:

## Flight: flightNumber, departureDateTime, arrivalDateTime, capacity, bookedSeats

## Passenger: passengerId, name, contactInfo

## Seat: seatNumber, seatType (economy, business, first class), isBooked

## Function Members:

## Flight: bookSeat(Passenger passenger, Seat seat), cancelBooking(Seat seat)

## Passenger: reserveSeat(Flight flight, Seat seat), cancelReservation(Flight flight)

## Seat: isSeatAvailable(), bookSeat(), releaseSeat()

## TASK – 04 Library Management System:

## Develop a library management system that allows users to borrow and return books. Create classes for books, users, and the library. Implement features such as checking out books, returning books, searching for books by title or author, and managing user accounts with borrowing history.

## Data Members:

## Book: bookId, title, author, availability

## User: userId, name, contactInfo, borrowedBooks

## Library: booksAvailable, usersRegistered

## Function Members:

## Book: checkoutBook(User user), returnBook(User user)

## User: borrowBook(Book book), returnBook(Book book)

## Library: searchBookByTitle(String title), searchBookByAuthor(String author), addUser(User user), removeUser(User user)

## TASK – 05:

Build an online shopping cart system that allows users to add items to their cart, remove items, and proceed to checkout. Create classes for products, users, and the shopping cart. Implement features like adding items to the cart, calculating total cost, applying discounts, and managing user orders.

**Data Members:**

Product: productId, name, price, quantityAvailable

User: userId, name, shippingAddress, cart

ShoppingCart: items, totalCost

**Function Members:**

Product: addToCart(int quantity), removeFromCart(int quantity)

User: viewCart(), checkout()

ShoppingCart: calculateTotalCost(), applyDiscount(), confirmOrder()

## TASK – 06:

## Design a banking system that includes classes for bank accounts, customers, and transactions. Implement features such as creating new accounts, depositing and withdrawing funds, transferring money between accounts, and viewing transaction history. Ensure that the system handles concurrency and data consistency.

## Data Members:

## Account: accountId, accountType, balance, transactions

## Customer: customerId, name, address, accounts

## Bank: accountsList, customersList

## Function Members:

## Account: deposit(double amount), withdraw(double amount), transfer(Account destinationAccount, double amount)

## Customer: addAccount(Account account), removeAccount(Account account)

## Bank: createAccount(Customer customer, AccountType accountType), closeAccount(Account account)

## TASK – 07:

Develop a parking lot management system that allows cars to enter, exit, and find parking spaces. Create classes for parking lot, vehicles, and parking spaces. Implement features like checking availability of parking spaces, allocating spaces for incoming vehicles, and calculating parking fees based on duration. Bonus points for handling special cases like handicapped parking or reserved spaces.

**Data Members:**

ParkingLot: capacity, availableSpaces, occupiedSpaces

Vehicle: vehicleNumber, vehicleType, entryTime, exitTime, parkingSpace

ParkingSpace: spaceNumber, isOccupied, vehicleParked

**sFunction Members:**

ParkingLot: parkVehicle(Vehicle vehicle), releaseSpace(ParkingSpace parkingSpace)

Vehicle: enterParkingLot(), exitParkingLot()

ParkingSpace: allocateSpace(Vehicle vehicle), releaseSpace()