

**Lab Manual- Java Class & Object**

**Prepared for**:

**Date:** 18th June 2024

**Prepared by:**

Document Name: Lab Manual **Document Number** SDLab333

**Contributor:**

Contents

[1. What are Java Class 3](#_Toc172281390)

[2. Create Two Java File Book and ExampleBook in Same Directory 3](#_Toc172281391)

[1. Create the Book Class 3](#_Toc172281392)

[2. Step 2: Create the ExampleBooks Class 3](#_Toc172281393)

[3. Step 3: Compile the Classes 4](#_Toc172281394)

[1. Step 4: Execute the ExampleBooks Class 4](#_Toc172281395)

[3. Create Two Java File Book and ExampleBook in Different Directory or Package 5](#_Toc172281396)

[1. Create Directory Structure: 5](#_Toc172281397)

[2. Create Book.java in Library Directory 5](#_Toc172281398)

[3. Create ExampleBook.java in main Directory 5](#_Toc172281399)

[4. Steps to Compile and Run with Different Packages 6](#_Toc172281400)

[4. Multiple Constructor 6](#_Toc172281401)

[1. Types of Constructors in Java 7](#_Toc172281402)

[2. Why Do We Need Constructors? 7](#_Toc172281403)

[3. Example Explanation 7](#_Toc172281404)

[5. Instance Variable and This Keyword 9](#_Toc172281405)

[6. Class Variable, Instance Variable and This Keyword 11](#_Toc172281406)

[1. Class Variables (Static Variables) 11](#_Toc172281407)

[2. Key Differences ( Class vs Instance Variable) 11](#_Toc172281408)

[3. Key Points: 13](#_Toc172281409)

[4. Summary: 13](#_Toc172281410)

[7. Class Method 13](#_Toc172281411)

[1. Why Use Getters? 13](#_Toc172281412)

# What are Java Class

# Create Two Java File Book and ExampleBook in Same Directory

### Create the Book Class

1. Open your preferred Java IDE (e.g., Eclipse, IntelliJ IDEA) or a simple text editor (e.g., Notepad++).
2. Create a new file named Book.java.
3. Define the Book class with the specified fields:

// Book.java

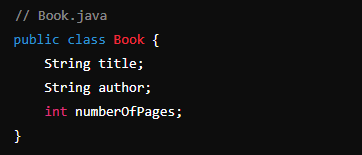
public class Book {

String title;

String author;

int numberOfPages;

}



### Step 2: Create the ExampleBooks Class

1. Create another new file named ExampleBooks.java.
2. Define the ExampleBooks class with the main method:

// ExampleBooks.java

public class ExampleBooks {

public static void main(String[] args) {

// Create a new instance of the Book class

Book b = new Book();

// Set the fields of the Book instance

b.title = "Thinking in Java";

b.author = "Bruce Eckel";

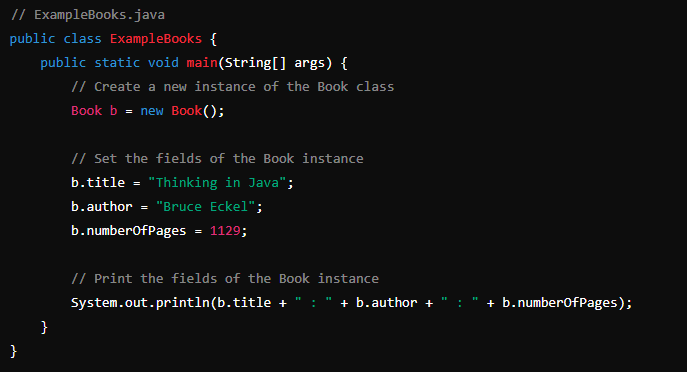
b.numberOfPages = 1129;

// Print the fields of the Book instance

System.out.println(b.title + " : " + b.author + " : " + b.numberOfPages);

}

}



### Step 3: Compile the Classes

1. Open the command prompt (Windows) or terminal (macOS/Linux).
2. Navigate to the directory where you saved the Book.java and ExampleBooks.java files.
3. Compile both Java files using the javac command:



This will generate Book.class and ExampleBooks.class files in the **same directory.**

### Step 4: Execute the ExampleBooks Class

1. Run the ExampleBooks class using the java command:

java ExampleBooks



# Create Two Java File Book and ExampleBook in Different Directory or Package

## Create Directory Structure:

* Create a directory library for Book.java.
* Create a directory main for ExampleBooks.java.

## Create Book.java in Library Directory

**Book.java** (in package library)

package library;

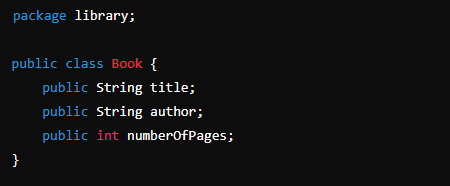
public class Book {

public String title;

public String author;

public int numberOfPages;

}



## Create ExampleBook.java in main Directory

package main;

import library.Book;

public class ExampleBooks {

public static void main(String[] args) {

// Create a new instance of the Book class

Book b = new Book();

// Set the fields of the Book instance

b.title = "Thinking in Java";

b.author = "Bruce Eckel";

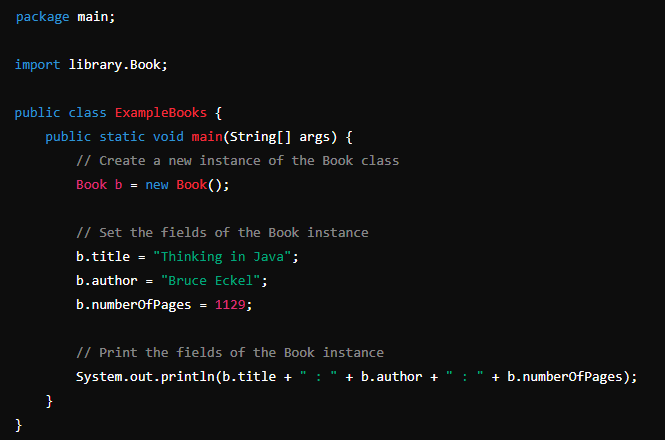
b.numberOfPages = 1129;

// Print the fields of the Book instance

System.out.println(b.title + " : " + b.author + " : " + b.numberOfPages);

}

}



## Steps to Compile and Run with Different Packages

**javac library/Book.java**

**javac main/ExampleBooks.java**



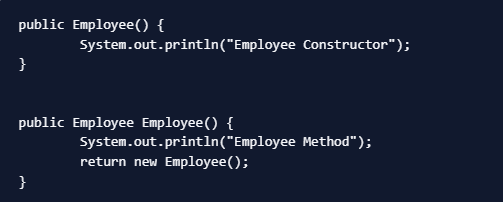
**java main.ExampleBooks**



# Multiple Constructor

A **constructor** is a special function in a class that helps create and set up an object when it is first made.

Whenever we use **new** keyword to create an instance of a class, the constructor is invoked and the object of the class is returned. Since constructor can only return the object to class, it’s implicitly done by java runtime and **we are not supposed to add a return type to it**. **If we add a return type to a constructor, then it will become a method of the class**. This is the way java runtime distinguish between a **normal method** and a **constructor**. Let’s assume we have following code in Employee class.



Here the **first one is a constructor**, notice that there is **no return type** and no return statement. The **second one is a normal method** where we are again **calling the first constructor** to get Employee instance and return it. It’s recommended to not have method name same as the class name because it creates confusion.

### [Types of Constructors in Java](https://www.digitalocean.com/community/tutorials/constructor-in-java#types-of-constructor-in-java)

There are three types of constructor in java.

* Default Constructor
* No-Args constructor
* Parameterized constructor

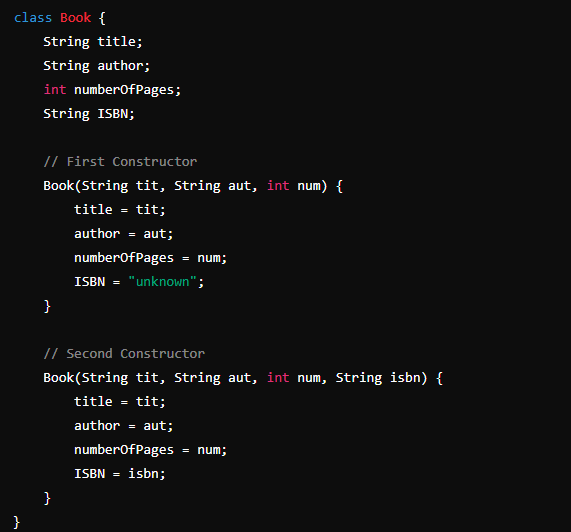
### Why Do We Need Constructors?

When you create an object, you want it to have specific values or settings right from the start. Constructors allow you to do this easily.

### Example Explanation

Imagine you have a class called **Book** that represents a book. You want each book to have a **title**, an **author**, a **number of pages**, and an **ISBN**.

Here's the **Book class** with its **two constructors**:



ExampleBooks.java

// ExampleBooks.java

// ExampleBooks.java

class ExampleBooks {

public static void main(String[] args)

{

Book b1 = new Book("Thinking in Java","Bruce Eckel",1129);

System.out.println(b1.title + " : " + b1.author +" : " + b1.numberOfPages + " : " + b1.ISBN);

Book b2 = new Book("Thinking in Java","Bruce Eckel",1129,"0-13-027363-5");

System.out.println(b2.title + " : " + b2.author +" : " + b2.numberOfPages + " : " + b2.ISBN);

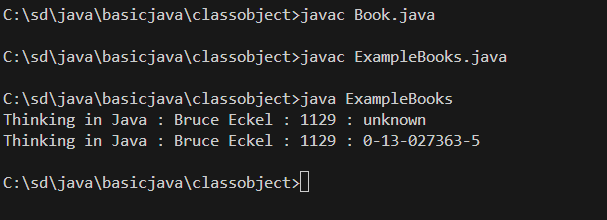
}

}

**javac Book.java**

**javac ExampleBooks.java**

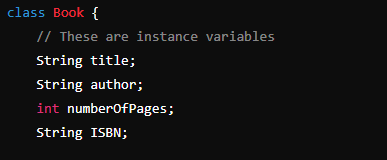
**java ExampleBooks**

****

# Instance Variable and This Keyword

#### What are Instance Variables?

Instance variables are variables that belong to an instance of a class. Each object created from the class has its own copy of these variables. They are declared inside a class but outside any method, constructor, or block.



#### Characteristics of Instance Variables:

1. **Scope:** Instance variables are accessible throughout the class in which they are declared, and they are accessible by all methods in that class.
2. **Initialization:** They can be initialized when declared or inside constructors. If not explicitly initialized, they get default values (e.g., null for objects, 0 for integers).
3. **Each Object Has Its Own Copy:** Every object created from the class has its own separate copy of the instance variables. Changes made to instance variables of one object do not affect other objects.

#### Purpose of this Keyword

The this keyword in Java is used to refer to the **current instance of the class**. It helps to distinguish between **instance variables** (properties of the object) and **parameters** (variables passed to a method or constructor) when they have the same name.

#### Why Use this?

1. **Clarity:** It makes your code more readable and clear by explicitly stating that you are referring to the instance variables.
2. **Avoid Confusion:** It avoids confusion between instance variables and parameters or local variables that have the same name.

**Book.java**

class Book {

    // These are instance variables

    String title;

    String author;

    int numberOfPages;

    String ISBN;

    // First Constructor

    Book(String title, String author, int numberOfPages) {

        this.title = title;

        this.author = author;

        this.numberOfPages = numberOfPages;

        this.ISBN = "unknown";

    }

    // Second Constructor

    Book(String title, String author, int numberOfPages, String ISBN) {

        this.title = title;

        this.author = author;

        this.numberOfPages = numberOfPages;

        this.ISBN = ISBN;

    }

}

**ExampleBooks.java**

public class ExampleBooks {

    public static void main(String[] args) {

        // Create a book using the first constructor

        Book book1 = new Book("Harry Potter", "J.K. Rowling", 500);

        // Create a book using the second constructor

        Book book2 = new Book("Harry Potter", "J.K. Rowling", 500, "1234567890");

        // Print book details

        System.out.println("Book 1:");

        System.out.println("Title: " + book1.title); // Accessing instance variable 'title'

        System.out.println("Author: " + book1.author); // Accessing instance variable 'author'

        System.out.println("Number of Pages: " + book1.numberOfPages); // Accessing instance variable 'numberOfPages'

        System.out.println("ISBN: " + book1.ISBN);  // Accessing instance variable 'ISBN'

        System.out.println("\nBook 2:");

        System.out.println("Title: " + book2.title); // Accessing instance variable 'title'

        System.out.println("Author: " + book2.author); // Accessing instance variable 'author'

        System.out.println("Number of Pages: " + book2.numberOfPages); // Accessing instance variable 'numberOfPages'

        System.out.println("ISBN: " + book2.ISBN); // Accessing instance variable 'ISBN'

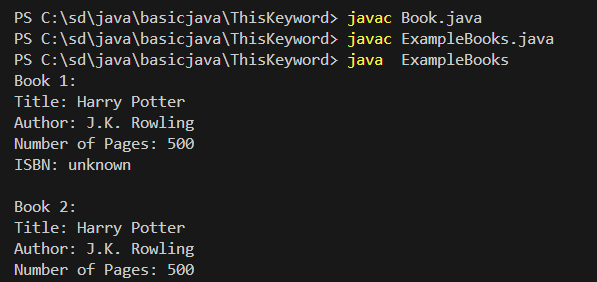
    }

}

**javac Book.java**

**javac ExampleBooks.java**

**java ExampleBooks**

****

# Class Variable, Instance Variable and This Keyword

### Class Variables (Static Variables)

1. **Definition:**
   * Declared with the **static** keyword within a class, but outside any method, constructor, or block.
   * Only one copy of these variables is created, shared among all instances of the class.
2. **Scope:**
   * Accessible by all methods and constructors of the class.
   * Can be accessed directly using the class name, without creating an instance of the class.
3. **Lifetime:**
   * Exist for the lifetime of the program, as long as the class is loaded in memory

### Key Differences ( Class vs Instance Variable)

1. **Memory Allocation:**
   * **Instance Variables:** Allocated memory in the heap, separately for each object.
   * **Class Variables:** Allocated memory once in the method area (static area), shared by all objects.
2. **Initialization:**
   * **Instance Variables:** Initialized when the object is created, either through constructors or explicitly in the class definition.
   * **Class Variables:** Initialized when the class is first loaded into memory, either through explicit initialization or static blocks.
3. **Access:**
   * **Instance Variables:** Accessed through object references.
   * **Class Variables:** Accessed through the class name, without needing an object.
4. **Use Cases:**
   * **Instance Variables:** Used for properties that should differ between objects (e.g., title and author of a book).
   * **Class Variables:** Used for properties that should be the same across all objects (e.g., keeping track of the number of book objects created).

Book.java

class Book {

    // Static variable (class variable)

    static int bookCount = 0;

    // Instance variables

    String title;

    String author;

    int numberOfPages;

    String ISBN;

    // Constructor

    Book(String title, String author, int numberOfPages) {

        this.title = title;

        this.author = author;

        this.numberOfPages = numberOfPages;

        this.ISBN = "unknown";

        bookCount++; // Increment the static variable whenever a new Book object is created

    }

    // Another constructor with ISBN

    Book(String title, String author, int numberOfPages, String ISBN) {

        this.title = title;

        this.author = author;

        this.numberOfPages = numberOfPages;

        this.ISBN = ISBN;

        bookCount++; // Increment the static variable whenever a new Book object is created

    }

    // Static method to get the count of books

    static int getBookCount() {

        return bookCount;

    }

}

ExampleBooks.java

public class ExampleBooks {

    public static void main(String[] args) {

        // Create book objects

        Book book1 = new Book("Harry Potter", "J.K. Rowling", 500);

        Book book2 = new Book("Harry Potter", "J.K. Rowling", 500, "1234567890");

        // Accessing the static variable using the class name (preferred way)

        System.out.println("Total number of books: " + Book.getBookCount()); // Outputs: 2

        // Accessing static method through an instance (not recommended, but valid)

        System.out.println("Total number of books (accessed through book1): " + book1.getBookCount()); // Outputs: 2

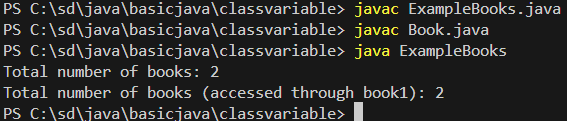
    }

}

javac Book.java

javac ExampleBooks.java

java ExampleBooks



### Key Points:

1. **Shared Across Instances:**
   * The bookCount variable is shared among all Book instances. Whenever a new Book object is created, bookCount is incremented.
2. **Access via Class Name:**
   * The static method getBookCount() can be called using the class name Book.getBookCount(), which returns the total number of Book instances created.
3. **Access via Instance:**
   * Accessing the static method via an instance (e.g., book1.getBookCount()) is possible but not recommended, as it can be misleading. It implies that the count is related to the specific instance, which it is not.

### Summary:

* **Static (Class) Variables:** Declared with the static keyword. Shared among all instances of the class. Accessible via the class name.
* **Instance Variables:** Declared without the static keyword. Each object has its own copy. Accessible via object instances.

In essence, any variable declared with the static keyword is a class variable, and it has behaviors and properties that are distinct from those of instance variables.

# Class Method

Note : **getters** are indeed methods in Java. They are specifically used to access the values of private instance variables from outside the class. In Java, private variables cannot be accessed directly outside the class they are declared in. Getters allow controlled access to these variables.

### Why Use Getters?

1. **Encapsulation**: By keeping variables private and providing public getters (and setters if needed), you protect the internal state of the object. This allows you to change the internal implementation without affecting external code that uses your class.
2. **Read-Only Access**: Getters can provide read-only access to private variables. This means external code can retrieve the value but not modify it directly.
3. **Consistency and Validation**: You can add additional logic inside getters if needed (e.g., formatting or validation before returning the value).

**Book.java**

public class Book {

    // Instance variables

    private String title;

    private String author;

    private int pages;

    private String isbn;

    private static int bookCount = 0; // Static variable to keep track of the number of books

    // Constructor with ISBN

    public Book(String title, String author, int pages, String isbn) {

        this.title = title;

        this.author = author;

        this.pages = pages;

        this.isbn = isbn;

        bookCount++; // Increment the count of books whenever a new book is created

    }

    // Constructor without ISBN

    public Book(String title, String author, int pages) {

        this(title, author, pages, ""); // Call the other constructor with an empty ISBN

    }

    // Static method to get the total number of books

    public static int getBookCount() {

        return bookCount;

    }

    // Getters for instance variables (optional, but useful)

    public String getTitle() {

        return title;

    }

    public String getAuthor() {

        return author;

    }

    public int getPages() {

        return pages;

    }

    public String getIsbn() {

        return isbn;

    }

}

 **getBookCount()**: This is a static method that returns the total number of Book objects created. Since bookCount is a static variable, it’s shared among all instances of the Book class.

 **Getters**: Methods like getTitle(), getAuthor(), getPages(), and getIsbn() provide access to the private instance variables. They are optional but are useful for accessing the values of these variables outside the class.

**ExampleBoos.java**

public class ExampleBooks {

    public static void main(String[] args) {

        // Create book objects

        Book book1 = new Book("Harry Potter", "J.K. Rowling", 500);

        Book book2 = new Book("Harry Potter", "J.K. Rowling", 500, "1234567890");

        // Accessing the static variable

        System.out.println("Total number of books: " + Book.getBookCount()); // Outputs: 2

        // Accessing static variable through an instance (not recommended)

        System.out.println("Total number of books (accessed through book1): " + book1.getBookCount()); // Outputs: 2

        System.out.println("Title: " + book1.getTitle());

        System.out.println("Author: " + book1.getAuthor());

        System.out.println("Pages: " + book1.getPages());

        System.out.println("ISBN: " + book1.getIsbn());

    }

}

**javac Book.java ExampleBooks.java**

**java ExampleBooks**

 **System.out.println("Total number of books: " + Book.getBookCount());**: Calls the static method getBookCount() from the Book class to get the total number of books created and prints it. This should output 2, as two Book objects have been created.

 **System.out.println("Total number of books (accessed through book1): " + book1.getBookCount());**: Demonstrates accessing the static method getBookCount() through an instance (book1). While it's not recommended to access static methods through instances, it still works and will print 2.

