TG ACADEMIC RESOURCE SERIES

JAVA PROGRAMMING

Object-Oriented Programming
Classes and Objects

SUBJECT	Java Programming / Object-Oriented Programming
LEVEL	BTech / BCA / MCA / BSc / MSc / MTech
RESOURCE TYPE	Sample learning note
PUBLICATION ID	JAVA-001 | Version 1.0

Prepared by Tanmoy Gupta
Computer Science Educator & Industry Professional

This short sample introduces the relationship between a class and its objects, demonstrates object creation and method calls in Java, and includes a small state-tracing exercise.

A focused preview of a structured programming-learning approach—not a complete OOP course or reference book.

1. Learning Objectives

After studying this note, you should be able to:

- Define a class and an object in Java.
- Identify fields (instance variables) and methods.
- Create objects using the **new** keyword.
- Access an object's members using the dot operator.
- Distinguish a class definition from an object created at runtime.

2. What Is a Class?

A **class** is a user-defined reference type that describes the data and behaviour its objects can have. It acts as a blueprint: it declares fields to represent state and methods to define behaviour.

3. What Is an Object?

An **object** is a runtime instance of a class. Each object has its own instance state. Objects created from the same class share the class's method definitions, but their instance fields can hold different values.

Blueprint and instances

Consider a **Student** class. It may define fields such as name and marks, and a method such as displayDetails(). One Student object may represent Asha with 82 marks; another may represent Ravi with 91 marks.

Class vs. Object

Class	Object
A definition or blueprint.	A runtime instance of a class.
Declares fields and methods.	Has its own instance-field values.
Declared using the class keyword.	Commonly created using new.

Remember: A class declaration does not mean that an object has already been created. Object creation happens separately at runtime.

4. Java Example: Create and Use Objects

The following program defines a Student class, creates two Student objects, assigns different values to their fields, and calls an instance method for each object.

```
class Student { String name; int marks; void displayDetails() { System.out.println(name + " scored " + marks); } } public class Main { public static void main(String[] args) { Student s1 = new Student(); Student s2 = new Student(); s1.name = "Asha"; s1.marks = 82; s2.name = "Ravi"; s2.marks = 91; s1.displayDetails(); s2.displayDetails(); } }
```

Expected Output

```
Asha scored 82
Ravi scored 91
```

Explanation

1. **class Student** declares the fields name and marks and the method displayDetails().
2. **new Student()** creates a new Student object and returns a reference to it.
3. The references s1 and s2 refer to two distinct objects.
4. Assignments through s1 update the first object's fields; assignments through s2 update the second object's fields.
5. Calling s1.displayDetails() and s2.displayDetails() invokes the method for the corresponding object.

The method uses the current object's instance fields. Thus, the same method call produces different output for s1 and s2 because their states differ.

5. Object-State Trace

Track the state of each object after the assignments in main().

Statement executed	s1.name	s1.marks	s2.name	s2.marks
After object creation	null	0	null	0
s1.name = "Asha";	Asha	0	null	0
s1.marks = 82;	Asha	82	null	0
s2.name = "Ravi";	Asha	82	Ravi	0
s2.marks = 91;	Asha	82	Ravi	91

Default field values

For a newly created object, Java assigns default values to its instance fields before the constructor body runs. For the field types in this example, String (a reference type) initially has the value null and int initially has the value 0. Local variables do not receive these automatic default values.

6. Important Terms

Instance variable: A non-static field declared in a class; each object has its own value.
Instance method: A non-static method called on an object.
Reference variable: A variable that can hold a reference to an object, such as s1 or s2.
Dot operator (.): Used to access accessible members through an object reference.

A useful distinction

In **Student s1 = new Student();**, Student is the type, s1 is the reference variable, and new Student() creates the object. The variable s1 is not the object itself; it holds the reference used to access that object.

7. Check Your Understanding

Attempt these short questions to check whether you can apply the concepts rather than only recall the definitions.

Practice Questions

Q1. A class Book has fields title and price. How many separate title and price field values are present when three Book objects are created?

Q2. What is the role of the new keyword in the statement **Book b = new Book();**?

Q3. Suppose two objects, p and q, are created from the same class. If p.score is changed, does q.score automatically change? Explain.

Q4. Identify the class name, reference variable and object-creation expression in **Student s = new Student();**.

Suggested Answers (brief)

A1. Three separate title fields and three separate price fields—one pair per object.

A2. It creates an object of the specified class and yields a reference to that object.

A3. No. Each object has its own instance field. Changing p.score does not by itself change q.score.

A4. Class/type: Student; reference variable: s; object-creation expression: new Student().

Where to Go Next

This sample covers the first building block of Java OOP. A complete OOP sequence can extend into constructors, method overloading, encapsulation, inheritance, polymorphism, abstraction, interfaces, object relationships and design exercises.

In guided classes: students can practise writing classes, tracing object state, debugging common errors, and applying OOP principles through progressively challenging programming tasks.

About this resource
TG Academic Resource Series | JAVA-001 | Version 1.0
Prepared by Tanmoy Gupta — Computer Science Educator & Industry Professional

Original educational material prepared for student learning and revision. Redistribution or commercial reproduction without permission is not authorised.