https://chatgpt.com/share/6936f596-cebc-8012-9e21-17ba0d5cf779

Bro… you really switching lanes mid-traffic. Fine.
Here, clean first 5 experiments' codes, all written properly, no PDF, no drama.
Take them before I pretend I didn't see your message.

✔ EXPERIMENT 1 — Patterns using loops

```
class Patterns {
    public static void main(String[] args) {
        for (int i = 1; i <= 5; i++) {
            for (int j = 1; j <= i; j++) {
                System.out.print("* ");
            }
            System.out.println();
        }
    }
}
```

✔ EXPERIMENT 2 — Employee class + create instance

```
class Employee {
    String name;
    int id;

    Employee(String name, int id) {
        this.name = name;
        this.id = id;
    }

    public static void main(String[] args) {
        Employee e = new Employee("Rutuj", 101);
        System.out.println(e.name + " " + e.id);
    }
}
```

✔ EXPERIMENT 3 — Book class with parameterized constructor

```
class Book {
    String title;
    String author;
    double price;

    Book(String title, String author, double price) {
        this.title = title;
        this.author = author;
        this.price = price;
    }

    void show() {
        System.out.println(title + " | " + author + " | " + price);
    }

    public static void main(String[] args) {
        Book b1 = new Book("Java", "James", 500);
        Book b2 = new Book("AI", "Rutuj", 900);
        b1.show();
        b2.show();
    }
}
```

✔ EXPERIMENT 4 — Encapsulation in Employee

```
class Employee {
    private String name;
    private int salary;

    public void setName(String name) {
        this.name = name;
    }
    public String getName() {
        return name;
    }
    public void setSalary(int salary) {
        this.salary = salary;
    }
    public int getSalary() {
        return salary;
    }

    public static void main(String[] args) {
        Employee e = new Employee();
        e.setName("Aman");
        e.setSalary(25000);

        System.out.println(e.getName() + " " + e.getSalary());
    }
}
```

✔ EXPERIMENT 5 — Default values of static & instance variables

```
class Demo {
    static int a;     // static variable
    int b;            // instance variable

    public static void main(String[] args) {
        Demo d = new Demo();

        System.out.println(a);  // default = 0
        System.out.println(d.b); // default = 0
    }
}
```

Say “next 5” if you want Experiments 6 to 10.
Say “all remaining” if you want me to drop the whole arsenal.

Alright Mr. “no shortcuts”, fine… here are Experiments 6 to 10 written clean, proper, full code.
Don’t act like you’re not saving these for your journal submission.

✔ EXPERIMENT 6 — Boxing & Unboxing (proper demo)

```
class BoxingUnboxingDemo {

    public static void main(String[] args) {

        int a = 10;                    // primitive
        Integer obj = Integer.valueOf(a); // boxing
        int b = obj.intValue();           // unboxing

        System.out.println("Primitive a: " + a);
        System.out.println("Boxed Integer obj: " + obj);
        System.out.println("Unboxed b: " + b);

        Integer auto = a;     // auto-boxing
        int auto2 = auto;     // auto-unboxing

        System.out.println("Auto-boxing: " + auto);
        System.out.println("Auto-unboxing: " + auto2);
    }
}
```

✔ EXPERIMENT 7 — Different Types of Inheritance (single, multilevel, hierarchical)

```
// Single Inheritance
class A {
    void showA() {
        System.out.println("Class A");
    }
}

class B extends A {
    void showB() {
        System.out.println("Class B");
    }
}
```

```
// Multilevel Inheritance
class X {
    void showX() {
        System.out.println("Class X");
    }
}

class Y extends X {
    void showY() {
        System.out.println("Class Y");
    }
}

class Z extends Y {
    void showZ() {
        System.out.println("Class Z");
    }
}

// Hierarchical Inheritance
class Parent {
    void display() {
        System.out.println("Parent class");
    }
}

class Child1 extends Parent {
    void show1() {
        System.out.println("Child 1");
    }
}

class Child2 extends Parent {
    void show2() {
        System.out.println("Child 2");
    }
}

class TestInheritance {
    public static void main(String[] args) {

        // Single
        B b = new B();
        b.showA();
        b.showB();

        // Multilevel
        Z z = new Z();
        z.showX();
        z.showY();
        z.showZ();

        // Hierarchical
        Child1 c1 = new Child1();
        c1.display();
        c1.show1();

        Child2 c2 = new Child2();
        c2.display();
        c2.show2();
    }
}
```

✔ EXPERIMENT 8 — Multilevel Inheritance using Packages

package p1

```
package p1;

public class A {
    public void msgA() {
        System.out.println("Inside class A (package p1)");
    }
}
```

package p2

```
package p2;
import p1.A;

public class B extends A {
    public void msgB() {
        System.out.println("Inside class B (package p2)");
    }
}
```

package p3 (Main)

```
package p3;
import p2.B;

public class TestPackages {
    public static void main(String[] args) {
        B obj = new B();
        obj.msgA();
        obj.msgB();
    }
}
```

Folder Structure must be:

```
p1/A.java
p2/B.java
p3/TestPackages.java
```

Compile with:

```
javac p1/A.java
javac p2/B.java
javac p3/TestPackages.java
```

Run:

```
java p3.TestPackages
```

✔ EXPERIMENT 9 — Array of Employee Objects

```
class Employee {
    String name;
    int id;

    Employee(String name, int id) {
        this.name = name;
        this.id = id;
    }
    void display() {
        System.out.println(name + " " + id);
    }
}

class EmployeeArray {
    public static void main(String[] args) {

        Employee[] arr = new Employee[3];

        arr[0] = new Employee("Aman", 101);
        arr[1] = new Employee("Siya", 102);
        arr[2] = new Employee("Ravi", 103);

        for (int i = 0; i < arr.length; i++) {
            arr[i].display();
        }
    }
}
```

✔ EXPERIMENT 10 — Access Modifiers Demo

```
class DemoAccess {

    public int pub = 10;
    private int pri = 20;
    protected int pro = 30;
    int def = 40;    // default access

    void display() {
        System.out.println("Public: " + pub);
        System.out.println("Private: " + pri);
        System.out.println("Protected: " + pro);
        System.out.println("Default: " + def);
    }
}

class TestAccess {
    public static void main(String[] args) {

        DemoAccess d = new DemoAccess();
        d.display();

        System.out.println("Accessing public: " + d.pub);
        // d.pri;  // cannot access private
        System.out.println("Accessing protected: " + d.pro);
        System.out.println("Accessing default: " + d.def);
    }
}
```

If you want 11 + 12, say
"Next 2"
and I’ll drop them clean too.

Alright, final stretch. Here are Experiment 11 and 12, full, clean, journal-ready code.
No shortcuts, no half-baked stuff. Let’s finish this syllabus before you pass out.

✔ EXPERIMENT 11 — Vehicle Class Hierarchy + Polymorphism

```
class Vehicle {
    void move() {
        System.out.println("Vehicle is moving");
    }
}

class Car extends Vehicle {
    void move() {
        System.out.println("Car is driving on the road");
    }
}

class Bike extends Vehicle {
    void move() {
        System.out.println("Bike is speeding through traffic");
    }
}
```

```
class Truck extends Vehicle {
    void move() {
        System.out.println("Truck is hauling cargo");
    }
}

class TestPolymorphism {
    public static void main(String[] args) {

        Vehicle v1 = new Vehicle();
        Vehicle v2 = new Car();
        Vehicle v3 = new Bike();
        Vehicle v4 = new Truck();

        v1.move();
        v2.move();
        v3.move();
        v4.move();
    }
}
```

This demonstrates runtime polymorphism using method overriding.

✔ EXPERIMENT 12 — User-Defined Checked & Unchecked Exceptions

Custom Checked Exception

```
class MyCheckedException extends Exception {
    MyCheckedException(String msg) {
        super(msg);
    }
}
```

Custom Unchecked Exception

```
class MyUncheckedException extends RuntimeException {
    MyUncheckedException(String msg) {
        super(msg);
    }
}
```

Main Program

```
class TestExceptions {

    static void checkAge(int age) throws MyCheckedException {
        if (age < 18) {
            throw new MyCheckedException("Age below 18. Not allowed!");
        }
    }

    static void checkMarks(int marks) {
        if (marks < 0) {
            throw new MyUncheckedException("Marks cannot be negative!");
        }
    }

    public static void main(String[] args) {

        try {
            checkAge(15);
        } catch (MyCheckedException e) {
            System.out.println("Caught Checked Exception: " + e.getMessage());
        }

        try {
            checkMarks(-5);
        } catch (MyUncheckedException e) {
            System.out.println("Caught Unchecked Exception: " + e.getMessage());
        }

        System.out.println("Program finished smoothly");
    }
}
```

If you want: • viva questions for each experiment
• output screenshots
• explanation PDFs
• or a combined final “Journal-ready” document

Just say it.