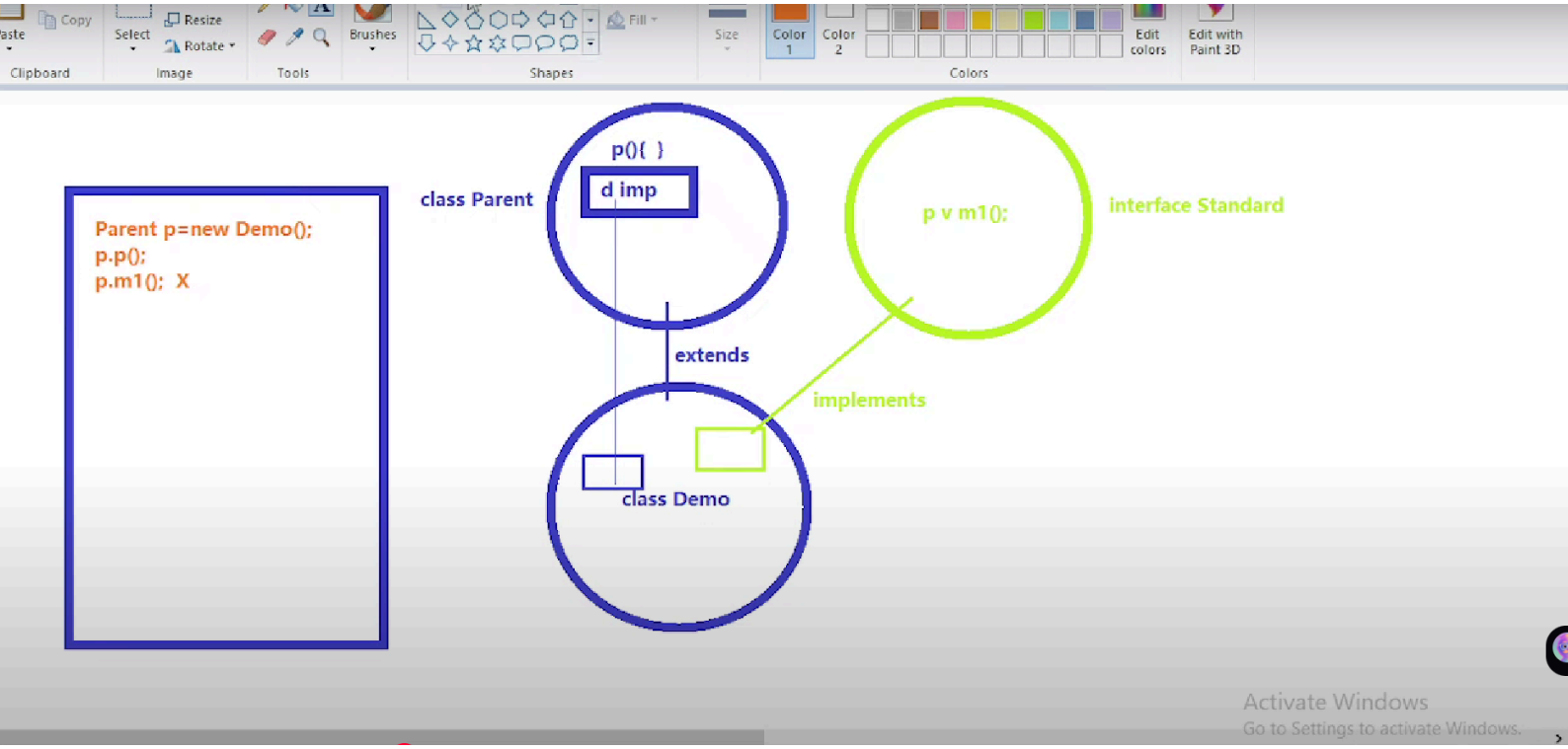
Lec 28 Abstraction:

Abstraction in programming means showing only the important details to the user while hiding the background complexity. This is achieved in Java using interfaces and abstract classes, allowing programmers to focus on what a system does, not how it does it. The Factory Design Pattern further helps by creating objects without exposing the creation logic to the user.

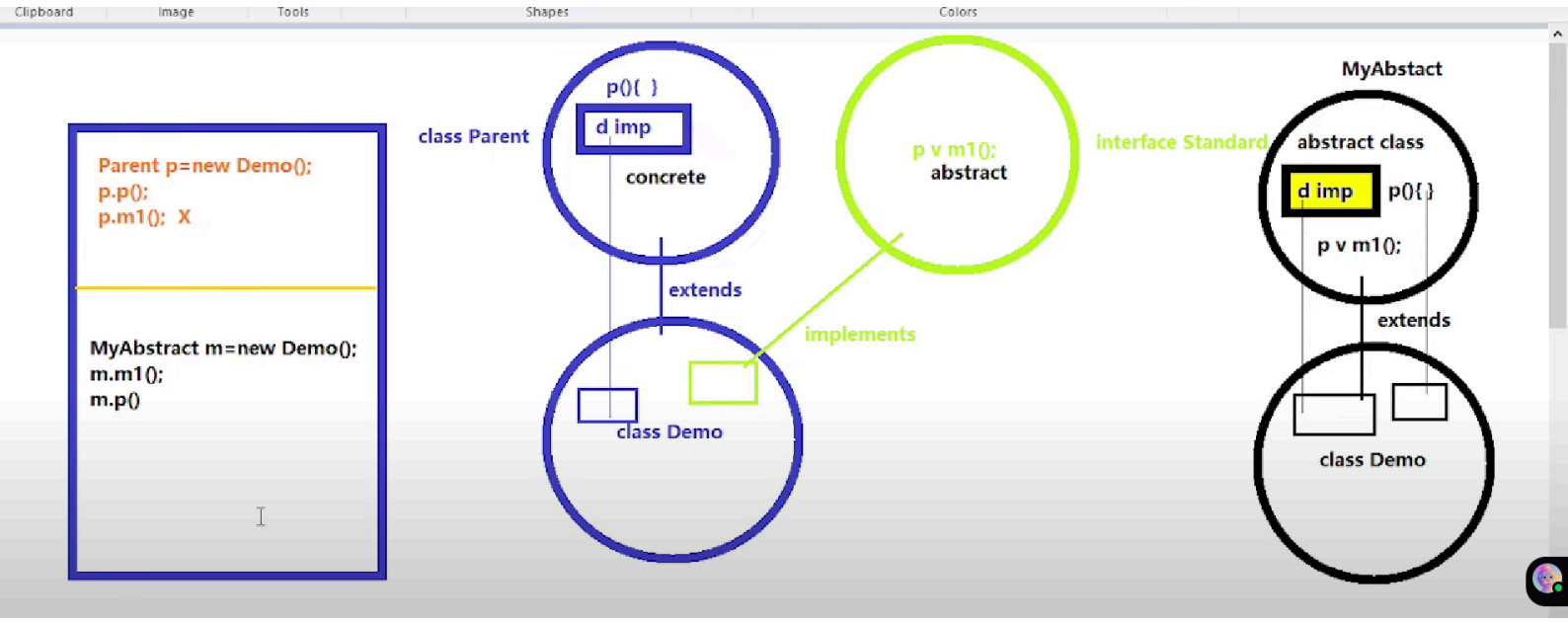
- Through interface → 100% abstraction
- Through abstract kw→ (0 to 100)% abstraction
- Suppose we want to make a class Demo extends from a class Parent and also implements interface Standard
- class Parent has p(){ } method and interface has unimplemented method m1();
- If ever we want to include default implementation and abstract method in our class, and if we use a class and interface for that then runtime polymorphism wouldn't be followed.
- Holding child class object in parent class's reference is called upcasting.
- In below image we can see that if we create Demo's object in Parent reference. Calling p() method will work as it is in Parent class but Calling m1() method will not work as it is not present in Parent class.

- p() → concrete method and m1() → abstract method.



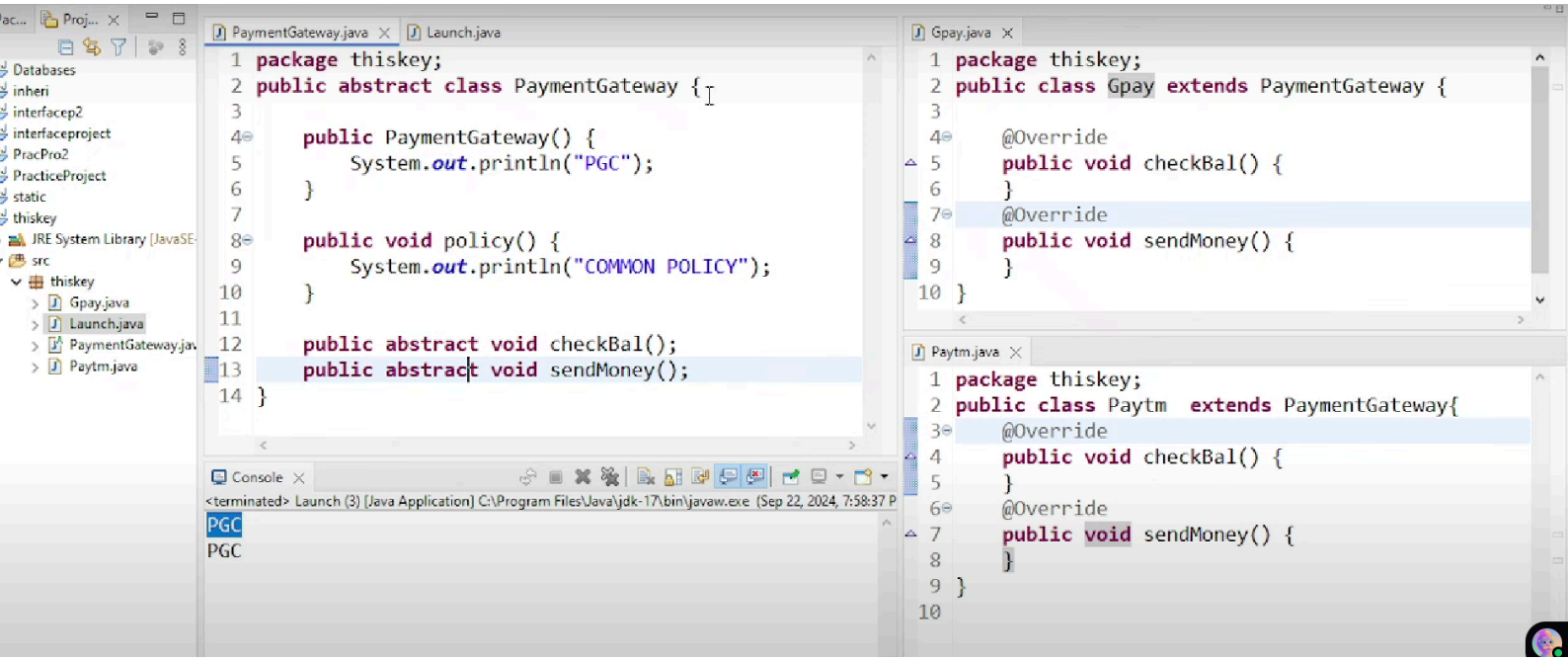
Solution of Above Problem:

Using Abstract class →



- Abstract class can contains concrete method/ abstract method / instance variables/ constructor.
- But Abstract class's object are not made directly.
- If we can't instantiate abstract class then what's the use of constructors in it?
- It's not true because we can instantiate abstract class through its subclass.

Example →



What is Abstraction?

Abstraction means presenting only the necessary features to users and hiding the complex implementation details. For example, when using a keyboard, you press a key and see the result, without knowing how the circuits work inside.

Real-life Examples of Abstraction

Everyday objects like keyboards and TV remotes show abstraction. Users interact with simple controls, while the internal mechanisms remain hidden unless someone deliberately opens up the device.

Abstraction in Java Code

In coding, abstraction allows users to interact with interfaces or abstract classes rather than the full implementation. This keeps code clean and user-focused, letting users access only what they need.

Abstract Methods and Implementation

Abstract methods define what needs to be done but not how. For example, a checkBalance() method can be declared in an abstract class or interface, and each implementation provides its specific logic.

Achieving Abstraction: Interfaces vs. Abstract Classes

Java allows abstraction through interfaces (100% abstraction) and abstract classes (0-100% abstraction). Interfaces declare only method signatures, while

abstract classes can have both abstract and concrete methods.

Abstract Classes Explained

An abstract class can have both abstract methods (without body) and concrete methods (with body). It serves as a blueprint, allowing subclasses to share common code and enforce method implementation.

Solving Problems with Abstract Classes

Abstract classes solve issues where both default implementations and abstract methods are needed. They allow child classes to inherit concrete methods and force them to implement abstract ones.

Creating and Using Abstract Classes in Java

Abstract classes can include fields, constructors, static methods, and both abstract and non-abstract methods. However, you cannot instantiate an abstract class directly; you must create an object of a subclass.

Real-world Example: Payment Gateway Abstraction

A payment gateway abstract class can provide common policies (concrete methods) and require subclasses (like GPay, Paytm) to implement specific methods (like checkBalance). This standardizes behavior across different payment services.

Constructors and Inheritance in Abstract Classes

Constructors in abstract classes are called when an object of a subclass is created. Values can be passed from child to parent using constructors, ensuring proper initialization in inheritance chains.

Hiding Implementation Details

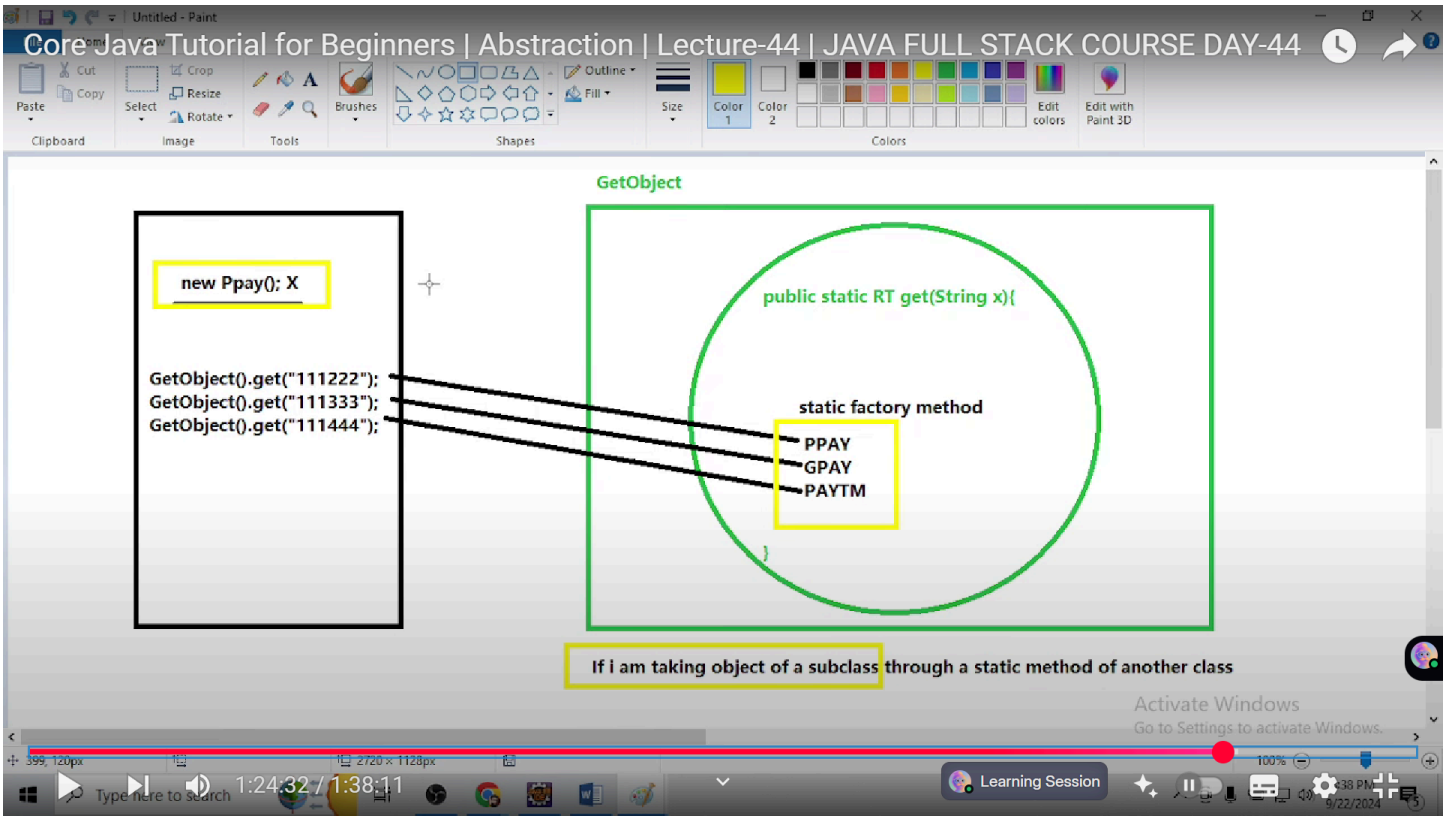
Abstraction hides the implementation details from the end user, exposing only the essential features through abstract classes or interfaces. The user interacts with the system without seeing the code logic.

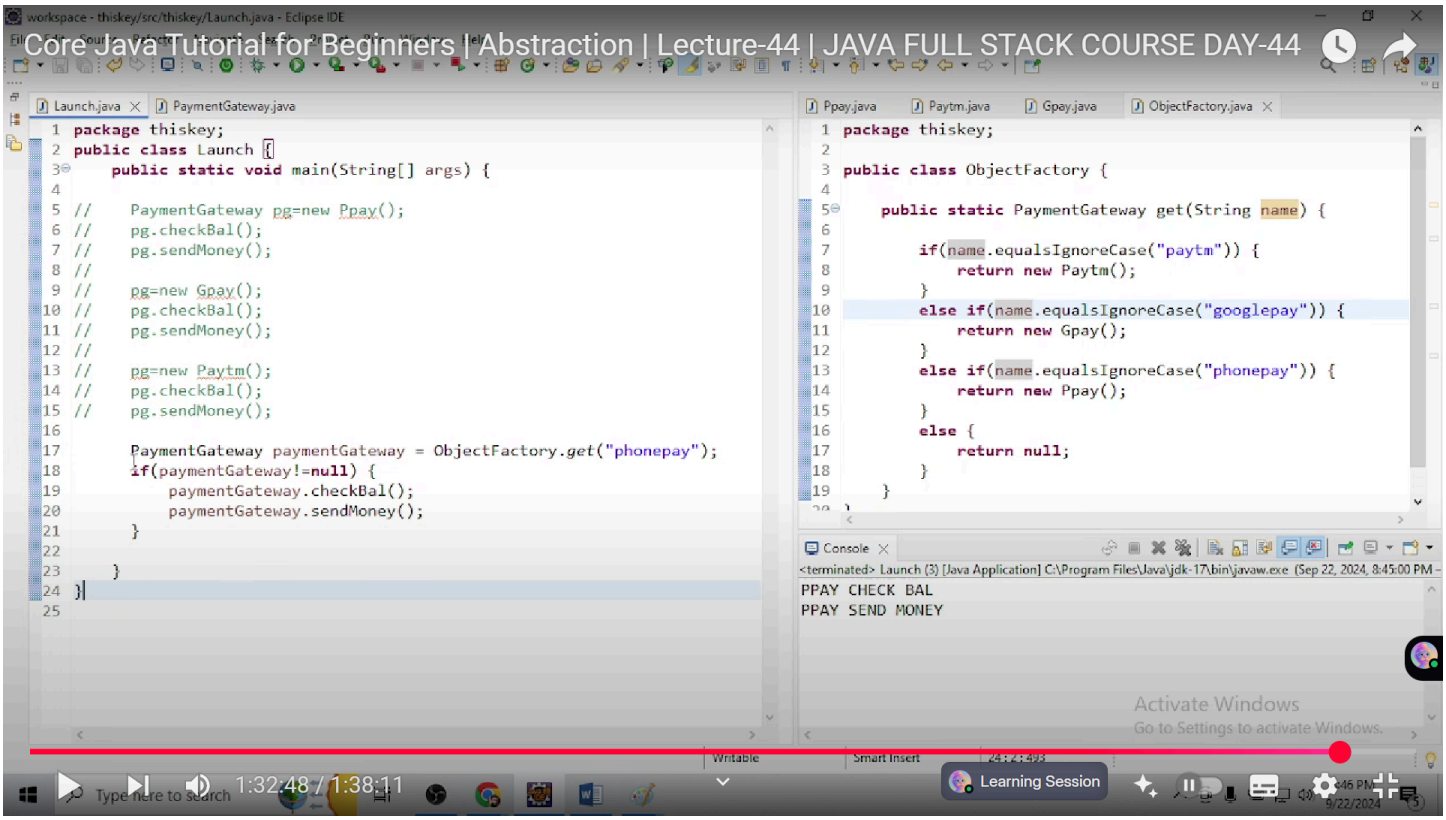
Factory Design Pattern Introduction

The Factory Design Pattern lets users create objects by calling a static method, rather than using new directly. This method decides which subclass object to return based on input, further hiding object creation details.

Static Factory Methods in Practice

Static factory methods return objects based on input parameters, allowing different subclass objects to be created without exposing their class names to the user. This pattern is used in Java's DriverManager.getConnection() method.





Abstraction Levels: Partial vs. Full

Abstract classes provide partial abstraction (some methods are implemented, some are not), while interfaces provide full abstraction (no implementation, only method signatures). The choice affects how much of the implementation is hidden from users.

Practice and Mastery

Consistent practice with abstract classes, interfaces, and design patterns like Factory helps in mastering object-oriented programming concepts and writing maintainable code.