**Use case**:Prototype Pattern

**Scenario: You need to create objects that are expensive to create or need complex initialization. The Prototype pattern allows you to create new objects by copying existing ones**

Working:

1. **Custom Clonable Interface**:
   * We’ve defined a custom interface called Clonable, which has just one method: duplicate. This method is meant for creating a copy of any object that implements this interface.
2. **Person Class**:
   * The Person class implements our Clonable interface.
   * It has two private fields: fullName and emailAddress, which hold the person's information.
   * The constructor takes in the full name and email address when creating a new Person object, initializing those fields with the provided values.
   * The duplicate method is where the magic happens. It creates a new Person object using the same fullName and emailAddress as the current object. This means we’re doing a shallow copy here. Essentially, while we’re copying the values of these fields, we’re not creating new String objects for them. Instead, both the original and the copied objects will refer to the same String instances.
3. **Main Method**:
   * In the main method, we start by creating a new Person object called initialUser, filled with some example data.
   * We then call the duplicate method on initialUser to create a copy of it, which we store in a variable named copiedUser.
   * Since duplicate does a shallow copy, both initialUser and copiedUser will point to the same String objects for fullName and emailAddress.
   * Finally, we print out the details of both the original and copied users, thanks to the toString method in the Person class.

### Points to Consider:

* This implementation of cloning is custom and doesn’t use Java’s built-in clone method from the Object class.
* Because we’re doing a shallow copy, if the Person class had any fields that were reference types (like objects), both initialUser and copiedUser would end up sharing those same references. This means changes to one could affect the other.
* If we wanted a true deep copy where all objects are independently copied—we’d need to enhance the duplicate method to clone any referenced objects within the Person class.