**Creator Pattern**

**Singleton Pattern**

**Singleton Design Pattern** in **Java**, commonly used in scenarios where exactly **one instance** of a class is needed globally — such as a **Logger**, **Database Connection Pool**, or **Configuration Manager**.

**Real-World Use Case: Logger**

In most applications, you want to ensure that only one instance of a Logger exists. This avoids issues like creating multiple log files or duplicated logs.

**A screenshot of a computer program

AI-generated content may be incorrect.**

**A screenshot of a computer program

AI-generated content may be incorrect.**

### Key Concepts Used:

* **Private Constructor**: To block external instantiation
* **Static Instance**: To hold the one and only instance
* **Lazy Initialization**: Created only when needed
* **Thread-safe**: Ensures singleton even in multithreaded environments

**Mind Twisting in Singleton**

**A screenshot of a computer program

AI-generated content may be incorrect.**

**A screenshot of a computer program

AI-generated content may be incorrect.**

**A screenshot of a computer

AI-generated content may be incorrect.**

**A screenshot of a computer program

AI-generated content may be incorrect.**

**A screenshot of a black screen

AI-generated content may be incorrect.**

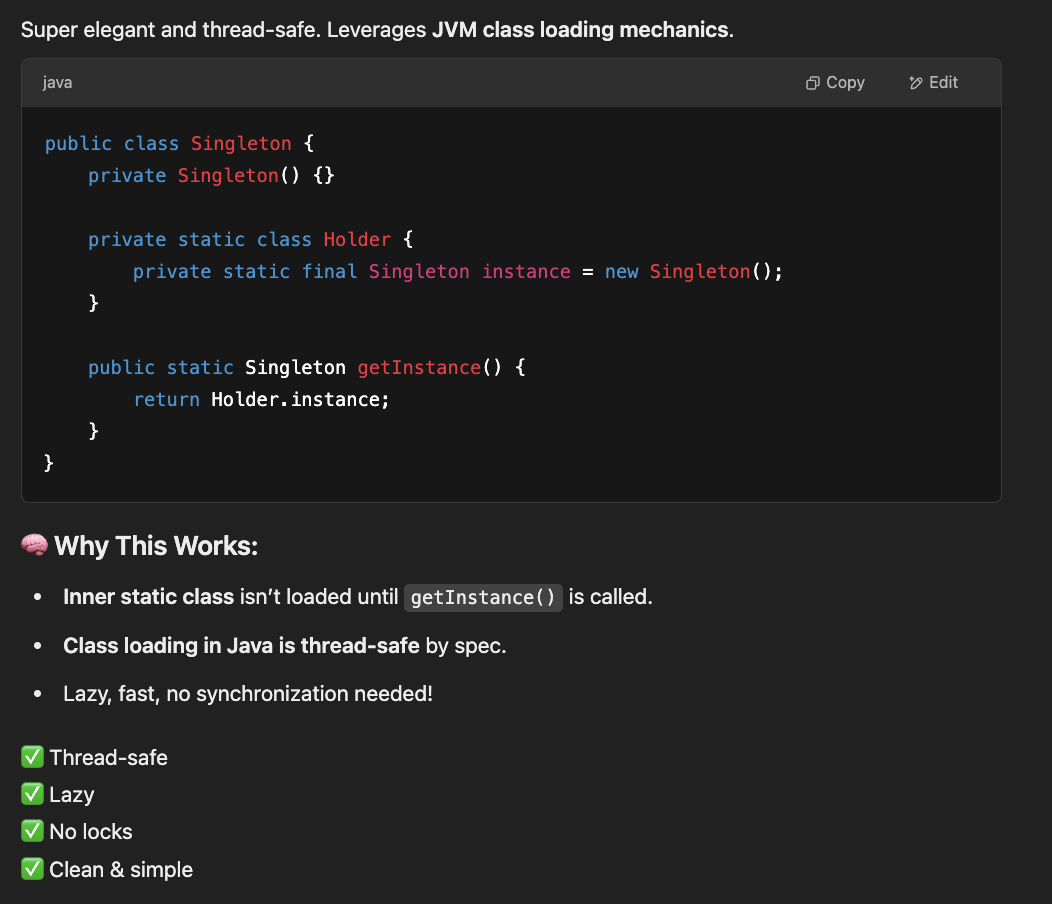
**A screenshot of a computer

AI-generated content may be incorrect.**

**A screenshot of a black box

AI-generated content may be incorrect.**

**Bill Pugh Singleton**

****

**Factory Pattern**

A black and white screen with white text

AI-generated content may be incorrect.

**A screenshot of a computer program

AI-generated content may be incorrect.**

**A screenshot of a computer program

AI-generated content may be incorrect.**

**A screenshot of a black screen

AI-generated content may be incorrect.**

**Mind Blowing Concepts Factory Pattern**

**A screenshot of a computer program

AI-generated content may be incorrect.**

**A screenshot of a computer

AI-generated content may be incorrect.**

**A screenshot of a black screen

AI-generated content may be incorrect.**

**A screenshot of a computer

AI-generated content may be incorrect.**

**A screenshot of a computer program

AI-generated content may be incorrect.**

**A screenshot of a black screen

AI-generated content may be incorrect.**

****

**A screenshot of a computer program

AI-generated content may be incorrect.**

**Abstract Factory Pattern**

**A black screen with white text

AI-generated content may be incorrect.A screenshot of a computer

AI-generated content may be incorrect.A black and white text on a black background

AI-generated content may be incorrect.**

**A screenshot of a computer program

AI-generated content may be incorrect.**

**A screenshot of a computer program

AI-generated content may be incorrect.**

**A screenshot of a computer program

AI-generated content may be incorrect.**

**A screenshot of a computer program

AI-generated content may be incorrect.**

**A screenshot of a computer

AI-generated content may be incorrect.**

**Another Abstract Factory Design Pattern Example**

|  |
| --- |
| interface SMS {  public void sendSmsNotification();  }  interface Email {  public void sendEmailNotification();  }  class GlobalSMSNotification implements SMS {  public void sendSmsNotification(){  System.out.println("Welcome(SMS)");  }  }  class GlobalEmailNotification implements Email {  public void sendEmailNotification(){  System.out.println("Welcome(Email)");  }  }  class IndiaSMSNotification implements SMS {  public void sendSmsNotification(){  System.out.println("Namaste(SMS)");  }  }  class IndiaEmailNotification implements Email {  public void sendEmailNotification(){  System.out.println("Namaste(Email)");  }  }  class JapanEmailNotification implements Email {  public void sendEmailNotification(){  System.out.println("savadeka(Email)");  }  }  class NullSMSNotification implements SMS {  public void sendSmsNotification(){  System.out.println("No valid support");  }  }  class NullEmailNotification implements Email {  public void sendEmailNotification(){  System.out.println("No valid support");  }  }  interface NotificationFactory {  public SMS prepareSMSNotification();  public Email prepareEmailNotification();  }    class GlobalFactory implements NotificationFactory {  private GlobalSMSNotification globalSMSNotification;  private GlobalEmailNotification globalEmailNotification;  GlobalFactory(){  globalSMSNotification = new GlobalSMSNotification();  globalEmailNotification = new GlobalEmailNotification();  }  public SMS prepareSMSNotification() {  return globalSMSNotification;  }  public Email prepareEmailNotification() {  return globalEmailNotification;  }  }    class IndianFactory implements NotificationFactory {  private IndiaSMSNotification indianSMSNotification;  private IndiaEmailNotification indianEmailNotification;  IndianFactory(){  indianSMSNotification = new IndiaSMSNotification();  indianEmailNotification = new IndiaEmailNotification();  }  public SMS prepareSMSNotification() {  return indianSMSNotification;  }  public Email prepareEmailNotification() {  return indianEmailNotification;  }  }  class JapanFactory implements NotificationFactory {  private JapanEmailNotification japanEmailNotification;  JapanFactory(){  japanEmailNotification = new JapanEmailNotification();  }  public SMS prepareSMSNotification() {  return new NullSMSNotification();  }  public Email prepareEmailNotification() {  return japanEmailNotification;  }  }    class Notification {  NotificationFactory factory;  Notification(NotificationFactory factory) {  this.factory = factory;  }  public void sendNotification(){  SMS sms = this.factory.prepareSMSNotification();  Email email = this.factory.prepareEmailNotification();  sms.sendSmsNotification();  email.sendEmailNotification();  }  }    public class CPNS {  public static void main(String args[]) {  NotificationFactory indianFactory = new IndianFactory();  NotificationFactory globalFactory = new GlobalFactory();  NotificationFactory japanFactory = new JapanFactory();  Notification notification = new Notification(indianFactory);  notification.sendNotification();  notification = new Notification(globalFactory);  notification.sendNotification();  notification = new Notification(japanFactory);  notification.sendNotification();  }  } |