DePaul University

Design Patterns:

Singleton

Object-oriented Software Development
SE 350– Spring 2021

Vahid Alizadeh

Week 7.2
May 13, 2021

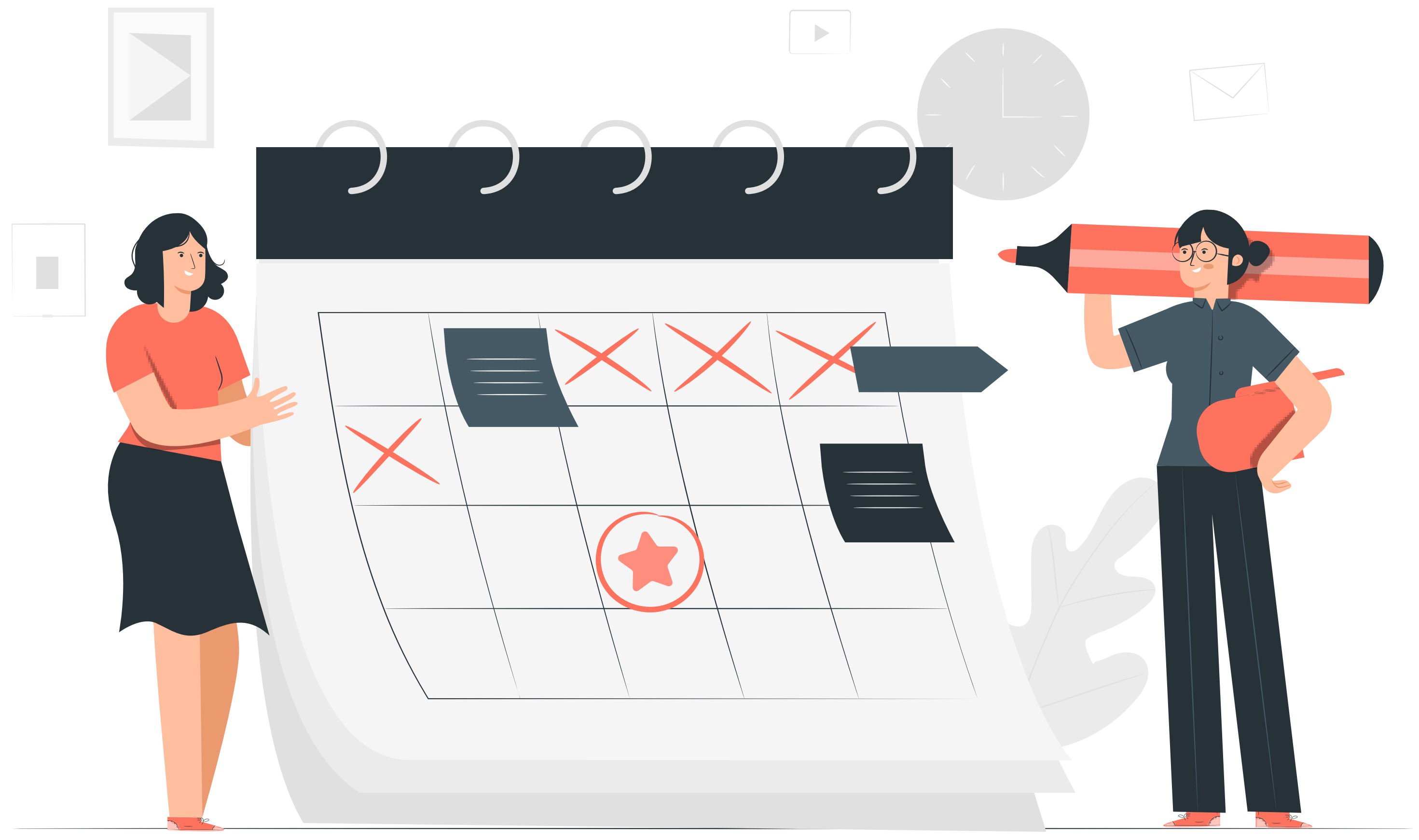
Announcements

Future Schedule

Assignment 2 is graded

Midterm is graded

- ~~**Assignment 1**~~
- ~~**Assignment 2**~~
- ~~**Mid Term Exam**~~
- **Assignment 3:**
  - Release: Week 7 (TODAY)
  - Due: Week 8
- **Assignment 4:**
  - Release: Week 8
  - Due: Week 9
- **Bonus Research Project:**
  - Presentation Due: Week 10
  - Report Due: Week 11
- **Final Exam:**
  - Week 11



Assignment 3

Due: May 20, 2021

SE 350: OO Software Development

Assignment 3: Design Principles and Design Patterns

Instructor: Vahid Alizadeh

Email: v.alizadeh@depaul.edu

Quarter: Spring 2021

DePaul University College of Computing and Digital Media

Last update: May 11, 2021

Bonus Credits: Research Paper & Presentation

- **Research Presentation**
  - **Due** Week 10
  - Max 10 slides - ~7 min talk
  - **Template**: your choice!
- **Research Report**
  - **Due** Week 11
  - Writing requirement 3-4 pages
  - At least 2 external references
    - (Conference paper, articles, journals, books)
  - **Template**: ACM Proceedings (Link)
    - LaTex or Word Template (Also uploaded on D2L)
    - Overleaf Latex template (Link)
- **Research on**
  - Object-oriented programming related concepts/principles
  - Design Patterns topics.
- **Select and Announce Your Topic On:**
  - MS Teams: Research Project Channel

Confirm your project and topic by:

May 16, 2021

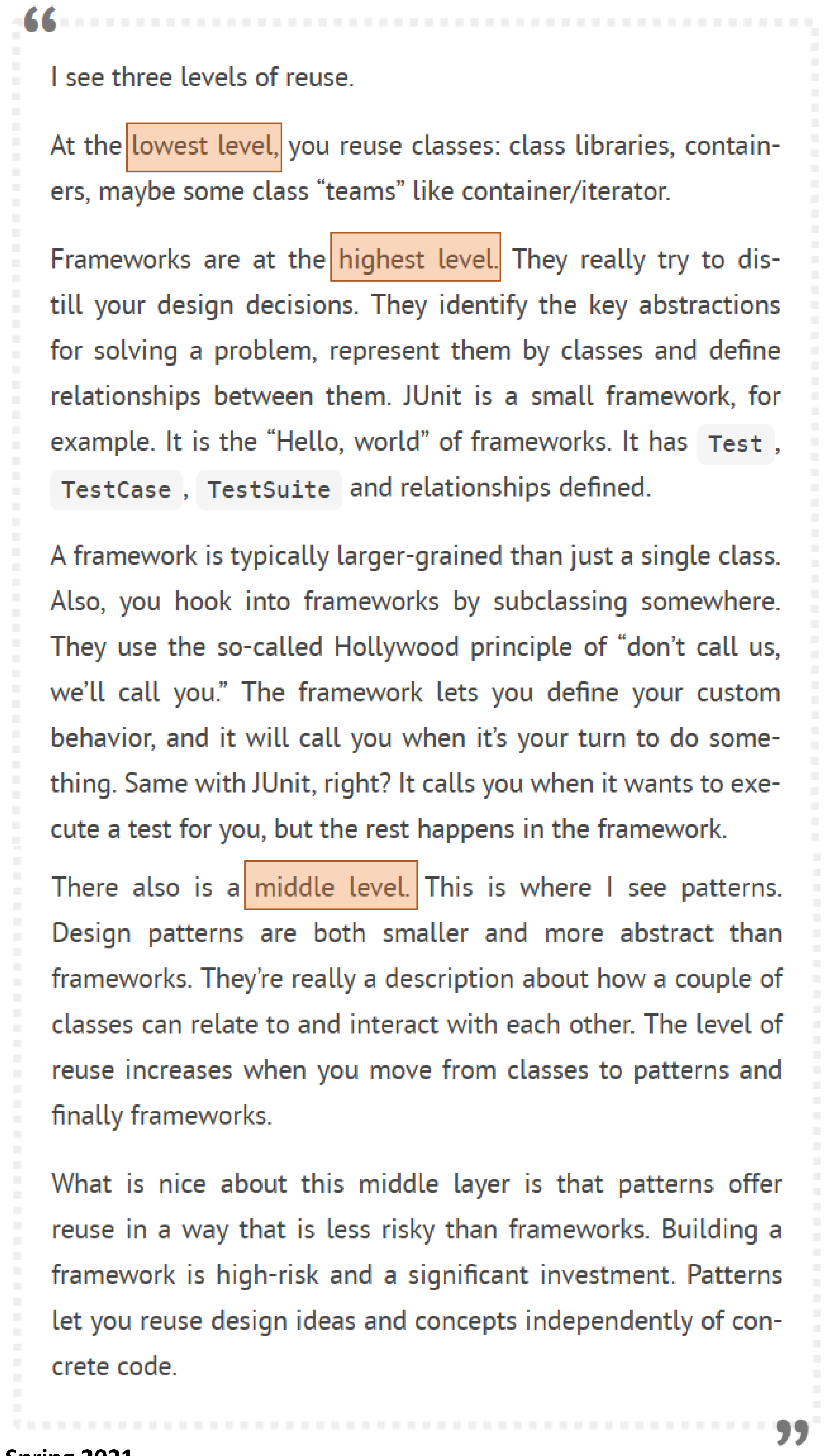
Software Design Patterns

GoF Patterns

Features of Good Design: “Code Reuse”

- Cost and time are critical in software development.
- Code reuse can reduce development costs.
- Using design patterns is one way to increase flexibility of software components and make them easier to reuse.

- [Erich Gamma on Flexibility and Reuse]()



“

I see three levels of reuse.

At the lowest level, you reuse classes: class libraries, containers, maybe some class “teams” like container/iterator.

Frameworks are at the highest level. They really try to distill your design decisions. They identify the key abstractions for solving a problem, represent them by classes and define relationships between them. JUnit is a small framework, for example. It is the “Hello, world” of frameworks. It has `Test`, `TestCase`, `TestSuite` and relationships defined.

A framework is typically larger-grained than just a single class. Also, you hook into frameworks by subclassing somewhere. They use the so-called Hollywood principle of “don’t call us, we’ll call you.” The framework lets you define your custom behavior, and it will call you when it’s your turn to do something. Same with JUnit, right? It calls you when it wants to execute a test for you, but the rest happens in the framework.

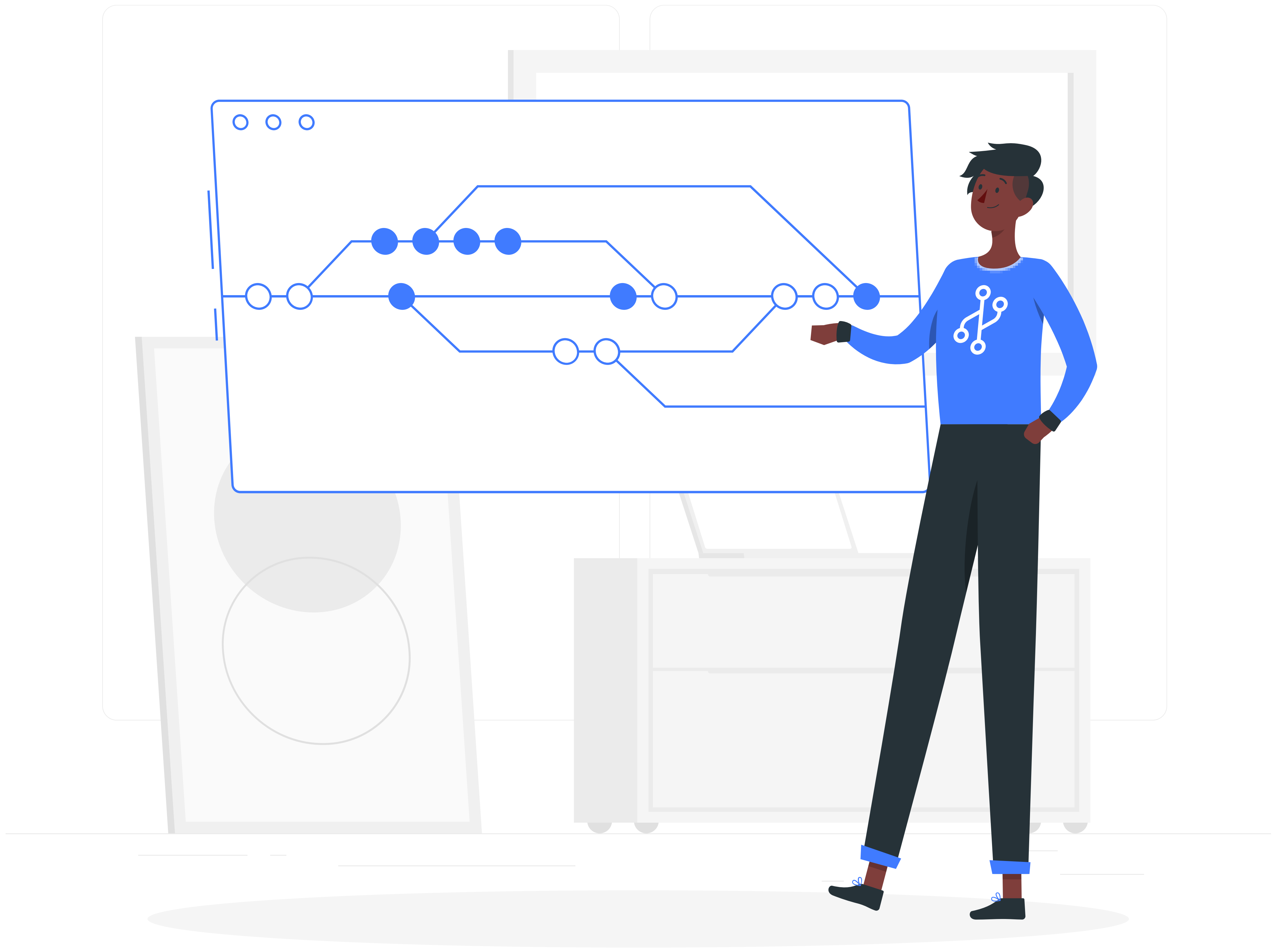
There also is a middle level. This is where I see patterns. Design patterns are both smaller and more abstract than frameworks. They’re really a description about how a couple of classes can relate to and interact with each other. The level of reuse increases when you move from classes to patterns and finally frameworks.

What is nice about this middle layer is that patterns offer reuse in a way that is less risky than frameworks. Building a framework is high-risk and a significant investment. Patterns let you reuse design ideas and concepts independently of concrete code.

”

Features of Good Design: “Extensibility”

- **Change is the only constant thing in a programmer’s life.**
  - Ex. Video game, eCommerce website, GUI framework
- **Some reasons for change:**
  - You understand your code better by the end of the project and know your code quality is bad.
  - Changes beyond your control
    - Change in framework, security issues, unsupported tools
  - Changes in requirements by stackholders

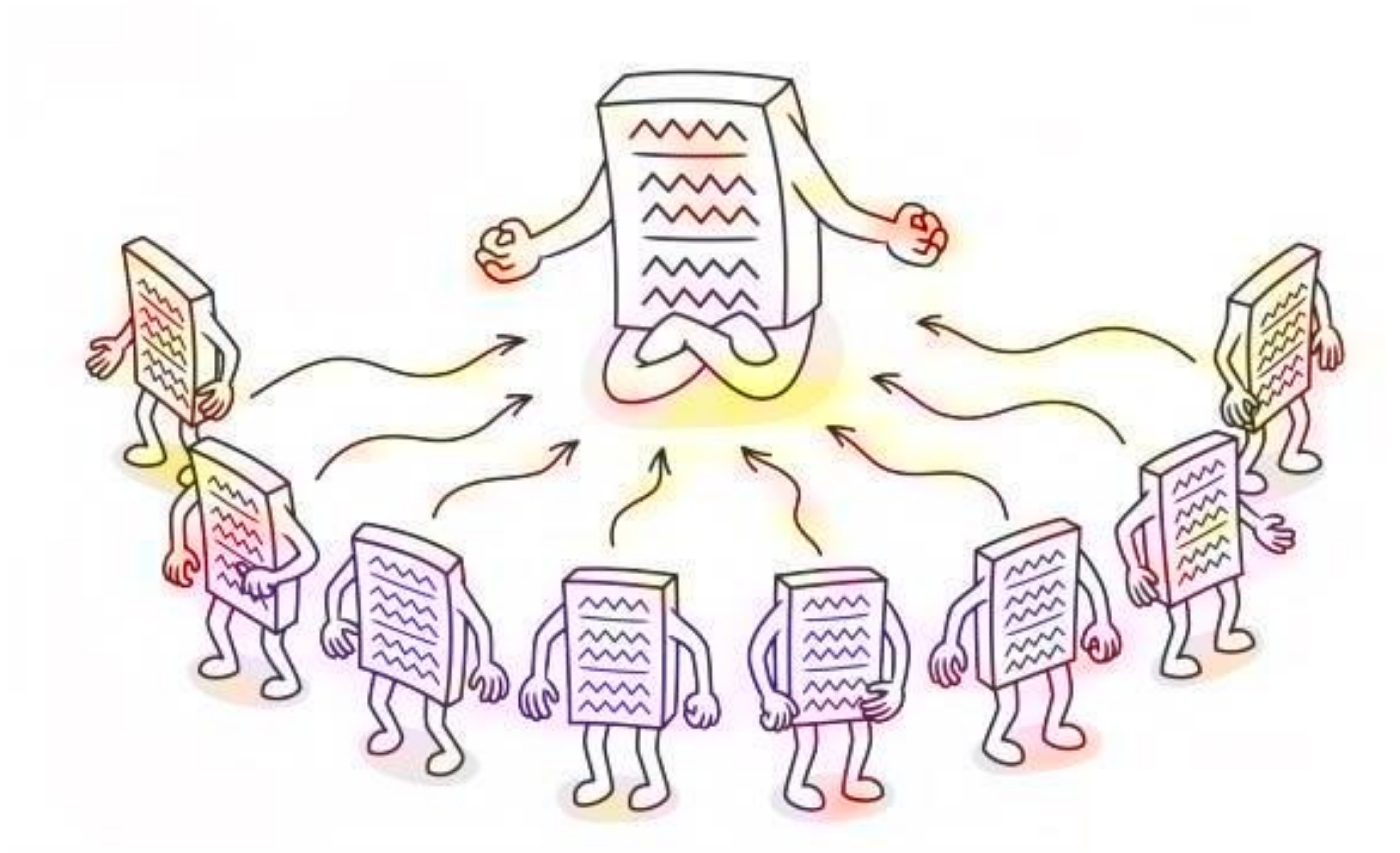


Singleton Design Pattern

CREATIONAL

Singleton Pattern Introduction

Singleton is a creational design pattern that ensure a class has only one instance, while providing a global access point to this instance.



Singleton Design Pattern

INTENT

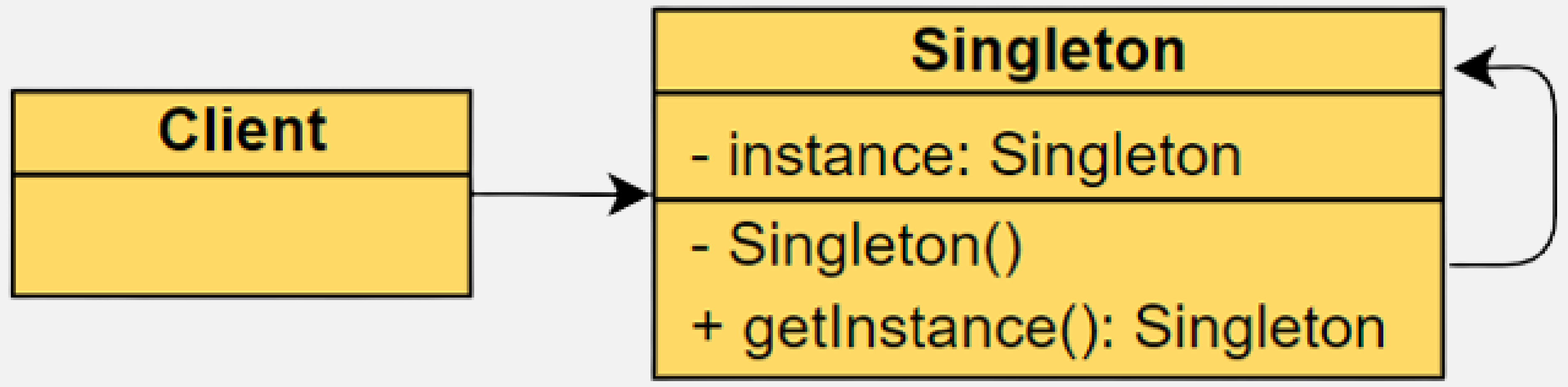
- Ensure a class has only one instance and provide a global point of access to it.
- Encapsulated "just-in-time initialization" or "initialization on first use".

PROBLEM

- Application needs one, and only one, instance of an object.
- Provide a global access point to that instance.

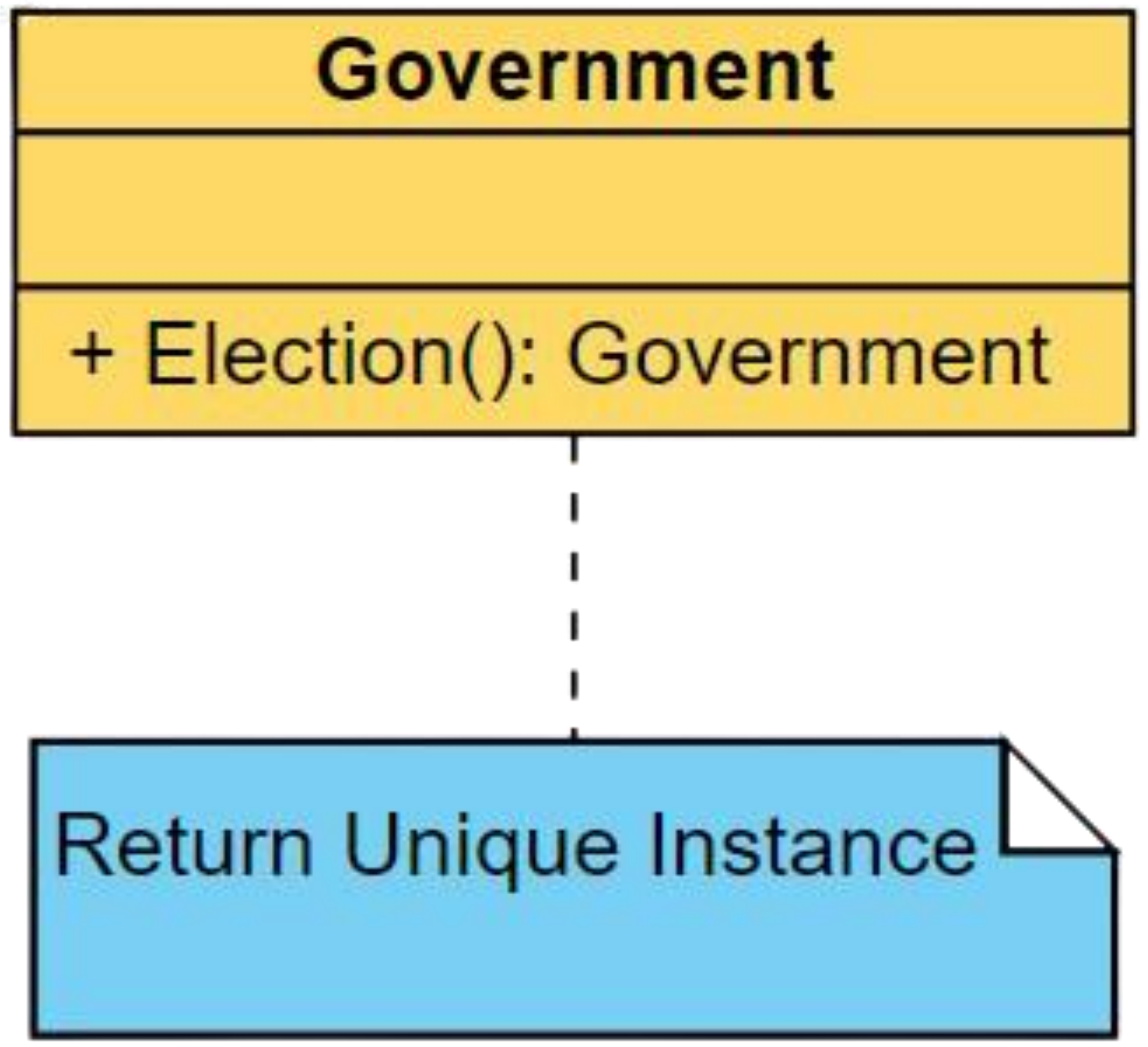
STRUCTURE

- Make the class of the single instance responsible for access and "initialization on first use".



Real-world Example

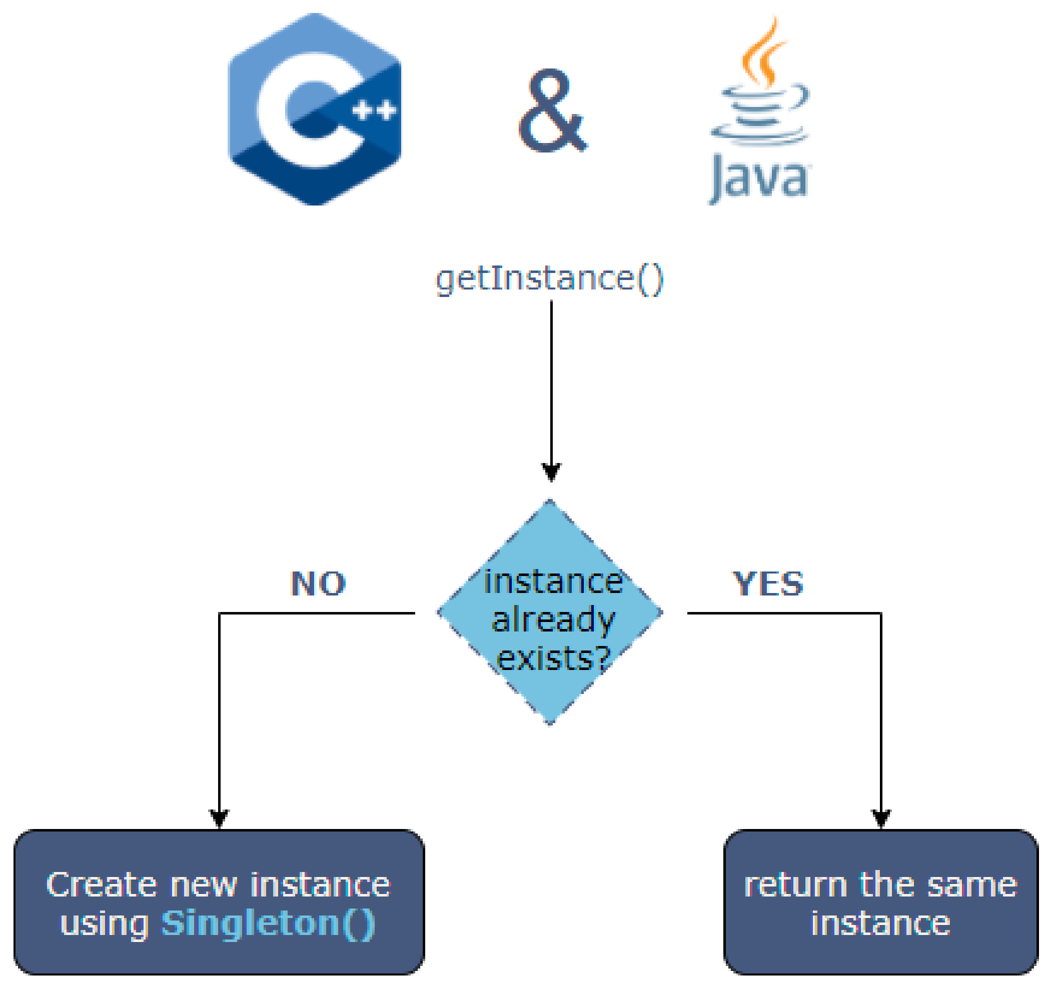
- The office of the President of the United States or the government is example of the Singleton pattern.



Implementation

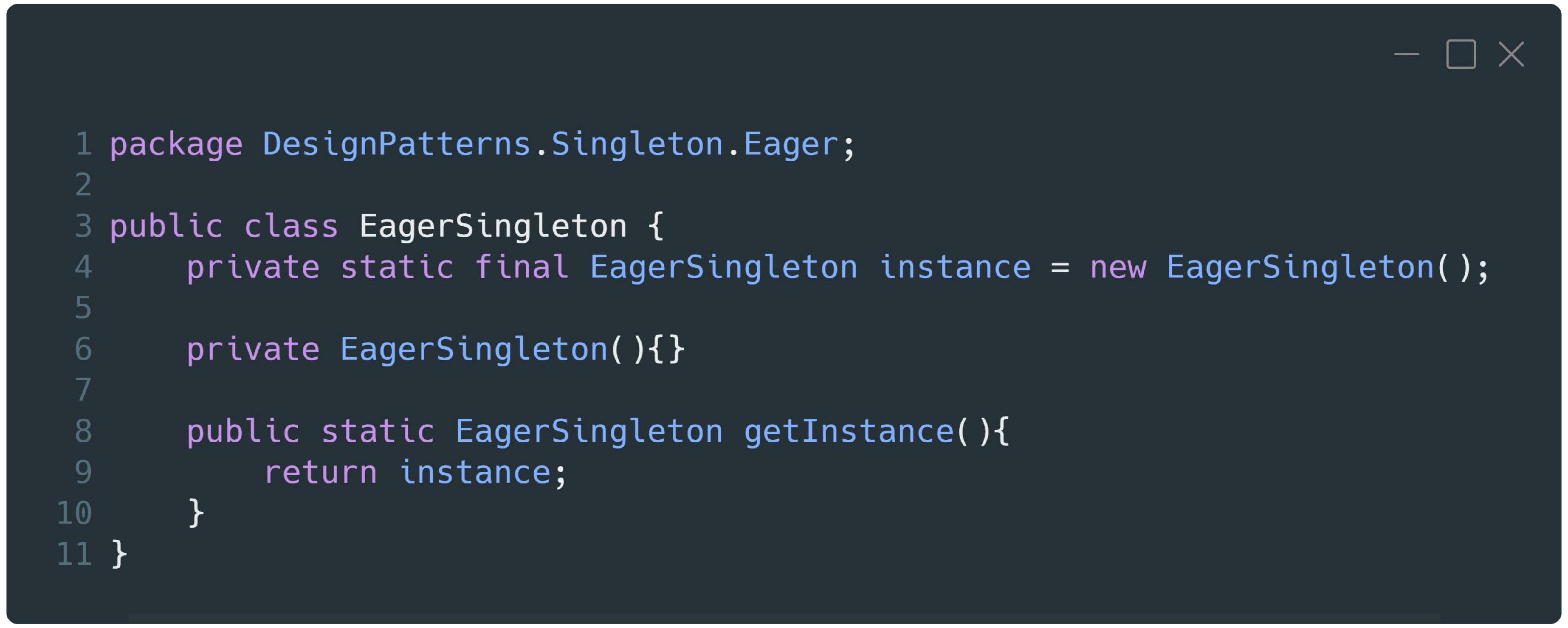
- **Common Implementation Steps:**
  - **Private constructor** to restrict instantiation of the class from other classes.
  - **Private static variable** of the same class that is the only instance of the class.
  - **Public static method** that returns the instance of the class, this is the global access point for outer world to get the instance of the singleton class.
- **Singleton categories based on instantiation:**
  - Early/Eager Loading Singleton
  - Lazy Loading Singleton

- **Singleton Implementation Approaches:**
  - Eager initialization
  - Static block initialization
  - Lazy Initialization
  - Bill Pugh Singleton Implementation

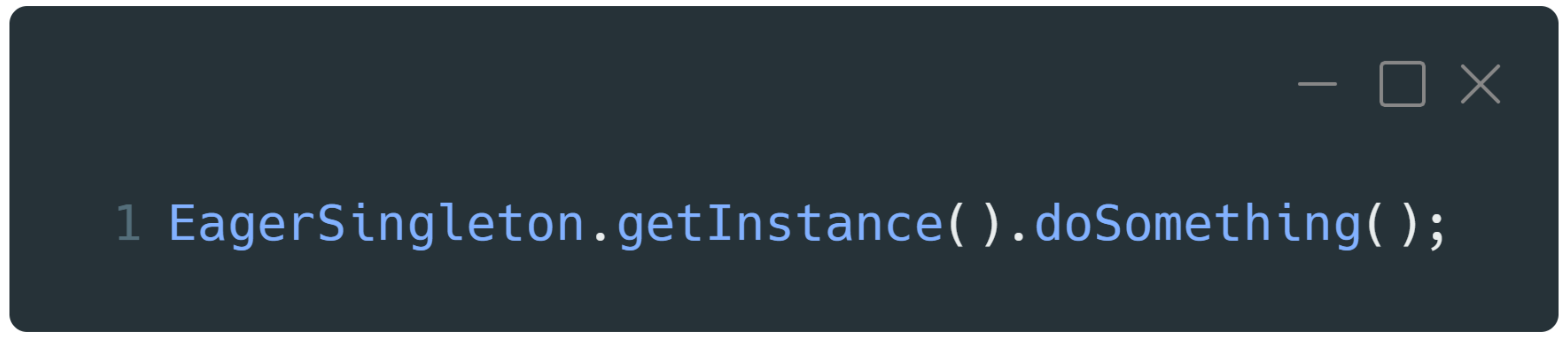


Singleton: Eager Initialization

- **What is it?**
  - Instance is created at the loading time.
- **Pros:**
  - Easiest method.
  - It is straightforward and cleaner.
- **Cons:**
  - The instance is created even the client is not using it.
  - No exception handling.
- **Only good if the Singleton class is not using many resources.**



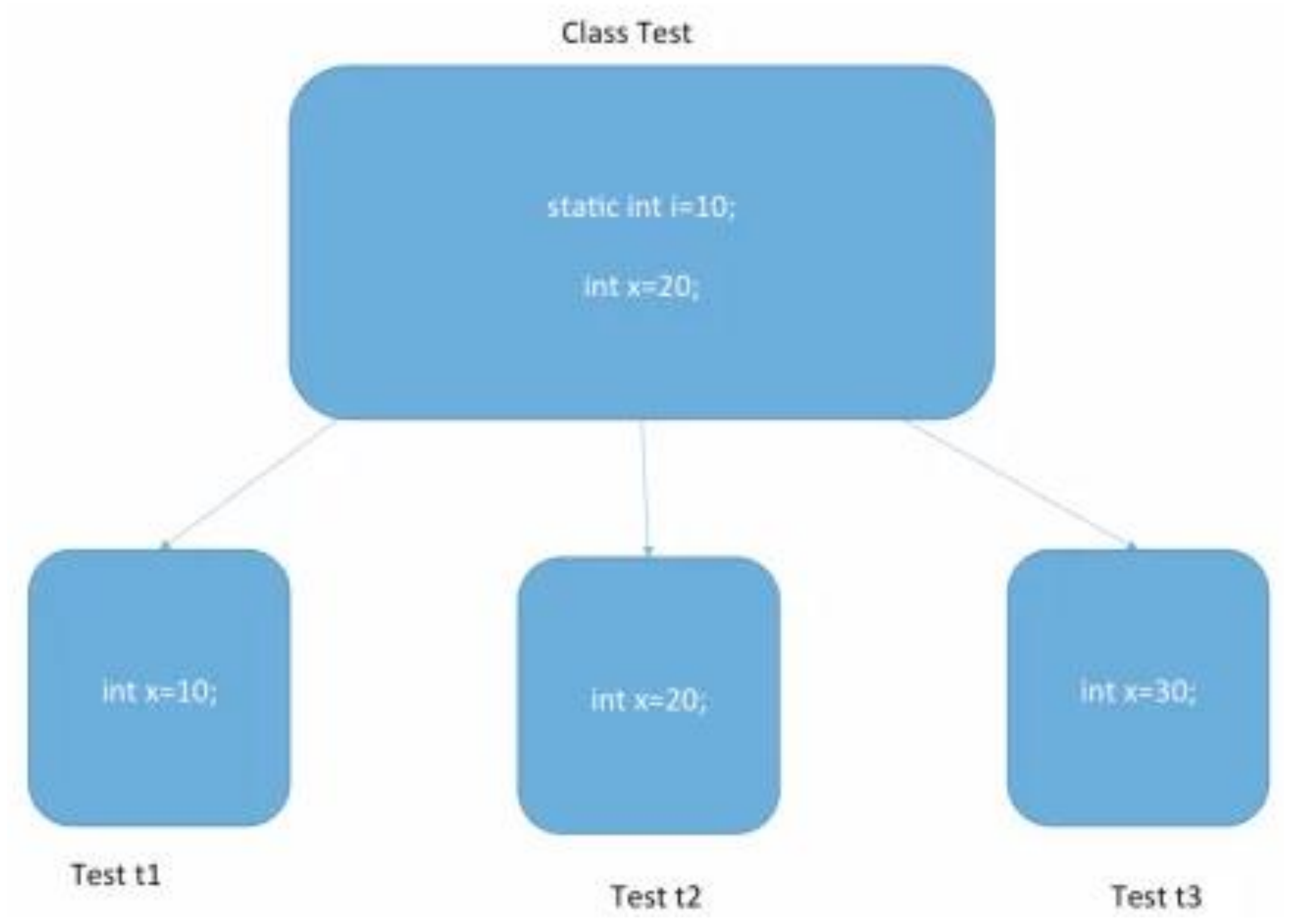
```java
package DesignPatterns.Singleton.Eager;

public class EagerSingleton {
    private static final EagerSingleton instance = new EagerSingleton();

    private EagerSingleton(){}

    public static EagerSingleton getInstance(){
        return instance;
    }
}
```

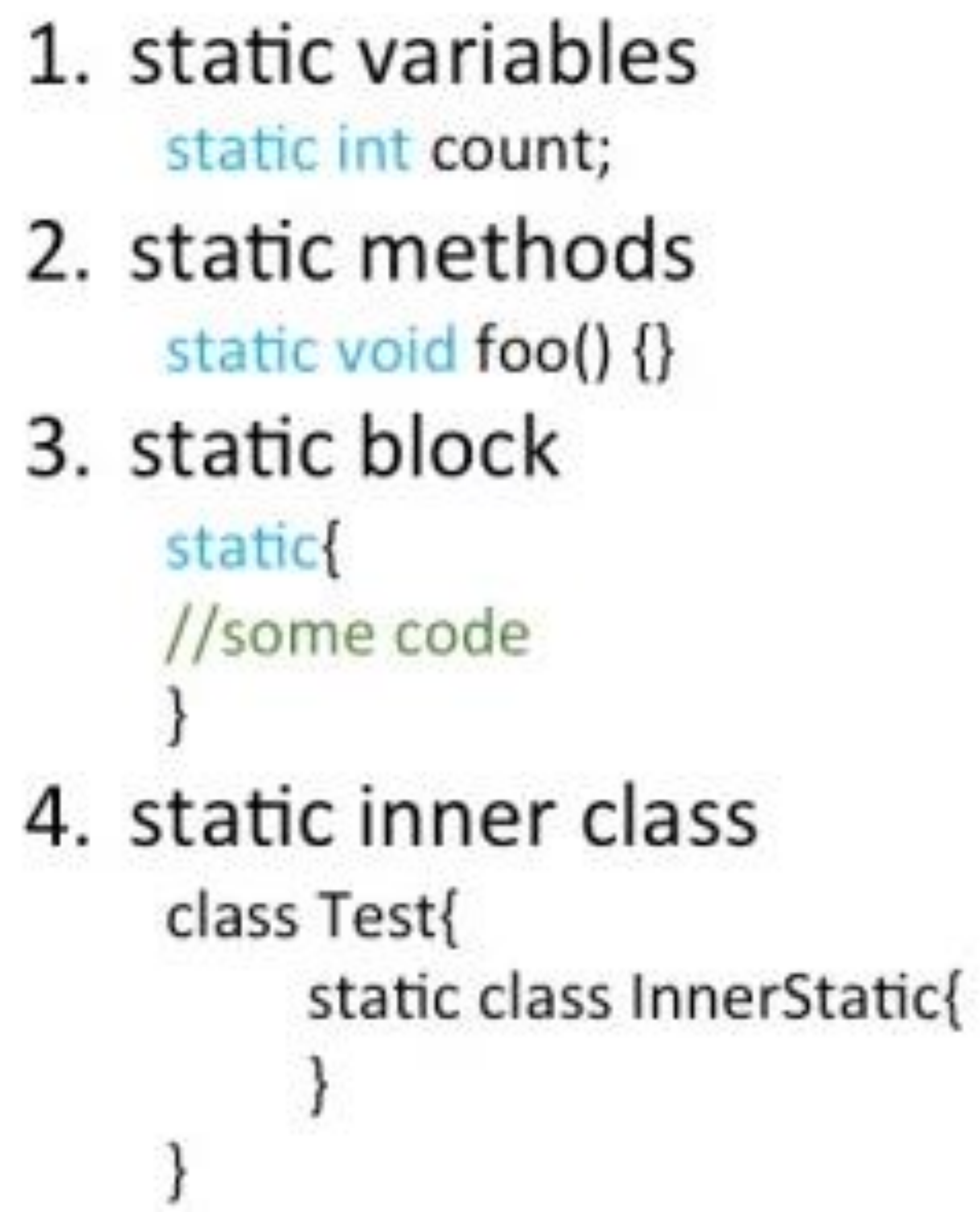


```java
EagerSingleton.getInstance().doSomething();
```

Recap: Static Keyword

- **Static variables**
  - Static variable is a class variable
  - Access via *ClassName.variableName*
- **Static methods**
  - Static method belong to class
  - Static method can access only static variables
- **Static block**
  - Group of statements that gets executed when the class is loaded into memory.
  - used to create static resources
- **Static inner class**
  - Used with nested classes

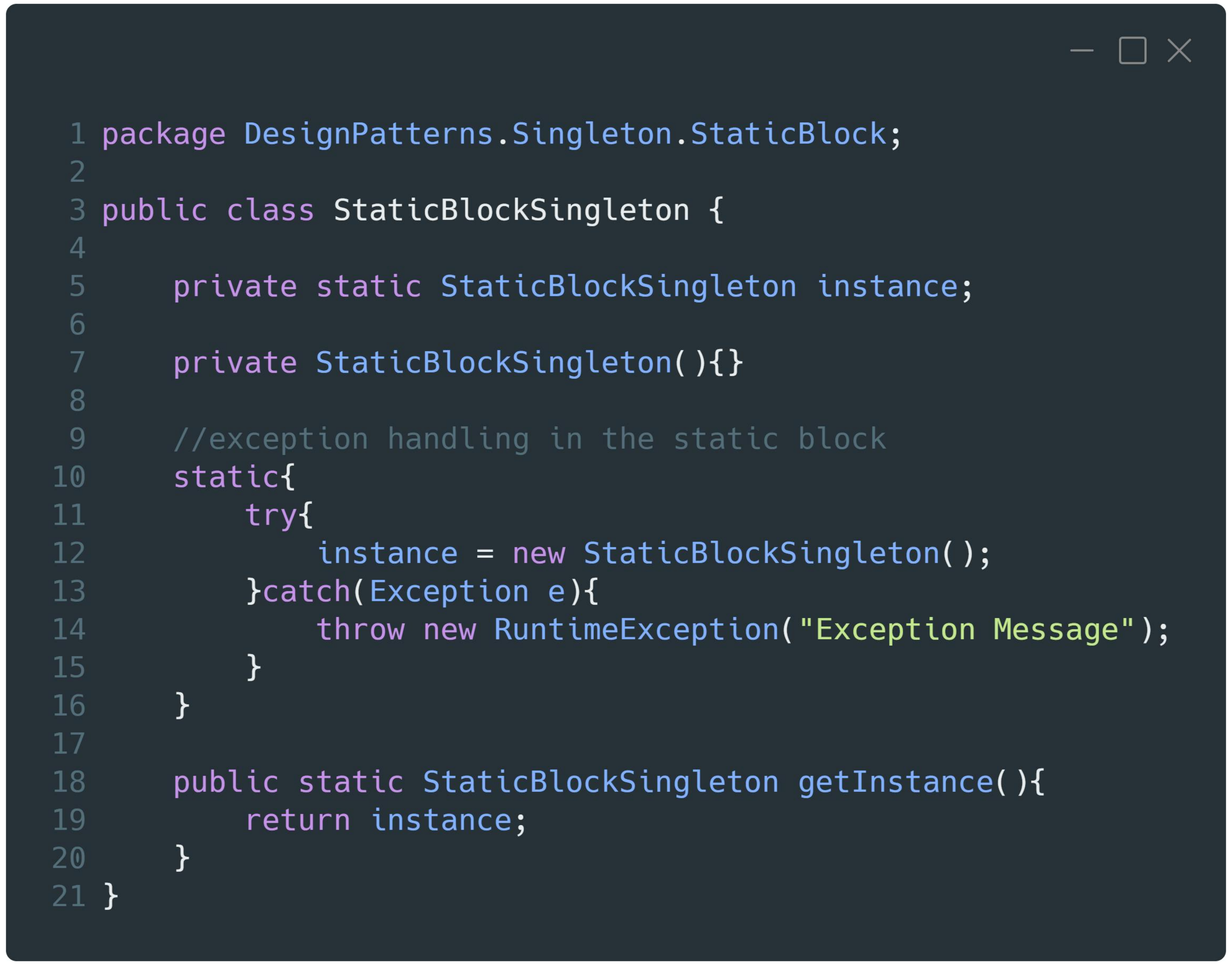




1. static variables
```
static int count;
```
2. static methods
```
static void foo() {}
```
3. static block
```
static{
//some code
}
```
4. static inner class
```
class Test{
        static class InnerStatic{
        }
}
```

Singleton: Static Block Initialization

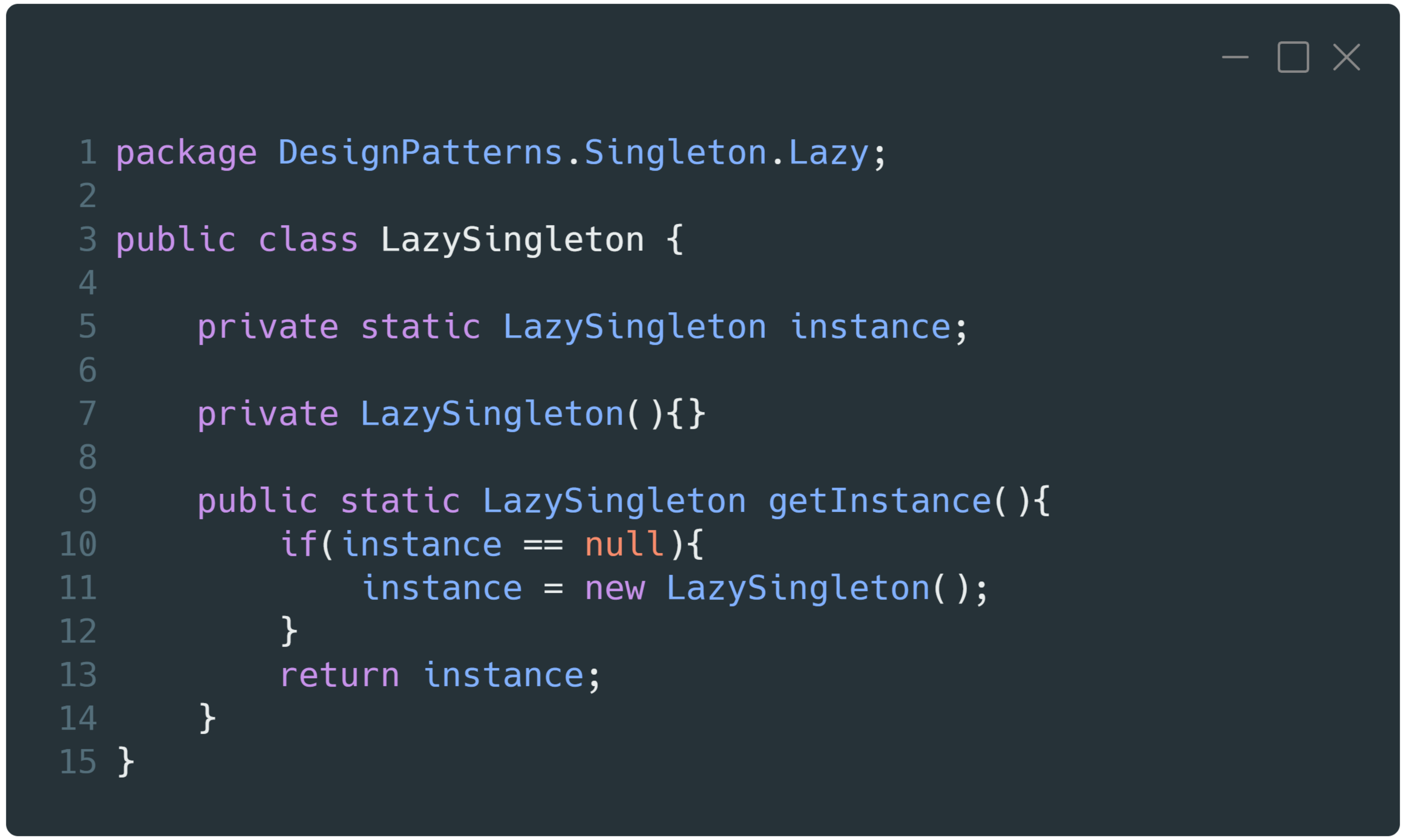
- **What is it?**
  - The concept is like Eager but with exception handling option.
- **Pros:**
  - Exception handling
- **Cons:**
  - The instance is created even the client is not using it.
  - If we have multiple instances, all of them are created even if we need some of them.
- **Only good if the Singleton class is not using many resources.**



```java
package DesignPatterns.Singleton.StaticBlock;

public class StaticBlockSingleton {

    private static StaticBlockSingleton instance;

    private StaticBlockSingleton(){}

    //exception handling in the static block
    static{
        try{
            instance = new StaticBlockSingleton();
        }catch(Exception e){
            throw new RuntimeException("Exception Message");
        }
    }

    public static StaticBlockSingleton getInstance(){
        return instance;
    }
}
```

Singleton: Lazy Initialization

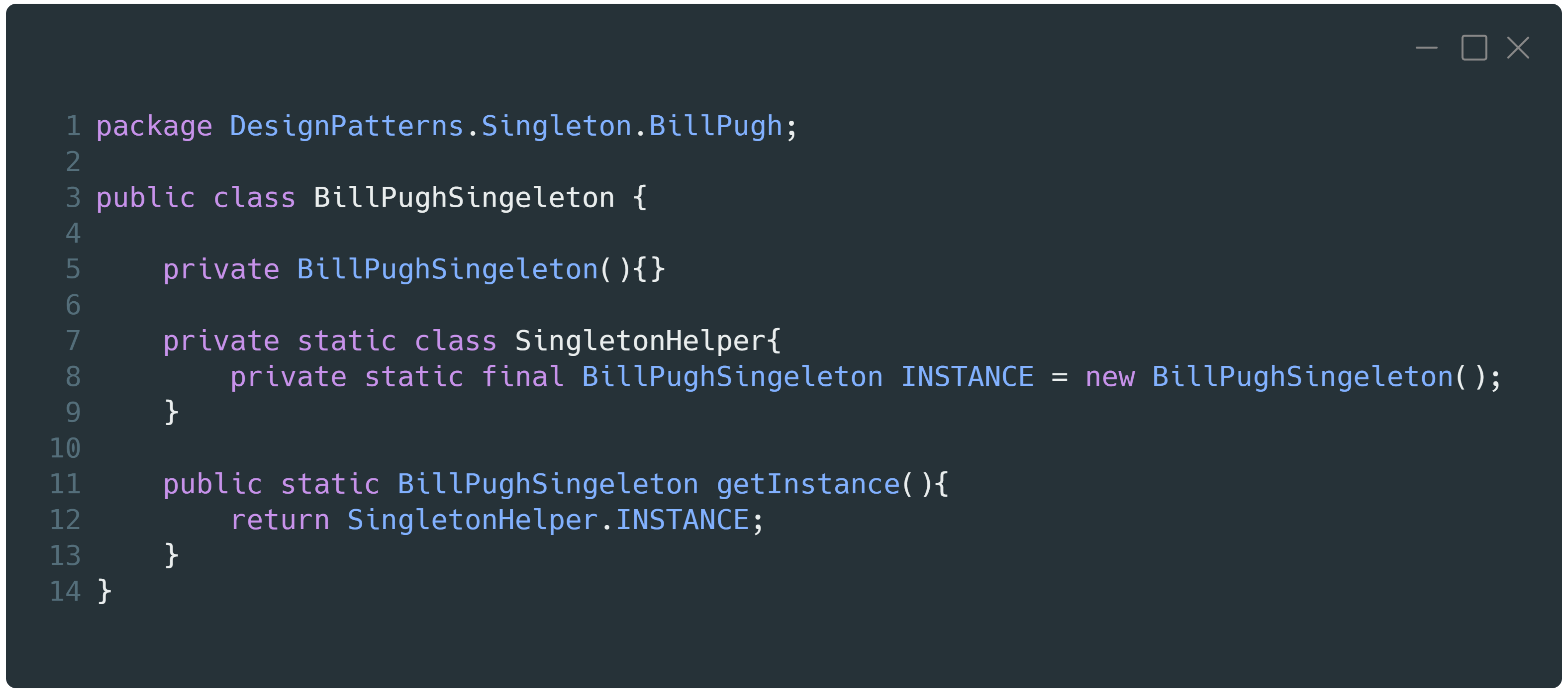
- **What is it?**
  - Creates the instance in the global access method.
- **Pros:**
  - Creates the instance when it is needed
- **Cons:**
  - Can be problematic in multi-threaded systems



```java
package DesignPatterns.Singleton.Lazy;

public class LazySingleton {

    private static LazySingleton instance;

    private LazySingleton(){}

    public static LazySingleton getInstance(){
        if(instance == null){
            instance = new LazySingleton();
        }
        return instance;
    }
}
```

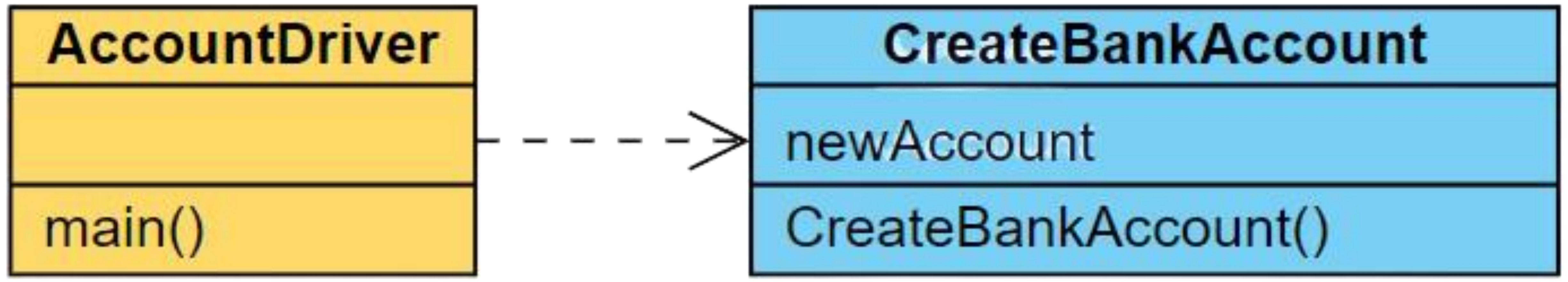
Singleton: Bill Pugh Singleton Implementation

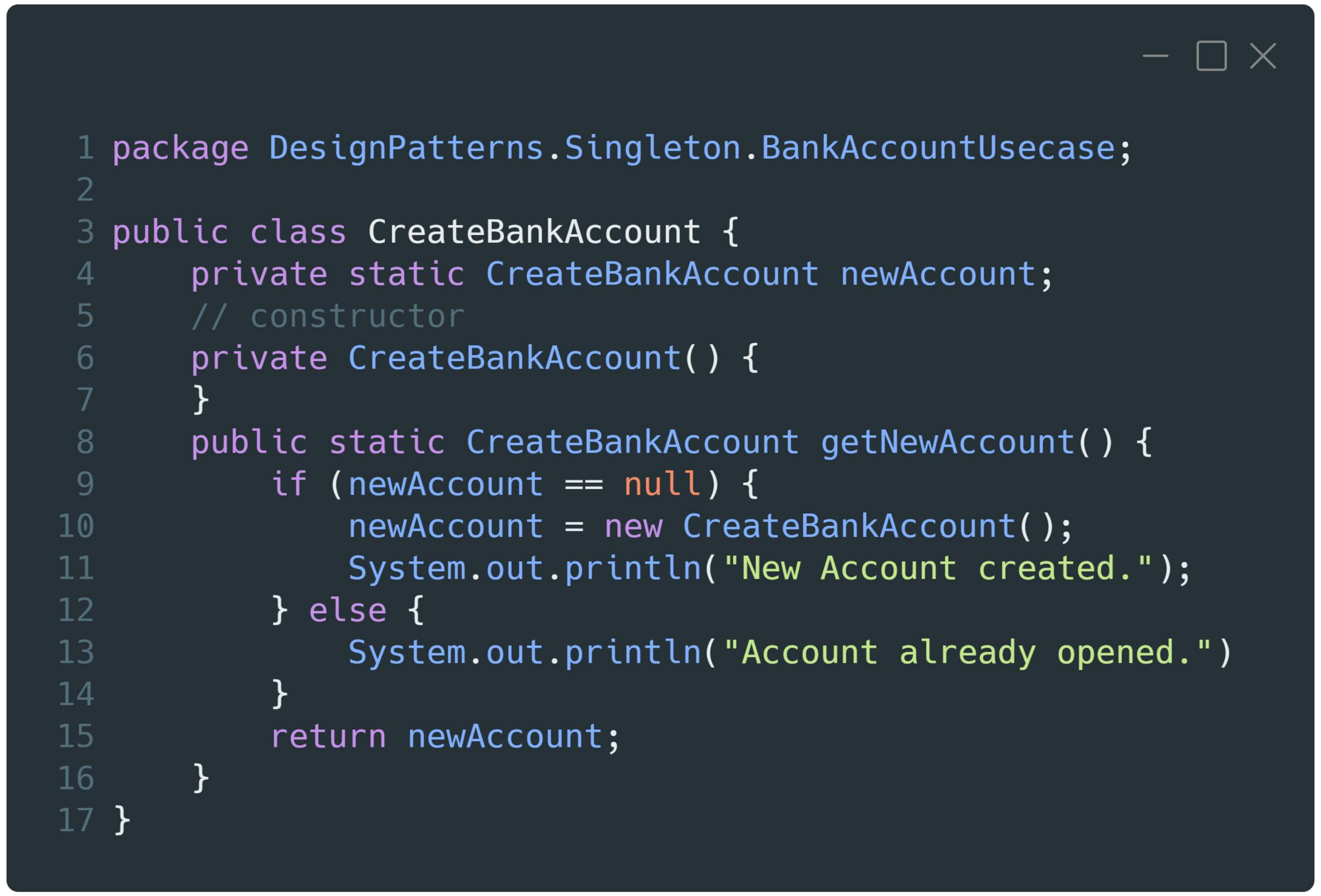
- **What is it?**
  - Using static inner helper class
- **Pros:**
  - Create instance(s) as needed
- **initialization-on-demand holder**



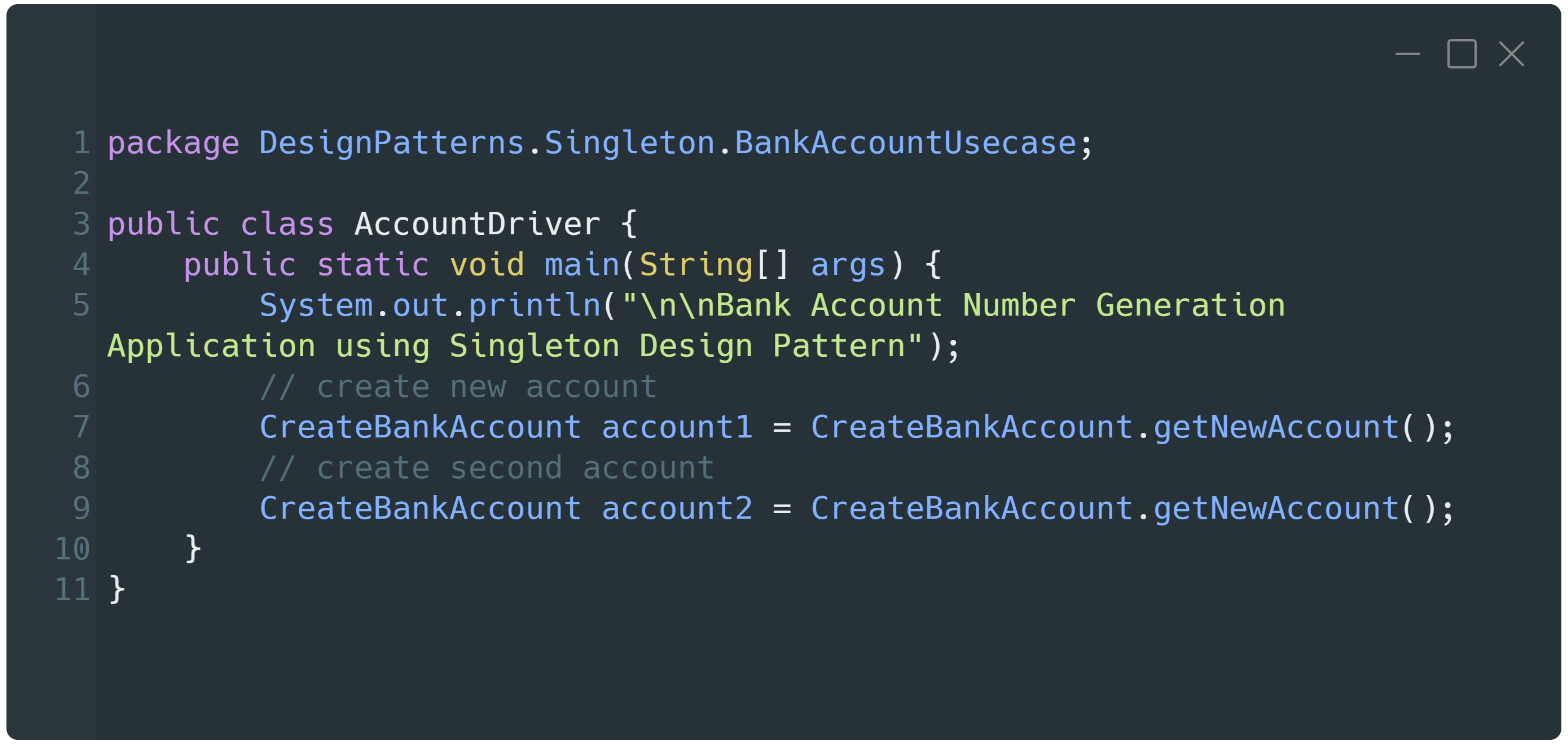
```java
package DesignPatterns.Singleton.BillPugh;

public class BillPughSingeleton {

    private BillPughSingeleton(){}

    private static class SingletonHelper{
        private static final BillPughSingeleton INSTANCE = new BillPughSingeleton();
    }

    public static BillPughSingeleton getInstance(){
        return SingletonHelper.INSTANCE;
    }
}
```

Use case Example: Bank Account





```java
package DesignPatterns.Singleton.BankAccountUsecase;

public class CreateBankAccount {
    private static CreateBankAccount newAccount;
    // constructor
    private CreateBankAccount() {
    }
    public static CreateBankAccount getNewAccount() {
        if (newAccount == null) {
            newAccount = new CreateBankAccount();
            System.out.println("New Account created.");
        } else {
            System.out.println("Account already opened.")
        }
        return newAccount;
    }
}
```



```java
package DesignPatterns.Singleton.BankAccountUsecase;

public class AccountDriver {
    public static void main(String[] args) {
        System.out.println("\n\nBank Account Number Generation
Application using Singleton Design Pattern");
        // create new account
        CreateBankAccount account1 = CreateBankAccount.getNewAccount();
        // create second account
        CreateBankAccount account2 = CreateBankAccount.getNewAccount();
    }
}
```

Find the source codes in the course GitHub repository.

Singleton Pattern Pros & Cons

Pros	Cons
✓ The class has only a single instance.	✗ Can lead to bad design.
✓ Global access point to that instance.	✗ Requires special treatment in multithreaded systems.
✓ Initialized only when is requested.	✗ Make unit testing difficult.

Any Question

????????????????

How do you feel about the course?

Powered by Poll Everywhere

Start the presentation to see live content. For screen share software, share the entire screen. Get help at **pollev.com/app**

Please Send Your Question or Feedback...

Top

New

Powered by Poll Everywhere

Start the presentation to see live content. For screen share software, share the entire screen. Get help at **pollev.com/app**