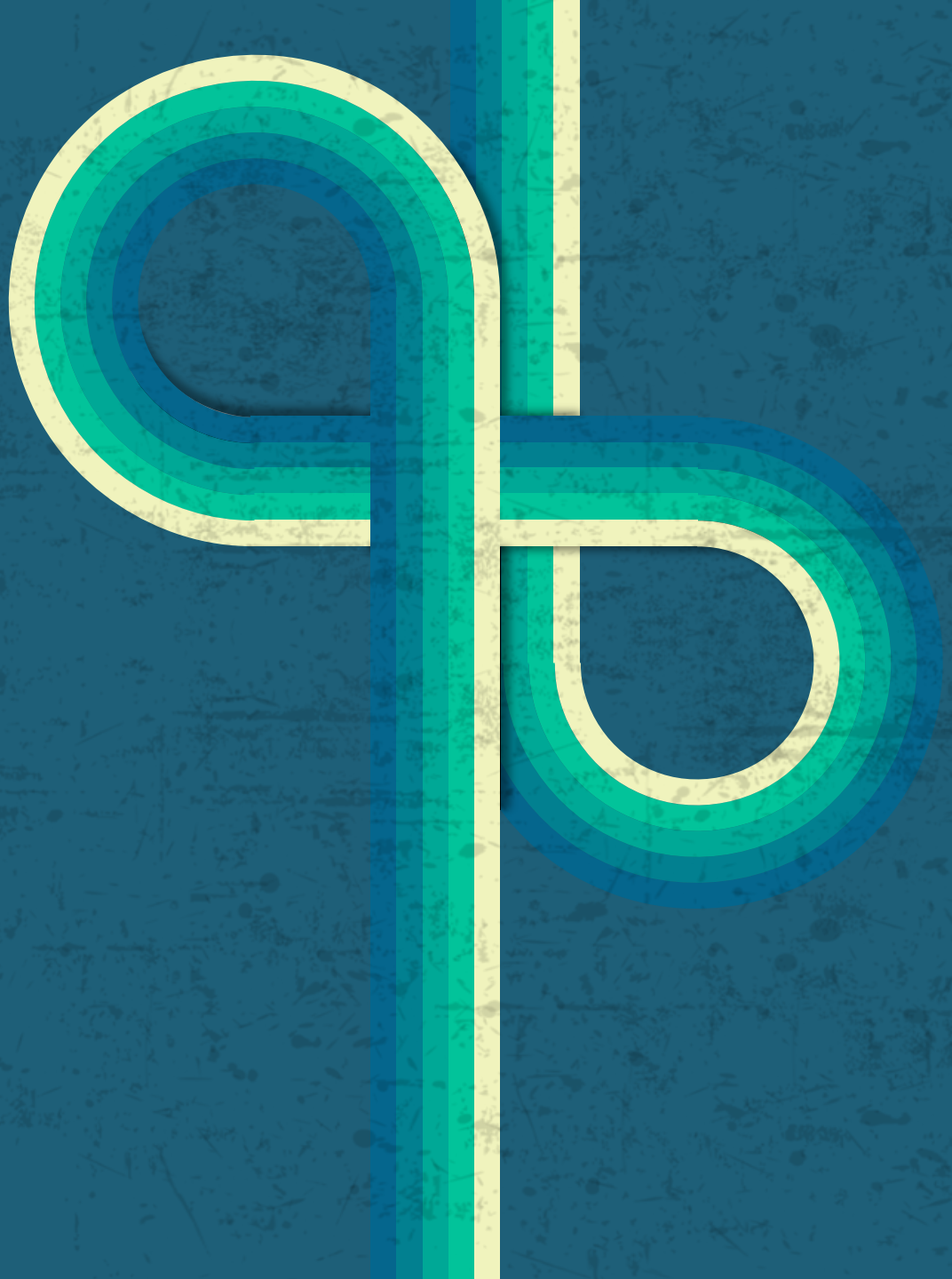


Порождающие паттерны проектирования в **Java:**

– Singleton
– Factory

Студенты группы 5030102/20202
Тишковец Сергей
Ткачев Михаил

План презентации

01 | Singleton

Для чего вообще нужен этот паттерн?

02 | Ways to implement

Способы реализации паттерна

03 | Example

Пример использования

04 | Factory

Какие проблемы решает?

05 | Benefits

Преимущества использования

06 | Example

Наглядная демонстрация паттерна

01 Singleton

- Дает гарантию, что у класса будет всего один экземпляр класса
- Предоставляет глобальную точку доступа к экземпляру данного класса

Особенности:

1. Приватный конструктор
2. Публичный статический метод, возвращающий экземпляр класса

02

Способы реализации

1. Simple Solution

```
public class Singleton {
        private static final Singleton INSTANCE = new Singleton();
        private Singleton() {
        }
        public static Singleton getInstance() {
                return INSTANCE;
        }
}
```

+ Простота и прозрачность кода
+ Потокобезопасность
+ Высокая производительность в многопоточной среде

- Не ленивая инициализация

Способы реализации

2. Lazy Initialization

```
public class Singleton {
        private static Singleton INSTANCE;
        private Singleton() {
        }
        public static Singleton getInstance() {
                if (INSTANCE == null) {
                        INSTANCE = new Singleton();
                }
                return INSTANCE;
        }
}
```

+ Ленивая инициализация

- Не потокобезопасно

Способы реализации

3. Synchronized Accessor

```
public class Singleton {
        private static Singleton INSTANCE;
        private Singleton() {
        }
        public static synchronized Singleton getInstance() {
                if (INSTANCE == null) {
                        INSTANCE = new Singleton();
                }
                return INSTANCE;
        }
}
```

+ Ленивая инициализация
+ Потокобезопасность

- Низкая производительность

Способы реализации

4. Bill Pugh Singleton

```
public class Singleton {
        private Singleton() {
        }
        private static class SingletonHelper {
                private static final Singleton INSTANCE = new Singleton();
        }
        public static Singleton getInstance() {
                return SingletonHelper.INSTANCE;
        }
}
```

+ Ленивая инициализация
+ Потокобезопасность
+ Высокая производительность в многопоточной среде

03 Пример использования шаблона

```
class Database {
  private static Database dbObject;
  private Database() {  }
  public static Database getInstance() {
    // create object if it's not already created
    if(dbObject == null) {
      dbObject = new Database();
    }
     // returns the singleton object
      return dbObject;
  }
  public void getConnection() {
     System.out.println("You are now connected to the
database.");
  }
}
```

```
public lass Main {
  public static void main(String[] args) {
    Database db1;

    // refers to the only object of Database
    db1= Database.getInstance();

    db1.getConnection();
  }
}
```

```
You are now connected to the database.
```

04 Factory

Проблема:

- необходимость гибкого создания объектов различных типов, когда заранее неизвестно, какой конкретный класс потребуется создать.

Решение:

- создание объекта не напрямую, используя оператор ***new***, а через вызов фабричного метода. Подклассы класса, который содержит фабричный метод, могут изменять создаваемые объекты конкретных создаваемых объектов.

Структура

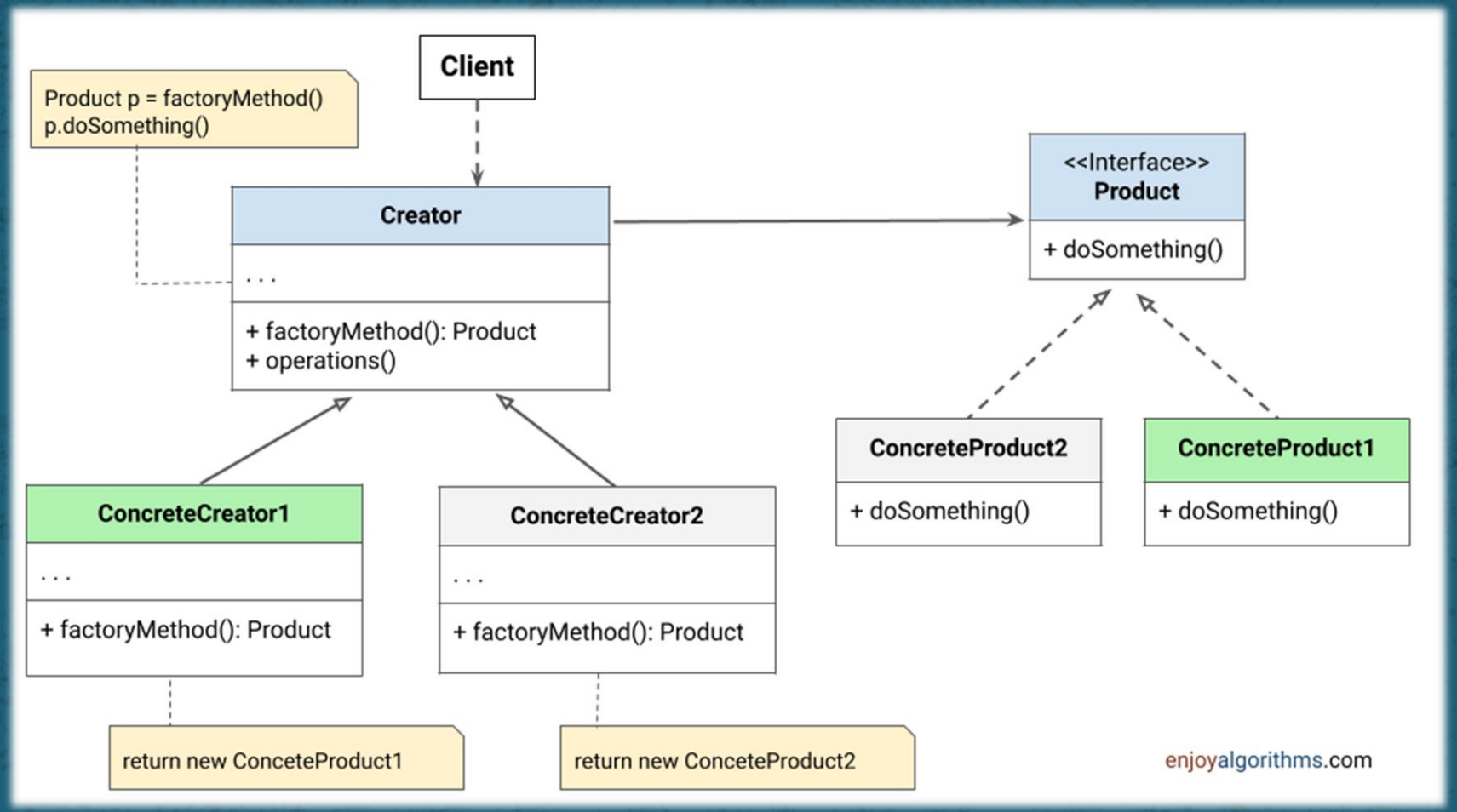
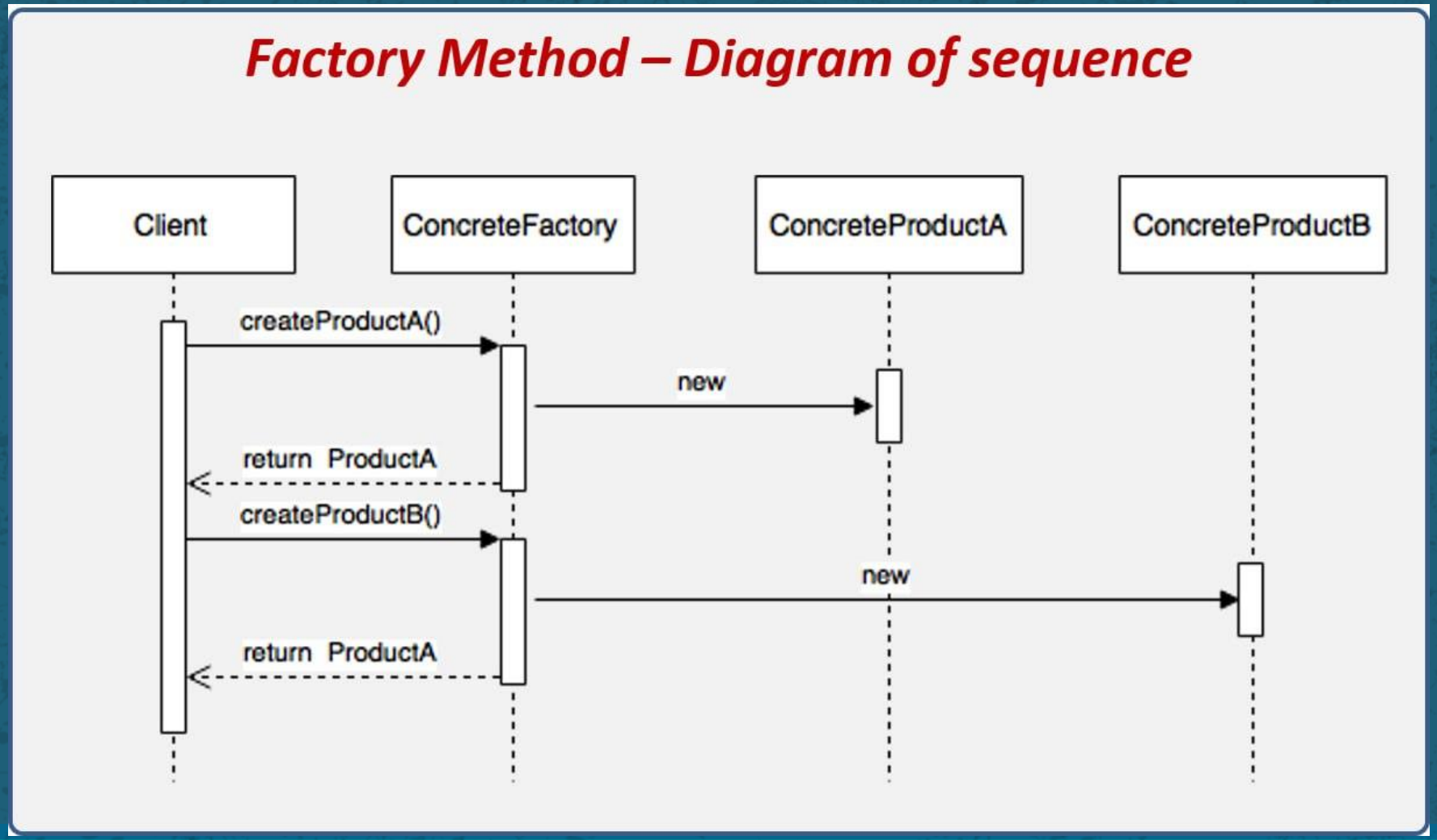

Client
Product p = factoryMethod()
p.doSomething()
Creator
. . .
+ factoryMethod(): Product
+ operations()
<<Interface>>
Product
+ doSomething()
ConcreteProduct2
+ doSomething()
ConcreteProduct1
+ doSomething()
ConcreteCreator1
. . .
+ factoryMethod(): Product
ConcreteCreator2
. . .
+ factoryMethod(): Product
return new ConceteProduct1
return new ConceteProduct2
enjoyalgorithms.com

Diagram of sequence

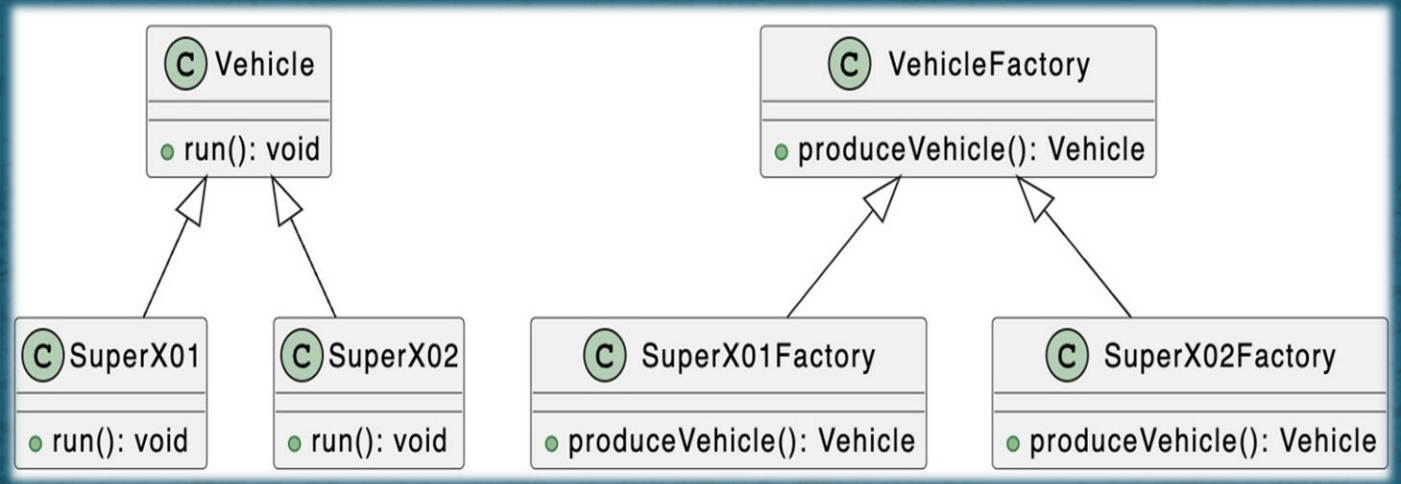


05

Преимущества использования паттерна

- Уменьшение связности: позволяет изменять классы объектов без изменения клиентского кода
- Упрощение расширения: легко добавлять новые типы объектов
- Повышенная читаемость и поддержка кода: код становится более структурированным

06 Примеры использования шаблона



- Product (Vehicle): абстрактный продукт;
- Concrete Product (SuperX01): конкретный продукт;
- Factory (VehicleFactory): абстрактная фабрика;
- ConcreteFactory(SuperX01Factory): конкретная фабрика.

Источники

- https://javarush.com/groups/posts/2365-patternih-proektirovanija-singleton
- https://www.geeksforgeeks.org/singleton-class-java/
- https://javarush.com/groups/posts/2370-pattern-proektirovanija-factory
- https://www.geeksforgeeks.org/factory-method-design-pattern-in-java/

Спасибо за внимание