**PRAKTIKKUM PDPL**

***CREATIONAL PATTERN - SINGLETON***

**Mata Kuliah : Pola Desain Perangkat Lunak**

**Semester : 6 (Enam)**

**Dosen : Tifanny Nabarian, S.Kom., M.T.I.**

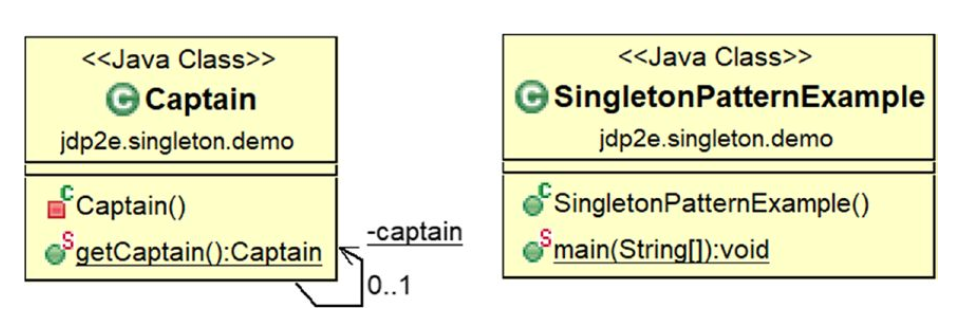
**Nama Mahasiswa : Muhammad Azhar Rasyad**

**NIM : 0110217029**

**Instruksi**

* Kerjakan tugas di bawah ini secara **individu.**
* Kerjakan secara **berurutan.**
* Modifikasi *source code*, dengan cara menambahkan **NIM** Anda pada setiap *caption* yang ditampilkan di *result* yang di*screenshot*.

1. **Buatlah 2 buah kelas dengan struktur sebagai berikut:**



**Captain.java**

|  |
| --- |
| **package** jdp2e.singleton.demo;  **final** **class** Captain {  **private** **static** Captain *captain*;  //We make the constructor private to prevent the use of "new"  **private** Captain() { }  **public** **static** **synchronized** Captain getCaptain()  {  // Lazy initialization  **if** (*captain* == **null**)  {  *captain* = **new** Captain();  System.***out***.println("New captain is elected for your team.");  }  **else**  {  System.***out***.print("You already have a captain for your team.");  System.***out***.println("Send him for the toss.");  }  **return** *captain*;  }  } |

**SingletonPatternExample.java**

|  |
| --- |
| **package** jdp2e.singleton.demo;  **public** **class** SingletonPatternExample {  **public** **static** **void** main(String[] args) {  System.***out***.println("\*\*\*Singleton Pattern Demo\*\*\*\n");  System.***out***.println("Trying to make a captain for your team:");  //Constructor is private.We cannot use "new" here.  //Captain c3 = new Captain();//error    Captain captain1 = Captain.*getCaptain*();  System.***out***.println("Trying to make another captain for your team:");    Captain captain2 = Captain.*getCaptain*();  System.***out***.println("captain1:"+captain1);  System.***out***.println("captain2:"+captain2);  **if** (captain1 == captain2)  {  System.***out***.println("captain1 and captain2 are same instance.");  }  }  } |

1. **Apa hasil dari** *source code* **pada langkah 1 tersebut?Berikan analisis singkat Anda!**

|  |
| --- |
| *Screenshot* Hasil Running *Source code*    Analisis:   * Pada langkah 1 terdapat 2 kelas yaitu Captain.java dan SingletonPatternExample.java. * Kelas Captain.java digunakan untuk menginisialisasi objek Captain dengan menggunakan constructor yang bersifat private. Dalam Captain.java terdapat method getCaptain() yang berisi pengecekan kondisi apakah objek captain sudah ada atau tidak ada   + Jika tidak ada objek captain maka print peringatan captain baru telah dipilih kedalam tim.   + Jika sudah ada captain maka print peringatan bahwa captain sudah mendapatkan tim dan kemudian diberi peringatan agar captain tersebut melakukan toss coin.   Setelah dilakukan pengecekan maka dikembalikan nilai dari objek captain.   * Kelas SingletonPatternExample digunakan sebagai main program untuk membuat objek captain yang baru, akan tetapi untuk membuat objek captain yang baru tidak bisa menggunakan “new” dikarenakan constructor dari kelas Captain.java private hal tersebut bertujuan untuk menghalangi kelas untuk menggunakan new pada objek Captain. Cara lain untuk membuat objek captain dengan memanggil menggunakan methodnya. Setelah satu objek captain terbuat maka dibuat objek captain yang kedua, hal tersebut dilakukan untuk melakukan pengecekan antara objek satu dan kedua apakah sama atau tidak, jika sama maka kedua objek tersebut dalam interface yang sama. Singleton ini bertujuan untuk pengecekan instansiasi suatu objek. |

1. **Perhatikan *source code* pada langkah 1. Dengan kondisi demikian, apakah kelas Captain bisa menurunkan sifatnya ke kelas lain?Jelaskan jawaban Anda!**

|  |
| --- |
| Tidak bisa, karena constructornya bersifat private dan disynchronized sehingga tidak dapat diwariskan sifatnya ke kelas lain |

1. **Buat kembali sebuah kelas, yang merupakan *Lazy Inialization* dari langkah 1, dengan** source code **sebagai berikut:**

**LazyCaptain.java**

|  |
| --- |
| **package** jdp2e.singleton.demo;  **public** **class** LazyCaptain {  **private** **static** LazyCaptain *lazyCaptain*;  // We make the constructor private to prevent the use of "new"  **private** LazyCaptain() {  }  **public** **static** LazyCaptain getCaptain() {  // Lazy initialization  **if** (*lazyCaptain* == **null**) {  *lazyCaptain* = **new** LazyCaptain();  System.***out***.println("New captain is elected for your team.");  } **else** {  System.***out***.print("You already have a captain for your team.");  System.***out***.println("Send him for the toss.");  }  **return** *lazyCaptain*;  }  **public** **static** **void** main(String[] args) {  System.***out***.println("\*\*\*Singleton Pattern Demo\*\*\*\n");  System.***out***.println("Trying to make a captain for your team with lazy style:");    LazyCaptain captain1 = LazyCaptain.*getCaptain*();  System.***out***.println("Captain1:" + captain1);    LazyCaptain captain2 = LazyCaptain.*getCaptain*();  System.***out***.println("Captain2:" + captain2);  }  } |

1. **Apa hasil dari** *source code* **pada langkah empat tersebut? Berikan analisis singkat Anda!**

|  |
| --- |
| *Screenshot* Hasil Running *Source code*    Analisis:   * Pada kelas LazyCaptain.java hampir sama dengan gabungan kelas Captain.java dan SingletonPatternExample.java. Perbedaannya tidak ada synchronized pada method getCaptain() dan pada kelas main tidak terdapat pengecekan antara kedua objek tersebut. * Alur prosesnya yaitu membuat objek captain satu kemudian dicek apakah sudah dimasukkan tim atau belum, karena diawal belum dimasukkan ke dalam tim maka objek captain satu diinisialisasi terlebih dahulu supaya masuk ke dalam tim, dan kemudian dibuat objek captain kedua namun karena objek captain satu sudah diinisialisasi diawal maka akan muncul peringatan bahwa objek captain telah dimasukkan kedalam tim, dan untuk objek captain kedua tidak perlu diinisialisasi lagi sehingga langsung dapat melakukan toss coin. |

1. **Buat kembali sebuah kelas, yang merupakan contoh *Eager Inialization* dari singleton, dengan** source code **sebagai berikut:**

**Captain.java**

|  |
| --- |
| **package** jdp2e.singleton.demo1;  **public** **class** Captain {  //Early initialization  **private** **static** **final** Captain ***captain*** = **new** Captain();  //We make the constructor private to prevent the use of "new"  **private** Captain()  {  System.***out***.println("A captain is elected for your team.");  }  /\* Global point of access.The method getCaptain() is a public static  method\*/  **public** **static** Captain getCaptain()  {  System.***out***.println("You have a captain for your team.");  **return** ***captain***;  }  **public** **static** **void** dummyMethod()  {  System.***out***.println("It is a dummy method");  }  } |

**EagerInitializationExample.java**

|  |
| --- |
| **package** jdp2e.singleton.demo1;  **public** **class** EagerInitializationExample {  **public** **static** **void** main(String[] args) {  System.***out***.println("\*\*\*Singleton Pattern Demo With Eager Initialization\*\*\*\n");  Captain.*dummyMethod*();  }  } |

1. **Apa hasil dari** *source code* **pada langkah enam tersebut?**

**Apakah kalimat “A captain is elected for your team.” masih muncul pada hasil running source code Anda?**

**Berikan penjelasan dari analisis Anda!**

|  |
| --- |
| *Screenshot* Hasil Running *Source code*    Analisis:   * Kaliman “A captain is elected for your team.” Mengapa? Karena Eager Initialitation membuat kelas tanpa ada pengecekan dan otomatis objek terbuat akan tetapi hanya satu objek, itulah mengapa diawal tidak keluar peringatan “It is a dummy” method diawal melainkan diakhir setelah dirun sehingga membuat singleton. |

**\*\*\***