**soal:**

1. Ceritakan apa itu OOP, object oriented programming
2. Di java 8 ada dua komponen untuk implementasi abstraksi lebih baik, yaitu stream dan lambda. Ceritakan apa itu. Pada saat apakah kita membutuhkan stream atau lambda.
3. Apa perbedaan antara interface, abstract kelas.
4. Apa itu functional interface, dibutuhkan pada saat apakah?
5. Terangkan apa itu SOLID principle. Apakah berguna?
6. Jika saya mempunyai integer : 1234567 , buat lah fungsi yang keluarannya 7654321 tanpa menggunakan library, atau string prosesing.
7. Saya memiliki matrix sebagai berikut:

1 2 3 10

⎡4 5 6⎤×⎡11⎤ ⎢⎣7 8 9⎥⎦ ⎢⎣12⎥⎦

Buat matrix class yang mempunya behavior untuk perkalian matrix. Coba untuk menyelesaikan perkalian matrix diatas.

1. Apa itu unit testing. Bedakah dengan integration testing? Apa bedanya?
2. Terangkan apa itu TDD.

Jawaban ..

1. OOP (Object Oriented Programming) adalah suatu metode pemrograman yang berorientasi kepada objek. Tujuan dari OOP diciptakan adalah untuk mempermudah pengembangan program dengan cara mengikuti model yang telah ada di kehidupan sehari-hari.

* Ada 3 istilah dalam OOP yaitu Class, function / methode, Attribut

Contoh penjelasannya:

1. Class kucing
2. Function Sleep(); walk(); eat();
3. Attribut , Warna , berat, Panjang, lebar

* 4 Pilar OOP *Inheritance, Encapsulation , Abstraction, Polymorphism*
* Sederhananya OOP itu sangat diperlukan untuk menghindari kompleksitas suatu program. Jika kita mengikuti prinsip-prinsip ini kita dapat membuat sebuah sistem yang mudah kita gunakan dan kembangkan kembali, kokoh, dan tidak rapuh.

1. – Stream adalah urutan elemen yang mendukung operasi agregat sekuensial dan paralel. Operasi yang dimaksud dapat berupa pemfilteran, modifikasi, dan jenis transformasi yang berbeda.

* Lambda adalah blok kode pendek yang mengambil parameter dan mengembalikan nilai. (***return a value***). Ekspresi lambda mirip dengan metode, tetapi tidak memerlukan nama ***(anonymous function)***.
* Biasanya stream dan lambda digunakan untuk mentransformasi data , misalnya semua data dari collection akan di filter, lalu di uppercase dll. Dan apabila menggunakan Stream dan lambda expression boiler plate akan menjadi lebih singkat dibandingkan dengan ***anonymous class***.

1. Abstrak Class

* Bisa berisi abstract dan non-abstract method.
* Method boleh bersifat static.
* Suatu abstract class hanya bisa meng-*extend* satu abstract class dan meng-implement beberapa interface.

Interface

* Hanya boleh berisi abstract method.
* Method tidak boleh bersifat static.
* Suatu interface bisa meng-*extend*  satu atau lebih interface lainnya.
* Suatu interface hanya bisa meng-*extend* interface lainnya.  Dan tidak bisa meng-*implement* class atau interface lainnya.

1. *Functional Interface* adalah sebuah interface yang hanya mempunyai satu method abstract saja. Jadi semua interface yang kita buat jika interface tersebut hanya memiliki satu abstarct method saja, secara otomatis akan menjadi Functional Interface.

* Biasanya dibutuhkan untuk memanggil fungsi fungsi lambda expression.
* Di karenakan menggunakan lambda expression boilerplate code akan menjadi singkat.

1. SOLID merupakan kumpulan dari beberapa principle yang diwujudkan oleh engineer-engineer yang ahli dibidangnya. SOLID membantu kita  mengembangkan sebuah perangkat lunak dengan tingkat kekukuhan yang tinggi. Sangat perlu karena

* Kita jadi paham bahwa dengan mengikuti prinsip SOLID, kode yang kita buat dapat dengan mudah diekstensi *(extended)* dan dipertahankan *(maintained)*.
* Prinsip SOLID bukanlah suatu hukum atau aturan tertentu yang wajib kita patuhi, melainkan sebuah prinsip yang dimaksudkan untuk membantu kita dalam menuliskan kode yang rapi.
* Toleran terhadap perubahan
* Mudah dipahami
* Komponen dasar dapat digunakan kembali dalam bentuk *software system* lainnya (*reuseable*)

1. Github
2. Github
3. Berbeda,  
   - **Unit** **Test** adalah sebuah mekanisme testing yang kita tujukan untuk codebase yang berjalan di internal aplikasi. (Testing dari sisi code).

* **Integration Test** adalah mekanisme testing dimana kita melakukan running sistem termasuk untuk berhubungan dengan service-service di luar sistem aplikasi kita, misal berhubungan dengan REST API atau mungkin berhubungan dengan service Thrift. (SIT Environment)

1. **Test** **Driven** **Development** adalah pengembangan yang disetir oleh Test. Mudahnya, kita wajib menuliskan *test* terlebih dahulu baru *production code*.

* Sebelum menulis kode, tuliskan *test-*nya terlebih dahulu. Pastikan kita memasukkan semua kemungkinan yang dapat kita pikirkan untuk input dan outputnya.
* Jalankan *test-*nya, dan pastikan *test-*nya *fail* karena belum ada kode apapun untuk membuat *test-*nya *pass*.
* Ketik *working code* seminimum mungkin dengan tujuan agar *test-*nya *pass*.
* Jalankan *test* dan cek apakah *test-*nya *pass*. Jika belum *pass*, maka perbaiki *working code* kita sampai memenuhi ekspektasi dari *test*.
* Merasa *working code* yang kita tulis tadi berantakan ? Jangan khawatir, *Refactor the code*, *do some cleaning and DRY-ing*. Selama *test-*nya masih *pass*, berarti tidak ada masalah dengan kode yang di *refactor* tersebut.