1. OOP (Object Oriented Programming) adalah metode pemrograman yang berorientasi object, yang memecahkan masalahnya dibagi berdasarkan object-object yang berkaitan. Keunggulan dari oop itu, jika dia memiliki code yang sama, maka dia tidak perlu mengulangi code tersebut, kita bisa menggunakan kembali code yang telah ada dengan cara memanggil method nya.

2. Stream adalah komponen implementasi abstraksi yang menawarkan alternatif untuk pengolahan collection, stream ini memberikan kita cara
untuk memproses data lebih efisien, contoh ketika kita memproses data di dalam array, kita bisa mengubahnya menggunakan method stream

Lambda adalah sebuah ekspresi yang menggunakan fungsi anonim (anonymus function) fungsi yang didefinisikan tanpa menggunakan identifier tujuannya untuk menyederhanakan penggunaan interface yang hanya memiliki satu metode abstrak.

Penggunaan stream kita bisa implementasikan saat kita mengolah data yang membutuhkan iterator, dengan adanya stream kita bisa lakukan itu dengan lebih efisien
sedangkan lambda bisa diimplementasikan saat ada sebuah interface yang hanya memiliki satu abstract method (functional interface)

3. - Abstract class adalah sebuah class yang berisi abstract method atau non abstract method yang fungsinya untuk keperluan inheritance atau turunan, method-method yang ada di abstract class akan menjadi definisi umum bagi class-class turunannya.
- sedangkan interface adalah sebuah class yang hanya menampung tipe data constan dan method abstract, dimana method-method ini hanya memuat deklarasi dan struktur method. Interface bisa di implementasikan oleh class turunan yang tidak memiliki method spesifik dari pewarisnya.

4. functional interface adalah class interface yang hanya memiliki satu method abstract. Jadi, setiap class interface yang kita buat hanya memiliki satu method abstract maka itu dinamakan functional interface. Kita bisa mengimplementasikan functional interface pada saat ekspresi lambda.

5. solid principle adalah sebuah prinsip penulisan code dalam metode Object Oriented agar penulisan code lebih baik menurut prinsip penulisan code OOP

berikut saya jelaskan ringkasan dari SOLID principle ini
S = Single Responsibility Principle

membagi kelas sesuai fungsinya, dan kelas itu hanya memiliki satu fungsi dan tanggung jawab, kelas itu dapat berubah sesuai dengan tugasnya

O = Open For extension, Closed For Modification Principle

ini berguna untuk proses pewarisan, jadi kelas induk itu harus mudah untuk diwariskan dan tidak boleh diubah ubah lagi, dan untuk mengubahnya yaitu dengan cara mengubah pada kelas turunannya

L = Liskov Substitute Principle

kelas yang menjadi kelas turunan adalah kelas yang menggunakan/mengimplementasikan behavior dari parentnya, jangan sampai generate overide tapi kita mengosongkan codenya

I = Interface Segregation Principle

pembagian fungsi interface, ini berfungsi agar kelas interface mempunyai tugas spesifik untuk setiap pengimplementasinya

D = Dependency inversion Principle

ini prinsip ketergantungan, jadi modul level lebih tinggi tidak boleh mempunya ketergantungan kepada level yang lebih rendah, kedua level tersebut bergantung pada abstraksi

SOLID Principal ini sangat berguna sekali, agar penulisan code lebih baik menurut prinsipnnya

6.
```
public class TestSoalNo6 {

    public static int fungsiMembalik(int param){
        int tamWhile=param;
        int a=1;
        int jumlah = 0;
        String balik = "";
        while(a<param){
            a=a*10;

        }

        int b=1;
        while (b<a) {
            int z =tamWhile%10;
            balik += String.valueOf(z);
            tamWhile = tamWhile/10;
            b=b*10;
        }
        return Integer.parseInt(balik);
    }

    public static void main(String[] args) {

        Scanner scan = new Scanner(System.in);

        System.out.print("Silahkan Inputkan Angka : ");
        int input = scan.nextInt();
        System.out.println("HASIL BALIK = "+ fungsiMembalik(input));

    }
```

```
}

7. public class TestNo7 {

    public static Scanner inputData = new Scanner(System.in);

    public static void main(String[] args) {

        //variabel
        int barisMatriksA, kolomMatriksA, barisMatriksB, kolomMatriksB;

        System.out.print("Masukkan Jumlah Baris Matriks A : ");
        barisMatriksA = inputData.nextInt();

        System.out.print("Masukkan Jumlah Kolom Matriks A : ");
        kolomMatriksA = inputData.nextInt();

        System.out.print("Masukkan Jumlah Baris Matriks B : ");
        barisMatriksB = inputData.nextInt();

        System.out.print("Masukkan Jumlah Kolom Matriks B : ");
        kolomMatriksB = inputData.nextInt();
        int hasil[][] = new int[barisMatriksA][kolomMatriksA];
        if (kolomMatriksA != barisMatriksB) {
            System.out.println("Tidak Memenuhi Syarat Perkalian Matriks");
        } else {

                hasil = hasilKaliMatriks(barisMatriksA, kolomMatriksA, barisMatriksB,
kolomMatriksB);
            System.out.println("Hasil");
            for (int i = 0; i < barisMatriksA; i++) {
                for (int j = 0; j < kolomMatriksA; j++) {
```

```
                System.out.print(hasil[i][j] + " ");
            }
            System.out.println("");
        }
    }

}

    public static int[][] hasilKaliMatriks(int barisMatriksA, int kolomMatriksA, int barisMatriksB, int kolomMatriksB) {
        int matriksA[][] = new int[barisMatriksA][kolomMatriksA];
        int matriksB[][] = new int[barisMatriksB][kolomMatriksB];
        int hasil[][] = new int[barisMatriksA][kolomMatriksA];
        System.out.println("Matriks A");
        for (int i = 0; i < barisMatriksA; i++) {
            for (int j = 0; j < kolomMatriksA; j++) {
                System.out.print("Matriks A baris - " + (i + 1) + " Kolom - " + (j + 1) + " : ");
                matriksA[i][j] = inputData.nextInt();
            }
        }

        for (int i = 0; i <= barisMatriksA - 1; i++) {
            for (int j = 0; j <= kolomMatriksA - 1; j++) {
                System.out.print(matriksA[i][j] + " ");
            }
            System.out.println();
        }

        System.out.println("Matriks B");
        for (int i = 0; i < barisMatriksB; i++) {
            for (int j = 0; j < kolomMatriksB; j++) {
```

```
            System.out.print("Matriks B baris - " + (i + 1) + " Kolom - " + (j + 1) +
" : ");
            matriksB[i][j] = inputData.nextInt();
        }
    }

    for (int i = 0; i <= barisMatriksB - 1; i++) {
        for (int j = 0; j <= kolomMatriksB - 1; j++) {
            System.out.print(matriksB[i][j] + " ");
        }
        System.out.println();
    }

    for (int i = 0; i < barisMatriksA; i++) {
        for (int j = 0; j < kolomMatriksA; j++) {
            int hasils = 0;
            if (kolomMatriksB == kolomMatriksA) {
                for (int z = 0; z < kolomMatriksB; z++) {
                    hasils += matriksA[i][z] * matriksB[z][j];

                }
            } else {
                for (int z = 0; z < kolomMatriksB; z++) {
                    hasils += matriksA[i][j] * matriksB[j][z];
                }
            }
            hasil[i][j] = hasils;
        }
    }

    return hasil;
  }
```

```
}
```

8. Unit Testing adalah sebuah testing yang dilakukan terhadap objek-objek atau method-method yang dibuat untuk menguji kebenaran dan ketepatan code. Pengujian ini dilakukan di local code.
Sedangkan Integration Testing adalah sebuah testing yang dilakukan terhadap kode yang sudah diimplementasikan. Pengujian ini cenderung menguji aplikasi apakah sudah berjalan sesuai yang diinginkan.

9. TDD adalah singkatan dari Test-driven development merupakan sebuah siklus ppembangunan perangkat lunak yang sangat bergantung pada uji coba yang berulang dan issue yang perubahannya dinamis