Nama : Selpia Meilani

NIM : 20220040148

Kelas : TI22A

Matkul : Pemograman Berorientasi Objek

TUGAS SESI 10

Menganalisa Program

Percobaan 5

Jalankan program dibawah ini, berikan analisa penggunaan try dan catch pada program dibawah ini.

```
public class Exception5 {

    public static void main(String[] args) {
        int bil=10;
        try
        {
            System.out.println(bil/0);
        }
        catch(ArithmeticException e)
        {
            System.out.println("Pesan error:  ");
            System.out.println(e.getMessage());
            System.out.println("Info stack erase");
            e.printStackTrace();
            e.printStackTrace(System.out);
        }
        catch(Exception e)
        {
            System.out.println("Ini menghandle error yang terjadi");
        }
    }

}
```

Analisa dari program tersebut :

Program ini menunjukan bagaimana **try** dan **catch** dapat diguakan untuk menangani pengecualian yang mungkin terjadi saat melakukan operasi yang berpotensi berbahaya. Penggunaan **catch** blok khusus untuk **ArithmeticException** memungkinkan program untuk memberikan penaganan yang lebih spesifik untuk pengecualian tersebut. Blok **cacth** yang lebih umum untuk **Exception** memberikan penanganan umum utnuk semua jenis pengecualian lainnya. Juga Stack trace yang dicetak oleh **e.printStackTrace()** memberikan informasi berharga tentang lokasi dan penyebab pengecualian, yang memvantu dalam debugging.

Pemograman **try** dan **cacth** adalah mekanisme penting dalam pemograman java untuk menangani pengcualian dan memastikan program berjalan denga lancar meskipun terjadi kesalahan.

Percobaan 6

Jalankan program dibawah ini, berikan analisa penggunaan try dan catch pada program dibawah ini.

```
public class ThrowExample {

    static void demo()
    {
        NullPointerException t;
        t=new NullPointerException("Coba Throw");
        throw t;
```



```
        // Baris ini tidak lagi dikerjakan;
        System.out.println("Ini tidak lagi dicetak");
    }

    public static void main(String[] args) {
        try
        {
            demo();
            System.out.println("Selesai");
        }
        catch(NullPointerException e)
        {
            System.out.println("Ada pesan error: "+e);
        }
    }

}
```

Analisa dari program tersebut :

Bisa menggunakan **throw** untuk menangani kondisi error dalam kode yang mungkin tidak secara langsung menghasilkan pengecualian. Misalnya ika ingin memastikan bahwa input user berada dalam range tertentu, Anda bisa menggunakan **throw** untuk memicu **IlleglArgumentException** jika input tidak valid.

Throw dan **try-cacth** adalah alat yang penting dalam pemograman java untuk menangani error dan memastikan program berjalan dengan baik, bahkan ketika terjadi kondisi yang tidak diharapkan.

Percobaan 7

Jalankan program dibawah ini, berikan analisa penggunaan try dan catch pada program dibawah ini.

```
public class ThrowExample2 {

    public static void main(String[] args) {
        try
        {
            throw new Exception("Here's my Exception");
        }
        catch(Exception e)
        {
            System.out.println("Caught Exception");
            System.out.println("e.getMessage():"+e.getMessage());
            System.out.println("e.toString():"+e.toString());
            System.out.println("e.printStackTrace():");
            e.printStackTrace();
        }
    }
}
```

Analisa dari program tersebut :

main() Method:

- **try** block: Kode yang berpotensi menyebabkan pengecualian (yaitu, melempar pengecualian **Exception** baru) diletakkan di dalam **try** block.
- **throw new Exception("Here's my Exception");**: Kode ini memicu pengecualian **Exception** baru dengan pesan "Here's my Exception".
- **catch(Exception e)**: **catch** block ini menangkap pengecualian **Exception** yang dilemparkan dalam **try** block.
- **System.out.println("Caught Exception");**: Menampilkan pesan "Caught Exception" untuk menunjukkan bahwa pengecualian telah ditangkap.
- **System.out.println("e.getMessage():"+e.getMessage());**: Menampilkan pesan error yang terkait dengan pengecualian. Dalam hal ini, akan menampilkan "Here's my Exception".
- **System.out.println("e.toString():"+e.toString());**: Menampilkan representasi teks dari objek pengecualian. Ini biasanya berisi nama kelas pengecualian dan pesan error. System.out.println("e.printStackTrace():");: Menampilkan stack trace dari pengecualian. Stack trace menunjukkan urutan method yang dipanggil sebelum pengecualian terjadi. Ini membantu dalam debugging untuk menemukan sumber masalah.
- **e.printStackTrace();**: Menampilkan stack trace ke aliran output standar (System.out).

Percobaan 8

Jalankan program dibawah ini, berikan analisa penggunaan throws pada program dibawah ini.

```
import java.io.*;

public class Test3 {
    public void methodA(){
        System.out.println("Method A");
    }
    public void methodB() throws IOException
    {
        System.out.println(20/0);
        System.out.println("Method B");
    }
}
```



```
class Utama
{
     public static void main(String[] args) throws IOException
     {
        Test3 c=new Test3();
        c.methodA();
        c.methodB();
     }
}
```

Analisa dari program tersebut :

throws dalam methodB():

- **public void methodB() throws IOException** menyatakan bahwa **methodB()** mungkin melempar pengecualian jenis **IOException**. Ini memberi tahu pemanggil bahwa mereka harus siap untuk menangani pengecualian tersebut atau mendeklarasikannya ulang.

throws dalam main():

- **public static void main(String[] args) throws IOException** menyatakan bahwa **main()** juga mungkin melempar pengecualian jenis IOException. Karena **main()** adalah titik awal program, hal ini berarti program akan berakhir jika pengecualian yang dilempar oleh **methodB()** tidak ditangani.

Kemudian coba ubah class utama diatas dengan yang program baru di bawah ini:

```
class Utama
{
     public static void main(String[] args)
     {
           Test3 o=new Test3();
           o.methodA();
           try
           {
                o.methodB();
           }
           catch(Exception e)
           {
               System.out.println("Error di Method B");
           }
           finally
           {
               System.out.println("Ini selalu dicetak");
           };
     }
}
```

Analisa kelanjutan dari program tersebut:

try-catch dalam main():

- **try** block membungkus pemanggilan **o.methodB().** Ini menandakan bahwa kode di dalam **try** block berpotensi melempar pengecualian.
- **catch(Exception e)** menangkap setiap pengecualian yang dilempar oleh **methodB(**). Dalam hal ini, itu adalah **ArithmeticException.**
- **finally** block akan selalu dijalankan setelah **try** block, baik pengecualian terjadi atau tidak.