VERSION 1.0

13 Maret 2024

PEMROGRAMAN BERORIENTASI OBJEK

MODUL 5 – ARRAYS, ARRAYLISTS, ITERATOR, EXCEPTIONS

DISUSUN OLEH:

TAUFIQ RAMADHAN

SUTRISNO ADIT PRATAMA

DIAUDIT OLEH:

Ir. Galih Wasis Wicaksono, S.Kom, M.Cs.

PRESENTED BY: TIM LAB. IT

UNIVERSITAS MUHAMMADIYAH MALANG

PEMROGRAMAN BERORIENTASI OBJEK

TUJUAN

1. Mahasiswa dapa memahami konsep dari *Arrays* dan *ArrayList*.
2. Mahasiswa dapat memahami *exception* dan *error* pada program.
3. Mahasiswa dapat memahami *exception handling* dengan Java.

TARGET MODUL

1. Mahasiswa dapat menerapkan *error/exception handling* pada program Java.
2. Mahasiswa dapat menerapkan pembuatan *custom exception.*
3. Mahasiswa dapat menerapkan pembuatan Arrays dan ArrayList.
4. Mahasiswa mengetahui perbedaan *Arrays* dan *ArrayList.*
5. Mahasiswa dapat membuat *Arrays* dan *ArrayList.*

PERSIAPAN

1. Java Development Kit.
2. Text Editor / IDE (Visual Studio Code, Netbeans, Intellij IDEA, atau yang lainnya).

KEYWORDS

- Arrays
- ArrayList
- Error
- Custom Exception
- Exception Handling

TEORI

- **Arrays**

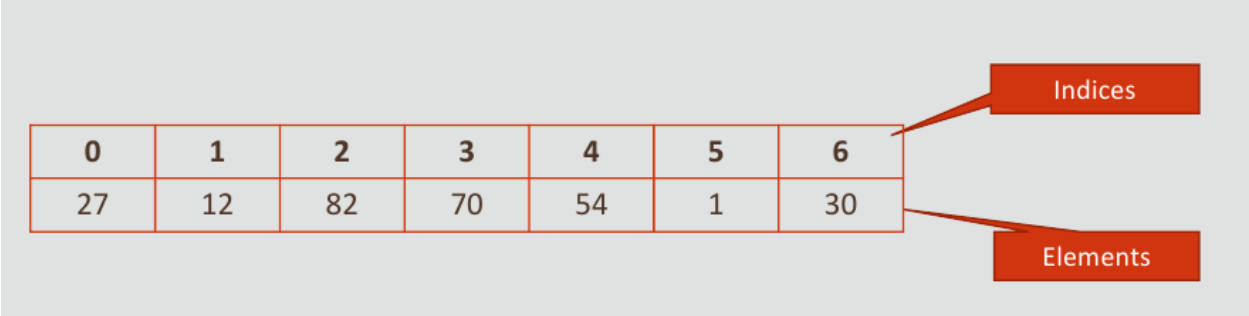
Arrays adalah sebuah variabel yang bisa menyimpan banyak data dalam satu variabel. Array menggunakan indeks untuk melakukan penyimpanan dan juga akses terhadap data yang disimpan di dalamnya.

Permasalahan ketika ingin menyimpan satu data yang sejenis seperti berikut:

```
String nama1 = "Andi";
String nama2 = "Faizal";
String nama3 = "Wempy";
String nama4 = "Yusuf";
```

Ketika dilihat pada kode di atas, menyimpan data nama bisa menggunakan sebuah pemberian nomor pada setiap variabel nama1, nama2, nama3, nama4. Jika data memang benar hanya ada 4 macam, hal itu bukan permasalahan. Tetapi bagaimana jika sebuah data yang ingin disimpan adalah berisi data nama yang berjumlah 1000? Tentu sangat tidak efisien jika menulis semua data secara manual.

Maka dari itu ini adalah fungsi dari penggunaan *arrays.* Secara sederhana bentuk dari *arrays* bisa diilustrasikan seperti berikut:



Indices artinya indeks dari sebuah array dan *Elements* adalah sebuah nilai yang ada di indeks tersebut. Hal yang perlu diperhatikan di sini adalah indeks pertama ialah indeks **0** dan indeks terakhir adalah **6** dengan panjang array adalah **7**. Cara baca sebuah array jika kita ingin mengakses sebuah nilai 70, maka indeks ke-3 yang berada di urutan ke-4 memiliki nilai elemen 70.

Arrays bisa digunakan pada setiap tipe data baik itu tipe data primitif ataupun tipe data non-primitif. Tetapi, setiap elemen di dalam *arrays* harus memiliki tipe data yang sama.

Ketika membuat sebuah array dengan tipe data String maka isi di dalam array tersebut haruslah mengandung tipe data String semua, begitupun jika membuat array dengan tipe data Int maka isi di dalam array haruslah mengandung tipe data Int semua.

Cara untuk mendekrasikan sebuah *Arrays* kosong, terdapat 3 macam yaitu:

```
// cara pertama
String[] nama;

//cara kedua
String nama[];

// cara ketiga
String[] nama = new String[5];
```

Ketiga kode di atas meskipun berbeda dalam penulisan, tetapi memiliki arti yang sama yaitu mendeklarasikan sebuah *Arrays* kosong. Berikut adalah penjelasannya:

- Menggunakan kurung siku **[ ]** untuk membuat array.
- Kurung siku bisa diletakkan setelah tipe data atau nama array.
- Angka 5 dalam kurung artinya batas atau ukuran dari array yang dibuat.
- Nilai dari array harus ditentukan karena hal itu menentukan nilai array yang bisa disimpan.
- Array tidak bisa bertambah besar atau mengecil, jika array dibuat dengan ukuran 5 maka seterusnya hanya bisa menyimpan 5.

Sebagai contoh jika ingin membuat sebuah array yang bisa menyimpan sampai 5 data dengan tipe data String, maka bisa menulis kode seperti berikut:

```
public class Main {
    public static void main(String[] args) {
        String nama[];
        nama = new String[100];
    }
}
```

Atau bisa juga seperti berikut:

```
public class Main {
    public static void main(String[] args) {
        String nama[] = new String[100];
    }
}
```

Arrays kosong dengan data yang bisa menyimpan data sebanyak 5 data dengan tipe data String sudah siap dipakai. *Arrays* tersebut bisa diisi dengan cara seperti berikut:

```
public class Main {
    public static void main(String[] args) {
        String nama[] = new String[5];

        nama[0] = "Andi";
        nama[1] = "Faizal";
        nama[2] = "Wempy";
        nama[3] = "Yusuf";
        nama[4] = "Adit";
    }
}
```

Atau bisa juga langsung seperti berikut:

```
public class Main {
    public static void main(String[] args) {
        String nama[] = {"Andi", "Faizal", "Wempy", "Yusuf", "Adit"};
    }
}
```

Bisa dilihat pada kode kedua berikut berbeda dengan kode yang sebelumnya, cara inisialisasi tidak menggunakan ukuran array tetapi langsung isi dari array-nya. Hal tersebut sama saja tetapi untuk nilai array harus di dalam kurung kurawal **{ }** dan panjang array akan mengikuti pada isi data di dalam array.

Untuk cara mengakses atau mengambil data dari array secara individu bisa menggunakan *notation bracket*, contoh:

```
String nama[] = {"Andi", "Faizal", "Wempy", "Yusuf", "Adit"};

System.out.println(nama[2]);
```

Ketika dijalankan maka output program yaitu **“Wempy”**. Karena yang berada di indeks ke-2 adalah nilai String **“Wempy”**.

Cara untuk mengubah nilai yang ada di dalam array, bisa menggunakan cara seperti mengisi nilai ke dalam array-nya. Contoh:

```
public class Main {
    public static void main(String[] args) {
    String nama[] = {"Andi", "Faizal", "Wempy", "Yusuf", "Adit"};

    nama[0] = "Rahma";
```

```
    nama[1] = "Maylani";

    System.out.println(nama[0]);
    System.out.println(nama[1]);
    }
}
```

Ketika program dijalankan, maka outputnya akan seperti berikut:

```
Rahma
Maylani
```

Ketika sebuah *Arrays* dideklarasi tetapi belum diinisialisasi sebuah nilai ke dalamnya, maka isi elemen di masing-masing indeks akan diisi dengan default value sesuai tipe data. Contoh jika mempunyai array dengan tipe data String dan Int seperti berikut:

```
public class Main {
    public static void main(String[] args) {
    String nama[] = new String[5];
    int umur[] = new int[5];
    }
}
```

Lalu coba untuk mengakses data yang ada di dalam masing-masing array.

```
public class Main {
    public static void main(String[] args) {
    String nama[] = new String[5];
    int umur[] = new int[5];

    System.out.println(nama[0]);
    System.out.println(umur[0]);
    }
}
```

Output program akan seperti berikut:

```
null
0
```

Lebih jauh tentang array, ketika membuat array dengan cara deklarasi ataupun langsung inisialisasi, isi dari panjang array tersebut sudah tidak bisa diubah lagi. Untuk cara mengakses panjang dari sebuah array alih-alih menghitung manual, bisa gunakan properti *length* dari array itu sendiri.

```java
public class Main {
    public static void main(String[] args) {
    String nama[] = {"Andi", "Faizal", "Wempy", "Yusuf", "Adit"};
    System.out.println("Panjang array: " + nama.length);
    }
}
```

Output program:

```
Panjang array: 5
PS C:\Users\radan>
```

Untuk cara mengakses semua array secara otomatis dan bisa menyesuaikan pada panjang array bisa menggunakan *Looping*. Contoh untuk mengakses nama-nama pada array String nama sebelumnya, bisa menggunakan cara seperti ini:

```java
public class Main {
    public static void main(String[] args) {
        String nama[] = {"Andi", "Faizal", "Wempy", "Yusuf", "Adit"};

        for (int i = 0; i < nama.length; i++){
            System.out.println(nama[i]);
        }
    }
}
```

Output program:

```
Andi
Faizal
Wempy
Yusuf
Adit
PS C:\Users\radan>
```

Atau juga bisa menggunakan *for-each* agar lebih mudah:

```java
public class Main {
    public static void main(String[] args) {
        String nama[] = {"Andi", "Faizal", "Wempy", "Yusuf", "Adit"};

        for (String nm : nama) {
            System.out.println(nm);
        }
    }
}
```

Dengan output program yang sama.

- **Arrays Multi-Dimensi**

Array multi dimensi artinya array yang memiliki lebih dari satu dimensi, atau yang biasa disebut dengan array di dalam array. Untuk jumlah dimensi yang bisa dibuat di dalam array multi-dimensi tidak terbatas, sesuai dengan kebutuhan. Contoh berikut adalah Array multi-dimensi:

```
public class Main {
    public static void main(String[] args) {
        String[][] students = {
            {"Adit", "203"},
            {"Wempy", "120"},
            {"Luthfia", "182"}
        };
    }
}
```

Berbeda dengan indeks array biasa, indeks pada array multi-dimensi contoh ke-0 akan berisi array **{“Adit”, “203”}.**

	0	1
0	"Adit"	"203"
1	"Wempy"	"120"
2	"Luthfia"	"182"

Sebagai ilustrasi ialah tabel di atas, ketika mengakses array di atas bisa menggunakan kode berikut:

```
System.out.println(students[0][0]);
```

Arti dari kode di atas adalah mengakses isi array pada variabel students pada indeks ke-0 yang di dalamnya akses indeks ke-0, maka akan output yaitu **Adit**. Jika diilustrasikan pada tabel maka kurung siku pertama akan mengakses pada baris dan untuk kurung siku kedua akan mengakses pada kolom yang ditunjuk sesuai baris. Untuk cara mengakses array multi-dimensi bisa menggunakan *Looping* seperti berikut:

```
public class Main {
    public static void main(String[] args) {
        String[][] students = {
            {"Adit", "203"},
            {"Wempy", "120"},
```

```
            {"Luthfia", "182"}
        };

        for (int i = 0; i < students.length; i++){
            for (int j = 0; j < students[i].length; j++){
                if (j == 0){
                    System.out.println("Nama : " + students[i][j]);
                } else {
                    System.out.println("Nim : " + students[i][j]);
                }
            }
        }
    }
}
```

Output program:

```
Nama : Adit
Nim : 203
Nama : Wempy
Nim : 120
Nama : Luthfia
Nim : 182
PS C:\Users\radan>
```

Jika dilihat pada cara mengakses data di dalam array multi-dimensi di atas, loop yang digunakan ialah *looping* bersarang. Hal demikian juga jika membuat sebuah array dengan 3D, 4D, 5D dan seterusnya looping yang digunakan untuk mengakses adalah *looping* bersarang menyesuaikan dengan jumlah dimensi array-nya.

- **ArrayLists**

  Arrays yang sebelumnya dibahas dan dicoba untuk dibuat di atas sebenarnya memiliki kelemahan, yaitu:

  - Tidak bisa menyimpan data dengan tipe data yang berbeda
  - Tidak bisa menghapus atau menambahkan data baru jika array sudah penuh
  - Ukuran tidak dinamis, tidak bisa dikurangi atau juga ditambah

  Maka dari itu, ArrayList ada untuk melengkapi kekurangan yang ada di *Arrays* biasa. *Arraylists* adalah sebuah class yang memungkinkan untuk membuat sebuah objek untuk menampung data dalam bentuk apapun.

Untuk menggunakan *ArrayLists*, hal pertama yang harus dilakukan adalah meng-*import* terlebih dahulu di bagian atas kode. Seperti berikut:

```
import java.util.ArrayList;
```

Setelah melakukan import, lalu bisa buat sebuah objek *ArrayLists* seperti berikut:

```
Run | Debug
public static void main(String[] args) {
    ArrayList penampung = new ArrayList<>();
}
```

Keistimewaan *ArrayLists* bisa menyimpan sebuah data dengan tipe data yang berbeda, tetapi masih dalam bentuk object, termasuk juga class yang dibuat secara manual.

Ketika ingin mengakses pada elemen yang ada di dalam *ArrayLists*, tidak bisa lagi menggunakan *index notation* seperti pada *Arrays* biasa. Untuk cara mengaksesnya bisa menggunakan pada *method-method* yang sudah disediakan pada class *ArrayLists*. Berikut adalah beberapa method yang ada di *ArrayLists*.

add(value)	Menambahkan *value* ke akhir dari sebuah array list
add(index, value)	Memasukkan *value* tepat sebelum index yang ditentukan, menggeser nilai sebelumnya ke sebelah kanan
clear()	Menghapus semua elemen dari arraylist
indexOf(value)	Mengembalikan index pertama yang di mana *value* ditemukan di dalam array list (return -1 jika tidak ditemukan)
get(index)	Mengembalikan nilai pada index yang ditentukan
remove(index)	Menghapus nilai pada index yang ditentukan, menggeser nilai setelahnya ke kiri
set(index, value)	Mengganti *value* pada index yang ditentukan dengan *value* yang diberikan
size()	Mengembalikan jumlah elemen pada array list
toString()	Mengembalikan representasi String dari arraylist, seperti "[3, 42, -7, 15"

Untuk lebih lengkap method apa saja yang ada di class *ArrayLists* bisa cek link di sini

Kelebihan jika menggunakan *ArrayLists*, yaitu:

1) Ukuran *ArrayList* bisa bertambah besar jika ditambah elemen baru
2) Ukuran *ArrayList* bisa berkurang jika elemen dihapus

3) Terdapat beberapa method yang memudahkan dalam pengoperasiannya

Contoh implementasi dengan menggunakan *ArrayLists*:

```
import java.util.ArrayList;

public class Main {
    public static void main(String[] args) {
        // membuat objek arraylist
        ArrayList penampung = new ArrayList<>();

        // mengisi array list dengan 5 data dan tipe yang berbeda
        penampung.add("Adit");
        penampung.add(1994);
        penampung.add("Informatika");
        penampung.add(3.154232);
        penampung.add(true);

        // cara outputkan semua
        System.out.println(penampung);

        // menghapus 1994 dari penampung
        penampung.remove(1);
        System.out.println(penampung);

        // akses dengan index tertentu
        System.out.println(penampung.get(1));

        // melihat ukuran arraylist
        System.out.println("Array berisi : " + penampung.size());
    }
}
```

Output program:

```
[Adit, 1994, Informatika, 3.154232, true]
[Adit, Informatika, 3.154232, true]
Informatika
Array berisi : 4
PS C:\Users\radan>
```

Untuk mengakses *ArrayList* bisa menggunakan *looping for* ataupun *foreach*. Seperti berikut:

```
import java.util.ArrayList;

public class Main {
```

```
    public static void main(String[] args) {
        // membuat objek arraylist
        ArrayList penampung = new ArrayList<>();

        // mengisi array list dengan 5 data dan tipe yang berbeda
        penampung.add("Adit");
        penampung.add(1994);
        penampung.add("Informatika");
        penampung.add(3.154232);
        penampung.add(true);

        // akses dengan foreach
        for (Object objek : penampung) {
            System.out.println(objek);
        }

        // akses dengan looping for
        for (int i = 0; i < penampung.size(); i++){
            System.out.println(penampung.get(i));
        }
    }
}
```

Setelah mengetahui cara untuk membuat sebuah *ArrayList*, selanjutnya ialah cara agar sebuah *ArrayList* hanya bisa menyimpan satu data yang sejenis maka bisa digunakan sebuah *Generics data type.* Pengertian lengkap tentang apa itu *Generics* akan dibahas di semester yang akan datang, secara sederhana *Generics data type* pada *ArrayLists* digunakan untuk mespesifikkan tipe data yang digunakan. Contoh, jika ingin membuat *ArrayList* dengan tipe data *String* untuk menyimpan data nama:

```
public static void main(String[] args) throws Exception {

    ArrayList<String> names = new ArrayList<>();

}
```

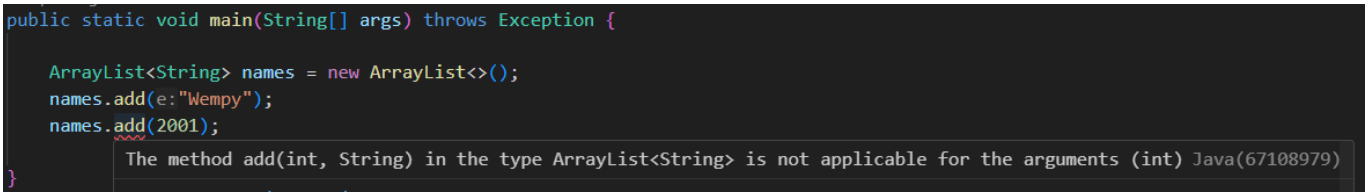
Perhatikan pada kode **<String>**, ini adalah contoh pembuatan ArrayList dengan tipe data String. Ketika sudah membuat *ArrayList* seperti ini, maka data yang bisa ditambahkan ke dalam variabel *names* hanya data dengan tipe data *String*. Adapun contoh pengisiannya seperti berikut:

```
public static void main(String[] args) throws Exception {

    ArrayList<String> names = new ArrayList<>();
    names.add("Wempy");
    names.add("Rahma");
    names.add("Adit");

}
```

Jika memaksa untuk menambahkan data selain tipe data yang sesuai maka akan muncul error seperti ini:



- **ArrayLists Multi-Dimensi**

  ArrayLists tidak hanya memiliki sebuah kemampuan penyimpanan data yang dinamis, *ArrayLists* juga bisa digunakan untuk menyimpan sebuah data dalam bentuk 2 dimensi, 3 dimensi dan seterusnya sesuai dengan kebutuhan. Untuk pembuatan sebuah *ArrayLists* multi-dimensi berbeda dengan membuat *Arrays* multi-dimensi. Sebuah contoh jika ingin menyimpan sebuah data nama dan nim dengan masing-masing nama dan nim tersimpan terpisah, bisa digunakan sebuah *ArrayList* di dalam *ArrayList*, seperti berikut:

```
public static void main(String[] args) {

    ArrayList<ArrayList<String>> identity = new ArrayList<ArrayList<String>>();

}
```

Untuk cara menambahkan data ke dalam variabel *identity*, diharuskan untuk membuat sebuah variabel tambahan yang digunakan untuk menyimpan *ArrayList* 1 dimesi sebelum diinputkan ke dalam *ArrayList* 2 dimensi. Contohnya:

```
public static void main(String[] args){

    ArrayList<ArrayList<String>> identity = new ArrayList<ArrayList<String>>();

    // variabel pembantu dengan nama temp
    ArrayList<String> temp = new ArrayList<>();
```

```
    temp.add("Sutrisno Adit Pratama");
    temp.add("202210370311203");

    // input hasil variabel temp ke dalam ArrayList pembantu
    identity.add(temp);
}
```

Untuk cara mengakses data yang ada di dalam *ArrayList* multi-dimensi caranya hampir sama dengan mengakses *Arrays* multi-dimensi, yaitu dengan cara mengakses pada index paling luar dan dilanjutkan pada index yang bagian dalamnya. Contoh cara aksesnya:

```
    System.out.println("Nama : " + identity.get(0).get(0));
    System.out.println("NIM : " + identity.get(0).get(1));
```

Maka output program akan seperti ini:

```
Nama : Sutrisno Adit Pratama
NIM : 202210370311203
PS C:\Users\radan\Music\modul5>
```

Untuk cara mengakses data pada *ArrayList* dengan jumlah data yang banyak, bisa gunakan looping dan sesuaikan seperti contoh pada *Arrays* multi-dimensi.

- **Iterator**

Iterator adalah library bawaan dari java yang bisa digunakan untuk menampilkan isi dari *ArrayList* dengan step maju atau dengan kata lain mengakses isi dari *index* terendah ke *index* terbesar. Iterator memiliki hubungan seperti halnya sebuah *Node* di Java, untuk lebih lengkapnya terkait *Node* akan dibahas di semester selanjutnya.

Penggunaan *Iterator* mengharuskan import sebuah library **java.util.Iterator** dan method yang ada di dalam *Iterator* yaitu *hasNext(), next(), remove().* Contoh penggunaan *Iterator*:

```
public static void main(String[] args){
    ArrayList<String> names = new ArrayList<>();
    names.add("Wempy");
    names.add("Yusuf");
    names.add("Zaky");
    Iterator<String> iterator = names.iterator();
    while (iterator.hasNext()) {
        System.out.println("Nama: " + iterator.next());
    }
}
```

Pada kode yang dicetak tebal, **Iterator** adalah sebuah class *Iterator,* **<String>** adalah sebuah *Generics data type* yang harus sesuai dengan *Generics data type* yang ada di *ArrayList* yang bersangkutan. **iterator** adalah sebuah nama variabel, **names.iterator()** adalah *ArrayList* yang digunakan untuk menyimpan nama dan diubah menjadi *Iterator.* Dan pada kode *while loop* digunakan untuk mengecek apakah variabel **iterator** memiliki isi, jika iya maka output isinya dengan menggunakan method **next()**.

- **ListIterator**

  ListIterator adalah sebuah *class* di java yang memiliki karakteristik hampir sama dengan *Iterator.* Hal yang membedakan yaitu, *ListIterator* tidak memiliki method **remove()** dan *ListIterator* bisa digunakan untuk mengakses sebuah *ArrayList* dari index terkecil ke terbesar dan terbesar ke terkecil. Untuk menggunakan *ListIterator* diharuskan untuk import library **java.util.ListIterator**.

  Contoh penggunaannya:

```
import java.util.ArrayList;
import java.util.ListIterator;

public class App {
    public static void main(String[] args){
        ArrayList<String> names = new ArrayList<>();
        names.add("Wempy");
        names.add("Yusuf");
        names.add("Zaky");

        ListIterator<String> listIter = names.listIterator();
        System.out.println("Akses dari index terkecil");
        while (listIter.hasNext()) {
            System.out.println("Nama: " + listIter.next());
        }

        System.out.println("Akses dari index terbesar");
        while (listIter.hasPrevious()) {
            System.out.println("Nama: " + listIter.previous());
        }
    }
}
```

- **Wrapper Classes**

Sebuah *ArrayList* hanya bisa menyimpan sebuah tipe data yang berbentuk object bukan tipe data primitif. Penyimpanan data dalam *ArrayList* tidak bisa seperti berikut:

```
Run | Debug
public static void main(String[] args){
    ArrayList<int> numbers = new ArrayList<int>();
```

Hal ini dikarenakan **int** adalah tipe data primitif bukan sebuah tipe object. Untuk menyimpan sebuah data **int** ke dalam *ArrayList* menggunakan sebuah *Wrapper class* dari **int**. Jadi perbaikan pada kode di atas menjadi seperti ini:

```
Run | Debug
public static void main(String[] args){
    ArrayList<Integer> numbers = new ArrayList<Integer>();
```

Wrapper classes adalah sebuah class yang disediakan oleh Java untuk bertanggungjawab atau menangani sebuah tipe data primitif. *Wrapper classes* akan melakukan enkapsulasi, atau membungkus sebuah tipe data primitif dengan bentuk *Object.* Untuk daftar *wrapper class,* yaitu:

Primitive Type	Wrapper class
boolean	Boolean
char	Character
byte	Byte
short	Short
int	Integer
long	Long
float	Float
double	Double

Dalam *wrapper classes* terdapat 2 istilah yaitu *AutoBoxing* dan *Unboxing*. Kedua istilah ini adalah sebuah fitur yang disediakan oleh java untuk otomatis melakukan konversi tipe data primitif ke *wrapper class*-nya dan sebaliknya. Hal ini bisa membuat kode yang sedang dibangun lebih mudah ditulis dan bersih, sehingga mudah dibaca ulang.

a) *AutoBoxing*

Adalah konversi otomatis yang dilakukan oleh compiler dari tipe data primitif ke *wrapper class*-nya. Contoh:

```
Double nilai = 80.19;
```

b) *Unboxing*

Adalah konversi manual yang dilakukan dari *wrapper class* ke tipe data primitif yang bersangkutan. Contoh:

```
Double nilai = 80.19;
double nilai_asli = nilai; // unboxing
```

Contoh untuk lebih lengkapnya:

```
import java.util.ArrayList;
public class App {
    public static void main(String[] args){
        ArrayList<Integer> numbers = new ArrayList<>();
        for (int i = 1; i < 50; i++){
            numbers.add(i); // bentuk AutoBoxing
        }

        for (Integer i: numbers){
            int no = i; // bentuk Unboxing
            System.out.println(no);
        }
    }
}
```

- **Exception**

Exception adalah sebuah eror yang muncul ketika program dieksekusi (dijalankan) yang mengganggu pada alur normal program java berjalan. Tetapi hal tersebut bisa ditangani di dalam program dan memberikan sebuah kondisi perbaikan jika program membutuhkan sehingga program bisa melanjutkan eksekusinya atau istilahnya adalah *exception handling.*

Sebuah *exception* harus ditangani karena ketika *exception* muncul ketika program berjalan, maka program akan *terminated* dan sebuah *stack trace* akan muncul dengan detail penyebab expetion terjadi di console. Contoh ketika tidak menangani sebuah *exception* pada program sederhana berikut:

```
public static void main(String[] args){
    int devider = 0;
    int value = 10 / devider;
    System.out.println(value);
}
```

Pada contoh kode di atas, bagian kode **int value = 10 / devider;** akan memunculkan sebuah *exception* karena tidak bisa dibagi dengan nilai 0, maka baris kode di bawahnya tidak akan dieksekusi, program akan berhenti dan sebuah *stack trace* akan muncul di *console*.

Jadi ketika sebuah program tidak ada penanganan *exception* maka ketika Java menemukan sebuah eror atau kondisi dimana mencegah eksekusi program secara normal, maka Java akan melemparkan sebuah *exception* atau “throws exception”. Dan jika exeption tidak tertangkap atau tidak tertangani dengan baik, maka program akan *crashes.* Deskripsi tentang exception dan *stack trace* akan ditampilkan di *console.*

- **Exception Handling**

Salah satu cara untuk menangani sebuah *exception* ialah dengan cara menghindarinya dari awal. Contoh pada kode di atas bisa diperbaiki dan ditangani *exception* dengan menggunakan sebuah kondisi:

```
public static void main(String[] args){
    int devider = 0;
    if (devider == 0){
```

```
        System.out.println("tidak bisa dibagi dengan 0");
    } else {
        int value = 10 / devider;
        System.out.println(value);
    }
}
```

Di dalam java, *exception* terbagi menjadi 2 kategori yaitu:

a) *Checked Exceptions*

Adalah sebuah *exception* yang dilakukan pengecekan oleh compiler ketika program dilakukan *compiling*. Jika *exception* tidak ditangani dengan baik di dalam program, maka akan memunculkan *compilation error.* Contohnya *FileNotFoundException, IOException.*

b) *Unchecked Exceptions*

Adalah sebuah exception yang tidak dilakukan pengecekan ketika sedang *compiling.* Contohnya *ArrayIndexOutOfBoundsException, NullPointerException, ArithmeticException.*

- **try/catch block**

Tidak semua *exception* bisa ditangani dengan sebuah kondisi, karena kita tidak selalu tahu sebuah operasi apa yang akan memunculkan sebuah *exception.* Maka dari itu strategi lain ialah menggunakan *try/catch block* untuk menangani *exception.*

Untuk memahami tentang sebuah *try/catch block*, bisa ikuti step berikut:

- Untuk kode yang berisiko memunculkan sebuah *exception*, bisa ditulis di dalam blok "try"
- Asosiasikan sebuah *exception handlers* dengan blok "try" dengan menyediakan 1 atau lebih blok "catch" setelah blok "try".
- Setiap blok *catch* menangani tipe *exception* yang diindikasikan pada argumennya.
- Argumen tipe *exception* mendeklarasikan tipe dari *exception*

Contoh struktur:

```
public static void main(String[] args){
    try {
        // kode yang beresiko dan menyebabkan
        // sebuah exceptin
    } catch (Exception e) {
        // handle exception
    }
    System.out.println("Setelah exception");
}
```

Untuk alur *try/catch block* yang berhasil, kode di dalam *try* dieksekusi dan tidak ada exception maka kode di dalam *catch* akan dilewati dan lanjut ke kode **System.out.println**. Untuk alur *try/catch block* yang gagal, kode di dalam *try dieksekusi* dan terdapat *exception* yang muncul, maka kode di dalam *catch* akan dieksekusi juga dan lanjut ke kode di bawahnya.

Berikut adalah contoh lengkapnya, yaitu program sederhana input nilai A dan B. *Exception* akan bangkit ketika nilai A dan B yang diinput bernilai sama:

```
import java.util.Scanner;
public class App {
    public static void main(String[] args){
        int value = 100, result;
        try {
            System.out.print("Input sebuah nilai: ");
            Scanner scanner = new Scanner(System.in);
            int inputA = scanner.nextInt();
            System.out.print("Input nilai kedua: ");
            int inputB = scanner.nextInt();
            result = value / (inputA - inputB);
            System.out.println("Hasilnya : " + result);
        } catch (Exception e) {
            String errorMessage = e.getMessage();
            System.out.println(errorMessage);
        }
        System.out.println("setelah blok try/catch");
    }
}
```

- **try...catch...finally**

  Penerapan *exceptions handling* adalah menggunakan *scope try/catch* dan finally (opsional), kode yang ada pada blok *try* adalah kode yang kemungkinan akan menimbulkan *exception*, sedangkan kode pada blok *catch* adalah bagaimana kita akan menangani *exception* tersebut, yang terakhir blok *finally* adalah blok kode yang akan selalu dieksekusi baik terjadi *exception* maupun tidak. Untuk struktur penulisannya sebagai berikut:

```
try {
    /*
        * Kode yang kemungkinan akan
        * menimbulkan exception
        */
} catch ([TipeException] [variabelException]) {
    /*
        * Kode penanganan exception yang muncul.
        */
} catch (ArithmeticException aException) { // <- contoh tipe
    /*
        * Blok catch dapat lebih dari 1 (min. 1).
        */
} finally {
    /*
        * Kode yang akan selalu dieksekusi baik
        * muncul exception ataupun tidak.
        * Blok finally bersifat opsional.
        */
}
```

Contoh kasus-kasus exception lain:

a) Mengakses variabel/object yang bernilai null

```
public class App {
    public static void main(String[] args){
        String text = null;
        try {
            System.out.println(text.length());
        } catch (NullPointerException e) {
            System.err.println(e.getMessage());
        }
    }
```

```
}
```

Output program:

```
Cannot invoke "String.length()" because "text" is null
```

b) Mengakses index array lebih panjangnya

```
public class App {
    public static void main(String[] args){
        try {
            int[] value = new int[5];
            value[10] = 99;
        } catch (ArrayIndexOutOfBoundsException e) {
            System.err.println(e.getMessage());
        }
    }
}
```

Output:

```
Index 10 out of bounds for length 5
```

- **Custom Exception**

Selain menggunakan sebuah *exception* bawaan dari java, kita bisa juga membuat sebuah *custom exception* (kelas turunan *Exception*) yang bertujuan agar kode yang dibuat lebih mudah dibaca. Selain itu membuat *custom exception* bisa digunakan untuk menkategorikan atau memfilter saat nanti *exception* tersebut di-*catch*. Berikut contohnya:

```
public class InvalidRangeException extends Exception{
    InvalidRangeException(String message){
        super(message);
    }

    InvalidRangeException(String message, Throwable cause){
        super(message, cause);
    }
}
```

Contoh di atas adalah sebuah kode yang *exception* baru yang dibuat dengan nama **InvalidRangeException**, pembuatan *exception* harus mewarisi *class Exception*, dengan *constructor* berparameter *message* untuk pesan apa yang akan kita gunakan untuk disampaikan saat *exception* tersebut di-*catch*, sedangkan parameter **cause** untuk

penggunaan lebih lanjut, jika penasaran silahkan pelajari tentang **Throwable** dan **Exception Chaining** karena materi tersebut tidak termasuk PBO dasar.

- **Throw dan Throws**

  Setelah membuat *custom exception* di atas, kita bisa menggunakannya dengan keyword **throw** dan menandai suatu method yang melempar *exception* dengan keyword **throws**. Berikut contohnya:

```
public class App {
    public static void main(String[] args){
        int age = -1;
        try {
            validDateAge(age);
        } catch (InvalidRangeException e) {
            System.err.println(e.getMessage());
        }
    }

    public static boolean validDateAge(int age) throws InvalidRangeException {
        if (age < 0 || age > 150){
            throw new InvalidRangeException("Umur harus dalam rentang 0 sampai
150");
        }
        return true;
    }
}
```

  Perhatikan kode di atas, untuk melemparkan sebuah *exception* digunakan sebuah keyword **throw new**, sedangkan keyword **throws** sebelum kurung kurawal (**{**) digunakan untuk menandai bahwa method tersebut melemparkan sebuah *exception*.

CODELAB

Buatlah sebuah program sederhana untuk menginputkan sebuah nama-nama mahasiswa ke dalam sebuah *ArrayList.* Ketentuan sebagai berikut:

- *ArrayList* menggunakan tipe data String
- Ketika input kosong maka program akan **throw** sebuah **IllegalArgumentException** dengan parameter “Nama tidak boleh kosong”.
- Selagi user tidak input “selesai” maka input bisa terus berlanjut, input berhenti ketika user input “selesai”
- nama ke-{i} tidak bertambah jika input kosong
- Ketika *exception* di-*catch* tampilkan message error-nya
- Setelah itu outputkan semua data yang sudah ada di *ArrayList*

Contoh output program:

```
Masukkan nama ke-1: zaky
Masukkan nama ke-2: rahma
Masukkan nama ke-3: rama
Masukkan nama ke-4:
Nama tidak boleh kosong!
Masukkan nama ke-4:
Nama tidak boleh kosong!
Masukkan nama ke-4:
Nama tidak boleh kosong!
Masukkan nama ke-4: adit
Masukkan nama ke-5:
Nama tidak boleh kosong!
Masukkan nama ke-5:
Nama tidak boleh kosong!
Masukkan nama ke-5: selesai
Daftar mahasiswa yang diinputkan:
- zaky
- rahma
- rama
- adit
```

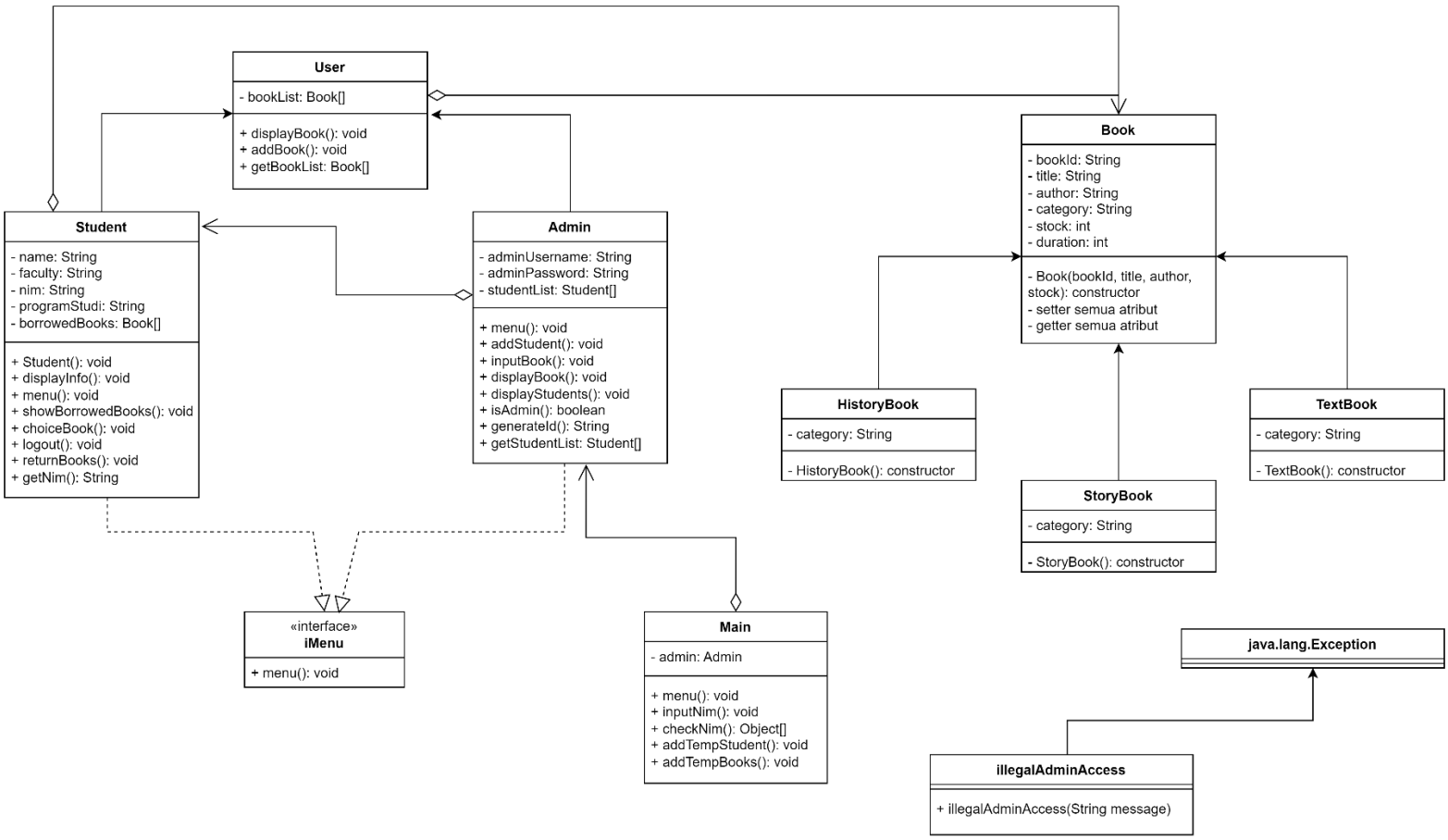
TUGAS

Melanjutkan pada tugas pada modul 4 sebelumnya, pada tugas modul 5 ini silahkan kembangkan program dengan ketentuan berikut:

- Ubah semua penyimpanan data yang sebelumnya menggunakan *Arrays* biasa menjadi *ArrayList* agar data dinamis
- Tambahkan sebuah blok *try/catch* di dalam class Admin di bagian method menu().
- Tambahkan sebuah blok *try/catch* di dalam class Student di bagian method menu().
- Buat package baru dengan nama **exception.custom** dan buat *custom exception* dengan nama **illegalAdminAccess** yang extends dari ***Exception***
- Pada class Admin di method isAdmin() silahkan buat **new throw illegalAdminAcces** dengan parameter “Invalid credentials”
- Tambahkan *exception* handling pada method menu() di class Main atau LibrarySystem dengan benar

Harap pahami dengan baik dan benar pada materi modul 5 ini, karena ketika demo tidak hanya ditanyakan mengenai kode yang kalian buat.

Diagram class tidak jauh beda, hanya menambahkan sebuah exception:



RUBRIK PENILAIAN

ASPEK PENILAIAN	POIN
Codelab	20
Tugas	30
Pemahaman	50
TOTAL	**100**

Selamat Mengerjakan

Tetap Semangat