VERSION 1.3

23 April 2025

PEMROGRAMAN BERORIENTASI OBJEK

MODUL 5 – ARRAYS, ARRAYLISTS, ITERATOR, EXCEPTIONS

DISUSUN OLEH:

WIRA YUDHA AJI PRATAMA

KEN ARYO BIMANTORO

DIAUDIT OLEH:

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

PRESENTED BY: TIM LAB. IT

UNIVERSITAS MUHAMMADIYAH MALANG

PENDAHULUAN

TUJUAN

1. Mahasiswa memahami konsep arrays, ArrayLists, iterator, dan exception handling dalam Java.
2. Mahasiswa memahami perbedaan antara arrays dan ArrayLists serta bagaimana menggunakannya secara efektif dalam program.
3. Mahasiswa memahami bagaimana menangani exception untuk meningkatkan ketahanan program terhadap error dan bug.

TARGET MODUL

1. Mahasiswa dapat membuat program yang menerapkan arrays, ArrayLists, iterator, dan exception handling dalam Java.
2. Mahasiswa dapat menggunakan iterator untuk melakukan iterasi pada koleksi data.
3. Mahasiswa dapat mengimplementasikan try-catch-finally dan throw/throws dalam penanganan exception.
4. Mahasiswa dapat mengelola kode menggunakan GitHub, termasuk commit, push, pull, serta memahami konsep branching dalam pengembangan perangkat lunak.

PERSIAPAN

1. Device (Laptop/PC)
2. IDE (IntelliJ)

KEYWORDS

Java, Arrays, ArrayLists, Iterator, Exception Handling, Try-Catch, Throw, Throws

TABLE OF CONTENTS

ARRAYS

TEORI

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.

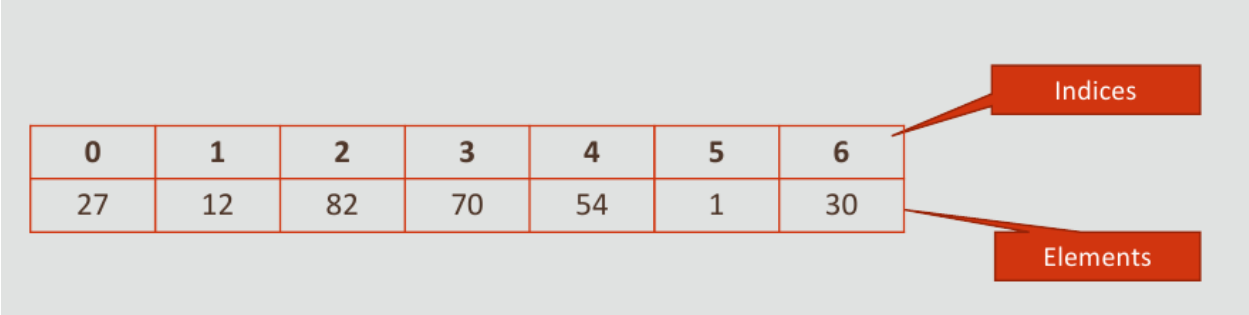
MATERI

Permasalahan ketika ingin menyimpan satu data yang sejenis seperti berikut:

```
String nama1 = "Titan";
String nama2 = "Rafi";
String nama3 = "Ega";
String nama4 = "Rahul";
```

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 mendeklarasikan sebuah *Arrays* kosong, terdapat 3 macam yaitu:

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

String nama[]; //cara kedua

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

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:

```java
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:

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

        nama[0] = "Ymir";
        nama[1] = "Reiner";
        nama[2] = "Bertholdt";
        nama[3] = "Zeke";
        nama[4] = "Eren";
    }
}
```

Atau bisa juga langsung seperti berikut:

```java
public class Main {
    public static void main(String[] args) {
        String nama[] = {"Ymir", "Reiner", "Bertholdt", "Zeke", "Eren"};
    }
}
```

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:

```java
String nama[] = {"Ymir", "Reiner", "Bertholdt", "Zeke", "Eren"};

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

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

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 hero[] = {"Dyroth", "Fanny", "Balmond", "Nana", "Miya"};

        hero[0] = "Layla";
        hero[1] = "Alucard";

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

Ketika program dijalankan, maka outputnya akan seperti berikut:

```
Layla
Alucard

Process finished with exit code 0
```

Ketika sebuah *Arrays* di deklarasi 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

Process finished with exit code 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.

```
public class Main {
    public static void main(String[] args) {
        String agents[] = {"Omen", "Breach", "Raze", "Reyna", "Cypher"};
        System.out.println("Panjang array: " + agents.length);
    }
}
```

Output program:

```
Panjang array: 5

Process finished with exit code 0
```

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[] = {"Lala", "Lili", "Lulu", "Lele", "Mubarok"};


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

Output program:

```
Lala
Lili
Lulu
Lele
Mubarok

Process finished with exit code 0
```

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

```java
public class Main {
    public static void main(String[] args) {
        String nama[] = {"Lala", "Lili", "Lulu", "Lele", "Khairul Umam"};


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

Dengan output program yang sama.

```
Lala
Lili
Lulu
Lele
Khairul Umam

Process finished with exit code 0
```

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:

```java
public class Main {
    public static void main(String[] args) {
        String[][] chraracters = {
            {"Luffy", "Straw Hat"},
            {"Zoro", "Pirate Hunter"},
            {"Shanks", "Red Hair"}
        };
    }
}
```

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

	0	1
0	"Luffy"	"Straw Hat"
1	"Zoro"	"Pirate Hunter"
2	"Shanks"	"Red Hair"

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

```java
System.out.println(characterss[0][0]);
```

Arti dari kode di atas adalah mengakses isi array pada variabel characters pada indeks ke-0 yang di dalamnya akses indeks ke-0, maka akan output yaitu **Luffy**. 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[][] chraracters = {
            {"Luffy", "Straw Hat"},
            {"Zoro", "Pirate Hunter"},
            {"Shanks", "Red Hair"}
        };

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

Output program:

```
Nama : Luffy
A.K.A : Straw Hat
Nama : Zoro
A.K.A : Pirate Hunter
Nama : Shanks
A.K.A : Red Hair

Process finished with exit code 0
```

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.

ARRAYLIST

TEORI

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.

MATERI

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:

```
public class Main {
    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*:

```java
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("Wira");
        penampung.add(2004);
        penampung.add("Informatika");
        penampung.add(50.55);
        penampung.add(false);

        // 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:

```
[Wira, 2004, Informatika, 50.55, false]
[Wira, Informatika, 50.55, false]
Informatika
Array berisi : 4

Process finished with exit code 0
```

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

```
// 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 menspesifikkan 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("Nisrina");
    names.add("Maharani");
    names.add("Amelia");
}
```

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

```
1   import java.util.ArrayList;
2
    no usages
3   public class Main {
4       static void main(String[] args) throws Exception {
5           ArrayList<String> names = new ArrayList<>();
6
7           names.add("Nisrina");
8           names.add(2004);
9       }
10  }
```

```
© Main.java C:\Users\WIRA YUDHA\Desktop\folder wira\project\InteliJ project\test\src 5 problems
  'add(java.lang.String)' in 'java.util.ArrayList' cannot be applied to '(int)' :8
```

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 *Array* 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 dimensi 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("Wira Yudha Aji Pratama");
    temp.add("202310370311010");

    // 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));
```

```
Nama : Wira Yudha Aji Pratama
NIM : 202310370311010

Process finished with exit code 0
```

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

ITERATOR

TEORI

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.

MATERI

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("Ega");
    names.add("Titan");
    names.add("Zam");
    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("Ken");
        names.add("Wira");
        names.add("Dika");

        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:

```
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:

```
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);
        }
    }
}
```

EXCEPTIONS

TEORI

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.*

MATERI

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:

```java
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:

```java
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:

```java
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

Process finished with exit code 0
```

b. Mengakses index array lebih panjangnya

```java
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 program:

```
Index 10 out of bounds for length 5

Process finished with exit code 0
```

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 **mengkategorikan** atau **memfilter** saat nanti *exception* tersebut **di-*catch***. Berikut contohnya:

```java
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*.

PRAKTEK

Sekarang, kita akan mencoba langsung bagaimana menggunakan **Arrays**, **ArrayLists**, **Iterator**, dan menangani **Exception** dasar dalam Java. Mari kita lihat bagaimana kita bisa mengelola sekumpulan data dan membuat program kita lebih bagus terhadap **error**.

Langkah 1: Mengenal Arrays (Koleksi Ukuran Tetap)

Arrays berguna saat kita tahu persis berapa banyak elemen yang ingin kita simpan dan ukurannya tidak akan berubah.

- **Membuat dan Mengisi Array:**

Buatlah sebuah proyek Java baru atau gunakan kelas Main yang sudah ada. Coba kode berikut ini untuk membuat array string yang menyimpan beberapa nama hari:

```
// Di dalam method main
String[] namaHari = new String[3]; // Membuat array String dengan ukuran 3
namaHari[0] = "Senin";
namaHari[1] = "Selasa";
namaHari[2] = "Rabu";

// Cara lain langsung inisialisasi
int[] angkaKeberuntungan = {7, 11, 21, 3};
```

- Pada kode di atas, **namaHari** adalah array yang bisa menampung 3 string. **angkaKeberuntungan** adalah array integer yang langsung diisi nilainya.
- **Mengakses Elemen Array dan Panjangnya:**

  Kita bisa mengakses elemen menggunakan indeks (dimulai dari 0) dan mengetahui panjang array menggunakan properti .length.

```
System.out.println("Hari pertama: " + namaHari[0]);
System.out.println("Hari terakhir: " + namaHari[namaHari.length - 1]);
System.out.println("Panjang array angkaKeberuntungan: " + angkaKeberuntungan.length);
```

- **Iterasi (Looping) pada Array:**

  Untuk menampilkan semua elemen, kita bisa menggunakan loop:

```
System.out.println("\nNama-nama Hari:");
for (int i = 0; i < namaHari.length; i++) {
    System.out.println("Indeks ke-" + i + ": " + namaHari[i]);
}

System.out.println("\nAngka Keberuntungan (for-each):");
for (int angka : angkaKeberuntungan) {
    System.out.println(angka);
}
```

Langkah 2: Menggunakan ArrayLists (Koleksi Dinamis)

ArrayLists lebih fleksibel karena ukurannya bisa bertambah atau berkurang secara dinamis.

- **Membuat dan Menambah Elemen ke ArrayList:**

  Jangan lupa **import java.util.ArrayList;** di awal file.

```
// Di dalam method main, setelah kode array
ArrayList<String> daftarBelanja = new ArrayList<>(); // Membuat ArrayList String
daftarBelanja.add("Apel");
daftarBelanja.add("Susu");
daftarBelanja.add("Roti");
daftarBelanja.add(1, "Kopi"); // Menambahkan "Kopi" di indeks ke-1

System.out.println("\nDaftar Belanja Awal: " + daftarBelanja);
```

- **Mengakses, Mengubah, dan Menghapus Elemen:**

```
System.out.println("Elemen pertama: " + daftarBelanja.get(0));
daftarBelanja.set(0, "Apel Merah"); // Mengubah elemen di indeks 0
System.out.println("Setelah diubah: " + daftarBelanja);

daftarBelanja.remove(2); // Menghapus elemen di indeks 2 ("Susu")
System.out.println("Setelah 'Susu' dihapus: " + daftarBelanja);
System.out.println("Ukuran ArrayList sekarang: " + daftarBelanja.size());
```

Langkah 3: Iterasi pada ArrayList Menggunakan Iterator

Iterator menyediakan cara standar untuk menelusuri elemen dalam sebuah koleksi.

Jangan lupa **import java.util.Iterator;**

```
// Di dalam method main, setelah kode ArrayList
System.out.println("\nMenampilkan Daftar Belanja dengan Iterator:");
Iterator<String> it = daftarBelanja.iterator();
while (it.hasNext()) { // Cek apakah masih ada elemen berikutnya
    String item = it.next(); // Ambil elemen berikutnya
    System.out.println("- " + item);
}
```

Menggunakan ***enhanced for-loop*** juga sangat umum dan lebih ringkas untuk iterasi ArrayList:

```
System.out.println("\nMenampilkan Daftar Belanja dengan for-each:");
for(String item : daftarBelanja) {
    System.out.println("* " + item);
}
```

Langkah 4: Dasar Exception Handling (try-catch)

Exception handling membantu program kita agar tidak langsung berhenti (crash) saat terjadi error.

- **Menangani InputMismatchException:**

  Saat meminta input angka dari pengguna, bisa saja pengguna memasukkan teks. Jangan lupa **import java.util.Scanner;** dan **import java.util.InputMismatchException;**

  ```
  // Di dalam method main
  Scanner inputScanner = new Scanner(System.in);
  System.out.print("\nMasukkan sebuah angka: ");
  try {
      int angkaInput = inputScanner.nextInt();
      System.out.println("Angka yang dimasukkan: " + angkaInput);
  } catch (InputMismatchException e) {
      System.out.println("Error: Input yang dimasukkan bukan angka yang valid!");
  }
  ```

- **Menangani IndexOutOfBoundsException:**

  Terjadi jika kita mencoba mengakses indeks **array** atau **ArrayList** yang tidak ada.

```
System.out.println("\nMencoba akses indeks di luar batas ArrayList (daftarBelanja):");
try {
    System.out.println("Mencoba ambil elemen ke-10: " + daftarBelanja.get(10));
} catch (IndexOutOfBoundsException e) {
    System.out.println("Error: Indeks yang diakses di luar batas ukuran ArrayList!");
    // System.out.println("Detail error: " + e.getMessage());
} finally {
    System.out.println("Blok finally selalu dieksekusi, baik ada error maupun tidak.");
    inputScanner.close(); // Tutup scanner di akhir
}
```

- Blok **finally** akan selalu dieksekusi, berguna untuk melepaskan resource seperti menutup scanner atau file.

Kesimpulan Praktek:

Dari praktek ini, kita bisa melihat:

- **Arrays** cocok untuk data yang ukurannya sudah pasti.
- **ArrayLists** memberikan fleksibilitas dengan ukuran dinamis dan method yang kaya.
- **Iterator** (dan for-each loop) adalah cara yang efektif untuk menelusuri elemen koleksi.
- **Exception Handling** (**try-catch-finally**) sangat penting untuk membuat program yang lebih stabil dan tidak mudah berhenti saat menghadapi situasi tak terduga.

Silakan coba jalankan kode-kode di atas, modifikasi nilainya, dan lihat bagaimana setiap bagian bekerja! Eksplorasi lebih lanjut method lain dari **ArrayList** atau jenis exception lain yang mungkin terjadi.

CODELAB & TUGAS

CODELAB

Buatlah program sederhana untuk mengelola daftar stok barang. Program ini harus memungkinkan penambahan **barang baru**, **penampilan semua barang**, dan **pengurangan stok barang** yang ada. Program juga harus mampu menangani input yang salah dan kondisi stok tidak mencukupi.

Langkah-langkah Implementasi:

1. **Buat Kelas Barang:**
   - Buat sebuah kelas **Barang**.
   - Tambahkan atribut **private String** nama dan **private int stok**.
   - Buat constructor untuk menginisialisasi nama dan stok awal.
   - Buat method getter (**getNama**(), **getStok**()) dan setter (**setStok**()).
2. **Buat Custom Exception StokTidakCukupException:**
   - Buat kelas exception baru bernama **StokTidakCukupException**.
   - Pastikan kelas ini mewarisi (extends) dari kelas **Exception**.
   - Buat constructor yang menerima pesan error (String) dan meneruskannya ke constructor superclass (**super(message)**).
3. **Implementasi Logika Utama di main:**
   - Buat kelas utama (misalnya **ManajemenStok**).
   - Di dalam method main:
     - Buat sebuah **ArrayList** untuk menyimpan objek Barang: **ArrayList<Barang> daftarBarang = new ArrayList<>();.**
     - Tambahkan beberapa objek Barang sebagai data awal ke dalam **daftarBarang** untuk pengujian.
     - Buat instance Scanner untuk menerima input.

- Buat loop menu utama (misalnya menggunakan **while** dan **switch** atau **if-else**) yang menampilkan opsi berikut:
  - Tambah Barang Baru
  - Tampilkan Semua Barang
  - Kurangi Stok Barang
  - Keluar
- **Implementasi Opsi 1 (Tambah Barang):**
  - Minta input nama barang (String) dan stok awal (int).
  - Gunakan blok **try-catch** untuk menangani **java.util.InputMismatchException** saat membaca input stok. Jika terjadi, tampilkan pesan error ("Input stok harus berupa angka!").
  - Jika input valid, buat objek **Barang** baru dan tambahkan ke **daftarBarang**.
- **Implementasi Opsi 2 (Tampilkan Semua Barang):**
  - Gunakan **Iterator** atau **enhanced for-loop** untuk menelusuri **daftarBarang**.
  - Untuk setiap Barang, tampilkan nama dan stoknya. Jika **daftarBarang** kosong, tampilkan pesan "Stok barang kosong.".
- **Implementasi Opsi 3 (Kurangi Stok):**
  - Tampilkan daftar barang yang ada beserta nomor indeksnya (misal, mulai dari 0).
  - Minta input nomor indeks barang yang stoknya akan dikurangi.
  - Minta input jumlah stok yang akan diambil.
  - Gunakan blok **try-catch** (bisa bersarang atau multiple catch) untuk menangani:
  - **InputMismatchException**: Jika input indeks atau jumlah bukan angka. Tampilkan pesan error.

- **IndexOutOfBoundsException**: Jika indeks yang dimasukkan tidak valid untuk **daftarBarang**. Tampilkan pesan error.
- **StokTidakCukupException**: Jika jumlah yang diminta melebihi stok yang ada (dijelaskan di bawah). Tampilkan pesan dari exception.
- Di dalam blok **try** utama untuk opsi ini:
- Ambil objek **Barang** dari **daftarBarang** berdasarkan indeks yang valid.
- Lakukan validasi: Jika jumlahDiambil > barang.getStok(), **throw new StokTidakCukupException("Stok untuk " + barang.getNama() + " hanya tersisa " + barang.getStok())**.
- Jika validasi berhasil (stok cukup), kurangi stok barang tersebut: **barang.setStok(barang.getStok() - jumlahDiambil)**. Tampilkan pesan konfirmasi pengurangan stok berhasil.

- **Implementasi Opsi 0 (Keluar):**
  - Akhiri loop menu utama. Tampilkan pesan "Terima kasih!".
- Pastikan untuk menutup objek Scanner (**scanner.close();**) setelah loop selesai untuk menghindari resource leak.

Contoh Output yang Diharapkan:

```
===== Menu Manajemen Stok =====
1. Tambah Barang Baru
2. Tampilkan Semua Barang
3. Kurangi Stok Barang
0. Keluar
Pilih opsi: 1
Masukkan nama barang: Pensil
Masukkan stok awal: 50
Barang 'Pensil' berhasil ditambahkan.

===== Menu Manajemen Stok =====
1. Tambah Barang Baru
2. Tampilkan Semua Barang
3. Kurangi Stok Barang
0. Keluar
Pilih opsi: 2

--- Daftar Barang ---
0. Nama: Pensil, Stok: 50
--------------------

===== Menu Manajemen Stok =====
1. Tambah Barang Baru
2. Tampilkan Semua Barang
3. Kurangi Stok Barang
0. Keluar
Pilih opsi: 3

--- Daftar Barang (Pilih untuk Kurangi Stok) ---
0. Pensil (Stok: 50)
Masukkan nomor indeks barang: 0
Masukkan jumlah stok yang akan diambil: 10
Stok barang 'Pensil' berhasil dikurangi. Sisa stok: 40

===== Menu Manajemen Stok =====
1. Tambah Barang Baru
2. Tampilkan Semua Barang
3. Kurangi Stok Barang
0. Keluar
Pilih opsi: 0
Terima kasih! Program berakhir.
```

TUGAS

Lanjutkan pengembangan program sistem login dan pelaporan dari Modul 4. Pada tugas kali ini, Kita akan mengimplementasikan penyimpanan data menggunakan **ArrayList**, melakukan iterasi data, dan menangani potensi error menggunakan **Exception Handling** untuk membuat program lebih dinamis.

Kita akan mengganti logika placeholder kemarin dengan penyimpanan data riil menggunakan **ArrayList**, mengimplementasikan fitur inti (**lapor**, **lihat**, **kelola data**), dan meningkatkan **robustitas** program dengan menangani error input dan data.

Struktur Program & Modifikasi:

1. **Persiapan Penyimpanan Data:**
    - **Buat Kelas Item:**
        - Di package yang sesuai (misal **com.praktikum.data** atau **com.praktikum.models**), buat kelas baru bernama **Item**.
        - Tambahkan atribut-atribut yang relevan untuk menyimpan **detail barang**, misalnya: **String itemName**, **String description**, **String location**, **String status** (misal: "Reported", "Claimed"). Gunakan modifier **private**.
        - Buat constructor untuk menginisialisasi objek **Item**.
        - Buat method **getter** (dan **setter** jika diperlukan) untuk semua atribut.
    - **Buat Penyimpanan Pusat (ArrayList):**
        - Di kelas **LoginSystem** (atau kelas lain yang berperan sebagai pusat data jika Kalian membuatnya), deklarasikan dua **ArrayList *static*** untuk menyimpan data secara global selama program berjalan:

```
// Di dalam kelas LoginSystem (misalnya)
static ArrayList<User> userList = new ArrayList<>();
static ArrayList<Item> reportedItems = new ArrayList<>();
```

        - Inisialisasi **userList**: Hapus logika login yang membandingkan dengan nilai hardcoded. Sebagai gantinya, tambahkan beberapa objek **Admin** dan **Mahasiswa** secara ***default*** ke dalam **userList** saat program dimulai

2. **Update Logika Login:**
    - Modifikasi method **login()** di kelas **Admin** dan **Mahasiswa** (atau method verifikasi pusat di **LoginSystem**) agar melakukan pencarian di dalam **userList**.
    - Gunakan **loop** (for-each atau for loop biasa) untuk memeriksa setiap **User** dalam **userList**.
    - Gunakan **instanceof** untuk membedakan antara **Admin** dan **Mahasiswa** saat memeriksa kredensial yang sesuai (username/password untuk Admin, nama/NIM untuk Mahasiswa).
    - Jika user ditemukan dan kredensial cocok, kembalikan objek **User** tersebut. Jika tidak ditemukan setelah memeriksa seluruh list, kembalikan **null** atau (jika Kalian ingin menerapkan **custom exception**) throw sebuah **exception**.

3. **Implementasi Fitur Inti (Menggunakan ArrayList):**

    - **Mahasiswa.reportItem():**
        - Setelah mendapatkan input detail barang dari **Scanner**, buat objek **Item** baru.
        - Set status item menjadi "**Reported**".
        - Tambahkan objek **Item** ini ke **ArrayList reportedItems** yang static.
        - Tampilkan pesan konfirmasi.

    - **Mahasiswa.viewReportedItems():**
        - Hapus pesan placeholder.
        - Cek apakah **reportedItems** kosong. Jika ya, tampilkan pesan "**Belum ada laporan barang.**".
        - Jika tidak kosong, gunakan **Iterator** atau **enhanced for-loop** untuk mengiterasi **reportedItems**.
        - Untuk setiap **Item** dalam list, tampilkan detailnya (nama, deskripsi, lokasi) ke konsol. *Hanya tampilkan item yang statusnya "**Reported**"*.

    - **Admin.manageItems():**
        - Hapus pesan placeholder.
        - Implementasikan dua sub-menu:
            - **1. Lihat Semua Laporan:** Iterasi **reportedItems** (gunakan Iterator/enhanced for-loop), tampilkan detail ***semua*** item beserta statusnya ("Reported" / "Claimed").
            - **2. Tandai Barang Telah Diambil (Claimed):**

        - Tampilkan dulu semua item yang statusnya "**Reported**" beserta nomor indeksnya (mulai dari 0 atau 1).
        - Minta **Admin** memasukkan nomor indeks barang yang ingin ditandai.
        - Gunakan **Exception Handling** (lihat langkah 4) saat mengambil input dan mengakses list.
        - Jika indeks **valid** dan item **ada**, ubah status Item pada indeks tersebut di **reportedItems** menjadi "**Claimed**". Tampilkan pesan sukses.

    - **Admin.manageUsers():**
        - Hapus pesan placeholder.
        - Implementasikan dua sub-menu dasar:
            - **1. Tambah Mahasiswa:** Minta input **Nama** dan **NIM**, buat objek **Mahasiswa** baru, tambahkan ke **userList**.
            - **2. Hapus Mahasiswa:** Minta input **NIM Mahasiswa** yang akan dihapus. Cari Mahasiswa dengan NIM tersebut di **userList** (gunakan Iterator/loop). Jika ditemukan, hapus dari list. Tampilkan pesan sukses/gagal.

4. **Implementasi Exception Handling:**

    - **Input Angka**: Di semua tempat yang meminta input angka dari **Scanner** (misal: pilihan menu, indeks barang), gunakan **try-catch** block untuk menangani **java.util.InputMismatchException**. Jika terjadi **exception**, tampilkan pesan error yang jelas **("Input harus berupa angka!")** dan minta pengguna mencoba lagi atau kembali ke menu.

    - **Akses Indeks:** Dalam fitur "**Tandai Barang Telah Diambil**" di **Admin.manageItems(),** setelah mendapatkan input indeks dari **Admin**, gunakan **try-catch** block saat mengakses **reportedItems.get(index)** atau saat mengubah status **reportedItems.set(index, ...)** untuk menangani jenis error **IndexOutOfBoundsException**. Beri pesan jika indeks yang dimasukkan **tidak valid**.

Cara Kerja Program (Setelah Modifikasi):

- Program dimulai, **userList** diinisialisasi dengan data **default**.
- Login dilakukan dengan memvalidasi input terhadap data di **userList**.
- Jika login berhasil, menu yang sesuai ditampilkan.
- Mahasiswa bisa melaporkan item (objek **Item** baru ditambahkan ke **reportedItems**) dan

melihat daftar item yang statusnya "**Reported**".

- **Admin** bisa melihat semua item (beserta status), menandai item sebagai "Claimed" (mengubah status objek **Item** di **reportedItems**), menambah objek **Mahasiswa** baru ke **userList**, atau menghapus objek **Mahasiswa** dari **userList**.
- Program lebih tahan terhadap input yang salah (angka vs teks) dan akses indeks yang tidak valid berkat **try-catch**.

Petunjuk Implementasi:

- Fokus pada penggunaan **ArrayList** untuk menyimpan objek **User** dan **Item**.
- Ganti semua logika placeholder dengan interaksi nyata ke **ArrayList** tersebut.
- Gunakan **Iterator** atau **enhanced for-loop** untuk membaca data dari **ArrayList**.
- Terapkan **Exception Handling (try-catch)** minimal untuk **InputMismatchException** dan **IndexOutOfBoundsException** pada skenario yang relevan.
- Pastikan kelas **Item** dibuat dengan benar dan ditempatkan di package yang sesuai.
- Perbarui logika login untuk menggunakan **userList**.

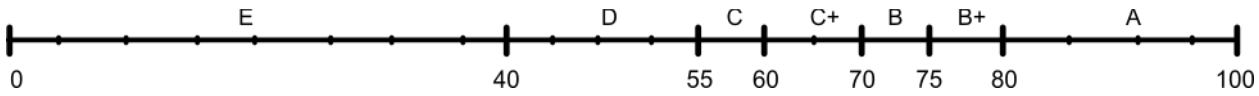
PENILAIAN

RUBRIK PENILAIAN

Aspek Penilaian	Poin
CODELAB	**Total 25%**
Kerapian kode	5%
Ketepatan kode & output	10%
Kekreativitasan kode	5%
Kode orisinal (tidak nyontek)	5%
TUGAS	**Total 75%**
Kerapian kode	5%
Ketepatan kode & output	20%

Kode orisinal (tidak nyontek)	5%
Kemampuan menjelaskan	20%
Menjawab pertanyaan	25%
TOTAL	**100%**

SKALA PENILAIAN



A = (81 - 100) → Sepuh

B+ = (75 - 80) → Sangat baik

B = (70 - 74) → Baik

C+ = (60 - 69) → Cukup baik

C = (55 - 59) → Cukup

D = (41 - 54) → Kurang

E = (0 - 40) → Bro really...

SUMMARY AKHIR MODUL

Gimana Modul 5 nya? Mulai kerasa kan program yang kalian bangun dari awal sekarang jadi beneran bisa "hidup"? Di modul ini, kita udah nggak cuma ngomongin struktur dan konsep PBO aja, tapi udah mulai *action* dengan data pakai **Arrays** dan **ArrayLists**, menelusurinya pakai Iterator, plus bikin program kita jadi lebih tahan dari error pakai **Exception Handling**.

Mengelola kumpulan data dan mengantisipasi berbagai kemungkinan error itu makanan sehari-hari programmer, lho. Mungkin awalnya agak ribet ya, mikirin kapan pakai **ArrayList**, gimana cara loop yang bener, atau **try-catch** error apa aja. Tapi, percayalah, semakin sering kalian coba dan utak-atik, insting kalian bakal makin tajam. Sekarang, program kalian nggak cuma punya struktur OOP yang bagus, tapi juga udah bisa nyimpen data beneran dan nggak langsung *crash* kalau ada input aneh atau kondisi tak terduga.

Ingat, setiap baris kode yang kalian tulis, setiap *bug* yang berhasil kalian selesaikan, itu semua nambah jam terbang. **Jangan takut salah**, justru dari situ pemahaman kalian tentang **ArrayList** yang dinamis atau kapan harus **throw new Exception()** jadi makin mantap. Kalau masih ada yang bikin pusing, diskusikan, cari referensi, dan yang paling penting, praktikkan lagi!

Nah, setelah program kita punya **"otak"** (logika OOP dari Modul 1-4) dan **"ingatan"** (penyimpanan data dari Modul 5), di modul terakhir (Modul 6) nanti, kita bakal kasih program kita **"wajah"** yang menarik pakai GUI (JavaFX). Kebayang kan, aplikasi yang kalian rancang ini bakal bisa diklik-klik dan punya tampilan visual! Jadi, **pastikan fondasi di Modul 5 ini kokoh ya. Semangat terus, tinggal selangkah lagi menuju aplikasi yang lebih interaktif!**