UNIVERSITAS INDONESIA

FAKULTAS ILMU KOMPUTER

Dasar Dasar Pemrograman 2

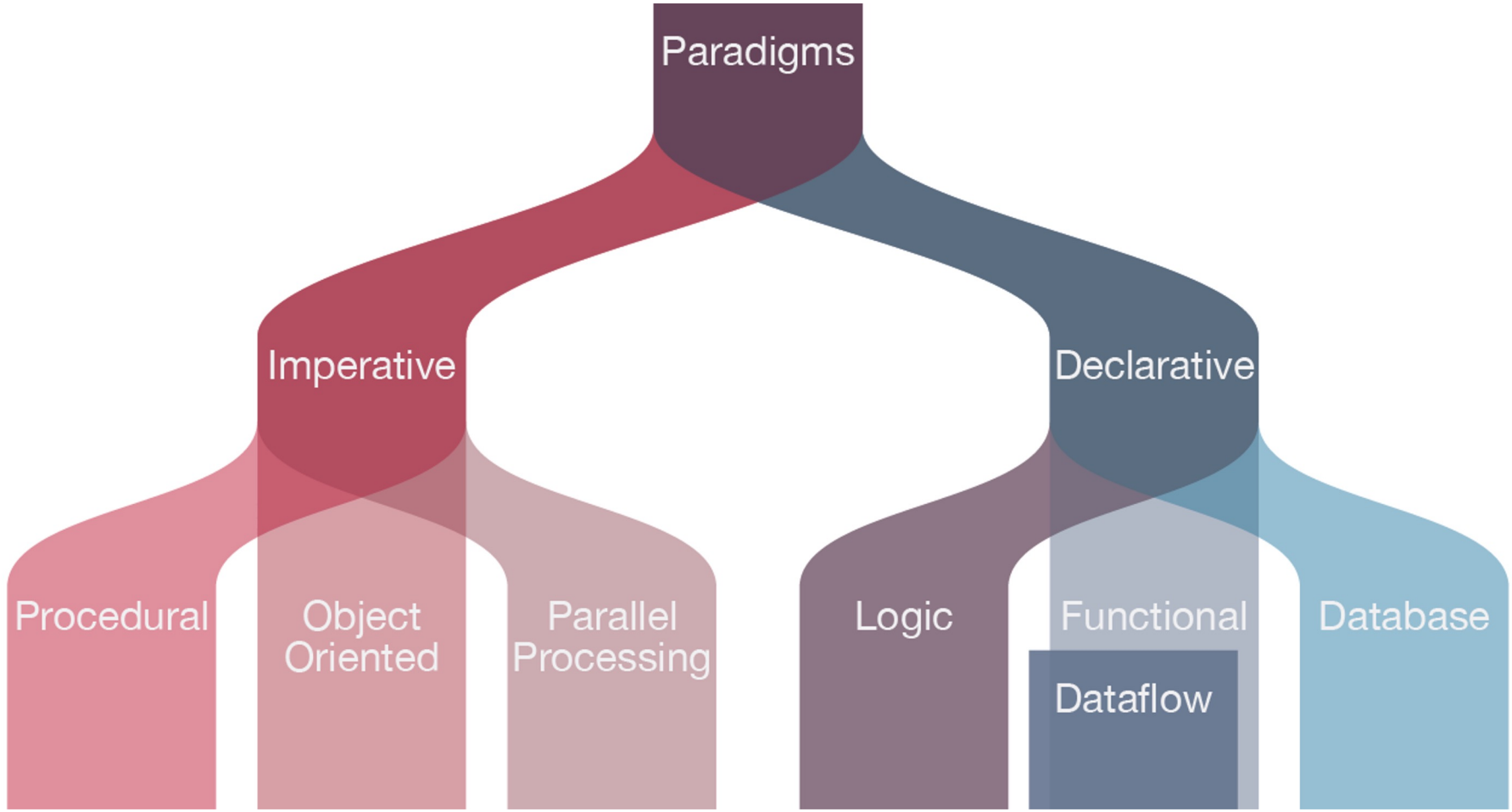
Acuan: Introduction to Java Programming and Data Structure, Bab 19 dan Oracle Java Tutorial on Generics

Sumber Slide: Liang,

Dimodifikasi untuk Fasilkom UI oleh Ade Azurat

Topik 13: Null Safety and Lambda

Programming Paradigma

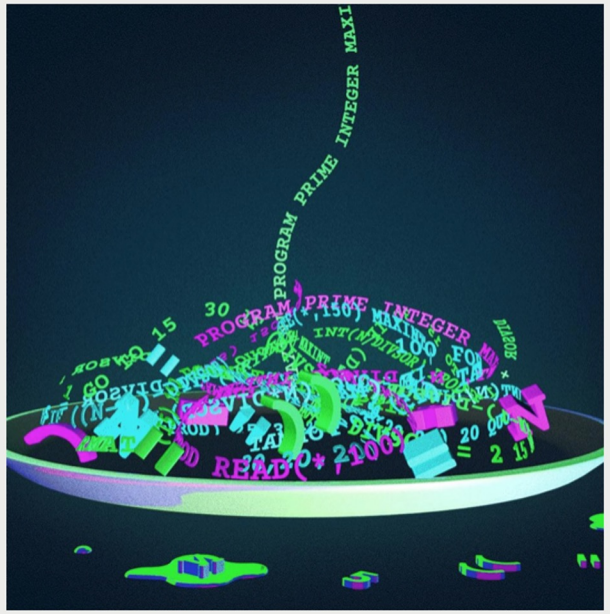


IEEE Spectrum

Type to search

https://spectrum.ieee.org/functional-programming

Articles by Charles Scalfani



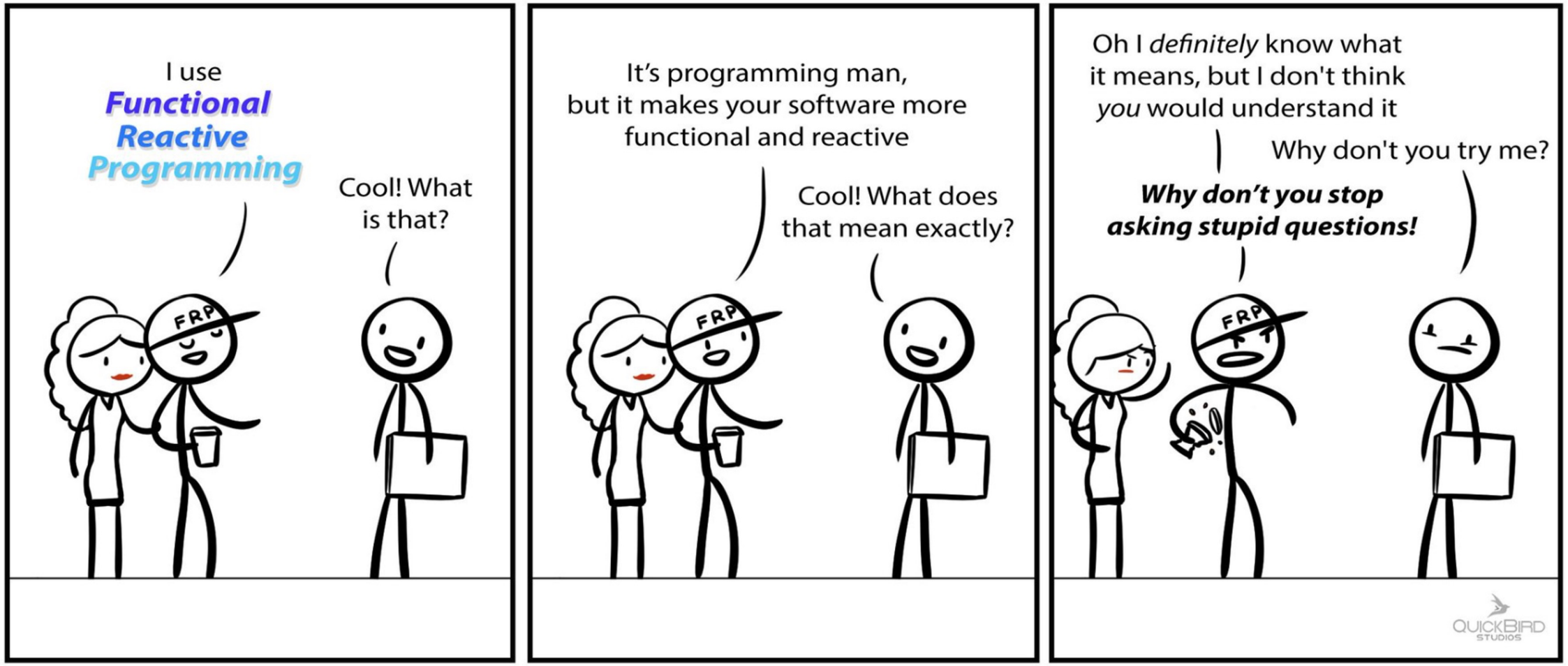
FEATURE | COMPUTING

Why Functional Programming Should Be the Future of Software Development

› It's hard to learn, but your code will produce fewer nasty surprises

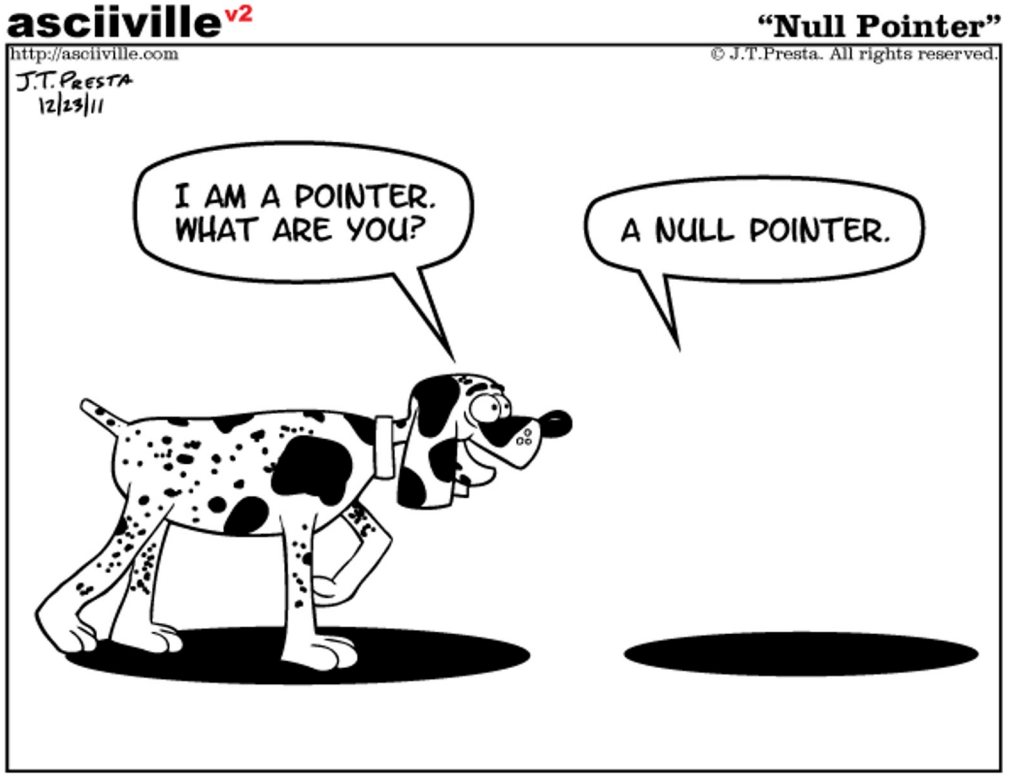
23 OCT 2022 | 11 MIN READ |

Is it really hard to learn?



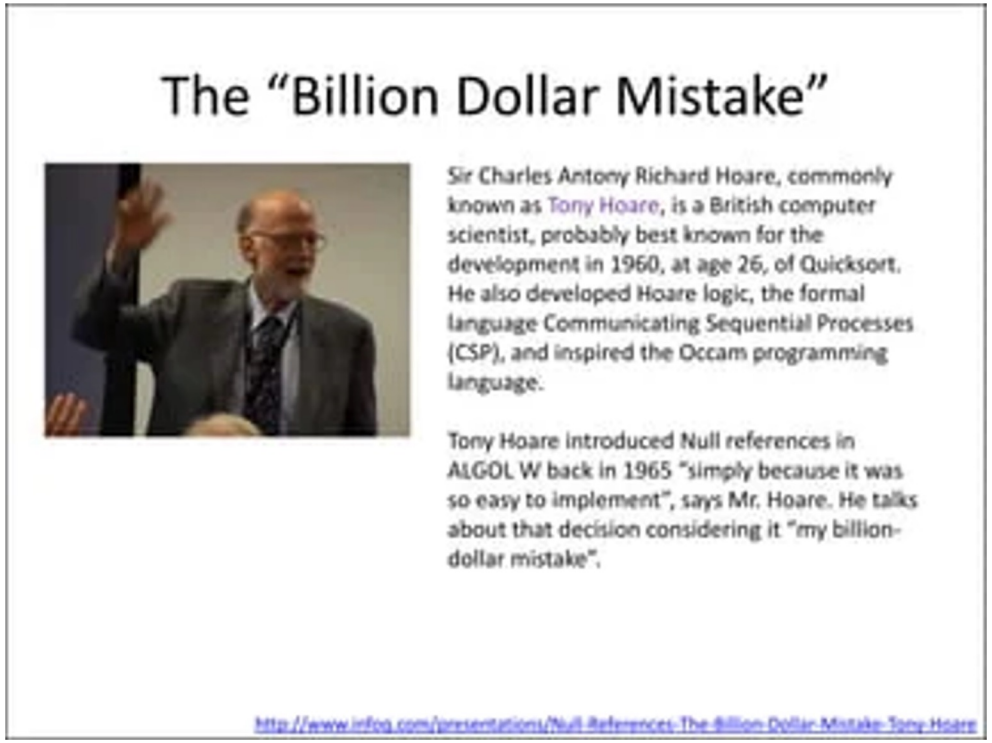
Null reference

- Null reference is a reference to a place in memory points to nothing at all.
- If you try to use this reference, an error ensues.
- So programmers have to remember to check whether something is null before trying to read or change what it references.



Null reference

- Nearly every popular language today has this flaw. A pioneering computer scientist Tony Hoare introduced null references in the ALGOL language back in 1965, and it was later incorporated into numerous other languages.
- Hoare explained that he did this “simply because it was so easy to implement,” but lately he considers it to be a “billion-dollar mistake.”
- That’s because it has caused countless bugs when a reference that the programmer expects to be valid is really a null reference.



What is Null Safety?

- Some programming language already follow Hoare's design on Algol. So will also will find null in Java.
- But people already realize the possible danger of null.
- Handling null can be tricky,
- They are not only hard to identify but also complex to deal with.
- Any missing check of null on compile time may lead to a runtime error of NullPointerException
- So, modern professionals are trying to avoid it.
- On way to avoid it is using Optional<T>

Optional<T>

- Since Java 8, the class Optional has been introduced
- Pada class Optional disediakan method orElse yang akan memberikan default value yang diberikan bila object nya ternyata memang null.
- Hal ini mengurangi kompleksitas pemeriksaan null dengan if
- Lihat berkas CobaOptional.Java

```
import java.util.Optional;

public class CobaOptional {
   public static void main(String[] args){
                Pesan pesan = null;

                // Without Optional
                if (pesan != null
                   && pesan.getPesan() != null) {
                                System.out.println(
                                "Pesan tanpa Optional: " + pesan.getPesan());
                } else {
                                System.out.println(
                                "Pesan tanpa Optional:Pesan default");
                }

                // With Optional
                Optional<Pesan> optionalPesan =   Optional.ofNullable(pesan);
                System.out.println("Pesan dengan Optional: " +
                optionalPesan.orElse(new Pesan("Pesan default")));
   }
}

class Pesan {
                private String pesan;

                public Pesan(String pesan) {
                                this.pesan = pesan;
                }

                public String getPesan() {
                                return pesan;
                }

                public String toString() {
                                return pesan;
                }
```

Optional<T> orElse

- Object yang mungkin null di-”bungkus” dengan object Optional
- Pada kode tersebut Optional<Pesan>
- Pada method orElse, akan memberikan value dari pesan atau memberikan default value yang disedikan pada parameter orElse.
- Pada kode tersebut default nya adalah “Pesan default”

- Penggunaan Optional ini menghindari kemungkinan terjadi Runtime dan mengurangi kompleksitas dengan if-else

Lambda Function

- Konsep lain yang ingin diperkenalkan adalah Lambda Function (diliteratur lebih disebut dengan lambda expression).
- Secara sederhana lambda function adalah fungsi yang tidak memiliki nama. Fungsi yang bisa dibuat sesuai kebutuhan.
- Seperti method yang mengembalikan value (bukan void).
- Lambda di java bisa disimpan dalam variable dengan tipe Consumer<T>

```
import java.util.function.Consumer;

public class CobaLambda {
    public static void main(String[] args) {
            // A simple lambda expression example with a parameter
            Consumer<String> greet = name -> System.out.println("Hello, " + name + "!");
            greet.accept("Lambda");
    }
}
```

Lambda dan Functional Interface

- Selain menyimpan pada variable dalam tipe Consumer<T>
- Kita dapat juga menggunakan Lambda dengan Functional Interface
- Functional Interface adalah sebagaimana interface yang kita pelajari sebelumnya, hanya men-deklarasikan tipe dengan method, tapi tidak berisi implementasi.
- Functional Interface dibatasi hanya memiliki sebuah abstract method saja.
- (notes: Interface dapat memiliki default implementation, begitu juga functional interface, namun kita tidak membahasnya di kuliah DDP2)

Functional Interface

```
@FunctionalInterface
interface StringFungsi {
  String fungsi(String str);
}

public class ContohFunctionalInterface {

 public static void main(String[] args) {

    StringFungsi seru = (s) -> s + "!";
    StringFungsi tanya = (s) -> s + "?";

    System.out.println(seru.fungsi("Hello"));
    System.out.println(tanya.fungsi("Hello"));
  }
}
```

Stream

- Stream adalah sekumpulan (sequence) element atau object atau kumpulan data yang bisa diproses baik secara sequential ataupun parallel.
- Setiap Collection, misalnya ArrayList adalah sebuah Collection, bisa diambil representation Stream nya dengan method Collection.stream()
- Dengan menggunakan Stream kita dapat mengolah kumpulan data dengan lebih elegant.

Stream, forEach

```
import java.util.Arrays;
import java.util.List;

public class CobaStream {
        public static void main(String[] args) {
                List<String> names = Arrays.asList("Alice", "Bob", "Charlie",
                                                   "David", "Eve");

                names.stream()
                    .forEach(datum -> System.out.println(datum));

                names.stream()
                    .forEach(System.out::println); // point free style
        }
}
```

Stream, forEach

- Dengan stream kita bisa mengolah setiap elemen pada koleksi didalam stream tersebut menggunakan perintah

  forEach

- forEach menerima sebuah lambda function dengan satu parameter.
- Setiap elemen pada koleksi kita akan menjadi parameter pada lambda function tersebut. Akan diproses satu persatu.

Point free style

- Belakangan ini dikalangan programmer javascript muncul terminology Point-free style.
- Didalam literatur Point-free style dikenal juga dengan istilah Tacit-Programming
- Istilah ini digunakan untuk programming yang tidak perlu secara eksplisit menuliskan nama parameter-nya.
- Hal ini punya manfaat untuk mengurangi beban programmer ketika membaca program. Programmer tidak perlu membaca dan men-tracking nama variable yang memang tidak terlalu relevan.

Stream, filter

```
import java.util.Arrays;
import java.util.List;

public class CobaStream {
        public static void main(String[] args) {
                List<String> names = Arrays.asList("Alice", "Bob", "Charlie",
                                                   "David", "Eve");

                names.stream()
                    .filter(x -> x.length()<4)
                    .forEach(System.out::println); // point free style
        }
}
```

Stream, filter

- Filter adalah sebuah standard fungsi dalam functional programming.
- Filter dalam java stream, filter akan menerima sebuah lambda function yang me-return Boolean (true atau false).
- Kemudian akan menyisakan hanya data yang memenuhi kriteria filter yang diberikan.
- Setelah di filter, data pada koleksi hanya akan berisi data yang memenuhi syarat filter.
- Pada contoh program, syarat filter adalah length()<4.

Stream, map

```
public class CobaMap {
        public static void main(String[] args) {
                // Create a list of numbers
                List<Integer> numbers =
                        Arrays.asList(1, 2, 3, 4, 5);

                // Use the map function to square each number
                numbers.stream()
                        .map(number -> number * number)
                        .forEach(System.out::println);
        }
}
```

Stream, map

- map adalah sebuah standard fungsi dalam functional programming.
- map dalam pembahasan kali ini bukan lah map dalam struktur data atau class di dalam java Class seperti HashMap, melainkan sebuah fungsi dalam java stream
- Map akan menerima sebuah lambda function yang return sebuah type.
- Map akan memproses setiap data dalam koleksi kita dan menggantinya dengan hasil perhitungan sesuai dengan lambda function yang diberikan,
- Pada contoh program, map akan mengubah data menjadi hasil quadrat bilangan sebelumnya.

Stream, collect, Collector, toList

```
public class CobaCollect{
        public static void main(String[] args) {
                // Create a list of numbers
                List<Integer> numbers =
                        Arrays.asList(1, 2, 3, 4, 5);

                List<Integer> squares =
                        numbers.stream()
                                .map(number -> number * number)
                                .collect(Collectors.toList());

                System.out.println("Squares: " + squares);
        }
}
```

Stream, collect, Collector, toList

- Data yang kita miliki bisa lebih mudah kita oleh menggunakan stream. Dengan stream processing bisa juga dilakukan secara concurrent (parallel memanfaatkan multiple core yang dimiliki mesin computer, namun tidak kita bahas di DDP2)
- Setelah processing, kadang kita ingin menggunakan tipe class yang lebih mudah kita gunakan seperti List.
- Kita bisa menggunakan perintah collect untuk memproses keseluruhan data, kemudian kita panggil method Collector.toList() untuk menghasilkan object list.

Problem solving: Imperative vs Declarative

- Misalkan kita diminta membuat program yang:
  - Menerima sebuah array bilangan bulat.
  - Menghasilkan array yang merupakan quadrat
  - dari bilangan yang genap saja.

- Contoh:
  - Input: [1, 2 , 3, 8, 7, 6]
  - Output: [4, 64, 36]

- Bagaimana menyelesaikan permasalahan ini?

Imperative Style in Java

```
public static List<Integer> squareEvens(List<Integer> list){

        List<Integer> squareList = new ArrayList<>();

        for(Integer element : list){
                if(element % 2 == 0){
                        squareList.add(element * element);
                }
        }

        return squareList;
}
```

Declarative Style in Java

```
public static List<Integer> squareEvens(List<Integer> list){

        return list.stream()
                        .filter(x -> x % 2 == 0)
                        .map(y -> y * y)
                        .collect(Collectors.toList());
}
```

There are more … !

- Masih banyak lagi hal terkait java yang belum sempat kita bahas. Misalnya: reduce, Enum, Thread, inner class, dan lain-lain..
- Peserta diharapkan sudah memiliki basic yang kuat sehingga bisa mengembangkan sendiri pengetahuannya.

- Semangat terus berlatih!

UNIVERSITAS INDONESIA

FAKULTAS ILMU KOMPUTER

Ada Pertanyaan?

Persiapan Ujian Akhir Semester (28-05-2025)

- Perhatikan kembali, kerjakan kembali secara mandiri TP 3 dan TP 4!
- Pastikan anda memahami, bisa membuat kembali dan bisa melanjutkannya dengan menerapkan konsep yang telah dipelajari.
- Perhatikan juga unit test.
- Pahami semua konsep yang telah dipelajari dari awal kuliah.
- Materi ujian akhir mencakup materi dari awal kuliah dengan penekanan pada bagian setelah UTS.
- Selamat mempersiapkan diri untuk ujian.

UNIVERSITAS INDONESIA

FAKULTAS ILMU KOMPUTER

Selamat Berlatih!

Perhatikan lagi List Objective yang perlu dikuasai pekan ini.

Mari mencoba dan mempelajari nya di repl

Baca buku acuan dan berlatih!
Bila masih belum yakin tanyakan ke dosen, tutor atau Kak Burhan.

Semangat !