

LAPORAN PRAKTIKUM
PEMROGRAMAN PERANGKAT BERGERAK



JUDUL :
FUNDAMENTAL DART




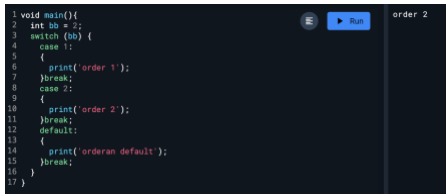

Disusun oleh:
ALFA YUDHA NUGRAHA (21102194)

TEKNIK INFORMATIKA
INSTITUT TEKNOLOGI TELKOM PURWOKERTO
BANYUMAS, JAWA TENGAH
2024

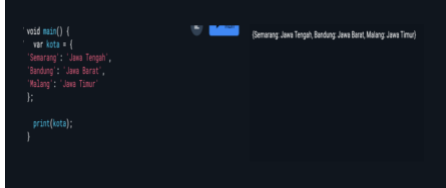




Pembahasan

Pada pertemuan kedua, praktikan mempelajari dasar-dasar Dart, yaitu bahasa pemrograman yang dikembangkan oleh Google dan digunakan untuk membangun aplikasi web, server, desktop, dan seluler. Praktikan mempelajari beberapa konsep dasar dalam Dart, seperti pembuatan variabel, tipe data yang didukung oleh Dart, penggunaan control flow, list, spread operator, set, dan map. Selain itu, praktikan juga mempelajari konsep paradigma pemrograman fungsional dan Pemrograman Berorientasi Objek untuk membuat program dengan struktur yang lebih terorganisir dan mudah dipelihara.

Langkah-Langkah Praktikum

Langkah Praktikum	Pembahasan
	Membuat dan mencetak "hello, world" di dalam fungsi main.
	Melakukan deklarasi dan inisialisasi variabel, kemudian menampilkannya.
	Melakukan deklarasi tipe data variabel secara eksplisit, kemudian menampilkannya.
	Menggunakan kondisi if else untuk menentukan apakah sebuah bilangan positif atau negatif.
	Menggunakan switch case untuk memilih berdasarkan nilai dari ekspresi tertentu, di mana dalam kode tersebut adalah mencetak nilai case 2.
	Menggunakan for untuk melakukan perulangan, di mana dalam kode tersebut dimulai dari satu dan ter-increment hingga 10.

<pre>void main() { var i = 1; while (i <= 10) { print(i); i++; } }</pre>	<p>Menggunakan while yang akan menjalankan blok kodenya selama kondisi tertentu terpenuhi.</p>
<pre>void main() { var i = 1; do { print(i); i++; } while (i <= 10); }</pre>	<p>Menggunakan do while, di mana blok kode tersebut akan dijalankan sekali, kemudian kondisi booleannya dievaluasi dan akan dijalankan kembali jika bernilai true.</p>
<pre>void main() { List<int> numberList = [1, 2, 3, 4, 5]; print(numberList); }</pre>	<p>Kode tersebut merupakan contoh sebuah objek List yang berisi kumpulan data bertipe integer.</p>
<pre>void main() { List dynamicList = [1, 'Informatika', true]; // List-dynamic print(dynamicList); }</pre>	<p>Membuat sebuah list yang dapat berisi elemen-elemen dengan tipe data apa pun, karena mengizinkan nilai-nilai dari jenis apa pun dalam list.</p>
<pre>void main() { var buah = ['Mangga', 'Apel', 'Jeruk', 'Manggis']; var hewan = ['Ayam', 'Kelinci', 'Kucing']; var allFavorites = [buah, hewan]; print(allFavorites); }</pre>	<p>Ketika tidak menggunakan operator penyebaran (spread operator), nilai dari list tidak akan tergabung secara otomatis. Sehingga, variabel allFavorites akan menjadi sebuah list yang menampung dua list di dalamnya.</p>
<pre>void main() { var buah = ['Mangga', 'Apel', 'Jeruk', 'Manggis']; var hewan = ['Ayam', 'Kelinci', 'Kucing']; var allFavorites = [...buah, ...hewan]; print(allFavorites); }</pre>	<p>Menggunakan spread operator akan menggabungkan nilai dari variabel hewan dan buah menjadi satu list. Sehingga ketika diprint, keduanya akan menjadi satu list yang tergabung.</p>
<pre>void main() { var angkaSet = {1, 4, 6}; Set<int> bilanganSet = new Set.from([1, 4, 6, 4, 1]); print(bilanganSet); }</pre>	<p>Membuat collection menggunakan set dan menyimpan nilai yang unik dan tidak duplikat, kemudian diprint.</p>

 <pre> void main() { var kota = { Semarang : Jawa Tengah, Bandung : Jawa Barat, Malang : Jawa Timur }; print(kota); } </pre>	<p>Membuat collection menggunakan map, yang datanya disampaikan menggunakan format keyvalue.</p>
 <pre> void main() { var kucing = Hewan('Ketty', 2, 3.2); kucing.makan(); kucing.tidur(); print(kucing.berat); } class Hewan { String nama; int umur; double berat; Hewan(this.nama, this.umur, this.berat); void makan() { print('\$nama makan. '); berat = berat + 0.2; } void tidur() { print('\$nama sedang tidur'); } } </pre>	<p>Menyusun sebuah kelas Hewan yang mencakup atribut-atribut dan metodenya, kemudian memanggil metode atau perilaku hewan tersebut berdasarkan informasi yang ada dalam kelas tersebut</p>
 <pre> void main() { var kucing = Animal('Ketty', 2, 3.2); kucing.on(); kucing.sleep(); kucing.pomp(); print(kucing.weight); } class Animal { String nama = ''; int age = 0; double weight = 0; Animal(this.nama, this.age, this.weight); // Setter set_name(String value) { .name = value; } // Getter double get_weight => .weight; void on() { print('\$nama is eating. '); weight = .weight + 0.2; } void sleep() { print('\$nama is sleeping. '); } void pomp() { print('\$nama is pumping. '); weight = .weight + 0.1; } } </pre>	<p>Implementasi metode setter dan getter memungkinkan akses dan modifikasi nilai properti dari luar kelas.</p>
 <pre> void main() { var kucing = Hewan('Ketty', 2, 3.2); kucing.makan(); kucing.tidur(); print(kucing.berat); } abstract class Hewan { String nama; int umur; double berat; Hewan(this.nama, this.umur, this.berat); void makan() { print('\$nama makan. '); berat = berat + 0.2; } void tidur() { print('\$nama sedang tidur'); } } </pre>	<p>Membuat kelas Hewan menjadi abstrak sehingga tidak dapat lagi diinisialisasi menjadi sebuah objek.</p>
 <pre> void main() { var elang = Burung('Elang', 5, 1.5, 'Coklat'); elang.makan(); elang.tidur(); elang.fly(); print(berat.elang.berat); print(berat.burung.berat); } class Hewan { String nama; int umur; double berat; Hewan(this.nama, this.umur, this.berat); void makan() { print('\$nama makan. '); berat = berat + 0.2; } void tidur() { print('\$nama sedang tidur'); } } class Burung extends Hewan implements Flyable { String warna; Burung(String nama, int umur, double berat, this.berat) : super(nama, umur, berat); override void fly() { print('\$nama is flying. '); } } abstract class Flyable { void fly(); } </pre>	<p>Membuat object Burung yang merupakan subclass dari class Hewan dan mengimplementasikan interface Flyable yang secara implisit diterapkan pada class Burung, lalu mengakses metode dan properti dari class tersebut dan mencetak informasi terkait burung yang telah didefinisikan sebelumnya.</p>

<pre> 1 void main() { 2 print(Pelangi.alam); 3 print(Pelangi.kuning); 4 print(Pelangi.hijau); 5 } 6 7 enum Pelangi { 8 merah, kuning, hijau, biru, nila, ungu 9 } 10 11 enum Status { 12 Tuto, Di_Progress, Di_Review, Done 13 } </pre>	<p>Membuat enum pelangi yang mendefinisikan kumpulan nilai konstan untuk warna-warna pelangi dan enum Status mendefinisikan kumpulan nilai konstan untuk status tugas guna memudahkan penggunaan dan manipulasi nilai-nilai tersebut dalam kode.</p>
<pre> 1 void main() { 2 int sum(int angka1, int angka2) { 3 return angka1 + angka2; 4 } 5 6 int hasil = sum(5, 3); 7 print('Hasil penjumlahan: \$hasil'); 8 } </pre>	<p>Memanggil fungsi sum berisikan parameternya, kemudian hasil penjumlahan tersebut diprint.</p>
<pre> 1 void main() { 2 int fibonacci(int n) { 3 if (n <= 0) { 4 return 0; 5 } else if (n == 1) { 6 return 1; 7 } else { 8 return fibonacci(n - 1) + fibonacci(n - 2); 9 } 10 } 11 12 int hasil = fibonacci(5); 13 print('Hasil fibonacci: \$hasil'); 14 } </pre>	<p>Membuat fungsi Fibonacci menggunakan pendekatan rekursif untuk menghitung angka dalam deret Fibonacci hingga mencapai nilai yang diinginkan.</p>
<pre> 1 void main() { 2 var sum = (int angka1, int angka2) { 3 return angka1 + angka2; 4 }; 5 6 Function printLambda = () { 7 print('Ini adalah fungsi lambda'); 8 }; 9 10 printLambda(); 11 print(sum(1, 4)); 12 } </pre> <pre> 1 void main() { 2 var sum = (int angka1, int angka2) => angka1 + angka2; 3 Function printLambda = () => print('This is lambda function'); 4 5 printLambda(); 6 print(sum(1, 4)); 7 } </pre>	<p>menggunakan anonymous function untuk variabel sum yang menjumlahkan dua angka, dan lambda function untuk variabel printLambda yang mencetak pesan. Lalu disederhanakan menggunakan fat arrow (=>).</p>
<pre> 1 void main() { 2 void containHigherOrderFunction(String message, Function myFunction) { 3 print(message); 4 print(myFunction(1, 4)); 5 } 6 7 // Ops 1 8 Function sum = (int num1, int num2) => num1 + num2; 9 containHigherOrderFunction('Hello', sum); 10 11 // Ops 2 12 containHigherOrderFunction('Hello', (num1, num2) => num1 + num2); 13 } </pre>	<p>Menggunakan higher-order function untuk menerima pesan dan fungsi, serta mendefinisikan fungsi secara langsung atau menggunakan lambda.</p>
<pre> 1 void main() { 2 var contohClosure = penjumlahan(); 3 contohClosure(); 4 contohClosure(); 5 } 6 7 Function penjumlahan(base) { 8 var a = 1; 9 return () => print('Nilainya adalah \$base + a++'); 10 } </pre>	<p>menggunakan closure untuk membuat fungsi penjumlahan yang menyimpan nilai base dan variabel lokal a, agar memungkinkan pemanggilan berulang.</p>

<pre>void main() { List<int> bilangan = [1, 2, 3, 4, 5]; List<String> kata = ['Informatika', 'Flutter', 'Penerjemahan']; List<dynamic> dynamicList = [1, 2, 3, 'empat']; print('Bilangan: \$bilangan'); print('Kata: \$kata'); print('Dynamic List: \$dynamicList'); }</pre>	<p>Kode tersebut menggunakan generic <code>List<int></code> dan <code>List<String></code> untuk menyimpan tipe data spesifik, serta <code>List<dynamic></code> untuk tipe data dinamis.</p>
<pre>void main() { Map<String, dynamic> jurusan = {'prodi': 'Informatika', 'angkatan': 2020}; print('Jurusan: \$jurusan'); }</pre>	<p>menggunakan type inference untuk menentukan tipe data Map secara otomatis berdasarkan nilai yang diberikan.</p>
<pre>void main() { getProduct().then((value) { print(value); }); } Future<String> getProduct() { return Future.delayed(Duration(seconds: 3), () { return 'Matcha Latte'; }); }</pre>	<p>menggunakan Future untuk menangani pemanggilan asinkron dan akan menunggu tiga detik sebelum mencetak nilai 'Matcha Latte'.</p>
<pre>void main() { getProduct().then((value) { print('Your product: \$value'); }); print('Getting your product...'); } Future<String> getProduct() { return Future.delayed(Duration(seconds: 3), () { return 'Matcha Latte'; }); }</pre>	<p>Membuat handler yang akan diprint terlebih dahulu dan setelah 3 detik pesan productnya akan diprint setelah proses future selesai.</p>
<pre>void main() { getProduct().then((value) { print('Your product: \$value'); }) .catchError((error) { print('Sorry, \$error'); }); print('Getting your product...'); } Future<String> getProduct() { return Future.delayed(Duration(seconds: 3), () { var isProductAvailable = false; if (isProductAvailable) { return 'Coffee Boba'; } else { throw 'Our stock is not enough.'; } }); }</pre>	<p>Membuat exception untuk mengatasi error di dalam future menggunakan catch error untuk menangkap error yang dihasilkan oleh future dan print pesan kesalahan ke layar.</p>
<pre>void main() async { print('Getting your product...'); try { var order = await getProduct(); print('You order: \$order'); } catch (error) { print('Sorry, \$error'); } finally { print('Thank you'); } } Future<String> getProduct() { return Future.delayed(Duration(seconds: 3), () { var isProductAvailable = false; if (isProductAvailable) { return 'Matcha Latte'; } else { throw 'Our stock is not enough.'; } }); }</pre>	<p>menggunakan <code>async-await</code> untuk menunggu hasil panggilan <code>getProduct</code> yang bersifat asinkron. Jika produk tidak tersedia, kesalahan yang dihasilkan oleh <code>throw</code> akan ditangkap oleh blok <code>catch</code>, sementara blok <code>finally</code> akan tetap dieksekusi.</p>