**Ders 1 - Dart’ın Temel Özellikleri :**

Dart, Google tarafından geliştirilen bir programlama dilidir. Dart, genel amaçlı bir dil olup, web uygulamaları, mobil uygulamalar, masaüstü uygulamaları ve IoT cihazları gibi birçok alanda kullanılabilir. İşte Dart programlama dilinin bazı özellikleri:

1. Sınıf Tabanlı: Dart, nesne yönelimli bir programlama dilidir. Sınıflar ve nesneler kullanarak programlama yapılır.
2. Tiplendirme: Dart, isteğe bağlı olarak tip kontrolü yapabilen bir dil olduğundan, statik olarak veya dinamik olarak tiplendirilebilir.
3. Açık Kaynaklı: Dart, açık kaynaklı bir projedir. Bu nedenle, topluluk geliştiricileri tarafından kullanılabilir ve geliştirilebilir.
4. Hızlı: Dart, hızlı çalışan bir dil olarak bilinir. Dart'ın JIT (Just-In-Time) ve AOT (Ahead-Of-Time) derleyicileri sayesinde, uygulamalar hızlı bir şekilde çalışır.
5. Kolay Öğrenilebilir: Dart, C, C++, Java ve JavaScript gibi diğer popüler dillerden özellikler alır ve bu nedenle kolay öğrenilebilir.
6. Asenkron Programlama: Dart, asenkron programlama yapmak için async ve await anahtar kelimelerini içerir. Bu, uzun süre çalışan işlemleri etkili bir şekilde yönetmenizi sağlar.
7. Platform Bağımsız: Dart, farklı platformlarda çalışabilen bir dildir. Web, mobil, masaüstü ve IoT gibi birçok platformda çalışabilir.
8. Güncel: Dart, düzenli olarak güncellenen bir dil olarak bilinir. Bu nedenle, yeni özellikler eklenir ve hatalar düzeltilir.

**Ders 2 - Main Metodu :**

Dart dilinde, herhangi bir uygulamayı çalıştırmak için main() metodu gereklidir. Main() metodu, uygulamanın giriş noktasını temsil eder ve uygulamanın çalışması için gereken kodları içerir. Dart dilinde main() metodu şu şekilde yazılır:

void main() {

// uygulama kodları burada yer alır

}

Main() metodunun başında **void** anahtar kelimesi yer alır, bu anahtar kelimesi metotun bir değer döndürmediğini ifade eder. Parantezlerin içine herhangi bir parametre yazılmadığında, main() metodu herhangi bir parametre almaz. Uygulamanın başlatılması ve çalıştırılması sırasında, main() metodu otomatik olarak çağrılır ve uygulamanın çalışması başlar.

**Ders 3 - Değişkenler :**

Dart dilinde değişkenler, değerleri saklamak için kullanılan isimlendirilmiş bellek konumlarıdır. Değişkenler, veri tiplerine sahiptir ve bu veri tiplerine uygun değerler saklarlar. Dart dilinde değişkenlerin tanımlanması ve kullanımı şu şekildedir:

void main() {

  // değişken tanımlama

//veri\_tipi degisken\_adi = deger;

// örnek değişken tanımlamaları

  int sayi = 10;

  double ondalik\_sayi = 3.14;

  String metin = "Merhaba Dünya!";

  bool durum = true;

  print(sayi);

  print(ondalik\_sayi);

  print(metin);

  print(durum);

}

Yukarıdaki örnekte, değişkenlerin tanımlanmasında, önce veri tipi belirtilir, sonra değişkenin adı ve son olarak da değeri belirtilir. Dart dilinde kullanılan bazı veri tipleri şunlardır:

* **int**: tamsayıları temsil eder.
* **double**: ondalıklı sayıları temsil eder.
* **String**: metinleri temsil eder.
* **bool**: mantıksal değerleri temsil eder (true veya false).

Ayrıca, **var** anahtar kelimesi ile değişkenler tanımlanabilir. **var** kullanıldığında, değişkenin veri tipi, atanacak değerden otomatik olarak belirlenir. Örneğin:

 var isim = "Ahmet";

  var yas = 25;

**Ders 4 – Dart Veri Tipleri :**

Dart dilinde kullanılan veri tipleri şunlardır:

1. Numbers: Tamsayılar (int) ve ondalık sayılar (double) olmak üzere iki tür sayısal veri tipi vardır.
2. Strings: Metinsel ifadeleri temsil eder. Tek tırnak veya çift tırnak içinde tanımlanabilir.
3. Booleans: Mantıksal ifadeleri temsil eder. Sadece true veya false değerlerini alabilir.
4. Lists: Dizileri temsil eder. Aynı türden veya farklı türlerde elemanlar içerebilir.
5. Sets: Listeler gibi farklı türden veya aynı türden elemanlar içerebilir. Fakat, set'ler elemanları benzersizdir.
6. Maps: Key-value çiftleri şeklinde verileri temsil eder. Key'ler ve value'lar herhangi bir veri tipinde olabilir.
7. Runes: Unicode karakterleri temsil eder.
8. Symbols: Programcıların kodlarını yazarken kullanabileceği sembolik ifadeleri temsil eder.

**Ders 5 – final ve const :**

Dart dilinde **final** ve **const** anahtar kelimeleri, değişkenlerin değerlerinin değiştirilmesini önlemek için kullanılır.

**final** anahtar kelimesi, değeri tanımlandıktan sonra değiştirilemeyen değişkenler için kullanılır. **final** anahtar kelimesi ile tanımlanan bir değişken, yalnızca bir kez değer atanabilir. Örneğin:

final int sayi = 10;

Yukarıdaki örnekte, **sayi** değişkeni, **final** anahtar kelimesi ile tanımlandığı için yalnızca bir kez değer atanabilir ve daha sonra değeri değiştirilemez.

**const** anahtar kelimesi de **final** gibi davranır, ancak **const** değişkenleri derleme zamanında sabit olarak belirlenir. **const** anahtar kelimesi ile tanımlanan bir değişken, yalnızca derleme zamanında değer atanabilir. Örneğin:

const pi = 3.14;

Yukarıdaki örnekte, **pi** değişkeni, **const** anahtar kelimesi ile tanımlandığı için yalnızca derleme zamanında değer atanabilir ve daha sonra değeri değiştirilemez.

Genel olarak, **final** anahtar kelimesi ile tanımlanan değişkenler çalışma zamanında değer atanırken, **const** anahtar kelimesi ile tanımlanan değişkenler derleme zamanında değer atanır ve programın çalışma hızını artırır.

**Ders 6 – if-else :**

Dart dilinde **if** ve **else** ifadeleri, koşullu karar yapıları oluşturmak için kullanılır. **if** ifadesi, belirtilen koşulun doğru olması durumunda belirtilen kod bloğunu çalıştırır, **else** ifadesi ise koşul yanlış olduğunda çalışacak kod bloğunu belirtir.

Aşağıdaki örnek, kullanıcının yaşına göre belirli bir mesajı gösterir:

void main() {

  int yas = 18;

  if (yas >= 18) {

    print("Yaşınız 18 veya daha büyük.");

  } else {

    print("Yaşınız 18'den küçük.");

  }

}

Yukarıdaki örnekte, **yas** değişkeni 18'e eşit olduğu için **if** bloğu çalışır ve "Yaşınız 18 veya daha büyük." mesajı görüntülenir. Eğer **yas** değişkeni 18'den küçük olsaydı, **else** bloğu çalışacaktı ve "Yaşınız 18'den küçük." mesajı görüntülenirdi.

Ayrıca, **else if** ifadesi de kullanılarak daha fazla koşul belirtilerek daha fazla karar bloğu eklemek mümkündür. Aşağıdaki örnek, kullanıcının notuna göre farklı mesajlar gösterir:

void main() {

  int not = 75;

  if (not >= 90) {

    print("Notunuz AA");

  } else if (not >= 80) {

    print("Notunuz BA");

  } else if (not >= 70) {

    print("Notunuz BB");

  } else if (not >= 60) {

    print("Notunuz CB");

  } else if (not >= 50) {

    print("Notunuz CC");

  } else {

    print("Dersten kaldınız.");

  }

}

Yukarıdaki örnekte, **if** bloğu ilk olarak **not >= 90** koşulunu kontrol eder. Eğer koşul doğru ise "Notunuz AA" mesajı görüntülenir. Eğer koşul yanlış ise **else if** bloğuna geçilir ve diğer koşullar teker teker kontrol edilir. Eğer hiçbir koşul sağlanmazsa **else** bloğu çalışır ve "Dersten kaldınız." mesajı görüntülenir.

**Ders 7 – switch-case :**

Dart dilinde, switch ifadesi, bir değişkenin farklı değerleri için farklı işlemler yapmak için kullanılır. switch ifadesi, bir koşuldan çok farklı değerler için farklı işlemler yapmak için kullanılabilir. Aşağıdaki örnek, kullanıcının seçtiği renge göre farklı mesajlar görüntüler:

void main() {

  String renk = 'kırmızı';

  switch (renk) {

    case 'kırmızı':

      print('Kırmızı renk seçildi.');

      break;

    case 'mavi':

      print('Mavi renk seçildi.');

      break;

    case 'yeşil':

      print('Yeşil renk seçildi.');

      break;

    default:

      print('Geçersiz renk seçildi.');

  }

}

Yukarıdaki örnekte, **switch** ifadesi, **renk** değişkeninin değerini kontrol eder ve belirtilen **case** ifadelerini karşılaştırır. Eğer **renk** değişkeni **'kırmızı'** ise, **"Kırmızı renk seçildi."** mesajı görüntülenir. Eğer **renk** değişkeni **'mavi'** ise, **"Mavi renk seçildi."** mesajı görüntülenir. Eğer **renk** değişkeni **'yeşil'** ise, **"Yeşil renk seçildi."** mesajı görüntülenir. Eğer **renk** değişkeni hiçbir **case** ifadesine uymazsa, **default** ifadesi çalışır ve **"Geçersiz renk seçildi."** mesajı görüntülenir.

Her **case** ifadesinin sonunda **break** anahtar kelimesi kullanılması gerektiğine dikkat edin. Bu, karşılaşılan durumdan çıkıldığını ve diğer **case** ifadelerine bakılmayacağını belirtir. Ayrıca, **default** ifadesi, hiçbir **case** ifadesine uymayan durumlar için bir yedek seçenek olarak kullanılabilir.

**Ders 8 – for - döngüsü :**

Dart dilinde, **for** döngüsü, belirli bir aralıktaki değerler üzerinde tekrarlanan işlemler yapmak için kullanılır.

**for** döngüsü, aşağıdaki gibi üç bileşenden oluşur:

cssCopy code

for (başlatıcı; koşul; artırım) { // işlemler }

* **başlatıcı**: Döngü başlamadan önce bir kere çalışır ve genellikle döngü değişkenlerinin başlangıç değerleri atanır.
* **koşul**: Döngü her çalıştığında kontrol edilir. Koşul sağlandığı sürece döngü çalışmaya devam eder.
* **artırım**: Döngü sonunda her defasında bir kere çalışır ve genellikle döngü değişkenlerinin artırılması gibi işlemler yapılır.

Aşağıdaki örnek, **for** döngüsünün kullanımını göstermektedir:

void main() {

  for (int i = 0; i < 5; i++) {

    print(i);

  }

}

Yukarıdaki örnekte, **for** döngüsü, **i** değişkenini **0** olarak başlatır, **i** değişkeni **5** olduğu sürece çalışır ve her çalıştığında **i** değişkenini **1** artırır. Döngü her çalıştığında **i** değişkeni ekrana yazdırılır.

**for** döngüsü, bir dizi üzerinde de çalıştırılabilir. Aşağıdaki örnekte, **for** döngüsü bir **List** üzerinde çalıştırılmaktadır:

void main() {

  List<int> liste = [1, 2, 3, 4, 5];

  for (int eleman in liste) {

    print(eleman);

  }

}

Yukarıdaki örnekte, **for** döngüsü, **liste** adlı bir **List** üzerinde çalıştırılır. Döngü, her seferinde **liste** içindeki bir elemanı alır ve **eleman** değişkenine atar. Döngü her çalıştığında **eleman** değişkeni ekrana yazdırılır.

**Ders 9 – while ve do-while döngüsü :**

Dart dilinde while ve do-while döngüleri, belirli bir koşul sağlandığı sürece tekrarlanan işlemler yapmak için kullanılır.

**While Döngüsü**

while döngüsü, döngü koşulu doğru olduğu sürece işlemleri tekrarlar. Döngü koşulu yanlış olduğunda, döngü sona erer. Aşağıdaki örnek, while döngüsünün kullanımını göstermektedir:

void main() {

  int i = 0;

  while (i < 5) {

    print(i);

    i++;

  }

}

Yukarıdaki örnekte, **while** döngüsü, **i** değişkeni **5** olana kadar çalışır. Her seferinde, **i** değişkeni ekrana yazdırılır ve **i** değişkeni **1** artırılır.

**Do-While Döngüsü**

**do-while** döngüsü, döngü koşulunu en sona yazarak, işlemlerin en az bir kez yapılmasını garanti eder. Aşağıdaki örnek, **do-while** döngüsünün kullanımını göstermektedir:

void main() {

  int i = 0;

  do {

    print(i);

    i++;

  } while (i < 5);

}

Yukarıdaki örnekte, **do-while** döngüsü, **i** değişkeni **5** olana kadar çalışır. Her seferinde, **i** değişkeni ekrana yazdırılır ve **i** değişkeni **1** artırılır. **do-while** döngüsü, koşul doğru olduğu sürece çalıştığı için, koşul sağlanmadığı halde döngü en az bir kere çalışır.

**Ders 10 – break ve continue :**

Dart dilinde **break** ve **continue** ifadeleri, döngülerde kullanılan kontrol ifadeleridir.

**break** ifadesi, döngü içindeki işlemi tamamlar ve döngüden çıkar. Örneğin, aşağıdaki örnekte, **while** döngüsü, **i** değişkeni **3** olduğunda durdurulur ve döngüden çıkar:

void main() {

  int i = 0;

  while (i < 5) {

    print(i);

    i++;

    if (i == 3) {

      break;

    }

  }

}

Yukarıdaki örnekte, **while** döngüsü **i** değişkeni **3** olduğunda durdurulur ve döngüden çıkar.

**continue** ifadesi ise, döngü içindeki işlemi atlayarak döngünün bir sonraki adımına devam eder. Örneğin, aşağıdaki örnekte, **for** döngüsü, **i** değişkeni **3** olduğunda atlanır ve döngünün bir sonraki adımına geçilir:

void main() {

  for (int i = 0; i < 5; i++) {

    if (i == 3) {

      continue;

    }

    print(i);

  }

}

Yukarıdaki örnekte, **for** döngüsü **i** değişkeni **3** olduğunda atlanır ve döngünün bir sonraki adımına geçilir. Bu nedenle **3** sayısı yazdırılmaz ve diğer sayılar yazdırılır.

**break** ve **continue** ifadeleri, döngülerde kullanılarak programın kontrolünü sağlamada oldukça faydalıdır.

**Ders 11 – null safety ve late kavramı :**

Dart dilinde, null safety özelliği ile birlikte late anahtar kelimesi de kullanılmaktadır.

null safety, kodun daha güvenli hale getirilmesini sağlar. Bu özellik sayesinde, değişkenlerin null olup olmadıklarını kontrol etmek ve null değerlerle ilgili hataları önlemek daha kolay hale gelir. null safety özelliği varsayılan olarak Dart 2.12 ve sonraki sürümlerde kullanılmaktadır.

late anahtar kelimesi ise değişkenlerin değerlerinin ileride atanacağını belirtmek için kullanılır. Böylece, değişkenin ilk değeri atanmadan kullanılması durumunda hata alınmaz.

Aşağıdaki örnekte, null safety ve late anahtar kelimelerinin kullanımı gösterilmektedir:

late String lateExample;

String? nullableExample;

void main() {

  // nullableExample = null;  // Hata! nullableExample null olabilir.

  lateExample = "Late example";

  print(lateExample); // Geç atama yapıldığı için hata alınmaz.

}

Yukarıdaki örnekte, **nullableExample** değişkeni null olabilirken, **lateExample** değişkeninin geç atanacağı belirtilmiştir. **lateExample** değişkeni kullanılmadan önce atanmadığı halde, program hata vermez. Ancak **nullableExample** değişkeni null olduğunda, kullanılmadan önce atanmamış olması hata verecektir.

**Ders 12 – List Yapısı :**

Dart programlama dilinde List yapısı, birden fazla veri elemanını tutmak için kullanılır. List yapısı, bir sıralı koleksiyondur, yani elemanlar sırayla indekslenir ve bu indekslerle erişilebilir.

Dart'ta List yapısı iki türde tanımlanabilir:

1. Fixed-Length List (Sabit Uzunluklu Liste): Belirli bir uzunluğa sahip bir liste tanımlarız. Bu liste boyutu tanımlanırken belirlenir ve daha sonra değiştirilemez. Bu türdeki bir liste aşağıdaki gibi tanımlanır:

dartCopy code

List<int> fixedList = List(5);

1. Growable List (Genişletilebilir Liste): Boyutu önceden belirlenmemiş bir listeyi ifade eder. Yani, liste boyutu dinamik olarak değiştirilebilir. Bu türdeki bir liste aşağıdaki gibi tanımlanabilir:

dartCopy code

List<int> growableList = [];

List yapısı, Dart'ta standart kütüphanenin bir parçasıdır, bu nedenle kullanmak için ekstra bir kütüphane yüklemenize gerek yoktur. Liste elemanlarına, indeks numarasıyla erişilebilir. Örneğin:

void main() {

  var myList = [1, 2, 3, 4, 5];

  print(myList[0]); // 1

  print(myList[3]); // 4

  // eleman ekleme

  myList.add(6);

  myList.addAll([7, 8]);

// eleman silme

  myList.remove(5);

  myList.removeAt(0);

// eleman filtreleme

  var filteredList = myList.where((element) => element % 2 == 0).toList();

// eleman sıralama

  myList.sort();

}

**Ders 13 – List Metodları :**

Dart List yapısı, birçok farklı metoda sahiptir. Aşağıda, en yaygın kullanılan Dart List metotlarından bazıları açıklanmıştır:

add(): Bir eleman ekler.

addAll(): Birden fazla eleman ekler.

insert(): Belirli bir indekse eleman ekler.

remove(): Belirtilen elemanı listeden siler.

removeAt(): Belirtilen indeksteki elemanı listeden siler.

clear(): Listeyi temizler.

length: Listenin uzunluğunu döndürür.

indexOf(): Belirtilen elemanın indeksini döndürür.

lastIndexOf(): Belirtilen elemanın son indeksini döndürür.

contains(): Belirtilen elemanın listede olup olmadığını kontrol eder.

forEach(): Listenin her bir elemanı üzerinde bir işlem yapar.

map(): Listenin her bir elemanı üzerinde bir işlem yapar ve yeni bir liste döndürür.

where(): Belirli bir koşulu sağlayan elemanları içeren yeni bir liste döndürür.

sort(): Listeyi belirtilen bir kurala göre sıralar.

sublist(): Belirtilen aralıktaki elemanları içeren yeni bir liste döndürür.

void main() {

  // Listeyi oluştur

  List<String> myFruits = ['elma', 'armut', 'muz'];

  // Eleman ekle

  myFruits.add('çilek');

  print(myFruits); // ['elma', 'armut', 'muz', 'çilek']

  // Birden fazla eleman ekle

  myFruits.addAll(['kivi', 'portakal']);

  print(myFruits); // ['elma', 'armut', 'muz', 'çilek', 'kivi', 'portakal']

  // Belirli bir indekse eleman ekle

  myFruits.insert(1, 'şeftali');

  print(myFruits); // ['elma', 'şeftali', 'armut', 'muz', 'çilek', 'kivi', 'portakal']

  // Belirtilen elemanı listeden sil

  myFruits.remove('elma');

  print(myFruits); // ['şeftali', 'armut', 'muz', 'çilek', 'kivi', 'portakal']

  // Belirtilen indeksteki elemanı listeden sil

  myFruits.removeAt(2);

  print(myFruits); // ['şeftali', 'armut', 'çilek', 'kivi', 'portakal']

  // Listeyi temizle

  myFruits.clear();

  print(myFruits); // []

  // Liste uzunluğunu döndür

  List<int> myNumbers = [1, 2, 3];

  print(myNumbers.length); // 3

  // Belirtilen elemanın indeksini döndür

  List<String> myColors = ['kırmızı', 'mavi', 'yeşil'];

  print(myColors.indexOf('mavi')); // 1

  // Belirli bir koşulu sağlayan elemanları içeren yeni bir liste döndür

  List<int> myValues = [1, 2, 3, 4, 5];

  var filteredList = myValues.where((element) => element % 2 == 0).toList();

  print(filteredList); // [2, 4]

  // Listeyi belirtilen bir kurala göre sırala

  List<int> mySortList = [5, 3, 1, 4, 2];

  mySortList.sort();

  print(mySortList); // [1, 2, 3, 4, 5]

}

**Ders 14 – Liste Üzerinde Gezinme :**

void main() {

  List<String> myFruits = ['elma', 'armut', 'muz'];

  // 1. Yöntem: For döngüsü kullanarak gezinme

  for (int i = 0; i < myFruits.length; i++) {

    print(myFruits[i]);

  }

  // 2. Yöntem: For-in döngüsü kullanarak gezinme

  for (String fruit in myFruits) {

    print(fruit);

  }

  // 3. Yöntem: ForEach metodu kullanarak gezinme

  myFruits.forEach((fruit) => print(fruit));

  // 4. Yöntem: Iterator kullanarak gezinme

  var iterator = myFruits.iterator;

  while (iterator.moveNext()) {

    print(iterator.current);

  }

}

Yukarıdaki kod örneğinde, öncelikle **List<String>** türünde bir **myFruits** liste tanımlanır. Daha sonra, bu listedeki öğeleri dört farklı yöntemle gezinir ve ekrana yazdırılır:

1. For döngüsü kullanarak gezinme: Listenin uzunluğuna göre bir **for** döngüsü kullanılır ve listenin her bir öğesi, öğe indeksi kullanılarak ekrana yazdırılır.
2. For-in döngüsü kullanarak gezinme: Listenin her bir öğesi, **for-in** döngüsü kullanılarak doğrudan döngü değişkeni olarak belirtilir ve ekrana yazdırılır.
3. ForEach metodu kullanarak gezinme: **forEach** metodu, listedeki her bir öğe için belirtilen işlemi gerçekleştirir. Bu örnekte, her bir öğe doğrudan ekrana yazdırılır.
4. Iterator kullanarak gezinme: Iterator, listedeki her bir öğeye sırayla erişir ve **while** döngüsü kullanılarak bu öğeler ekrana yazdırılır.

**Ders 15 – Class Yapısı :**

// Sınıf yapısı

class Araba {

  // Sınıf özellikleri (instance variables)

  String marka;

  String model;

  int yil;

  // Yapıcı metot (constructor)

  Araba(this.marka, this.model, this.yil);

  // Metotlar

  void calistir() {

    print("$marka $model çalıştı");

  }

  void durdur() {

    print("$marka $model durdu");

  }

}

// Sınıf örneği oluşturma

void main() {

  var araba1 = Araba("Toyota", "Corolla", 2020);

  araba1.calistir(); // çıktı: Toyota Corolla çalıştı

  araba1.durdur(); // çıktı: Toyota Corolla durdu

  var araba2 = Araba("Honda", "Civic", 2018);

  araba2.calistir(); // çıktı: Honda Civic çalıştı

  araba2.durdur(); // çıktı: Honda Civic durdu

}

Yukarıdaki kodda **Araba** adında bir sınıf tanımlanmıştır. Sınıf, **marka**, **model** ve **yil** adında üç özellik içermektedir. Bunlar, her bir sınıf örneği (instance) için benzersiz değerler alabilirler.

Sınıf ayrıca **calistir** ve **durdur** adında iki metot içerir. Bu metotlar, her bir sınıf örneği için ayrı ayrı çağrılabilirler. **calistir** metodu, araba marka ve modelini çıktı olarak verirken, **durdur** metodu sadece araba marka ve modelini verir.

**main** fonksiyonunda, **Araba** sınıfından iki örnek (**araba1** ve **araba2**) oluşturulmuştur. Bu örnekler, farklı **marka**, **model** ve **yil** değerlerine sahiptirler. Her iki örnek de **calistir** ve **durdur** metotlarıyla çağrılabilirler.

**Ders 16 – Metotlar :**

// Parametre almayan ve geriye değer döndürmeyen metot

void selamla() {

  print("Merhaba!");

}

// Parametre alan ve geriye değer döndürmeyen metot

void selamVer(String isim) {

  print("Merhaba, $isim!");

}

// Parametre alan ve geriye değer döndüren metot

int topla(int sayi1, int sayi2) {

  return sayi1 + sayi2;

}

// İsimlendirilmiş parametreler kullanan metot

void bilgiVer({String? ad, int? yas}) {

  print("Adı: $ad, Yaşı: $yas");

}

// Opsiyonel parametreler kullanan metot

void bilgiGoster(String ad, {int yas = 0, String sehir = "Bilinmiyor"}) {

  print("Adı: $ad, Yaşı: $yas, Şehri: $sehir");

}

void main() {

  selamla(); // çıktı: Merhaba!

  selamVer("Ahmet"); // çıktı: Merhaba, Ahmet!

  int sonuc = topla(5, 3);

  print("Toplam: $sonuc"); // çıktı: Toplam: 8

  bilgiVer(ad: "Mehmet", yas: 35); // çıktı: Adı: Mehmet, Yaşı: 35

  bilgiGoster("Ayşe",

      yas: 28, sehir: "Ankara"); // çıktı: Adı: Ayşe, Yaşı: 28, Şehri: Ankara

}

Yukarıdaki kodda, farklı türde metotlar tanımlanmıştır. İlk olarak **selamla** adında bir metot tanımlanmıştır. Bu metot, hiçbir parametre almaz ve hiçbir değer döndürmez. Sadece "Merhaba!" metnini çıktı olarak verir.

**selamVer** metodu ise bir parametre alır, bu parametre bir **String** türünde **isim** değişkenidir. Bu metot da hiçbir değer döndürmez, ancak parametre olarak aldığı ismi kullanarak "Merhaba, {isim}!" şeklinde bir çıktı verir.

**topla** adlı metot, iki tane **int** tipinde parametre alır ve bu parametreleri toplayarak bir sonuç döndürür.

**bilgiVer** adlı metot, **ad** ve **yas** adlı iki tane isimlendirilmiş parametre alır. Bu parametreler, metodu çağıran tarafından isimleriyle birlikte verilmelidir. Bu metot, aldığı parametreleri kullanarak basit bir bilgi çıktısı verir.

**bilgiGoster** adlı metot ise **ad**, **yas** ve **sehir** adlı parametreler alır. **yas** ve **sehir** parametreleri opsi yoneldir, yani kullanıcı tarafından verilmesi isteğe bağlıdır. Bu metot, aldığı parametreleri kullanarak bilgi çıktısı verir. Eğer **yas** ve **sehir** parametreleri verilmezse, **yas** için varsayılan değer olarak **0**, **sehir** için ise varsayılan olarak **"Bilinmiyor"** atanır.

Ana fonksiyon (**main**) içinde bu farklı metotlar çağrılır ve çıktıları ekrana yazdırılır.

**Ders 17 – Constructors :**

// Örnek sınıf tanımı

class Personel {

  String? ad;

  int? yas;

  // Default constructor

  Personel() {

    ad = "İsim Yok";

    yas = 0;

  }

  // Named constructor

  Personel.yasBelirle(String ad, int yas) {

    this.ad = ad;

    this.yas = yas;

  }

  // Kurucu metot

  Personel.kurucuMetot(this.ad, this.yas);

}

void main() {

  // Default constructor örneği

  Personel p1 = Personel();

  print("p1 adı: ${p1.ad}, yaş: ${p1.yas}"); // çıktı: p1 adı: İsim Yok, yaş: 0

  // Named constructor örneği

  Personel p2 = Personel.yasBelirle("Ali", 25);

  print("p2 adı: ${p2.ad}, yaş: ${p2.yas}"); // çıktı: p2 adı: Ali, yaş: 25

  // Kurucu metot örneği

  Personel p3 = Personel.kurucuMetot("Ayşe", 30);

  print("p3 adı: ${p3.ad}, yaş: ${p3.yas}"); // çıktı: p3 adı: Ayşe, yaş: 30

}

Yukarıdaki kodda, **Personel** adında bir sınıf tanımlanmıştır. Bu sınıfın iki adet değişkeni (**ad** ve **yas**) vardır.

Sınıfın default constructor'ı (**Personel()**) hiçbir parametre almaz ve sınıf örnekleri oluşturulurken **ad** değişkenine "İsim Yok", **yas** değişkenine 0 değeri atanır.

Named constructor (**Personel.yasBelirle**) ise **ad** ve **yas** adlı iki tane parametre alır ve bu parametreleri kullanarak bir sınıf örneği oluşturur.

Kurucu metot (**Personel.kurucuMetot**) ise değişkenleri kısa bir şekilde tanımlamak için kullanılır. Bu kurucu metot, **ad** ve **yas** adlı iki tane parametre alır ve bu parametreleri sınıfın **ad** ve **yas** değişkenlerine atar.

Ana fonksiyonda (**main**) bu farklı constructor'lar kullanılarak sınıf örnekleri oluşturulur ve örneklerin özellikleri ekrana yazdırılır.

**Ders 18 – Getter ve Setter :**

class Ogrenci {

  String \_ad;

  int \_not;

  // Constructor

  Ogrenci(this.\_ad, this.\_not);

  // Getter

  String? get ad => \_ad;

  int get notu => \_not;

  // Setter

  set notu(int yeninot) {

    if (yeninot >= 0 && yeninot <= 100) {

      \_not = yeninot;

    } else {

      print("Hatalı not girişi!");

    }

  }

}

void main() {

  // Ogrenci örneği oluştur

  Ogrenci ogrenci1 = Ogrenci("Ahmet", 80);

  // Getter kullanımı

  print("Öğrenci adı: ${ogrenci1.ad}");

  print("Öğrenci notu: ${ogrenci1.notu}");

  // Setter kullanımı

  ogrenci1.notu = 110; // Hatalı not girişi!

  ogrenci1.notu = 95;

  print("Yeni notu: ${ogrenci1.notu}");

}

Yukarıdaki örnekte, **Ogrenci** adında bir sınıf tanımlanmıştır. Sınıfın **\_ad** ve **\_not** adında iki adet private değişkeni vardır.

Sınıfın constructor'ı **\_ad** ve **\_not** parametreleri alır ve bu parametrelerle bir **Ogrenci** örneği oluşturur.

Sınıfın **ad** ve **notu** adında iki adet public getter'ı vardır. **ad** getter'ı **\_ad** değişkeninin değerini döndürürken, **notu** getter'ı **\_not** değişkeninin değerini döndürür.

Sınıfın **notu** adında bir setter'ı vardır. Bu setter, **\_not** değişkeninin değerini değiştirirken, **yeninot** adlı parametrenin geçerli bir not olup olmadığını kontrol eder. Eğer **yeninot** değişkeni 0 ile 100 arasında bir değer ise **\_not** değişkenine atanır, aksi halde hatalı not girişi mesajı ekrana yazdırılır.

Ana fonksiyonda (**main**) bu sınıftan bir **ogrenci1** örneği oluşturulur. Örneğin **ad** ve **notu** değerleri ekrana yazdırılır. Sonrasında **ogrenci1.notu** değişkenine geçersiz bir değer atanmaya çalışılır ve setter tarafından engellenir. Sonrasında **ogrenci1.notu** değişkenine geçerli bir değer atanarak değerinin değiştirildiği ekrana yazdırılır.

**Ders 19 – Kalıtım:**

class Insan {

  String ad;

  int yas;

  Insan(this.ad, this.yas);

  void bilgileriYazdir() {

    print("Adı: $ad");

    print("Yaşı: $yas");

  }

}

class Ogrenci extends Insan {

  String bolum;

  Ogrenci(String ad, int yas, this.bolum) : super(ad, yas);

  void bolumYazdir() {

    print("Bölümü: $bolum");

  }

  void bilgileriYazdir() {

    super.bilgileriYazdir();

    print("Bölümü: $bolum");

  }

}

void main() {

  // Ogrenci örneği oluştur

  Ogrenci ogrenci1 = Ogrenci("Ahmet", 22, "Bilgisayar Mühendisliği");

  // Ogrenci'nin adı, yaşı ve bölümü yazdırılıyor

  ogrenci1.bilgileriYazdir();

  // Ogrenci'nin bölümü yazdırılıyor

  ogrenci1.bolumYazdir();

}

Yukarıdaki örnekte, Insan adında bir üst sınıf (ya da ana sınıf) ve Ogrenci adında bir alt sınıf tanımlanmıştır. Insan sınıfında ad ve yas adında iki değişken ve bilgileriYazdir() adında bir fonksiyon bulunur.

Ogrenci sınıfı, Insan sınıfından kalıtım alır ve bolum adında bir değişken eklenir. Ogrenci sınıfının constructor'ı, ad, yas ve bolum parametrelerini alarak bir Ogrenci örneği oluşturur ve super anahtar kelimesiyle üst sınıfın constructor'ını çağırır.

Ogrenci sınıfında ayrıca bolumYazdir() adında bir fonksiyon daha bulunur.

Ogrenci sınıfında bilgileriYazdir() fonksiyonu, üst sınıftaki bilgileriYazdir() fonksiyonunu super anahtar kelimesiyle çağırarak önce ad ve yas bilgilerini ekrana yazdırır, ardından da bolum bilgisini yazdırır.

Ana fonksiyonda (main) bir Ogrenci örneği oluşturulur ve bilgileriYazdir() ve bolumYazdir() fonksiyonları çağrılarak ilgili bilgiler ekrana yazdırılır.

**Ders 20 – Çok Biçimlilik:**

class Sekil {

  double alan() {

    return 0;

  }

}

class Daire extends Sekil {

  double yaricap;

  Daire(this.yaricap);

  @override

  double alan() {

    return 3.14 \* yaricap \* yaricap;

  }

}

class Dikdortgen extends Sekil {

  double en;

  double boy;

  Dikdortgen(this.en, this.boy);

  @override

  double alan() {

    return en \* boy;

  }

}

void main() {

  List<Sekil> sekiller = [Daire(5), Dikdortgen(4, 6)];

  for (var sekil in sekiller) {

    print(sekil.alan());

  }

}

Yukarıdaki örnekte, Sekil adında bir üst sınıf (ya da ana sınıf) ve Daire ve Dikdortgen adında iki alt sınıf tanımlanmıştır. Sekil sınıfında alan() adında bir fonksiyon bulunur. Bu fonksiyon varsayılan olarak 0 döndürür.

Daire sınıfı, Sekil sınıfından kalıtım alır ve yaricap adında bir değişken ekler. Daire sınıfının alan() fonksiyonu, dairenin alanını hesaplar ve döndürür.

Dikdortgen sınıfı da Sekil sınıfından kalıtım alır ve en ve boy adında iki değişken ekler. Dikdortgen sınıfının alan() fonksiyonu, dikdörtgenin alanını hesaplar ve döndürür.

Ana fonksiyonda (main), Sekil tipindeki iki farklı şekil (bir daire ve bir dikdörtgen) oluşturulur ve bu şekiller bir liste içine konulur. Daha sonra liste üzerinde döngü oluşturulur ve her bir şeklin alanı ekrana yazdırılır.

Bu örnekte, Sekil sınıfı, alan() adında bir fonksiyon içermesine rağmen, alt sınıfların bu fonksiyonu ezmesine (override) izin verilir. Böylece alan() fonksiyonu, her alt sınıfta şeklin türüne bağlı olarak farklı bir işlev görür ve çok biçimlilik yapısı uygulanmış olur.

**Ders 21 – Referans Yapısı:**

Dart dilinde, sınıfların birden fazla arayüz (interface) uygulamasına olanak tanıyan bir dil yapısı bulunmaktadır. Arayüzler, bir sınıfın uygulaması gereken yöntemlerin ve özelliklerin tanımlandığı bir sözleşmedir. Sözleşme yerine getirilirse, o sınıf, o arayüzü uygulamış olur.

Aşağıdaki örnek, bir arayüzün nasıl kullanılabileceğini gösterir:

abstract class Seyirlik {

  void kalkisYap();

  void havalan();

  void durumGoster();

}

class Ucak implements Seyirlik {

  @override

  void kalkisYap() {

    print('Uçak kalkış yaptı.');

  }

  @override

  void havalan() {

    print('Uçak havalanıyor.');

  }

  @override

  void durumGoster() {

    print('Uçak havada.');

  }

}

class Helikopter implements Seyirlik {

  @override

  void kalkisYap() {

    print('Helikopter kalkış yaptı.');

  }

  @override

  void havalan() {

    print('Helikopter havalanıyor.');

  }

  @override

  void durumGoster() {

    print('Helikopter havada.');

  }

}

void main() {

  var seyirlikler = [Ucak(), Helikopter()];

  for (var seyirlik in seyirlikler) {

    seyirlik.kalkisYap();

    seyirlik.havalan();

    seyirlik.durumGoster();

  }

}

Dart dilinde, sınıfların birden fazla arayüz (interface) uygulamasına olanak tanıyan bir dil yapısı bulunmaktadır. Arayüzler, bir sınıfın uygulaması gereken yöntemlerin ve özelliklerin tanımlandığı bir sözleşmedir. Sözleşme yerine getirilirse, o sınıf, o arayüzü uygulamış olur.

Aşağıdaki örnek, bir arayüzün nasıl kullanılabileceğini gösterir:

dartCopy code

abstract class Seyirlik { void kalkisYap(); void havalan(); void durumGoster(); } class Ucak implements Seyirlik { @override void kalkisYap() { print('Uçak kalkış yaptı.'); } @override void havalan() { print('Uçak havalanıyor.'); } @override void durumGoster() { print('Uçak havada.'); } } class Helikopter implements Seyirlik { @override void kalkisYap() { print('Helikopter kalkış yaptı.'); } @override void havalan() { print('Helikopter havalanıyor.'); } @override void durumGoster() { print('Helikopter havada.'); } } void main() { var seyirlikler = [Ucak(), Helikopter()]; for (var seyirlik in seyirlikler) { seyirlik.kalkisYap(); seyirlik.havalan(); seyirlik.durumGoster(); } }

Bu örnekte, **Seyirlik** adında bir arayüz (interface) tanımlanmıştır. Arayüzde **kalkisYap()**, **havalan()** ve **durumGoster()** adında üç farklı fonksiyon tanımlanmıştır.

Ardından **Ucak** ve **Helikopter** adında iki farklı sınıf tanımlanmıştır. Her iki sınıf da **Seyirlik** arayüzünü uygular. Yani, **Ucak** ve **Helikopter** sınıfları, **kalkisYap()**, **havalan()** ve **durumGoster()** fonksiyonlarını uygulamak zorundadırlar.

Ana fonksiyonda (**main**), **Ucak** ve **Helikopter** sınıflarından oluşan bir dizi oluşturulur. Bu dizideki her bir sınıf, sırasıyla **kalkisYap()**, **havalan()** ve **durumGoster()** fonksiyonlarını çağırır.

Bu örnekte, **Seyirlik** arayüzü, farklı sınıfların aynı sözleşmeyi yerine getirmesini sağlar. Bu, daha modüler ve esnek bir kod yazımına olanak tanır.