

Modul III. Lesson 8

Records and Pattern

Repeat the previous lesson

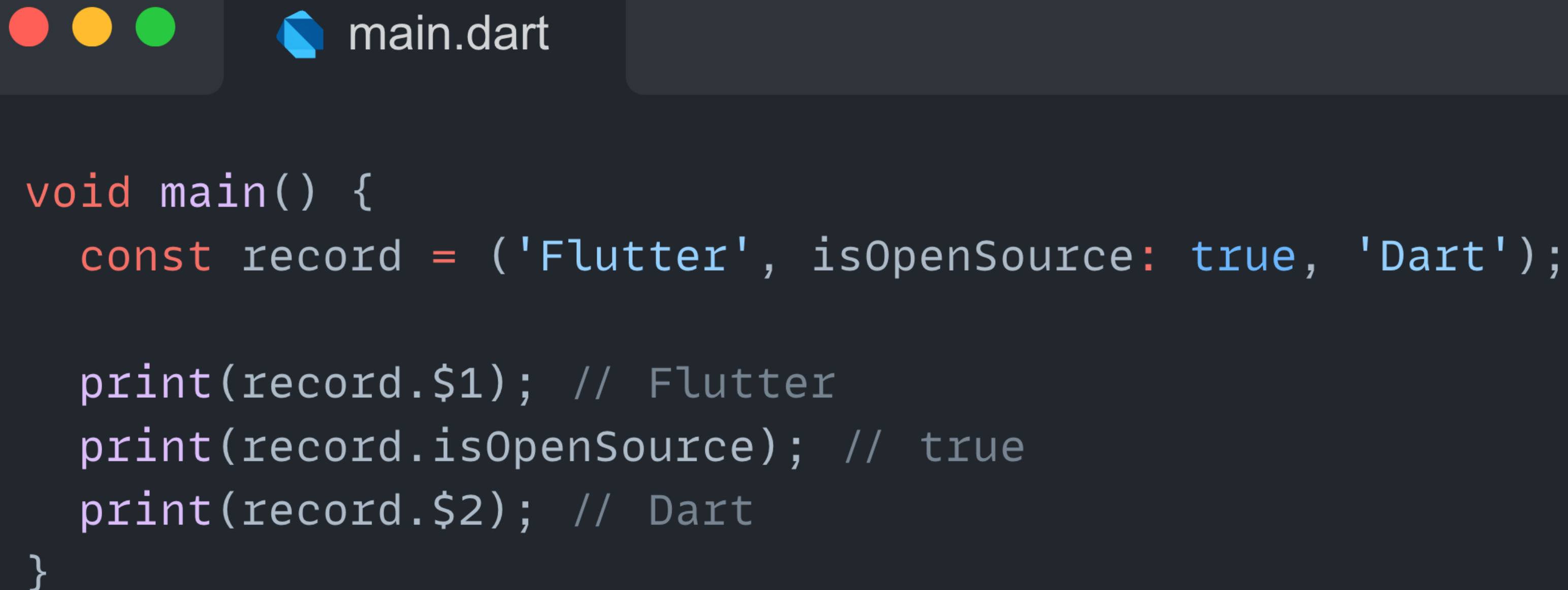
- Error and Exception
- Stack
- Stack Traces
- Call Stack
- Debugging
- BreakPoint
- Running Debug Mode
- Assertion
- Handling Exceptions
- Try / Catch / on / Finally
- Writing Custom Exception
- Throwing Exception
- Rethrow
- Error Handling

Plan

- Records
- Pattern

Records

- Rekord bu bir nechta turdagi qiymatlarni yagona ob'ektga birlashtiradigan "anonim sinf" turidir. U bir qator maydon (field) larga ega bo'lib, ular nomlangan (named) yoki pozitsion (positional) bo'lishi mumkin, va uning sintaksisi funksiya argumentlarining ro'yxatiga o'xshaydi.



```
void main() {  
  const record = ('Flutter', isOpenSource: true, 'Dart');  
  
  print(record.$1); // Flutter  
  print(record.isOpenSource); // true  
  print(record.$2); // Dart  
}
```

- Rekord yaratilgandan so'ng, uning maydonlari getterlar yordamida olinadi. Nomlangan maydonlar o'zlarining nomlari bilan bir xil getterlarni taqdim etadi, pozitsion maydonlar esa tartib raqami bilan birga dollar belgisidan keyin joylashgan getterlarni taqdim etadi.
- `isOpenSource` nomli getter bo'lib, `\$2` esa pozitsion getter hisoblanadi:


```
print(record.isOpenSource); // has the same name as the named field  
print(record.$2); // '2' refers to the second positional field in the list
```

- Pozitsion maydonlar maydonlar ro'yxatidagi 1-asosli indeks pozitsiyasiga mos kelgan dollar belgisi bilan olinadi. Masalan, `\$1` ro'yxatdagi birinchi pozitsion maydonga murojaat qiladi (agar mavjud bo'lsa).

- Rekordlar ichki tuzilishda qurilishi, kengaytirilishi, aralashtirilishi yoki amalga oshirilishi mumkin bo'lmagan Record sinfi orqali ifodalanadi.

```
const record = (1.2, name: 'abc', true, count: 3);
```

- Yuqorida keltirilgan rekord deklaratsiyasi ichki ravishda quyidagi sinf kabi ko'rinadigan Record sinfiga aylantiriladi (bu oddiygina tushuntirish uchun keltirilgan misol):



```
class _ extends Record {  
  double get $1;  
  String get name;  
  bool get $2;  
  int get count;  
}
```

- Barcha rekordlar Record sinfining pastki turlari hisoblanadi va o'zgarmasdir.
- Record sinfida Object sinfidan meros qilingan a'zolaridan boshqa instansiya a'zolari yo'q, hech qanday ommaviy konstruktorlarni taqdim etmaydi va operator== hamda hashCode ni qayta yozadi.
- Bu xuddi Function sinfi barcha funksiya turlari uchun asosiy sinf bo'lgani kabi.
- Qavs ichidagi ifodalarni chalkashtirmaslik uchun, faqat bitta pozitsion maydoni bo'lgan rekordda oxirgi vergul bo'lishi kerak.
- `()` ifodasi maydonlarsiz bo'sh rekordni ko'rsatadi.



main.dart

```
void main() {  
  const number = (1); // The number '1' with parenthesis  
  const record = (1,); // A record with a single field  
  const emptyRecord = (); // A record with no fields  
  
  print(number.runtimeType); // int  
  print(record.runtimeType); // (int)  
  print(emptyRecord.runtimeType); // ()  
}
```

- Rekord turi funksiya turining parametrlar ro'yxatiga o'xshash ko'rinishga ega. Tur qavs ichiga olinadi va unda vergul bilan ajratilgan pozitsion maydonlar bo'lishi mumkin. Shunday qilib, uchta mumkin kombinatsiya mavjud:



main.dart

```
// Faqat pozitsion maydonlar:  
(int, String, bool) myRecord = (1, 'A', true);  
  
// Faqat nomlangan maydonlar:  
({int a, String b, bool c}) myRecord = (a: 1, b: 'A', c: true);  
  
// Pozitsion va nomlangan maydonlar birga:  
(int, {String b, bool c}) myRecord = (1, b: 'A', c: true);
```

- Nomlangan maydonlar doimo pozitsion maydonlardan keyin belgilanishi kerak. Tur annotatsiyalari funksiya parametrlari ro'yxati bilan bir xil tarzda ishlaydi, lekin yagona farq shundaki, "required" so'zi ruxsat etilmaydi. Mana yana bir nechta tegishli ma'lumotlar:
- Bir rekord bir xil nomlangan maydonni bir nechta marta aniqlay olmaydi. Bundan tashqari, nomlangan maydonlar pastki chiziq (_) bilan boshlanishi mumkin emas.
- Rekordlar Object sinfini kengaytirganligi sababli, ular uning barcha metodlarini meros qilib oladi. Natijada, hashCode, runtimeType, noSuchMethod yoki toString kabi nomlarga ega bo'lgan nomlangan maydonlar bo'lishi mumkin emas.

- Rekordning ish vaqtidagi turi uning maydonlarining ish vaqtidagi turiga bog'liq.

```
(Object, num, bool) record = (1, 2, true);  
print(record is (Object, int, bool)); // true  
print(record is (int, int, bool)); // true
```

- Amaliy jihatdan, rekordlar tez-tez funksiyalardan bir nechta qiymatlarni qaytarish uchun ishlatiladi. Ular "bir vaqtda sinflarni yaratish" orqali bir nechta qiymatlarni bitta ob'ektga birlashtira oladilar.



main.dart

```
Future<(double, double, DateTime)> computeGeoPosition() async {  
  final position = await currentPosition();  
  return (position.latitude, position.longitude, DateTime.now());  
}
```

- Funksiya turli turlardagi bir nechta qiymatlarni "o'rab" olgan yagona ob'ekt (rekord) qaytaradi. Rekord ishlatmasdan ham xuddi shu ishni bajarish mumkin edi, lekin bu ko'proq so'z talab qilgan bo'lar edi.



 geo_position_data.dart

```
class GeoPositionData {  
  final double latitude;  
  final double longitude;  
  final DateTime date;  
  const GeoPositionData(this.latitude, this.longitude, this.date);  
  // Missing 'operator==', 'hashCode' and 'toString' overrides!  
}  
  
Future<GeoPositionData> computeGeoPosition() async {  
  final position = await currentPosition();  
  return GeoPositionData(  
    position.latitude,  
    position.longitude,  
    DateTime.now(),  
  );  
}
```


- Rekord xatti-harakatlariga to'liq mos kelish uchun, `GeoPositionData` ham `operator==`, `hashCode` va `toString` metodlarini qayta yozishi kerak edi.
- Tegishli testlar va texnik xizmat ko'rsatish harakatlarini hisobga olmagan holda, kod yanada ko'proq so'z talab qilgan bo'lar edi.
- Buning o'rniga, rekordlardan foydalangan holda, biz Dartga ikki yoki undan ko'p qiymatlarni o'zgarmas ob'ektda qulay tarzda o'rash uchun "bir vaqtda sinflarni" (classes on the fly) yaratishga imkon beramiz.

Exercise

1. x va y koordinatalari uchun fieldlar bilan 2D fazoda nuqta uchun record hosil qiling. Yangi nuqtani yaratadigan va koordinatalarni chop etadigan funksiyani yozing.
2. Nom va narx uchun fieldlar bilan ikki mahsulot uchun recordlarni hosil qiling. Ikkala mahsulotni narxiga qarab taqqoslaydigan va arzonrog'ini qaytaruvchi funksiyani yozing.

Pattern

Pattern

- Dart tilida "patterns" (namunalar) bu ma'lumotlar bilan yanada samarali ishlash usullaridir. Ular sizga murakkab ma'lumotlarni, masalan bir recordda ko'p qiymatli ma'lumotlarni oddiy qismlarga ajratishga imkon beradi.
- Misol uchun, agar sizda bir string, list va boolean qiymatlarini o'z ichiga olgan records bo'lsa, siz ularni alohida o'zgaruvchilarga osonlik bilan ajrata olasiz. Bu jarayon "destrukturing" deyiladi. Bu go'yo bir qutidagi narsalarni chiqarib, har birini o'z o'rniga qo'yish kabi bir narsa.
- Siz listlar yoki maplardan to'g'ridan-to'g'ri qiymatlarni bir-biriga qo'shishingiz mumkin, har bir elementni alohida-alohida olishingiz shart emas. Bu ma'lumotlar tuzilmasi bilan ishlashni soddalashtiradi va kodingizni toza va tushunarli qiladi.

- Dart, expression va statementlarni qo'llab-quvvatlash bilan birga, "pattern"ni ham qo'llab-quvvatlaydi.
- Siz "pattern"ni murakkab ma'lumotlarni tekshirish yoki undan ma'lumotlarni chiqarish yo'llarini taqdim etuvchi til xususiyati sifatida ko'rishingiz mumkin.
- Amalda, ular "odatdagidan kamroq kod yordamida obyektlar va ma'lumotlar ustida ishlash uchun ishlatiladi".
- "Patterns" tilning turli sohalarida qo'llanilishi mumkin, masalan:
 - O'zgaruvchi deklaratsiyalarida.
 - Switch statement va expressionlarida.
 - If-case statementlarida.

Variable declaration

- Agar sizda record turida saqlangan bir qator qiymatlar bo'lsa, siz ularni alohida o'zgaruvchilarga ajratish orqali "record destructuring" pattern'ini ishlatishingiz mumkin.

```
final (a, b, c) = ("str", [1, 2], false);  
//      ^^^^^^      ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^  
//    destructuring    record object  
print(a.runtimeType); // String  
print(b.runtimeType); // List<int>  
print(c.runtimeType); // bool
```

- Amaliy jihatdan, "destructuring" har bir obyekt qiymatini yangi o'zgaruvchiga biriktirishni anglatadi. Bu holatda, biz a, b va c o'zgaruvchilarni recordning uchta qiymatini saqlash uchun yaratdik. Bu ikkita turli narsalarni eslab qoling:



main.dart

```
// This is destructuring
final (a, b) = (1, 2);
print(a.runtimeType); // int
print(b.runtimeType); // int
```

```
// This is assigning a 'Record' object to a variable
final record = (1, 2);
print(record.runtimeType); // (int, int)
```

- "Destructuring" har bir obyekt yoki record qiymati uchun bittadan o'zgaruvchini yaratganda ishlaydi.
- Ikkinchi holat yozuvni "destructure" qilmaydi: bu faqat obyektни o'zgaruvchiga biriktiradi.
- Siz listlar va maplar kabi to'plamlarni ham "destructure" qilishingiz mumkin.

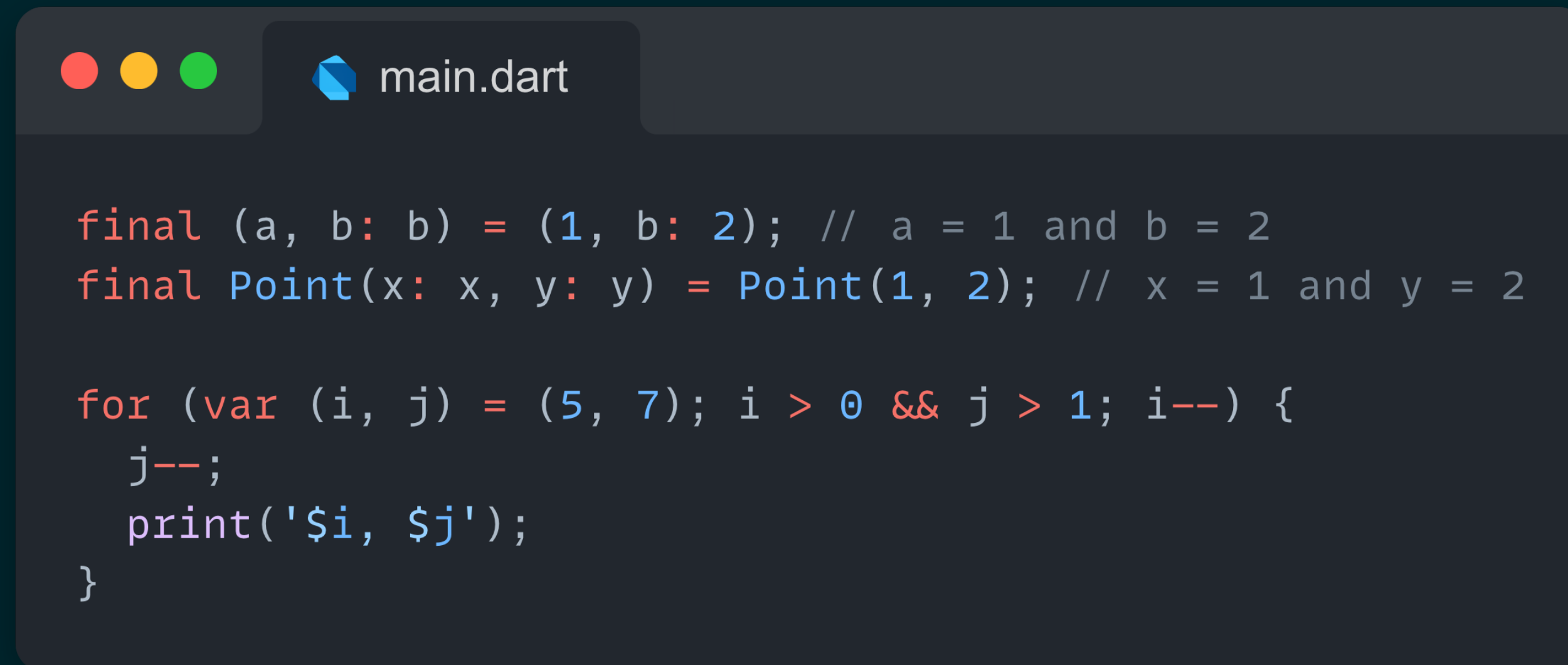


main.dart

```
final [a, b, c] = <int>[4, 8, 12];  
print('sum = ${a + b + c}'); // sum = 24
```

```
final {'one': one, 'two': two} = <String, int>{'one': 1, 'two': 2};  
print('sum = ${one + two}'); // sum = 3
```

- List yoki Map interfeysini amalga oshiradigan boshqa har qanday sinf ham shu tarzda "destructured" qilinishi mumkin. Mana yana bir nechta "destructuring" misollar:



```
final (a, b: b) = (1, b: 2); // a = 1 and b = 2
final Point(x: x, y: y) = Point(1, 2); // x = 1 and y = 2

for (var (i, j) = (5, 7); i > 0 && j > 1; i--) {
  j--;
  print('$i, $j');
}
```

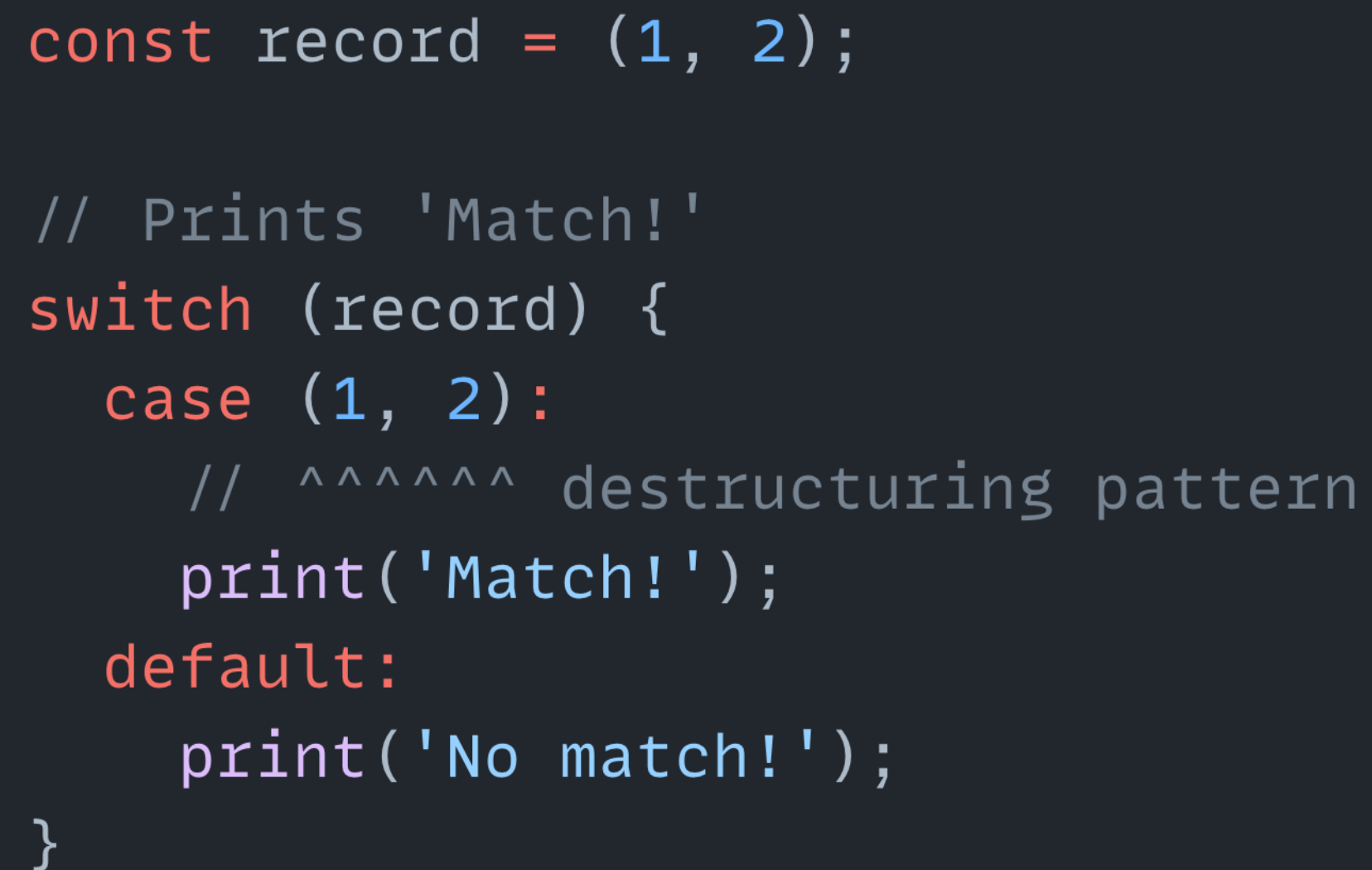
- Dartda o'zgaruvchilarni e'lon qilishda "patterns" dan foydalanishingiz mumkin, bu turli "for" bayonotlarida va shuningdek turli holatlarda foydali bo'lishi mumkin.

Exercise

1. Kun, oy, yilni ifodalovchi recordni qabul qiluvchi funksiya hosil qiling va "KK/OO/YYYY" formatida sanani chop etish uchun patterndan foydalaning.

Switch statement

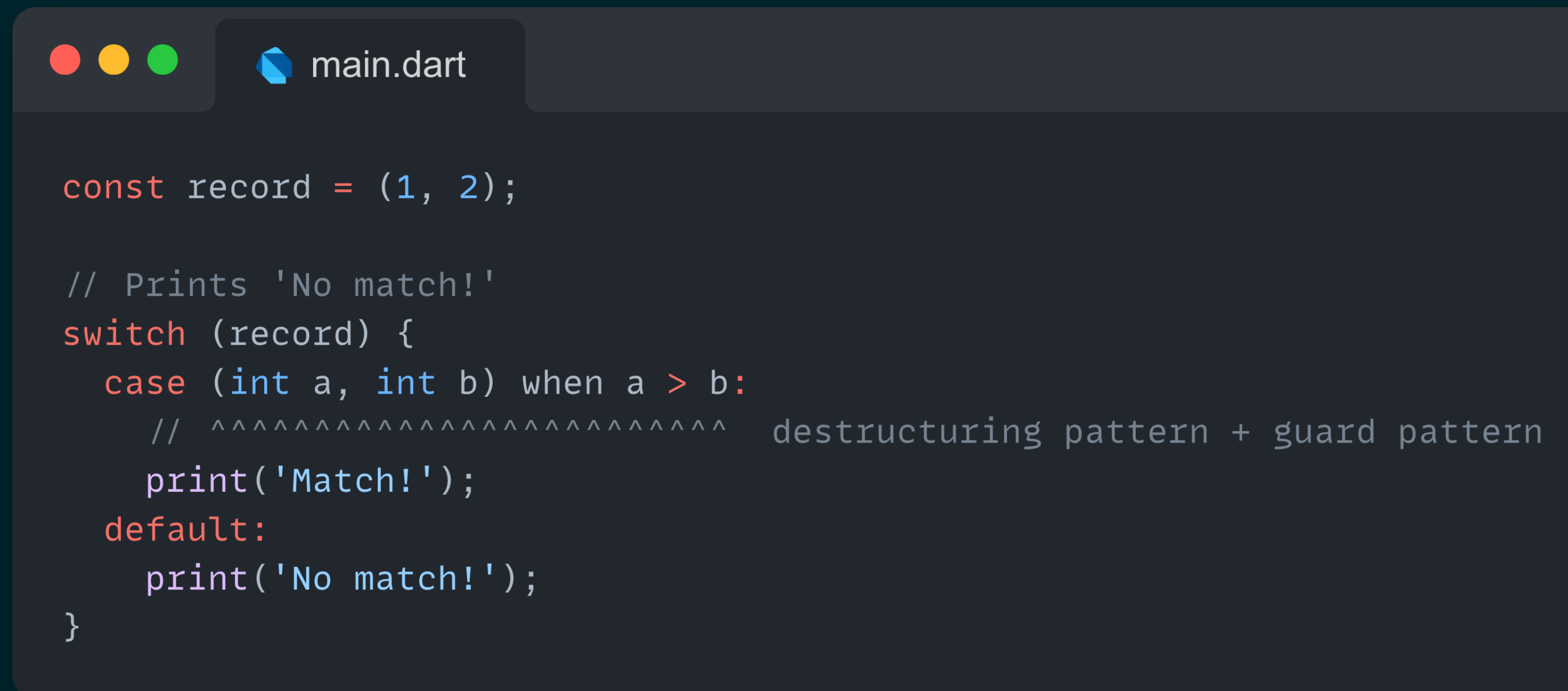
- Siz switch statement yordamida to'g'ridan-to'g'ri holatlarda turli pattern'larni ishlatishingiz mumkin. Record obyektiga mos keladigan pattern'ni case deklaratsiyasida darhol ajratib olishingiz mumkin:



```
const record = (1, 2);

// Prints 'Match!'
switch (record) {
  case (1, 2):
    // ^^^^^ destructuring pattern
    print('Match!');
  default:
    print('No match!');
}
```

- Bu misolda, birinchi case record obyektini ajratib oladi va u mos kelishini tekshiradi yoki yo'q. Shuningdek, mos kelgandan keyin when kalit so'zi bilan qo'shimcha shart qo'shish imkoniyati ham mavjud.



```
const record = (1, 2);

// Prints 'No match!'
switch (record) {
  case (int a, int b) when a > b:
    // ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ destructuring pattern + guard pattern
    print('Match!');
  default:
    print('No match!');
}
```

- Record ajratib olindan keyin, when kalit so'zi a ning b dan katta ekanligini tekshirish uchun ishlatiladi. Bu misolda, a kichikroq ekanligi sababli birinchi filial rad etiladi. Dastur keyingi holatlarga o'tadi va default ga yetguncha davom etadi, bu esa har doim mos keladi.

Switch expression

- Switch statement ko'pincha bir qator shartli ifodalarni baholash uchun ishlatiladi. Bu, uzoq if-else bloklar seriyasiga nisbatan o'qilishi osonroq alternativadir. Masalan, ushbu kodni ko'rib chiqing:

```
enum Number {  
    one,  
    two,  
    three,  
}  
  
int myFunction(Number number) {  
    switch (number) {  
        case Number.one:  
            return 1;  
        case Number.two:  
            return 2;  
        case Number.three:  
            return 3;  
    }  
}
```

- Kodni switch expressiondan foydalanib, teng shaklda qayta yozish mumkin. Masalan:



main.dart

```
int myFunction(Number number) {  
  return switch (number) {  
    Number.one => 1,  
    Number.two => 2,  
    Number.three => 3,  
  };  
}
```

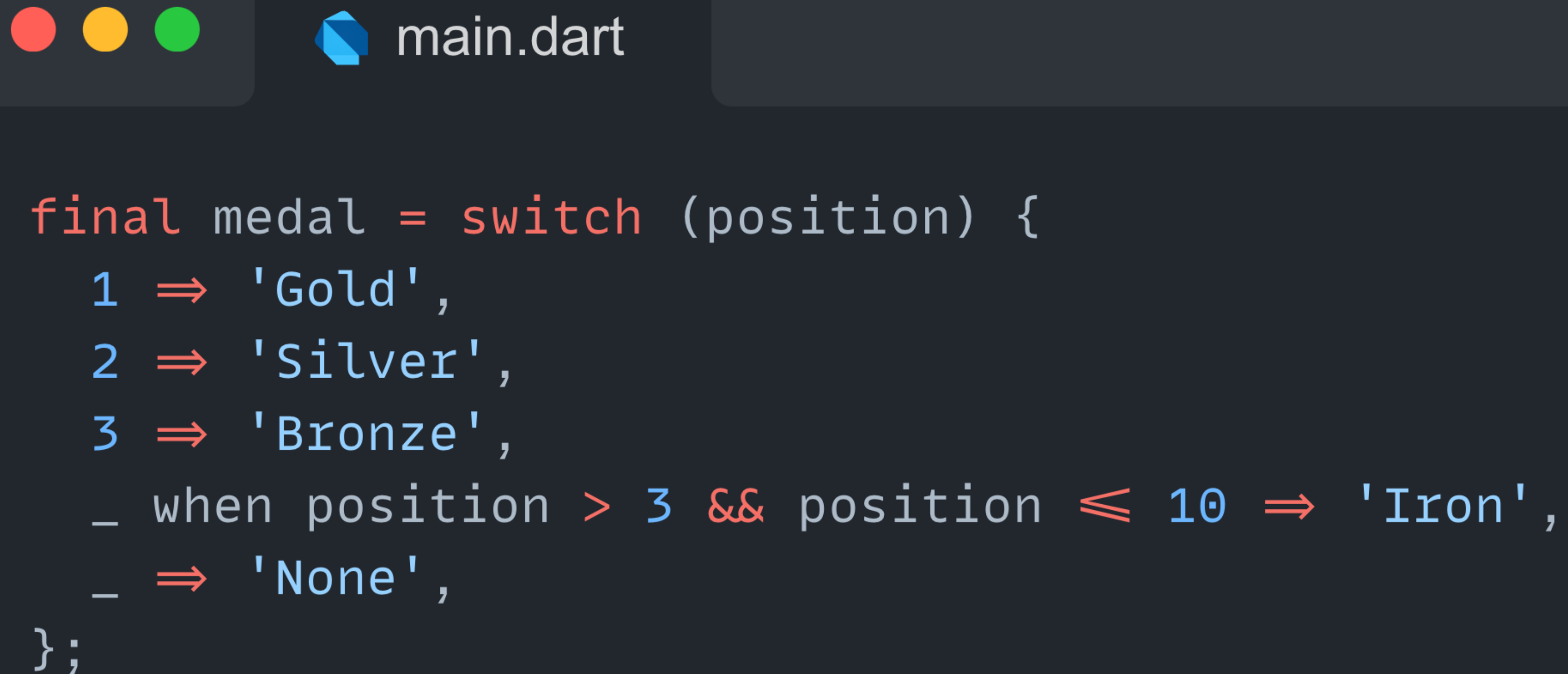
- Switch expressionni qisqacha takrorlab ketamiz:
- Pastki chiziq - standart holatni ifodalaydi va odatda, u har doim eng quyida joylashishi kerak.



main.dart

```
// Assume that 'position' is an 'int'  
final medal = switch (position) {  
  1 => 'Gold',  
  2 => 'Silver',  
  3 => 'Bronze',  
  _ => 'None', // The fallback 'default' clause  
};
```

- Har qanday switch expression ichida "when" sharti ishlatilishi mumkin. Masalan, bu kodda, agar pozitsiya 4 dan 10 gacha bo'lsa, "Iron" beriladi:



```
final medal = switch (position) {  
  1 => 'Gold',  
  2 => 'Silver',  
  3 => 'Bronze',  
  _ when position > 3 && position ≤ 10 => 'Iron',  
  _ => 'None',  
};
```

- Agar pozitsiya manfiy bo'lsa yoki 10 dan katta bo'lsa, "None" qaytariladi.

Exercise

1. Har xil turdagi bildirishnomalar (masalan, Email, SMS, PushNotification) uchun enum hosil qiling. Bildirishnomani o'z ichiga olgan recordni qabul qiluvchi va bildirishnoma turiga qarab turli xabarlarni chop etuvchi funksiyani yozing.

If-case statement

- Pattern if statementlari ichida ham paydo bo'lishi mumkin. Tuzilmani ajratish va holat patternlarini birlashtirish orqali, if statement switch xususiyatlariga ega bo'ladi. Misol uchun, quyidagi uzun kodni ko'rib chiqing:

```
main.dart

final json = {
  'skills': ['Dart', 'Flutter']
};

// Lots of code
if (json.containsKey('skills') && json.length == 1) {
  final skills = json['skills'];

  if (skills is List<String> && skills.length == 2) {
    final first = skills.first;
    final last = skills.last;
    print('1st $first | 2nd $last');
  }
}
```


- Bu kod ma'lum bir kalit mavjudligini va ro'yxatda aniq ikkita String elementi borligini ta'minlash uchun bir qator tekshiruvlardan o'tkaziladi. Pattern mosligidan foydalanib, yuqoridagi misolimizdan ko'ra qisqaroq shaklda ifodalashimiz mumkin:

```
main.dart

final json = {
  'skills': ['Dart', 'Flutter']
};

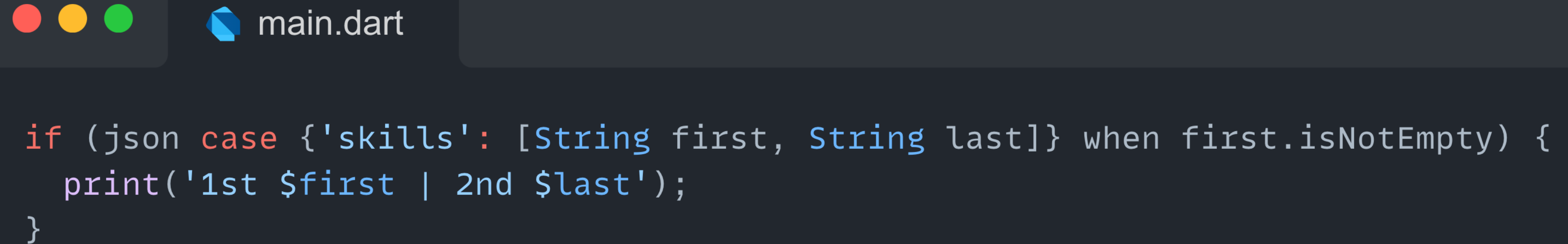
if (json case {'skills': [String first, String last]}) {
  // ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ destruct. pattern
  print('1st $first | 2nd $last');
}
```

- case bo'limi rekord yoki to'plamning tuzilmasini ajratib, mosligini tekshiradi. Misolda, if bayonoti json obyektining mapga (skills deb nomlangan kalit) va qiymat sifatida ikki stringdan iborat listga mosligini tekshiradi.
- Agar null elementi mumkin bo'lsa, biz ? operatori yordamida rekord yoki to'plamni tuzilmasini ajratamiz. Misolda, agar biz nullable String? turidan foydalanmasak, moslik muvaffaqiyatsiz bo'ladi.

```
final json = {
  'skills': ['Dart', null] // nullable element
};

if (json case {'skills': [String first, String? last]}) {
  print('1st $first | 2nd $last'); // matches 'null'
}
```

- Shartlarni qo'llash ham mumkin. Masalan:

A code editor window with a dark theme. The title bar shows three colored circles (red, yellow, green) and a tab labeled 'main.dart' with a blue icon. The code is written in Dart and uses syntax highlighting: 'if' is red, 'json' is blue, 'case' is red, 'String' is blue, 'first' and 'last' are blue, 'when' is blue, 'first.isEmpty' is blue, 'print' is purple, and '1st', '2nd' are blue.

```
if (json case {'skills': [String first, String last]} when first.isNotEmpty) {  
  print('1st $first | 2nd $last');  
}
```

- Yuqoridagi misol bodysi ichida bitta holat(case) mavjud bo'lgan switch bayonoti ekvivalentidir.

Exercise

1. Mahsulotni ifodalovchi JSON obyektini qabul qiluvchi va id, nom, narx fieldlarini ajratib olish uchun patterdan foydalanadigan Dart funksiyasini yarating va bu fieldlarni o'z ichiga olgan recordni qaytaring.

Summary

Interview Questions

- Dart dasturlash tilida record nima?
- U qanday sintaksisda e'lon qilinadi?
- Recordning fieldlari nima?
- Recordda fieldlarning nechta turi mavjud?
- Recordni e'lon qilayotganda required kalit so'zini ishlatish mumkinmi?
- Recordning named fieldlarida qanday nom va belgilardan foydalanish mumkin emas?
- Amaliy jihatdan recordni qayerlarda qo'llashimiz foydali bo'ladi?

- Dart dasturlash tilida pattern nima?
- Pattern “destructuring” nima?
- Dart tilida patterndan qayerlarda foydalanamiz?
- List va Map turidagi o’zgaruvchilarni ham destructure qilish mumkinmi?

Homework

1. "name", "age" va "city" keylari bilan mapni qabul qiluvchi Dart funksiyasini yarating. Bu qiymatlarni ajratib olish uchun patterndan foydalaning va ularni record sifatida qaytaring.
2. To'liq ismni ifodalovchi recordni qabul qiluvchi Dart funksiyasini yozing (ism, otasining ismi, familiya) va pattern yordamida ismdan har bir komponentni alohida ajratib chiqarish va chop etish uchun foydalaning.
3. Telefon raqami va email manzili uchun ixtiyoriy fieldlar bilan foydalanuvchi profili uchun recordni belgilang. Yangi foydalanuvchi profilini yaratish va profildagi tafsilotlarni chop etuvchi, ixtiyoriy fieldlar null bo'lishi mumkin bo'lgan holatlar bilan shug'ullanadigan funksiyani yozing.

4. Har xil shakllar (masalan, Aylana, Kvadrat, To'rtburchak) ro'yxatini qabul qiluvchi va barcha shakllarning umumiy maydonini qaytaruvchi Dart funksiyasini yarating. Har bir shaklni aniqlash va uning maydonini hisoblash uchun pattern matchingdan foydalaning.
5. Universitet uchun ichki recordlarni belgilaydigan Dart dasturini yozing. Tashqi record universitetni nomi va bo'limlar ro'yxati bilan ifodalashi kerak. Har bir bo'lim bo'lim nomi va professorlar ro'yxati uchun fieldlar bilan record bo'lishi kerak. Har bir professor professorning ismi va ular o'qitadigan kurslar ro'yxati uchun fieldlar bilan record bo'lishi kerak. Pattern matchingdan foydalanib, yangi bo'limlar, professorlar va kurslar qo'shish funksiyalarini amalga oshiring.
6. Turli ma'lumot turlari (butun sonlar, satrlar va listlar) listini qabul qiluvchi va pattern matching yordamida ularni uchta alohida listga ajratadigan Dart funksiyasini amalga oshiring: butun sonlar uchun, satrlar uchun va listlar uchun. Bu listlarni record sifatida qaytaring.

Resource

- <https://dart.dev/language/records>
- <https://dart.dev/language/patterns>
- <https://dart.dev/language/pattern-types>

Q&A

Thank you for your time!