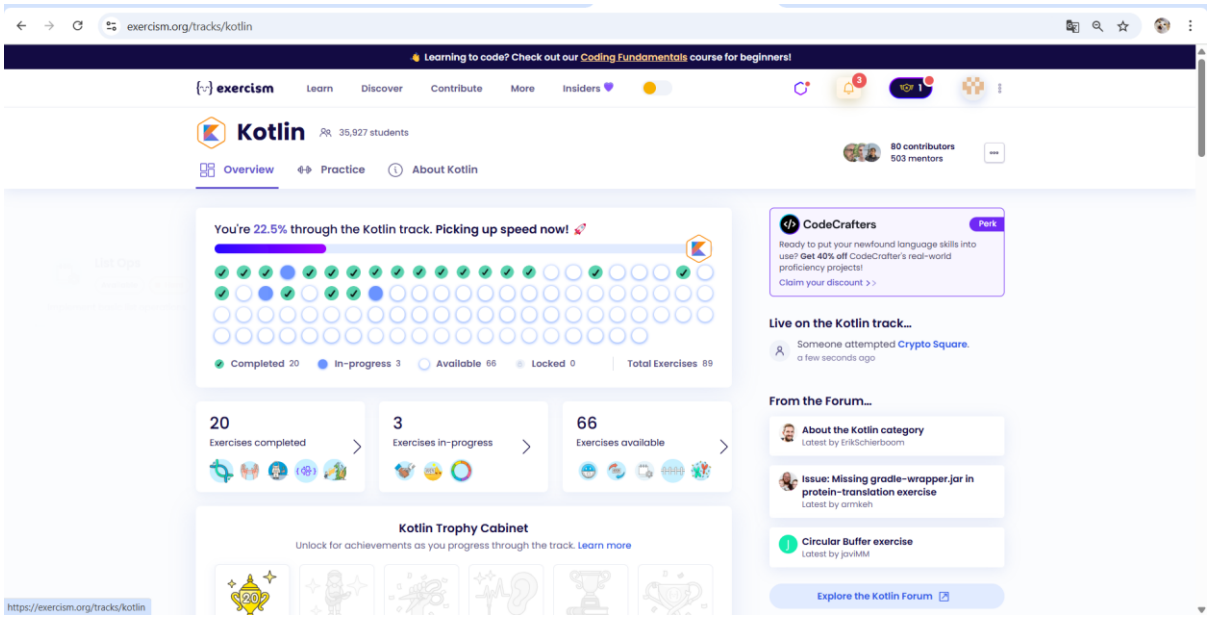
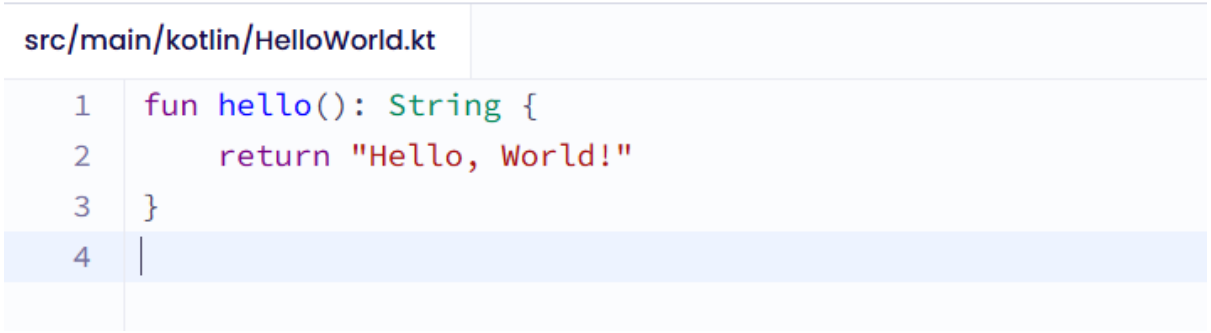
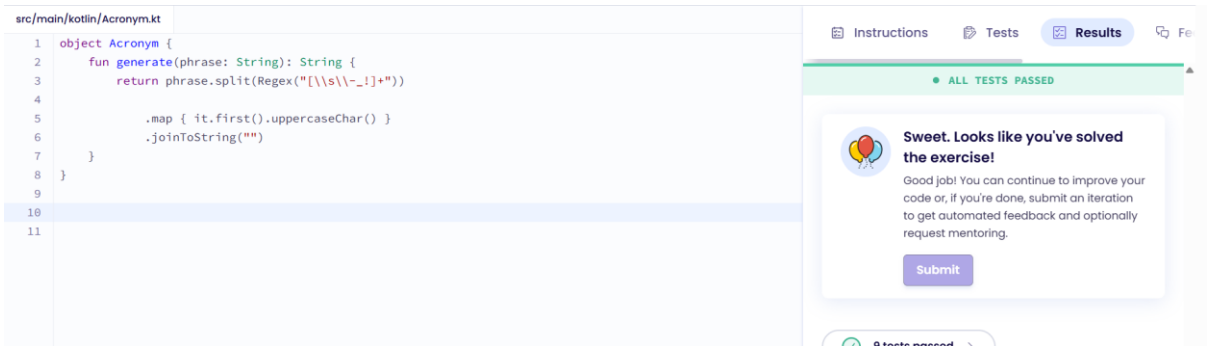






Đoạn code này định nghĩa một object Kotlin tên "Acronym" với một hàm "generate" để tạo từ viết tắt từ một cụm từ. Cụ thể:

- Hàm nhận vào một chuỗi (phrase) và trả về một chuỗi viết tắt
- `split(Regex("[\\s\\-_!]+"))` chia chuỗi thành các từ dựa trên khoảng trắng, dấu gạch ngang, gạch dưới, dấu chấm than

- `map { it.first().uppercaseChar() }` lấy ký tự đầu tiên của mỗi từ và chuyển thành chữ hoa
- `joinToString("")` ghép các ký tự đầu tiên lại thành một chuỗi không có ký tự phân cách

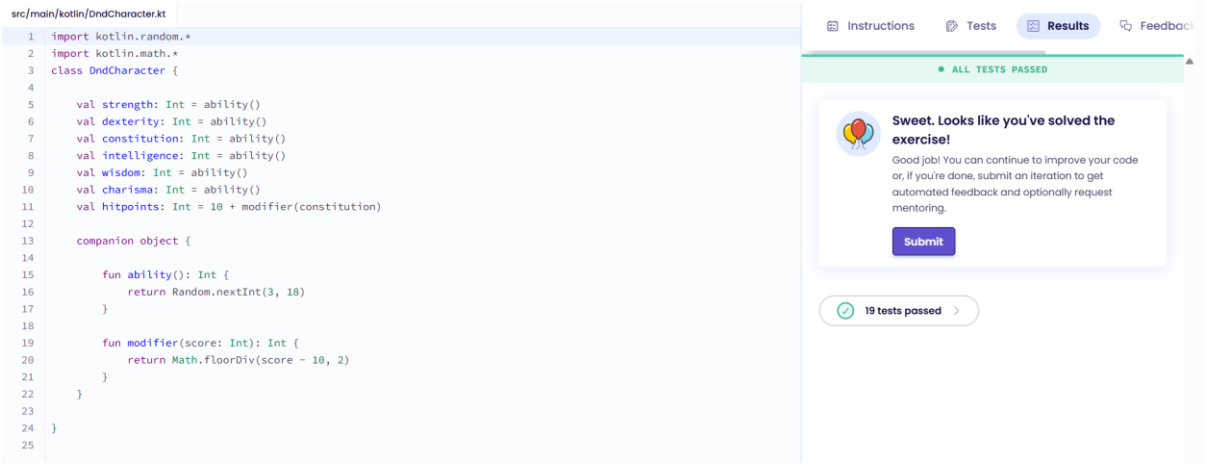
Đoạn code này định nghĩa một object Kotlin tên "ArmstrongNumber" với một hàm "check" để kiểm tra xem một số có phải là số Armstrong hay không. Cụ thể:

- Hàm nhận vào một số nguyên (input) và trả về kết quả Boolean
- input.toString().map{it.digitToInt()} chuyển số thành chuỗi rồi tách thành mảng các chữ số
- val power = digits.size xác định số lượng chữ số của số đầu vào
- digits.sumOf{it.toDouble().pow(power).toInt()} tính tổng của mỗi chữ số lũy thừa với số lượng chữ số
- Cuối cùng, so sánh tổng này với số ban đầu để xác định có phải số Armstrong không

Số Armstrong là số mà tổng của các chữ số, mỗi chữ số được lũy thừa với số lượng chữ số, bằng chính số đó.

Đoạn code này định nghĩa một object Kotlin tên "CryptoSquare" với một hàm "ciphertext" để mã hóa văn bản theo phương pháp mã hóa hình vuông. Cụ thể:

- Hàm nhận vào một chuỗi văn bản thường (plaintext) và trả về chuỗi đã mã hóa
- `plaintext.lowercase().filter { it.isLetterOrDigit() }` chuyển văn bản thành chữ thường và loại bỏ tất cả ký tự không phải chữ cái hoặc số
- `.also { if (it.isEmpty()) return "" }` kiểm tra nếu chuỗi rỗng thì trả về chuỗi rỗng
- `sqrt(normalized.length.toDouble()).roundToInt()` tính số hàng dựa trên căn bậc hai của độ dài chuỗi
- Tính số cột bằng số hàng, nếu không đủ chỗ thì tăng thêm 1
- `normalized.padEnd(rows * columns, ' ')` thêm khoảng trắng vào cuối chuỗi để đủ kích thước ma trận
- `.windowed(columns, columns)` chia chuỗi thành các hàng với độ dài bằng số cột
- Cuối cùng, tạo chuỗi mã hóa bằng cách đọc theo cột, cách nhau bởi dấu cách



Đoạn code này định nghĩa một class Kotlin tên "DndCharacter" để tạo nhân vật trong trò chơi Dungeons & Dragons. Cụ thể:

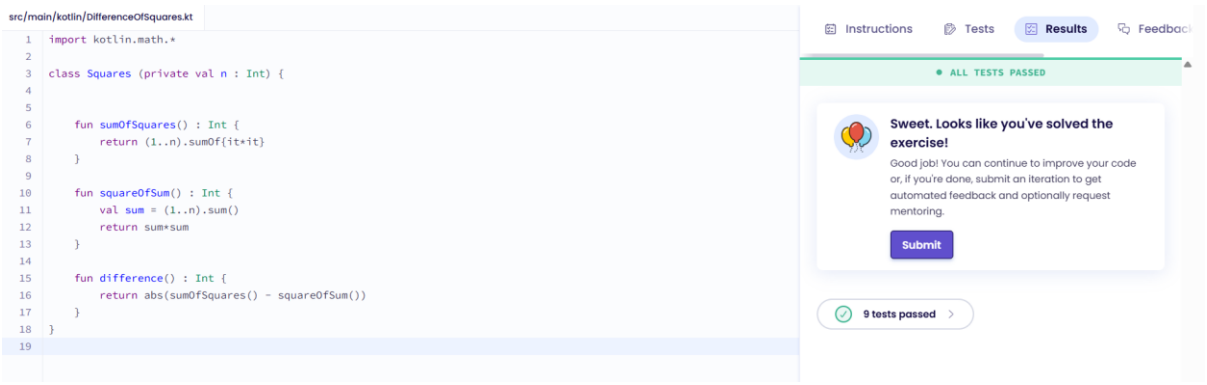
- Class này có 7 thuộc tính: 6 chỉ số cơ bản (strength, dexterity, constitution, intelligence, wisdom, charisma) và hitpoints
- Mỗi chỉ số cơ bản được khởi tạo bằng phương thức `ability()` để tạo giá trị ngẫu nhiên
- Điểm hitpoints được tính dựa trên công thức: 10 + modifier của constitution

Trong companion object có hai phương thức:

- `ability()`: Tạo một số ngẫu nhiên từ 3 đến 17 (đại diện cho chỉ số thuộc tính của nhân vật)
- `modifier(score)`: Tính toán hệ số điều chỉnh dựa trên công thức (score - 10) / 2, làm tròn xuống

Đoạn code này định nghĩa một object Kotlin tên "Darts" với một hàm "score" để tính điểm trong trò chơi phi tiêu. Cụ thể:

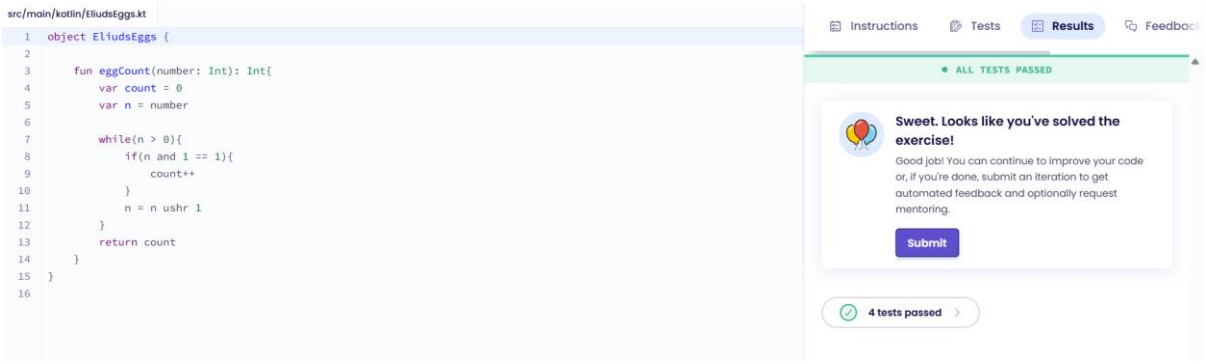
- Hàm nhận vào hai tham số x và y (kiểu Number) đại diện cho tọa độ nơi phi tiêu rơi
- `hypot(abs(x.toDouble()), abs(y.toDouble()))` tính khoảng cách từ điểm (x,y) đến tâm (0,0) bằng định lý Pythagoras
- Biến `do_lech` lưu trữ khoảng cách này (đại diện cho độ lệch so với tâm)
- Hàm trả về điểm số dựa trên khoảng cách:
    - 10 điểm nếu phi tiêu rơi trong vòng tròn bán kính 1 đơn vị
    - 5 điểm nếu phi tiêu rơi trong vòng tròn bán kính 5 đơn vị
    - 1 điểm nếu phi tiêu rơi trong vòng tròn bán kính 10 đơn vị
    - 0 điểm nếu phi tiêu rơi ngoài bảng (khoảng cách > 10)



Đoạn code này định nghĩa một class Kotlin tên "Squares" với ba phương thức để tính toán các giá trị liên quan đến tổng và bình phương. Cụ thể:

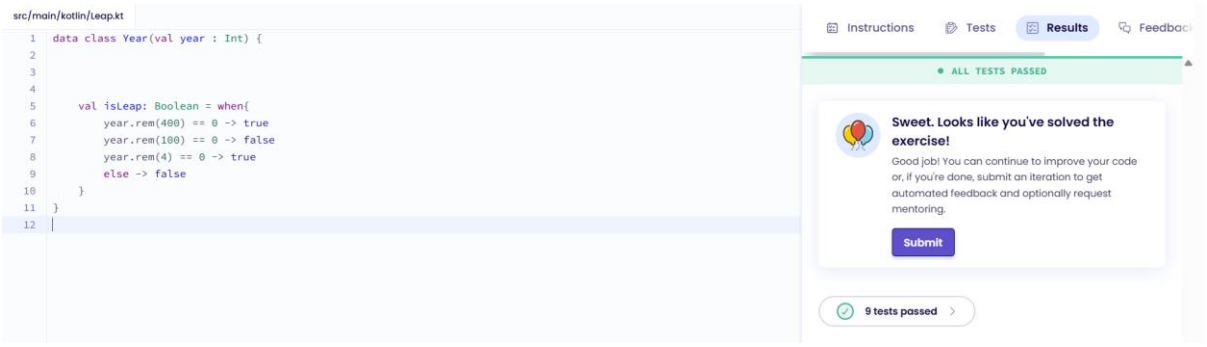
- Class nhận một tham số `n` (kiểu Int) trong constructor để xác định phạm vi tính toán từ 1 đến n
- Phương thức `sumOfSquares()`: Tính tổng các bình phương của các số từ 1 đến n
    - Sử dụng `(1..n).sumOf{it*it}` để tính $1^2 + 2^2 + 3^2 + ... + n^2$
- Phương thức `squareOfSum()`: Tính bình phương của tổng các số từ 1 đến n
    - Đầu tiên tính tổng `(1..n).sum()` để có 1 + 2 + 3 + ... + n
    - Sau đó bình phương tổng này: $(1 + 2 + 3 + ... + n)^2$

- Phương thức `difference()`: Tính hiệu giữa bình phương của tổng và tổng các bình phương
    - Sử dụng `abs()` để lấy giá trị tuyệt đối của hiệu số



Đoạn code này định nghĩa một object Kotlin tên "EliudsEggs" với một hàm "eggCount" để đếm số bit 1 trong biểu diễn nhị phân của một số nguyên. Cụ thể:

- Hàm nhận vào một số nguyên (number) và trả về số lượng bit 1 trong biểu diễn nhị phân của số đó
- Khởi tạo biến `count` bằng 0 để đếm số bit 1
- Khởi tạo biến `n` bằng số đầu vào để thực hiện các phép toán
- Vòng lặp `while(n > 0)` thực hiện cho đến khi n trở thành 0
- Trong mỗi vòng lặp:
    - `n and 1` kiểm tra bit cuối cùng của n có phải là 1 không
    - Nếu là 1, tăng biến count lên 1
    - `n = n ushr 1` dịch phải không dấu n một bit, loại bỏ bit cuối cùng đã kiểm tra
- Cuối cùng trả về biến count là số lượng bit 1

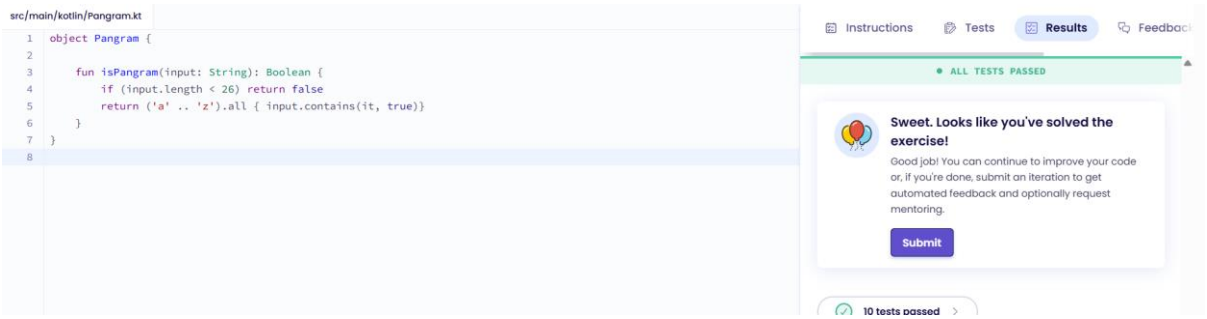


Đoạn code này định nghĩa một data class Kotlin tên "Year" với thuộc tính để kiểm tra năm nhuận. Cụ thể:

- Class nhận một tham số `year` (kiểu Int) trong constructor để đại diện cho một năm
- Thuộc tính `isLeap` (kiểu Boolean) được tính toán ngay khi khởi tạo đối tượng để xác định xem năm đó có phải là năm nhuận hay không
- Việc xác định năm nhuận sử dụng biểu thức `when` với các quy tắc sau:
    - Nếu năm chia hết cho 400 (year.rem(400) == 0) → là năm nhuận (true)
    - Nếu năm chia hết cho 100 (year.rem(100) == 0) → không phải năm nhuận (false)
    - Nếu năm chia hết cho 4 (year.rem(4) == 0) → là năm nhuận (true)
    - Các trường hợp còn lại → không phải năm nhuận (false)

Đây là quy tắc xác định năm nhuận trong lịch Gregorian:

1. Năm nhuận là năm chia hết cho 4
2. Tuy nhiên, năm chia hết cho 100 không phải là năm nhuận
3. Ngoại lệ: năm chia hết cho 400 vẫn là năm nhuận



Đoạn code này định nghĩa một object Kotlin tên "Pangram" với một hàm "isPangram" để kiểm tra xem một chuỗi có phải là pangram hay không. Cụ thể:

- Hàm nhận vào một chuỗi (input) và trả về kết quả Boolean
- Kiểm tra nhanh đầu tiên: `if (input.length < 26) return false` - nếu chuỗi ngắn hơn 26 ký tự, chắc chắn không thể chứa đủ 26 chữ cái trong bảng chữ cái tiếng Anh
- Sau đó sử dụng `('a' .. 'z').all { input.contains(it, true) }` để kiểm tra:
    - `('a' .. 'z')` tạo một dãy các ký tự từ 'a' đến 'z'
    - `.all { ... }` kiểm tra xem tất cả các ký tự trong dãy có thỏa mãn điều kiện không
    - `input.contains(it, true)` kiểm tra xem chuỗi input có chứa ký tự đang xét không, với tham số `true` cho phép so sánh không phân biệt chữ hoa/thường

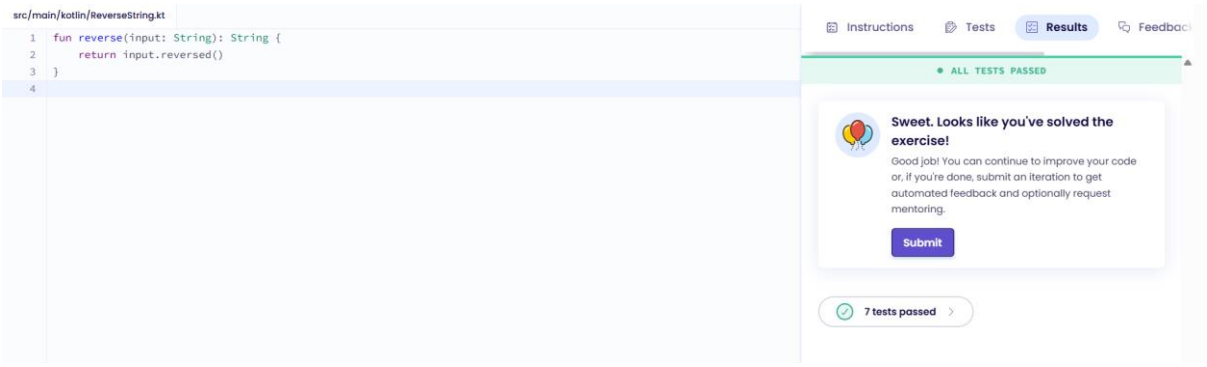
Pangram là một câu chứa tất cả các chữ cái trong bảng chữ cái. Hàm này kiểm tra xem chuỗi đầu vào có chứa đủ 26 chữ cái từ a đến z (không phân biệt hoa thường) hay không.

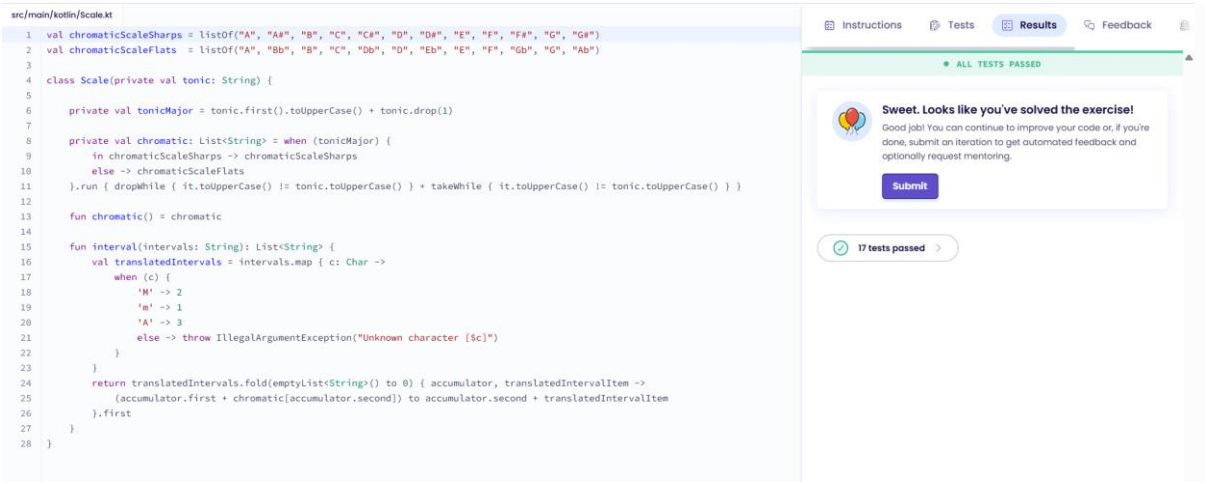
Đoạn code này định nghĩa một enum class "Classification" và một hàm "classify" để phân loại số tự nhiên theo tổng các ước số của nó. Cụ thể:

- Enum class `Classification` định nghĩa ba loại số:
  - `DEFICIENT`: Số thiếu (tổng các ước số nhỏ hơn chính số đó)
  - `PERFECT`: Số hoàn hảo (tổng các ước số bằng chính số đó)
  - `ABUNDANT`: Số dư (tổng các ước số lớn hơn chính số đó)
- Hàm `classify` nhận vào một số tự nhiên và trả về phân loại của số đó:
  - Kiểm tra nếu số đầu vào không phải số tự nhiên dương (≤ 0), ném ngoại lệ IllegalArgumentException
  - Tính `aliquotSum` (tổng các ước số thực sự) bằng cách:
    - Tạo dãy `(1 until naturalNumber)` gồm các số từ 1 đến (naturalNumber-1)
    - Lọc ra các số là ước của naturalNumber bằng `filter { naturalNumber % it == 0 }`
    - Tính tổng các ước số này bằng `.sum()`
  - Trả về phân loại dựa trên mối quan hệ giữa aliquotSum và naturalNumber:
    - Nếu bằng nhau → PERFECT
    - Nếu aliquotSum lớn hơn → ABUNDANT
    - Còn lại → DEFICIENT

Đoạn code này định nghĩa một object Kotlin tên "Raindrops" với một hàm "convert" để chuyển đổi số nguyên thành chuỗi theo quy tắc đặc biệt. Cụ thể:

- Hàm nhận vào một số nguyên (n) và trả về một chuỗi
- Khởi tạo biến `result` là chuỗi rỗng để lưu kết quả
- Kiểm tra và thêm vào result theo các quy tắc:
    - Nếu n chia hết cho 3 (`n % 3 == 0`), thêm "Pling" vào result
    - Nếu n chia hết cho 5 (`n % 5 == 0`), thêm "Plang" vào result
    - Nếu n chia hết cho 7 (`n % 7 == 0`), thêm "Plong" vào result
- Cuối cùng, kiểm tra:
    - Nếu result vẫn là chuỗi rỗng (không thỏa mãn điều kiện nào), trả về chuỗi biểu diễn của n
    - Ngược lại, trả về chuỗi result đã được tạo

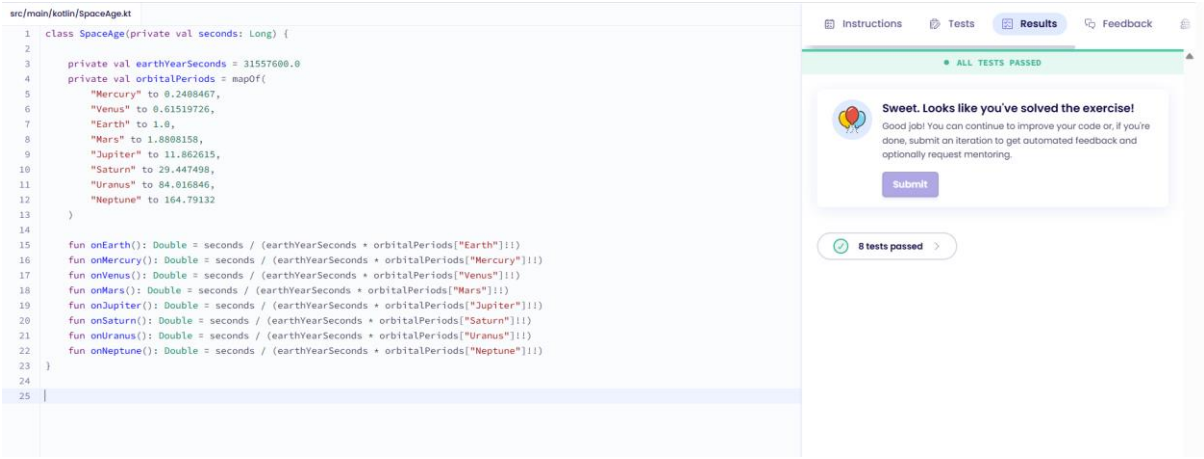




Đoạn code này định nghĩa một class Kotlin tên "Scale" để làm việc với thang âm trong âm nhạc. Cụ thể:

- Hai biến toàn cục `chromaticScaleSharps` và `chromaticScaleFlats` chứa các nốt trong thang âm chromatic với dấu thăng (#) và dấu giáng (b)
- Class `Scale` nhận một tham số `tonic` (kiểu String) trong constructor đại diện cho nốt gốc của thang âm
- Biến `tonicMajor` chuẩn hóa nốt gốc bằng cách viết hoa chữ cái đầu
- Biến `chromatic` xác định thang âm chromatic bắt đầu từ nốt gốc:
    - Chọn thang âm sharps hoặc flats dựa trên nốt gốc
    - Sử dụng `.run { ... }` để tạo thang âm chromatic bắt đầu từ nốt gốc bằng cách:
        - `dropWhile { it.toUpperCase() != tonic.toUpperCase() }` loại bỏ các nốt trước nốt gốc

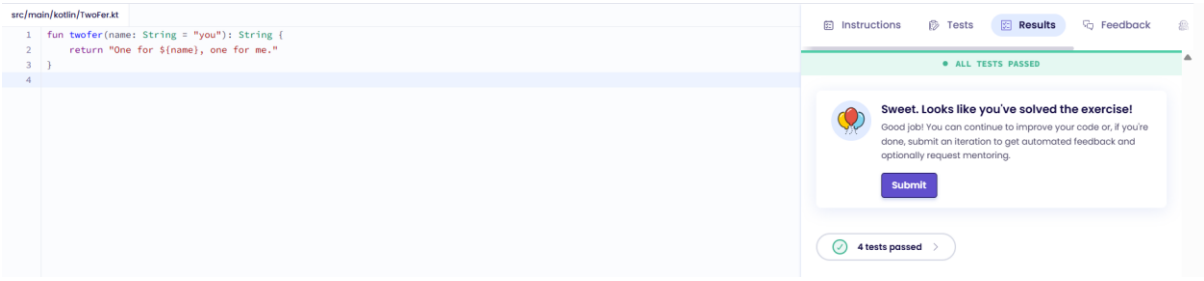
        - `+ takeWhile { it.toUpperCase() != tonic.toUpperCase() }` thêm các nốt từ đầu thang âm đến trước nốt gốc
- Phương thức `chromatic()` trả về thang âm chromatic đã được tạo
- Phương thức `interval(intervals)` tạo thang âm dựa trên các khoảng cách cho trước:
    - Chuyển đổi các ký tự trong chuỗi intervals thành số bước di chuyển:
        - 'M' (Major) → 2 bước
        - 'm' (minor) → 1 bước
        - 'A' (Augmented) → 3 bước
    - Sử dụng `fold` để tích lũy kết quả, bắt đầu với danh sách rỗng và vị trí 0
    - Trong mỗi bước fold, thêm nốt tại vị trí hiện tại vào danh sách kết quả và cập nhật vị trí mới
    - Cuối cùng trả về danh sách các nốt trong thang âm

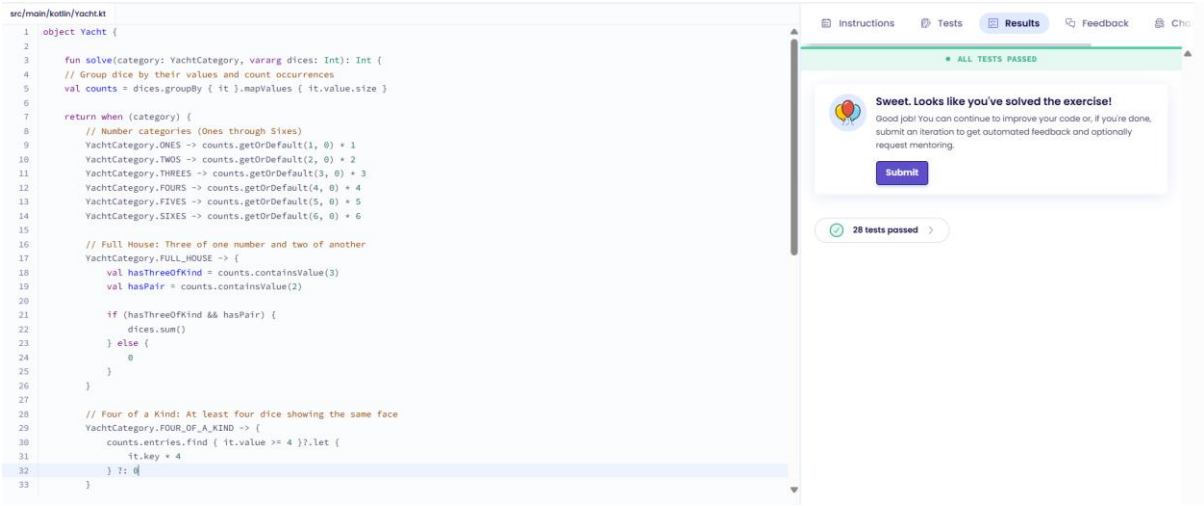


Đoạn code này định nghĩa một class Kotlin tên "SpaceAge" để tính tuổi trên các hành tinh khác nhau trong hệ Mặt Trời. Cụ thể:

- Class nhận một tham số `seconds` (kiểu Long) trong constructor đại diện cho số giây tuổi

- Biến `earthYearSeconds` lưu số giây trong một năm Trái Đất (31,557,600 giây)
- Biến `orbitalPeriods` là một map chứa chu kỳ quỹ đạo tương đối của các hành tinh so với Trái Đất:
    - Mỗi giá trị là tỷ lệ giữa chu kỳ quỹ đạo của hành tinh đó với chu kỳ quỹ đạo của Trái Đất
    - Ví dụ: Sao Thủy (Mercury) có chu kỳ quỹ đạo bằng 0.2408467 lần chu kỳ quỹ đạo Trái Đất
- Class cung cấp 8 phương thức để tính tuổi trên 8 hành tinh khác nhau:
    - Mỗi phương thức trả về tuổi (kiểu Double) trên hành tinh tương ứng
    - Công thức tính: `seconds / (earthYearSeconds * orbitalPeriods[planet]!!)`
    - Toán tử `!!` đảm bảo giá trị không null (an toàn trong trường hợp này vì các khóa đều tồn tại)





Đoạn code này định nghĩa một object Kotlin tên "Yacht" với một hàm "solve" để tính

điểm cho trò chơi Yacht (tương tự Yahtzee) dựa trên các xúc xắc và loại điểm được chọn. Cụ thể:

- Hàm nhận hai tham số: `category` (loại điểm) và `dices` (mảng các giá trị xúc xắc)
- `vararg dices: Int` cho phép truyền vào nhiều giá trị xúc xắc
- `val counts = dices.groupBy { it }.mapValues { it.value.size }` tạo một map với:
    - Khóa: giá trị xúc xắc (1-6)
    - Giá trị: số lần xuất hiện của giá trị đó

Hàm tính điểm dựa trên loại điểm (category) được chọn:

1. Các loại số (ONES đến SIXES):
    a. Tính tổng giá trị của tất cả xúc xắc có mặt tương ứng
    b. Ví dụ: FOURS trả về 4 × (số lượng xúc xắc có giá trị 4)
2. FULL_HOUSE (ba xúc xắc giống nhau và hai xúc xắc giống nhau):
    a. Kiểm tra xem có chính xác một bộ ba và một bộ đôi không
    b. Nếu có, trả về tổng tất cả xúc xắc
    c. Nếu không, trả về 0
3. FOUR_OF_A_KIND (bốn xúc xắc giống nhau):
    a. Tìm giá trị xuất hiện ít nhất 4 lần
    b. Trả về giá trị đó × 4 (nếu tìm thấy), hoặc 0 (nếu không tìm thấy)
4. LITTLE_STRAIGHT (dãy nhỏ: 1-2-3-4-5):
    a. Kiểm tra xem các xúc xắc sau khi sắp xếp có tạo thành dãy 1-2-3-4-5 không
    b. Nếu có, trả về 30 điểm
    c. Nếu không, trả về 0
5. BIG_STRAIGHT (dãy lớn: 2-3-4-5-6):
    a. Kiểm tra xem các xúc xắc sau khi sắp xếp có tạo thành dãy 2-3-4-5-6 không
    b. Nếu có, trả về 30 điểm

    c. Nếu không, trả về 0
6. CHOICE (lựa chọn):
    a. Trả về tổng tất cả các xúc xắc
7. YACHT (năm xúc xắc giống nhau):
    a. Kiểm tra xem tất cả 5 xúc xắc có cùng giá trị không
    b. Nếu có, trả về 50 điểm
    c. Nếu không, trả về 0