UNIVERSIDAD COMPLUTENSE DE MADRID

ntic master
revolucionamos la comunicación

Ejercicios propuestos - ScalaTrain

TrainJourneyPlanner: Aplicación para planificar trenes.

Tendrá: Trains, Stations, Times, JourneyPlanner y Schedules.

Vamos a crear esta aplicaciones según vayamos avanzando los temas

001 define una class

- Creación de la clase: Train
- Creación en el directorio src/main/scala
- Train.scala como fichero
- Una vez hecho hay que crear una instancia desde un Worksheet

002 define class parameters

- A la case Train hay que añadir un parámetro:
    - number como nombre
    - Int como tipo
- Desde un worksheet:
    - Intentemos crear una instancia como sin pasarle el parámetro.
    - Creemos la instancia Train correctamente.

003 promueve class parameters

003_1

- Desde el worksheet, hay que instanciar un Train y tratar de acceder a number
- Añade (prepend) un parámetro de clase kind de tipo String a la clase Train
- Convierte ambos parámetros de clase en atributos inmutables
- Desde el worksheet, hay que instanciar un Train y tratar de acceder a number y kind

003_2

- Crea la clase Time con dos parámetros de clase hours y minutes y conviértelos en atributos.
- Dentro del cuerpo de la clase Time: establecer comentarios con TODO: verificar hours entre 0 < x < 23
- Dentro del cuerpo de la clase Time: establecer comentarios con TODO: verificar minutes entre 0 < x < 59

004 define un field

- Dentro de la clase Time definamos un atributo inmutable asMinutes

- Inicializa el atributo el valor de: hours*60 + minutes

005 define un method

- A la clase Time añádele el método minus:
    - cuyo parámetro de entrada debe tener el tipo Time
    - el tipo de retorno debe ser: Int
    - el método debe devolver la diferencia entre ambas instancia de Time en minutos.

006 define an operator

- A la clase Time añádele el método -:
    - funcionará como un alias de minus: el cuerpo de - invoca a minus
- invoca a - desde el worksheet
- también invoca a minus in infix operator notation

007 use default arguments

- Modifica los parámetros de la clase Time, hours y minutes para que tengan un valor por defecto = 0
- crea en el worksheet algunas instancias de Time sin argumentos para hours, minutes y ambos

008 use packages

- recordemos que la estructura de directorios determina la jerarquía de paquetes
- creemos un paquete: com.ntic.clases.scalaTrain
- reubica las clases Time y Train en com.ntic.clases.scalaTrain:
    - se puede usar desde el IDE el asistente: “refactor > Move…”
    - O añadir manualmente el indicador del paquete al que pertenece la clase y mover la clase al directorio: com/ntic/clases/scalaTrain que es a su vez subdirectorio de: src/main/scala
- desde el worksheet crea una instancia de Time

009 define un companion object

- crea un companion object para Time
- crea un método fromMinutes que tome como parámetro minutes de tipo Int y devuelva una instancia de Time:

    - creará una instancia de Time que tendrá normalizadas los valores para hours y minutes: minutes entre 0 y 59 y hours entre 0 y 23
- ponlo a prueba desde el worksheet

010 check precondiciones

- en los TODO de Time aplicar require para asegurarnos que la hours y minutes son válidos.

011 define case classes

- convierte en case class a Time y Train:
    - Elimina val de los parámetros de clases, incluso si no molestan
    - Elimina new de Time.fromMinutes, incluso si no molestan
- ponlo a prueba en el worksheet

012 use sequence

- crea una case clase que se llame Station
    - añade un parámetro name de tipo String
- añade un parámetro de clase a Train de nombre schedule y que sea inmutable y de tipo Seq de Station
    - verifica que schedule tenga como mínimo dos elementos

013 use map

- adapta el tipo de Train.schedule a un valor inmutable de tipo Seq[(Time, Station)]
- añade un atributo a Train llamado stations
    - el tipo de stations será Seq de Station
    - se inicializa con todas las Stations que estén en schedule

014 use flatmap

- crea la clase JourneyPlanner
    - añade trains como parámetro de clase de tipo Set de Train
- añade el atributo inmutable stations a JourneyPlanner
    - inicializa el atributo con todas las Stations de todos los Trains
    - ¿qué tipo tendría sentido?: Set[Stations]
    - Intenta: map primera, muestra el tipo con el que se quda

015 use filter

- añade el método trainsAt a JourneyPlanner que tome como parámetro station de tipo Station:
    - debe devolver todos los Train que contenga el valor de station en su atributo stations
    - ¿qué tipo tiene sentido que tenga?

016 use for expressions

- añade el método stopsAt a JourneyPlanner que tome como parámetro station de tipo Station:
    - debe devolver un Set de Tuple2 de Time y Train
    - devolverá las paradas de todos los Train en la station data
    - Hint: para la implementación usa for-expression con dos generadores y un filtro

017 override tostring

- La función toString ya imprime en un formato legible la case class Time, pero se puede mejorar y que devuelva algo como: 10:30
    - sobre escribe toString cualificándolo como lazy val
    - usa el formato %02d para hours y minutes
        - %02d hace que los enteros tengan 2 dígitos completando con 0 a la izquierda

018 define un adt

- crea una clase sealed abstract TrainInfo
    - define un método abstracto number que devuelva Int
- crea las case clases InterCityExpress, RegionalExpress y BavarianRegional que extiendan TrainInfo
    - declara el parámetro de clase hasWifi de tipo Boolean con valor por defecto false en InterCityExpress
- refactoriza Train.kind y Train.number con un parámetro de clase que se llame info de tipo TrainInfo

019 use trait

- se va a usar el trait Ordered y debe ser extendido por Time:
    - Ordered es parte de la librería estándar de Scala; hay que revisarlo en las API docs
    - Ordered define operadores como <; <=

    - Ordered también declara un método abstracto que se debe definir

020 use match expressions

- Definición:
    - Un viaje entre dos estaciones es breve si:
        - existe una conexión con un mismo tren
        - hay como mucho una estación entre las dos estaciones dadas.
- añade un método isShortTrip a JourneyPlanner:
    - añade los parámetros from y to de tipo Station
    - devuelve true si existe un Train en trains donde entre las stations exiten from y to con como mucho una única Station entre ambas.
    - Hint: use los métodos de colecciones exists y dropWhile y un match expression con el patrón de secuencia
    - se tiene stations y schedule, cuál usar. prueba con dropWhile (documentación: the longest suffix of this collection whose first element does not satisfy the predicate p.) + take

021 use patterns

- Hay que demostrar como usar el pattern tupla para deshacerse el clumsy tuple
- Your instructor will demonstrate how to use a tuple pattern to get rid of the clumsy (rústico, torpe) tuple field accessors in the stopsAt method of JourneyPlanner
- Consiste en desacoplar timeAndStation en una tupla con nombres a las posiciones

022 use option

- añade el método timeAt a Train:
    - añade un parámetro de tipo Station
    - devuelve un Option de Time. Some Time si el Train se detiene en la estación, si no, devuelve None
    - uso de la función: find():
        - con stations
        - con schedule
        - y usando pattern matching
- refactoriza stopsAt en JourneyPlanner de tal manera que use el nuevo método

023 use try

- añade el método toMap a Time
    - devuelva un Map que represente la hora:

        - Map(“hours”->s"$hours", “minutes”->s"$minutes")
- añade el método fromMap al companion object Time:
    - parámetro de tipo Map[String, String]
    - envuelve la conversión de los valores del mapa con Try(s)
        - intentar conversiones con map
        - intentar for-expression con match
        - intentar for-expression con recover
    - devuelve un Option de Time si ha sido exitoso, en caso contrario None