Étape 1 (PGU) [11/13]

Classe Preconditions [1/1]

Classe Interval [2/2]

Classe ClosedInterval [2/2]

- **-0 :** (note) for better visibility you can write interval checkings as min <= val && val < max

Classe RightOpenInterval [2/2]

- **-0 :** (note) it would be better to have floorMod static
- **-0 :** there is an f missing in toString, look at the warning

Classe Angle [1/2]

- **-1 :** La constante n'est pas nommée (you already defined nearly all constants, follow this reasoning for all values even 60 (and 3600=60*60) which reprensent MIN_PER_DEG and SEC_PER_MIN, get rid of all magic numbers even if it seems overkill)
- **-0 :** (note) other than this the class is real nice

Classe Polynomial [3/4]

- **-1 :** L'attribut n'est pas final
- **-0 :** (note) Generally the code is clean, readable and well documented. Good job.

Étape 2 (GME) [10/11]

Classe SphericalCoordinates [2/2]

Classe GeographicCoordinates [1/2]

- **-1 :** Le try catch dans isValidLonDeg et isValidLatDeg n'est absolument pas nécessaire, c'est même une grosse erreur de style. Vous auriez simplement du return checkInInterval(...);

Classe HorizontalCoordinates [3/3]

Classe EquatorialCoordinates [2/2]

Classe EclipticCoordinates [2/2]

- **-0 :** A part le try catch, bon travail :)

Étape 3 (OEM) [8/11]

Type énuméré Epoch [2/2]

- **-0 :** Utilisez des constantes pré-existantes quand vous le pouvez. En particulier ici, pour J2000 et J2010, vous auriez gagné en lisibilité en utilisant

LocalTime.MIDNIGHT, LocalTime.NOON et ZoneOffset.UTC.

- **-0 :** La constante n'est pas nommée (vous ne devriez pas avoir de nombres magiques dans votre code. En particulier ici, vous auriez dû stocker le nombre de millisecondes par jour et le nombre de millisecondes par siècle dans une constante statique. En outre, vous feriez mieux de laisser le calcul des nombres magiques visibles, afin qu'on puisse savoir de quoi vous parlez (en particulier ici, je pense à 86400000 où vous auriez mieux fait de laisser 1000 * 60 * 60 * 24)).

Classe SiderealTime [0/3]

- **-1 :** L'expression devrait être stockée dans un attribut (vous auriez dû stocker s0 et s1 dans des constantes statiques afin de ne pas les recalculer à chaque fois).
- **-0 :** La constante n'est pas nommée (ici aussi vous auriez dû stocker 3600000 dans une constante statique).
- **-2 :** Une méthode existante aurait dû être utilisée (vous auriez dû utiliser normalizePositive sur le résultat final plutôt que de réduire avec HR_INTERVAL).
- **-0 :** l'utilisation d'un polynôme pour le calcul de s1 est ici inutile, il s'agit simplement d'une multiplication.

Classe EclipticToEquatorialConversion [3/3]

- **-0 :** L'expression devrait être stockée dans un attribut (vous auriez dû stocker le polynôme dans une constante statique).
- **-0 :** Une méthode existante aurait dû être utilisée (vous auriez dû utiliser ofArcSec plutôt que ofDMS)

Classe EquatorialToHorizontalConversion [3/3]

Étape 4 (MBE) [11/13]

Classe CartesianCoordinates [2/2]

Classe StereographicProjection [4/5]

- **-0 :** Une méthode existante aurait dû être utilisée dans toString(), center.toString() aurait pu être utilisée
- **-1 :** L'expression est recalculée plusieurs fois (rho*rho)

Classe CelestialObject [3/3]

Classe Planet [1/1]

Classe Moon [0/1]

- **-1 :** La constante n'est pas statique l'intervalle [0, 1] aurait pu être stocké dans une constante

Classe Sun [1/1]

Étape 5 (FGA) [12/15]

Classe Star [3/3]

Constructeur:

-0 : Une méthode existante aurait dû être utilisée : Preconditions.checkInterval
-0 : (0.92 * color) pourrait n'être calculé qu'une seule fois

Classe Asterism [1/2]

stars:

-1 : Copie inutile d'une valeur immuable List.copyOf est déjà immuable.

Interface CelestialObjectModel [1/1]

Type énuméré SunModel [2/3]

-1 : Le code n'est pas mis en page proprement : Mauvais usage des espaces, en particulier dans les expressions mathématiques.
-0 : latEcliptique est redondant, mettez directement 0 dans EclipticCoordinates.of()

Type énuméré PlanetModel [5/6]

-0 : Utilisez values() pour calculer ALL
-0 : L'expression devrait être stockée dans un attribut : Angle.TAU / 365.242191 / tropicalYear, le sinus et le cosinus de orbitalInclination
-1 : Duplication de code : (meanAnomaly -), (realAnomaly - realAnomalyEarth), (r - rEarth) et (l - lEarth). Modularisez votre code.
-0 : Math.sin(l-lonAscendingNode) est calculé deux fois
-0 : Le code n'est pas mis en page proprement : Mauvais usage des espaces dans les expressions mathématiques et dans les arguments des méthodes.
-0 : C'est une très mauvaise idée d'utiliser un attribut qui a le même nom qu'une variable sans utiliser this. (l.126 et 127)
-0 : Une méthode existante aurait dû être utilisée : Angles.ofArcSec(x) à la place de Angle.ofDMS(0,0, x)
-0 : Vous pouvez sauvegarder this.angularSize directement en radians. Cela vous permet de calculer directement angularSize sans avoir à le transformer après.
-0 : Utilisez plutôt demiGrandAxe pour savoir si une planète est inférieure ou supérieure.

Étape 6 (RPA) [11/17]

Type énuméré MoonModel [6/6]

-0 : La constante n'est pas nommée (Plus de constantes auraient pu être nommées)
-0 : L'expression devrait être stockée dans un attribut (si vous utilisez le sin/cos d'une constante, vous devriez le stocker)
-0 : L'expression est recalculée plusieurs fois (recalcule du sin/cos de la même valeur)

Classe StarCatalogue [1/3]

-0 : Stockage inutile d'une liste d'astérismes
-2 : Complexité quadratique pour trouver les étoiles d'un astérisme
-0 : Copie inutile d'une valeur immuable (unmodifiable est suffisant pour les getters)

Classe StarCatalogue.Builder [2/2]

Type énuméré HygDatabaseLoader [1/3]

- **-1 :** En cas d'exceptions, le flot ne sera pas fermé
- **-1 :** Duplication de code pour les cas par défaut

Type énuméré AsterismLoader [1/3]

- **-1 :** En cas d'exceptions, le flot ne sera pas fermé (deuxieme occurence)
- **-1 :** Complexité quadratique pour trouver les étoiles d'un astérisme (deuxieme occurence)