Space Travel
TP1

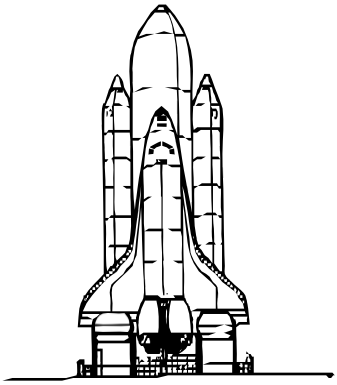
Avant propos

Nous allons créer notre première application Java ! Nous allons mettre en oeuvre différentes notions :

- *Classes*
- *Boucles et Embranchements*
- *Afficher et récupérer des valeurs*
- *Énumération de valeurs*
- *Manipulation de Date et de Calendrier*

Contexte

Nous allons nous placer dans le contexte d'une agence de voyage intersidéral : votre programme doit permettre au client de calculer son temps de trajet lors du déplacement d'une planète à une autre. De Mars à Saturne, combien de temps faut-il à la vitesse de la lumière ? À la vitesse d'une fusée ? Si je pars demain matin, quand arriverai-je ? Voici les questions auquel nous chercherons à répondre.



Nous allons fonctionner par étape et rajouter des fonctionnalités au fur et à mesure.

Supports créés par Loïc AVANTHEY, Laurent BEAUDOIN et Antoine GADEMER

Etape 0 : Welcome to the SpaceTravel agency

EXERCICE 1 (*Hello World*)

Ouvrez un éditeur de texte.
Déclarez une nouvelle classe publique `SpaceTravel` qui contient la méthode **main** qui affichera la phrase `"Welcome to the SpaceTravel agency"`.

Classe publique En Java, une classe ou une énumération publique DOIT être dans une fichier de MÊME nom.

Fonction main En Java, la méthode **main** possède toujours la même signature :
`public static void main(String[] args)`

Afficher du texte En Java, la méthode d'affichage la plus courante est :
`System.out.println(String s)`

Compilation Pour compiler, on utilise la commande (dans le Terminal) :
`javac SpaceTravel.java`
Pour exécuter, on utilise la commande :
`java SpaceTravel`
Vous devriez voir :

```
Welcome to the SpaceTravel agency
```

Etape 1 : Récupération des choix de l'utilisateur

EXERCICE 2

Nous aimerions que le programme SpaceTravel reste en attente d'une instruction utilisateur (uneboucle infinie). Pour sortir de cette boucle et quitter le programme l'utilisateur devra saisir la lettre `q` (quit). Pour afficher la liste complète des lettres qui lui sont disponibles, l'utilisateur devra saisir la lettre `h` (help).

- Rajoutez les lignes permettant la récupération d'une chaine de caractère au clavier
- puis si la chaîne est non nulle (de taille > 0), récupérez le premier caractère de cette chaine.
- Enfin, utilisez un **switch** pour gérer les différents cas possible (`'h'` et `'q'` pour le moment).

Exemple d'affichage de votre programme :

```
Welcome to the SpaceTravel agency
What do you want do? [h for help]
a
Unknown command. Type h for help
What do you want do? [h for help]
h
h: print this help screen
q: quit the program
What do you want do? [h for help]
q
Bye bye!
```

Récupérer une information textuelle

Pour récupérer une information textuelle on utilise la méthode `next()` de la classe `Scanner`.
https://docs.oracle.com/en/java/javase/16/docs/api/java.base/java/util/Scanner.html
ex :

```
Scanner scan = new Scanner(System.in);
String choice = scan.next();
```

Importer la classe Scanner Pour pouvoir utiliser la classe `Scanner` penser à rajouter **import** `java.util.Scanner;` au début de votre programme (avant la déclaration de la classe).

Attention, `next()` renvoie une chaîne de caractère (`String`) et pas un unique caractère (**char**), comment faire ? Soit en comparant les `String` :

```
// Condition simple
if(choice.equals("q")) { // Ici on utilise "q" avec des guillemets en tant que String.
   /* Do Something */
}

// Condition multiple
switch(myString) {
   /* ... */
   case "q" : // Ici on utilise "q" avec des guillemets en tant que String.
   /* ... */
}
```

Soit en extrayant le premier caractère de la chaîne :

```
// Condition simple
if(choice.length() > 0) {// Si il y a au moins un caractere !
   if(choice.charAt(0) == 'q') { // Ici on utilise 'q' avec des apostrophes en tant que char
      /* Do Something */
   }
}

// Condition multiple
// Condition simple
if(choice.length() > 0) {// Si il y a au moins un caractere !
   switch(choice.charAt(0)) {
      /* ... */
      case 'q': // Ici on utilise 'q' avec des apostrophes en tant que char
      /* ... */
   }
}
```

Plus d'info: https://docs.oracle.com/en/java/javase/16/docs/api/java.base/java/lang/String.html

Etape 2 : Énumérons les destinations possibles

Énumération : Une énumération est une structure particulière qui permet de créer un type possédant un nombre prédéterminé d'instances.

Comment définir une énumération simple ?

En écrivant dans un fichier dédié,

```
public enum MyEnum { VALUE1, VALUE2, etc. }
```

Les noms des instances d'une énumération sont toujours en MAJUSCULES pourles

différencier du reste.

On accède aux valeurs de l'énumération (qui sont des objets et des instances de classe) par le biaisdu nom de l'énumération suivi du nom de l'instance :

```
MyEnum.VALUE1
```

Ici, `VALUE1` est un objet. On peut accéder à son nom avec la méthode `name()`

```
System.out.println( MyEnum.VALUE1.name() );
```

```
VALUE1
```

et à sa position dans la liste avec `ordinal()`.

```
System.out.println( MyEnum.VALUE1.ordinal() ); // La numeration commence a 0
```

```
0
```

On peut par ailleurs récupérer la liste de toutes les valeurs avec la méthode de classe `MyEnum.values()`

```
MyEnum[] valuesArray = MyEnum.values(); // Recuperation du tableau
for (int i = 0 ; i < valuesArray.length ; i++) { // Parcours du tableau
    System.out.println("MyEnum contient l'instance "+valuesArray[i].name()+" qui est a la
        position "+valuesArray[i].ordinal());
}
```

```
MyEnum contient l'instance VALUE1 qui est a la position 0
MyEnum contient l'instance VALUE2 qui est a la position 1
etc.
```

Enumeration dans un fichier séparé

Attention, comme votre énumération est publique elle doit aller dans un fichier de même nom que l'énumération (comme pour les classes).

Ex :

```
// DANS LE FICHIER WeekDay.java
public enum WeekDay { MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY, SATURDAY, SUNDAY };
```

Comment utiliser les énumérations dans les switch ?

Les énumérations ont été conçu pour fonctionner dans les choix multiples (**switch**).

On les utilise très simplement **sans avoir besoin de répéter le nom de l'énumération** :

```
Weekday today; // Referent
today = WeekDay.MONDAY; // Choix de la valeur

switch(today) {
   case MONDAY: // Pas besoin de mettre WeekDay.MONDAY
   case TUESDAY:
   case WEDNESDAY:
   case THURSDAY:
   case FRIDAY:
```

```
        System.out.println("Au travail !"); break;
    case SATURDAY:
    case SUNDAY:
        System.out.println("C'est le Weekend !"); break;
    default: // Pas necessaire, il ne peut PAS y avoir d'autre cas !
    }
```

Enumération et new

Par définition, les énumérations sont en nombre **fini** déclarées **à l'avance**. Leur constructeur est **privé** (cas très rare) et on ne peut pas utiliser **new**

ex :

```
WeekDay today = new WeekDay("Escourgeon"); // INTERDIT
```

EXERCICE 3

Dans un nouveau fichier `Planet.java`, créez une énumération **public** `enum Planet` avec les différentes planètes du système solaire : MERCURY , VENUS, EARTH, MARS, JUPITER, SATURN, URANUS, NEPTUNE.

Visibilité des fichiers entre eux Toutes les classes et les énumérations présentes dans un même répertoire sont considérées comme faisant partie d'un même package et **ne nécessite donc pas de import.**

EXERCICE 4

Dans votre classe `SpaceTravel`, rajoutez une option associée à la lettre `l` qui liste l'ensemble des planètes de destination disponibles (référez vous à l'exemple d'énumération ci-dessus).

Exemple d'affichage de votre programme :

```
Welcome to the SpaceTravel agency
What do you want do? [h for help]
h
l: list the planets
h: print this help screen
q: quit the program
What do you want do? [h for help]
l
MERCURY, VENUS, EARTH, MARS, JUPITER, SATURN, URANUS, NEPTUNE,
What do you want do? [h for help]
q
Bye bye!
```

Etape 3 : Les énumérations, c'est la classe !

Lien entre enum et class

Dans le cours de POO, on a parlé que de **classe**, c'est quoi ces `enum` ??

Si l'on revient à l'énumération définie à l'étape précédente :

```
public enum MyEnum { VALUE1, VALUE2, etc. }
```

Il faut voir cette syntaxe comme une **forme condensée de déclaration de classe**, c'est presque comme si nous avions déclaré :

```
public class MyEnum {

    private int ordinal;
    private String name;

    private MyEnum(String name, int ordinal) { // Constructeur prive
        this.name = name;
        this.ordinal = ordinal;
    }

    public int ordinal() { // accesseur public
        return ordinal;
    }

    public String name() { // accesseur public
        return name;
    }

    public String toString() {
        return this.name();
    }

   /* Liste des instances de classe possibles */
    public static final MyEnum VALUE1 = new MyEnum(0, "VALUE1");
    public static final MyEnum VALUE2 = new MyEnum(1, "VALUE2");
        // etc.
}
```

Avouez que la première manière est beaucoup plus condensée ! Et comme elle certifie qu'il n'y a pas d'autre instances que celle prévues, l'`enum` permet aussi d'être utilisée dans les **switch** ce qui n'est pas le cas de la deuxième syntaxe.

Comment personnaliser nos énumérations ?

Si les `enum` sont des classes, cela veut dire que l'on peut leur déclarer des nouveaux **attributs** et de nouvelle **méthodes**. Ce qui va les rendre encore plus intéressantes !

Pour cela il va falloir rajouter trois choses :
- des attributs privés
- des accesseurs publics
- un constructeur (privé) pour **initialiser les attributs**

Considérons une énumération des mois de l'année :

```
// Fichier Month.java
public enum Month {
   JANUARY, FEBRUARY, MARCH, APRIL, MAY, JUNE, JULY, AUGUST, SEPTEMBER, OCTOBER, NOVEMBER,
       DECEMBER;
};
```

Mettons que l'on veuille associer l'information du **nombre de jour du mois** dans notre énumération.

Nous pourrions rajouter un attribut entier privé ainsi que l'accesseur associé.

```java
public enum Month {
// Fichier Month.java
   JANUARY, FEBRUARY, MARCH, APRIL, MAY, JUNE, JULY, AUGUST, SEPTEMBER, OCTOBER, NOVEMBER,
       DECEMBER;

   private int nbDayInMonth; // Private field
   public int getNbDayInMonth() { // Public getter
      return nbDayInMonth;
   }
}
```

Néanmoins notre attribut n'est pas initialisé. Tous les mois ont donc pour valeur 0 (valeur par défaut pour toute variable en Java).

Pour l'initialiser il nous faut rajouter un **constructeur**. Ce constructeur prendra un paramètre servant à initialiser notre attribut.

Rappelez-vous que les `enum` ne peuvent avoir que des constructeurs privés pour éviter d'avoir ne nouvelles instances non prévues.

```java
   private Month(int nbDayInMonth) { // Private constructor
      this.nbDayInMonth = nbDayInMonth;
   }
```

Nous avons notre constructeur, mais comment l'appeler ? La syntaxe des `enum` ne possède pas de **new** !

Si c'était une classe nous pourrions écrire :

```java
public class Month {

   /* ... */

   /* Liste des instances de classe possibles */
    public static final Month JANUARY = new Month(31);
    public static final Month FEBRUARY = new Month(28);
   //etc.

}
```

La syntaxe des `enum` se contente d'écrire :

```java
public enum Month {
   JANUARY(31), FEBRUARY(28), /* ... */
}
```

Ce qui semble assez logique quand on comprend d'où elle vient.

Au final, notre énumération devient :

```java
public enum Month {
// Fichier Month.java
   JANUARY(31), FEBRUARY(28), MARCH(31), APRIL(30), MAY(31), JUNE(30), JULY(31), AUGUST(31),
       SEPTEMBER(30), OCTOBER(31), NOVEMBER(30), DECEMBER(31); // Instance list with field
       initialization

   private int nbDayInMonth; // Private field
   public int getNbDayInMonth() { // Public getter
      return nbDayInMonth;
   }

   private Month(int nbDayInMonth) { // Private constructor
      this.nbDayInMonth = nbDayInMonth;
   }
};
```

Plusieurs attributs Évidemment la démarche ci-dessus s'applique pour des énumérations avec plusieurs attributs.

```
public enum Month {
// Fichier Month.java
   JANUARY(31, "Winter"), /* ... */

   private Month(int nbDayInMonth, String season) { // Private constructor
      this.nbDayInMonth = nbDayInMonth;
      this.season = season;
   }
};
```

Les méthodes name() et ordinal() sont hérités de Enum et sont toujours disponibles malgré nos modifications !

```
System.out.println(Month.JANUARY.ordinal());
```

```
0
```

Plus d'information sur les `enum` :

https://docs.oracle.com/en/java/javase/16/docs/api/java.base/java/lang/Enum.html

https://docs.oracle.com/en/java/javase/16/docs/api/java.base/java/util/EnumMap.html

EXERCICE 5

Modifiez votre énumération pour intégrer le nom en français des planètes ainsi que leur distance moyenne au soleil en Unité Astronomique :
http://www.universetoday.com/15462/how-far-are-the-planets-from-the-sun/.
Pensez à rajouter les accesseurs `public double getDistanceFromTheSunInAstronomicalUnit()` et `public String getCommonName()`.
Pensez aussi a redéfinir `public String toString()`

Pourquoi redéfinir la méthode toString() héritée de Object ?

La méthode `String toString()` de la classe `Object` permet à Java de toujours pouvoir afficher unevariable de type objet (qui hérite donc de `Object`) dans la console.

```
Car myCar = new Car("Toyota");
System.out.println("My car is "+myCar); // Appel implicite a myCar.toString()
```

Par défaut l'affichage prend la forme : nom de la classe, suivi d'une référence unique correspondant à l'objet.

```
My car is Car@7e938b4a
```

Pour avoir un affichage plus lisible pour l'utilisateur, on redéfinie cette méthode.

```
public class Car {

  private String brand;

  @Override // Redefinition de la methode
  public String toString() {
    return "a "+brand+".";
  }
}
```

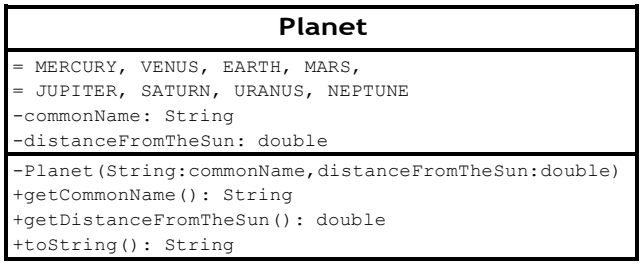
Du coup le code initial affiche désormais :

```
My car is a Toyota.
```

Sans modifier votre fonction **main**, votre programme devrait afficher :

```
Welcome to the SpaceTravel agency
What do you want do? [h for help]
l
Mercure (0.387 UA), Venus (0.722 UA), Terre (1.0 UA), Mars (1.52 UA), Jupiter (5.2 UA),
    Saturne (9.58 UA), Uranus (19.2 UA), Neptune (30.1 UA),
What do you want do? [h for help]
q
Bye bye!
```

Le diagramme UML de la classe `Planet` est pour le moment :



Etape 4 : Prenons nos distances

EXERCICE 6

Modifiez votre énumération pour ajouter :

- la constante `UA_IN_KM = 149597871.0` correspondant au nombre de kilomètre dans une Unité Astronomique
- la constante `LIGHT_SPEED_IN_KM_PER_S = 299792.458` correspondant à la vitesse de la lumière en *km/s*
- la méthode **public double** `distanceInUATo(Planet otherPlanet)` qui fournit la distance en U.A. entre la planète et une autre planète. (NB : une distance est toujours positive !)
- la méthode **public double** `distanceInKMTo(Planet otherPlanet)` qui fournit la distance en KM entre la planète et une autre planète.
- la méthode **public double** `travelTimeInSTo(Planet otherPlanet)` qui calcule le temps de voyage en secondes de la planète à une autre planète en fonction de la vitesse de la lumière (NB : $v = \frac{d}{t}$)
- la méthode **public double** `travelTimeInSTo(Planet otherPlanet, double speedInKmPerS)` qui calcule le temps de voyage en secondes de la planète à une autre planète en fonction de la vitesse (en *km/s*) passée en paramètre.
- Du coup, rajoutez à votre classe `SpaceTravel` la constante `ROCKET_SPEED_IN_KM_PER_S = 4.0` correspondant à la vitesse d'une fusée en *km/s* (il faut au moins 8*km/s* pour s'extraire de la gravité terrestre, mais le coût en carburant est trop élevé pour maintenir cette vitesse. 4*km/s* est la vitesse prévue pour un voyage Terre-Mars.)

Constante En Java, une constante est un attribut de classe déclaré **public static final**.

Fonctions mathématiques la classe Math possède de nombreuse méthode statique de calcul mathématique :
https://docs.oracle.com/en/java/javase/16/docs/api/java.base/java/lang/Math.html

Pourquoi un seul paramètre pour la méthode double distanceInUaTo(Planet otherPlanet) ?

La méthode `distanceInUATo(Planet otherPlanet)` est définie dans l'énumération Planet comme une méthode d'instance (pas **static**). On peut donc l'appeler depuis n'importe quelle instance :

```
Planet departurePlanet = Planet.MERCURY;
System.out.println("Mercury to Neptune: "+ departurePlanet.distanceInUATo( Planet.NEPTUNE ) + "
    UA") ;

System.out.println("Earth to Mars: "+ Planet.EARTH.distanceInUATo( Planet.MARS ) + " UA" ) ;
```

```
Mercury to Neptune: 29.713 UA
Earth to Mars: 0.52 UA
```

En objet, on ne considère **pas une méthode extérieure qui prendrait deux planètes en paramètre** pour calculer leur distance, **mais on demande** (envoi de message) **à l'une des planètes "Quelle est ta distance à cette autre planète ?"**

Le diagramme UML des classes deviennent :

Planet
= MERCURY, VENUS, EARTH, MARS, = JUPITER, SATURN, URANUS, NEPTUNE +UA_IN_KM: double = 149597871 +LIGHT_SPEED_IN_KM_PER_S: double = 299792.458 -commonName: String -distanceFromTheSun: double
-Planet(String:commonName,distanceFromTheSun:double) +getCommonName(): String +getDistanceFromTheSun(): double +toString(): String +distanceInUATo(otherPlanet:Planet): double +distanceInKMTo(otherPlanet:Planet): double +travelTimeInSTo(otherPlanet:Planet): double +travelTimeInSTo(otherPlanet:Planet,speedInKmPerS:double): double

SpaceTravel
+ROCKET_SPEED_IN_KM_PER_S: double = 4
+main(args:String[]): void

EXERCICE 7

Testez vos nouvelles fonctions dans la méthode **main**, puis passez à la suite.

EXERCICE 8

Rajoutez une nouvelle méthode de classe **private static** `Planet choosePlanet(Scanner scan)` dans `SpaceTravel` qui affiche la liste des planètes disponibles puis récupère au clavier le choix de l'utilisateur. En comparant la chaîne de caractère saisie et le nom courant (en français) de la planète, la fonction renvoie l'instance de `Planet` correspondante ou **repose la question** si aucune réponse valide n'a été trouvé.

Pour éviter de redéclarer l'object Scanner, celui-ci est passé en paramètre depuis le **main**.

Pour comparer deux String On peux utiliser les méthodes `equals`, `equalsIgnoreCase`, `compareTo`, `compareToIgnoreCase`, `startsWith`, `endsWith`, `contains`, `matches`. Ici nous cherchonsà comparer le nom de la planète avec un version incomplète du nom (par `mer` pour `Mercure`),quelle semble être la meilleur méthode ?
https://docs.oracle.com/en/java/javase/16/docs/api/java.base/java/lang/String.html

Pour rechercher un élément dans une collection. C'est toujours le même algorithme :

- Supposer que l'élément n'y est pas
- Comparer l'élément recherché à chacun des éléments de la collection. Si vous le trouvez,

vous pouvez vous arrêter.
— Si vous atteignez la fin de la collection, c'est que votre hypothèse de départ était vraie.

Le diagramme UML de `SpaceTravel` devient :

SpaceTravel
+ROCKET_SPEED_IN_KM_PER_S: double = 4
+main(args:String[]): void -choosePlanet(scan:Scanner): Planet

EXERCICE 9

Rajoutez une option associé à la lettre `'d'` qui demande de choisir une planète de départ et une planète d'arrivée puis affiche les informations de distance entre les deux planètes en U.A. et en million km ainsi que le temps nécessaire de trajet à la vitesse de la lumière (en minutes) et à la vitesse d'une fusée (en mois).

Exemple d'affichage :

```
Welcome to the SpaceTravel agency
What do you want do? [h for help]
d
What is your departure planet?
Mercure, Venus, Terre, Mars, Jupiter, Saturne, Uranus, Neptune,
Ter
Departure planet set to: Terre
What is your arrival planet?
Mercure, Venus, Terre, Mars, Jupiter, Saturne, Uranus, Neptune,
Mar
Arrival planet set to: Mars
The distance between Terre and Mars is 0,5200 UA
It is equivalent to 77,8 million of Km!
At the speed of light, you will need 4,3 minutes.
But with our current technology it's more 7,5 month!

What do you want do? [h for help]
q
Bye bye!
```

Changement d'unités Vos fonctions renvoie des valeurs en km ou en secondes. Pensez à diviser ces valeurs pour retrouver des informations plus parlantes en minutes, en mois ou en millions de kilomètres !

Affichage formaté de valeurs Pour retrouver le formatage d'affichage dont vous aviez l'habitude en C vous pouvez utiliser la méthode : `System.out.printf(String s, ...)` !

Exemple :

```
System.out.printf("At the speed of light, you will need %.1f minutes", time);
```

Nombre de chiffres Rappelez vous que le descripteur `%.Yf` permet d'afficher Y chiffre aprèsla virgule.

Etape 5 : Date d'arrivée prévue *Mars 2048 à minuit 12*

Nous sommes désormais capable d'établir le temps de trajet entre deux planètes. Mais si je pars demain, quand est-ce que j'aurai la chance d'observer le mont Olympe ? Quand dois-je partir pour voir Titan en Juin 2027 ?

Pour manipuler des dates en Java On utilise les classes `Calendar`, `Date` et `SimpleDateFormat`
https://docs.oracle.com/en/java/javase/16/docs/api/java.base/java/util/Calendar.html
https://docs.oracle.com/en/java/javase/16/docs/api/java.base/java/util/Date.html
https://docs.oracle.com/en/java/javase/16/docs/api/java.base/java/text/SimpleDateFormat.html

`Calendar` permet de se déplacer dans un calendrier virtuel et ainsi de manipuler des objets `Date` (le jouret l'heure). La classe `SimpleDateFormat` permet d'afficher ces `Date` dans un format plus parlant pour les humains.

Exemple d'utilisation :

```
Calendar calendar = Calendar.getInstance(); // Now
SimpleDateFormat dateFormatter = new SimpleDateFormat("d/M/yy HH:mm:ss"); // You can adjuste the Date and Time format as you want
SimpleDateFormat dateFormatter2 = new SimpleDateFormat("d MMMM y H:mm:s"); // Other possibility of format
Date myDate = calendar.getTime(); // Extract the date from the calendar
System.out.println(dateFormatter.format(myDate)); // Print formatted Date with first syntax
System.out.println(dateFormatter2.format(myDate)); // Print formatted Date with second syntax
calendar.add(Calendar.DAY_OF_MONTH, 1); // Add 1 day => tomorrow
myDate = calendar.getTime(); // Extract the date from the
System.out.println(dateFormatter2.format(myDate)); // Print formatted Date
calendar.set(Calendar.HOUR, 19); // Fix the hour field to 19
myDate = calendar.getTime(); // Extract the date from the calendar
System.out.println(dateFormatter2.format(myDate)); // Print formatted Date
calendar.set(2015, 10, 27, 21, 14, 03); // Fix the whole date to the 27/10/2015 21:14:03
myDate = calendar.getTime(); // Extract the date from the
System.out.println(dateFormatter2.format(myDate)); // Print formatted Date
```

```
22/10/13 17:33:48
22 octobre 2013 17:33:48
23 octobre 2013 17:33:48
23 octobre 2013 19:33:48
27 octobre 2015 21:14:03
```

Classes abstraites Remarquez que `Calendar` est une classe abstraite qui ne peut être instanciée directement. Nous utilisons à la place la méthode `getInstance` qui permet de récupérerl'instance de la classe fille la plus pertinente pour nous : en France, ce sera `GregorianCalendar`.

On peut changer la date courante du calendrier :

- soit en modifiant directement l'un des champs (jour du mois, heure, etc.)
- soit en ajoutant une valeur à l'un des champs avec la méthode `add` Cette méthode nous déplace effectivement dans le temps (ajouter 120 secondes équivaut à ajouter 2 minutes)
- soit en décalant la valeur d'un champ avec la méthode `roll` qui n'incrémente de manière circulaire que le champ concerné (Décembre 2013 + 1 mois = Janvier 2013)
- soit à partir d'un objet `Date` directement avec la méthode `setTime`.

On peut extraire la date courante dans un objet `Date` avec la méthode `getTime`.

EXERCICE 10

Instanciez deux objets `Calendar` et `SimpleDateFormat` dans votre fonction **main** (pensez au **import**).

Puis ajouter une méthode **private static** `Date chooseDate(Scanner scan, Calendar calendar)` quirécupère au clavier une date (année, mois, jour, heure, minute, seconde) saisie par l'utilisateur et retourne l'objet `Date` associé.

Exemple d'affichage :

```
When do you want to leave?
Year: 2014
Month: 03
Day: 24
Hour: 7
Minute: 10
Second: 0
```

Pour fixer le calendrier sur une date précise On utilise la méthode `setTime` sur chacun des champs que l'on cherche à fixer (cf. exemple au début de la section).

Le diagramme UML finale de la classe `SpaceTravel` :

+ROCKET_SPEED_IN_KM_PER_S: double = 4
+main(args:String[]): void +choosePlanet(scan:Scanner): Planet +chooseDate(scan:Scanner): Date

EXERCICE 11

Rajoutez une option associé à la lettre `'t'` qui vérifie que les planètes de départ et d'arrivée au bien été choisies, puis demande de choisir une date de départ. Elle affiche ensuite la date d'arrivée si l'on a voyagé à la vitesse d'une fusée.

```
Welcome to the SpaceTravel agency
What do you want do? [h for help]
t
Please choose first your planet with key 'd'
What do you want do? [h for help]
d
What is your departure planet?
Mercure, Venus, Terre, Mars, Jupiter, Saturne, Uranus, Neptune,
Ter
Departure planet set to: Terre
What is your arrival planet?
Mercure, Venus, Terre, Mars, Jupiter, Saturne, Uranus, Neptune,
Mar
Arrival planet set to: Mars
The distance between Terre and Mars is 0,5200 UA
It is equivalent to 77,8 million of Km!
At the speed of light, you will need 4,3 minutes.
But with our current technology it's more 7,5 month!

What do you want do? [h for help]
t
When do you want to leave?
Year: 2013
Month: 11
Day: 01
Hour: 12
Minute: 05
Second: 0

Rocket departure time: 1 nov. 2013 12:05:00
Rocket arrival time: 14 juin 2014 15:13:43
```

Comportement étrange ? après les `scan.nextInt()` de la méthode `Date chooseDate()`, la boucle principale affiche `"Unknown command"` alors qu'aucune commande n'a été tapée par l'utilisateur.

Ce qui se passe : `scan.nextInt()` ne consomme que les chiffres, il reste donc le retour à la ligne final dans le buffer du Scanner qui est donc lu par le `scan.next()` de la boucle principale comme une commande (mais n'estpas reconnu comme un caractère valide).

Solution :

Pour vider le buffer, rajoutez simplement la commande `scan.nextLine()` (sans récupérer la valeur) quiva "manger" la fin de la ligne et éviter le comportement observé.

EXERCICE 12 (*Bonus*)

Rajoutez une commande permettant de spécifier, non pas la date de départ mais la date d'arrivée et fournissant la date de départ maximum pour arriver à l'heure