

TD n°2 : Les débuts de l'informatique moderne : la classe ! (suite)

Rappel du cahier des charges : Imaginez maintenant que le département Informatique soit votre nouveau client et que pour les JPO, il vous demande de créer un jeu (une sorte de memory par exemple) qui permettrait d'associer quelques personnages célèbres de l'histoire de l'informatique et la machine sur laquelle ils ont travaillé.

Mettre en place ce jeu nécessite donc de pouvoir manipuler des personnages célèbres (**famous computer pioneer**) et des machines (**device**).

❑ Focus sur la classe Device :

La conception, l'implémentation et les tests autour de la classe **Device** ont été réalisés précédemment. Les artefacts produits à la séance précédente concernant la phase de conception et la phase d'implémentation (diagramme de classes et code de production écrit en java) vous sont redonnés à la page 3 cet énoncé 😊

❑ Focus sur la classe ComputerPioneer :

L'application devra manipuler des objets de type **ComputerPioneer**, tels que Ada Lovelace ou Alan Turing, ...

Pour commencer, le problème sera simplifié **en posant l'hypothèse suivante** :

Seule l'identité d'un personnage célèbre nous intéresse c.-à-d. :

- un (seul) prénom (le premier)
- un (seul) nom

1. Après cette rapide **analyse**, vous pouvez passer à la phase de **conception**. Dans la rubrique **Conception** de la page 3 du tableau récapitulatif de cet énoncé, modélisez la classe **ComputerPioneer** à l'aide d'un diagramme UML (à droite de la classe **Device** modélisée précédemment)
2. **Phase d'implémentation** : Implémentez la classe **ComputerPioneer** en Java dans la rubrique **Implémentation** de la page 4 du tableau récapitulatif de cet énoncé
3. **Phase de tests**

Complétez la méthode **main** en implémentant les deux jeux d'essais suivants :

- Instanciation d'un objet de type **ComputerPioneer** nommé **adaLovelace** dont l'appel de la méthode **toString()** sur cet objet permet de procéder via un **System.out.println** à l'affichage console suivant :
Ada Lovelace is a pioneer in Computer Science.
- Instanciation d'un objet de type **ComputerPioneer** nommé **adaLovelace** dont l'appel de la méthode **toString()** sur cet objet permet de procéder via un **System.out.println** à l'affichage console suivant :
Alan Turing is a pioneer in Computer Science.

❑ Ajouter une relation entre la classe ComputerPioneer et Device

Dans notre contexte métier (celui du jeu pour les JPO), on souhaite qu'un personnage célèbre (pioneer) puisse être associé au matériel (device) sur lequel il travaille.

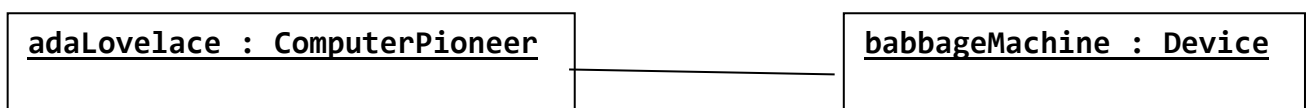
Pour commencer ce projet le plus rapidement possible, nous simplifierons le problème en **posant l'hypothèse suivante** :

Un personnage célèbre travaille uniquement sur un appareil (à un instant t)

D'un point de vue **Conception Orienté Objet**, cela signifie :

qu'un **objet** de type ComputerPioneer **est (re)lié à** un seul **objet** de type Device

comme le montre le **diagramme d'objets** (photo instantanée du système à un instant t) suivant entre adaLovelace et BabbageMachine.

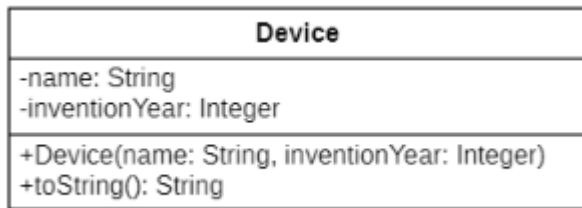


Comment ce lien entre objets (qui permet la communication entre ces deux objets) se répercute-t-elle au niveau du diagramme de classes ?

- Après avoir rappelé la **différence entre une classe et un objet**,
 - Interrogez-vous sur :
 - Quel **type de relation** ?
 - Quel(s) **sens de navigabilité** et pourquoi ce(s) choix (en précisant quel sera l'impact du choix de la navigabilité lors de la phase d'implémentation)
 - Quelle **multiplicité** ?
4. **Phase de conception** : Après avoir répondu aux questions précédentes, complétez le diagramme UML de la rubrique **Conception** de la page 3 de cet énoncé (qui contient déjà les classes **Device** et **ComputerPioneer**) de manière à modéliser la solution que vous avez retenue pour répondre à ce problème.
5. **Phase d'implémentation** : Modifiez, dans la rubrique Implémentation de la page 4, le code Java déjà écrit pour qu'il reflète la nouvelle conception (le nouveau diagramme de classes) permettant de savoir sur quel appareil un personnage célèbre travaille.
- Hypothèse : Pour simplifier le problème, on considère que pendant toute la durée du programme ce *device* restera inchangé c-a-d que le personnage célèbre travaille toujours sur un seul et même *device* pendant la durée du jeu.

Conception

Diagramme de classes (UML)



Implémentation

Ecriture du code dans le langage Java

```
public class Device {  
    private final String name;  
    private final Integer inventionYear;  
  
    public Device(String name, Integer inventionYear) {  
        this.name = name;  
        this.inventionYear = inventionYear;  
    }  
  
    public String toString() {  
        return "The " + this.name + " was invented in "  
                + this.inventionYear+".";  
    }  
}
```

Ecriture du code dans le langage Java

Tests (manuels)

Isabelle BLASQUEZ- Débuts de l'informatique moderne : la classe ! (suite) Conception & Implémentation Page 4

6. Phase de tests :

→ Réécrire la méthode `main` de manière à obtenir les deux jeux d'essais suivants :

- **Jeu d'essai n°1 :**

Après l'instanciation d'un `Device` nommé `babbageMachine` et d'un `ComputerPioneer` nommé `adaLovelace`, l'affichage attendu en mode console sur l'objet `adaLovelace` devra être :

The Babbage Analytical Machine was invented in 1837. Ada Lovelace is a pioneer in Computer Science who worked on it.

- **Jeu d'essai n°2 :**

Après l'instanciation d'un `Device` nommé `turingEngine` et d'un `ComputerPioneer` nommé `alanTuring`, l'affichage attendu en mode console sur l'objet `alanTuring` devra être :

The Turing Engine was invented in 1936. Alan Turing is a pioneer in Computer Science who worked on it.

❑ **Ajouter du comportement à la classe `ComputerPioneer` :**

Être capable de dire si un pionnier travaille sur un device donné

Pour mettre en place le jeu, il paraît indispensable de pouvoir disposer de la méthode suivante dans la classe `ComputerPioneer` permettant ainsi de savoir si le `device` passé en paramètre est le même device sur lequel travaille actuellement le `computerPioneer`

```
public boolean worksOn(Device device) ;
```

7. **Conception** : Ajouter cette opération dans le diagramme de classes précédent.

8. **Implémentation** : Implémenter cette méthode dans votre code Java déjà écrit précédemment.

9. **Test** :

- Pour tester le code que vous venez d'écrire vous ajoutez le bout de code suivant dans votre `main` comme **jeu d'essai n°3** :

```
public static void main(String[] args) {  
  
    //... code déjà écrit précédemment  
  
    System.out.println("Test case 3 ");  
    System.out.println("-----");  
    System.out.println(adaLovelace.worksOn(babbageMachine));  
    System.out.println(adaLovelace.worksOn(turingEngine));  
    System.out.println(alanTuring.worksOn(babbageMachine));  
    System.out.println(alanTuring.worksOn(turingEngine));  
    System.out.println("-----");  
}
```

⇒ Quel affichage obtenez-vous sur la console ?

- Vous décidez d'ajouter un dernier jeu d'essai dans votre main (**jeu d'essai n°4**) :

```
public static void main(String[] args) {

    //... code déjà écrit précédemment
    System.out.println("Test case 4 ");
    System.out.println("-----");
    Device babbage = new Device ("Babbage Analytical Machine",1837);
    Device turing = new Device ("Turing Engine",1936);
    System.out.println(adaLovelace.worksOn(babbage));
    System.out.println(adaLovelace.worksOn(turing));
    System.out.println(alanTuring.worksOn(babbage));
    System.out.println(alanTuring.worksOn(turing));
    System.out.println("-----");
}
```


Suivant l'implémentation que vous avez réalisée pour la méthode **worksOn** un des deux affichages suivants va apparaître sur la console : 😊

Affichage n°1 :	Affichage n°2 :
<pre>----- Test case 3 ----- true false false true ----- Test case 4 ----- false false false false -----</pre>	<pre>----- Test case 3 ----- true false false true ----- Test case 4 ----- true false false true -----</pre>

4 Quel affichage pensez-vous obtenir avec votre code actuel ? autrement dit, est-ce que les affichages votre jeu d'essai n°3 et votre jeu d'essai n°4 sont identique ?

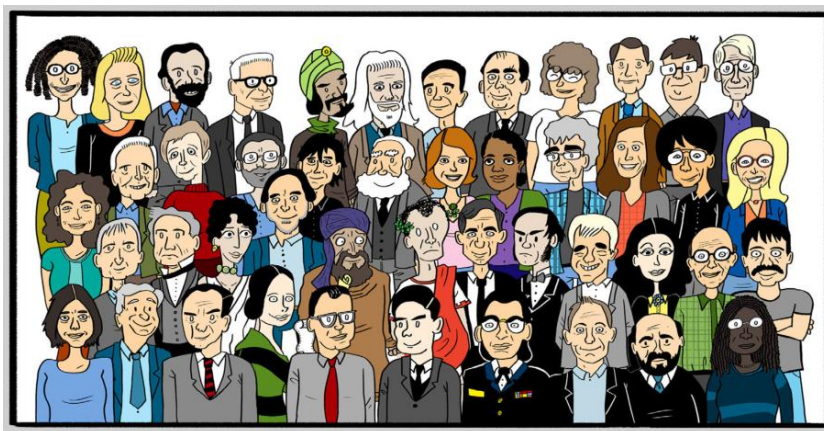
5. A votre avis, est-il souhaitable que les affichages du jeu d'essai n°3 et du jeu d'essai n°4 soient identique ?

Si vous pensez que oui et si tel n'est pas déjà le cas, comment procéderiez-vous pour arriver à ce résultat ?

	<p style="text-align: center;">A revoir ... (pour bien assimiler les notions mises en pratique dans ce TD)</p>
	<p>→ Dans la rubrique Classes Object et String Accès direct à la rubrique via : https://unil.im/butoo3 (URL complète : https://www.youtube.com/playlist?list=PLzzeuFUy_CnhW4RoeaQ36pZ5tqoK5lxr7)</p> <ul style="list-style-type: none"> <input type="checkbox"/> 01. Toutes les classes Java étendent la classe Object <input type="checkbox"/> 02. Méthode <code>equals()</code> de la classe Object <input type="checkbox"/> 03. Redéfinition de <code>equals()</code> par surcharge <input type="checkbox"/> 04. Surcharge de <code>equals</code> dans la classe String

Pour votre culture personnelle :

Vous connaissez sûrement bien d'autres personnalités célèbres en informatique que vous pourrez pendre le temps de (re)découvrir (après ce TD) avec le jeu des 7 Familles de l'informatique disponible à l'adresse suivante : <https://interstices.info/jeu-de-7-familles-de-linformatique/>



Une liste plus complète et plus formelle de pionniers de l'informatique est également disponible sur : <https://history.computer.org/pioneers/> ainsi qu'une timeline qui présente l'évolution de l'informatique de l'antiquité à nos jours : <https://ieeecs-media.computer.org/assets/pdf/timeline.pdf>

Un robot fortement typé

Travail à rendre au début de la séance de TP de la semaine prochaine :

(à commencer en TD si le temps le permet ...)

Vous allez participer au développement du projet qui va piloter le prochain robot (rover) sur la lune.

Pour simplifier le problème, on considère dans un premier temps que le robot se déplace sur une carte 2D. Le robot démarre toujours du centre de la carte c-a-d en (0,0) et il est dirigé vers le nord.

Le robot (rover) doit pouvoir avancer, tourner à droite et gauche.



❑ **Conception** : Proposez un diagramme de classes qui permet de modéliser ce problème.

Vous dessinerez ce diagramme **à la main** sur une feuille libre où vous mentionnerez nom, prénom et groupe.

❑ **Implémentation** (focus uniquement sur les **constructeurs**) Pour ce travail, vous ne devez pas implémenter tout le diagramme de classes modélisé précédemment, mais vous proposerez uniquement une implémentation des constructeurs.

→ En principe, **plusieurs constructeurs sont exposés dans une classe** avec des paramètres différents pour donner plus de flexibilité à votre code et permettre d'instancier un objet de différentes manières selon les besoins c.-à-d. en passant des paramètres d'entrée différents (en nombre, en type,...).

Le constructeur qui contient en paramètre tous les attributs de la classe est appelé **constructeur primaire**. Il doit toujours être proposé comme constructeur de la classe.

Les autres constructeurs sont des **constructeurs secondaires**.


Proposez plusieurs signatures de constructeurs (un constructeur primaire et un ou plusieurs constructeurs secondaires) pour pouvoir instancier un robot de différentes façons.

→ Implémentez les constructeurs proposés précédemment.

Remarque « Bonne pratique » :

Pour optimiser le code implémenté dans vos constructeurs secondaires, vous pouvez appeler le code de votre constructeur primaire. 😊

Pour savoir comment faire en Java, regarder la vidéo suivante de José Paumard :

	A découvrir
	<p>→ Dans la rubrique Constructeur d'objet Accès direct à la rubrique via : https://unil.im/butoo4 (URL complète : https://www.youtube.com/playlist?list=PLzzeuFUy_Cni3_xF9bl5oNDvrc757-4ih)</p> <p>❑ 02. Constructeur qui appelle un autre constructeur de la même classe</p>

→ **Utilisation des constructeurs :**

Pour chaque constructeur implémenté précédemment, écrire une instruction qui permet de montrer comment **instancier un robot à l'aide de ce constructeur**.

Vous pouvez par exemple appeler les robots (objets) : **viper, python, anaconda, ...**

Remarque : en Java, si vous n'écrivez pas de constructeur dans une classe, le **constructeur par défaut** (sans paramètre d'entrée) est disponible. Mais à partir du moment où vous implémentez un constructeur dans une classe, le constructeur par défaut n'est plus disponible. Si vous souhaitez utiliser le constructeur par défaut, il faut alors l'écrire aussi 😊.