Introduction à la Programmation : Révisions et Conclusion

Laboratoire d'Intelligence Artificielle
Faculté I&C

J. Sam

Qu'avons nous vu en programmation ?

Programmer c'est décomposer une **tâche** à automatiser en une séquence d'instructions (**traitements**) et des **données**

Algorithme	S.D.A.
Traitements	Données
Expressions & Opérateurs	Variables
Structures de contrôle	Types de base
Modularisation (méthodes)	Portée
Passage de paramètres par valeur	Chaînes de caractères
Récursivité (par la pratique)	Tableaux statiques, dynamiques

J. Sam

Qu'avons nous vu en programmation (2) ?

En programmation orientée objets, on regroupe dans le même objet **les traitements et les données** qui lui sont spécifiques (principe d'**encapsulation**)

Objet	
Encapsulation et Abstraction Classes Héritage Polymorphisme Classes abstraites Interfaces (Résolution des collisions de noms (`package`, `import`)) Gestion des exceptions Introduction à l'API, la classe `Object`, la classe `String`	
Traitements (méthodes)	Données (attributs)
Constructeurs Polymorphisme `abstract` Masquage(overriding)/Surcharge(overloading)	Encapsulation Copie
Privés/protégés/publiques Hérités/Masqués/`final`/`static`	

FONDAMENTAUX

1. conception (qu'est ce qu'on veut ?)
2. implémentation (comment ça se réalise ?)
3. syntaxe (comment ça s'écrit ?)
4. tests (où sont mes fautes, comment pourrais-je les tester ?)

J. Sam

« Fondamentaux » de la POO

1. **encapsulation** Objet = attributs + méthodes

```
class Rectangle {
public double surface() { ... };
  ...
private double hauteur;
private double largeur;
}
```

   Attributs et méthodes publiques ☞ Interface de la classe

2. **héritage**

```
class RectangleColore extends Rectangle{
   private Couleur couleur;
   //...};
```

3. **polymorphisme** le choix du type se fait à l'exécution, en fonction de la nature réelles des instances
   Ingrédients : Héritage du type + résolution dynamique des liens
4. en Java un objet est manipulé **via sa référence**

Rendu du mini-projet 2

- Si vous avez été involontairement seul.e pour faire ce projet, écrivez à cs107@epfl.ch
- Suivez les consignes de la page de rendu
- Attention au délai
- Faites attention à l'encodage (UTF-8)
- Faites attention à la casse dans le nommage de ressources ("Pickaxe" à la place de "pickaxe" par exemple)
- Si l'archive est trop grosse en raison d'extensions, écrivez nous un message à cs107@epfl.ch
  - ☞ mais ce doit rester une exception

J. Sam

Faire le point

1. par rapport aux concepts vus en cours :
   - prendre les tableaux synthétiques des transparents 2 et 3
   - prendre les fiches résumé
   - et pour chacun des points, se demander si on sait :
     - de quoi ça parle ?
     - ce que ça veut dire ?
     - l'utiliser ?

     ☞ se focaliser sur les **concepts**

   Les détails de syntaxe (comment ça s'écrit) peuvent être ensuite rapidement retrouvés dans la fiche résumé, si on sait ce qu'on cherche (c'est-à-dire si on a le concept)
2. par rapport à leur mise en pratique :
   - faites les exemples de tests des années passées (mis à disposition dans Moodle)

Faire le point

1. Attention aux études de cas

2. Attention aux modalités de construction (en particulier sur l'usage de `this`)

Méthodologie (1)

Le cours a aussi essayé de vous rendre attentifs à des bonnes pratiques :

1. **Du bon usage des modificateurs d'accessibilité** :
   - éviter le `public` et le `protected` pour les attributs,
   - réserver le `public` à quelques méthodes bien choisies (l'API de la classe),

J. Sam

Méthodologie (2)

2. **Du bon usage de l'héritage et du polymorphisme** :
   - La composition est souvent préférable à l'héritage (faire en sorte qu'une classe `A` ait comme attribut un objet d'une classe `B` plutôt qu'elle hérite de `B`).
   - Vos hiérarchies de classes doivent être de bons modèles de la réalité.
3. **Du bon usage des interfaces** : à utiliser pour mettre en place des liens de type «se-comporte-comme» et pour exposer les fonctionnalités plutôt que l'implémentation.
4. **A éviter absolument** : prolifération de méthodes et d'attributs (non finaux) statiques, la définition de méthodes artificielles au corps vide pour les simples besoins de la compilation (utilisez `abstract`).
5. **Du bon usage des références** : attention aux situations nécessitant la copie (éventuellement profonde)

☞ et surtout soyez au clair sur le pourquoi de ces préceptes et autres dogmes . . .

Mini-projet 2

En principe vous avez intensivement mis en pratique tout cela dans le mini-projet 2
Si vous avez éprouvé des difficultés importantes, cela met sans doute le doigt sur des lacunes :

- dans la compréhension des concepts :
  - ☞ Veillez à bien remettre à plat les choses en révisant les bases
  - ☞ Les BOOCs peuvent être un bon moyen de le faire
- et/ou dans l'approche et la méthodologie :
  - ☞ Bien lire les spécifications et réfléchir en amont de toute implémentation est important
  - ☞ N'ayez pas peur du debugger
  - ☞ Évitez le bricolage
- et/ou dans l'organisation : certains ont commencé le mini-projet 2 très tard.

Et puis, devenir un bon programmeur prend du temps...