U.B.S. - I.U.T. de Vannes

Département Informatique

TD1
(Partie GL)
R2.01
Développement Orienté Objets
BUT-Info-1A

I. Borne, R. Fleurquin, L. Naert, B. Le Trionnaire

Janvier 2023

On considère le code Java ci-dessous placé dans un fichier texte ainsi que le résultat de son exécution. Il est souhaitable de réaliser les opérations décrites dans cet exercice en travail personnel sur votre propre machine en usant du compilateur et de la machine virtuelle java (dernière version de préférence).

Question 1

Comment doit s'appeler ce fichier pour qu'il puisse être compilé par Java ? Savez-vous ce qu'est UNICODE, son lien avec Java ?

Si l'on suppose l'environnement qui suit : un répertoire `R201` comprenant un sous répertoire `td1` qui lui-même comprend 4 sous répertoires `ws` (dans lequel vous êtes depuis la console), `src` (dans lequel est le fichier de ce code), `class` et `doc`. Que devez-vous faire pour compiler ce code?

Que contiennent au final les différents répertoires ? Que contient en particulier le fichier d'extension « .class » ? Que savez-vous sur le « bytecode » Java, son lien avec les machines virtuelles Java (JVM) et son intérêt ?

Ouvrez ce fichier d'extension « `.class` » avec un éditeur quelconque. Affreux ? Utilisez ensuite la commande `javap` du JDK (avec puis sans l'option -c pour « regarder » ce bytecode)

Question 2

Rappelez la structure d'une classe Java.

Identifiez dans cette classe les *attributs* et les *méthodes.*

Donnez le diagramme de classes UML correspondant à cette classe.

En quoi cette classe Java illustre le principe d'encapsulation et de masquage d'information ?

Question 3

A quoi sert la méthode *main()* en Java ?

Comment exécuter ce code ?

A votre avis quelle était la structure et que faisait la classe *Start* que vous utilisiez précédemment ?

Que faudrait-il changer à ce code pour obtenir la même exécution en tapant sur la console la commande `java Start Debut` ?

Combien y-aurait-il alors d'objets créés au total lors de l'exécution ?

Question 4

Une méthode est de type *modificateur* si elle modifie l'état interne d'un objet lors de son exécution. C'est-à-dire si elle peut modifier la valeur d'un ou plusieurs des attributs d'un objet. Une méthode est de type *accesseur* si elle ne modifie jamais l'état d'un objet et retourne tout ou partie de l'information concernant l'état de l'objet sur lequel elle s'applique. Il est important pour un programmeur de distinguer ces deux types de méthodes. Les modificateurs peuvent faire émerger des bugs à l'intérieur même de l'objet exécutant cette méthode en le plaçant dans un état non souhaitable. A l'inverse les accesseurs peuvent faire émerger des bugs non dans l'objet exécutant cette méthode mais dans les objets clients qui ont demandé son exécution en leur fournissant des informations fausses.

Un cas particulier de modificateur sont les *setters* et d'accesseurs sont les *getters*. Documentez-vous sur ces types de méthode particulière en Java : intérêt, signature.

Précisez parmi ces méthodes celles qui sont de type *constructeur(s)*, *modificateurs (éventuellement setter)*, *accesseurs (éventuellement getter)*.

Comment les reconnait-on ?

Une méthode peut-elle ne relever d'aucune de ces 3 catégories ?

A quoi sert exactement un constructeur en Java?

Question 5

Rappelez la différence entre type primitif et type référence. Comment sont-ils initialisés en Java ? Identifiez dans ce code des variables ou attributs de ces deux catégories.

Une variable Java est une réservation d'une zone mémoire pour stocker de l'information. L'accès à cette zone se fait via le nom de la variable. En Java on distingue 3 grands types de variables : les attributs, les variables locales dans un bloc de code et les paramètres. On peut associer à chaque variable au moins 4 caractéristiques : son *type*, sa *portée*, sa *visibilité* et sa *durée de vie*. La portée d'un identificateur (correspondant à un nom de variable, de fonction, etc.) correspond aux parties du programme où cet identificateur peut être utilisé sans provoquer d'erreur à la compilation. La visibilité est un sous ensemble de la portée et correspond à la partie du code où une variable peut effectivement être utilisée en lecture ou en écriture (non masquée par une autre variable de même nom). La durée de vie est la seule de ces 4 caractéristiques associée à l'exécution. La durée de vie d'une variable correspond à la période depuis sa création (allocation de sa zone mémoire) à sa destruction (libération de sa zone mémoire correspondante). Donnez les 4 caractéristiques pour les variables `val`, `d1` et `v`. Comment illustrer la différence entre portée et visibilité ?

Soit une classe A, que font exactement les 2 instructions : `A unA ; unA= new A(4,5) ;`

Analysez le code et le résultat de l'exécution de la partie 1 du `main()`.

Rappelez pourquoi le compilateur n'accepte pas la ligne placée en commentaire dans cette partie.

Justifiez à l'aide ce code que l'on ne doit pas confondre la notion d'objet/référence avec celle de variable. Donnez en conséquence la bonne formulation pour : « l'objet d3 ».

Question 6

Analysez le code et le résultat de l'exécution de la partie 2 du *main()*. Donnez un diagramme de séquence UML décrivant les échanges de message du code de la partie 1 et 2. On ne représentera pas les messages *println()*.

Qu'appelle-t-on *état* d'un objet ? Comment évolue-t-il ?

Question 7

Analysez le code et le résultat de l'exécution de la partie 3 du *main()*. Complétez le diagramme de séquence UML précédent en ajoutant les échanges de message du code de la partie 3. On ne représentera pas les messages *println()*.

```
public class Debut {
	private int val;
	public static void main(String[] args){
		Debut d1, d2, d3;
		System.out.println("Partie 1 : creation, initialisation d'objets, references");
		d1=new Debut(10);
		d2=new Debut(20);
		System.out.println("d1="+d1);
		System.out.println("d2="+d2);
		// System.out.println("d3="+d3);
```

```
		d3=d1;
		System.out.println("et apres d3=d1...");
		System.out.println("d1="+d1);
		System.out.println("d2="+d2);
		System.out.println("d3="+d3);

		System.out.println("Partie 2 : envoi de message et evolution de l'etat");

		d1.setVal(30);
		System.out.println("le val de d1="+d1.getVal());
		d2.setVal(40);
		System.out.println("le val de d2="+d2.getVal());
		d3.setVal(50);
		System.out.println("le val de d3="+d3.getVal());
		System.out.println("le val de d1="+d1.getVal());

		System.out.println("Partie 3 : identite et collaboration inter objets");

		d1.compare(d2);
		d1.compare(d1);
		d1.compare(d3);
		d3=new Debut(40);
		d2.compare(d3);

	}

	public Debut(int v){
		System.out.println("Valeur initiale de val="+this.val);
		this.val=v;
	}

	public void setVal(int v){this.val=v;}

	public int getVal(){return this.val;}

	public void compare(Debut unDebut){

		if (unDebut==this){System.out.println("Je suis moi-même!");}
		else if (unDebut.getVal()==this.val){
			System.out.println("Nous somme deux mais tellement ressemblants!");
		}
		else {System.out.println("Nous sommes tellement differents!");}
	}
}
```

Le résultat de l'exécution…

```
Partie 1 : creation, initialisation d'objets, references
Valeur initiale de val=0
Valeur initiale de val=0
d1=Debut@2a139a55
d2=Debut@15db9742
et apres d3=d1...
d1=Debut@2a139a55
d2=Debut@15db9742
d3=Debut@2a139a55
Partie 2 : envoi de message et evolution de l'etat
le val de d1=30
le val de d2=40
le val de d3=50
le val de d1=50
Partie 3 : identite et collaboration inter objets
Nous sommes tellement differents!
Je suis moi-même!
Je suis moi-même!
Valeur initiale de val=0
Nous somme deux mais tellement ressemblants!
```