Programmation avancée et application – 2019 / 2020

# Contrôle Continu

## Exercice I (4 points)

**Cochez les cases correspondant à vos réponses.**

**Chaque bonne réponse vaut 0,5 point, chaque mauvaise réponse vaut −0,5 point. En cas d'absence de réponse, 0 point.**

1. Un variable de type **int** est stockée sur 6 octets.
   ☐ Vrai ☒ Faux

| Type | Valeur min | Valeur max | Taille mémoire |
|---|---|---|---|
| `byte` | $-2^7 = -128$ | $2^7 - 1 = 127$ | 1 octet |
| `short` | $-2^{15} = -32768$ | $2^{15} - 1 = 32767$ | 2 octets |
| `int` | $-2^{31} \simeq -2 \times 10^9$ | $2^{31} - 1 \simeq 2 \times 10^9$ | 4 octets |
| `long` | $-2^{63} \simeq 9 \times 10^{18}$ | $2^{63} - 1 \simeq 9 \times 10^{18}$ | 8 octets |

| Type | $m$ | $e$ | Taille mémoire |
|---|---|---|---|
| `float` | 23 bits | 8 bits | 4 octets |
| `double` | 52 bits | 11 bits | 8 octets |

Les nombres décimaux sont représentés sous la forme $x = s \times m \times 2^e$

- $s$ est le signe, représenté par un bit
- $m$ est la mantisse
- $e$ est l'exposant

| Type | | | Taille mémoire |
|---|---|---|---|
| `boolean` | `true` | `false` | dépend de la JVM |

2.
```
int n = 2 ;
int m = n ;
n++ ;
```

m vaut 3 après ces instructions.

☐ Vrai ☒ Faux

3. La concaténation de String crée de nouveaux objets.

☒ Vrai ☐ Faux

**constant.**

- On a déjà vu que les String sont des objets immuables : lors d'une opération sur l'objet, il n'est pas modifié, mais un autre objet est créé
  - Peu efficace : allocation d'une zone mémoire avec une référence pour y accéder, travail en plus pour le garbage collector lorsque les objets deviennent inutilisables
- Exemple : "Bon" + "jo" + "ur" crée cinq objets
  - Trois objets pour les chaines "Bon", "jo" et "ur"
  - Un objet pour le résultat de la première concaténation : "Bonjo"
  - Un objet pour le résultat de la second concaténation : "Bonjour"

4. Une classe abstraite peut implémenter une interface.

☒ Vrai ☐ Faux

### When an Abstract Class Implements an Interface

In the section on `Interfaces`, it was noted that a class that implements an interface must implement *all* of the interface's methods. It is possible, however, to define a class that does not implement all of the interface's methods, provided that the class is declared to be `abstract`. For example,

```
abstract class X implements Y {
  // implements all but one method of Y
}

class XX extends X {
  // implements the remaining method in Y
}
```

In this case, class `X` must be `abstract` because it does not fully implement `Y`, but class `XX` does, in fact, implement `Y`.

**Source : https://docs.oracle.com/javase/tutorial/java/landl/abstract.html**

Si toutes les méthodes de l'interface ne sont pas implémentées, la classe doit être déclarée abstraite

5. Une classe peut implémenter plusieurs interfaces.
☒ Vrai ☐ Faux

Une classe peut implémenter plusieurs interfaces :
**public class** A **implements** B, C { ... }

6.
```
public class A {
    public void f(){
        System.out.println('A');
    }
}

public class B extends A{
    public void f(){
        System.out.println('B');
    }
}
```

On crée la variable A obj = **new** B();.
L'instruction obj.f() ; affiche 'A'.
☐ Vrai ☒ Faux

7. Une classe membre statique peut accéder à tous les membres de la classe englobante.
☐ Vrai ☒ Faux

- Le modificateur `static` permet d'indiquer qu'une méthode peut être appelée sans instancier sa classe :
  - Dans la classe, on peut appeler directement la méthode : maMethode() ;
  - Depuis une autre classe, on l'appelle précédée du nom de la classe qui la définit : MaClasse.maMethode() ;
- Pour un attribut, `static` indique que sa valeur est donc partagée entre les différentes instances de sa classe (on parle d'attribut de classe)
- Une méthode `static` peut faire appel à des méthodes et des attributs `static`, mais aucune méthode ou aucun attribut lié à une de la classe

8. Avec Junit 5, le tag @Before permet d'indiquer qu'une méthode doit être executée avant chaque test unitaire.
☐ Vrai ☐ Faux

PAS dans le programme pour le CC 2020 – 2021

# Exercice II (2 points)

## #1

**Écrivez une méthode qui prend en entrée une liste d'Integer et retourne la liste des éléments d'indices pairs de la première liste.**

**Par exemple,**
**si la liste donnée en entrée contient les Integer [3,21,1,12,17] ,**
**la méthode retourne la liste [3,1,17]**

```
import java.util.ArrayList;
public class Exercice2 {
	/**
	 * Une méthode qui prend en entrée une liste d'Integer
	 * et retourne la liste des éléments d'indices pairs de la première liste.
	 * @param liste Une liste d'Integer
	 * @return La liste des éléments d'indices pairs de la première liste.
	 */
	public static ArrayList<Integer> elementsIndicesPairs(ArrayList<Integer> liste) {
		ArrayList<Integer> listeElementsIndicesPairs = new ArrayList<Integer>();
		for(int i = 0 ; i < liste.size(); i+= 2)
			listeElementsIndicesPairs.add( liste.get(i) );

		return listeElementsIndicesPairs;
	}
}
```

# Exercice II (2 points)

## #2

**Écrivez une méthode qui prend en entrée une liste d'Integer et retourne l'indice du plus grand élément de la liste.**

**Par exemple,**
**si la liste donnée en entrée contient les Integer [3,21,1,12,17] ,**
**la méthode retourne 1 (l'indice du nombre 21).**

```java
import java.util.ArrayList;
public class Exercice2 {
	/**
	 * Une méthode qui prend en entrée une liste d'Integer
	 * et retourne la liste des éléments d'indices pairs de la première liste.
	 * @param liste Une liste d'Integer
	 * @return La liste des éléments d'indices pairs de la première liste.
	 */
	public static ArrayList<Integer> elementsIndicesPairs(ArrayList<Integer> liste) {
		ArrayList<Integer> listeElementsIndicesPairs = new ArrayList<Integer>();
		for(int i = 0 ; i < liste.size(); i+= 2)
			listeElementsIndicesPairs.add( liste.get(i) );

		return listeElementsIndicesPairs;
	}

	/**
	 * Une méthode qui prend en entrée une liste d'Integer
	 * et retourne l'indice du plus grand élément de la liste.
	 * @param liste Une liste d'Integer
	 * @return L'indice du plus grand élément de la liste. Si la liste est vide : return -1
	 */
	public static int indicePlusGrandElement(ArrayList<Integer> liste) {
		if( liste.size() > 0){ // liste PAS vide
			Integer max = liste.get(0);
			int indiceMax = 0;

			for(int i = 1 ; i < liste.size() ; i++){
				/* Integer.compareTo(Integer anotherInteger)
				 * public int compareTo(Integer anotherInteger)
				 * Compares two Integer objects numerically.
				 * Parameters: anotherInteger - the Integer to be compared.
				 * Returns: the value 0 if this Integer is equal to the argument Integer;
				 * a value less than 0 if this Integer is numerically less than the
argument Integer;
				 * and a value greater than 0 if this Integer is numerically greater than
the argument Integer
				 * (signed comparison)
				 *
				 * -- Source : JavaDoc --
				 */

				if( liste.get(i).compareTo(max) > 0){
					max = liste.get(i);
					indiceMax = i;
				} // if( liste.get(i).compareTo(max) > 0)
			} // for

			return indiceMax;
		}

		return -1; // si liste vide
	}
}
```

# Exercice III (2 points)

On considère la classe suivante (fournie sans explication). Ré-écrivez cette classe pour éviter les répétitions de code. Répondez sur la copie d'examen.

```
public class Main {
  public static void main(String[] args) throws IOException {
    File file = new File('donnees.txt');
    Lecture simple = new AffichageSimple();
    try {
      simple.ecrireEntiers(file);
    }catch(IOException e){
      System.out.println('Traitement non effectue
    }
    Lecture doublee = new AffichageDoublee();
    try {
      doublee.ecrireEntiers(file);
    }catch(IOException e){
      System.out.println('Traitement non effectue');
    }
  }
}
```

PAS dans le programme pour le CC 2020 - 2021

# Exercice IV (2 points)

## Voici (une partie de) la définition de la classe Coordonnees.

**Complétez la définition de la classe Coordonnees pour que l'exécution de la fonction main donne exactement l'affichage suivant :** [6.0, 7.8]

```
public class Coordonnees {
  private double x ;
  private double y ;

  public Coordonnees(double x, double  y){
    this.x = x ;
    this.y = y ;
  }

  public static void main(String[] args){
    System.out.println(new Coordonnees(6,7.8));
  }
} // Fin de la classe Coordonnees
```

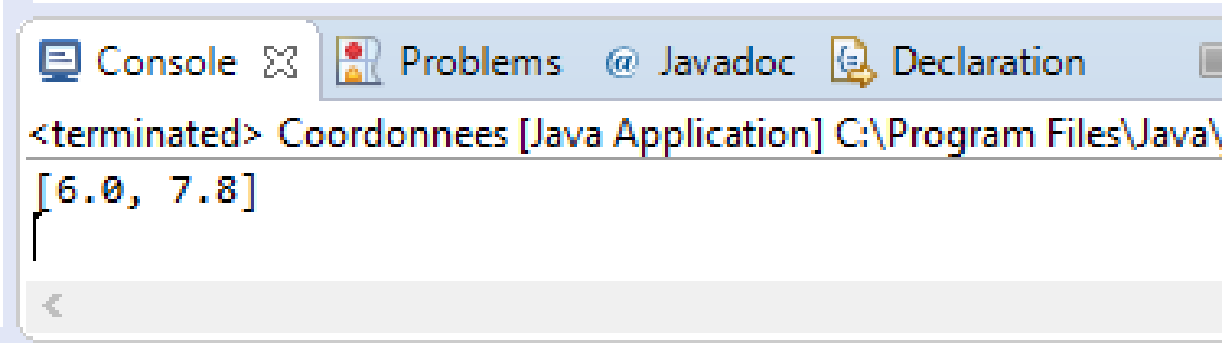
```java
public class Coordonnees {
	private double x;
	private double y;

	public Coordonnees (double x ,double y){
		this. x = x;
		this. y = y;
	}

	@Override
	public String toString() {
		return "[" + x + ", " + y + "]";
	}

	public static void main (String [] args){
		System.out.println( new Coordonnees (6, 7.8) );

	}
} // Fin de la classe Coordonnees
```




# Exercice V (8,5 points)

## #1

***Un dictionnaire est l'association d'un mot à ses différentes définitions.***

**Définissez une classe Dictionnaire qui permet de représenter cela.**

```java
import java.util.ArrayList;
import java.util.Map;
import java.util.HashMap;
public class Dictionnaire {
	private Map< String, ArrayList<String> > definitions;

	public Dictionnaire() {
		definitions = new HashMap< String, ArrayList<String> >();
	}
}
```

# Exercice V (8,5 points)

## #2

**Dans la classe Dictionnaire, définissez une méthode qui permet d'ajouter une définition à un mot du dictionnaire :**

public void ajouterDef(String mot, String def)

```
import java.util.ArrayList;
import java.util.Map;
import java.util.HashMap;
public class Dictionnaire {
	private Map< String, ArrayList<String> > definitions;

	public Dictionnaire() {
		definitions = new HashMap< String, ArrayList<String> >();
	}

	public void ajouterDef(String mot, String def) {
		if( !definitions.containsKey(mot) )
			definitions.put( mot, new ArrayList<String>() );
		definitions.get(mot).add(def);
	}
}
```

# Exercice V (8,5 points)

## #3

**Dans la classe Dictionnaire, définissez une méthode qui permet d'obtenir sous forme de String les différentes définitions d'un mot s'il apparaît dans le dictionnaire,**

**ou un message d'erreur ("Le mot XXX n'est pas dans le dictionnaire.", avec XXX le mot en question) dans le cas contraire**

public String chercher(String mot)

```java
import java.util.ArrayList;
import java.util.Map;
import java.util.HashMap;
public class Dictionnaire {
	private Map< String, ArrayList<String> > definitions;

	public Dictionnaire() {
		definitions = new HashMap< String, ArrayList<String> >();
	}

	public void ajouterDef(String mot, String def) {
		if( !definitions.containsKey(mot) )
			definitions.put( mot, new ArrayList<String>() );
		definitions.get(mot).add(def);
	}

	public String chercher(String mot) {
		ArrayList<String> definitionsMot = definitions.get(mot);
		/* Map.get(Object key)
		 * Returns the value to which the specified key is mapped,
		 * or null if this map contains no mapping for the key.
		 * Source : JavaDoc
		 */

		if( definitionsMot != null )
		{
			StringBuilder result = new StringBuilder(mot + " :");

			/* un mot du dictionnaire et ses définitions apparaissent sur une ligne,
			 * avec un nombre devant chaque définition
			 * (1 devant la première, 2 devant la deuxième,...).
			 */
			for(int i = 0; i < definitionsMot.size() ; i++)
				result.append( " " + i + " " + definitionsMot.get(i) );

			result.append("\n");
			return result.toString();
		}

		return "Le mot " + mot + " n'est pas dans le dictionnaire.";
	}
}
```

# Exercice V (8,5 points)

## #4

## On souhaite pouvoir afficher l'ensemble des mots d'un Dictionnaire avec leur définition,

**de la manière suivante : un mot du dictionnaire et ses définitions apparaissent sur une ligne, avec un nombre devant chaque définition (1 devant la première, 2 devant la deuxième,...). Un retour à la ligne sépare différents mots. Définissez une méthode qui permet d'obtenir une telle chaîne de caractère à partir d'un Dictionnaire.**

```java
import java.util.ArrayList;
import java.util.Map;
import java.util.HashMap;
public class Dictionnaire {
	private Map< String, ArrayList<String> > definitions;

	public Dictionnaire() {
		definitions = new HashMap< String, ArrayList<String> >();
	}

	public void ajouterDef(String mot, String def) {
		if( !definitions.containsKey(mot) )
			definitions.put( mot, new ArrayList<String>() );
		definitions.get(mot).add(def);
	}

	public String chercher(String mot) {
		ArrayList<String> definitionsMot = definitions.get(mot);
		/* Map.get(Object key)
		 * Returns the value to which the specified key is mapped,
		 * or null if this map contains no mapping for the key.
		 * Source : JavaDoc
		 */

		if( definitionsMot != null )
		{
			StringBuilder result = new StringBuilder(mot + " :");

			/* un mot du dictionnaire et ses définitions apparaissent sur une ligne,
			 * avec un nombre devant chaque définition
			 * (1 devant la première, 2 devant la deuxième,...).
			 */
			for(int i = 0; i < definitionsMot.size() ; i++)
				result.append( " " + i + " " + definitionsMot.get(i) );

			result.append("\n");
			return result.toString();
		}

		return "Le mot " + mot + " n'est pas dans le dictionnaire.";
	}

	@Override
	public String toString() {
		StringBuilder result = new StringBuilder("");

		for(String key : definitions.keySet())
			result.append( chercher(key) );
		return result.toString();
	}
}
```

# Exercice V (8,5 points)

## #5

**La question précédente permet d'afficher l'ensemble des mots d'un dictionnaire, mais sans respecter l'ordre alphabétique.**

**Définissez une classe DictionnaireAlphabetique qui hérite de Dictionnaire et qui garantit que les mots seront affichés dans l'ordre alphabétique.**

**Ajouter une méthode get dans la classe Dictionnaire**

```
	public Map< String,ArrayList<String> > getDefinitions(){
		return definitions;
	}
```

```
import java.util.List;
import java.util.ArrayList;
public class DictionnaireAlphabetique extends Dictionnaire{

	@Override
	public String toString() {
	/* La méthode keySet()  permet d'obtenir un ensemble de toutes les clés d'une collection Map.
	 * Mais les éléments de Set ne sont pas organisés dans aucun ordre, donc dans la première étape,
	 * nous copions tous les éléments de Set dans une collection ArrayList.
	 * Il existe une méthode appelée sort qui peut être exécutée sur une collection ArrayList et ainsi
	 * trier les mots du dictionnaire.
	 */
	 List<String> keyList = new ArrayList<String>();

		/* Set<String> Map.keySet()
		 * Returns a Set view of the keys contained in this map
		 */
		for(String key : getDefinitions().keySet())
			keyList.add( key );

		/* Parameter - the Comparator used to compare list elements.
		 * A null value indicates that the elements' natural ordering should be used
		 */
		keyList.sort(null);

		StringBuilder result = new StringBuilder("");
		for(int i = 0 ; i < keyList.size(); i++)
			result.append( chercher(keyList.get(i)) );
		return result.toString();
	}

}
```

# Exercice VI (1,5 points)

Définissez une méthode

**public double getMoyennePonderee(){ … }**

dans la classe Etudiant qui permet d'obtenir la moyenne d'un étudiant en tenant compte des coefficients des notes.

```
import java.util.Map;
import java.util.HashMap;
public class Etudiant {
	private Map<String, Note> notes;
	private String nom;

	public Etudiant(String nom){
		this.nom = nom;
		notes = new HashMap<String, Note>();
	}

	public void ajoutNote(String matiere, double note, int coef){
		if(!notes.containsKey(matiere)){
			notes.put(matiere, new Note(note, coef));
		}
	}

	public double getMoyennePonderee() {
		double moyennePonderee = 0;
		int sumCoef = 0;

		/* Collection<Note> java.util.Map.values()
		 * Returns a Collection view of the values contained in this map.
		 */
		for(Note note : notes.values()){
			moyennePonderee += ( note.getNote() * note.getCoef() );
			sumCoef += note.getCoef();
		}

		moyennePonderee = moyennePonderee / sumCoef;
		return moyennePonderee;
	}
}
```

```
public class Note {
	private double note;
	private int coef;

	public Note(double note, int coef){
		this.note = note;
		this.coef = coef;
	}

	public double getNote(){
		return note;
	}

	public int getCoef(){
		return coef;
	}
}
```