Nom Prénom :

Consignes relatives au déroulement de l'épreuve

Date :	29 mai 2017
Contrôle de :	**3IRC** - **Module « Programmation Orienté Objet » -**
Durée :	2h
Professeur responsable :	Françoise Perrin
Documents :	Supports de cours et TP autorisés Livres, calculatrice, téléphone, ordinateur et autres appareils de stockage de données numériques interdits.
Divers :	Les oreilles des candidats doivent être dégagées.

Rappels importants sur la discipline lors des examens

La présence à tous les examens est strictement obligatoire ; tout élève présent à une épreuve doit rendre une copie, même blanche, portant son nom, son prénom et la nature de l'épreuve.
Une absence non justifiée peut entrainer l'invalidation du module
Toute suspicion sur la régularité et le caractère équitable d'une épreuve est signalée à la direction des études qui pourra décider l'annulation de l'épreuve; tous les élèves concernés par l'épreuve sont alors convoqués à une épreuve de remplacement à une date fixée par le responsable d'année.
Toute fraude ou tentative de fraude est portée à la connaissance de la direction des études qui pourra réunir le Conseil de Discipline. Les sanctions prises peuvent aller jusqu'à l'exclusion définitive du (des) élève(s) mis en cause.

Recommandations

Pour chaque exercice, lisez bien tout l'énoncé avant de commencer.

Ne soyez pas impressionné par la longueur, il y a plus à lire qu'à écrire.

Les questions sont en gras et/ou numérotées.

Pour les questions qui attendent des réponses en français, ne récitez (recopiez ☹) pas le cours de manière théorique mais soyez **très près** de l'exemple. Par ailleurs, soyez **synthétique** mais **précis**.

Pour les instructions Java, la syntaxe n'a pas d'importance pourvu qu'elle soit compréhensible.

N'oubliez pas d'écrire votre nom sur le sujet ☺.

1° exercice : 4 points

Un groupe de composants Java joue à « Qui suis-je ? ».

Pour chacune des affirmations ci-dessous, **choisir une des réponses suivantes et écrire son N° dans le tableau**. Une réponse peut correspondre à plusieurs affirmations :

1. objet interne,
2. addActionListener(),
3. source d'événement,
4. paintComponent(),
5. composant Swing,
6. actionPerformed(),
7. interface écouteur,
8. JFrame,
9. événement.

Je suis le support de base de toute IHM.	
Tout type d'événement en a une.	
Je suis la méthode clé de l'écouteur d'actions.	
Vous lui ajoutez du code sans jamais l'appeler explicitement.	
Quand l'utilisateur fait quelque chose, c'est un...	
La plupart sont des sources d'événements.	
Une méthode addXxxListener( ) dit qu'un objet est une ...	
Je suis la méthode qui enregistre un écouteur d'actions.	
Je suis la méthode qui contient tout le code graphique d'un composant.	
Je suis généralement lié à une autre instance.	

2° exercice : 6 points + 2 bonus

Soit le programme suivant :

```
class Couleur {
	private char valeur;

	public Couleur(char c) {
		valeur = c;
	}

	public void afficher() {
		this.afficher(false);
	}
```

```
	public void afficher(boolean feminin) {
		switch (valeur) {
		case 'r':
			System.out.println("rouge");
			break;
		case 'v':
			System.out.print("vert");
			if (feminin) {
				System.out.println("e");
			}
			break;
		case 'b':
			System.out.print("bleu");
			if (feminin) {
				System.out.println("e");
			}
			break;
		case 'B':
			System.out.print("blanc");
			if (feminin) {
				System.out.println("he");
			}
			break;
		case 'n':
			System.out.print("noir");
			if (feminin) {
				System.out.println("e");
			}
			break;
		}
	}
}
// ----------------------------------------------------------------------

abstract class Carte {

	private int cout;

	public Carte() {
		cout = 0;
	}
	public Carte(int cout) {
		this.cout = cout;
	}

	public abstract void afficher();
}

// ----------------------------------------------------------------------

class Terrain extends Carte {
	private Couleur couleur;

	public Terrain(char c) {
		couleur = new Couleur(c);
		System.out.println("Un nouveau terrain.");
	}

	public void afficher() {
		System.out.print("Un terrain ");
		couleur.afficher();
		System.out.println();
	}

}
```

```
// ----------------------------------------------------------------------

class Creature extends Carte {
	private String nom;
	private int attaque;
	private int defense;

	public Creature(int cout, String nom, int attaque, int defense) {
		super(cout);
		this.nom = nom;
		this.attaque = attaque;
		this.defense = defense;
		System.out.println("Une nouvelle créature.");
	}
	public void afficher() {
		System.out.println("Une créature " + nom + " " + attaque + "/"
				+ defense + " ");
	}
}

// ----------------------------------------------------------------------

class Sortilege extends Carte {
	private String nom;
	private String description;

	public Sortilege(int cout, String nom, String desc) {
		super(cout);
		this.nom = nom;
		this.description = desc;
		System.out.println("Un sortilège de plus.");
	}
	public void afficher() {
		System.out.println("Un sortilège " + nom + " ");
	}
}

// ----------------------------------------------------------------------

class Jeu {
	private int nombreCartes;
	private Carte[] cartes;

	public Jeu(int nb) {
		nombreCartes = nb;
		cartes = new Carte[nb];
		System.out.println("On change de main");
	}
	/**
	 * Joue une carte après l'autre
	 */
	public void joue() {
		System.out.println("Je joue une carte...");
		int i = 0;
		while ((cartes[i] == null) && i < nombreCartes) {
			i++;
		}
		if ((i < nombreCartes) && (cartes[i] != null)) {
			System.out.println("La carte jouée est :");
			cartes[i].afficher();
			cartes[i] = null;
		} else {
			System.out.println("Plus de cartes");
		}
	}
```

```
	/**
	 * Ajoute une carte à  la collection
	 */
	public void piocher(Carte carte) {
		int i = 0;
		while ((i < nombreCartes) && (cartes[i] != null)) {
			i++;
		}
		if (i < nombreCartes) {
			cartes[i] = carte;
		} else {
			System.out.println("Nombre maximal de cartes atteint");
		}
	}

	public void afficher() {
		for (int i = 0; i < nombreCartes; ++i) {
			if (cartes[i] != null) {
				cartes[i].afficher();
			}
		}
	}
}

// ----------------------------------------------------------------------

public class Magic {
	public static void main(String[] args) {
		Jeu maMain = new Jeu(10);

		maMain.piocher(new Terrain('b'));
		maMain.piocher(new Creature(6, "Golem", 4, 6));
		maMain.piocher(new Sortilege(1, "Croissance Gigantesque",
				"La créature ciblée gagne +3/+3 jusqu'à  la fin du tour"));

		System.out.println("Là , j'ai en stock :");
		maMain.afficher();
		maMain.joue();
	}
}
```

1. **Donnez la trace d'exécution** (ce qui est affiché sur l'écran) **du programme ci-dessus.**

2. Imaginons que la classe jeu ne contienne pas un tableau de cartes mais une liste de carte implémentée dans une ArrayList<Cartes>. Le début de la classe pourrait s'écrire ainsi :

```
class Jeu {
	private int nombreCartes;
	private List<Cartes> cartes;

	public Jeu(int nb) {
		nombreCartes = nb;
		cartes = new ArrayList<Cartes>();
		System.out.println("On change de main");
	}
```

a. Réécrivez la méthode afficher() de la classe Jeu en tenant compte de ce changement.

b. Faut-il modifier la fonction main() ? Pourquoi ?

c. Au regard de ce qui précède, comment pourriez-vous expliquer le principe d'encapsulation – soyez très près de l'exemple ?

3° exercice : 10 points

Pour cet exercice, les questions sont relatives à la portion de code identifiée par un numéro dans le listing du programme suivant. **Les questions proprement dites sont écrites en gras.**

Les réponses doivent être brèves : 1 phrase.

```
/**
 * La classe "Grammaire"
 * doit permettre de retirer d'une liste de mots "candidates"
 * ceux qui ne sont pas présents dans une deuxième liste, "references".
 */
public class Grammaire {

	private List<String> candidates;
	private List<String> reference;

	/**
	 * @param candidates liste de mots candidats
	 * @param references liste de référence des mots autorisés
	 */
	public Grammaire(List<String> candidates, List<String> references) {
		this.candidates = new ArrayList<String>( candidates);
		this.references = new ArrayList<String>( references);
	}

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

	public void nettoie() {

		Iterator<String> i = candidates.iterator();
		while (i.hasNext()) {
			String s = i.next();

			- Q 1 -
			if (!( isAllowedA(s) || isAllowedB(s) || isAllowedC(s) )) {

				- Q 4 -
				/* Instruction A */ i.remove();
				/* Instruction B */ candidates.remove(s);
				/* Instruction C */ s = null;
			}
		}
	}
	/** Methode A
	 * @return true si 'str' fait partie des mots autorisés
	 */
	private boolean isAllowedA(String str) {
		boolean ret = false;
		for (String s : references) {
			if(s.equals(str)) {          - Q 2 -     - Q 3 -
				ret = true;
			}
		}
		return ret;
	}
```

```
	/** Methode B
	 * @return true si 'str' fait partie des mots autorisés
	 */
	private boolean isAllowedB(String str) {
		boolean ret = false;
		int i = 0;
		while (i<references.size()) {
			if(references.get(i).equals(str)) {
				ret = true;
				break;
			} else {
				i++;
			}
		}
		return ret;
	}
	/** Methode C
	 * @return true si 'str' fait partie des mots autorisés
	 */
	private boolean isAllowedC(String str) {
		return reference.contains(str);
	}
	public static void main(String[] args) {

		// Crée 2 listes à partir des valeurs d'un tableau
		List<String> vals = new ArrayList<String>(
				Arrays.asList("hibou","chou","loup",null,"genou"));
		List<String> pluralWithX = new ArrayList<String>(
				 Arrays.asList("hibou","chou","caillou","genou"));

		Grammaire gram = new Grammaire(vals, pluralWithX);
		System.out.println(gram);// Affiche "[hibou, chou, loup, null, genou]"

		gram.nettoie();
		System.out.println(gram); // Affiche "[hibou, chou, genou]"

		System.out.println(vals);     - Q 5 -
		// Affiche "[hibou, chou, loup, null, genou]"
	}
}
```

- Q 1 -

Parmi les 3 méthodes proposées (`isAllowedA()`, `…B()`, `…C()`), dites celles que l'on peut utiliser pour obtenir le résultat attendu et/ou celles qui ne doivent pas être écrites et pourquoi.

- Q 2 -

Pourrait-on aussi écrire "`if (s==str)`" ? Justifiez.

- Q 3 -

Pourrait-on aussi écrire "`if (str.equals(s))`" ? Justifiez.

- Q 4 -

Parmi les 3 instructions proposées (A, B, C), laquelle conserver pour obtenir le résultat attendu ? Justifiez votre choix relativement à chaque instruction.

Exemple :« Instruction W : OUI car… Instruction K : NON attendu que… Instruction Y : etc...

- Q 5 -

L'affichage nous montre qu'on a encore 5 éléments dans la liste référencée par `vals`. Pourquoi sont-ils encore tous présents ?