Nom Prénom :

Consignes relatives au déroulement de l'épreuve

Date :	28 mai 2018
Contrôle de :	**3IRC - Module « Programmation Orienté Objet » - DS1**
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.

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 : 14 points

Soit le programme suivant :

```
public class Match {

	public static void main(String[] argv){
		String names[] = {"Griezmann", "Payet", "Pogba", "Giroud", "Mbappé"};
		Football game = new Football(names);

		game.envoyerLaBalle("Pogba");
		game.faireUnePasse("Pogba", "Griezmann");
		game.quiALaBalle();
		game.faireUnePasse("Griezmann", "Mbappé");
		game.quiALaBalle();
		game.donnerCarton("Giroud");
		game.checkExpulsion("Giroud");
		game.faireUnePasse("Giroud", "Pogba");
		game.donnerCarton("Giroud");
		game.checkExpulsion("Giroud");
		game.faireUnePasse("Mbappé", "Giroud");
	}
}
class Football {
	JoueurFoot[] joueurs;

	public Football(String[] names) {
		super();
		joueurs = new JoueurFoot[10];
		for(int i=0; i<names.length; i++){
			joueurs.add(new JoueurFoot(names[i]));
		}
	}
	public void envoyerLaBalle(String name) {
		JoueurFoot joueur = chercheJoueur(joueurs,  name);
		joueur.recevoirLaBalle();
	}
	public void faireUnePasse(String name1, String name2) {
		JoueurFoot joueur1 = chercheJoueur(joueurs,  name1);
		JoueurFoot joueur2 = chercheJoueur(joueurs,  name2);
		if (joueur1.getPossession()){
			if (!joueur2.getExpulsion()){
				joueur1.perdreLaBalle();
				joueur2.recevoirLaBalle();
			}
			else {
				System.out.println("Le joueur " +
						joueur2.getName() +
						" ne peut recevoir la balle car il est expulse.");
			}
		}
		else {
			System.out.println("Le joueur " +
						joueur1.getName() + " ne possede pas le ballon.");
		}
	}
	public void quiALaBalle() {
		for (int i = 0; i <joueurs.length; i++){
			if (joueurs[i].getPossession()){
				System.out.println("C'est le joueur " +
						joueurs[i].getName() + " qui a la balle.");
				break;
			}
		}
	}
```

```
	public void donnerCarton(String name) {
		JoueurFoot joueur = chercheJoueur(joueurs,  name);
		joueur.recevoirCarton();
	}
	public void checkExpulsion(String name) {
		JoueurFoot joueur = chercheJoueur(joueurs,  name);
		if (joueur.getNombreCartons() == 2){
			joueur.expulsion();
		}
	}
	private  JoueurFoot chercheJoueur(JoueurFoot[] tab, String name){
		for (int i = 0; i <tab.length; i++){
			if (tab[i].getName().equals(name)){
				return tab[i];
			}
		}
		return null;
	}
}

class JoueurFoot {

	private boolean possessionDeLaBalle;
	private int cartonJaune;
	private boolean expulsion;
	private String name;

	public JoueurFoot(String name) {
		this.possessionDeLaBalle = false;
		this.cartonJaune = 0;
		this.expulsion = false;
		this.setName(name);
	}
	public void recevoirLaBalle(){
		this.possessionDeLaBalle = true;
	}
	public void recevoirCarton(){
		this.cartonJaune++;
	}
	public void perdreLaBalle(){
		this.possessionDeLaBalle = false;
	}
	public boolean getPossession(){
		return this.possessionDeLaBalle;
	}
	public int getNombreCartons(){
		return this.cartonJaune;
	}
	public void expulsion(){
		this.expulsion = true;
	}
	public boolean getExpulsion(){
		return this.expulsion;
	}
	public String getName() {
		return name;
	}
	private void setName(String name) {
		this.name = name;
	}
}
```

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

2. **Combien d'instances de la classe `Football` et de la classe `JoueurFoot` ont été créées ? Justifiez.**

3. Dans la classe `JoueurFoot` il n'y a pas de « setter » sur les attributs ou alors « private ». **Ce mode de conception vous parait-il normal ? Justifiez.**

4. **Que fait la méthode `chercheJoueur()` ?**

5. **Pourquoi la méthode `chercheJoueur()` est-elle déclarée avec le qualificateur « `private` » ?**

6. **Dans la méthode `chercheJoueur()` aurait-on pu écrire**
   ```
   if (tab[i].getName() == equals(name))
   ```
   au lieu de
   ```
   if (tab[i].getName().equals(name))
   ```
   ? Expliquez pourquoi.

7. **Est-il utile de passer le tableau de `JoueurFoot` en paramètre à la méthode `chercheJoueur()` dans ce contexte ? Justifiez. Existe-t-il des contextes où cela peut être pertinent de passer un attribut à une méthode de la classe qui le déclare ? Justifiez.**

8. Imaginons que la classe `Football` ne contienne plus un tableau de `JoueurFoot` mais une `List` de `JoueurFoot` implémentée dans une `LinkedList<JoueurFoot>`. Le constructeur de la classe pourrait s'écrire ainsi :

```
public Football(String[] names) {
	super();
	joueurs = new LinkedList<JoueurFoot>() ;
	for(int i=0; i<names.length; i++){
		joueurs.add(new JoueurFoot(names[i]));
	}
}
```

   a. **Réécrivez la méthode `quiALaBalle()` de la classe `Football` en tenant compte de ce changement.**

   b. **Faut-il réécrire d'autre(s) méthode(s) de la classe `Football` ? Si oui la(les)quelle(s) ? Justifiez.**

   c. **Faut-il modifier la fonction `main()` ? Pourquoi ?**

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

2° exercice : 6 points

Soit le programme suivant :

```
public class Animaux {
	public static void main(String[] args) {
		AnimalDs titi = new Canari(3);
		titi.vieillir();
		titi.vieillir(2);
		titi.crier();
	}
}
interface AnimalDs {
	public void crier();
	public void vieillir() ;
	public void vieillir(int a) ;
}
abstract class AbstractAnimal implements AnimalDs {
	private boolean vivant;
	private int age;
	private int ageMaxi = 10;
	public AbstractAnimal() {
		this(1);
	}
	public AbstractAnimal (int a) {
		age = a;
		vivant = true;
		System.out.println("Un animal de " + a  + " an(s) vient d'être créé");
	}
	public abstract void crier();

	public void vieillir() {
		vieillir(1);
	}
	public void vieillir(int a) {
		age += a;
		afficheAge();
		if (age > ageMaxi) {
			mourir();
		}
	}
	private void afficheAge() {
		System.out.println("C'est l'anniversaire de cet animal.");
		System.out.println("Il a maintenant " + age + " an(s)");
	}
	private void mourir() {
		vivant = false;
		System.out.println ("Cet animal est mort");
```

```
	}
}
class Canari extends AbstractAnimal {
	Canari(int a) {
		super(a);
	}
	public void crier() {
		System.out.println("Cui-cui !");
	}
}
```

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

2. **A quoi sert l'interface** `AnimalDs` (votre réponse doit permettre de répondre à la question plus générale « à quoi sert une classe interface ») **?**

3. **A quoi sert la classe** `AbstractAnimal` (votre réponse doit permettre de répondre à la question plus générale « à quoi sert une classe abstraite ») **?**