UEF4.3. Programmation Orientée Objet

EXAMEN FINAL

02 heures-Documents interdits

Exercice 1 : (6pts)

Soit les classes suivantes :

```
Abstract class Crayon {public abstract String ecrire() ;}
interface Effacable {public void effacer();}
class CrayonGraphite extends Crayon implements Effacable{
public void effacer(){System.out.println("effacer trait");}
public String ecrire(){return"Graphite";}
}
class CrayonGraphiteDur extends CrayonGraphite {
public String ecrire(){return"Gris";}
}
class CrayonGraphiteGras extends CrayonGraphite {}
class CrayonCouleur extends Crayon {
public String ecrire(){return"Couleur";}
}
class CrayonBleu extends CrayonCouleur {
public String ecrire(){return"Bleu";}
}
class CrayonBleuCiel extends CrayonBleu {
public String ecrire(){return super.ecrire()+" Ciel";}
}
class CrayonRouge extends CrayonCouleur {}
```

1. Tracez le diagramme de classes en précisant les types des classes et les relations entre elles **(1pt)**
2. Qu'affiche le programme suivant ?**(1,25pts)**

```
Public class Exercice{
  Public static void main(String args[]){
    Crayon[] trousse = new Crayon[5];
    trousse[0] = new CrayonBleuCiel();trousse[1] = new
    CrayonGraphiteDur();
    trousse[2] = new CrayonRouge();trousse[3] = new CrayonGraphiteGras();
    trousse[4] = new CrayonBleu();
    for (inti=0; i<trousse.length; i++)
        System.out.println("Crayon "+trousse[i].ecrire());  }
}
```

2. On effectue les affectations suivantes. Pour chaque affectation, dites si elle s'exécute sans erreur, si elle provoque une erreur à la compilation ou si elle provoque une erreur à l'exécution. S'l y a une erreur expliquez pourquoi. **(3,75 pts)**

```
CrayonCouleur c0 = trousse[0];
CrayonRouge c2 = trousse[2];
CrayonBleu c3= (CrayonBleu) trousse[3];
Crayon c4 = trousse[4];
c0 = trousse[1];
```


Scanned by CamScanner

```
CrayonGraphite c5 = trousse[0];
c3 = trousse[0];
Effacable e1=  trousse[1];
```

Exercice 2 : (14pts)

Un syndicat d'employés d'une grande entreprise souhaite numériser le vote de son bureau et son président afin d'assurer l'anonymat et avoir rapidement les résultats. Le bureau du syndicat est composé de représentant de chaque fonction de l'entreprise (ingénieur, technicien, chef de projet, directeur).

Le scrutin est caractérisé par une date, un état (non ouvert, en cours, clos), la liste des candidats par fonction et la liste des votants qui sont fixées à la création.

La liste des votants comporte tous les employés de l'entreprise, où chacun peut choisir un représentant au bureau parmi les candidats correspondant à sa fonction: les techniciens ne peuvent voter que pour un candidat technicien, les ingénieurs votent pour un candidat ingénieur, …etc. Ces candidats sont donc des employés de l'entreprise qui vont aussi voter.

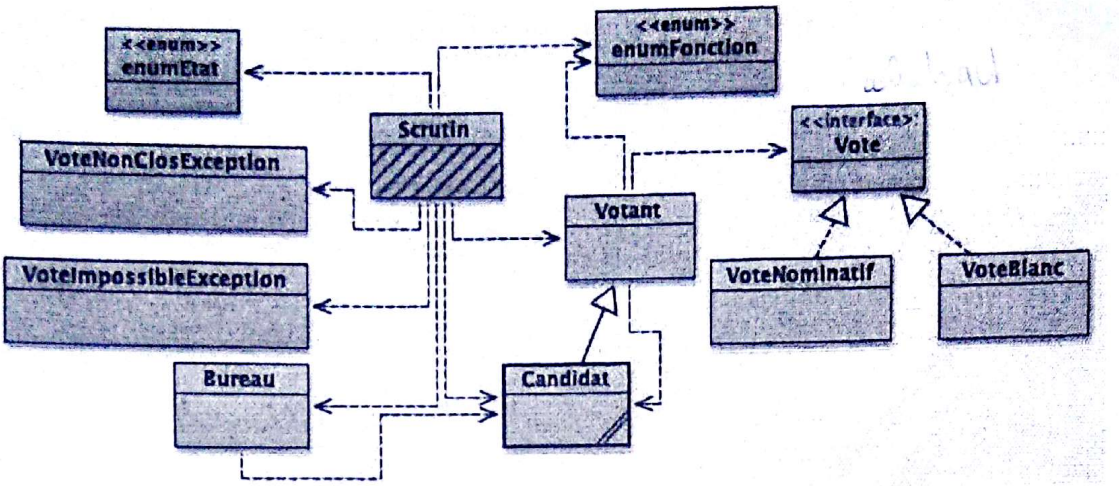
Un votant est caractérisé par un identifiant et sa fonction en tant 'employé dans l'entreprise. Il ne peut voter qu'une seule fois par scrutin et aucun vote n'est possible une fois que le scrutin a été déclaré clos. Le choix d'un votant peut porter sur un candidat (vote nominatif) comme il peut être blanc. Ce choix se fait via sa méthode ***public Vote choisir(Set<Candidat> choixPossible)*** qui est appelée par la méthode ***voter*** de la classe Scrutin dont l'entête est définie ainsi : ***public final void voter(Votant votant) throws VoteImpossibleException***. Le code de ces méthodes doit assurer l'anonymat.

Pour déterminer le bureau du syndicat, la classe Scrutin dispose de la méthode ***getBureau()***qui retourne la liste des candidats qui ont remporté le scrutin, et qui choisit le président selon le candidat ayant le plus d'année d'expérience dans l'entreprise (peu importe sa fonction). On dispose également de la méthode ***getStatistique()*** qui affiche le nombre de votants, le nombre de votes blancs, le nombre de voix par fonction et par candidat. Une exception ***VoteNonClosException*** est levée si ces méthodes sont invoquées avant la clôture du scrutin. On suppose qu'il ne peut y avoir d'égalité à un scrutin.

Le scrutin doit également proposer les méthodes :***tauxParticipation()*** qui retourne à tout moment le taux de participation au scrutin, c'est-à-dire le pourcentage de votants par rapport au nombre d'inscrits au scrutin ; ***depouille()*** qui lorsqu'elle est appelée clôt le scrutin et comptabilise pour chaque candidat son score.

1. a. Quels sont l'intérêt et l'importance du qualificatif final dans la méthode *voter* ?**(1pt)**
   b. Quand l'exception *VoteImpossibleException* doit-elle être déclenchée ? **(0,5 pt)**
2. En s'intéressant uniquement au système de vote (pas aux employés et à la gestion d'entreprise) nous vous proposons la modélisation orientée objet présentée dans le diagramme de classe suivant :

Scanned by CamScanner



a. Pourquoi la classe Candidat hérite-elle de la classe Votant ?**(0,25)**
b. Quel est l'intérêt de créer un héritage entre VoteBlanc, VoteNominatif et Vote ? **(0,5)**
c. Expliquez brièvement quelle collection doit-on utiliser pour :**(1,5)**
   i. assurer qu'un vote n'est enregistré qu'une fois ?
   ii. récupérer l'ensemble des candidats par fonction ?
   iii. vérifier qu'un votant a déjà voté ?
d. Comment peut-on assurer l'anonymat du vote ? **(0.5)**

3. Détaillez dans un tableau les attributs, constructeurs et méthodes des classes Scrutin, Vote et ses dérivés (VoteBlanc et VoteNominatif),Votant, Candidat et Bureau.**(7,75 pts)**
4. Donnez le code JAVA de la méthode *voter* de la classe Scrutin**(2pts)**.

Scanned by CamScanner