esprit Ecole Supérieure Privée d'Ingénierie et de Technologies	**Année Universitaire : 2014-2015** **Examen principal**

Module : Conception par objet et programmation Java		**Documents autorisés** : Non
Enseignants: Equipe java		**Nombre de pages** : 4
Date : 07/01/2015	**Heure :**9h	**Durée** : 1h30
Classes : **3 info** A,**3 SIGMA**,**4INFNI**		

Partie1 : (3 points)

1 -Corrigez les erreurs du code ci-dessous sachant qu'il ya **4 erreurs.**

2-Complétezles codes numérotés en utilisant les propositions suivantes:

`getInt("prix"),getString(1),rs.next(),getString("immatriculation")`

`getString("libelle"),getInt(1),rs.hasNext(),getInt(0),getString(2)`

Indication : La table Voiture contient l'immatriculation de type varchar(20)et le prix de type int

NB : Vous n'avez pas besoin de reproduire le code,vous écrivez que les lignes à **c**orriger ou et les numéros à**c**ompléter

```
import java.sql.*;
public class ConnexionBD {
1-privatestatic String url = "jdbc:mysql://localhost:3306/esprit ;
 2-  privatestatic String user = "root";
 3-  privatestatic String pwd = "root";
4-  public staticvoidmain(String[] args) {
5-try {
6-          Connectioncnx = DriverManager.getConnection(url,user, pwd)
7-          Statement st = null;
 8-          st = cnx.createStatement();
String sqlInsertion = "insert into Voiture (immatriculation,prix)values('Tunis
2020',20000)";
9-          st.execute~~Query~~(sqlInsertion); or executeUpdate(sql
10-         String sqlLecture = "Select * from Voiture ";
11-         ResultSetrs = st.executeUpdate(sqlLecture); executeQuery(sql)
12-       ..... (1)...while(rs.next()) ..... {
13-System.out.println("Immatriculation :"
+rs.getString(1)+"Prix:"+rs.getInt(2));
14-         }
      } catch (SQLException ex) {
System.out.println(ex);
        }
```

0.5

0.5

0.5

0.5

0.5

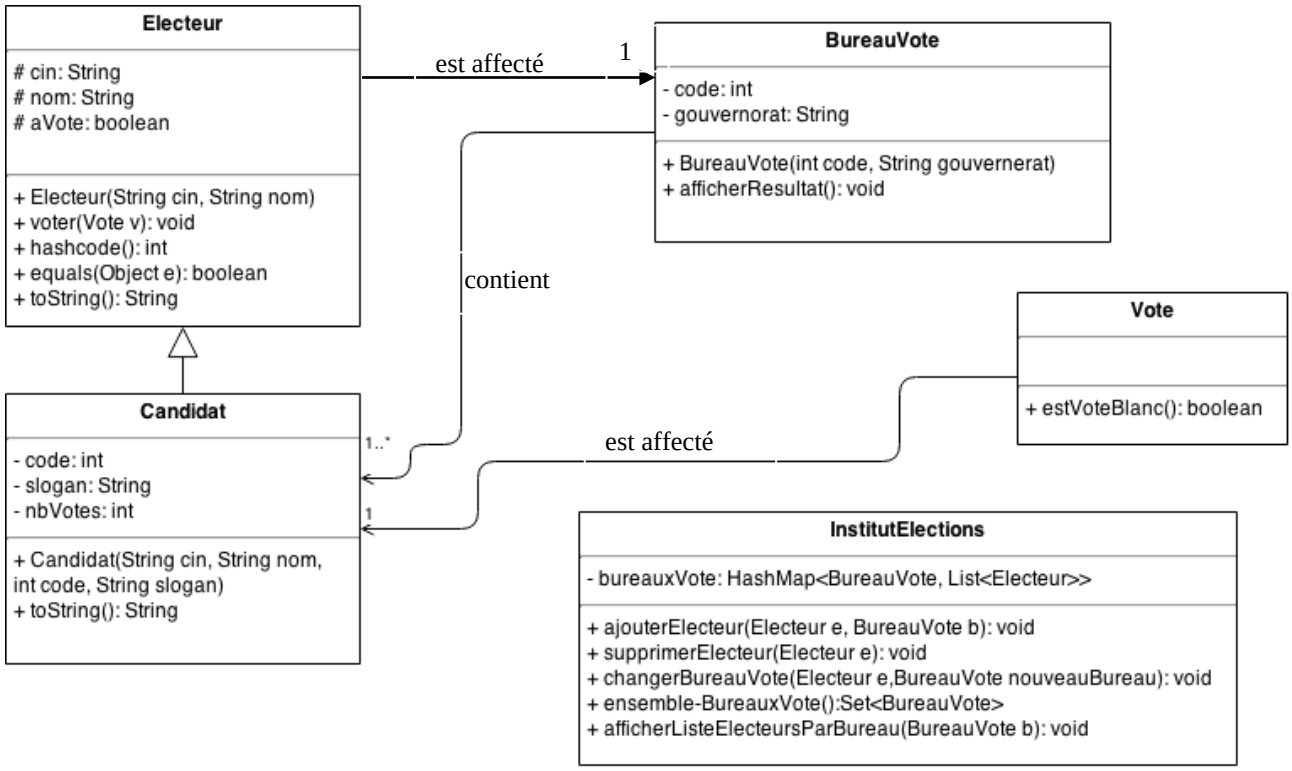
0.5

```
    } }
```

SYSTEME DES ÉLECTIONS PRESIDENTIELLES (17 POINTS)

En prévision des élections de 2019, le service informatique de l'ISIE souhaite mettre en place une application gérant les données concernant les élections présidentielles.

Le diagramme declasse suivant présente les principales classes de l'application ainsi que les différentes relations entre celles-ci :



NB : <u>On admet que les getters et les setters de toutes les classes sont déjà implémentés</u>. Vous pouvez ajouter des classes en cas de besoin.

Chaque électeur est affecté à un bureau de vote.

Question 1 :

Implémentez la classe **<u>Vote:</u>**

- La méthode ***booleanestVoteBlanc()***
    - Permet de déterminer si le vote est blanc. Un vote est considéré blanc lorsqu'il est dépourvu de tout candidat.(n'est pas attribué à aucun candidat).

```
public class Vote {

    private Condidat c;

    public boolean estVoteBlanc() {
        if (c == null) {
            return true;
        }
        return false;
    }
}
```

0.25

0.75

Question 2 :

Implémentez la classe ***VoteImpossibleException*** 1

```
public class VoteImpossibleException extends Exception {

    public VoteImpossibleException(String message) {
        super(message);
    }
}
```

Question 3:*(2,5 points)*

Implémentez la classe **Electeur**

- Le constructeur paramétré
- Permet d'initialiser les attributs ***cin*** et ***nom***.

```
public class Electeur {

    private String cin;
    private String nom;
    private boolean aVote;
    private BureauVote bureauVote;

    public Electeur(String cin, String nom) {

        this.cin = cin;
        this.nom = nom;
    }
```

0.25

0.25

0.5

- Les méthodes ***hashCode*** et ***equals***
  - Le critère d'unicité des électeurs est le cin.

```java
public int hashCode() {
    return 5;
}

@Override
public boolean equals(Object obj) {
    if (obj != null && (obj instanceof Electeur)) {
        Electeur e = (Electeur) obj;
        return e.cin.equals(cin);
    }
    return false;
}
```

0.25

0.75

- La méthode ***toString*** qui permet de retourner le cin et le nom

```java
@Override
public String toString() {
    return cin + " " + nom;
}
```

0.25

Question 4 :*(1 point)*

Implémentez la classe **Candidat:**

- Le constructeur paramétré
- La méthode ***toString :*** elle permet de retourner le cin ,le nom, le slogan et le nombre de votes***(0.5 point)***

```java
public class Condidat extends Electeur implements Comparable<Condidat>{
    private int code;
    private String slogan;
    private int nbVotes;

    public Condidat(int code, String slogan, String cin, String nom) {
        super(cin, nom);
        this.code = code;
        this.slogan = slogan;
        nbVotes=0;
    }
   @Override
    public String toString() {
        return  super.toString()+ " "+slogan+" "+nbVotes;
    }
```

si nbVotes est en paramètre du constructeur note =0.25 else 0.5

0.5

Question 5 :

- La méthode ***booleanvoter(Vote vote)***
  - Un électeur n'a le droit de voter qu'une seule fois. Dans le cas échéant, une exception de type ***VoteImpossibleException*** sera levée. Le vote est

comptabiliséaubureau de vote auquel le lecteur appartient, et **le nombre de votes**relatif au candidat choisi**sera incrémenté de 1**.

```
public void voter(Vote vote) throws VoteImpossibleException {
    if (aVote) {
        throw new VoteImpossibleException("vote IMPOSSIBLE!!!");
    } else {
        aVote = true;
        Condidat condidat = vote.getC();
        condidat.setNbVotes(condidat.getNbVotes() + 1);
        bureauVote.getCondidats().add(condidat);
    }
}
```

0.25 | 0.5 | 025 | 0.5 | 0.5 | 0.5

Question 6 :

Implémentez la classe ***BureauVote*** regroupantun ensemble de candidats

```
public class BureauVote {
    private int code;
    private String gouvernorat;
    private Set<Condidat> Condidats;
```

0.5

- Les candidats sont **triés** selon le nombre de votes dans l'ordre décroissant et la **duplication**des candidats **n'est pas permise**.

```
@Override
public int compareTo(Condidat o) {
  if(o.getNbVotes()==nbVotes){
      return code-o.getCode();
  }else{
      return o.getNbVotes()-nbVotes;
  }

}
```

1

- Le constructeur paramétré
    - Permet d'initialiser les attributs ***code***et ***gouvernorat***

```
public BureauVote(int code, String gouvernorat) {
    this.code = code;
    this.gouvernorat = gouvernorat;
    this.Condidats= new TreeSet<>();
}
```
0.5

- La méthode **voidafficherResultat()*(1 point)***
    - Permet d'afficher le résultat des candidatsdu bureau

```
public void afficherResultat(){
    Iterator it= Condidats.iterator();
    System.out.println("****Resultat bureau "+gouvernorat+"*******");
    while (it.hasNext()) {
        System.out.println(it.next());
    }
}
```
1

Question 7: (6.5 ***points)***

Implémentez la classe **<u>InstitutElections</u>** ***(0.5 points) (voir le diagramme de classes)***

```
public class InstituElection {

    private HashMap<BureauVote, List<Electeur>> bureauxVote;

    public InstituElection() {

        bureauxVote = new HashMap<>();
    }
```
0.5

- La méthode ***Set<BureauVote>ensembleBureauVote()***
    - Permet de récupérer un ensemble des bureaux de vote.

```
public Set<BureauVote> ensembleBureauVote() {
    return bureauxVote.keySet();
}
```
0.5

- La méthode ***afficherListeElecteurParBureau(BureauVote b)(1.5 points)***
    - Permet d'afficher la liste triée des électeurs selon leurs ***CIN***

```
public void afficherListElecteurParBureau(BureauVote b) {

    Collections.sort(bureauxVote.get(b), new ElecteurComparator
    for (Electeur e : bureauxVote.get(b)) {
        System.out.println(e);
    }
}
```
0.5 classe comparator

0.5

0.5

- La méthode ***ajouterElecteur(Electeur e, BureauVote b)(1.5 points)***
    - Permet d'affecter un électeur dans un bureau du vote après avoir vérifié que :
        - Le bureau de vote existe réellement
        - L'électeur n'est déjà affectéà aucun bureau.

```
public void ajouterElecteur(Electeur e, BureauVote b) {
    if (bureauxVote.containsKey(b)) {
        if (!bureauxVote.get(b).contains(e)) {
            bureauxVote.get(b).add(e);
            e.setBureauVote(b);
        }

    }
}
```

0.5 0.5 0.5

- La méthode ***supprimerElecteur(Electeur e)(1.5 points)***
    - Permet de supprimer un électeur déjà inscritdans un bureau de vote.

```
public void supprimerElecteur(Electeur e) {
    Iterator it = bureauxVote.entrySet().iterator();
    while (it.hasNext()) {

        Map.Entry mp = (Map.Entry) it.next();
        BureauVote b = (BureauVote) mp.getKey();
        List<Electeur> electeurs = (List<Electeur>) mp.getValue();
        if (electeurs.contains(e)) {
            bureauxVote.get(b).remove(e);
        }

    }
}
```

- La méthode ***changerBureauVote(Electeur e, BureauVotenouveauBureau)(1 point)***
    - Déduire le traitement de cette méthode sachant qu'elle permet à un électeur de changer de bureau de vote.

```
public void changerBureauVote(Electeur e, BureauVote b) {
    supprimerElecteur(e);
    ajouterElecteur(e, b);
}
```

Bon travail ☺