esprit Ecole Supérieure Privée d'Ingénierie et de Technologies	**Année Universitaire : 2011-2012** **Examen principale**

Module : Conception orientée objet et programmation Java		**Documents autorisés** : Non
Enseignants : Sofiene G,Rochdi R,Adel KDIDI,Ibrahim B, Saif B, Bassem H ,Ibtihel S,Imen B,Sana BF,Emna BC		**Nombre de pages** :
Date : Samedi 28 Janvier 2012	**Heure :** 9h	**Durée** : 1h30
Classes : **3 info** A1,2,4,5,6,7,8,9,10,11,12,13 **4 info** B1,2,3,4,5 **4 Tel** B1,2 **4INFNI**		

GESTION DE PRODUIT INDUSTRIEL

Nous souhaitons développer une application simplifiée de gestion de produit industriel. Ainsi, nos données seront organisées comme suit :

- Un produit contient un ensemble d'assemblages.
- Un assemblage est composé d'un ensemble de pièce.
- Une pièce peut être soit une pièce mécanique ou une pièce électronique.

Pour réaliser notre application, nous vous demandons de compléter les classes présentées ci-dessous sachant que la solution doit contenir les **exceptions** suivantes :

- Une exception **VisualisationException** signalant une erreur de visualisation de pièce.
- Une exception **SupressionException** en cas de suppression d'une pièce qui n'existe.

Les parties à compléter sont numérotées de 1 à 19. Vous devez *mettre dans votre copie le numéro correspondant à chaque code* ajouté. Vous *mettez juste la partie à compléter* sans réécrire aucune ligne de code déjà donnée dans l'énoncé.

Travail demandé :

En précisant le numéro de chaque partie à ajouter :

1) Complétez la classe **Piece**, **PieceElectrique, PieceMecanique.** Deux pièces sont égales en cas d'égalité de référence.
2) Complétez la classe **Assemblage** regroupant un ensemble de pièces :
   - **ajouterPiece**() : permet d'ajouter une pièce (la duplication des pièces est permise).
   - **rechercherPiece**() : permet de rechercher une pièce à partir de sa référence.
   - **supprimerPiece**() : permet de supprimer une pièce à partir de sa référence.
   - **Visualiser**() : permet de visualiser les pièces de l'assemblage.
3) Complétez la classe **Produit** regroupant un ensemble d'assemblage.
   - Un Produit possède un libelle.
   - Un Produit peut avoir plusieurs assemblages. Nous souhaitons repérer un assemblage à partir d'un code.
   - Construction d'un Produit en initialisant son libellé (String).
   - **ajouterPiece** () : permet d'ajouter une pièce dans un assemblage à partir de son code. Cette méthode recevra en paramètre le code de l'assemblage dans le quel la pièce sera insérée et la pièce à

insérer. N.B : il faut prendre en compte le cas où le code n'existe pas, dans ce cas il faut ajouter un nouvel assemblage.

- **supprimerPiece**() : supprimer une pièce dans un assemblage à partir de son code.
- **Visualiser**() : visualiser un produit.

4) Ecrire un programme de test qui :

- Ajouter p1 et p2 dans un assemblage de code `7690`
- Ajouter p2, p3 et p4 dans un assemblage de code `7691`
- Visualiser le produit « prod »

Remarque : Toutes les classes sont présentées dans l'ANNEXE ci-dessous, vous êtes amenés à compléter juste le code incomplet.

ANNEXE :

```
public interface Visualisable {
public void visualiser()throws VisualisationException;
}
```

```
public class Piece implements Visualisable{
	protected String reference;
	protected String matiere;
	protected float poid;
	public Piece(String reference, String matiere, float poid) {
	/*à completer*/ (1) (0,5pt)
	}
	public void visualiser() throws VisualisationException {
		System.out.println("REFERENCE: "+reference);
		System.out.println("MATIERE: "+matiere);
		System.out.println("POID" +poid);
	}
	public String getReference() {
		return reference;
	}
	public void setReference(String reference) {
		this.reference = reference;
	}
	public float getPoid() {
		return poid;
	}
	public void setPoid(float poid) {
		this.poid = poid;
	}
	public void setMatiere(String matiere) {
		this.matiere = matiere;
	}
	public String getMatiere() {
		return matiere;
	}
	public boolean equals(Object obj) {/*à completer*/ (2) (1pt)}
```

```
public class PieceElectrique extends Piece{
	public boolean hasElectricPlan;
	public PieceElectrique(String reference, String matiere, float poid,
			boolean hasElectricPlan) {
		/*à completer*/(3) (1,5pt)
	}
```

```
	public void visualiser()/*à completer*/ (4) (0,5pt){
		if(hasElectricPlan){
			/*à compléter*/(5)(1pt)
		}
		else{
		throw new VisualisationException(« piéce non éléctronique ») ;
		}
	}
}
```

```
public class PieceMecanique extends Piece{
	public boolean hasMecanicalPlan ;
	public PieceMecanique(String reference, String matiere, float poid,
			boolean hasMecanicalPlan) {
	/*à compléter*/(6)(0,5pt)
	}
	public void visualiser()/*à completer*/ (7) (0,5pt){
	 if(hasMecanicalPlan){
		/*à compléter*/(8)(0,5pt)
		}
		else{
			throw new VisualisationException("piéce non mécanique");
		}
	}
}
```

```
public class Assemblage implements Visualisable {
	private List<Piece> pieces;

	public Assemblage(){
		/*à compléter*/ (9) (1pt)
	}
	public void ajouterPiece(Piece piece) {
		/*à compléter*/ (10) (1pt)    }
	public Piece rechercherPiece(String reference) {
		/*à compléter*/ (11) (2pt)
	}
	public void supprimerPiece(String reference)/*à compléter*/(12) (0,5pt){
		Piece piece = rechercherPiece(reference) ;
		/*à compléter*/ (13) (1pt)
	}
	public void visualiser() throws VisualisationException {
		System.out.println("Pieces: \n");
			/*à compléter*/ (14)(1,5pt)
	}
	public void setPieces(List<Piece> pieces) {
		this.pieces = pieces;
	}
	public List<Piece> getPieces() {
		return pieces;
	}
}
```

```
public class Produit implements Visualisable {
	private String libelle;
	private Map<Integer, Assemblage> assemblages;
	public Produit(String libelle) {
		/*à completer*/ (15) (1pt)
	}
	public void ajouterPiece(Integer code, Piece piece) {
		if(!assemblages.containsKey(code)){
			/*à compléter*/ (16) (1pt)
		}assemblages.get(code).ajouterPiece(piece);}
```

```
	public void supprimerPiece(Integer code,String referencePiece)
				throws SupressionException {
		Assemblage assemblage = assemblages.get(code);
		/*à compléter*/ (17) (2pt)}
	public void visualiser() throws VisualisationException {
		for (Map.Entry<Integer, Assemblage> e: assemblages.entrySet()) {
			/*à compléter*/ (18) (1pt)
		}
	}
	public void setLibelle(String libelle) {
		this.libelle = libelle;
	}
	public String getLibelle() {
		return libelle;
	}
	public void setAssemblages(Map<Integer, Assemblage> assemblages) {
		this.assemblages = assemblages;
	}
	public Map<Integer, Assemblage> getAssemblages() {
		return assemblages;
	}
}
```

```
public class SupressionException extends Exception {
	public SupressionException(String message){
		super(message);
	}
}
```

```
public class VisualisationException extends Exception{
	public VisualisationException(String message){
		super(message);
	}
}
```

```
public class Main {
	public static void main(String[] args) {
		Produit prod=new Produit("Voit60FR");
		Piece p1=new Piece("ADF345","Fer",5);
		Piece p2=new PieceMecanique("ADF346","Cuivre",3,true);
		Piece p3=new PieceElectrique("ADF347","Aluminium",6,true);
		Piece p4=new Piece("ADF348","Fer",10);
		/*à compléter*/ (19)}} (2pt)
```