université de BORDEAUX Collège Sciences & Technologies	**ANNÉE UNIVERSITAIRE 2015/2016** **DS Programmation 2** **Code UE :** J1IN4W01 **Date :** 11 avril 2016 **Heure :** 11h00 **Durée :** 1h20 Documents non autorisés **3** pages

Exercice 1 : Exceptions

Précisez ce que sont les exceptions contrôlées et les exceptions non contrôlées en java en décrivant ce qui les différencie et en décrivant la façon d'utiliser une exception contrôlée. Utilisez le code de la classe `VectorInteger` de l'exercice 3 pour illustrer votre propos.

Exercice 2 : Algorithme de sélection

On considère le code suivant :

```
class A{}
class B extends A {}
class C extends B {}

class X{
    public void m(C c) {System.out.println("X param C");}
    public void m(A a) {System.out.println("X param A");}
}
class Y extends X {
    public void m(A a) {System.out.println("Y param A");}
    public void m(B b) {System.out.println("Y param B");}
}
public class TestAlgoSelection {
    public static void main(String[] argv) {
        C c = new C();
        B b = c;
        A a = c;
        Y y = new Y();
        X x = y;
        x.m(a);
        y.m(a);
        x.m(b);
        y.m(b);
        x.m(c);
        y.m(c);
    }
}
```

1) Donnez le type déclaré et le type réel des variables déclarées dans la méthode `main`.
2) Pour chacun des six appels de méthodes, précisez la signature des méthodes viables et la signature de la méthode sélectionnée à la compilation et précisez l'affichage obtenu à l'exécution.

Exercice 3 : Table d'association (clé, valeur)

L'objectif de cette partie est de modéliser une structure de données permettant d'associer à une clé une valeur. Ce type de structure de données est appelée une table d'association. Nous travaillerons uniquement avec des clés entières et nous appelerons notre type `Vector`. Les services qu'une table d'association doit offrir sont les suivants :

— `set` : Affecte la valeur pour la clé `k`.
— `get` : retourne la valeur pour la clé `k`;
— `toString` : retourne sous forme d'une chaine de caractère la liste des couples (clé, valeur) de la table.
— `clone` : duplique la table (pas les valeurs)

Pour garantir l'homogénéité de la structure de données la méthode `set` peut lever une exception si l'objet passé en paramètre n'est pas du bon type. Le programme suivant donne une implémentation de l'interface `Vector` pour stocker des valeurs de type `Integer`.

```
public class VectorInteger implements Vector{
    private Object _data[];

    public VectorInteger(){
        _data= new Object[5];
    }

    public VectorInteger(VectorInteger v){
        _data= new Object[v._data.length];
        for(int i=0; i<_data.length; ++i)
            _data[i]= v._data[i];
    }

    public void set(int index, Object o) throws Exception {
        checkType(o);
        resize(index);
        _data[index]= o;
    }

    public Object get(int index){
        return _data[index];
    }

    public VectorInteger clone(){
        return new VectorInteger(this);
    }

    public String toString(){
        StringBuffer s= new StringBuffer();
        for(int i=0; i<_data.length; ++i){
            if(_data[i] != null)
                s.append("(").append(i).append(",").append(_data[i]).append(")");
        }
        return s.toString();
    }

    protected void checkType(Object o) throws Exception{
        if(!(o instanceof Integer))
            throw new Exception("Invalid type for that array");
    }

    private void resize(int size){
        if(size < _data.length)
            return;
        if(size < _data.length * 2)
            size= _data.length * 2;
```

```
        Object[] tmp= new Object[size+1];
        for(int i=0; i<_data.length && i<size; ++i)
            tmp[i]=_data[i];
        _data= tmp;
    }
}
```

1) En Java, écrivez entièrement l'interface `Vector` déclarant l'ensemble des services disponibles sur nos tables d'associations.
2) Nous voulons maintenant écrire une implémentation de `Vector` dont la valeur stockée est de type `Float`. Proposez une implémentation de la classe `VectorFloat`. Cette implémentation devra impérativement hériter de la classe `VectorInteger`.
3) Dans le programme suivant, expliquez pourquoi on peut ajouter un Float dans un `VectorInteger` sans lever une exception à la ligne indiquée par un commentaire.

```
public class TestVector{
    public static void main(String[] argv){
        VectorFloat vFlt= new VectorFloat();
        try{
            vFlt.set(3, new Float(4));
            vFlt.set(30, new Float(6));
            vFlt.set(7, new Float(8));
            System.out.println(vFlt);
            VectorInteger vInt= vFlt.clone();
            vInt.set(70, new Float(12.5)); // question 3
            System.out.println(vInt);
        }catch(Exception e){
            System.err.println(e);
        }
    }
}
```

4) Pour résoudre l'incohérence présentée dans la question 3 il faut enlever l'héritage entre la classe `VectorFloat` et la classe `VectorInteger`. Donnez l'implémentation d'une solution composée d'une interface `Vector`, d'une classe abstraite `VectorAbstract` et de deux classes `VectorFloat`, `VectorInteger`.
5) Ajoutez dans votre architecture la classe `VectorString` qui étend `VectorAbstract` et qui permet de manipuler des `String`.