1 Serialisation Simple

Question 1 Créé une classe SerializerBuffer qui contient un ByteBuffer et des méthodes pour lire et écrire des entiers et des flottants.
void writeInt(int), void writeFloat(float), int readInt(), float readFloat()

```
public class SerializerBuffer {
  ByteBuffer buff;

  public SerializerBuffer(int i){
    buff = ByteBuffer.allocateDirect(i);
  }

  public void writeInt(int i){
    buff.putInt(i);
  }
  public void writeFloat(float f){
    buff.putFloat(f);
  }

  public int readInt(){
    return buff.getInt();
  }

  public float readFloat(){
    return buff.getFloat();
  }
}
```

Question 2 Ajouté à la classe précédente des méthodes pour lire et écrire des chaînes de caractère encodé en UTF-8.
void writeString(String), String readString()

```
  public void writeString(String s){
    Charset c= Charset.forName("UTF-8");
    ByteBuffer ce = c.encode(s);
    buff.putInt(ce.remaining());
    buff.put(ce);
  }

  public String readString(){
    int size = buff.getInt();
    int limit = buff.limit();
    Charset c = Charset.forName("UTF-8");
    buff.limit(buff.position()+size);
    String s = c.decode(buff).toString();
    buff.limit(limit);
    return s;
  }
```

Soit l'interface suivante :

```
public interface MySerialisable {
  public void writeToBuff(SerializerBuffer ms);
  public void readFromBuff(SerializerBuffer ms);
}
```

Question 3 Écrire une classe de test (Test1) qui contient un entier et une chaîne de caractère et qui implémente l'interface MySerialisable. La classe doit surcharger toString() pour renvoyer un message informatif.

```
public class Test1 implements MySerialisable {
  String s;
  int i;

  public Test1(String s,Test2 t, int i){
    this.s=s;
    this.i=i;
  }

  public String toString(){
    return "Test1("+s+","+t+","+i+")";
  }
  @Override
  public void writeToBuff(SerializerBuffer ms) {
    ms.writeInt(i);
    ms.writeString(s);
   }
  @Override
  public void readFromBuff(SerializerBuffer ms) {
    i=ms.readInt();
    s=ms.readString();
   }
}
```

Question 4 Ajouté un constructeur sans argument à la classe de test. Écrire une méthode main qui fabrique un objet de la classe de test le sérialise et le desérialise. *La déserialisation commence par appeler le constructeur vide puis appel la méthode readFromBuffer().*

```
public class Test1 implements MySerialisable {
  ...
  public Test1(){};

  public static void main(String[] args) {
    Test1 t = new Test1("foo",32);
    System.out.println("Pre :"+t);
    SerializerBuffer sb = new SerializerBuffer(1024);
    t.writeToBuff(sb);
    sb.buff.flip();
```

```
    Test1 t2 = new Test1();
    t2.readFromBuff(sb);
    System.out.println("Post:"+t2);
  }
}
```

Question 5 Écrire une seconde classe de test (Test2) qui contient deux flottants et qui implémente l'interface MySerialisable. Ajouter à la classe Test1 un champ de type Test2. Mettre à jour les fonctions de sérialisation.

```
public class Test2 implements MySerialisable {
  float f1,f2;

  public Test2(){};
  public Test2(float f1, float f2){
    this.f1=f1;
    this.f2=f2;
  }

  public String toString(){
    return "Test2("+f1+","+f2+")";
  }
  @Override
  public void writeToBuff(SerializerBuffer ms) {
    ms.writeFloat(f1);
    ms.writeFloat(f2);

  }
  @Override
  public void readFromBuff(SerializerBuffer ms) {
    f1=ms.readFloat();
    f2=ms.readFloat();
  }
}
```

```
public class Test1 implements MySerialisable {
  String s;
  Test2 t;
  int i;

  public Test1(){};
  public Test1(String s,Test2 t, int i){
    this.s=s;
    this.i=i;
    this.t=t;
  }

  public String toString(){
    return "Test1("+s+","+t+","+i+")";
  }
```

```
  @Override
  public void writeToBuff(SerializerBuffer ms) {
    ms.writeInt(i);
    ms.writeString(s);
    t.writeToBuff(ms);
  }
  @Override
  public void readFromBuff(SerializerBuffer ms) {
    i=ms.readInt();
    s=ms.readString();
    t= new Test2();
    t.readFromBuff(ms);
  }
}
```

Question 6 Que ce passe-t-il,quand le champ Test2 de la classe Test1 == **null**.

```
Pre :Test1(foo,null,32)
Exception in thread "main" java.lang.NullPointerException
  at TD.Test1.writeToBuff(Test1.java:26)
  at TD.Test1.main(Test1.java:41)
```

2 Gestion des objets null

Soit l'interface

```
public interface Creator<T extends MySerialisable>{
  public abstract T init();
}
```

Question 7 Ajouté un champ **public static final** Creator CREATOR aux classes Test1 et Test2 qui appel le constructeur vide de la classe.

Question 8 Ajouté à SerializerBuffer deux méthodes :

```
  public void writeMySerialisable( MySerialisable );
  public <T extends MySerialisable> T readMySerialisable( Creator<T> );
```

Modifier les classes Test1 et Test2 Afin qu'elle utilise ces méthodes pour la sérialisation.

Question 9 Modifier les deux méthodes la question précédente pour gérer le cas des objets **null**.

3 Gestion des structures récursives circulaires

Question 10 Ajouté à la classe SerializerBuffer les champs suivant :

```
  private Map<Integer,Integer> objMap = new HashMap<>();
  private ArrayList<MySerialisable> objArray = new ArrayList<>();
```

Question 11 Modifier la méthode writeMySerialisable pour qu'elle enregistre tous les objets qu'elle sérialise dans le tableau objArray. On utilisera de plus System.identityHashCode(Object) pour générer un identifiant unique de chacun des objets que l'on ajoutera à la table objMap avec la position de l'objet dans le tableau.

Question 12 Modifier les méthodes writeMySerialisable() et readMySerialisable() pour gérer les structures récursives circulaires.