IUT Paris Descartes – Base de la Programmation Objet
DST 2017 – 2018

2 heures – tous documents autorisés – *barème indicatif*

Les deux parties sont indépendantes. Lisez toutes les questions avant de commencer.

Partie 1 – Carafes

Le problème des carafes consiste à obtenir une carafe contenant 5 litres d'eau à partir de deux carafes dont l'une a une capacité de 7 litres et l'autre une capacité de 4 litres. Au départ les carafes sont vides. Les seules opérations autorisées sont : vider une carafe, remplir complètement une carafe et transvaser l'eau d'une carafe dans une autre. Il peut arriver lors d'un transvasement que la carafe cible ait une capacité insuffisante pour contenir l'eau des deux carafes. Dans ce cas, la carafe cible est remplie complètement et l'autre carafe contient le reste de l'eau.

Vous devez réaliser un programme permettant de manipuler les deux carafes du problème en respectant ces règles, jusqu'à ce qu'une des deux carafes contienne 5 litres. Le choix d'une opération sera codé par ri pour remplir la carafe i, vi pour vider la carafe i et tij pour transvaser de la carafe i dans la carafe j. Voici un exemple de session :

```
0/7 0/4 [but : 5]
choix : r2
0/7 4/4 [but : 5]
choix : t21
4/7 0/4 [but : 5]
choix : r2
```

```
4/7 4/4 [but : 5]
choix : t21
7/7 1/4 [but : 5]
choix : v1
0/7 1/4 [but : 5]
choix : t21
```

```
1/7 0/4 [but : 5]
choix : r2
1/7 4/4 [but : 5]
choix : t21
5/7 0/4 [but : 5]
Bravo !
```

Les prototypes des méthodes publiques de la classe `Carafe` sont donnés ci-dessous. Le rôle de cette classe est de représenter une carafe et d'assurer qu'elle sera manipulée sans tricher, c'est à dire qu'elle ne permet que les opérations autorisées par les règles du problème. Les noms des méthodes sont suffisamment explicites. Le constructeur permet de créer une carafe vide de telle ou telle capacité.

```
public class Carafe {
  public Carafe(int capacité) {...}
  public int contenu() {...}
  public void remplir() {...}
  public void vider() {...}
  public void transvaserDans(Carafe c) {...}
}
```

1. (2 points) Dans le programme encadré ci-dessous, remplacez les commentaires par le code Java approprié. Comme certains des commentaires sont répétés ou similaires, *vous écrirez seulement sur votre copie par quoi remplacer chacun des commentaire en gras et seulement ceux ci.* Par exemple, vous écrirez ceci :

```
/* instancier une carafe de 7 litres */ : xxxxxxxxxxx
/* remplir c1 */ : yyyyyyyyyy
/* contenu de c1 */ : zzzzzzzzzz
```

etc.

```
public class Appli {
  public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
    int but = 5;
    Carafe c1 = /* instancier une carafe de 7 litres */;
    Carafe c2 = /* instancier une carafe de 4 litres */;
    do {
      System.out.println(c1 + "␣" + c2 + "␣[but␣:␣" + but + "]");
      System.out.print("choix␣:␣");
      String choix = sc.next();
      if (choix.equals("r1"))
        // remplir c1
      else if (choix.equals("r2"))
        // remplir c2
      else if (choix.equals("v1"))
        // vider c1
      else if (choix.equals("v2"))
        // vider c2
      else if (choix.equals("t12"))
        // transvaser c1 dans c2
      else if (choix.equals("t21"))
        // transvaser c2 dans c1
      else
        System.out.println("choix␣incorrect");
    } while (but != /* contenu de c1 */ && but != /* contenu de c2 */);
    System.out.println(c1 + "␣" + c2 + "␣[but␣:␣" + but + "]");
    System.out.println("Bravo␣!");
  }
}
```

2. (4 points) Programmez la classe `Carafe`.

3. (1 point) Pour que les affichages réalisés par le programme ci-dessus soient corrects, il est nécessaire d'ajouter une méthode `toString` à la classe `Carafe`. Programmez cette méthode en vous référant à la trace d'exécution donnée précédemment.

Partie 2 – Loto

L'objectif est la conception d'un ensemble de classes permettant de mémoriser des tirages du Loto et de faire quelques statistiques simples.

Chaque tirage est réalisé à une date donnée. Pour simplifier, nous nous limitons au jour (un entier compris entre 1 et 31) et au mois (un entier compris entre 1 et 12). Un tirage est composé d'une suite de 7 entiers (un tableau) compris entre 1 et 49. Un tirage doit pouvoir être créé même si on en connaît que la date. Les numéros le composant doivent pouvoir être enregistrés ultérieurement.

Un ensemble de tirages est caractérisé par une liste de tirages. Vous devez nécessairement employer une liste de la bibliothèque standard pour sa représentation interne. Il doit être possible de créer un ensemble vide et d'ajouter des tirages ultérieurement. Il doit aussi être possible de connaître le numéro étant sorti le plus souvent (le plus petit en cas d'égalité) ainsi que la liste des dates auxquelles est sorti un numéro donné.

4. (8 points) Définissez les classes nécessaires à la manipulation d'un ensemble de tirages. Toutes les fonctionnalités précisées ci-dessus doivent être prises en compte. Vous êtes libres de faire des hypothèses pour les points non-précisés par le sujet. Toutefois, vous devez expliciter vos hypothèses. Toutes les préconditions doivent être vérifiées et une exception doit être levée en cas de non-respect (de type `IllegalArgumentException` si les paramètres sont incorrects et `IllegalStateException` si l'état de l'objet n'autorise pas l'invocation d'une méthode). Notez bien qu'il ne vous est pas demandé de produire un programme principal mais uniquement les classes nécessaires.

5. (2 points) Dessinez le diagramme de classes correspondant.

IUT Paris Descartes

Département STID ~~/~~ INFO

Nom : LAPOSTOLET

Prénom : Anselme

Groupe : 112

Date : 21/03/18

Matière : BPO

Note : 16.25

Appréciation :

PARTIE 1

1) /* instancier une carafe de 7 litres */

Carafe c1 = new Carafe(7);

// Remplir c1

c1.remplir();

// Vider c1

c1.vider();

// Transvaser c1 dans c2

c1.transvaserDans(c2);

* contenu de A *

A.contenu();

2)

```
public class canafe {
    private int capacite;
    private int contenu;

    public canafe(int cap) {
        capacite = cap;
    }
```

- accesseur
- constructeur

```
public int contenu () {
    return contenu;
}
```

```
public void remplir () {
    contenu = capacite;
}
```

```
public void vider () {
    contenu = 0;
```

```
public void transvaserDans(Garafe c) {
    if (this.contenu <= c.capacite) {
        c.contenu = this.contenu;
    }
    else {
        c.contenu = c.capacite;
        this.contenu = this.contenu - c.capacite;
    }
}
```

```
public String toString() {
    StringBuilder s = new StringBuilder();
    s.append(contenu);
    s.append("/");
    s.append(capacite);
    return s.toString;
}
```

PARTIE 2

```
private class Tirage {
    public static final int nbNum = 7;
    private int jour;
    private int mois;
    private int[nbNum] numéros;
}
```

```
public Tirage(int jour, int mois)
throws IllegalArgumentException {
    if (jour < 1 || jour > 31) {
        throw new IllegalArgument
        Exception("jour invalide");
    else if(mois < 1 || mois > 12) {
        throw new IllegalArgument
        Exception("mois invalide");
    else {
        this.jour = jour;
        this.mois = mois;
    }
}
```

et le tableau plus nécessaire : le mettre à null

```
public Tirage(int jour, int mois,
int[] t) throws IllegalArgumentException{
    try {
        this(jour, mois)
    }
    catch(IllegalArgumentException e){
        throw e;
    }
```

Scanned by CamScanner

```
    }
    if (t.length == nbNum){
        this.numeros = t;
    }
    else {
        throw new IllegalArgumentException
        ("numeros invalides");
    }
}

public void ajouterNum(int i, int num)
throws IllegalArgumentException {
    if (i < 0 || i > nbNum){
        throw new IllegalArgumentException
        ("indice invalide");
    }
    else if (num < 1 || num > 49){
        throw new IllegalArgument
        Exception ("numero invalide");
    }
    else{
        numeros[i] = num;
    }
}
```

```
public int[] getNums() throws IllegalStateException {
    if (numéros.size() != nbNuméros) {
        throw new IllegalStateException("Numéros non valides");
    }
    else {
        return numéros;
    }
}

public int getJour() {
    return jour;
}

public int getMois() {
    return mois
}
}

public class EnsembleTirages {
    ArrayList<Tirage> tirages;

    public EnsembleTirages {
        tirages = new ArrayList<Tirage>();
    }
```

```
public void ajouterTirage(Tirage t)
throws IllegalArgumentException {
    try {
        int[] test = t.getNums();
    }
    catch (IllegalStateException e) {
        throw new IllegalArgument
        Exception("Tirage invalide");
    }
    tirages.addLast(t);
}

public int nombreSouvent() throws
IllegalStateException {
    if (tirages.size() > 1) {
        throw new IllegalStateExcep
        ("Etat non valide");
    }
```

```
else {
    ArrayList<int> numTest =
    new ArrayList<int>(50);
    for(Image t: tImages) {
        try {
            for(int n: t.getNums()){
                numTest.set(n, numTest.
                get(n)+1);
            }
        }
        catch(IllegalStateException){
            throw new IllegalState
            Exception("un t.image est
            invalide");
        }
    }
    int MAX = -1;
    for(int n: numTest){
        if(n > MAX){
            MAX = n
        }
    }
}
```

10

```
        return MAX;
    }
}
```

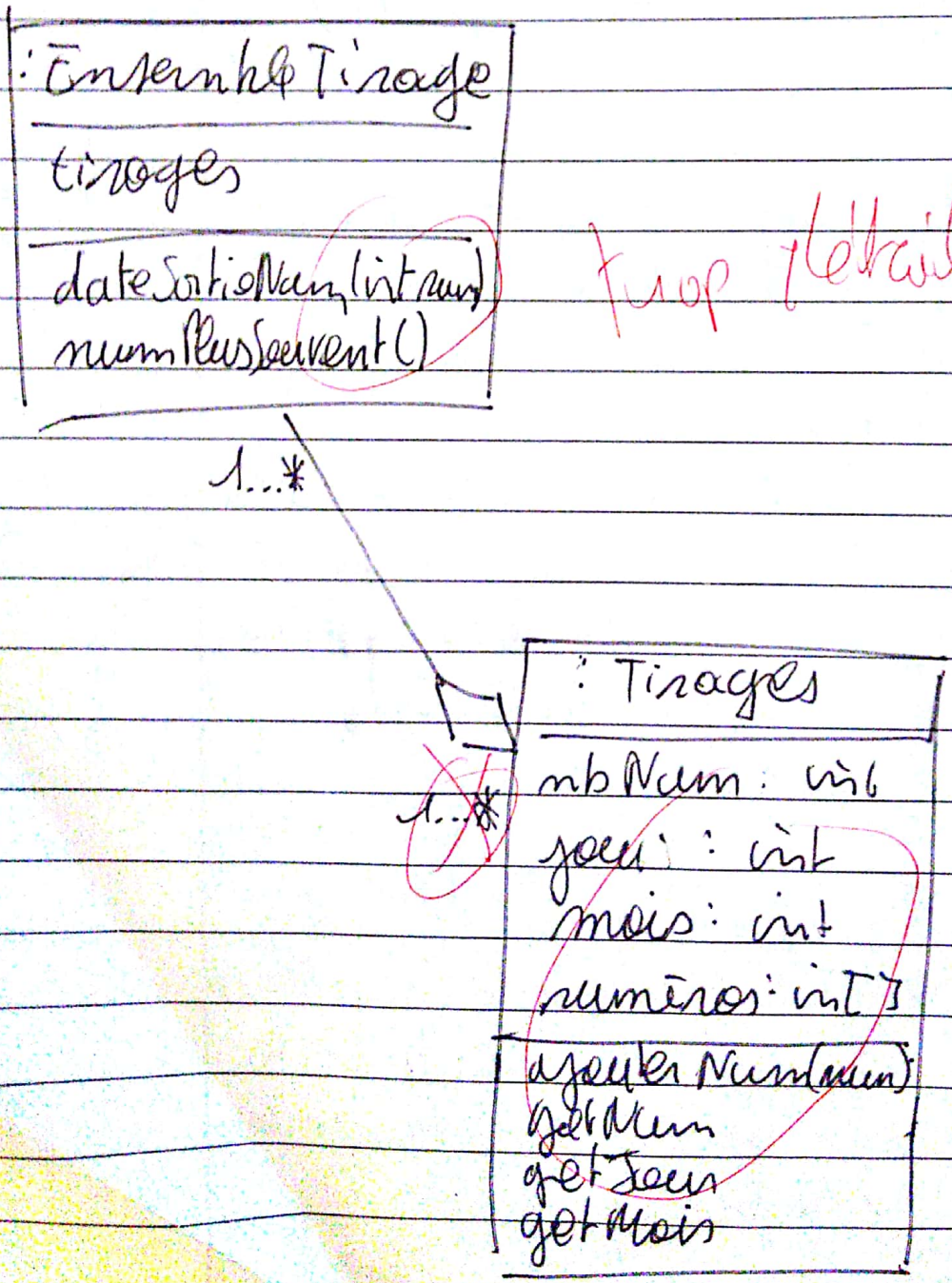
manque le test pour l'erreur !!

```
public HashMap<int,int> dateSortiePlusNum
() throws IllegalStateException {
    if (tirages.size() > 1) {
        throw new IllegalStateExce
        ption("Etat non valide");
    }
    else {
        HashMap<int,int> dates =
        new HashMap<int,int> ();
        for (Tirage t : tirages) {
            try {
                for (int n : t.getNums()) {
                    if (n == num) {
                        dates.add(t.getJour,
                        .t.getMois);
                    }
                }
            }
        }
        catch (IllegalStateException e) {
            throw new IllegalState
            Exception("un tirage est
            invalide")
        }
```

```
    }
    return dates;
  }
}
```

classes : 2,5
données : 2
algos : 2
exécutions : 1

2) Diagramme UML :



trop détaillé

1

12