Université Paris Est Créteil
IUT de Créteil-Vitry
DUT R&T 1ère année

Travaux Dirigés

Module M2207
Consolidation des bases de la programmation
Langage Java

Y. AMIRAT

TD 1 : Types simples, chaînes de caractères et tableaux, quelques classes utiles

Le but de ce TD est de vous familiariser avec la syntaxe de Java et d'étudier quelques classes utiles.

Exercice 1 : En utilisant l'éditeur Eclipse, écrire un programme Java qui affiche **Bonjour**.

Exercice 2 : Corriger les programmes suivants :

```
public class Scope {
	public static void main(String args[]){
		int i =0;
		for (int i=0 ;i<5 ;i++)  {System.out.print(i+",") ;}
		System.out.print("\n") ;
	}
}
```

```
public class Variables {
	public static void main(String args[]){
		float a=3.0 ;
		double b=4 ;
		float c ;
		c=Math.sqrt(a*a+b*b) ;
		System.out.print("c="+c) ;
	}
}
```

```
public class Promote {
	public static void main(String args[]){
		byte b=42 ;
		char c='a' ;
		short s=1024 ;
		int i=50000 ;
		float f=5.67f,
		double d=.1234 ;
		double resultat=(f*b)+(i/c)-(d*s) ;
		System.out.print((f*b)+ "+"+(i/c)+ "-"+(d*s)) ;
		System.out.println("="+resulat) ;

		byte b2=10 ;
		byte b3=b2*b;
		System.out.println("b3="+b3) ;
	}
}
```

Exercice 3: Soit le fichier source java ci-après :

```
public class Array {
        public static void main(String args[]){
                String jour_semaine[]=new String [7];
                int jour=Integer.parseInt(args[0]) ;
                jour_semaine[0]="dimanche" ;
                jour_semaine[1]= "lundi" ;
                jour_semaine[2]= "mardi" ;
                jour_semaine[3]= "mercredi" ;
                jour_semaine[4]= "jeudi" ;
                jour_semaine[5]= "vendredi" ;
                jour_semaine[6]= "ça me dit" ;
                System.out.println("Moi, le préfère le "+jour_semaine[jour]) ;
        }
}
```

Que se passera-t-il si on lance les commandes :
$java Array 2
$java Array 7
Expliquer.

Exercice 4 : On considère la méthode suivante :

```
1 public static String buildString(String s) {
2       String r="";
3       for(int i=0;i<s.length();i++) {
4               if(keepChar(s.charAt(i),i)) {
5                       r=r+s.charAt(i);
6               }
7       }
8       return r;
9 }
```

On remarque que la méthode utilise la méthode keepChar.

1. On suppose que la méthode keepChar est la suivante :

```
1 public static boolean keepChar(char c,int position) {
2       return true;
3 }
```

Que fait alors la méthode buildString (justifiez votre réponse) ?

2. On suppose que la méthode keepChar est la suivante :

```
1 public static boolean keepChar(char c,int position) {
2       if(position%2==1) {
3               return true;
4       } else {
5               return false;
6       }
7 }
```

Quel est le résultat de la méthode buildString si on lui transmet la chaîne "NBVCX" ?
Que fait la méthode buildString ?

3. Pour chaque résultat souhaité pour la méthode buildString proposez une méthode keep-Char qui permette son obtention :

(a) La chaîne renvoyée par buildString doit contenir un caractère sur trois de la chaîne paramètre, en conservant le caractère de position 1, puis celui de position 4, etc.
L'image de "ABCDEF" est donc "BE".

(b) La chaîne renvoyée par buildString doit contenir les caractères qui ne sont pas des chiffres contenus dans la chaîne paramètre. L'image de "AB2C3D6EF" est donc "ABCDEF".

Vous pourrez utiliser la classe Character qui propose une méthode isDigit[1] qui à un char associe true si et seulement si ce caractère correspond à un chiffre.

[1] Exemple: `boolean test=Character.isDigit('A');`

TD 2 : Notions de classe, objets, Héritage

Exercice 1:

1. Créer un fichier TestPoint.

2. Réaliser une classe ***Point2D*** permettant de manipuler un point d'un plan. On prévoira:
-un constructeur recevant en arguments les coordonnées (float) d'un point,
-une méthode d'instance ***deplace*** effectuant une translation définie par ses deux arguments (***float***),
-une méthode d'instance ***affiche*** se contentant d'afficher les coordonnées cartésiennes du point.
-une méthode d'instance ***String toString()*** permettant d'afficher sous forme textuelle les coordonnées cartésiennes d'un objet ***Point2D***.
Les coordonnées du point seront des attributs privés.

Ecrire par ailleurs, une classe de test ***TestPoint*** déclarant un point, l'affichant, le déplaçant et l'affichant à nouveau. Pour l'affichage, on testera 2 méthodes.

3. Adapter la classe précédente de manière à ce que la méthode de classe ***afficheNbPoints*** affiche le nombre d'objets de type ***Point2D*** créés.

Tester cette nouvelle classe dans la classe de test ***TestPoint***.

4. Réaliser une classe ***Point2Dbis***, analogue à la précédente, mais ne comportant pas de méthode ***affiche***. Pour respecter le principe d'encapsulation des données, prévoir deux méthodes d'instance publiques (nommées ***abscisse*** et ***ordonnée***) fournissant en retour respectivement l'abscisse et l'ordonnée d'un point.
Tester la classe dans la classe de test ***TestPoint***

5. Ajouter à la classe précédente de nouvelles méthodes d'instance:
-***rotation*** qui effectue une rotation dont l'angle est fourni en argument,
-***rho*** et ***theta*** qui fournissent en retour les coordonnées polaires du point.
Tester cette nouvelle classe dans la classe de test ***TestPoint***

Exercice 2 : Soit la classe ***Point2D*** de l'exercice 1 (question 2) .

1. Créer un fichier TestPoint3.

2. Expliquer la différence entre les spécificateurs d'accès **private, private protected** et **protected.**

3. Créer une classe ***Point2Dter***, dérivée de ***Point2D*** comportant une nouvelle méthode nommée **rho**, fournissant la valeur du rayon vecteur (première coordonnée polaire) d'un point.
Quel doit être le spécificateur d'accès pour les variables d'instance x et y de la classe ***Point2D*** pour qu'elles soient accessibles depuis la classe fille ***Point2Dter*** ?

4. Quelles sont les méthodes utilisables pour une instance de type ***Point2Dbter*** ? Ecrire une classe de test TestPoint3

Exercice 3 : Soit la classe ***Point2D*** de l'exercice 1 (question 2).

1. Créer une classe ***Point2DAvecCouleur***, dérivée de ***Point2D***, comprenant :
-un attribut **private** supplémentaire **cl**, de type String, destiné à contenir la « couleur » d'un point.
-les méthodes suivantes :
-**affiche**(redéfinie), qui affiche les coordonnées et la couleur d'un objet de type ***Point2DAvecCouleur***,
-**colorie**(String couleur), qui permet de définir la couleur d'un objet de type ***Point2DAvecCouleur***
-un constructeur permettant de définir la couleur et les coordonnées d'un point

2. Tester cette classe dans la classe de test TestPoint3

Exercice 4 : Soit la classe ***Point2Dbis*** de l'exercice 1 (question 4)

1. Créer un fichier TestPoint2D4 contenant la classe de test TestPoint2D4. Qu'affiche le code de la classe TestPoint2D4 ci-dessous :

```
Point2Dbis p1=new Point2Dbis (1,2);
Point2Dbis p2=p1;
Point2Dbis p3=new Point2Dbis (1,2);
System.out.println(p1==p2);
System.out.println(p1==p3);
```

2. Écrire dans la classe Point2Dbis une méthode ***estIdentiqueA()*** (à vous de trouver la signature exacte de la méthode) qui renvoie true si deux points Point2Dbis ont les mêmes coordonnées cartésiennes.

3. Remarquons qu'il existe déjà une méthode nommé ***equals()*** dans la classe Object dont le rôle est de tester si deux objets sont égaux sémantiquement.
Transformer la méthode ***estIdentiqueA ()*** en méthode ***equals()***.

4. Qu'affiche le code suivant :

```
Object p=new Point2Dbis (1,2);
Object p2=new Point2Dbis (1,2);
System.out.println(p.equals(p2));
```

Expliquer ce résultat. Comment obtenir un résultat plus logique ?

TD 3 : Interfaces-Exceptions

Exercice 1 : Expliquer le comportement des programmes.

1.

```
class Essai1Exception extends Exception{
        Essai1Exception(String s){
                super(s) ;
        }
}
class Essai2Exception extends Essai1Exception{
        Essai2Exception(String s){
                super(s) ;
        }
}
public class Exn{
        static void throwEssais(int i) throws Exception {
                switch (i){
                        case 1:  System.out.println("Lancement de Essai1Exception") ;
                                 throw new Essai1Exception ("Essai1Exception de throwEssais") ;
                        case 2:  System.out.println("Lancement de Essai2Exception") ;
                                 throw new Essai2Exception ("Essai2Exception de throwEssais") ;
                        default: System.out.println("Lancement de Exception") ;
                                 throw new Exception("Exception de throwEssais") ;
                        }
        }
        public static void main(String[] args){
                for (int i = 1 ; i <=3 ; i++){
                        try {
                                 throwEssais(i) ;
                                 System.out.println("Retour d'exception") ;
                        }
                        catch (Essai2Exception e){
                                 System.out.println("Catch Essai2: " + e.getMessage()) ;
                        }
                        catch (Essai1Exception e){
                                 System.out.println("Catch Essai1: " + e.getMessage()) ;
                        }
                        catch (Exception e){
                                 System.out.println("Catch Exception : " + e.getMessage()) ;
                        }
                        finally {
                                 System.out.println("Finally de main.") ;
                        }
                }
        }
}
```

2.

```
class Essai1Exception extends Exception{
        Essai1Exception(String s){
                super(s) ;
        }
}
class Essai2Exception extends Exception{
        Essai2Exception(String s){
                super(s) ;
        }
}
public class Exnbis{
        static void throwEssais(int i) throws Exception {
                switch (i){
                        case 1: System.out.println("Lancement de Essai1Exception") ;
                                throw new Essai1Exception ("Essai1Exception de throwEssais") ;
                        case 2: System.out.println("Lancement de Essai2Exception") ;
                                throw new Essai2Exception ("Essai2Exception de throwEssais") ;
                        default: System.out.println("Lancement de Exception") ;
                                throw new Exception("Exception de throwEssais") ;
                }
        }
        public static void main(String[] args){
                for (int i = 1 ; i <=3 ; i++){
                        try {
                                throwEssais(i) ;
                                System.out.println("Retour d'exception") ;
                        }
                        catch (Essai1Exception e){
                                System.out.println("Catch Essai1: " + e.getMessage()) ;
                        }
                        catch (Essai2Exception e){
                                System.out.println("Catch Essai2: " + e.getMessage()) ;
                        }
                        catch (Exception e){
                                System.out.println("Catch Exception : " + e.getMessage()) ;
                        }
                        finally {
                                System.out.println("Finally de main.") ;
                        }
                }
        }
}
```

TD 4 : Entrées-Sorties

Exercice 1 :

1. Analyser la classe suivante :

```
import java.net.*;
import java.io.*;

public class URL2Fichier {
/* Méthode principale */
public static void main(String[] args) {
   final int TAILLE_TAMPON = 4096;
   URL uneUrl ;
   InputStream fluxE=null ;
   FileOutputStream fluxS=null ;
   BufferedOutputStream donneesEcrites = null;
   BufferedInputStream donneesLues = null;
   int nbLus=0 ;
   byte [] tampon ;                    //tampon de lecture et d'écriture

   if (args.length != 2) {
         System.err.println(" Le programme requiert 2 arguments ");
         System.err.println("l'URL du fichier à copier");
         System.err.println("le nom du fichier destination");
        System.exit(1) ;
      }

      try {
         // Ouverture de l'URL pour une lecture
         uneUrl = new URL(args[0]);
         //Convertir l'URL en InputStream
        fluxE=uneUrl.openStream() ;
        //Construction du flux de type BufferedInputStream
        donneesLues = new BufferedInputStream(fluxE);
         // construction d'un FileOutputStream.
         fluxS = new FileOutputStream(args[1]);
         // construction d'un BufferedOutputStream
         donneesEcrites = new BufferedOutputStream(fluxS);
         System.out.println("Connextion établie") ;

         tampon=new byte[TAILLE_TAMPON] ;
         do{
             nbLus=donneesLues.read(tampon) ;
             if (nbLus !=-1) {
                 donneesEcrites.write(tampon,0,nbLus) ;}
            }while(nbLus !=-1) ;
          donneesEcrites.close() ;     //écriture de la fin du fichier
          System.out.println("Fin d'écriture") ;
      }
      catch (MalformedURLException excp) {
         System.out.println (args[0] + " : impossible de traiter cette URL");
         System.out.println (excp);
      }
      catch (IOException excp) {
         System.out.println ("Erreur : " + excp);
      }
   }
}
```

Tester cette classe en créant un fichier URL2Fichier.java.

Exercice 2 :

On veut créer une classe qui copie un fichier texte source vers un fichier texte destination ligne par ligne. Compléter le programme de cette application dont le squelette est donné ci-dessous :

```
public class CopieFichierTexte {
        private String source;
        private String destination;
        public CopieFichierTexte(...) {
                ...
        }
        public static void main(String[] args) {
                try {
                        ...
                }
                catch (...) {
                        System.out.println("erreur à l'ouverture des flux");
                }
                catch (...) {
                        System.out.println("erreur lors des lectures/écritures");
                }
        }
        public void copieLignes() throws ... {
                ...
        }
}
```

TD 5 : Applications client-serveur

Exercice 1 : Création d'un serveur TCP/IP mono-client

On veut créer un serveur de temps simple (par sockets TCP-IP) à partir de l'exemple de l'application serveur étudiée en cours. Modifier le programme de façon à ce que le serveur retourne l'heure et la date du système. On utilisera ici le port 6666 comme port d'écoute du serveur. Pour l'acquisition de la date et de l'heure, on utilisera une instance de Date :

```
Date d=new Date() ;    //d contient la date et l'heure au moment de son instantiation
String s=d.toString() ; //s contient la date et l'heure sous la forme d'un String
```

Utiliser un client Telnet pour tester cette application.

Exercice 2 : Création d'un client TCP/IP

Ecrire un programme client permettant de lire la date sur un serveur de date.

Exercice 3: Synthèse

1. On veut créer un client simple en java permettant de se connecter à un serveur pour lire un fichier texte qui a pour URL : http://www.u-pec.fr/index.html. Chaque ligne lue est ensuite affichée à l'écran (côté poste client).

2. Modifier l'application précédente pour que cette fois-ci chaque ligne lue par le client soit envoyée vers un serveur d'adresse 194.214.10. 245 écoutant sur le port TCP 80.