

EXERCICES DE JAVA

Constructeurs et « destructeurs »

Exercice CSTR-DSTR-JAVA

a) Donner la sortie du programme Java suivant et commenter très brièvement.

```
class Truc {
    public Truc() { System.out.println (« ++ normal »); }
    public void finalize() { System.out.println (« --normal »); }
    public static void main(String args[]) {
        Truc x;
    }
}
```

a') Même question avec :

```
public static void main(String args[]) {
    Truc x = new Truc();
}
```

a'') Même question avec :

```
public static void main(String args[]) {
    Truc x = new Truc();
    x.finalize()
}
```

Exercice CLE-1

Soit le programme Java suivant:

```
class Cle {
    public int a;
    public int b;
    public Cle (int x, int y) { a = x; b = y; System.out.println ("++ cle " + this + " " + a + " " + b); }
    public void detruire() { System.out.println ("-- cle " + this + " " + a + " " + b); }
}
class Coffre {
    public Cle cle;
    public Coffre(Cle c) { cle = c;
        System.out.println ("++ coffre CLE " + this + " " + cle.a + " " + cle.b); }
    public Coffre(int n) { cle = new Cle(100+n, 1000+n);
        System.out.println ("++ coffre ENTIER " + this + " " + cle.a + " " + cle.b); }
    public Coffre(Coffre c) { cle = new Cle(10+ c.cle.a, 10+ c.cle.b);
        System.out.println ("++ coffre COFFRE " + this + " " + cle.a + " " + cle.b); }
    public void detruire() { cle.detruire(); System.out.println ("-- coffre " + this); }
    public static void main (String argv[]) { Cle cle1 = new Cle(1, 2);
        Coffre c1 = new Coffre(cle1); Coffre c2 = new Coffre(3); Coffre c3 = new Coffre(c1);
        c3.detruire() ; c2.detruire() ; c1.detruire() ; cle1.detruire() ; }
}
```

Donner la sortie du programme.

Exercice CLE-2

Soit le programme Java suivant:

```
class Cle {
    public int a;
    public int b;
    public Cle (int x, int y) { a = x; b = y; System.out.println ("++ cle " + this + " " + a + " " + b); }
    public void detruire() { System.out.println ("-- cle " + this + " " + a + " " + b); }
}
class Coffre {
    public Cle cle;
    public Coffre(Cle c) { cle = new Cle(10+c.a, 10+c.b);
        System.out.println ("++ coffre CLE " + this + " " + cle.a + " " + cle.b); }
    public Coffre(int n) { cle = new Cle(100+n, 1000+n);
        System.out.println ("++ coffre ENTIER " + this + " " + cle.a + " " + cle.b); }
    public Coffre(Coffre c) { cle = c.cle;
        System.out.println ("++ coffre COFFRE " + this + " " + cle.a + " " + cle.b); }
    public void detruire() { cle.detruire(); System.out.println ("-- coffre " + this); }
    public static void main (String argv[]) { Cle cle1 = new Cle(4, 5);
        Coffre c1 = new Coffre(cle1); Coffre c2 = new Coffre(6); Coffre c3 = new Coffre(c1);
        c3.detruire() ; c2.detruire() ; c1.detruire() ; cle1.detruire() ; }
}
```

Donner la sortie du programme.

Les méthodes EQUALS, CLONE et TOSTRINGExercice EQUALS

Donner la sortie du programme Java suivant.

```
class Truc {
    public int i ;
    public Truc(int a) { i = a ; }
    public Truc(Truc t) { i = t.i ; }
    public boolean equals( Truc t ) { return (t.i==i) ; }
    public static void main(String args[]) {
        Truc y = new Truc(1);
        Truc z = y;
        Truc w = new Truc(y);
        if (z==y) System.out.println ( " 1 " ) ;
        if (w==y) System.out.println ( " 2 " ) ;
        if (z.equals(y)) System.out.println ( " 3 " ) ;
        if (w.equals(y)) System.out.println ( " 4 " ) ;
    }
}
```

Exercice CLONE

a) Soit le programme JAVA suivant :

```

class Machin {
    public int bidule ;
    public Machin(int b)
    {
        System.out.println ( « ++ Machin normal » ) ;
        bidule = b ;
    }
}

class Truc {
    public Machin machin ;
    public Truc(Machin m)
    {
        System.out.println ( « ++ Truc normal » ) ;
        machin = m ;
    }
    public static void main(String args[])
    {
        Machin m = new Machin(1) ;
        Truc x = new Truc(m) ;
        Truc y = (Truc) x.clone() ;
    }
}

```

Donner la sortie du programme en précisant le nombre d'objets créés et détruits et la nature (plantage ou pas) de la terminaison de l'exécution. A quoi sert le cast (Truc) ?

b') Redéfinir la méthode `Object clone()` dans la classe `Truc` et donner la sortie du programme en précisant le nombre d'objets créés et détruits et la nature (plantage ou pas) de la terminaison de l'exécution.

b'') Redéfinir la méthode `clone()` dans la classe `Machin`, modifier la méthode `clone()` de b') et donner la sortie du programme en précisant le nombre d'objets créés et détruits et la nature (plantage ou pas) de la terminaison de l'exécution.

b^{ter}) Définir la méthode `finalize()` dans les classes `Machin` et `Truc`, donner la sortie du programme Java suivant en précisant le nombre d'objets créés et détruits et la nature (plantage ou pas) de la terminaison de l'exécution.

```

public static void main(String args[])
{
    Machin m = new Machin(1) ;
    Truc x = new Truc(m) ;
    Truc y = x.clone() ;
    y.finalize() ;
    x.finalize() ;
}

```

Références, pointeurs et instances en C++ et en Java

Exercice REFERENCE-POINTEUR-JAVA-C++

a) Le programme Java suivant est-il correct ?

```
class Truc {
    public Truc() { System.out.println (« ++ normal »); }
    public Truc(Truc t) { System.out.println (« ++ copie »); }
    public static void main(String args[]) {
        Truc x;
        Truc y = x;
    }
}
```

b) Même question en supprimant la ligne `Truc y = x;` Quelle est la sortie du programme ?

c) Donner la sortie du programme C++ suivant et commenter très brièvement.

```
#include <iostream.h>
class Truc { public :
    Truc() { cout << « ++ normal » << endl ; }
    Truc(Truc & t) { cout << « ++ copie » << endl ; }
} ;
void main() {
    Truc x;
    Truc y = x;
}
```

d) Même question en rajoutant le caractère `&` devant le caractère `y`. Comment s'appelle ce type de déclaration C++ ?

c') Le programme C++ du c) appelle le constructeur normal pour créer `x` et le constructeur par recopie pour créer `y`. Modifier le programme Java du a) pour qu'il fasse de même.

Quelle méthode prédéfinie en Java peut-on utiliser à la place du constructeur par recopie ?

Le programme Java crée-t-il des objets dynamiques ou statiques ? Et le programme C++ du c) ? Est-ce possible de créer des objets Java locaux non dynamiques ? En C++ ?

d') On souhaite écrire un programme Java qui ressemble au programme C++ du d). On veut donc que ce programme Java crée un objet `x` puis fasse une référence `y` vers cet objet. Ecrire ce programme Java (qui affiche la même chose que le programme C++ du d)).

Une référence Java est-elle identique à une référence C++ ?

c'') On a vu à la question c') que le programme Java faisait la même chose que le programme C++ du c) (appel du constructeur normal pour construire `x` puis appel du constructeur par recopie pour construire `y`) mais en utilisant des objets dynamiques au lieu d'objets locaux statiques. Modifier le programme C++ du c) pour qu'il utilise des objets dynamiques, comme le programme Java du c').

A quel type de donnée C++, une référence Java correspond-elle ?

d'') Existe-t-il un type de donnée Java correspondant à une référence C++ ?

Le programme C++ du d) utilisait le mécanisme de référence spécifique à C++ n'existant pas en Java. Re-écrire le programme C++ du c) sans ce mécanisme spécifique en utilisant le type de données C++ correspondant aux références Java.

Que pensez-vous de la phrase « En Java il n'y a pas de pointeurs mais que des références » ?

Propriétés de classe ou propriété d'instance

Exercice STATIC

Soit le programme JAVA suivant :

```

class R {
    public static R notreR;
    public static int j;
    public int i;
    public R monR;
    public R(int a) { i = a; monR = null; j++; }
    public static void p(R r) {
        j = 0;
        notreR = new R(0);
        m(notreR);
    }
    public static void q(R r) {
        System.out.print("R.notreR = "); n(notreR);
        System.out.println("R.j = " + j);
    }
    public static void m(R r) {
        System.out.println("R, i = " + r.i);
    }
    public static void n(R r) {
        m(r);
        System.out.print("monR = ");
        if (r.monR!=null) r.monR.m(r.monR);
        else System.out.println("null");
    }
}

class S {
    public static void main() {
        R.p(null);
        R r1 = new R(1);
        R.notreR.monR = r1;
        R r2 = new R(2);
        System.out.print("r2 = ");
        R.n(r2);
        r1.monR = r2;
        System.out.print("r1 = ");
        R.n(r1);
        R.q(r1);
    }
}

```

a) Le mot-clé 'static' est correctement placé en ce qui concerne les attributs mais pas pour les méthodes. Reprogrammer ce programme pour que sa sortie toujours la même, que les mot-clés 'static' soient partout correctement placés, et enfin que les paramètres inutiles soient enlevés.

b) Donner le pour et le contre de la phrase : « le constructeur est une méthode d'instance ».

HéritageExercice TITI-TOTO

On a les classes suivantes:

```
// bubu.java
import java.io.*;

class Bubu {
    public int x;
    public Bubu(int a) {
        System.out.println( "++ Bubu debut");
        x = a;
        System.out.println( "++ Bubu x = " + x);
        System.out.println( "++ Bubu fin");
    }
    public void detruire(){
        System.out.println( "-- Bubu debut x = " + x);
        System.out.println( "-- Bubu fin");
    }
}

class Toto {
    public Bubu bu;
    public int y;
    public Toto(int a, int b) {
        System.out.println( "++ Toto debut");
        y = b;
        System.out.println( "++ Toto y = " + y);
        bu = new Bubu(a);
        System.out.println( "++ Toto fin");
    }
    public void detruire(){
        System.out.println( "-- Toto debut y = " + y);
        bu.detruire();
        System.out.println( "-- Toto fin");
    }
}

class Titi extends Toto {
    int z;
    Titi(int a, int b, int c) {
        super(a, a+b) ;
        System.out.println( "++ Titi debut");
        z = c;
        System.out.println( "++ Titi z = " + z);
        System.out.println( "++ Titi fin");
    }
    public void detruire() {
        System.out.println( "-- Titi debut z = " + z);
        System.out.println( "-- Titi fin");
        super.detruire() ;
    }
}
}
```

```

class Test {
    public static void main(String argv[]) throws IOException {
        System.out.println( "main debut construction");
        System.in.read();
        Toto to = new Toto(1, 2);
        System.out.println( "main milieu construction");
        System.in.read();
        Titi ti = new Titi(3, 4, 5);
        System.out.println( "main fin construction");
        System.out.println( "main debut destruction");
        System.in.read();
        ti.detruire();
        ti = null;
        System.out.println( "main milieu destruction");
        System.in.read();
        to.detruire();
        ti = null;
        System.out.println( "main fin destruction");
        System.in.read();
    }
}

```

Question a

Donner la sortie du programme.

Question b

On ajoute la méthode `imprimer()` dans la classe `Toto` et on modifie le `main()` :

```

class Toto {
    // ici, la meme chose qu'avant
    public void imprimer(){
        System.out.println( "*** bu.x = " + bu.x + " y = " + y);
    }
}
class Test {
    public static void main (String argv[]) {
        // ici, les memes constructions qu'avant
        System.out.println( "to = ");
        to.imprimer();           // instruction A
        System.out.println( "ti = ");
        ti.imprimer();         // instruction B
        // ici, les memes destructions qu'avant
    }
}

```

Donner la sortie des instructions A et B.

Question c

On ajoute la méthode `imprimer()` dans la classe `Titi` :

```

class Titi {
    // ici, la meme chose qu'avant
    public void imprimer(){
        System.out.println( "*** bu.x = " + bu.x + " y = " + y + " z = " + z); }
}

```

Donner la sortie des instructions A et B de la question précédente.

Exercice SUPER-THIS

Donner la sortie du programme Java et expliquer :

```

class Less {
    public int n ;
    public Less(int x) { n = x ; }
    public String toString() { return "Less ! " ; }
    public void test1() {
        System.out.println("Less.test1 : " + this.toString());
    }
    public void test2() {
        System.out.println("Less.test2 : " + this.n);
    }
}

class More extends Less{
    public int n ;
    public More(int x, int y) { super(x) ; this.n = y ; }
    public String toString() { return "More ! " ; }
    public void test1() {
        super.test1() ;
        System.out.println(
            "More.test1 : " + this.toString() + super.toString());
    }
    public void test2() {
        super.test2() ;
        System.out.println( "More.test2 : " + this.n + super.n);
    }
    public static void main(String argv[]) {
        More m = new More(1, 2) ;
        m.test1() ;
        m.test2() ;
        Less n = new Less(0) ;
        n.test1() ;
        n.test2() ;
    }
}

```

Exercice ABCDE

Donner la sortie du programme suivant :

```
// abcde.java
import java.io.*;
class A {
public A monA ;
    public void maMethode() { imprimer(" A ") ; }
    public void saMethode() { monA.maMethode(); }
    public void imprimer(String s) { System.out.println(s) ; }
}
class B extends A{
    public C monC ;
    public void saMethode() { monC.maMethode() ; }
}
class C extends A {
    public B monB ;
    public void maMethode() { imprimer(" C ") ; }
    public void saMethode() { monB.maMethode() ; }
}
class D extends B{
    public E monE;
    public void maMethode() { imprimer(" D ") ; }
    public void saMethode() { monE.maMethode() ; }
}
class E extends B {
    public D monD ;
    public void maMethode() { imprimer(" E ") ; }
}
class M {
    public static void main(String argv[]) {
        A a = new A();
        B b = new B();
        C c = new C();
        D d = new D();
        E e = new E();
        a.monA = b;
        b.monA = c;
        c.monA = d;
        d.monA = e;
        e.monA = a;
        b.monC = c;
        c.monB = b;
        d.monC = c;
        e.monC = c;
        d.monE = e;
        e.monD = d;

        a.maMethode();
        b.maMethode();
        c.maMethode();
        d.maMethode();
        e.maMethode();

        a.saMethode();
        b.saMethode();
        c.saMethode();
        d.saMethode();
        e.saMethode();
    }
}
}
```

Exercice CSTR-HERTIAGE-JAVA-C++

a) Donner la sortie du programme C++ suivant.

b) Traduire les constructeurs en Java (On supprimera les `System.out.println`).

```
// mistral.cpp

#include <iostream.h>
#include <string.h>

class Objet { public:
    char nom[64];
    Objet(char *);
};
class Loge;
class EtreVivant : public Objet { public:
    Loge * maPremiereLoge;
    EtreVivant(char *);
};
class EtreHumain : public EtreVivant { public:
    Loge * maSecondeLoge;
    EtreHumain(char *);
};
class Loge : public Objet { public:
    EtreVivant * monEtreVivant;
    Loge(char *, EtreVivant *);
};
Objet::Objet(char * n) {
    cout << "++ objet" << endl;
    strcpy(nom, n);
}
EtreVivant::EtreVivant(char * n) : Objet(n) {
    cout << "++ EV debut" << endl;
    maPremiereLoge = new Loge("", this);
    cout << "++ EV fin" << endl;
}
EtreHumain::EtreHumain(char * n) : EtreVivant(n) {
    cout << "++ EH debut" << endl;
    maSecondeLoge = new Loge("", this);
    cout << "++ EH fin" << endl;
}
Loge::Loge(char * n, EtreVivant * ev) : Objet(n) {
    cout << "++ Loge debut" << endl;
    monEtreVivant = ev;
    cout << "++ Loge fin" << endl;
}
void main() {
    cout << "Naissance de Mistral" << endl;
    EtreHumain * mistral = new EtreHumain("mistral");
    cout << "Mort de Mistral" << endl;
}
```

ThreadsExercice JOUEUR-BALLON

- Quels sont les deux méthodes pour utiliser des threads Java ?
- Dans quel cas utiliser une extension de la classe Thread ? l'interface Runnable ?
- Que fait signifie le mot-clé synchronized ?
- Le programme Java suivant lance 2 threads. Donner la sortie du programme.

```

class Ballon extends Object {
    public String nom;
    public int x;
    public Ballon(String n, int a) { nom = n; x = a; }
    public String toString() { return("<" + nom + "> x = " + x ); }
    public void deplacer(int o) {x = x + o;}
    public synchronized void prendre_donner(Joueur j) {
        /* System.out.println(j + " DEMANDE le ballon " + this);
        this.wait(); */
        System.out.println(j + " POSSEDE le ballon");
        deplacer(j.offset);
        j.sleep(j.periode*1000);
        /* this.notify(); */
        System.out.println(j + " LACHE le ballon " + this);
    }
    /* public synchronized void donner() {
        System.out.println("ballon " + this + " donne.");
        this.notify();
    } */

    public static void main(String args[]) {
        Ballon a = new Ballon("adidas", 0);
        Joueur gauche = new Joueur(a, -7, 7, "zidane");
        Joueur droite = new Joueur(a, 5, 5, "ronaldo");
        gauche.start();
        droite.start();
        /* Thread.sleep(5*1000);
        a.donner(); */
    }
}

class Joueur extends Thread {
    public Ballon monBallon;
    public int offset;
    public int periode;
    public String nom;
    public Joueur(Ballon b, int o, int p, String n) {
        monBallon = b; offset = o; periode = p; nom = n;
    }
    public String toString() {
        return("{ " + nom + " } " + offset + " " + periode);
    }
    public void run() {
        for(;;) monBallon.prendre_donner(this);
    }
}

```

- A quoi servent les mots-clés wait() et notify() en Java ?
- On enlève les commentaires du programme précédent. Donner la sortie du programme. Expliquer ce qui se passe.

(les blocs try catch autour des sleep et wait ont été enlevés pour la lisibilité du programme).

Exercice SEMAPHORE

a) Donner la sortie du programme suivant.

```

class Semaphore extends Object {

    public String nom;
    public int places;
    public int libres;

    public Semaphore(String s, int p, int l) {
        nom = s;
        places = p;
        libres = l;
    }

    public String toString() {
        return("<" + nom + "> " + libres + "/" + places );
    }

    public synchronized void reserver(Processus p) {
        if (libres>0) {
            p.etat = 2;
            libres--;
        }
        else {
            p.etat = 1;
            wait();
            p.etat = 2;
            libres--;
        }
    }

    public synchronized void liberer(Processus p) {
        if (libres < places) {
            libres++;
            notify();
            p.etat = 0;
        }
    }

    public static void main(String args[]) {
        Semaphore s = new Semaphore("semaphore", 2, 2);

        Processus p1 = new Processus(s, "a", 10);
        Processus p2 = new Processus(s, "b", 15);
        Processus p3 = new Processus(s, "c", 25);
        Processus p4 = new Processus(s, "d", 35);
        Processus p5 = new Processus(s, "e", 45);
        Processus p6 = new Processus(s, "f", 55);
        Processus p7 = new Processus(s, "g", 50);

        p1.start(); Thread.sleep(1000);
        p2.start(); Thread.sleep(1000);
        p3.start(); Thread.sleep(1000);
        p4.start(); Thread.sleep(10000);
        p5.start(); Thread.sleep(10000);
        p6.start(); Thread.sleep(10000);
        p7.start();

    }
}

```

```

class Processus extends Thread {
    public static Liste mesProcessus = new Liste();

    public Semaphore monSemaphore;
    public String nom;
    public int periode;
    public int nombre;
    public int etat; // 0 = libre, 1 = attente , 2 = propri

    public Processus(Semaphore s, String n, int p) {
        monSemaphore = s;
        nom = n;
        periode = p;
        etat = 0;
        nombre = 0;
        mesProcessus.ajouter(this);
    }

    public String toString() {
        String s = new String(nom + nombre);
        switch (etat) {
            case 0: s = s + "  "; break;
            case 1: s = s + " - "; break;
            case 2: s = s + " + "; break;
        }
        return s;
    }

    public void run() {

        System.out.println(this + " démarre.");
        System.out.println(mesProcessus);

        for(;;) {
            monSemaphore.reserver(this);
            System.out.println(mesProcessus);
            sleep(periode*1000);
            monSemaphore.liberer(this);
            nombre++;
            System.out.println(mesProcessus);
            sleep(nombre*periode*1000);
        }
    }
}

```

(les blocs try catch autour des sleep et wait ont été enlevés pour la lisibilité du programme).

ExceptionsExercice EXCEPTION

- a) En supposant que les valeurs de retour des méthodes qui sont négatives (respectivement positives ou nulles) correspondent à un traitement exceptionnel (respectivement normal), ré-écrire le programme Java suivant avec des exceptions.

On définira les exceptions `ExceptionOhJeMeurs`, `ExceptionChatMort`, `ExceptionGrangeEnFeu`, `ExceptionChateauDetruit`.

```
class Marquise {
    public static void main(String argv[]) {
        int a = laBelleVie () ;
        if (a==0) System.out.println(« oh ! la vie est délicieuse ») ;
        if (a== -1) System.out.println(« oh ! je meurs ») ;
    }

    public static int laBelleVie(String argv[]) {
        int i ;

        for (i=0 ; i<100; i++) {
            int a = QuelleNouvelle () ;
            if (a==0) {
                System.out.println(«tout va bien, marquise») ;
                i = i+25 ;
            }
            if (a== -1) {
                System.out.println(« le petit chat est mort ») ;
                i = i+5 ;
            }
            if (a== -2) System.out.println(« la grange a pris feu ») ;
            if (a== -3) {
                System.out.println(« le château est détruit ») ;
                return -1 ;
            }
        }
        return 0 ;
    }

    public static int QuelleNouvelle () {
        System.out.println(«Marquise, tapez un nombre entre 0 et 3») ;
        int i = KeyBoard.getInt() ;
        return -i ;
    }
}
```