# AP4B - Examen final du 25 Juin 2021

## Durée 1h45 heures – aucun document autorisé

**Exercice 1.** **Questions.**

1. A quoi correspond le concept d’interface en Java, comment est-il exprimé et utilisé dans le langage ?
2. Dites ce qui est affiché à l'écran lors de l'exécution du programme suivant et pourquoi :

public class Mystere{

 int a, b;

 public Mystere(int i, int j){

 a = i; b = j;

 }

 public static void echange(Mystere x, Mystere y){

 Mystere aux;

 aux = x; x = y; y = aux;

 }

 public void affiche(){

 System.out.println("a="+a+"b="+b);

 }

 public static void main(String argv[]){

 Mystere u,v;

 u = new Mystere(10,30);

 v = new Mystere(40,50);

 Mystere.echange(u,v);

 u.affiche();

 v.affiche();

 }

}

1. Rappeler le modèle de classe du pattern composite. Où retrouve-t-on son utilisation en Java ?

**Exercice 2.** **Diagramme Etat/Transition.**

Donner un diagramme Etat/Transition modélisant la conduite d’une automobile : avec par exemple la phase de démarrage moteur, la conduite, l’accélération, le freinage, l’arrêt et incluant des actions utiles à préciser comme tourner le volant, mettre le clignotant, appuyer sur frein, sur accélérateur, etc., qu’on traitera comme des événements. Indiquer les états, les événements et donner un diagramme Etat/Transition.

**Exercice 3. Chronomètre graphique.** Lorsque le chronomètre démarre, un cadran augmente (le chronomètre tourne) jusqu'à remplir un disque ; le chronomètre s'arrête quand le tour est fini. La durée du "tour" est un paramètre du chronomètre. On peut arrêter le chronomètre; si on le fait redémarrer, il repart à 0. On peut suspendre le chronomètre pendant qu'il tourne ; si on le fait reprendre, il repart du point où il était lorsqu'on l'a arrêté. Lorsque le chronomètre a fini de tourner, on peut le faire redémarrer. Nous vous demandons de compléter le code de la classe Chrono pour laquelle nous vous donnons un programme incomplet dans les pages suivantes. La classe Chrono peut au choix étendre la classe Thread ou bien implémenter l'interface Runnable ; c'est ce second choix que vous implanterez. On utilisera la méthode statique currentTimeMillis() de la classe System qui retourne le nombre de millisecondes écoulées depuis le début de l'année 1970, exprimé avec un long.

L'objectif ici est d'illustrer comment démarrer une thread, ou bien la suspendre, ou bien la faire reprendre où elle en était avant d'avoir été suspendue, ou bien l'arrêter définitivement. Il faut donc utiliser des variables partagées avec accès synchronisé. Vous pourrez remarquer l'utilisation de la méthode isAlive() qui permet de savoir si une thread est, ou non, en cours d'exécution. La méthode retourne la valeur true si l'exécution de la méthode run est commencée et non terminée ; false sinon.

**Exercice 4.** **Etude de cas.**

On s'intéresse à une société qui fabrique des composants de moteurs. Sur le plan des moyens de production, la société dispose de 6 usines dont une dédiée au montage. La fabrication comporte une phase d'usinage, une phase de traitement thermique et de surface, puis de montage des sous-ensembles qui s'intégreront dans les moteurs. La direction commerciale reçoit chaque jour les commandes clients. Compte-tenu des délais de fabrication (cycles de 8 à 15 mois), les clients pressés peuvent passer des commandes prioritaires mais au prix d'une sur-facturation de 20 % de la commande. Les usines les traitent alors en priorité. Les gros clients (plus de 10 commandes par an) sont traités en priorité aussi. Après un contrôle technique des articles commandés, la direction commerciale introduit la commande par ordinateur et obtient en sortie l'éclatement des ensembles composés en composants élémentaires qui appartiennent à deux catégories distinctes :

pièces fabriquées par l'entreprise,

pièces sous-traitées.

Elle imprime alors un listing de composants "maison" à destination des usines et un listing des composants sous-traités à destination du service de gestion du stock de l'entreprise pour approvisionnement. Ce service gère aussi l'approvisionnement bimensuel en matières premières. La commande globale est, elle, transmise à l'usine de montage qui ne peut finaliser la commande qu'après réception de l'ensemble des composants requis. Des modifications (avenants) aux commandes clients peuvent être apportées après l'enregistrement de la commande, afin d'amender :

les codes articles (dans le cas d'une évolution technique de l'article),

ou les autres caractéristiques de l'article (délai, quantité commandée).

La direction commerciale se charge aussi du suivi de la facturation. Les coûts réels sont comptabilisés par commande reçue, avec une gestion du nombre d'heures passées (heures productives transmises par les usines) et de la valeur (prix d'achat) des matières premières ou pièces semi-finies en provenance de fournisseurs divers (fournie par le service de gestion du stock). La facture est alors transmise à l'usine de

montage qui gère aussi la livraison de la commande.

1) Donner le diagramme des cas d'utilisation de ce système.

2) Donner une diagramme de classes du système.

3) Décrire le scénario « Traitement d’une commande » par un diagramme de séquence.

**public** **class** Chrono ... {

 **private** **long** duree;

 **private** **int** x, y, diametre;

 **private** JComponent proprietaire;

 **private** Thread deroulement;

 **private** **long** momentDebut = 0;

 **private** **long** tempsEcoule = 0;

 **private** **long** momentSuspension;

 **private** **boolean** continuer; // mis à faux pour suspendre

 **private** **boolean** finir; // à vrai pour finir

 **public** Chrono(JComponent proprietaire,**int** duree, **int** x, **int** y, **int** diametre)

{

 **this**.duree = (**long**)duree\*1000; // en ms

 **this**.x = x;

 **this**.y = y;

 **this**.diametre = diametre;

 **this**.proprietaire = proprietaire;

 }

 **public** **void** demarrer()

{

 **if** (enFonctionnement()) {

 arreter();

 **try** {

 deroulement.join(); // attend que le thread soit arrete

 }

 **catch**(InterruptedException exc) {}

 }

 deroulement = ... ; // création du thread

 deroulement ... ; // démarrage du thread

 }

 **public** ... **void** suspendre()

{

 **if** (... () && ...) {

 momentSuspension = System.currentTimeMillis();

 continuer = ...;

 }

 }

 **public** ... **void** reprendre()

{

 **if** (enFonctionnement() && !continuer) {

 momentDebut += System.currentTimeMillis() - momentSuspension;

 continuer = ...;

 ... ();

 }

 }

 **public** ... **void** arreter()

{

 **if** (... ()) {

 finir = ...;

 ... ();

 }

 }

 **public** **void** run()

{

 Thread.currentThread().setPriority(Thread.MIN\_PRIORITY);

 continuer = ...;

 finir = ...;

 momentDebut = System.currentTimeMillis();

 **while**((tempsEcoule < duree) && (!finir)) {

 tempsEcoule = System.currentTimeMillis() - momentDebut;

 proprietaire.repaint(**new** Rectangle(x, y, diametre, diametre));

 **try** {

 Thread.currentThread().sleep(200);

 ... (**this**) {

 **while** (!continuer && !finir)

 ... ();

 }

 }

 **catch**(InterruptedException e){}

 }

 }

 **public** **boolean** enFonctionnement()

{

 **return** (deroulement!= ...) && (deroulement....);

 }

 **public** **void** dessine(Graphics g) {

 g.setColor(Color.yellow);

 g.fillArc(x, y, diametre, diametre, 90, (**int**)(360-tempsEcoule\*360/duree));

 g.setColor(Color.green);

 g.fillArc(x, y, diametre, diametre, 90, (**int**)(-tempsEcoule\*360/duree));

 }

}

**public** **class** EssaiChrono **extends** JPanel **implements** ActionListener

{

 JButton demarrer = **new** JButton("demarrer");

 JButton suspendre = **new** JButton("suspendre");

 JButton reprendre = **new** JButton("reprendre");

 JButton arreter = **new** JButton("arreter");

 Chrono chrono;

 **public** EssaiChrono(**int** duree) {

 setPreferredSize(**new** Dimension(500, 300));

 setLayout(**new** FlowLayout(FlowLayout.***CENTER***, 10, 10));

 chrono = **new** Chrono(**this**, duree, 150, 120, 100);

 demarrer.addActionListener(**this**);

 suspendre.addActionListener(**this**);

 reprendre.addActionListener(**this**);

 arreter.addActionListener(**this**);

 add(demarrer);

 add(suspendre);

 add(reprendre);

 add(arreter);

 setVisible(**true**);

 }

 **public** **void** actionPerformed(ActionEvent evt) {

 Object source = evt.getSource();

 **if** (source == demarrer)

 chrono.demarrer();

 **else** **if** (source == suspendre)

 chrono.suspendre();

 **else** **if** (source == reprendre)

 chrono.reprendre();

 **else** **if** (source == arreter)

 chrono.arreter();

 }

 **public** **void** paintComponent(Graphics g) {

 **super**.paintComponent(g);

 chrono.dessine(g);

 }

 **public** **static** **void** main(String[] argv) {

 JFrame fenetre = **new** JFrame();

 fenetre.setContentPane(**new** EssaiChrono(Integer.*parseInt*(argv[0])));

 fenetre.addWindowListener(**new** WindowAdapter() {

 **public** **void** windowClosing(WindowEvent evt) {

 System.*exit*(0);

 }

 });

 fenetre.setLocation(100, 100);

 fenetre.pack();

 fenetre.setVisible(**true**);

 }

}