Sujet DS S2 POO Juin 2017

Documents autorisés : tout sauf livres. Durée : 2h

Vous devez lire chaque exercice en entier avant de commencer à répondre.

On ne vous demande pas de mettre vos attributs en private ni de donner les accesseurs.   
Vous devez donner les constructeurs.

# EXERCICE 1: collections et généricité (7 points)

On vous donne de la documentation en annexe 1 sur les itérateurs et les collections.

1. Est-ce que Iterator est une classe concrète, une interface ou une classe abstraite ?

|  |
| --- |
| D’après la documentation, c’est une interface puisqu’il y a le mot clé “interface” (1 point) |

1. Est-ce que Iterator fait intervenir la programmation générique ? comment le voyez-vous dans la documentation fournie en annexe ?

|  |
| --- |
| L’interface Iterator fait intervenir la programmation générique puisqu’il y a le symbole <E> (1 point) |

1. Vous devez créer une classe RandomIterator qui implémente un itérateur parcourant les éléments d’une liste au hasard (dans un ordre aléatoire). Votre classe devra fonctionner pas seulement avec des ArrayList mais aussi avec des LinkedList, des Stack, ... Voici un exemple d’utilisation de RandomIterator :

import java.util.\*;  
public class TestRandomIterator {  
 public static void main (String args []) {  
 // Créer un arraylist et y ajouter des nombres  
 ArrayList<Integer> liste = new ArrayList<Integer>();  
 liste.add(8);  
 liste.add(1);  
 liste.add(4);  
 liste.add(6);  
  
 // Créer un itérateur aléatoire sur cet arraylist  
 **RandomIterator**<Integer> it =   
 new **RandomIterator**<Integer>(liste, new Random());  
  
 // Parcourir l'array list dans un ordre aléatoire grace à l'itérateur  
 while (it.hasNext()) {System.out.println (it.next());}  
 }  
}  
Voici un exemple d’utilisation de cette classe et l’exécution qui en découle :

4

6

1

8

Une autre exécution donne l’affichage suivant :

4

8

1

6

On vous donne le code de la classe RandomIterator de manière incomplète ci-dessous :

Code complété :

|  |
| --- |
| **class RandomIterator<E> implements Iterator<E> {** (2 points : 0,5 pour le 1er <E>, 0,5 pour impléments, 0.5 pour Iterator, 0,5 pour le 2ème <E>)  private final boolean[] dejaTire ;   private int nbDejaTire ;   private final int TAILLE ;   private Random rand ;   private int indiceMin, indiceMax ;   **private final List<E> liste ;** (1 point : 0,5 pour List, 0,5 pour <E>)  **public RandomIterator (List<E> l, Random r) {** (1 point : 0,5 pour <E>, 0,5 pour le reste)  liste = l ;   rand = r ;   TAILLE = l.size();  dejaTire = new boolean [TAILLE];   nbDejaTire = 0 ;   indiceMin = 0 ;   indiceMax = TAILLE - 1 ;   }   public boolean hasNext() {  return nbDejaTire < TAILLE ;  }   **public E next() {** (0,5 point) int etendue = indiceMax - indiceMin + 1 ;   int indice ;    do { indice = indiceMin + rand.nextInt(etendue);}  while (dejaTire[indice]);   if (indice == indiceMin) indiceMin++;   else if (indice == indiceMax) indiceMax--;  dejaTire[indice] = true ;   nbDejaTire ++ ;   **return liste.get(indice);** (0,5 point) }  } |

# 

# EXERCICE 2: interfaces graphiques (13 points)

1) Quel est l’intérêt du modèle MVC (3 lignes max) ? (1 point)

|  |
| --- |
| MODULARITE : on sépare bien les données (modèle), leurs présentations (graphique) pour l’utilisateur (les vues) et la manière dont les actions de l’utilisateur sur l’interface impacteront les données et les vues (les contrôleurs). |

Vous devez programmez un programme de coaching conseillant l’utilisateur et suivant ses activités sportives.

|  |  |
| --- | --- |
| boutons-fenix-3-300x257.jpg | Bouton 1 : LIGHT  Bouton 2 : START/STOP  Bouton 3 : BACK+LAP  Bouton 4 : DOWN  Bouton 5 : UP  (dans cet énoncé on ne s’occupera que des boutons qui sont sur le côté droit : START/STOP et BACK/LAP). |

Vous commencerez par programmer une application en Java sous Eclipse.

Une activité peut être de plusieurs types. On se limitera ici pour simplifier à course à pied mais il pourrait y en avoir d’autres. Toutes les activités ont les informations suivantes : vitesse actuelle, nombre de calories brûlées jusqu’à maintenant. L’activité de course à pied a en plus : un chrono (minutes et secondes) qui indique le temps écoulé depuis le début de l’activité et une valeur de cadence (nombre de pas par minute).

2) Donner les définitions de ModeleActivite, ModeleActiviteCourse (à vous de choisir si ce sont des interfaces, des classes abstraites ou des classes concrètes).

(2,5 pts)

0,5 entete ModeleActivite

0.5 attributs ModeleActivite

0,5 entete ModeleActiviteCourse

0.5 attributs ModeleActiviteCourse

0,5 package

|  |
| --- |
| package modeles;  **public abstract class ModeleActivite {**  **private double vitesseActuelle ;**  **private double nbCaloriesBrulees ;**  **}**  package modeles;  **public class ModeleActiviteCourse extends ModeleActivite {**  **private int heureChrono ;**  **private int minutesChrono ;**  **private int cadence ;**  } |

## 

On vous donne le code de la classe VueHeure ainsi que ce qu’elle affiche :



package vues;  
import java.awt.\*;  
import java.text.\*;  
import java.util.\*;  
  
import javax.swing.\*;  
  
public class VueHeure extends JPanel {  
 public VueHeure () {  
 this.setBounds(10,10,300,300);  
  
 // Récupérer la date, puis l'heure actuelle  
 Date dNow = new Date( );  
 SimpleDateFormat ft = new SimpleDateFormat ("hh:mm");  
 add (new JLabel (ft.format(dNow)));  
 }  
   
 public static void main (String args []) {  
 JFrame f = new JFrame ();  
 f.setSize(300, 300);  
 f.add(new VueHeure ());  
 f.setVisible(true);  
 }  
}  
3) C’est une vue sur quel type de donnée ? Pourquoi est-ce qu’il n’y a pas une classe ModeleHeure ? (1 pt)

|  |
| --- |
| C’est une vue sur l’heure actuelle. (0,5 pt)  Il n’y a pas de modèle car l’heure actuelle est récupérée directement durant l’exécution à l’aide de la classe Date. (0,5 pt) |

4) On vous donne ci-dessous l’exécution de la classe VueActiviteCourse. Donnez le code Java.

(question sur 3 pts :

entete classe : 0,5

Entete constructeur : 0,5

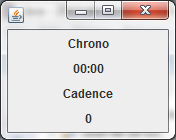
Mémoriser le modèle dans un attribut : 0,5

Attributs 0,5

Créer les composants : 0,5

Mise en page : 0,5

)



|  |
| --- |
| package vues;  import java.awt.\*;  import java.text.\*;  import java.util.\*;  import javax.swing.\*;  import modeles.\* ;  **public class VueActiviteCourse extends JPanel {**  **ModeleActivite m ;**    JLabel chrono ;  JLabel valeurChrono ;  JLabel cadence ;  JLabel valeurCadence ;    **public VueActiviteCourse (ModeleActivite m) {**  this.m = m ;  setBounds(10,10,300,300);    **// Créer les composants**  chrono = new JLabel ("Chrono");  cadence = new JLabel ("Cadence");  valeurChrono = new JLabel ("00:00");  valeurCadence = new JLabel ("0");  chrono.setHorizontalAlignment(SwingConstants.CENTER);  cadence.setHorizontalAlignment(SwingConstants.CENTER);  valeurChrono.setHorizontalAlignment(SwingConstants.CENTER);  valeurCadence.setHorizontalAlignment(SwingConstants.CENTER);    // Les mettre en page  **setLayout(new GridLayout (4, 1));**  add(chrono);  add(valeurChrono);  add(cadence);  add(valeurCadence);  }  public static void main (String args []) {  JFrame f = new JFrame ();  f.setSize(300, 300);  ModeleActiviteCourse m = new ModeleActiviteCourse();  **f.add(new VueActiviteCourse (m));**  f.setVisible(true);  }  } |

5) Est-ce que cette VueActiviteCourse affiche la valeur de tous les attributs d’une activité de course ? Est-ce normal ? (1 pt)

|  |
| --- |
| VueActivitéCourse n’affiche pas tous les attributs d’une activité course : vitesseActuelle et nbCaloriesBrulees. (0,5 pt)  C’est normal : c’est juste un type de vue sur certaines informations du modèle : une vue ne donne qu’un point de vue choisi par le concepteur et n’affiche donc pas forcément toutes les informations disponibles dans le modèle. (0,5 pt) |

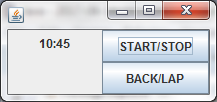
6) Donner le code de la classe Controleur permettant d’afficher les éléments suivants:

(3 pts)

1 pt gestion des événements (selon la solution qu’ils ont choisie)

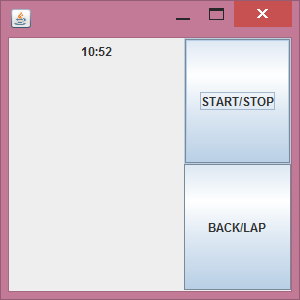
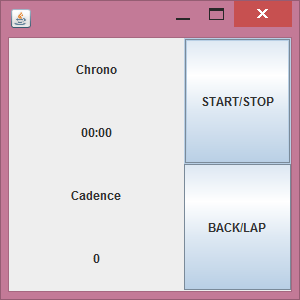
1 pt déclaration, création et mise en page des composants

1 pt nécéssité que l’écouteur ait une référence vers le composant qui a une référence vers la vue (une copie de la valeur ne suffit pas puisqu’il faut pouvoir modifier la vue).



Puis quand on clique sur le bouton START / STOP, la vue heure passe à une vue ActiviteCourse

|  |
| --- |
| package controleurs;  import java.awt.GridLayout;  import java.awt.Panel;  import java.awt.event.ActionEvent;  import java.awt.event.ActionListener;  import javax.swing.\*;  import vues.VueActiviteCourse;  import vues.VueHeure;  import modeles.ModeleActivite;  import modeles.ModeleActiviteCourse;  public class ControleurDroite extends Panel implements ActionListener {  private ModeleActivite m ;  private JPanel vue ;    private JButton startStop ;  private JButton lap ;  private ApplicationSport a ;    public ControleurDroite (ModeleActivite m, JPanel vue, ApplicationSport a) {  // Mémoriser les paramètres dans des attributs pour pouvoir y accéder dans actionPerformed  super();  this.m = m;  this.vue = vue;  this.a = a ;    // Créer les composants graphiques  startStop = new JButton ("START/STOP");  lap = new JButton ("BACK/LAP");    // Gérer les événements  startStop.addActionListener(this);  lap.addActionListener(this);    // Mettre en page  setLayout (new GridLayout (2, 1));  add (startStop);  add (lap);  }  public static void main(String[] args) {  JFrame f = new JFrame ();  f.setSize(300, 300);  f.add(new ControleurDroite (null, null, null));  f.setVisible(true);  }  public void actionPerformed(ActionEvent arg0) {  if (vue instanceof VueHeure) {  // Créer un modèle d'activité de course et l'affecter à ce controleur  m = new ModeleActiviteCourse ();    // Remplacer la vue actuelle par une vue sur l'activité de course  a.setVue(new VueActiviteCourse (m));  }  }  }  package controleurs;  import java.awt.BorderLayout;  import javax.swing.JFrame;  import javax.swing.JPanel;  import modeles.ModeleActiviteCourse;  import vues.VueHeure;  public class ApplicationSport extends JPanel {  JPanel v ;  ControleurDroite c ;    public ApplicationSport () {  setLayout(new BorderLayout());  v = new VueHeure ();  ModeleActiviteCourse m = new ModeleActiviteCourse();  c = new ControleurDroite (m, v, this);  add (v, BorderLayout.CENTER);  add (c, BorderLayout.EAST);  }    public static void main(String[] args) {  JFrame f = new JFrame ();  f.setSize(300, 300);    ApplicationSport a = new ApplicationSport();  f.add(a);  f.setVisible(true);  }    public JPanel getVue() {  return v;  }  public void setVue(JPanel v) {  // Supprimer tous les composants de ce panel  removeAll();  // Changer la vue  this.v = v;    // Ré-ajouter les composants  add (v, BorderLayout.CENTER);  add (c, BorderLayout.EAST);    // Rafraichir le panel  updateUI();  }  } |

7) Votre programme doit être en partie traduit en un autre langage qui permet d’embarquer des programmes sur une montre connectée : Monkey C. Lisez en annexe des éléments sur ce langage.

Donnez et expliquez 3 différences entre le langage MonkeyC et Java (vous les numéroterez a) b) c)

Pour chacune expliquez en quelques mots comment cela fonctionne en MonkeyC et comment cela fonctionne en Java. (1,5 pts)

0,5 pts pour a

0,5 pts pour b

0,5 pts pour c

|  |
| --- |
| a) En MonkeyC, il n’y a pas de types de base (int, float …) : tout est objet. En java on distingue les types de base (par exemple int) et les classes (Integer, String, Voiture, …).  b) En MonkeyC les paramètres passés aux méthodes (appelées fonctions en MonkeyC) ne sont pas typés. On peut donc passer n’importe quel type. En java les paramètres des types sont définis dans l’entête de la méthode et le compilateur vérifie que le type des paramètres passés lors d’un appel est compatible avec celui défini pour le paramètre correspondant.  c) En Monkey C, on peut stocker une méthode dans une variable pour la rappeler plus tard.  Nous avons vu un processus similaire en interfaces graphiques avec la possibilité d’enregistrer des écouteurs et d’appeler ensuite une méthode connue de ces écouteurs (par exemple actionPerformed).  Pour information (non vu en cours)  Il y a une classe  public final class Method extends AccessibleObject implements GenericDeclaration, Member A Method provides information about, and access to, a single method on a class or interface. The reflected method may be a class method or an instance method (including an abstract method). A Method permits widening conversions to occur when matching the actual parameters to invoke with the underlying method's formal parameters, but it throws an IllegalArgumentException if a narrowing conversion would occur.  Object invoke(Object obj, Object... args) Invokes the underlying method represented by this Method object, on the specified object with the specified parameters. |

Annexe 1 : Itérateurs et collections

java.util  
Interface Iterator<E>  
  
Type Parameters:  
E - the type of elements returned by this iterator

public interface Iterator<E>  
An iterator over a collection.

Methods   
boolean hasNext()  
Returns true if the iteration has more elements.

E next()  
Returns the next element in the iteration.

void remove()  
Removes from the underlying collection the last element returned by this iterator (optional operation).

java.util  
Interface List<E>  
  
Type Parameters:  
E - the type of elements in this list

All Superinterfaces:  
Collection<E>, Iterable<E>

All Known Implementing Classes:  
AbstractList, AbstractSequentialList, ArrayList, AttributeList, CopyOnWriteArrayList, LinkedList, RoleList, RoleUnresolvedList, Stack, Vector

Annexe 2 : Monkey C

Like Java, Monkey C compiles into byte code that is interpreted by a virtual machine. Also like Java, all objects are allocated on the heap, and the virtual machine cleans up memory (Java through garbage collection, Monkey C through reference counting). Unlike Java, Monkey C does not have primitive types—integers, floats, and chars are objects. This means primitives can have methods just like other objects.

While Java is a statically typed language, Monkey C is duck typed. In Java, the developer must declare the types for all parameters of a function, and declare the return value type. The Java compiler checks these at compile time to ensure type safety. Duck typing is the concept of “if it walks like a duck, and quacks like a duck, then it must be a duck”[1]. For example:  
function add( a, b ) {  
 return a + b;  
}  
  
function thisFunctionUsesAdd() {  
 var a = add( 1, 3 ); // Return 4  
 var b = add( "Hello ", "World" ); // Returns "Hello World"  
}  
The Monkey C compiler does not verify type safety, and instead causes a runtime error if a function mishandles a method.

Callbacks

Functions in Monkey C are not first class, meaning you cannot pass them as objects directly. On a class instance, you can get a Method object. This object allows you to store a method in a variable for later invocation.

class Foo  
{  
 function operation(a, b) {  
 // The code here is really amazing. Like mind blowing amazing. You wish this method was in your program.  
 }  
}  
function usageSample() {  
 var v = new Foo();  
 // Get the callback for the operation method.  
 var m = v.method(:operation);  
 // Invoke v's operation method.  
 m.invoke(1,2);  
}

A Method object will invoke a method on the instance of an object it came from. It keeps a strong reference to the source object.