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 ?
2. Est-ce que Iterator fait intervenir la programmation générique ? comment le voyez-vous dans la documentation fournie en annexe ?
3. 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 :

*(1)* class RandomIterator……………………………………………………………………………………………………………………………

private final boolean[] dejaTire ;   
 private int nbDejaTire ;   
 private final int TAILLE ;   
 private Random rand ;   
 private int indiceMin, indiceMax ;

*(2)* private final ………………………………………………………………………… liste ;   
  
 *(3)* public RandomIterator………………………………………………………………………………………...  
 liste = l ;   
 rand = r ;   
 TAILLE = l.size();  
 dejaTire = new boolean [TAILLE];   
 nbDejaTire = 0 ;   
 indiceMin = 0 ;   
 indiceMax = TAILLE - 1 ;   
 }  
  
 public boolean hasNext() {  
 return nbDejaTire < TAILLE ;  
 }  
  
 *(4)* public …………………………………………………………………………………………………………………… {

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 ++ ;

*(5)* return …………………………………………………………………………………………………………………...  
 }   
}

=> Compléter sur **votre copie** le code de la classe RandomIterator

Il y a 5 lignes de pointillés numérotées de (1) à (5)   
=> sur votre copie indiquez ce que vous mettriez (reprendre toute la ligne depuis le début).

Par exemple vous commencerez par écrire sur votre copie:

1. class RandomIterator …**(indiquez sur votre copie comment vous compléteriez les “…”)**.

**Vous avez donc 5 lignes à écrire sur votre copie.**

# 

# EXERCICE 2: interfaces graphiques (13 points)

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

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).

## 

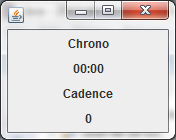
On vous donne le code de la classe VueHeure ainsi que ce qu’elle affiche :



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);  
 }  
}

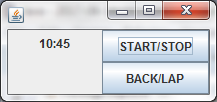
3) C’est une vue sur quel type de donnée ? Pourquoi est-ce qu’il n’y a pas une classe ModeleHeure ?

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



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

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



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

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.

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.