Numéro d'anonymat :

Modélisation et programmation par objets 2

Tous documents sur support papier autorisés. Durée : 2h
L'ensemble des réponses sera à donner sur les feuilles d'énoncé. Ne pas dégrapher les feuilles. Reportez votre numéro d'anonymat sur chaque feuille.

1 Interface, classe générique, streams

Dans cette partie, nous développons des éléments pour la représentation d'un parc de loisirs thématique qui propose des attractions de différentes sortes à différents publics.

Question 1. Ecrivez en Java une interface représentant une attraction dans le contexte d'un parc d'attractions thématique. Pour une attraction, on peut connaître un descriptif (chaîne de caractères), l'âge minimum conseillé (en années) pour les spectateurs et sa durée (en minutes).

Question 2. Ecrivez en Java une classe concrète représentant une attraction animalière dans le contexte d'un parc d'attractions thématique. En plus des informations proposées dans l'interface, une attraction animalière comprend une chaîne de caractères contenant les noms des espèces des animaux présentés lors de l'animation. N'écrivez que les attributs et que les méthodes suffisant à rendre la classe concrète. N'écrivez pas les constructeurs mais ils seront supposés exister, de même que les accesseurs non écrits.

Question 3. Un parc de loisirs thématique possède un nom et une liste d'attractions. Ecrivez en Java l'entête et les attributs (avec initialisation) d'une classe générique représentant un parc de loisirs thématique. La classe générique est paramétrée par un type d'attraction (conforme à l'interface définie plus haut).

Question 4. Ecrivez en Java, dans la classe représentant les parcs de loisirs, une méthode `dureeMoyAttrAns` retournant la durée moyenne des attractions destinées aux spectateurs qui ont plus (au sens large) d'un certain âge passé en paramètre. Cette méthode doit être impérativement écrite en utilisant les **opérations sur les flots (streams) et les lambdas expressions** et non pas une classique itération avec un `for` ou un `while`, ni avec un itérateur.

2 Liaison dynamique, surcharge statique

Question 5. Etudiez le programme suivant.

```
public class A {
    public void m2(){System.out.println("m2 de A");}
    public void m1(A a){System.out.println("m1 de A");this.m2();}       }

public class B extends A{
    public void m2(){System.out.println("m2 de B"); }
}

public class C extends B{
    public void m1(C c){System.out.println("m1 de C"); super.m1(c);}
    public void m2(){System.out.println("m2 de C"); }                  }

public class ProgrammeA {
    public static void main(String[] args) {
        A paramab = new C();    C paramcc = new C();
        C objC = new C();
        objC.m1(paramab);    System.out.println("----");    objC.m1(paramcc);
    }    }
```

a- La méthode `void m1(C c)` est-elle une redéfinition de la méthode `void m1(A a)`. Justifiez.

b- Indiquez ci-dessous ce que le programme affiche (justifiez grâce aux types dynamiques et statiques des objets et variables).

3 Diagramme d'états

Question 6. Vous dessinerez en UML le diagramme d'états d'une station de lavage automatique de voiture avec les informations suivantes. Nous ne traitons pas la partie paiement. Initialement la station est en attente. Puis la commande `démarrer()` amène dans un état complexe de `lavageComplet`. Le lavage complet est constitué d'une période de nettoyage et d'une période de séchage. La période de nettoyage est complexe et comprend elle-même quatre périodes durant chacune 4mn dont trois pendant lesquelles un balai nettoyant frotte les surfaces avec un savon adapté (nettoyage de l'avant-4mn, nettoyage du dessus de la voiture-4mn, nettoyage de l'arrière-4mn) et une quatrième pendant laquelle un rinçage est effectué (4mn). Enfin le séchage est effectué (5mn). A tout moment du lavage complet, la commande d'arrêt d'urgence peut être actionnée. La période d'arrêt est de 6mn. Pendant cette période, l'utilisateur peut à tout moment actionner la commande de reprise, qui reprend au début de la sous-période de nettoyage courante ou au début de l'étape de lustrage. Si au bout des 6mn la commande de reprise n'a pas été actionnée, on passe dans l'état final.
Nota : la notation d'un événement temporel de type "fin d'une période de temps" de $x$mn sur une transition s'écrit `after(x mn)`

4 Test

Listing 1 – SUT.java

```
package terminal17test;

public class SUT {
	private String mot;

	public SUT(String mot){
		this.mot=mot;
	}

	public boolean contientEspaceOuApostropheouVirgule(){
		return mot.contains("␣") || mot.contains(",")|| mot.contains("'");
	}

	private boolean estPalindromeAux(int start, int end)
	{
		if ((end - start) < 2)
		{
			return true;
		}
		if (mot.charAt(start) != mot.charAt(end))
		{
			return false;
		}
		start++;
		end --;
		return estPalindromeAux(start, end);
	}

	public boolean estPalindrome(){
		String motTemp=mot;
		miseEnMinuscules();
		boolean result=estPalindromeAux(0,mot.length()-1);
		mot=motTemp;
		return result;
	}

	public boolean estPalindromeModuloCaracteresSpeciaux(){
		String mot=this.mot;
		nettoyage();
		boolean result=estPalindrome();
		this.mot=mot;
		return result;
	}

	private void miseEnMinuscules(){
		mot=mot.toLowerCase();
	}

	private void nettoyage() {
		mot=mot.replaceAll("␣","");
		mot=mot.replaceAll(",","");
		mot=mot.replaceAll("'","");
	}
}
```

Une classe nommée SUT (pour System Under Test) a été développée. Elle vous est donnée au listing 1. Cette classe contient un attribut `mot` de type chaîne de caractères et des méthodes de manipulation de ce mot dont les méthodes publiques sont décrites ci-après.

- `contientEspaceOuApostropheouVirgule` vérifie si le mot contient un espace ou une apostrophe ou une virgule.
- `estPalindrome` vérifie si un mot est un palindrome, sans tenir compte des majuscules ou minuscules.
- `estPalindromeModuloCaracteresSpeciaux` vérifie si un mot est un palindrome, sans tenir compte des majuscules ou des minuscules, ni des caractères : espace, virgule ou apostrophe.

La classe SUT a été testée avec JUnit, via les classes `AllTests` (listing 2, `MonTest` (listing 3) et `TestParam` (listing 4) qui vous sont données.

Listing 2 – SUT.java

```
package terminal17test;

import org.junit.runner.RunWith;
import org.junit.runners.Suite;
import org.junit.runners.Suite.SuiteClasses;

@RunWith(Suite.class)
@SuiteClasses({ MonTest.class, TestParam.class })
public class AllTests {

}
```

Listing 3 – SUT.java

```
package terminal17test;
import static org.junit.Assert.*;

import java.util.ArrayList;

import org.junit.Before;
import org.junit.Test;

public class MonTest {

	SUT sut;

	ArrayList<String> vraisPalindromes;
	ArrayList<String> palindromesModuloPonctuation;

	@Before
	public void setUp(){
		vraisPalindromes=new ArrayList<String>();
		vraisPalindromes.add("Laval");
		vraisPalindromes.add("serres");
		palindromesModuloPonctuation=new ArrayList<String>();
		palindromesModuloPonctuation.add("ce reptile relit Perec");
		palindromesModuloPonctuation.add("et Luc lamina l'animal culte");
	}

	@Test
	public void testVraiPalindrome() {
		for (String mot:vraisPalindromes){
			sut=new SUT(mot);
			assertTrue(sut.estPalindrome());
		}
	}

	@Test
	public void testPalindromeModuloCaracteresSpeciaux(){
		for (String mot:palindromesModuloPonctuation){
			sut=new SUT(mot);
			assertTrue(sut.estPalindromeModuloCaracteresSpeciaux());
		}
	}
}
```

Listing 4 – SUT.java

```
package terminal17test;
import static org.junit.Assert.*;

import java.util.Arrays;
import java.util.Collection;

import org.junit.Test;
import org.junit.runner.RunWith;
import org.junit.runners.Parameterized;
import org.junit.runners.Parameterized.Parameters;

@RunWith(Parameterized.class)
public class TestParam {
	SUT sut;
	boolean estP;
	boolean estPmoduloNettoyage;

	@Test
	public void test() {
		assertEquals(sut.estPalindrome(), estP);
		assertEquals(sut.estPalindromeModuloCaracteresSpeciaux(), estPmoduloNettoyage);
	}
	@Parameters
	public static Collection data() {
		return Arrays.asList(new Object[][]{
			{"Laval", true, true},
			{"kayak", true, true},
			{"esope reste et se repose", false, true},
			{"esoperesteetserepose", true, true},
			{"tu l'as trop ecrase cesar ce port salut", false, true},
			{"essayasse", true, true},
			{"php", true, true},
			{"ressasser", true, true},
			{"abracadabra", false, false}
		});
	}


	public TestParam(String s, boolean b1, boolean b2) {
		sut=new SUT(s);
		estP=b1;
		estPmoduloNettoyage=b2;
	}

}
```

Question 7. Si on exécute la suite de test de la classe `AllTests` via JUnit, combien de tests sont exécutés ? (test est à comprendre ici au sens JUnit, c'est-à-dire méthode préfixée par l'annotation `Test`). Si une même méthode est exécutée par exemple 7 fois, cela compte pour 7 tests.

Question 8. Si on exécute les cas de test de la classe `MonTest` via JUnit, donnez l'ordre d'exécution des méthodes de cette même classe `MonTest`.

Question 9. Expliquez comment compléter la classe `TestParam` pour tester la méthode `contientEspaceOuApostropheouVirgule`.

Question 10. Les classes de test `MonTest` et `TestParam` procèdent de manière différente pour organiser les tests. Y a-t-il une de ces deux approches qui vous parait plus simple pour le diagnostic ? Si oui laquelle ? Si non pourquoi ? Justifier.