Programmation objet : concepts avancés
Examen 2016-2017

Durée: 2 heures

Aucun document est autorisé

Le soin apporté à la rédaction et à la présentation ainsi que la rigueur seront pris en compte dans la notation.

1 Scala (10 points)

Question 1 *Expliquez brièvement la différence entre l'extension d'une classe par héritage, et l'extension par composition mixin avec des traits.*
***Bonus** : Expliquez la différence avec l'héritage multiple.* □

```
class A {
  def m () = println ("A.m")
}
class B {
  def m () = println ("B.m")
}
abstract class C extends A {
  def n () : Unit
}
trait T1 {
  def m () = println ("T1.m")
}
trait T2 extends A {
  override def m () { super.m (); println ("T2.m") }
}
trait T3 extends C with T2 {
  override def n () { m () }
}
```

Question 2 *Pour chacune des déclarations suivantes évaluées dans le contexte du programme ci-dessus, indiquez si elle est acceptée par le compilateur* Scala *et ce que produit son évaluation. Si elle est rejetée, expliquez brièvement pourquoi.*

1. `val a = (new A with T1).m ()` rejete manque override
2. `val b = (new A with T2).m ()` A.m T2.m
3. `val c = (new B with T2).m ()` B.m T2.m
4. `val d = (new C with T3).n ()` A.m T2.m
5. `val e = (new C with T3 { override def m ()= n ()}).n ()` rejete

□

Question 3 *Pour la déclaration ci-dessous, dire quelles définitions sont utilisées pour des appels à `f.n()` et `f.m()` :*

```
val f = new C with T3 with T2
```

□

```
abstract class Ordered {
       type O
       def < (that: O) : Boolean
       def <= (that: O) : Boolean = this < that || this == that
}

abstract class Pair[T](var fst : T, var snd : T)

sealed abstract class OrderedPair[T <: Ordered {type O = T}](a:T, b:T, u:Unit)
       extends Pair(a, b) {
       def this(a : T, b : T) =
              this(if (a < b) a else b, if (a < b) b else a, ())
}

class OrderedInt(val a : Int) extends Ordered {
       type O = OrderedInt
       def < (that : OrderedInt) = a < that.a
}

implicit def intToOrderedInt (a : Int) : OrderedInt = new OrderedInt(a)
implicit def orderedIntToInt (a : OrderedInt) : Int = a.a

sealed class IntPair(a : Int, b : Int) extends OrderedPair[OrderedInt](a,b)
```

FIGURE 1 – Ordered

Question 4 *On considère le programme donné dans la Figure 1.*

1. *Expliquez le rôle de la classe* `Ordered` *et de son type* `O`*.*
2. *Expliquez le rôle des définitions de* `intToOrderedInt` *et* `orderedIntToInt`*.*
3. *Donnez la séquence des appels à exécuter à la construction d'un objet du type IntPair (par exemple* `val a = new IntPair(6, 3)`*).*
4. *Expliquez la contrainte du type T dans la classe* `OrderedPair`*.*

□

Question 5 *Étant donné le trait* `TR` *:*

```
trait TR[S <: TR[S]] {
   def self : S
   def modify : S = ...
}
```

1. *Que représente la contrainte de type du trait* `TR` *?*
2. *Quand est cette contrainte utile ?*

□

```
abstract class SubjectObserver {
  type O <: Observer
  type S <: Subject

  abstract class Subject {
    self:S =>
    var observers : List[O] = Nil
    def register (new_observers : O*) =
      observers ++= new_observers
    def notifyObservers =
      observers.foreach (_.onNotify (this))
  }
  abstract class Observer { def onNotify (s : S) }
}
```

FIGURE 2 – Subject-Observer

2 Sujet-Observateur (5 points)

Le code de la Figure 2 a été proposé lors du cours pour le pattern sujet-observateur.

Question 6 *À quel problème répond le patron de conception sujet-observateur ?* □

Question 7 *Quelle est l'avantage de travailler avec des types virtuels `O` et `S`, plutôt qu'utiliser directement les classes abstraites `Observer` et `Subject` ?* □

Question 8 *Quelle est le rôle de la ligne `self:S =>` dans la classe `Subject` ?* □

Question 9 *Quelles sont les garanties assurées par cette implémentation ? Quels sont ces désavantages ?* □

Question 10 *En utilisant le pattern de la Figure 2 donnez une instanciation simple ou le sujet est un compteur entier (peut être incrémenté), et les ~~sujets~~ imprime la valeur du compteur quand ils sont notifiés. Simplifiez votre implémentation au maximum.* □

3 Projet (5 points)

Question 11 *Donnez une vue globale des interfaces de l'implémentation de : la carte, les joueurs et les objets dans votre projet POCA. Indiquez les méthodes utilisées pour leurs interactions. Seulement l'éstructure générale de l'implémentation est demandée. Pas besoin d'inclure chaque fonction. Il est recommandé d'utiliser des diagrammes à la UML.* □

Question 12 *Expliquez quelles extensions sont simplifiées par votre architecture. Critiquez-la.* □