QCM SETMO Master SEMS année 2016-2017
lundi 9 octobre 2016

NOM :

PRENOM :

Il peut avoir plusieurs réponses à certaines questions, indiquer toutes les réponses correctes. Les réponses correctes sont indiquées en vert.

Programmation orientée objet et Java

Qu'est-ce que le JDK ?

a. Une application Java
b. L'ensemble des outils permettant de traiter un développement en langage Java
c. L'ensemble des librairies de base du langage
d. Cela n'a aucun rapport avec Java

Qu'est-ce que la JVM ?

a. Une bibliothèque pour créer des programmes Java
b. Un programme qui lit du byte code Java et l'exécute pour la machine courante
c. Un interpréteur Java
d. Cela n'a aucun rapport avec Java

Quels sont les notions fondamentales de la POO ?

a. L'héritage
b. Le polymorphisme
c. Le développement séquentiel
d. L'encapsulation

Qu'est-ce qu'une classe Java ?

a. Le plan pour créer des objets de cette classe
b. Un fichier au format quelconque
c. Une représentation d'un type construit par un programmeur
d. Un code test pour une partie du logiciel

Qu'elles sont les particularités d'un constructeur ?

a. Le mot `void` est indiqué comme type de retour
b. Le compilateur en définit un dans certains cas qui peut être enlevé dans certains autres cas
c. Il ne peut pas être surchargé
d. Il est toujours utilisé avec le mot clé réservé `new`

Qu’est-ce qu’une interface ?

a. Une classe concrète
b. Un plan ne pouvant contenir que des déclarations de méthodes et des variables publiques et statiques
c. Un outil permettant de modéliser l'héritage multiple (grr !)
d. Un outil indiquant des spécifications que devront implémenter des classes

A quoi sert le caractère @ dans un code Java ?

a. A indiquer une adresse mail dans le code
b. Un outil associé à la notion de Javadoc
c. Permet de décrire une erreur dans le code
d. Permet d’enrichir le code à l'aide d’annotations

Une annotation est un qualifieur qui peut être pris en compte par :

a. uniquement le compilateur Java
b. uniquement la JVM
c. uniquement un environnement d'exécution
d. éventuellement plusieurs des 3 cas précédents a), b), c)

A quoi peuvent servir les exceptions ?

a. A faire traiter des cas exceptionnels par le code appelant une méthode
b. A enrichir la lisibilité du code
c. A dérouter l'exécution d'un programme vers un traitant de cas exceptionnels spécifiques
d. A hiérarchiser les types de cas exceptionnels

Qu’est-il possible de faire pour le traitement des exceptions ?

a. On peut mettre un bloc `finally` qui sera toujours appelé
b. On peut utiliser le mot clé `throws` pour repousser le traitement de l'exception dans l'appelant de la méthode courante
c. On peut inclure un bloc `try-catch` pour traiter l’exception
d. On est obligé de mettre un bloc `finally`

La généricité

a. permet de construire des familles de classes traitant de types qui seront précisés plus tard dans le code
b. permet de construire des `ArrayList` qui traite d'objets d'une classe précise
c. permet l'héritage multiple
d. n'existe pas en langage Java

Le ramasse-miettes Java permet

a. De construire automatiquement des objets
b. Une prise en charge par la JVM de la destruction des objets devenus inaccessibles
c. L’allocation automatique des objets en entrée de bloc
d. La désallocation automatique des objets en sortie de bloc

L'héritage

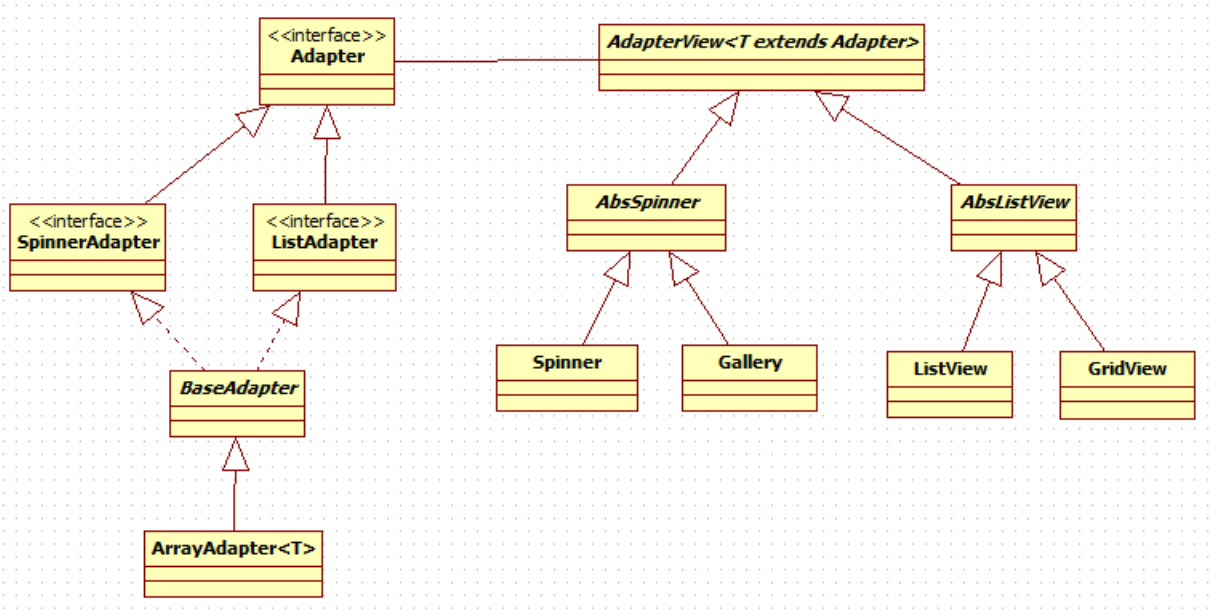
a. permet de récupérer intégralement toutes les méthodes et données membre d'une classe
b. permet de factoriser dans une classe des notions communes à plusieurs autres classes
c. peut être multiple
d. ne s'applique pas sur les interfaces

Le polymorphisme

a. est une notion uniquement traitée à la compilation
b. doit avoir l'héritage, la redéfinition de méthodes et une référence de classes de base pour être utilisé
c. permet de lancer du code sans savoir lequel au moment de la compilation
d. permet d'avoir des méthodes de même nom et de signatures différentes dans une classe

UML

Dans le diagramme suivant :



1ere question :

a. `ArrayAdapter<T>` est une classe générique
b. `ArrayAdapter<T>` dérive de la classe `BaseAdapter`
c. `ArrayAdapter<T>` implémente l'interface `BaseAdapter`
d. `ArrayAdapter<T>` est une interface

2ieme question :

a. `BaseAdapter` est une classe abstraite
b. `BaseAdapter` est une classe qui dérive des deux classes `SpinnerAdapter` et `ListAdapter`
c. `BaseAdapter` est une classe qui implémente les deux interfaces `SpinnerAdapter` et `ListAdapter`
d. `BaseAdapter` est une classe concrète

3ieme question :

a. `SpinnerAdapter` et `ListAdapter` sont deux classes qui dérivent de la classe `Adapter`
b. `SpinnerAdapter` et `ListAdapter` sont deux interfaces qui dérivent de l'interface `Adapter`
c. `SpinnerAdapter` et `ListAdapter` sont deux interfaces qui implémentent l'interface `Adapter`
d. `SpinnerAdapter` et `ListAdapter` sont les deux classes mère au sens de l'héritage de `BaseAdapter`

4ieme question :

a. `Adapter` dérive `AdapterView`
b. `AdapterView` dérive `Adapter`
c. `AdapterView` et `Adapter` sont associées c'est-à-dire que dans le code de l'un peut apparaître l'autre
d. `Adapter` est l'interface mère au sens de l'héritage des interfaces `SpinnerAdapter` et `ListAdapter`

5ieme question :

a. `AdapterView<T extends Adapter>` est une classe générique
b. `AdapterView<T extends Adapter>` est une classe abstraite
c. `T extends Adapter` signifie que la classe qui devra être mise à la place de `T` doit dériver de `Adapter`
d. `T extends Adapter` signifie que la classe qui devra être mise à la place de `T` doit être une classe ancêtre au sens de l'héritage de `Adapter`

6ieme question :

a. `AbsSpinner` est une interface
b. `AbsSpinner` est une classe dérivée de `AdapterView<T extends Adapter>`
c. `AbsSpinner` est une classe abstraite
d. `AbsSpinner` est un objet

7ieme question :

a. Il est indiqué que `AbsListView` est à la fois une `ListView` et une `GridView`
b. Il est indiqué qu'une `ListView` et une `GridView` sont des `AbsListView`
c. Les objets des classes `ListView` et `GridView` ont les champs et les méthodes de la classe `AdapterView<T extends Adapter>`
d. On peut créer des objets des classes `ListView` et `GridView`

Android

Pour Android, un AVD est :

a. le pilote qui gère l'audio et la vidéo
b. un émulateur de smartphone
c. une bibliothèque avancée pour construire des interfaces utilisateur
d. un débugger de code

L'interface graphique d'une activité a été définie dans un fichier `main.xml` mis dans le répertoire `res\layout`. Pour qu'une activité ait cette interface graphique il faut écrire le code :

a. `setContentView("/res/layout/main.xml");`
b. `setContentView("\res\layout\main.xml");`
c. `setContentView(R.layout.main);`
d. `setGUI("main.xml");`

Dans le fichier `main.xml` ci-dessus est déclaré `<Button ... android:id="@+id/mon_bouton ... />"` Pour repérer ce composant graphique `Button` dans le code Java de l'activité qui a chargé ce fichier `main.xml`, il faut écrire :

a. `Button bt = (Button) findViewbyId(R.id.mon_bouton);`
b. `Button bt = (Button) findViewbyId(R.layout.mon_bouton);`
c. `Button bt = (Button) findViewbyId(R.main.mon_bouton);`
d. `Button bt = (Button) getGUIComponent("mon_bouton");`

Le fichier `AndroidManifest.xml` d'une application Android

a. décrit en xml l'interface graphique de l'application Android
b. décrit la configuration de l'application Android (demande de permission d'accès à internet, déclaration des `activity`, etc.)
c. décrit les redirections à effectuer lorsqu'une erreur d'exécution se produit
d. peut être omis pour certaines Android apps

En android, la UI thread

a. est la première thread lancée
b. est rarement utilisée
c. s'occupe essentiellement des connexions réseau
d. traite les architectures client-serveur