|  |  |
| --- | --- |
|  |  |
|  | C:\Users\hamadi\AppData\Local\Microsoft\Windows\INetCache\Content.Word\logo_inti_vector-04.jpg  Institut des Nouvelles Technologies  de l’Information |

QCM

Design Patterns

Session 2016

## Guide d'utilisation

* Notez qu'en fonction de la question, il est possible de cocher une ou plusieurs réponses. Ce sera à vous de déterminer le nombre de bonnes réponses possibles.

## Quelques conseils de mise en œuvre

* Ne restez pas bloqué sur une question pour laquelle vous n'avez pas de réponse. Passez à la question suivante. Vous reviendrez par la suite sur les questions sans réponse.
* Identifiez les questions auxquelles vous avez répondu sans certitude. Que ces réponses soient justes ou non, cela vous permettra de mémoriser les questions sur lesquelles vous devrez revenir dans la correction du TEST.
* Ne cherchez pas à aller le plus vite possible. Le temps imparti doit être utilisé au maximum pour vous permettre d'effectuer une synthèse des points abordés dans le module.

**QCM**

**1. Qu’est-ce qu’un design pattern ?**

* Un segment de code réutilisable
* Une solution à un problème de conception
* Une solution à un problème algorithmique
* Les trois

**2. Qui a popularisé les design patterns ?**

* Les Three Amigos
* La Band of Five
* Le Gang of Four
* Brendan Eich

**3. Quel(s) principe(s) de la POO favorisent les patterns ?**

* Principe ouvert / fermé
* Couplage faible
* Programmer avec des interfaces
* Les trois

**4. Quel pattern semblerait adapté pour la création d’une station météo avec un capteur envoyant ses** **relevés à plusieurs témoins ou affichages ?**

* Fabrique
* Composite
* Observateur
* Visiteur

**5. Quel pattern semble évoquer le code suivant ?**

InterfaceA classeA = new ClasseA( new ClasseB( new ClasseC() ) );

* Décorateur
* Adaptateur
* Proxy
* Composite

**6. Quel est l’intérêt du pattern Stratégie ?**

* Encapsuler l’appel à la méthode d’une autre classe
* Déléguer l’implémentation de ses comportements à ses classe filles
* Encapsuler les comportements d’une classe et pouvoir les modifier dynamiquement
* Les trois

**7. Comment fonctionne le pattern Adaptateur ?**

* On implémente une interface cible et on utilise la classe source dans les méthodes via une composition
* On implémente une interface cible sur la classe source et on utilise ses propres méthodes
* On hérite de la classe source et l’on fait une composition de la classe implémentant l’interface cible
* On implémente une interface cible qui composera la classe source

**8. Quel pattern pourrait être utilisé pour ajouter un Logger à une classe ?**

* Proxy
* Adaptateur
* Observateur
* Etat

**9. Quel est l’intérêt du pattern Décorateur ?**

* Pouvoir encapsuler les comportements d’une classe et les modifier dynamiquement
* Pouvoir modifier le comportement d’un objet selon son contexte
* Pouvoir ajouter dynamiquement des responsabilités à un objet
* Pouvoir manipuler une arborescence d’objets comme un seul

**10. Laquelle de ces fabriques n’est pas un pattern ?**

* Fabrique Abstraite
* Fabrique
* Simple Fabrique
* Elles sont toutes trois des patterns

**11. Quel est l’objectif principal des Fabriques ?**

* Obtenir une conception qui ne repose pas sur des implémentations
* Centraliser la création des objets
* Remplacer les DAO
* Permettre de n’instancier les classes que quand on y fait directement appel pour optimiser l’utilisation de la mémoire

**12. Comment fonctionne la Fabrique (Factory Method) ?**

* La classe cliente délègue à une classe externe Fabrique l’instanciation des objets
* La classe cliente possède une méthode dédiée à l’instanciation des objets, celle-ci pourra être implémentée ou modifiée par les classes filles
* La classe cliente utilise une structure conditionnelle pour l’instanciation des objets
* On crée une interface fabrique qui sera implémentée par nos fabriques concrètes selon le type d’instance voulu

**13. A quoi sert une Fabrique Abstraite ?**

* A gérer l’instanciation d’une famille d’objets liés
* A ne pas utiliser de Fabrique ou Simple Fabrique
* A se passer d’interface dans la classe cliente
* A gérer l’instanciation d’une classe spécifique

**14. Une méthode de Fabrique doit être Static.**

* Vrai
* Faux

**15. Laquelle de ces affirmations ne correspond pas au pattern Commande ?**

* On peut l’utiliser pour la création d’interfaces utilisateur
* Il permet de découpler la classe appelante de la classe appelée
* Il peut permettre de gérer un historique ou un stack d’actions
* Il faut que les classes appelées implémentent la même interface

**16. Comment fonctionne le pattern Observateur ?**

* On implémente une classe Sujet sur la classe observable qui possède une méthode *accept(Observateur obs)*
* On implémente une classe Sujet sur la classe observable qui possède une méthode

*update()* avec ses attributs à transmettre en argument

* On implémente une interface Observateur sur les éléments observant qui possède une méthode *check(Sujet sujet)*
* On implémente une classe Sujet sur la classe observable qui possède une méthode *notifierObservateur()*

**17. Quel est l’objectif de la Facade ?**

* Regrouper tous les appels aux couches inférieures dans une seule classe
* Fournir une interface simplifiée aux classes clientes
* Faire le lien entre les couches d’accès aux données
* Permettre d’utilisation un groupe de classe comme une seule

**18. A quel pattern le Proxy ressemble-t-il dans son implémentation ?**

* Décorateur
* Adaptateur
* Stratégie
* État

**19. Quelle affirmation ne correspond pas au pattern État ?**

* Il permet de limiter les structures conditionnelles
* Il permet à un objet de changer de comportement selon le contexte
* Il permet d’ajouter de nouvelles méthodes à un objet selon le contexte
* Il permet l’ajout de nouveaux comportements et la maintenabilité de ceux-ci

**20. Le pattern Itérateur est implémenté par tous les tableaux et collections java.**

* Vrai
* Faux

FIN