Examinez les API Java ci-dessous et identifiez (en examinant sa structure) le patron de conception implémenté. Pour chaque patron de conception trouvé, expliquez comment il est implémenté.  
  
Java.lang.Runtime :   
 Dans ce API, chaque application Java a une seule instance de la classe Runtime. Le temps d’exécution peut être obtenu par la méthode getRuntime. La méthode getRuntime retourne UN objet associé avec la classe de Java actuelle et doit être invoquée avec respect à l’objet du temps d’exécution actuel. Donc, on peut dire que ce API est du patron Singleton parce qu’il est fréquent de retrouver des classes (cité ci-dessus) pour lesquelles il ne doit exister qu’UNE SEULE instance. Elle a aussi un constructeur public de ce qu’on peut en déduire parce qu’on peut obtenir l’objet de Runtime avec la méthode getRuntime de toutes les classes qui en ont besoin et il garantit ainsi que pas plus d’une instance sera créée.   
  
Java.xml.parsers.DocumentBuilderFactory :   
 Dans ce API, on définit « factory » APIs permettant aux applications d’obtenir un analyseur qui produit des arbres d’objet DOM des documents XML. Les constructeurs sont de type protected et ils empêchent l’instanciation. On utilise newInstance() pour obtenir (créer) une nouvelle instance de DocumentBuilderFactory. C’est une méthode static qui crée une nouvelle instance « factory ». Cette méthode vérifie entre la propriété système de javax.xml.parsers.DocumentBuilderFactory, les fichiers lib/jaxp.properties dans le répertoire JRE, les installations du fournisseur de service ou bien les paramètres système par défaut pour déterminer quelle classe à instancier une fois cette méthode appelée. Donc, on peut dire que ce API est du patron Fabrique parce que le cadriciel délègue la création des classes à une classe spécialisée appelée Fabrique qui est clairement une interface générique définie dans le cadriciel qui contient une méthode qui crée des instances de sous-classes d’une classe générique.   
  
Java.io.OutputStreamWriter :   
 Ce API hérite certaines méthodes de ces classes parents, Writer et Object et il est un pont entre les flux de caractères et les flux de bytes et les charactères qui sont écrit sont encodés en bytes utilisant un charset spécifié. Le charset utilisé peut être : spécifié par nom ou être donné explicitement ou le charset par défaut du plateforme pourrait être accepté. C’est aussi la classe parent de la classe FileWriter. Donc, on peut dire que ce API est du patron immuable parce qu’il n’y a pas des setters pour les variables d’instance.  
  
Javax.faces.event.PhaseListener :  
 Ce API est une interface qui hérite des interfaces EventListener et Serializable. Celle-ci est une interface implémentée par des objets qui veulent être notifiés au début et à la fin du processus pour chaque phase standard du cycle de traitement de la demande. Donc, on a des méthodes afterPhase qui traite une notification indiquant que le traitement d’une phase particulière vient d’être terminé, beforePhase, qui traite une notification indiquant que le traitement d’une phase particulière du cycle de vie du traitement de la demande est sur le point de commencer et getPhaseId qui retourne l’identificateur de la demande de la phase en cours de laquelle cet écouteur s’intéresse au traitement des événements PhaseEvent. Donc, on peut dire que ce API est du patron Observateur parce que ce API est une interface qui étend d’autres interfaces; Cela maximise la flexibilité du système qui utilisera cette API car ils peuvent choisir comment là mettre en œuvre. Cela réduit également l’interconnexion entre les classes qui nécessitent l’accès à ces méthodes.   
  
Javax.persistence.PersistenceContext :   
 Ce API exprime une dépendance sur un EntityManager géré par le conteneur et son contexte de persistance associé. Il fournit une implémentation optionnelle pour des méthodes pour le détail de l’élément. EntityManager est en soi une interface et ajouté à la liste de méthodes trouvés dans PersistenceContext à EntityManager force l’utilisateur à implémenter ces méthodes. Pour cette raison, EntityManager a délégué ces tâches à cette API, raison pour laquelle le modèle du patron Délégation est utilisé.