Master 1 informatique – Université Claude Bernard Lyon 1 – mardi 15 décembre 2015

# MIF 13 – Programmation Web – Examen

**Durée : 1 heure 30 – Documents autorisés – Ordinateurs, calculatrices et téléphones portables interdits**

## *Questions de cours (barème : 14 points)*

1. À quoi correspond un header « Content-Type » dans une requête HTTP et quand ne doit-il pas être utilisé ?
   Il permet de spécifier l'encodage des paramètres de la requête, à condition qu'ils soient envoyés dans le corps de cette requête.
   Il ne doit pas être utilisé dans une requête qui n'a pas de corps, comme GET, HEAD ou OPTIONS.
2. En quoi peut-on dire que le protocole AJP, utilisé dans Tomcat, fonctionne de façon similaire à CGI ?
   Il fait l'interface entre un serveur Web et un autre processus : AJP encapsule les requêtes dans des objets Java, les transmet au processus (Catalina), et désencapsule les réponses.
3. En JSP, quelle différence y a-t-il entre déclarer une variable dans un scriptlet et la déclarer dans une déclaration JSP ? À quoi cela correspond-il techniquement ?
   Dans un scriptlet, c'est une variable locale : elle sera dans le corps de la méthode de service de la servlet générée.
   Dans une déclaration, c'est une variable globale : ce sera une variable d'instance de la servlet.
4. Citez deux différences entre les mécanismes de gestion des Javabeans intégrés à un container de servlets et celui des beans Spring.
   Javabeans : instanciation par constructeur sans paramètre ; Spring : appel d'un constructeur avec les paramètres définis dans la configuration
   Javabeans : utilisation d'accesseurs simples (get, set) *via* jsp:getProperty / setProperty ; Spring : accès à toutes les méthodes
   Javabeans : pas de gestion des dépendances ; Spring : résolution du référentiel de dépendances entre les beans
   Gestion des scopes différente
5. Pourquoi dit-on que l'interface d'un service REST « est » le protocole HTTP et quelles sont les limites de cette expression ? Vous pouvez donner un exemple pour illustrer votre réponse.
   La sémantique des méthodes HTTP contraint les opérations sur les ressources exposées (GET -> récupérer, PUT : créer…). Cependant, une partie de cette sémantique est également portée par le format des représentations échangées et cette partie n'est pas documentée (que se passe-t-il si l'on fait un PUT avec uniquement un numéro de téléphone vers l'URI d'une personne : crée-t-on une personne sans nom, ou reçoit-on une erreur ? Que renvoie un GET sur une liste de ressources : les ressources ou des pointeurs (URI) sur ces ressources ?).
6. À quoi sert le plus souvent l'API DOM (HTML) dans une fonction de callback en JavaScript après une requête AJAX ?
   À ajouter les résultats de la requête dans la page.
7. Donnez deux raisons pour lesquelles le code suivant ne fonctionnera pas :
   jQuery("#monElementDiv").innerHTML = jQuery.get("http://example.com");
   1) la fonction jQuery renvoie un tableau d'éléments DOM wrappés dans des objets jQuery, alors qu'innerHTML s'applique directement à un élément DOM.
   2) une requête AJAX nécessite une fonction de callback ; on ne peut pas directement en affecter le résultat.

## *Étude de cas (barème : 8 points)*

Suite à la COP21, le Père Noël a décidé de rajouter à son site Web des informations sur le bilan carbone des jouets qu'il livre chaque année aux enfants. Son site actuel est fondé sur une application Web en Java et lui permet de gérer :

- Les produits (cadeaux de tous types)
- Les prospects (enfants ayant envoyé une lettre de demande de cadeau)
- Les demandes (cadeaux demandés, ordre des souhaits)
- Le suivi du stock de cadeaux (choisir les cadeaux à livrer en fonction des demandes et de la disponibilité)

Cette application a été conçue avec Spring Web MVC et comporte donc déjà des composants comme des contrôleurs (produit, prospect, demande…), des beans (POJOs contenant les classes de modèle correspondantes) et les JSP appropriées pour composer les vues HTML. Le Père Noël vous a choisi(e) pour rendre cette application plus dynamique et plus éco-responsable. Pour cela, vous devez la modifier pour permettre (en plus de ce qui existe déjà) de :

- rendre certaines parties de l'application RESTful et exposer les différentes propriétés :
  - des cadeaux : images, lieu de fabrication, coût d'acheminement…
  - des prospects : prénoms, adresse de livraison…

- requêter les images des cadeaux et les afficher dynamiquement (en AJAX) dans les pages,
- requêter, calculer (à l'aide d'une fonction JavaScript `calculCarbone(produit, prospect)` fournie ; cette fonction renvoie une valeur numérique indiquant un nombre de grammes de CO2) et afficher le bilan carbone des cadeaux, en fonction des données disponibles à propos des cadeaux (lieu de fabrication…) et des prospects (lieu de livraison).

## Conception (barème : 5 points)

8. Listez les différents éléments que vous allez rajouter à cette application, en indiquant leur nature : classes Java, pages statiques, scripts, servlets, JSP, beans (précisez le scope)…
   CECI EST UN EXEMPLE DE REPONSE – TOUT AUTRE STRUCTURE SENSEE EST TOUT AUSSI VALIDE.
   Côté serveur :
   Modification des contrôleurs pour prendre en compte les requêtes sur différentes méthodes http et sur les sous-ressources
   Amélioration du modèle (beans) : produit (scope = prototype pour Spring, application en servlet API « classique »), prospect (scope = session)
   Ajout de vues (XML ou JSON) : produit, prospect (-> négociation de contenus)
   Amélioration de la vue (JSP pour HTML) : produit pour appeler les scripts côté client
   Côté client :
   Script qui requête les images et les ajoute à la vue produit
   Script qui requête les données carbone, lance le calcul du bilan carbone et l'ajoute à la vue produit
   Bibliothèque JS directe : métier (jQuery)
9. Décrivez, à l'aide de diagrammes UML appropriés, leurs communications lorsqu'un utilisateur clique sur un lien lui permettant de visualiser un cadeau. Les éléments du SI externes à votre partie de l'application seront modélisés comme des boîtes noires.
   Faire un diagramme de séquence ou de communication. Dans tous les cas, un diagramme dynamique, représentant :
   Côté client : L'utilisateur clique sur un lien →requête synchrone
   Côté serveur : contrôleur Spring →contrôleur délégué →interrogation du modèle (cadeau) →renvoie les infos (nom, description…) du cadeau →retour au contrôleur délégué →composition de la vue côté serveur (JSP)
   Côté client : Affichage de la vue → onload : script de requêtage asynchrone image →requête(s) asynchrone(s) →callback : affichage de l'image
   Côté client : Affichage de la vue → onload : script de requêtage asynchrone carbone : infos carbone du cadeau + infos localisation du prospect → callback : appel de la fonction de calcul une fois toutes les réponses arrivées → affichage du bilan carbone

10. Programmation (barème : 3 points)

11. Écrivez l'ensemble des scripts JavaScript qui gèrent la partie calcul du bilan carbone, depuis la réception de la page principale jusqu'à l'affichage des données ajoutées.
    Cf. ci-dessus pour le dernier script :
    body onload : 1 fonction qui lance 2 scripts de requêtage asynchrones : infos carbone du cadeau + infos carbone du prospect (chaînage des appels ou lancement en parallèle) → callback : appel de la fonction de calcul une fois toutes les réponses arrivées (si appel en parallèle, vérifier que toutes les données des 2 réponses sont disponibles) → affichage du bilan carbone