|  |
| --- |
| Examen POO |
| Enseignants: H.Mhafdhi, S.Refai, S.Mejdoub Date et Durée: : 04/01/2024 pour 1h30'Classes : DSI2 Documents & Calculatrice: Non autorisés |

**NB : Le sujet contient 5 pages. Il vous est demandé d’apporter un soin particulier à la présentation de votre copie.**

**Exercice1 : 4 points**

1. Soit le bout de code suivant.

|  |
| --- |
| **public** **class** MainClass {    **void** readCard(**int** cardNo) **throws** Exception{  System.***out***.println("reading card");  }    **void** checkCard(**int** cardNo) **throws** RuntimeException{ //ligne1  System.***out***.println("cheking card");  }    **public** **static** **void** main(String[] args) {    MainClass c=**new** MainClass();  **int** cardNo=12344;  c.readCard(cardNo); //ligne2  c.checkCard(cardNo); //ligne3  }  } |

Qu’affiche-t-il lors de l’exécution ? Expliquez votre choix.

1. reading card

cheking card

1. erreur de compilation seulement à la ligne 1
2. erreur de compilation seulement à la ligne 2
3. erreur de compilation seulement à la ligne 3
4. erreur de compilation à la ligne 2 et ligne3

|  |
| --- |
|  |

1. Soit le bout de code suivant.

|  |
| --- |
| **public** **class** A {  **public** **void** afficheA()  {  System.***out***.println("A");  }  }  **public** **interface** I {    **public** **void** afficheI();  }  **public** **class** B **extends** A **implements** I{  @Override  **public** **void** afficheI() {  System.***out***.println("B");  }  }  -----------------------------------------  **public** **static** **void** main(String[] args) {  A obj1=**new** B();  I obj2=**new** B();    A s=obj2;  I t=obj1;  t.afficheI();  s.afficheA();  } |

Quel est le résultat de cette exécution ? Expliquez votre choix.

1. AA
2. BA
3. BB

d. Erreur de compilation

**Exercice2 : Etude de cas 16 points**

Notre mission consiste à développer un micro-système de blog, un site web est constitué par la réunion de billets accumulés au fil du temps. Ces billets peuvent être des paragraphes de texte (message), des images ou des vidéos.

Les billets de type image ou vidéo seront dits taggables dans le sens où il sera possible de leur associer des mots clés (tag). Il sera ainsi possible de rechercher tous les billets de ce type qui comporteront un certain ensemble de tags (par exemple, tous les billets comportant les tags geek, informatique, Gaming…). Au contraire, les billets de type message ne pourront être tagués, la recherche se fera alors sur le contenu du message.

Tous les billets sont des éléments publiables implémentant l’interface suivante :

|  |
| --- |
| **import** java.time.LocalDate;  **public** **interface** Publiable {  LocalDate getDatePublication();  String getAuteur();  } |

* + 1. Ecrivez la classe **Billet** qui implémente l’interface Publiable et qui est caractérisée par :
  + La date de publication (type LocalDate)
  + L’auteur du billet (type String).
  + Un constructeur **Billet(LocalDate datePub, String auteur)** qui initialise tous les attributs.
  + La méthode **String toString()** qui retourne une chaîne de caractères caractérisant un Billet.

1. Ecrivez la classe **Message** qui hérite de la classe Billet avec en plus :
   * Le contenu du message (type String)
   * Un constructeur **Message(LocalDate datePub, String auteur, String contenu)** qui initialise tous les attributs.
   * Un accesseur du contenu.

Les Billets taggables sont aussi des billets qui possèdent en plus :

* + LTags : une liste de tags (tableau de String).
  + nbTags : nombre effectif des tags.
  + Un constructeur **BilletTaggable(LocalDate datePub, String auteur, int capacité)** qui initialise tous les attributs.
  + La méthode **String toString()** qui retourne une chaîne de caractères qui contient les propriétés du billet ainsi que la liste des tags.

1. Ecrivez la classe **BilletTaggable** sachant qu’elle implémente l’interface **Taggable** :

|  |
| --- |
| **public** **interface** Taggable {  **void** ajoutTag(String tag);  **void** supprimeTag(String tag);  **int** nombreTags();  **int** rechercheTag(String tag);  } |

* **void ajoutTag(String tag) :** cette méthode servira à ajouter un tag. Elle n’autorisera pas à ajouter deux fois le même tag.
* **void supprimeTag(String tag) :** Cette méthode servira à supprimer un tag.
* **int nombreTags() :** cette méthode retournera le nombre de tags sur un billet.
* **int rechercheTag(String tag):** Cette méthode retournera la position d'un tag donné dans la liste des tags du billet.-1 sinon.

Les billets taggables sont de deux types : soit des **images**, soit des **vidéos**. Chacune de ces classes est caractérisée par son adresse web (**URL**).

|  |
| --- |
| **import** java.time.LocalDate;  **public** **class** Image **extends** BilletTaggable{    **private** String url;  **public** Image(LocalDate datePub, String auteur, **int** capacité, String url) {  **super**(datePub, auteur, capacité);  **this**.url = url;  }  **public** String getUrl() {  **return** url;  }  @Override  **public** String toString() {  **return** "Image ["+ **super**.toString()+ ", url=" + url + "]";  }    } |
| **import** java.time.LocalDate;  **public** **class** Video **extends** BilletTaggable{    **private** String url;  **public** Video(LocalDate datePub, String auteur, **int** capacité, String url) {  **super**(datePub, auteur, capacité);  **this**.url = url;  }  **public** String getUrl() {  **return** url;  }  @Override  **public** String toString() {  **return** "Video [ "+ **super**.toString()+ ", url=" + url + "]";  }  } |

1. Soit la classe **Blog** caractérisée par :
   * Le titre du blog (type String)
   * LBillets : La liste de tous les billets publiés sur le blog (tableau de **Publiable**)
   * nbBillets : nombre effectif des billets publiés sur le blog.
   * Un constructeur **Blog** **(String titre, int capacité)** qui initialise tous les attributs.
   * **void post(Publiable billet)** : Cette méthode permettra de publier un nouveau billet sur le blog. Les vidéos publiés doivent être sécurisés, donc il faut vérifier que l’url utilise le protocole "HTTPS"

Si ce n’est pas le cas, une exception de type **InvalidURLException** est levée.

**NB** : Vous pouvez utiliser la méthode ***myStr.startsWith (lookingForStr)*** définit dans la classe String et qui retourne ***true*** si la chaine myStr commence par la chaine lookingForStr.

* + **int getNombreBilletsTaggables() :**qui permettra de connaitre le nombre de billets publiés sur lesquels on peut ajouter des tags.
  + **Publiable[] RechercheBilletsTaggablesParAuteur(String auteur):** Cette méthode permettra de consulter tous les billets taggables rédigés par un auteur donné.
  + **Publiable getPlusRécentBillet():** permettra d’obtenir le billet le plus récent. On se basera sur la date de publication du billet et non pas sur l’ordre dans lequel les billets ont été ajoutés sur le blog.

**NB** : Pour comparer les dates, vous pouvez utiliser la méthode **Date1.isAfter(date2)** : définit dans la classe LocaDate et qui retourne true si date1 après date2 et false sinon.

* + **Publiable[] RechercheBilletsParContenu(String[] mots)** : Cette méthode permettra de consulter tous les messages dont le corps comporte tous les mots donnés en paramètre.

**NB** : Vous pouvez utiliser la méthode ***myStr.indexOf(lookingForStr)*** définit dans la classe String et qui retourne -1 si la chaine lookingForStr n’est pas contenu dans la chaine myStr.

1. Ecrivez la classe **InvalidURLException**
2. Ecrivez la classe **Blog**