LE GOUVERNEMENT
DU GRAND-DUCHÉ DE LUXEMBOURG
Ministère de l'Éducation nationale,
de l'Enfance et de la Jeunesse

EXAMEN DE FIN D'ÉTUDES SECONDAIRES CLASSIQUES

Sessions 2022

DISCIPLINE	SECTION(S)	ÉPREUVE ÉCRITE	
Sciences de la programmation	CI	*Date de l'épreuve :*	18.05.22
		Durée de l'épreuve :	08:15 - 11:25
		Numéro du candidat :	

BITCOIN MARKET

Partie modélisation (10 points)

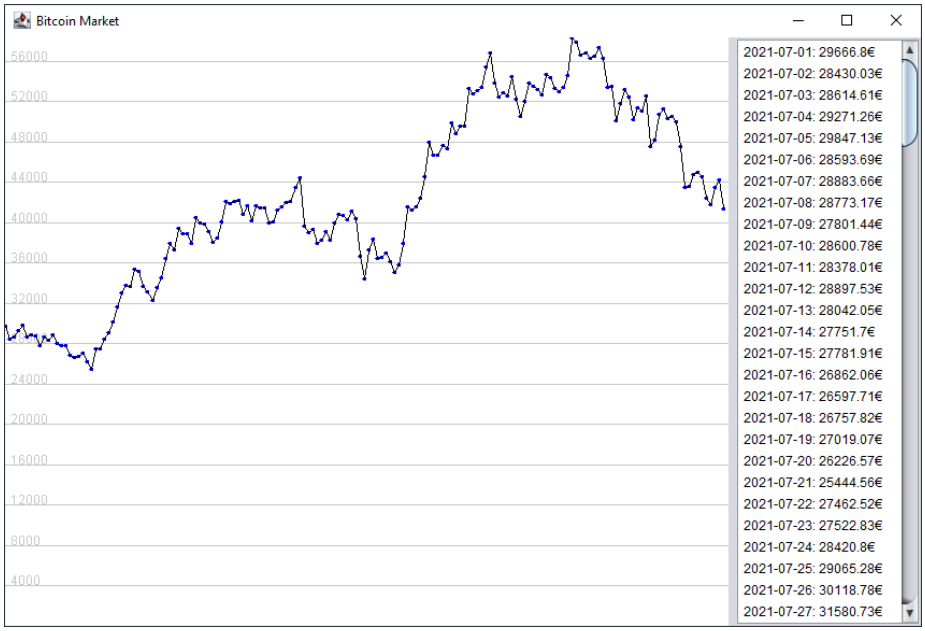
- La partie modélisation doit être rendue avant que la partie implémentation puisse être débutée. Cette partie est à réaliser sur papier.
- La partie modélisation est à rendre après 35 minutes au plus tard, mais vous être libre de la rendre plus tôt.

Vous souhaitez développer une application permettant de représenter sous forme de graphique linéaire l'évolution du marché de la cryptomonnaie « bitcoin ».

Principe de l'application

Vous disposez d'un fichier texte qui contient des dates avec la valeur monétaire (en €) de la cryptomonnaie à cette date. Une ligne de ce fichier texte a le format suivant : *« date »,« valeur en € »*. Les dates sont de type String et se trouvent dans le format ISO 8601 : YYYY-MM-DD. (Exemple : 2022-02-17)

Au lancement, l'application lit toutes les dates et valeurs monétaires depuis le fichier texte. Ces informations sont affichées sous forme de graphique linéaire ainsi que dans une liste.



Travail à réaliser

Réaliser un modèle de classe du **modèle** de l'application sur papier en respectant les conventions UML usuelles.

1. Limitez-vous uniquement aux **attributs** des classes.
2. Rajoutez la/les méthode(s) nécessaire(s) pour trier les informations en ordre chronologique en utilisant la méthode Collections.sort() de Java.
3. Rajoutez la/les méthode(s) nécessaire(s) pour afficher les informations sous forme textuelle dans une liste.
4. Rajoutez la/les méthode(s) nécessaire(s) pour lire les informations depuis un fichier texte.
5. Vous pouvez compléter votre modèle de classe par des explications supplémentaires.

LE GOUVERNEMENT
DU GRAND-DUCHÉ DE LUXEMBOURG
Ministère de l'Éducation nationale,
de l'Enfance et de la Jeunesse

EXAMEN DE FIN D'ÉTUDES SECONDAIRES CLASSIQUES

Sessions 2022

DISCIPLINE	SECTION(S)	ÉPREUVE ÉCRITE	
SCIPR	**CI, CLI**	*Date de l'épreuve :*	
		Durée de l'épreuve :	
		Numéro du candidat :	

Partie implémentation (50 points)

Dans votre répertoire de travail (à définir par chaque lycée), vous trouverez un dossier nommé **EXAMEN_CI**. Renommez ce dossier en remplaçant le nom actuel par votre code de l'examen (exemple de notation : **LXY_CI2_05**). Tous vos fichiers devront être sauvegardés à l'intérieur de ce dossier, qui sera appelé **votre dossier** par la suite.

Ouvrez dans votre dossier le projet **Bitcoin** mis à disposition. Complétez l'application en vous basant sur la version exécutable, le diagramme UML et les instructions données par la suite.

Classe « BTC » (4.5 points)

La classe *abstraite* **BTC** représente la valeur monétaire (en €) de la cryptomonnaie « bitcoin » à une certaine date. La date est de type String et se trouve dans le format ISO 8601 : YYYY-MM-DD. (Exemple : 2022-02-17)

Attributs, constructeur et méthodes : (4.5 points)

- **date** représente la date pour laquelle on dispose de la valeur monétaire.
- **price** représente la valeur monétaire de la cryptomonnaie à cette date.
- **hash** représente une valeur de hachage qui permet de vérifier l'authenticité de la valeur monétaire fournie. Des explications supplémentaires vous seront fournies plus tard.
- **radius** représente le rayon du disque utilisé pour représenter graphiquement la valeur monétaire dans un graphique linéaire. La valeur par défaut est 2.
- Le **constructeur** initialise la valeur des attributs **date**, **price** et **hash**.
- **getDate()** et **getPrice()** sont des accesseurs des attributs respectifs.
- La méthode **toString()** retourne la représentation textuelle. Déterminez le format du texte à partir de l'exécutable fourni.
- La méthode **draw()** est responsable du dessin du disque. Elle prend en paramètre l'abscisse **x** du centre du disque, la hauteur **height** du canevas de dessin et la valeur **pixelPerEuro** utilisée lors de la conversion du prix en pixels. Elle retourne un point avec les coordonnées du centre du disque.
  Exemple : Si le prix est 100€ et pixelPerEuro est 3, alors l'ordonné **y** du centre du disque *dans le repère du graphique linéaire* est : $100€ \times 3^{px}/_{€} = 300px$

Classes « ReliableBTC » et « UnreliableBTC » (4 points)

Les classes **ReliableBTC** et **UnreliableBTC** sont des sous-classes de **BTC** et représentent une valeur monétaire dont l'authenticité a pu être vérifiée et est donc fiable ou n'a pas pu être vérifiée et n'est donc pas fiable.

Une valeur monétaire fiable est représentée comme un disque bleu alors qu'une valeur pas fiable est représentée comme un disque rouge.

Utilisez les fonctionnalités de l'héritage pour réaliser l'implémentation de ces sous-classes.

Classe « Node » (3 points)

La classe **Node** représente un nœud d'une liste chaînée permettant de stocker une valeur monétaire. Implémentez la classe **Node** selon le schéma UML. La classe comprend un constructeur, des accesseurs et manipulateurs, la méthode **toString()** ainsi que la méthode **hasNext()** qui retourne *true* si le nœud possède un successeur et *false* s'il est le dernier nœud de la liste.

Classe « Market » (35 points)

La classe **Market** représente l'évolution du marché de la cryptomonnaie au fil du temps. Les valeurs monétaires sont sauvegardées dans une liste chainée implémentée manuellement. L'attribut **root** représente le premier nœud de cette liste. (0.5 points)

Question au choix I **(2.5 points)**

Alternative 1	La méthode itérative **add()** prend en paramètre une valeur monétaire et l'ajoute à la fin de la liste.
Alternative 2	Les méthodes récursives **add(BTC)** et **add(Node, BTC)** prennent en paramètre une valeur monétaire et l'ajoutent à la fin de la liste.

Question au choix II **(2 points)**

Alternative 1	La méthode itérative **size()** retourne la taille de la liste.
Alternative 2	Les méthodes récursives **size()** et **size(Node)** retournent la taille de la liste.

La méthode **toArray()** retourne un tableau statique contenant les éléments de la liste chaînée. Cette méthode permet d'afficher les valeurs monétaires dans une liste graphique. (3 points)

La méthode privée **max()** utilise la récursivité pour trouver la valeur monétaire la plus grande dans une liste chainée commençant au nœud passé en paramètre. (3 points)

La méthode publique **max()** utilise la méthode privée pour trouver la valeur monétaire la plus grande dans l'intégralité de la liste chainée. Si la liste est vide, la méthode lance une exception du type *NoSuchElementException*. (2 points)

Question au choix III **(5 points)**

La méthode **isValidHash()** prend en paramètre une valeur de hachage et détermine si cette valeur est fiable ou pas. Une valeur de hachage est fiable si la somme des chiffres (de. : die Quersumme) de la valeur de hachage est paire.

Alternative 1	La méthode utilise uniquement des opérations mathématiques sur un nombre entier pour calculer la somme des chiffres.
Alternative 2	La méthode convertit la valeur de hachage en chaine de caractères et utilise des opérations sur les chaines de caractères pour calculer la somme des chiffres.

Exemple : Si la valeur de hachage est « 543 » alors la somme des chiffres est : $5 + 4 + 3 = 12$. Comme cette valeur est paire, on peut faire confiance à la valeur de hachage et donc la méthode retourne *true*.

Recommandation : Comme cette méthode n'est pas cruciale au fonctionnement de l'application, on vous conseille de l'implémenter plus tard et de considérer chaque valeur comme fiable.

La méthode **readMarketFromFile()** lit les valeurs monétaires à différentes dates avec leurs valeurs de hachage et détermine si la valeur est fiable ou non. Une ligne a le format suivant : *« date »,« valeur en € »,« valeur de hachage »*. Les valeurs monétaires se trouvent déjà en ordre chronologique avec la valeur la plus récente se trouvant dans la dernière ligne. Toutes les exceptions doivent être transmises à la méthode appelante. (3.5 points)

La méthode **saveAsJson()** sauvegarde une représentation JSON des valeurs monétaires dans le fichier « bitcoin.json ». Le format à respecter est celui représenté ci-dessous qui est généré par GSON lorsqu'on sauvegarde une structure de données de type *table de hachage*. Toutes les exceptions doivent être transmises à la méthode appelante. (3.5 points)

```
{
  "2021-10-17": 52730.35,
  "2021-10-18": 53090.08,
  "2021-10-19": 53397.48,
  "2021-10-13": 48855.33,
  …
}
```

La méthode **draw()** prend en paramètre les dimensions du canevas de dessin. Elle peut être subdivisé en trois étapes :

1. Elle calcule l'espacement entre deux disques et la valeur pixelPerEuro (cette valeur dépend de la hauteur du canevas de dessin et de la plus grande valeur monétaire contenue dans la liste). Ensuite elle fait appel à la méthode **draw()** des classes-filles de **BTC** pour dessiner les disques du graphique. Elle traite les exceptions produites par la méthode **max()** en affichant le message de l'exception à la console. (4 points)
2. Elle relie deux disques consécutifs entre eux avec des traits noirs. (3 points)
3. Elle dessine des lignes de grille (eng. : major grid lines) en RGB(200, 200, 200) toutes les 4000€ et affiche la valeur de chaque ligne près du bord gauche du canevas. (3 points)

Recommandation : Comme les étapes 2 et 3 ne sont pas cruciales au fonctionnement de l'application, on vous conseille de les implémenter plus tard.

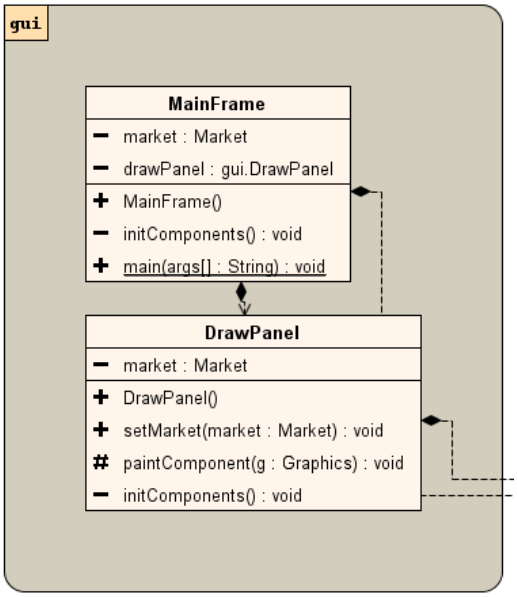
Classe DrawPanel (1 point)

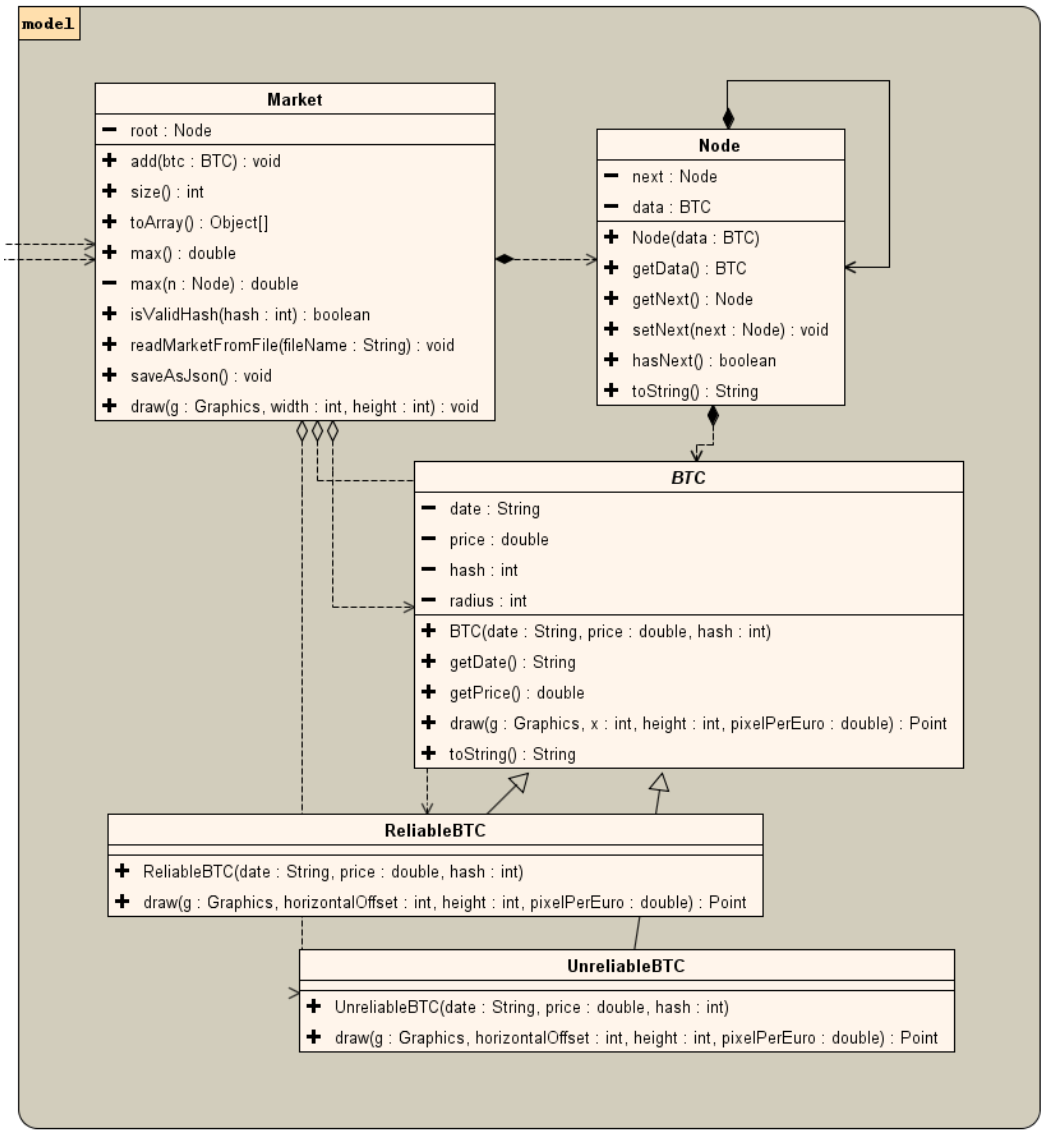
La classe **DrawPanel** (*JPanel*) est responsable de la visualisation du graphique. Rajoutez l'attribut **market** et le manipulateur **setMarket()**. Cette classe contient aussi la méthode **paintComponent()** qui affiche d'abord un fond blanc et ensuite le graphique linéaire.

Classe « MainFrame » (2.5 points)

Implémentez la classe **MainFrame** en vous basant sur le schéma UML et l'exécutable fourni. L'attribut **market** représente le modèle de l'application. Le constructeur lit les informations du marché à partir du fichier « bitcoin.csv » et sauvegarde ces informations sous format JSON dans le fichier « bitcoin.json ». Le texte des exceptions éventuelles est affiché à la console. (2 points)

Schémas UML


gui
MainFrame
- market : Market
- drawPanel : gui.DrawPanel
+ MainFrame()
- initComponents() : void
+ main(args[] : String) : void
DrawPanel
- market : Market
+ DrawPanel()
+ setMarket(market : Market) : void
# paintComponent(g : Graphics) : void
- initComponents() : void


model
Market
- root : Node
+ add(btc : BTC) : void
+ size() : int
+ toArray() : Object[]
+ max() : double
- max(n : Node) : double
+ isValidHash(hash : int) : boolean
+ readMarketFromFile(fileName : String) : void
+ saveAsJson() : void
+ draw(g : Graphics, width : int, height : int) : void
Node
- next : Node
- data : BTC
+ Node(data : BTC)
+ getData() : BTC
+ getNext() : Node
+ setNext(next : Node) : void
+ hasNext() : boolean
+ toString() : String
BTC
- date : String
- price : double
- hash : int
- radius : int
+ BTC(date : String, price : double, hash : int)
+ getDate() : String
+ getPrice() : double
+ draw(g : Graphics, x : int, height : int, pixelPerEuro : double) : Point
+ toString() : String
ReliableBTC
+ ReliableBTC(date : String, price : double, hash : int)
+ draw(g : Graphics, horizontalOffset : int, height : int, pixelPerEuro : double) : Point
UnreliableBTC
+ UnreliableBTC(date : String, price : double, hash : int)
+ draw(g : Graphics, horizontalOffset : int, height : int, pixelPerEuro : double) : Point