

L'architecture orientée objets

Table des matières

I. Architecture	3
II. Exercice : Quiz	5
III. Architecture orientée objet	6
IV. Exercice : Quiz	8
V. Essentiel	9
VI. Auto-évaluation	9
A. Exercice	9
B. Test	9
Solutions des exercices	10

I. Architecture

Durée : 1 h 30

Prérequis : aucun.

Environnement de travail : aucun.

Contexte

L'architecture orientée objets est une architecture logicielle qui met en avant des composants et applications organisés selon la technologie objet. Elle propose de créer un système basé autour de concepts et d'objets existant dans le monde réel, alors que l'approche fonctionnelle classique se base sur des opérations décrivant des méthodes purement fonctionnelles. Cette approche permet une utilisation poussée du code et permet l'encapsulation en modules, comme les catalogues dans lesquels on prendrait des modules pour les réutiliser.

Le démarrage de l'architecture orientée objets se déroule en trois phases. L'analyse consiste à étudier les attentes liées aux utilisateurs et les transformer en objets métiers correspondants à des concepts réels. La phase de conception ou le design qui décrit l'organisation des modules du logiciel, mise en œuvre des règles de gestion et persistance des données, définition de l'interface utilisateur. L'implémentation, qui correspond à la partie développement et qui affine les définitions de la partie design en fabriquant les composants logiciels nécessaires.

Rappel L'histoire de l'architecture

Au début de la décennie de 1990-2000, il y avait un grand nombre de représentants architecturaux distincts. Apparaissant en 1995, le langage model unifié (UML) est devenu une norme internationale (ISO/IEC 19501) spécialement utilisée pour représenter une architecture logicielle. Le développement orienté objet se répand dans l'industrie, les systèmes de gestion de base de données sont maintenant considérés comme un moyen pratique d'assurer l'existence d'objets. Les systèmes de gestion de la base de données objet-relationnels et objets font leur apparition. Nous voyons également des architectures dispersées ; les programmes informatiques ne sont plus considérés comme fournissant des services aux humains, mais également à d'autres programmes. L'émergence des réseaux ouverts, notamment Internet, a radicalement modifié le paysage architectural. Une architecture à trois niveaux centrée sur les données est désormais réalisée par un serveur de base de données, un serveur d'applications Web et de navigateurs Web.

L'étude académique de l'architecture logicielle s'intéresse davantage aux problèmes de couplage entre objets et d'interopérabilité syntaxique et sémantique. Le problème d'appariement est essentiellement considéré comme un problème de communication entre objets, voire de dialogue entre agents intelligents ; une architecture orientée agent émerge. Le principe de réutilisation des composants logiciels s'applique aujourd'hui à l'architecture. Pratiques, principes ou styles architecturaux pouvant être réutilisés.

Cette décennie a été caractérisée par un retour aux bases de données distribuées rendu possible par les technologies XML. XML est un ensemble de règles de représentation et de structuration des données, ce qui est une limitation de SGML. Ces données peuvent être normalisées syntaxiquement et la flexibilité de la technologie permet d'exprimer de nombreuses sémantiques différentes. L'architecture traditionnelle à 3 couches se présente sous la forme de trois couches d'intermédiaires de données : gestion des données XML, transformation et combinaison des données XML et présentation des données XML. La technologie permet de spécifier la syntaxe, la transformation, la présentation et la spécification sémantique. Il existe des bibliothèques de logiciels pour gérer les données XML et la plupart des systèmes de gestion de bases de données prennent actuellement en charge XML.

Une architecture logicielle

L'architecture logicielle décrit les différents éléments d'un ou plusieurs systèmes informatiques, leurs interactions et leurs interactions de manière symbolique et schématique. Contrairement aux spécifications créées par l'analyse fonctionnelle, les modèles architecturaux créés lors de la phase de conception doivent être conçus pour répondre aux spécifications plutôt que d'expliquer ce que le système informatique doit faire. L'analyse explique « *que faire* » et l'architecture explique « *comment le faire* ».

La phase de conception de logiciel est équivalente à la phase de conception d'ingénierie traditionnelle (génie civil, mécanique ou électrique) en informatique. Cette phase consiste en la réalisation complète du produit sous forme abstraite avant la production proprement dite. D'autre part, la nature immatérielle des logiciels (modélisés avec des informations) rend le lien entre l'architecture et le produit plus flou que l'ingénierie traditionnelle. Vous pouvez souligner la relation étroite entre l'architecture et le produit en utilisant l'outil CASE (*Computer Aided Software Engineering*) ou en créant l'architecture à partir du code lui-même et de la documentation du système. Après le produit (le logiciel lui-même), l'architecture logicielle est le plus grand résultat du processus logiciel. En fait, la phase de conception représente environ 40 % du travail de développement total, et en termes de charge de travail, elle doit être au moins aussi importante que la phase de codage. Votre travail dépend fortement du type de logiciel que vous développez et de l'expertise de votre équipe de développeurs. Cette expertise aura un impact sur le taux de réutilisation et sur les performances du processus logiciel.

Les deux objectifs de l'architecture logicielle sont :

1. Réduire les coûts : la réduction des coûts est principalement réalisée par la réutilisation des composants logiciels et par le gain de temps de maintenance (correction d'erreur et adaptation logicielle).
2. Améliorer la qualité du logiciel. La qualité est attribuée sur plusieurs critères. La norme ISO/IEC 25010 est un exemple d'ensemble de critères.

Les architectures faibles ou manquantes peuvent causer de graves problèmes lors de la maintenance de logiciels. En fait, de mauvaises modifications logicielles architecturales peuvent entraîner la dégradation (le principe de l'entropie logicielle). Les architectures informatiques doivent être systématiquement déployées de manière simple, afin de faciliter la réparation et l'extension.

Diminution de la dégradation du logiciel

La gestion du changement est la plus importante des processus d'itération. En fait, les exigences de l'utilisateur peuvent être modifiées. Il est donc implicitement supposé que l'environnement système peut être modifié. Par conséquent, les architectes informatiques sont responsables de prédire la pire situation et de concevoir l'architecture en conséquence.

Développement pour et par la réutilisation

La réutilisation des composants logiciels est l'activité qui permet les économies les plus importantes, proposant toujours des composants à réutiliser. De plus, la réutilisation des composants nécessite la création d'une architecture logicielle permettant l'intégration harmonieuse de ces composants. Par conséquent, le développement utilisant la réutilisation de logiciels nécessite un cycle perpétuel de production-réutilisation et une architecture logicielle standardisée.

La réutilisation rationalisée nécessite la création et la maintenance d'une bibliothèque de logiciels et un changement d'orientation ; créer une application signifie créer les composants de bibliothèque requis, puis créer l'application à l'aide de ces composants. Une telle bibliothèque, qui facilite le développement d'applications, est un cadre d'entreprise et son architecture, ainsi que sa documentation, est le fondement de la réutilisation des logiciels en entreprise.

En conséquence, le rôle de l'architecte informatique évolue vers celui d'un bibliothécaire. Les architectes informatiques doivent explorer la bibliothèque pour trouver les composants logiciels les plus appropriés aux besoins de l'application. Il doit ensuite créer les composants manquants tout en prenant soin de les documenter et de les intégrer dans la bibliothèque. Au sein des grandes entreprises ou des plus gros projets, ce rôle est assuré par l'architecte informatique en chef. Il est responsable du développement continu de la bibliothèque ainsi que du maintien de l'intégrité de son architecture.

Les modèles d'architecture

Indépendamment du format du diagramme architectural, le modèle architectural est considéré comme un point de vue du système. En fait, quelle est la raison de l'utilisation d'un diagramme lorsqu'il est inutile, ou si les choix de la décision de l'architecture sont vagues ou pas explicite ? Pour éviter de formuler la motivation de chaque diagramme, l'architecte crée divers diagrammes basés sur des modèles de conception et réutilise des modèles de conception éprouvés.

Les modèles de conception (ou architectures) sont constitués d'une série d'aspects de leurs différents types de graphiques respectifs. Le modèle architectural propose des moyens pour lier les différentes vues et diagrammes de manière à naviguer facilement, il s'agit des mécanismes de traçabilité architecturale. La traçabilité doit également étendre les spécifications du système et remplir ces spécifications. La devise des fondateurs d'UML est « *Centré sur l'architecture, piloté par les cas d'utilisation et au développement itératif et incrémentiel* ». Cette devise indique clairement que les décisions architecturales ne doivent pas être prises à moins d'être guidées par la mise en œuvre d'une spécification système (cas d'utilisation).

Exercice : Quiz

[solution n°1 p.11]

Question 1

Les deux objectifs principaux de l'architecture logicielle sont de réduire les coûts et d'améliorer la qualité du logiciel.

- ☐ Vrai
- ☐ Faux

Question 2

La réutilisation des composants logiciels est l'activité qui fait le plus de dépenses.

- ☐ Vrai
- ☐ Faux

Question 3

Les modèles de conception (ou architectures) sont constitués d'une série d'aspects de leurs différents types de graphiques respectifs.

- ☐ Vrai
- ☐ Faux

Question 4

L'architecture orientée objets est une architecture logicielle qui met en avant des composants et applications organisés selon la technologie objet.

- ☐ Vrai
- ☐ Faux

Question 5

Le démarrage de l'architecture orientée objets se déroule en deux phases.

- ☐ Vrai
- ☐ Faux

III. Architecture orientée objet

Une architecture orientée objet

La programmation orientée objet est une méthode de programmation informatique de plus en plus populaire, que ce soit en développement logiciel ou en science des données. Organisée autour d'objets ou de données, la programmation orientée objet offre de nombreux avantages.

La Programmation Orientée Objet (POO) est un paradigme informatique qui consiste à définir et à rendre interactifs des objets à l'aide de diverses technologies, notamment des langages de programmation (Python, Java, C++, Ruby, Visual Basic.NET, Simula, etc.). Un objet est un ensemble complexe de variables et de fonctions, comme un bouton ou une fenêtre sur un ordinateur, des personnes (avec des noms, des adresses, etc.), de la musique, des voitures, etc. tout peut être considéré comme un objet. L'objectif de la programmation orientée objet est de se concentrer sur l'objet lui-même et les données, plutôt que sur la logique nécessaire et les actions à entreprendre pour effectuer cette opération.

POO : les diverses étapes

La programmation orientée objet se déroule en plusieurs étapes. La première comprend la modélisation des données en définissant les objets que le programmeur veut manipuler et leurs interactions. Une fois les objets modélisés, ils seront conceptualisés comme une classe d'objets, qui aura des propriétés et des méthodes. Les propriétés correspondent aux propriétés de l'objet et les méthodes aux actions que l'objet peut effectuer. Lorsque l'objet est créé, il communique via une interface par des messages.

En masquant les variables et fonctions d'objet, les programmeurs créent un code source parfois complexe mais facile à utiliser, tout en augmentant la sécurité du système et en évitant les erreurs de données accidentelles. Grâce à ses propriétés d'héritage et de polymorphisme, une classe peut être réutilisée par le programme dans lequel elle a été créée, ainsi que par d'autres programmes orientés objet.

La data science et la POO

Les *data scientists* et autres professionnels de la science des données utilisent beaucoup le langage Python, qui donne accès à de nombreuses bibliothèques. Ces bibliothèques contiennent des modules qui contiennent eux-mêmes des classes. C'est le principe de l'encapsulation. En important ces bibliothèques, le data scientist n'a pas besoin de connaître le code développé dans les classes de la bibliothèque mais il doit comprendre les méthodes et leur logique de fonctionnement pour obtenir ce dont il a besoin.

Les points clé de la POO

Classe : une classe représente un ensemble de code composé de variables et de fonctions qui permettent la création d'objets. Une classe peut contenir plusieurs objets.

Objet : un objet est un bloc de code qui combine des variables et des fonctions, appelées respectivement propriétés et méthodes. Les propriétés définissent les caractéristiques d'un objet d'une classe, les méthodes définissent des fonctions spécifiques pour les instances d'une classe.

Encapsulation : l'encapsulation permet d'enfermer les données brutes dans une enceinte pour éviter la gestion des erreurs ou la corruption des données. Par conséquent, l'encapsulation permet de masquer des méthodes et des propriétés en dehors de la classe

Abstraction : le concept d'abstraction consiste à masquer les détails inutiles aux utilisateurs finaux d'une classe. Ainsi, il pourra utiliser une classe dans son code de programmation sans savoir comment elle a été développée.

Héritage : le concept d'héritage signifie qu'une classe B héritera des mêmes propriétés et méthodes qu'une classe A. Une fois qu'une instance de la classe B est créée, nous pouvons appeler les méthodes contenues dans la classe A par la classe B. Cela économisera du temps aux programmeurs.

Polymorphisme : lorsqu'une classe hérite des méthodes de la superclasse, il est possible de surcharger une méthode, notamment en redéfinissant la méthode de la superclasse afin que les deux classes n'effectuent pas les mêmes tâches.

Une architecture Agile

Les termes MVC (Modèle Vue Contrôleur), base de données, langage de programmation, framework, éditeur, SOA (Architecture Orientée Service), *Service Layer* (*couche de service*) ou REST (REpresentational State Transfer, c'est un style d'architecture) représentent des outils. De la même façon que sur le plan de construction d'une maison, vous ne regardez pas la pelle et la marque du béton qui est utilisée, vous ne voyez pas dans l'architecture d'un logiciel les termes ci-dessus.

L'architecture d'un système est une question d'usage. L'architecture doit indiquer l'objectif du système, et non les outils sur lesquels il sera construit.

Imaginez l'architecture d'une librairie ou d'un centre culturel, vous regardez le schéma et vous voyez des salles de lecture où les lecteurs peuvent tranquillement lire et faire leurs recherches, vous voyez de petits espaces remplis de lumière avec des fauteuils pour se reposer et des salles pleines d'étagères. Que voyez-vous dans l'architecture d'une bibliothèque ? Vous voyez une bibliothèque !

De plus, lorsque nous regardons l'architecture d'application de haut niveau, prenons les applications Web comme exemple, que voyons-nous ? Nous voyons souvent le framework MVC. On voit bien qu'il s'agit d'une application Web. D'autre part, les cas d'utilisation de cette application sont cachés par cette structure.

Qu'y a-t-il de plus important à voir à un niveau élevé pour cette application ? Disons qu'il s'agit d'une application Web de gestion des commandes. Nous verrons ce qu'est la « *Gestion des commandes* » ou la « *Gestion Web* » ? Évidemment, nous voulons voir « *Gestion des commandes* ».

En fait, MVC n'est rien de plus qu'un mécanisme de livraison. Pas plus, pas moins. Le mécanisme de livraison ne doit pas changer l'objectif du système et ne doit donc pas être omniprésent dans l'architecture.

L'objectif est que les cas d'utilisations soient au cœur du système. Lors d'une analyse de l'architecture de haut niveau, nous percevons ces cas d'utilisation ce qui nous permet de comprendre à quoi sert l'application.

Nous voulons séparer nos cas d'utilisation des autres classes du système, de sorte que les autres classes soient considérées comme des « *add-ons* » à notre classe la plus populaire, celle qui expose les cas d'utilisation réels du système.

Séparation de valeur

Vous pouvez imaginer que cela ne s'applique pas aux simples applications CRUD (Create, Read, Update, Delete). Ce qui est généralement plus important serait la couche d'interface utilisateur, qui n'a pas de règles métier significatives. L'interface utilisateur est importante, disons plus importante que les cas d'utilisation, cela signifie qu'elle ne sera pas affectée par les modifications apportées aux autres classes.

Cette division s'applique à toutes les dépendances en général. Cela permet, entre autres, de définir les coûts : en décomposant le système par « *responsabilité* », cela permet aussi aux entreprises de prendre des décisions plus prudentes et moins risquées. Si l'interface utilisateur est vraiment une partie importante du système en cours de développement, il est possible de donner une estimation plus précise, qui pourrait vous faire changer d'avis. Si c'est toujours important pour l'entreprise, c'est clairement communiqué. D'autre part, nous pouvons nous attacher à rendre le développement de cette partie aussi autonome et la classe aussi indépendante que possible. C'est ce que nous appelons la séparation des valeurs.

En tout état de cause, nous voulons que les décisions à propos de l'UI, des bases de données, des serveurs Web, des services, etc. soient détachées de nos cas d'utilisation. De plus, nous voulons que les cas d'utilisation ignorent des mécanismes de livraison.

Une bonne architecture logicielle permet d'optimiser les choses en cas de changement d'avis du client ou de demande de modifications. Cela permet d'effectuer les changements sans affecter les cas d'utilisation du système.

Cas d'utilisation

Un cas d'utilisation représente la façon dont un utilisateur interagit avec le système dans le but d'atteindre un certain objectif. Prenons l'exemple d'un système de gestion de commande.

Exemple

La création d'une commande :

- Liste des données :
 - Id-client
 - Info-contact-client
 - Destination-livraison
 - Méthode-livraison
 - Info-paiement
- Cas normal :
 - L'utilisateur émet un nouvel ordre composé des données ci-dessous,
 - Validation de toutes les données par le système,
 - Création de la commande et détermination du code de la commande par le système,
 - Livraison du code de la commande à l'utilisateur par le système.
- Les cas exceptionnels ou cas étendus :
 - Si erreur de validation : affichage d'un message d'erreur par le système.

Notez qu'il n'y a aucune notion de « *page* », bouton, etc. Tout ce que le cas d'utilisation montre ce sont comment l'interaction se manifeste. Le cas d'utilisation est un algorithme qui interprète des données en entrée et génère d'autres données en sortie. Ce qui implique la création d'un objet qui implémente ces opérations.

Attention, un cas d'utilisation est différent d'une *User Story*. Le cas d'utilisation est beaucoup plus « *large et complet* » qu'une « *user story* ». Un ensemble de user stories évolue en général en un cas d'utilisation.

Exercice : Quiz

[solution n°2 p.12]

Question 1

La programmation orientée objet est une méthode de programmation informatique.

- ☐ Vrai
- ☐ Faux

Question 2

En POO, un objet est un ensemble complexe de variables et de fonctions.

- ☐ Vrai
- ☐ Faux

Question 3

Un cas d'utilisation représente la façon avec laquelle un utilisateur interagit avec le système dans le but d'atteindre un certain objectif.

- ☐ Vrai
- ☐ Faux

Question 4

Un cas d'utilisation peut être apparenté à une User Story.

- ☐ Vrai
- ☐ Faux

Question 5

Il est possible d'importer des bibliothèques de classe pour les utiliser.

- ☐ Vrai
- ☐ Faux

V. Essentiel

La POO est d'abord très stricte, car il faut déclarer toutes les variables, donnant une certaine rigueur au programme. Le code est mieux organisé, plus structuré, mieux hiérarchisé, ce qui le rend plus facile à modifier et à faire évoluer. En effet, le code est plus sécurisé, les programmes sont plus clairs, la maintenance des applications est facilitée.

La POO permet, entre autres, la réutilisation de code, plus précisément de fonctions créées précédemment. Ainsi, cela permet d'éviter de réécrire plusieurs fois les mêmes fonctions dans un même projet, elles peuvent aussi être réutilisées pour d'autres projets (avec ou sans modifications mineures). De nouveaux programmes légèrement différents peuvent être facilement créés en héritant d'un programme existant. La POO permet de développer de gros programmes.

En un mot, la programmation orientée objet est faite pour une programmation plus rapide car les codes sont réutilisables et plus sécurisés. Les langages existants qui utilisent cette approche sont pour les langages compilés (il existe un compilateur qui les traduit en instructions processeur) : C++, C#, et bien d'autres langages. Pour les langages non compilés (ils nécessitent un interpréteur, ils sont censés être plus portables, plus simples, mais perdent en efficacité) : JAVA, PERL 5, 4 PHP et les suivants, et bien d'autres langages.

VI. Auto-évaluation

A. Exercice

Vous intégrez une équipe dans une entreprise et on vous demande d'expliquer la programmation orientée objets.

Question 1

[solution n°3 p.13]

Quels points essentiels citerez-vous ?

Question 2

[solution n°4 p.13]

Développer les points essentiels.

B. Test

Exercice 1 : Quiz

[solution n°5 p.13]

Question 1

Le rôle de l'architecte informatique est d'explorer la bibliothèque pour trouver les composants logiciels.

- ☐ Vrai
- ☐ Faux

Question 2

Une classe peut contenir un seul objet.

- ☐ Vrai
- ☐ Faux

Question 3

Un objet est un bloc de code.

- ☐ Vrai
- ☐ Faux

Question 4

Le code est mieux organisé et plus structuré grâce à la POO.

- ☐ Vrai
- ☐ Faux

Question 5

La programmation orientée objet est faite pour travailler plus rapidement.

- ☐ Vrai
- ☐ Faux


Solutions des exercices

Exercice p. 5 Solution n°1**Question 1**

Les deux objectifs principaux de l'architecture logicielle sont de réduire les coûts et d'améliorer la qualité du logiciel.

☒ Vrai

☐ Faux


 La réduction des coûts et la qualité sont principalement réalisées par la réutilisation des composants logiciels et la diminution des temps de maintenance.

Question 2

La réutilisation des composants logiciels est l'activité qui fait le plus de dépenses.

☐ Vrai

☒ Faux


 La réutilisation des composants logiciels est l'activité qui permet les économies les plus importantes, permettant de réutiliser des composants.

Question 3

Les modèles de conception (ou architectures) sont constitués d'une série d'aspects de leurs différents types de graphiques respectifs.

☒ Vrai

☐ Faux


 Le modèle de conception propose des moyens pour lier les différentes vues et diagrammes de manière à naviguer facilement, il s'agit des mécanismes de traçabilité architecturale.

Question 4

L'architecture orientée objets est une architecture logicielle qui met en avant des composants et applications organisés selon la technologie objet.

☒ Vrai

☐ Faux


 Elle propose de créer un système basé autour de concepts et d'objets existant dans le monde réel. Elle permet aussi une utilisation poussée du code et permet l'encapsulation en modules.

Question 5

Le démarrage de l'architecture orientée objets se déroule en deux phases.

☐ Vrai

☒ Faux

 Le démarrage de l'architecture orientée objets se déroule en trois phases : l'analyse, la conception et l'implémentation.


Exercice p. 8 Solution n°2

Question 1

La programmation orientée objet est une méthode de programmation informatique.

☒ Vrai

☐ Faux


 La Programmation Orientée Objet (POO) consiste à définir et à rendre interactifs des objets à l'aide de diverses technologies, notamment des langages de programmation.

Question 2

En POO, un objet est un ensemble complexe de variables et de fonctions.

☒ Vrai

☐ Faux


 Un objet est un ensemble complexe de variables et de fonctions, comme un bouton ou une fenêtre sur un ordinateur, des personnes, des voitures, etc. tout peut être considéré comme un objet.

Question 3

Un cas d'utilisation représente la façon avec laquelle un utilisateur interagit avec le système dans le but d'atteindre un certain objectif.

☒ Vrai

☐ Faux


 Le cas d'utilisation est un algorithme qui interprète des données en entrée et génère d'autres données en sortie. Il implique la création d'un objet qui implémente ces opérations.

Question 4

Un cas d'utilisation peut être apparenté à une User Story.

☐ Vrai

☒ Faux


 Le cas d'utilisation est beaucoup plus « *large et complet* » qu'une « *user story* ». Un ensemble d'user stories évolue en général en un cas d'utilisation.

Question 5

Il est possible d'importer des bibliothèques de classe pour les utiliser.

☒ Vrai

☐ Faux

 C'est le principe de l'encapsulation. En important des bibliothèques, on n'a pas besoin de connaître le code développé dans les classes de la bibliothèque mais on doit comprendre les méthodes et leur logique pour pouvoir les utiliser.

p. 9 Solution n°3

Les points essentiels à développer sur une programmation orientée objet sont la classe, l'objet en lui-même, l'encapsulation et l'héritage.

p. 9 Solution n°4

Une classe est un ensemble de code qui contient des variables et des fonctions qui permettent la création d'objets.

Un objet est un bloc de code qui combine des variables et des fonctions, appelées respectivement propriétés et méthodes.

L'encapsulation permet d'enfermer les données brutes dans une enceinte pour éviter la gestion des erreurs ou la corruption des données.


L'héritage permet de donner les mêmes propriétés et méthodes à une autre classe.

Exercice p. 9 Solution n°5**Question 1**

Le rôle de l'architecte informatique est d'explorer la bibliothèque pour trouver les composants logiciels.

☒ Vrai

☐ Faux


 L'architecte informatique doit explorer la bibliothèque pour trouver les composants logiciels appropriés puis créer les composants manquants, les documenter et les intégrer à la bibliothèque.

Question 2

Une classe peut contenir un seul objet.

☐ Vrai

☒ Faux


 Une classe est un ensemble de code qui contient des variables et des fonctions qui permettent la création d'objets. Une classe peut contenir plusieurs objets.

Question 3

Un objet est un bloc de code.

☒ Vrai

☐ Faux

 Un objet est un bloc de code qui combine des variables et des fonctions, appelées respectivement propriétés et méthodes.

Question 4

Le code est mieux organisé et plus structuré grâce à la POO.

☒ Vrai

☐ Faux

- Q Le code est mieux organisé, plus structuré, mieux hiérarchisé, ce qui le rend plus facile à modifier et à faire évoluer.

Question 5

La programmation orientée objet est faite pour travailler plus rapidement.

☒ Vrai

☐ Faux

- Q La programmation orientée objet est faite pour une programmation plus rapide car les codes sont réutilisables et plus sécurisés.