

Les méthodes : l'unité contractuelle

Référence d'article

Article complet

<https://b-ziq.com/fr/blog/filemaker-methodes-unite-contractuelle/>

Doctrine fondamentale

- Le comportement n'existe que sous forme de méthodes liées par contrat.
- Une méthode est une unité de comportement liée par contrat, avec une entrée et une sortie déclarées, indépendante du contexte d'exécution.
- Les scripts sont des mécanismes d'implémentation, non des unités comportementales.
- L'identité comportementale est `module.méthode`.
- Chaque méthode appartient à exactement une classe de méthode.
- Une méthode ne peut opérer que sur son entrée déclarée, les données explicitement accédées et sa propre logique interne.
- Une unité qui viole la frontière de pureté n'est pas une méthode dans le modèle architectural.

Définitions

Méthode

Unité de comportement liée par contrat, avec une entrée et une sortie déclarées, indépendante du contexte d'exécution.

Identité de méthode

L'identité logique `module.méthode`.

Classe de méthode

Classification architecturale décrivant ce qu'une méthode fait dans le modèle comportemental.

Frontière de pureté de méthode

Frontière qui limite une méthode à son entrée déclarée, aux données explicitement accédées et à sa logique interne, tout en excluant le contexte implicite et les entrées cachées.

Propriété

Couche logique

- Possède les méthodes.
- Possède le comportement du système.

Responsabilités

Méthode

- Possède le comportement.
- Déclare son entrée.
- Déclare sa sortie.
- Déclare les erreurs qu'elle peut retourner.
- Opère indépendamment du contexte d'exécution.

Contrat

- Déclare les entrées acceptées.
- Déclare les sorties produites.
- Déclare les erreurs possibles.
- Définit la continuité comportementale.

Module

- Sert de frontière de propriété logique.
- Regroupe des méthodes sous une responsabilité cohérente.

Contraintes

- Le comportement ne doit exister que sous forme de méthodes.
- Une méthode doit être liée par contrat.
- Une méthode doit posséder une entrée et une sortie déclarées.
- Une méthode doit être indépendante du contexte d'exécution.
- L'entrée doit être transmise explicitement avec une structure définie avant l'exécution.
- La sortie doit être retournée via un canal de retour déclaré avec une structure définie.
- L'entrée ne doit pas être héritée du contexte.
- L'entrée ne doit pas être lue depuis une variable globale.
- L'entrée ne doit pas être inférée depuis le modèle actif, l'enregistrement actif ou l'ensemble trouvé.
- L'identité d'une méthode doit être `module.méthode`.

- Une méthode doit appartenir à exactement une classe de méthode.
- Une méthode de lecture ne doit pas enregistrer de mutation.
- Une méthode de lecture ne doit ni créer ni détruire de données.
- Une méthode de lecture doit être sans effets de bord.
- Une méthode d'écriture doit posséder la mutation qu'elle effectue.
- La validation doit se produire à l'intérieur d'une méthode d'écriture avant la persistance.
- Une méthode d'écriture doit retourner un résultat structuré déclarant la réussite ou l'échec.
- Les mutations partielles ne doivent pas laisser le système dans un état indéfini.
- Une méthode d'écriture ne doit pas déléguer la validation à l'appelant.
- Une méthode d'interface ne doit contenir aucune règle métier.
- Une méthode d'interface doit exposer un comportement d'interface via le même modèle de routage et de contrat que les autres méthodes.
- Une méthode ne peut opérer que sur son entrée déclarée, les données explicitement accédées par des requêtes pilotées par paramètres et sa propre logique interne.
- Une méthode ne doit pas dépendre du modèle actif.
- Une méthode ne doit pas dépendre de l'enregistrement actif.
- Une méthode ne doit pas dépendre de l'état ambiant de l'ensemble trouvé.
- Une méthode ne doit pas dépendre de l'état de fenêtre.
- Une méthode ne doit pas dépendre des variables globales.
- Une méthode ne doit pas dépendre de l'état de l'interface.
- Une méthode ne doit pas dépendre du contexte de navigation.

Invariants

- Les méthodes sont les unités comportementales de l'architecture.
- Les scripts sont des mécanismes d'implémentation.
- L'identité comportementale est indépendante de l'identité d'implémentation.
- L'identité logique d'une méthode est `module.méthode`.
- La classe d'une méthode est déterminée par ce qu'elle fait.
- Une méthode n'opère que sur son entrée déclarée, les données explicitement accédées et sa logique interne.
- Les entrées cachées se trouvent hors de la frontière de pureté.
- La propriété du comportement appartient aux méthodes.

Validation

Contrat de méthode

- Vérifier l'existence d'une entrée déclarée.
- Vérifier l'existence d'une sortie déclarée.
- Vérifier l'existence d'erreurs déclarées.
- Vérifier l'indépendance vis-à-vis du contexte d'exécution.

Identité de méthode

- Vérifier l'existence d'une identité logique stable sous la forme `module.méthode`.
- Distinguer identité d'implémentation, identité logique et continuité contractuelle.

Classe de méthode

- Lecture → aucune mutation, aucun effet de bord.
- Écriture → possède la mutation et la validation.
- Interface → publie une capacité d'interface sans règle métier.

Frontière de pureté

- Vérifier que la méthode dépend uniquement :
 - de son entrée déclarée
 - des données explicitement accédées
 - de sa logique interne
- Vérifier l'absence de dépendance au :
 - modèle actif
 - enregistrement actif
 - ensemble trouvé
 - état de fenêtre
 - variables globales
 - état de l'interface
 - contexte de navigation

Conséquences des violations

- L'unité devient dépendante du contexte.
- L'unité ne satisfait plus aux exigences d'une méthode.
- L'unité n'est pas reconnue comme comportement par le modèle architectural.
- L'unité demeure un artefact d'implémentation.
- Le comportement ne peut pas être identifié par une identité stable.
- Le comportement ne peut pas être prédit à partir de l'entrée déclarée seule.
- Le comportement ne peut pas être gouverné par contrat.
- Le comportement devient dépendant d'entrées cachées.
- L'architecture ne peut pas raisonner sur l'unité comme comportement.
- L'unité devient invisible architecturalement.