

R3.03 – Analyse

R3.02 – Développement efficace

Jean-Philippe Prost
`Jean-Philippe.Prost@univ-amu.fr`

Aix-Marseille Université – IUT d'Aix

Préambule

- Les 2 ressources R3.03 Analyse et R3.02 Dév. efficace sont regroupées (R3.03-02 A-DE), mais 2 parties à l'examen, avec 2 notes
- Très peu de CM : 2h + 1h30 + 1h30
- 1 note de TP (coef. 0,5) + 1 note d'examen (coef. 0,5)
- Tous les TD et TP sont en salles machines
- TP en groupes entiers
- Point d'entrée des supports (CM, sujets) : cours sur AMETICE
- Les outils utilisés, pour tout le monde :
 - ▶ starUML
 - ▶ eclipse
 - ▶ gitHub
 - ▶ ametice

Plan

- 1 Introduction
- 2 Écointformatique : écoconception de logiciels et services numériques
- 3 Analyse des exigences
- 4 Modélisation objet
- 5 Structures de données
- 6 Java efficace (d'après *Effective Java*, Joshua Bloch)
- 7 Écoconception web : les 115 bonnes pratiques

Section 2

Introduction

Introduction

D'après le Programme National (PN) ...

Objectif R3.03 Analyse

Conforter les capacités d'analyse de l'informaticien, en étant capable de **comprendre les exigences d'un client et de les formaliser**.

(R3.03) Savoirs de référence étudiés

- **Analyse des exigences** (par ex. : recueil des besoins métier, des acteurs, cas d'utilisation, scénarios, spécification par exemple...)
- Renforcement de la **modélisation objet** pour l'analyse et le développement

(R3.03) Apprentissages critiques ciblés

- AC21.01 | Élaborer et implémenter les spécifications fonctionnelles et non fonctionnelles à partir des exigences
- AC22.04 | Évaluer l'impact environnemental et sociétal des solutions proposées
- AC25.02 | Formaliser les besoins du client et de l'utilisateur
- AC25.03 | Identifier les critères de faisabilité d'un projet informatique

(R3.03) Mots clés

Analyse des exigences Cas d'utilisation Scénarios Spécification

Objectif R3.02 Dév. efficace

Renforcer l'apprentissage de **l'algorithme** afin d'amener vers une **efficacité de développement**.

(R3.02) Savoirs de référence étudiés

- Développement de **structures de données complexes** (par ex. : collections, arbres, dictionnaires...)
- Premières approches de l'analyse de la performance (profiling, optimisation, greencode...)

(R3.02) Apprentissages critiques ciblés

- AC21.03 | Adopter de bonnes pratiques de conception et de programmation
- AC22.01 | Choisir des structures de données complexes adaptées au problème
- AC22.02 | Utiliser des techniques algorithmiques adaptées pour des problèmes complexes (par ex. recherche opérationnelle, méthodes arborescentes, optimisation globale, intelligence artificielle...)
- AC22.04 | Évaluer l'impact environnemental et sociétal des solutions proposées

(R3.02) Mots clés

Structure de données Performance

Plan

- 1 Introduction
- 2 Écointformatique : écoconception de logiciels et services numériques
- 3 Analyse des exigences
- 4 Modélisation objet
- 5 Structures de données
- 6 Java efficace (d'après *Effective Java*, Joshua Bloch)
- 7 Écoconception web : les 115 bonnes pratiques

Section 3

Écoinformatique : écoconception de logiciels et services numériques

Écoinformatique : écoconception de logiciels et services numériques

Contenu de la séance

- Références
- Quels sont les impacts environnementaux du numérique ?
- Analyse du Cycle de Vie (ACV)
- Écoconception

Références bibliographiques et webographiques

- Bordage, Frédéric, 2022. Écoconception web : les 115 bonnes pratiques. 4ème édition. Éd. Eyrolles.
- Les 115 fiches en version numérique (<https://github.com/cnumr/best-practices>)
- Référentiel de conception responsable de services numériques (<https://gr491.isit-europe.org/>), également détaillé sur <https://ecoresponsable.numerique.gouv.fr>

Acteurs de référence

- Collectif Conception Numérique Responsable (<https://collectif.greenit.fr/>)

Outils de référence

- Ecoindex (<https://www.ecoindex.fr>). Extensions navigateurs firefox/chrome :
 - ▶ GreenIT-Analysist : <https://github.com/didierfred/GreenIT-Analysis>
 - ▶ Website footprint : <https://www.ecoindex.fr>
- Rmq : ces 2 extensions calculent un ÉcoIndex selon la même formule, mais les résultats sont souvent différents...
- <http://www.websitecarbon.com/>

Lecture recommandée

Guillaume Pitron, 2023. L'enfer numérique. Voyage au bout d'un like. Éd. Les liens qui libèrent.

Le monde virtuel est-il vraiment virtuel ?

En y regardant de plus près. . .

- Chaque octet se **matérialise** dans le monde réel, et a donc un **impact** sur lui
 - ▶ consommation d'énergie
 - ▶ consommation de ressources physiques (métaux, silice, eau, etc.)
 - ▶ émission de gaz à effets de serre (eGES), déchets matériels
 - Matériel et logiciel ont un **cycle de vie**, qui impacte l'environnement depuis leur création jusqu'à leur destruction
-
- Internet, web, cloud : quel matériel pour implanter la dématérialisation ?
 - phénomène d'*obésiciel*
 - empreinte carbone du numérique mondiale $\simeq 4\%$ des eGES dans le monde, soit **x 1,5 l'empreinte du transport aérien mondial (GIEC 2022) !**
 - prévision du GIEC : à l'horizon 2030, le numérique devrait représenter 20% de la consommation d'énergie mondiale
 - etc.

Le monde virtuel est-il vraiment virtuel ?

Problème

Étant donné un **ensemble fini de ressources**, combien de temps peut-on appliquer une fonction croissante de consommation de ces ressources ?

Développement du numérique dans le temps

Quiz

- Combien de transistors dans un microprocesseur ?

1971 : 2300

2017 : 19+ milliards

Facteur $\sim 10^7$, soit 10 millions de fois plus !!

- Combien de personnes connectées ?

2010 : 2 milliards (pop. : 7 milliards)

2020 : 5 milliards (pop. : env. 7,8 milliards)

2025 : 6 milliards (pop. : env. 8,5 milliards)

- Combien d'objets connectés ?

2010 : 1 milliard

2025 : 75 milliards

- Combien pèse une page web ?

1995 : 14 Ko

2022 : 2200 Ko, soit 155 fois plus

- Réserve-t-on son billet de train 155 fois plus vite ?
- Écrit-on un email 155 fois plus vite ?
- Lit-on un article 155 fois plus vite ?

⇒ **inflation non justifiée**

Impact environnemental du numérique

Le phénomène *obésiciel* principal levier de l'obsolescence programmée du numérique

- services en lignes "trop gras" nous obligent à changer régulièrement de smartphone, et ordinateur
- appareils parfaitement fonctionnels, mais plus assez puissants pour afficher contenu web et mobile
- pages web et mobile toujours plus lourdes et mal conçues

Comment réduire l'empreinte environnementale de ces sites et services ?

Commencer par identifier les principales sources d'impact

L'empreinte environnementale du web

Selon l'étude de GreenIT.fr, l'empreinte annuelle mondiale du numérique connecté (épuiement des ressources naturelles non renouvelables, dérèglement climatique, érosion de la biodiversité, etc.) est de l'ordre de :

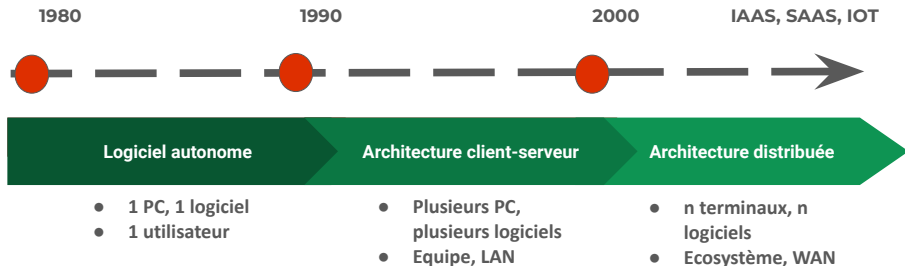
- **1500 millions de tonnes éq. CO₂**, soit env. 150 millions de français (2 fois la France)
- **7,8 milliards de m³ d'eau**, soit 145 millions de français (2 fois la France)

À l'échelle de chaque Européen, le numérique représente :

- **39% de notre budget annuel soutenable** en termes d'émissions de *gaz à effet de serre* (soit 1,7 T éq. CO₂ pour rester sous le réchauffement global de +1,5°C)
- **40% de notre budget annuel soutenable** en termes de *consommation de ressources*, répartis en :
 - ▶ 23% pour les matériaux
 - ▶ 17% pour les énergies fossiles

Ces chiffres sont **10 fois trop élevés !**

Evolution des services numériques



L'empreinte technique

Les éditeurs de services numériques sont les plus à même de lutter contre le phénomène d'obsolescence programmée en proposant des contenus et des services en ligne nécessitant peu de ressources (mémoire vive, bande passante, etc.) pour fonctionner.

Réduire cette "empreinte technique" aide les internautes à conserver plus longtemps leurs équipements, et les centres informatiques à pérenniser leurs serveurs.

- Le plus gros des économies d'énergies sera induit par cet allongement de durée de vie
- Les économies sur la phase d'utilisation ne constituent pas un objectif prioritaire : c'est une externalité positive

L'écoconception à la rescousse

Répartition des impacts et de l'empreinte technique

4 éléments au rôle prépondérant :

- **le type de terminal utilisé** : ordi fixe, portable, tablette, smartphone, console de jeux, etc. et **la taille de l'écran associé**
- **la durée de vie de ce terminal** (et dans une moindre mesure des serveurs)
- **le temps passé par l'internaute** sur un site/service en ligne
- **le type de connexion** : filaire ou mobile

Pour réduire l'empreinte d'un service,...

... nous allons jouer sur ces 4 leviers.

L'objectif est de fournir un service qui ...

- nécessite **la plus petite configuration requise** côté internaute
- monopolise **le moins longtemps possible** le réseau et les serveurs
- nécessite **le moins de serveurs possible**.

L'écoconception à la rescousse

L'idée de fond est de réduire :

- la puissance informatique nécessaire
- donc la quantité de traitements et de données tout au long de la chaîne applicative (dont la bande passante)
- le temps passé par l'internaute devant son terminal.

Écoconcevoir un service en ligne consiste, à niveau de qualité constant, à réduire la quantité de moyens informatiques et télécoms nécessaires, c'est-à-dire son empreinte matérielle.

Pour cela, intervenir à **chaque étape** du cycle de vie du service :

- expression du besoin
- conception fonctionnelle
- maquettage
- conception graphique
- conception technique
- réalisation (développement, intégration, etc.)
- hébergement
- maintenance évolutive et corrective.

L'écoconception à la rescousse

Il est donc essentiel de comprendre que **le travail de conception réalisé en amont aura un impact bien supérieur à l'optimisation minutieuse des lignes de code.**

Les 3 principes de l'écoconception numérique (**sobriété numérique**)

Simplicité : *chaque besoin exprimé est couvert par un ensemble cohérent de fonctionnalités regroupées dans une seule interface homogène.*

Démarche **qualitative**. En gros, *éviter les usines à gaz* ! La simplicité de l'IHM réduit l'effort pour utiliser le service (rejoint les démarches d'*expérience utilisateur (UX)*).

Frugalité : *limiter la couverture et la profondeur fonctionnelles à leur strict minimum.*

Démarche **quantitative**. Par ex., nombre d'éléments d'une liste, taux de compression d'image, etc.

Pertinence : *pertinence = utilité x rapidité x accessibilité*]. Une faiblesse sur l'un des 3 facteurs conduit généralement l'utilisateur à changer de service, et donc à 100 % d'impacts environnementaux inutiles.

Par ex., un résultat de recherche utile mais trop long à obtenir, ou inversement un résultat obtenu très rapidement mais inutile, poussent tous les 2 à aller voir ailleurs.

Plan

- 1 Introduction
- 2 Écointformatique : écoconception de logiciels et services numériques
- 3 Analyse des exigences**
- 4 Modélisation objet
- 5 Structures de données
- 6 Java efficace (d'après *Effective Java*, Joshua Bloch)
- 7 Écoconception web : les 115 bonnes pratiques

Section 4

Analyse des exigences

Sommerville, Ian. 2007. *Software Engineering 8*, Pearson Education Ltd. ISBN 978-0-321-31379-9

- Part 1, Overview (*Généralités*)
 - ▶ Ch. 4 – Software processes
- Part 2, Requirements (*Exigences/Besoins*)
 - ▶ Ch. 6 – Software requirements
 - ▶ Ch. 7 – Requirements engineering processes
 - ▶ Ch. 8 – System models
 - ▶ (Ch. 10 – Formal specification)
- Part 3, Design (*Conception*)
 - ▶ Ch. 14 – Object Oriented design (*conception Orientée Objet*)

Analyse des exigences (besoins)

Besoins/exigences logiciel(le)s du système cible

Référence

Sommerville, ch. 6 – Software requirements

Définition : exigences logicielles

On appelle **exigences** logicielles la description des propriétés et fonctionnalités du système ou service cible. Ces exigences sont le reflet des **besoins** de l'utilisateur.

Définition : ingénierie des exigences / *Requirement Engineering (RE)*

Le processus par lequel les exigences du système sont collectées auprès de l'utilisateur, analysées, puis présentées dans un document de synthèse appelé **Spécifications des exigences logicielles** (*System Requirements Specification (SRS)*).

Exigences fonctionnelles et non-fonctionnelles

Exigences fonctionnelles

Énoncé

- des services que le système doit fournir,
- de comment le système doit réagir à des entrées particulières, et
- de comment le système doit se comporter dans des situations particulières

Exigences non-fonctionnelles

Ensemble des contraintes qui s'exercent sur les services ou fonctionnalités offerts par le système, telles que les contraintes de temps de réponse, de processus de développement, de standards, etc.

Exigences fonctionnelles

- Décrivent les fonctionnalités ou services fournis par le système
- dépendent du type de logiciel, des utilisateurs attendus
- les exigences fonctionnelles *utilisateurs* peuvent être des énoncés de haut niveau, tandis que
- les exigences fonctionnelles *système* doivent décrire les services du système dans le détail

Exigences fonctionnelles

Exemple : le système BIBSYS

Exigences fonctionnelles *utilisateur*

- Un système de bibliothèque qui fournit une interface unique pour nombre de bases de données d'articles dans différentes bibliothèques.
- Les utilisateurs peuvent chercher, télécharger et imprimer ces articles à des fins personnelles.

Exigences fonctionnelles *système*

- L'utilisateur doit pouvoir chercher soit l'ensemble des BDD, soit un sous-ensemble d'entre elles qu'il aura pré-sélectionné.
- Le système doit fournir les applications de visionnage adaptées à une lecture par l'utilisateur des documents dans le stock de documents.
- Chaque commande doit se voir allouer un identifiant unique (ORDER_ID) que l'utilisateur doit pouvoir copier dans la zone de stockage permanent du compte.

Imprécision des exigences

- Des problèmes apparaissent quand les exigences ne sont pas énoncées suffisamment précisément.
- Des exigences ambiguës pourraient être interprétées de façons différentes par les développeurs et les utilisateurs.

Considérer, par exemple, l'expression "applications de visionnage adaptées" (visionneuses) :

- **intention de l'utilisateur** : une visionneuse spécifique pour chaque type de document différent
- **interprétation du dév.** : fournir une visionneuse texte qui montre le contenu d'un document.

Complétude et consistance logiques des exigences

En principe, des exigences doivent être à la fois logiquement **consistantes** et **complètes**.

Complètes Elles devraient couvrir **toutes** les fonctionnalités requises

Consistantes Elles ne devraient comporter **ni conflit ni contradiction** entre les propriétés du système.

En pratique. . .

. . . produire un document complet et consistant est impossible.

Exigences non-fonctionnelles

- Définissent les propriétés du système
 - ▶ Fiabilité
 - ▶ Temps de réponse
 - ▶ exigences stockage
 - ▶ etc.
- Définissent les contraintes du système
 - ▶ capacités d'E/S des périphériques
 - ▶ représentations système
 - ▶ etc.
- Des exigences peuvent également porter sur sur les processus d'analyse, conception, dév.
 - ▶ utilisation d'un AGL
 - ▶ langage de prog.
 - ▶ méthode de dév.

Ces exigences peuvent être plus critiques que les exigences fonctionnelles : lorsqu'elles ne sont pas satisfaites, le système peut devenir inutile.

Mesures d'exigences

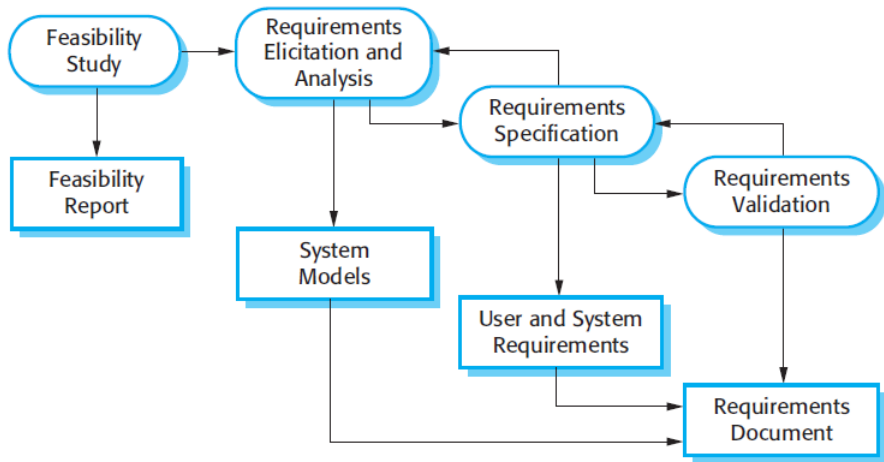
Propriété	Mesure
Vitesse	Nb de transactions exécutées / sec. temps de réponse d'un événement / utilisateur temps de rafraîchissement de l'écran
Taille	M Octets Nb de puces ROM
Facilité d'utilisation	temps de formation Nb d'écrans d'aide
Fiabilité	Temps moyen d'échec Probabilité de non-disponibilité Taux d'échec Disponibilité
Robustesse	Temps de redémarrage après échec Pourcentage d'événements cause d'échec Probabilité de corruption de données en cas d'échec
Portabilité	Pourcentage d'exigences dépendantes de la cible Nombre de systèmes cibles

Référence

Sommerville, ch. 7 – Requirements engineering processes

- Si les **études de faisabilité** sont concluantes, le processus d'ingénierie des exigences (IE) peut commencer
- Les processus d'IE varient largement selon les domaines d'application, les personnes impliquées et les organisations qui développent les exigences.
- Cependant, un certain nombre d'activités génériques sont communes à tous les processus :
 - ▶ élicitation des exigences
 - ▶ analyse des exigences
 - ▶ validation des exigences
 - ▶ gestion des exigences

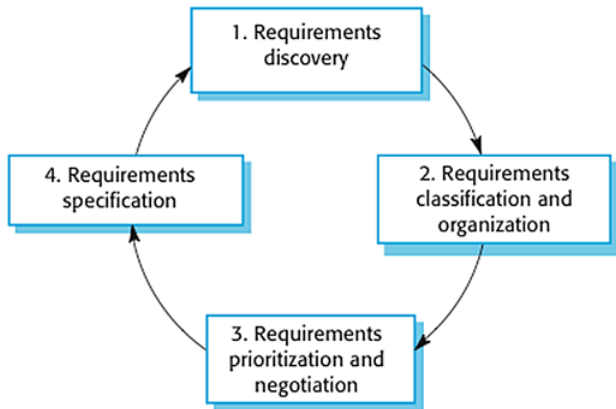
Processus d'ingénierie des exigences



Éliciter \simeq recueillir, obtenir, susciter, découvrir

- étape d'élicitation parfois appelée "étape de *découverte* des exigences"
- implique que l'équipe technique travaille avec le client pour découvrir le domaine d'application, les services attendus du système, et les contraintes opérationnelles.
- Peut impliquer les utilisateurs finaux, les managers, les ingénieurs de maintenance, les experts du domaine, les syndicats, etc. Ils sont **les acteurs**, ou **parties prenantes (stakeholders)**.

Processus d'élicitation et analyse des exigences



- Clients de la banque
- Clients d'autres banques
- Managers de la banque
- Personnel du guichet
- Administrateurs de BDD
- Gestionnaires de sécurité
- Département Marketing
- Ingénieurs de maintenance matérielle et logicielles
- Régulateurs bancaires

- Les scenarios sont des descriptions de comment est utilisé le système dans la vie réelle.
- Ils devraient comprendre :
 - ▶ une description de la situation de départ
 - ▶ une description du flux normal des événements
 - ▶ une description de ce qui peut mal se passer
 - ▶ des informations sur les activités parallèles
 - ▶ une description de l'état final.

Exemple de scénario

Le scénario BIBSYS

Conditions initiales : l'utilisateur est connecté au système BIBSYS et a trouvé l'emplacement du journal qui contient l'exemplaire de l'article.

Cas normal : l'utilisateur sélectionne l'article à copier. Il ou elle se voit demander par le système, soit de fournir des informations de souscription au journal, ou d'indiquer comment ils vont payer pour l'article. Les moyens de paiement possibles sont la carte de crédit, ou la facturation d'un numéro de compte organisationnel.

L'utilisateur se voit demander de remplir un formulaire de copyright qui enregistre les détails de la transaction, qu'ils soumettent ensuite au système BIBSYS.

Le formulaire de copyright est vérifié, et s'il est validé, la version pdf de l'article est téléchargée vers la zone de travail de BIBSYS sur l'ordinateur de l'utilisateur, et l'utilisateur est informé que l'article est disponible. L'utilisateur se voit demander de sélectionner une imprimante, et un exemplaire de l'article est imprimé. Si l'article est étiqueté "impression seulement" il est effacé de l'ordinateur de l'utilisateur une fois que l'utilisateur a confirmé que l'impression est terminée.

Exemple de scénario

Le scénario BIBSYS

Cas anormal : l'utilisateur peut échouer à remplir le formulaire de copyright correctement. Dans ce cas le formulaire doit être présenté à nouveau l'utilisateur pour correction. Si le formulaire nouvellement soumis est toujours incorrect alors la requête de l'utilisateur pour cet article est rejetée.

Le paiement peut être rejeté par le système. Dans ce cas la requête de l'utilisateur pour cet article est rejetée.

Le téléchargement de l'article peut échouer. Réessayer jusqu'au succès, ou jusqu'à ce que l'utilisateur termine la session.

Il peut arriver que l'impression soit impossible. Si l'article n'est pas étiqueté "impression seulement" alors il est conservé dans l'espace de travail de BIBSYS. Sinon l'article est effacé, et le compte utilisateur est crédité du prix de l'article.

Autres activités : téléchargements simultanés d'autres articles.

État final du système : l'utilisateur est connecté. L'article téléchargé a été effacé de l'espace de travail BIBSYS s'il était étiqueté "impression seulement".

Cas d'utilisation (*use cases*)

- Les cas d'utilisation sont une technique à base de scénario en UML, qui identifie les acteurs impliqués dans une interaction, et qui décrit l'interaction elle-même.
- Un ensemble de cas d'utilisation doit décrire toutes les interactions du système.
- Les diagrammes de séquence peuvent être utilisés pour détailler les cas d'utilisation en montrant la séquence de traitement des événements par le système.

Diagramme de cas d'utilisation

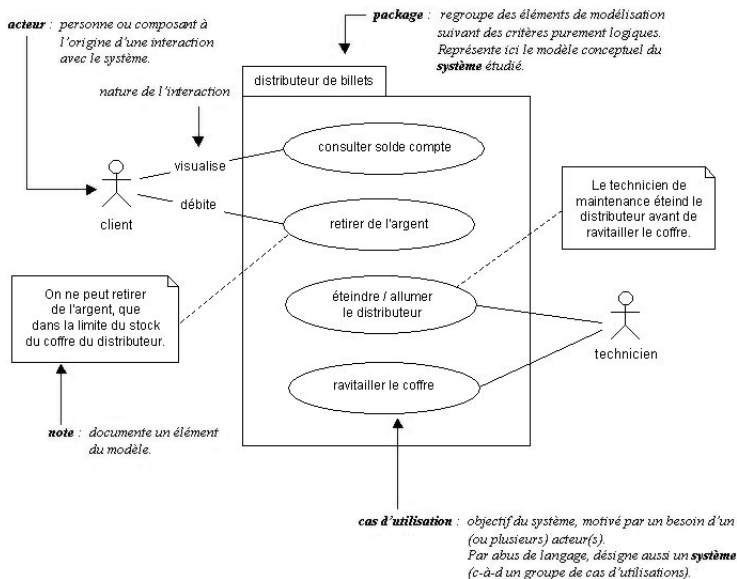


Diagramme de séquence

Exemple

Cas d'utilisation :

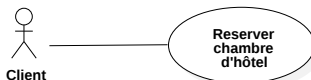


Diagramme de séquence correspondant :

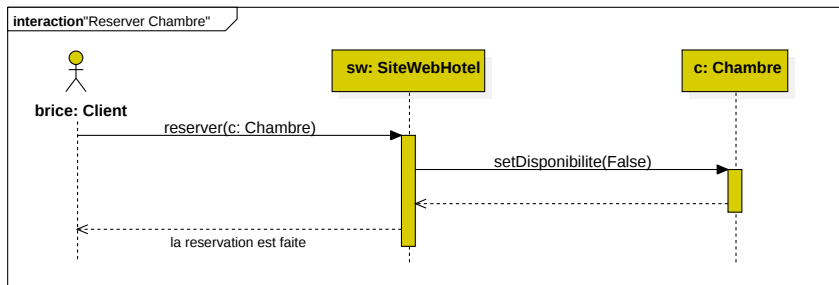
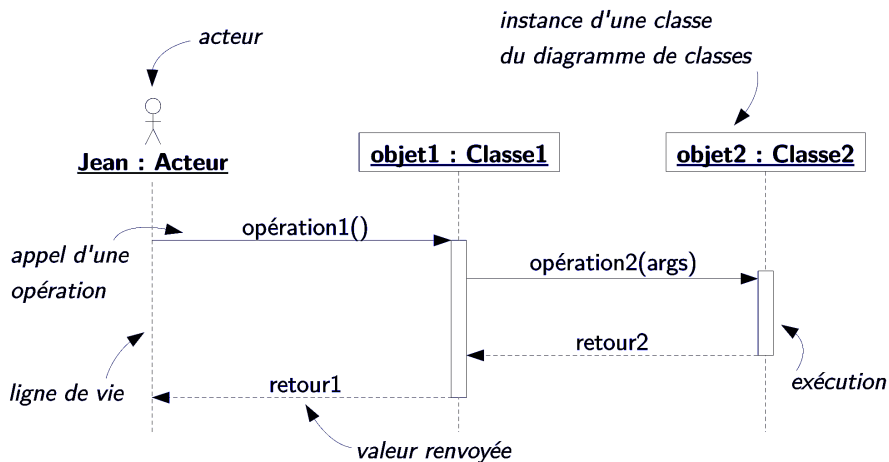


Diagramme de séquence

Éléments de base



Plan

- 1 Introduction
- 2 Écoinformatique : écoconception de logiciels et services numériques
- 3 Analyse des exigences
- 4 Modélisation objet**
- 5 Structures de données
- 6 Java efficace (d'après *Effective Java*, Joshua Bloch)
- 7 Écoconception web : les 115 bonnes pratiques

Section 5

Modélisation objet

Référence

Sommerville, § 8.4

Il existe plusieurs types de **modélisations** possibles pour décrire un système durant la phase d'analyse :

- modélisations contextuelles
- modélisations comportementales
- modélisations (orientées) données
- modélisations (orientées) objet

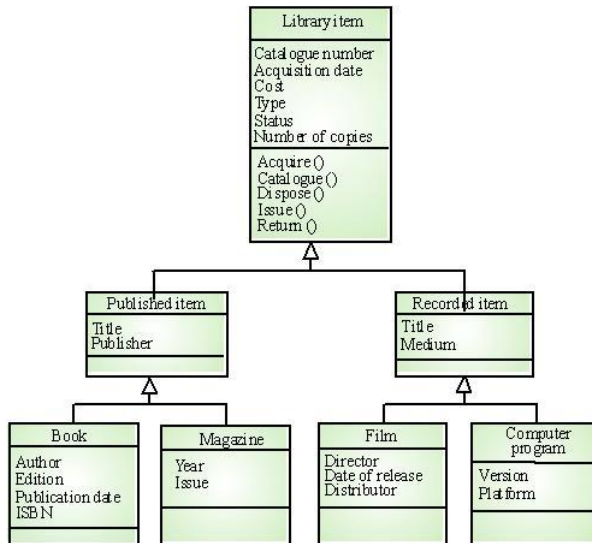
Modélisations objet

- Les modélisations objet décrivent le système en termes de **classes d'objets** et de leurs associations.
- Une classe d'objets est une **abstraction** d'un ensemble d'objets qui ont en commun des attributs (propriétés) et des services (opérations) fournis par chaque objet.
- différents modèles objet peuvent produire :
 - ▶ des modélisations d'héritage
 - ▶ des modélisations d'agrégation
 - ▶ des modélisations d'interaction.
- Façons naturelles de refléter les entités du monde réel manipulées par le système
- Les entités plus abstraites sont plus difficiles à modéliser avec cette approche
- Il est reconnu que **l'identification de classes d'objets est un processus difficile**, qui nécessite une connaissance profonde du domaine d'application.
- Les classes d'objets qui reflètent les entités du domaine sont réutilisables dans différents systèmes

- Elles organisent les classes d'objets du domaine en **hiérarchies**
- Les classes au sommet de la hiérarchie reflètent les propriétés communes à toutes les classes
- Les classes d'objets héritent leurs attributs et services d'une ou plusieurs **classes de base** (*super-classes*). Elles peuvent alors être **spécialisées** comme nécessaire.
- La conception d'une hiérarchie de classes peut être un processus difficile **si la duplication dans des branches différentes doit être évitée.**

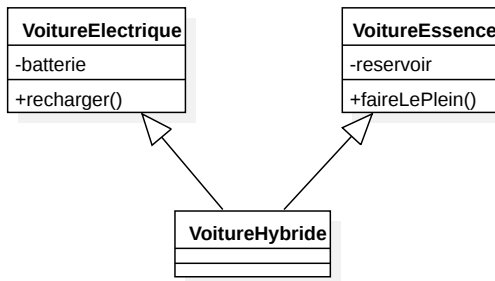
- UML est une représentation standard conçue par les développeurs de méthodes d'analyse et de conception orientée objet largement répandue.
- Cette représentation est devenue de fait un standard pour la modélisation orientée objet.
- Notation
 - ▶ Les classes d'objet sont des **rectangles** avec leur nom en haut, les attributs au milieu, et les opérations en bas
 - ▶ Les relations entre classes d'objets (appelées *associations*) sont représentées par des lignes entre les objets
 - ▶ L'héritage est appelé **généralisation**, et se représente avec une flèche "vers le haut", i.e. de la classe la plus spécialisée (classe fille) vers la classe de base (classe mère)

Hiérarchie de la classe ElementBibliothèque



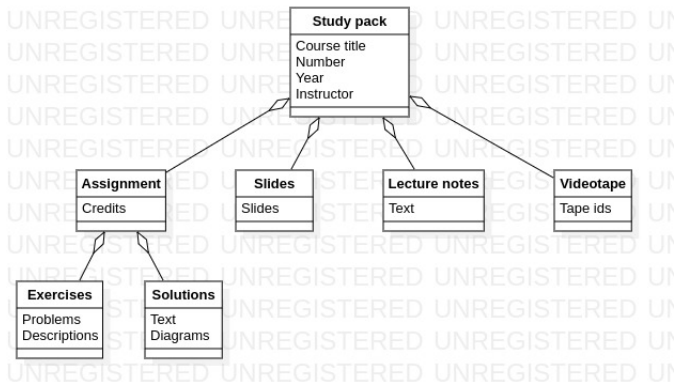
Héritage multiple

- Un système qui supporte l'héritage multiple autorise les classes d'objets à **hériter de plusieurs classes de bases**.
- Peut conduire à des **conflits sémantiques**, où les attributs et services de même nom mais membres de classes de bases différentes peuvent avoir des sémantiques différentes.
- L'héritage multiple rend la réorganisation d'une hiérarchie de classe plus complexe.



Agrégation d'objets











- Une modélisation **d'agrégation** montre comment les classes qui représentent des **collections** sont composées d'autres classes.
- Les modélisation d'agrégation sont semblables à la relation `partie_de` dans les modélisations de données sémantiques.



Modélisations objet comportementales

- Une modélisation comportementale montre les interactions entre objets pour produire un comportement particulier d'un système tel que spécifié dans un cas d'utilisation
- En UML les interactions entre objets sont représentées par des **diagrammes de séquence**, ou **diagrammes de collaboration**.

UML Cheatsheet

Shape	Description
	Package A collection of interfaces and classes.
	Interface Microsoft guidelines specify that interfaces should start with I. This graphic can also sometimes be used as an abstract class.
	Class Properties or attributes sit at the top, methods or operations at the bottom. + indicates public and # indicates protected.
 	<p>These are both typically drawn vertically:</p> Inheritance - B inherits from A. "is-a" relationship.
	Association - A and B call each other
	One way Association. A can call B's properties/methods, but not visa versa.
	Aggregation A "has-a" instance of B. B can survive if A is disposed.
	Composition A has an instance of B, B cannot exist without A.
	A note Some descriptive text attached to any item.

Associations and aggregation/composition can have *1 or n attached to either end of the relationship.

UML Cheatsheet

UML is a mechanism for communication. It is intended to convey the meaningful parts of your application. Include the data which will help someone understand your code, not everything must be included (unless it's an exam, then include everything).

Representing Classes

The basic method for representing fields and methods is:

Fields: **Methods:**
 name: Type name(paramName1: Type, paramName2: Type): returnType

Below is a general template for representing classes, and a small representation of the String class. If you're representing an interface, put <<interface>> above the class name, for an abstract class, put the name in *italics*.

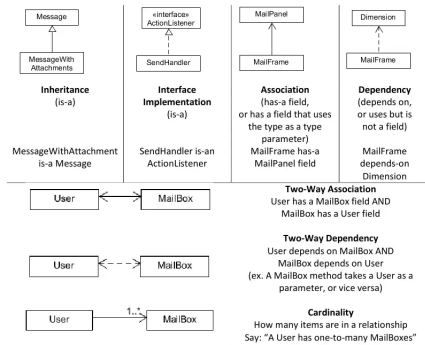
Template:

ClassName
fieldName: fieldType
methodName(paramName: paramType): returnType

String Representation:

String
data: char[]
substring(startIndex: int, endIndex: int): String
replace(toFind: String, toReplace: String): String

Arrows:



Plan

- 1 Introduction
- 2 Écointformatique : écoconception de logiciels et services numériques
- 3 Analyse des exigences
- 4 Modélisation objet
- 5 Structures de données**
- 6 Java efficace (d'après *Effective Java*, Joshua Bloch)
- 7 Écoconception web : les 115 bonnes pratiques

Section 6

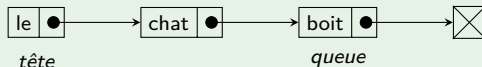
Structures de données

Références

Goodrich, M.T. and Tamassia, R., 2011. Data Structures and Algorithms in Java Fifth Edition International Student Version. Department of Computer Science University of California. Irvine, John Willey & Son.

Liste simplement chaînée

Une structure chaînée est une structure de données dont les éléments sont stockés dans des **nœuds**, reliés entre eux et organisés dans un **ordre linéaire**.



Liste simplement chaînée dont les éléments sont du texte. Les pointeurs suivants de chaque nœud sont représentés par des flèches. Le dernier nœud est l'objet `null`.

- chaque nœud contient une **référence** au nœud suivant
- le premier et le dernier nœud sont appelés respectivement **tête** et **queue**
- l'élément suivant du nœud de queue est `null`
- l'ordre des éléments contenus dans la liste est déterminé par la chaîne de suivants
- la taille d'une liste est indéfinie
- la position de chaque nœud n'est **PAS** indexée

Implémentation d'une liste simplement chaînée

On définit une classe Node comme suit :

```
1 public class Node {  
    private String element;  
3    private Node next;  
  
5    /** Cree un noeud avec l'element et le noeud suivant donnees */  
    public Node(String s, Node n) {  
7        element = s;  
        next = n;  
9    }  
  
11   /** Retourne l'element de ce noeud */  
    public String getElement() { return element; }  
13  
15   /** Retourne le noeud suivant de ce noeud */  
    public Node getNet() { return next; }  
  
17   /** Modifie l'element de ce noeud */  
    public void setElement(String newElem) { element = newElem; }  
19  
21   /** Modifie le noeud suivant de ce noeud */  
    public void setNext(Node newNext) { next = newNext; }  
}
```

Question

Remarquer que les éléments doivent être de type `String`. Comment définir un type de nœud qui puisse accueillir des éléments de type quelconque ?

Implémentation d'une liste simplement chaînée

Étant donnée la classe Node, on définit la classe SLinkedList comme suit :

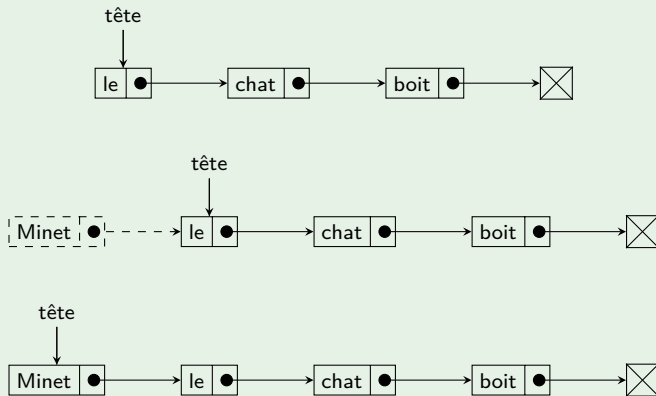
```
/** Liste simplement chainee */
2 public class SLinkedList {
    private Node head; // noeud tete de la liste
4     private long size; // nombre de noeuds de la liste

6     /** Constructeur par défaut, qui cree une liste vide */
    public SLinkedList() {
8         head = null;
        size = 0;
10    }

12    ...
}
```

Insertion dans une liste simplement chaînée

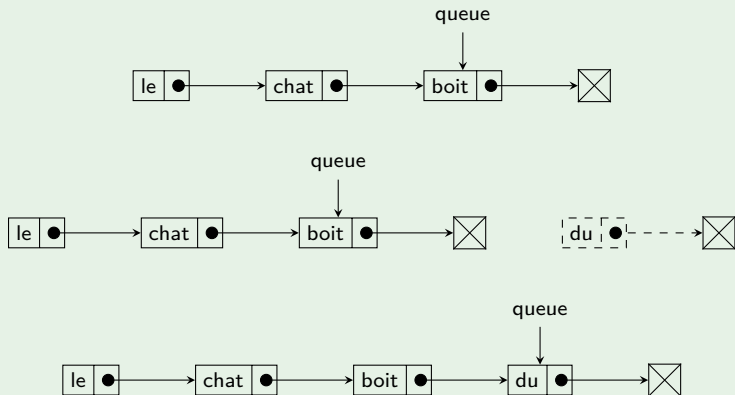
Insertion en tête de liste



Question

Que se passe-t-il lorsque la liste est vide ?

Insertion en queue de liste



Question

Que ce passe-t-il lorsque la liste est vide ?

Insertion en queue de liste

Implémentation

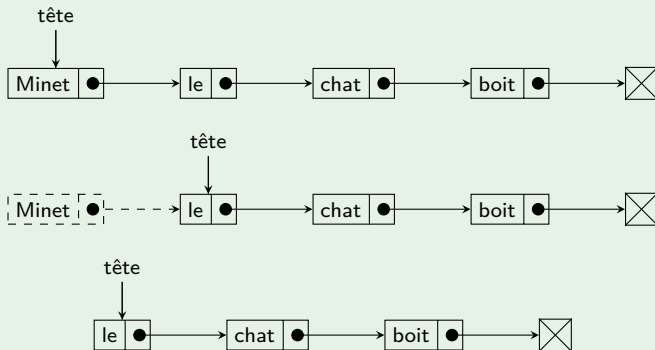
Algorithme $\text{addLast}(\nu)$

```
 $\nu.\text{setNext}(\text{null})$  {faire pointer le suivant nouveau nœud  $\nu$  vers l'objet  $\text{null}$ }  
 $\text{tail}.\text{setNext}(\nu)$  {faire pointer l'ancienne queue vers le nouveau nœud}  
 $\text{tail} \leftarrow \nu$  {modifier la queue pour qu'elle devienne le nouveau nœud}  
 $\text{size} \leftarrow \text{size} + 1$  {incrémenter le nombre d'éléments}
```



Implémenter cet algorithme en java...

Supression d'élément d'une liste simplement chaînée



Question

Que se passe-t-il lorsque la liste est vide ?

Question

Comment procéder pour supprimer en queue de liste ?

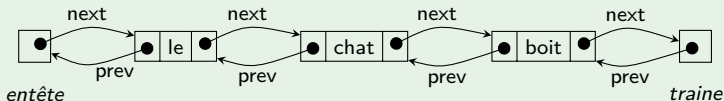
Liste doublement chaînée

Implémentation des nœuds : le type DNode

L'idée du double chaînage est que chaque nœud soit lié au suivant et au précédent.

```
1 /** Nœud d'une liste doublement chainee d'elements de type String */
2 public class DNode {
3     private String element; // element stocke par le noeud
4     private DNode next, prev; // pointeurs vers le noeud suivant et le noeud precedent
5
6     /** Constructeur qui cree un noeud avec les attributs donnees */
7     public DNode(String e, DNode p, DNode n) {
8         element = e;
9         prev = p;
10        next = n;
11    }
12
13    /** Les getters et setters sont definis ici */
14 }
```

Sentinelles d'entête et de traine



Les nœuds sentinelles...

- sont des nœuds spéciaux, placés juste avant le nœud de tête de liste (*entete*), et juste après le nœud de queue de liste (*traine*)
- sont là pour des raisons pratiques, pour simplifier la programmation
- sont en quelque sorte des nœuds "factices", qui ne contiennent aucun élément
- la sentinelle d'entete a un précédent (*prev*) null
- la sentinelle de traine a un suivant (*next*) null

Question

En quoi ces nœuds sentinelles simplifient-ils la programmation ?

Implémentation partielle d'une classe pour une liste doublement chaînée

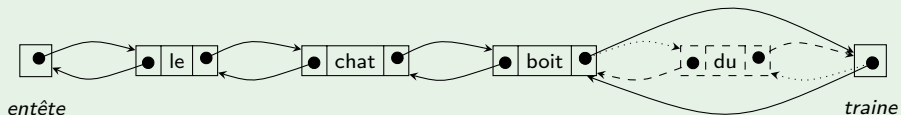
```
2 public class DLinkedList {  
3     private DNode header, trailer; // sentinelles : header=entete, trailer=train  
4     private long size; // nombre d'elements de la liste  
5  
6     /** Costructeur par default qui cree une liste vide */  
7     private DLinkedList() {  
8         header = new DNode(null, null, null);  
9         trailer = new DNode(null, header, null);  
10        header.setNext(trailer); // entete et traine pointent l'une vers l'autre  
11        size = 0;  
12    }  
13    ...  
14 }
```



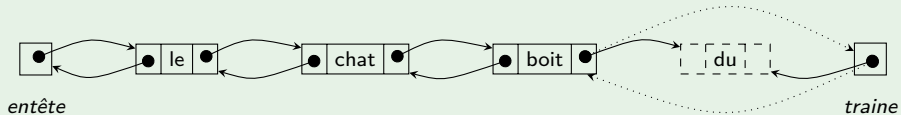
Compléter le code avec les méthodes qui vous semblent utiles (construction, modification, recherche, etc.).

Insertion ou suppression d'éléments

Insertion



Suppression



Questions

- Comment insérer/supprimer en milieu de liste ?
- Que se passe-t-il quand la liste est vide ?

- Les *types génériques* sont des **types abstraits** qui permettent d'éviter les opérations de **transtypage** (*cast*)
- un type générique est un type qui n'est **pas défini à la compilation**, mais qui devient **complètement spécifié à l'exécution**
- étant donné une classe définie pour un type générique <T>, un objet de cette classe est instancié en spécifiant un **type concret** à la création de l'objet (*effacement de type*).

Exemple 1

```
List<Integer> myIntList = new LinkedList<Integer>();  
2 myIntList.add(new Integer(0));  
Integer x = myIntList.iterator().next();
```

Exemple 2

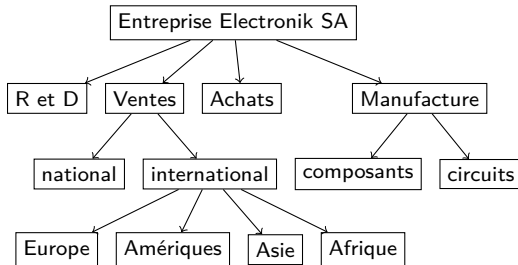
```
1 public class Pair<K,V> {
    K key;
3   V value;
    public void set(K k,V v){
5       key = k;
        value = v;
7   }
    public K getKey() { return key; }
9   public V getValue() { return value; }
    public String toString(){
11        return "[" + getKey() + ", " + getValue() + "];"
    }
13
    public static void main (String[] args) {
15        Pair<String,Integer> pair1 = new Pair<String,Integer>();
        pair1.set("height", new Integer(36));
17        System.out.println(pair1);
        Pair<Student,double> pair2 = new Pair<Student,Double>();
19        pair2.set(new Student("A5976,Sue",19), new Double(9.5));
        System.out.println(pair2).
21    }
}
```

Exemple 3

```
public static <K extends Comparable, V, L, W> int comparePairs(Pair<K,V> p, Pair<L,W> q) {  
2   return p.getKey().compareTo(q.getKey()); // la cle de p implemente compareTo  
}
```

Première définition et propriétés

- Un **arbre** est une *structure de donnée abstraite* qui stocke ses éléments de façon **hiérarchique**
- à l'exception de l'élément le plus haut (dans la hiérarchie), chaque élément d'un arbre a un élément **parent** et zéro ou plusieurs **enfants**
- l'élément le plus haut dans la hiérarchie est appelé la **racine** de l'arbre
- on représente généralement un arbre dans le sens de la hiérarchie qu'il représente, c'est-à-dire *la racine en haut*



Définition formelle et propriétés

On définit un **arbre** T comme un ensemble de **nœuds**, stockant des *éléments*, et tel que les nœuds sont dans une relation **parent-enfant** qui satisfait les propriétés suivantes :

- si T est non-vide, alors il a un nœud spécial appelé **racine** de T , qui n'a pas de parent
- chaque nœud v de T autre que la racine a un **nœud parent unique** ; chaque nœud dont le parent est w est un **enfant** de w

Question

D'après cette définition, un arbre peut-il être *vide*, i.e. n'avoir aucun nœud ?

Définition récursive

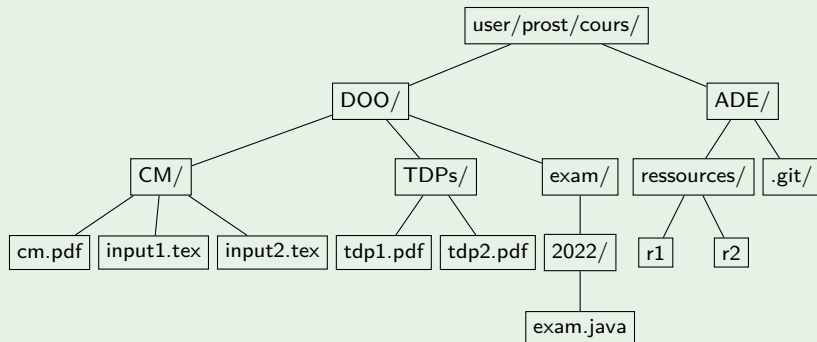
Un arbre T

- soit est un arbre vide
- soit est constitué d'un nœud r racine de T , et d'un ensemble possiblement vide d'*arbres* dont les racines sont les enfants de r .

Autres relations de nœuds

- Deux nœuds enfants d'un même parent sont **frères**
- Un nœud ν est dit **externe**, ou **feuille**, s'il n'a pas d'enfants
- Un nœud ν est dit **interne** s'il a un ou plusieurs enfants

Exemple



L'arbre comme structure de données abstraite

- Une structure d'arbre stocke des éléments dans des **nœuds**, semblables à ceux des listes
- un nœud d'un arbre est défini par rapport à ses **nœuds voisins**

Méthodes de base d'un objet nœud

`element()` : retourne l'objet stocké à ce nœud

`root()` : retourne la racine de l'arbre, ou une erreur si l'arbre est vide

`parent(ν)` : retourne le parent du nœud ν , ou une erreur si ν est la racine

`children(ν)` : retourne les enfants du nœud ν dans une collection iterable

Implémentation d'une structure d'arbre

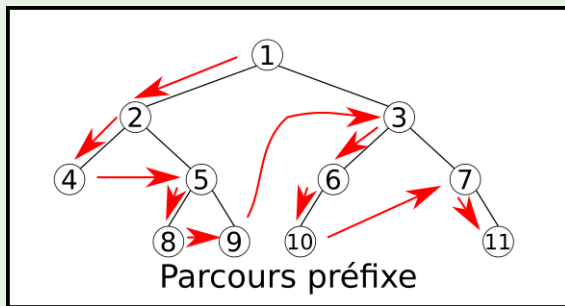


Implémenter comme une structure chaînée

Algorithmes de parcours d'arbre

Parcours préfixe : lecture de la racine en premier

Exemple



source : zonensi.fr

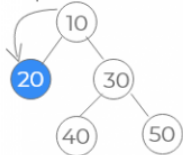
Algorithmes de parcours d'arbre

Parcours postfixe : lecture de la racine en dernier

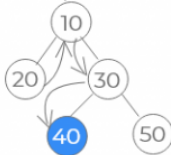
Exemple

Example - postorder Traversal in Binary Tree

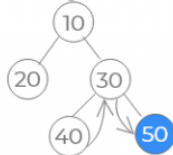
Step 1: Move to left subtree and print 20.



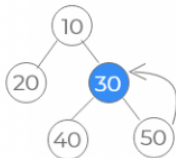
Step 2: Come back and goto right subtree and print 40.



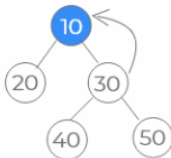
Step 3: move back and print it's right child i.e. 50



Step 4: Move back and print it's root i.e. 30



Step 5: Now move back and print root of the tree i.e. 10

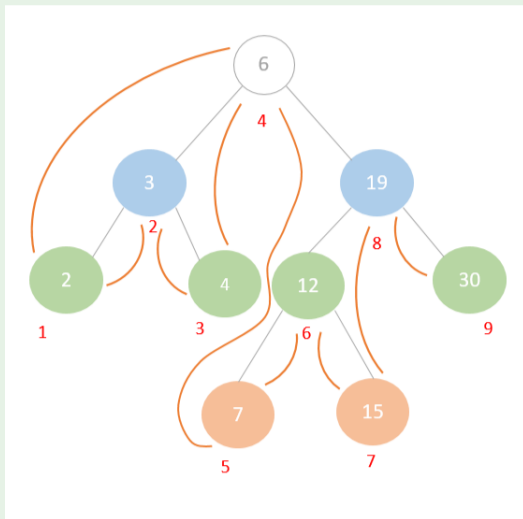


Postorder : 20 40 50 30 10

Algorithmes de parcours d'arbre

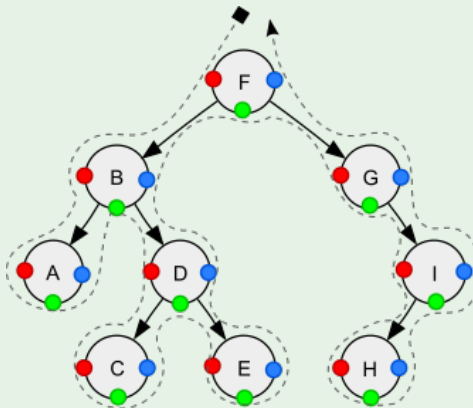
Parcours infixe : lecture de la racine au milieu (arbre binaire)

Exemple



source : baeldung.com

Algorithmes de parcours d'arbre



source : By user:Nomen4Omen, Public Domain, <https://commons.wikimedia.org/w/index.php?curid=99653725>

- Préfixe
- Infixe
- Postfixe

Tables associatives (*maps*)

- Un tableau associatif (*map*) est une structure de données qui permet de stocker des éléments de telle sorte qu'ils puissent être localisés rapidement en utilisant leur **clé**.
- Plus précisément, une map stocke des **paires clé-valeur** (k, v) , appelées **entrées**, où
 - ▶ k est la clé, et
 - ▶ v la valeur correspondante.
- chaque clé doit être unique, pour que l'association des clés aux valeurs définissent une *application*, au sens mathématique (*mapping*)

Du fait de l'unicité des clés, une table associative est le choix de structure approprié pour des situations où les clés servent à **indexer** les valeurs.

Exemple

Index des notes (valeurs) obtenues par chaque étudiant identifié par son numéro d'étudiant (clé)

Implémentation 1 : à base de liste simple

Implémentation 1

Stocker chaque entrée (i.e. chaque paire clé-valeur) dans une liste doublement chaînée

Simple, mais...

Inconvénient

Chaque méthode de base prend un temps $O(n)$ pour une table à n entrée, puisque dans le pire des cas chaque méthode nécessite de parcourir la liste entière.

Implémentation 2 : table de hachage (*hash table*)

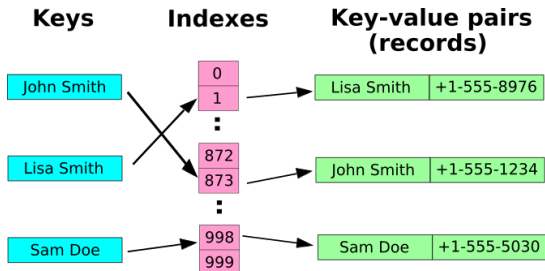
Implémentation 2

En général, une **table de hachage** consiste en 2 composants essentiels :

- un **tableau** de stockage des valeurs, appelé *tableau d'alvéoles* (*bucket array*)
- une **fonction de hachage**

Les valeurs sont indexées par une **valeur de hachage** (un nombre) qui résulte de l'application de la fonction de hachage à une clé donnée.

Autrement dit, les valeurs ne sont pas indexées par les clés elles-mêmes, mais par des valeurs calculées à partir de ce clés, en utilisant la fonction de hachage.



Domaine public, <https://commons.wikimedia.org/w/index.php?curid=441890>

Table de hachage

Avantage

Une table de hachage permet un accès en $O(1)$ en moyenne, comme un tableau ordinaire (i.e. un accès direct).

Inconvénient

Introduire une étape intermédiaire de calcul d'une valeur de hachage (à partir d'une clé) entraîne un risque de **collision**, lorsque 2 ou plusieurs clés distinctes se retrouvent associées à une seule et même valeur de hachage.

Fonction de hachage

Principe

Le rôle d'une fonction de hachage est de répartir les entrées (i.e. paires clé-valeur) dans un tableau d'alvéoles, en permettant de transformer une clé en une valeur de hachage qui indique une position dans le tableau d'alvéoles.

Le calcul de la valeur de hachage se fait parfois en 2 temps (c'est le cas en Java) :

- ① calcul d'un entier (le **hash code**, en Java), par une fonction propre à l'application
- ② **compression** de ce nombre (par une *fonction de compression*) pour le ramener à une valeur comprise dans le domaine de valeurs $[0, N - 1]$ des index du tableau d'alvéoles, où N est la taille du tableau. On utilisera généralement le reste modulo N .

Exemple

Si la clé est une chaîne de caractères, on pourra combiner les codes numériques (e.g. UTF8, ASCII) de chaque caractère avec une fonction telle que le ou exclusif pour obtenir un entier.

Choix d'une fonction de hachage

Les performances d'une fonction de hachage dépendent des facteurs suivants :

- réduction du risque de collision : uniformité de la distribution des valeurs de hachage

Contrexemple

Hacher une chaîne de caractères en associant le code ASCII/UTF de son premier caractère à la clé : quel risque de collision ?

- rapidité d'exécution de la fonction, compression comprise
- taille à réserver à l'espace de hachage

Choix d'une fonction de hachage

Une bonne fonction de hachage offre un bon compromis entre ces différents critères.

Exemples (d'après Wikipedia, "Table de hachage")

- Un ou exclusif de tous les caractères d'une clé fournissait souvent un compromis acceptable dans l'écriture de compilateurs au début des années 60.
- La **taille d'une table de hachage** est souvent un **nombre premier**, afin d'éviter les problèmes de diviseurs communs, qui créeraient un nombre important de collisions.
- La taille d'une table peut aussi être une **puissance de 2**, pour réaliser l'opération modulo par de simples décalages, et donc gagner en rapidité.

Plan

- 1 Introduction
- 2 Écointformatique : écoconception de logiciels et services numériques
- 3 Analyse des exigences
- 4 Modélisation objet
- 5 Structures de données
- 6 Java efficace (d'après *Effective Java*, Joshua Bloch)
- 7 Écoconception web : les 115 bonnes pratiques

Section 7

Java efficace (d'après *Effective Java*, Joshua Bloch)

Fiche 1 : envisager des méthodes de fabriques statiques (*static factory method*) à la place de constructeurs

Exemple (Qu'est-ce qu'une *méthode fabrique statique*?)

Dans Boolean (le type enveloppe du type primitif boolean) :

```
1  public static Boolean valueOf(boolean b) {  
    return b ? Boolean.TRUE : Boolean.FALSE;  
3  }
```

Remarque

Attention à ne pas confondre avec le pattern méthode Fabrique

- Avantages

- ▶ Les méthodes Fabriques statiques ont un nom, contrairement aux constructeurs
- ▶ elles ne sont pas obligées de générer un nouvel objet à chaque appel, contrairement aux constructeurs
- ▶ elles peuvent retourner un objet de n'importe quel type dérivé, contrairement aux constructeurs
- ▶ la classe de l'objet retourné peut varier d'un appel à l'autre en fonction des paramètres en entrée
- ▶ la classe de l'objet retourné par une méthode fabrique statique n'a pas besoin d'exister au moment d'écrire la classe qui contient la fabrique statique

- Inconvénients

- ▶ les classes sans constructeurs publics ou protégés ne peuvent pas être dérivées
- ▶ les méthodes fabrique statique peuvent être difficile à trouver par le programmeur

Remarque

Les constructeurs publics et les fabriques statiques ont chacun leur utilité, il faut connaître leurs mérites respectifs. Les fabriques statiques sont souvent préférables : éviter donc le réflexe des constructeurs publics sans envisager avant les fabriques statiques.

Fiche 2 : envisager un bâtisseur (*builder*) lorsqu'un constructeur a de nombreux paramètres

Fiche 6 : éviter de créer des objets non-nécessaires

Préférer réutiliser un objet existant qu'en redéfinir un nouveau.

Contre-exemple

```
1 String s = new String(''bikini''); // NE FAITES PAS CA !
```

Q : Pourquoi ? (cf. CM pour la réponse)

On peut souvent éviter la création d'objets non-nécessaires par l'utilisation de *méthodes Fabriques statiques* (cf. Fiche 1).

Contre-exemple

HORRIBLEMENT LENT ! Pouvez-vous trouver la création d'objet ?

```
1 private static long sum() {  
    Long sum = 0L;  
3     for (long i = 0; i <= Integer.MAX_VALUE; i++) {  
        sum += i;  
5     }  
    return sum;  
7 }
```

Fiche 6 : éviter de créer des objets non-nécessaires

Préférer réutiliser un objet existant qu'en redéfinir un nouveau.

Contre-exemple

```
1 String s = new String(''bikini''); // NE FAITES PAS CA !
```

Q : Pourquoi ? (cf. CM pour la réponse)

On peut souvent éviter la création d'objets non-nécessaires par l'utilisation de *méthodes Fabriques statiques* (cf. Fiche 1).

Contre-exemple

HORRIBLEMENT LENT ! Pouvez-vous trouver la création d'objet ?

```
1 private static long sum() {  
    Long sum = 0L;  
3     for (long i = 0; i <= Integer.MAX_VALUE; i++) {  
        sum += i;  
5     }  
    return sum;  
7 }
```

C'est l'histoire d'une coquille d'un seul caractère...

Fiche 7 : éliminer les références à un objet obsolète

- Présence en Java d'un mécanisme de Garbage Collection ne signifie pas qu'il faille ignorer la gestion de la mémoire
- les fuites de mémoires dans un système avec Garbage Collection sont insidieuses

Préconisations :

- ❶ définir les variables avec la portée la plus courte possible (cf. Fiche 57) et les laisser tomber hors de portée naturellement (option à privilégier)
- ❷ mettre les objets inutiles à `null` lorsque les structures gèrent elles-mêmes la mémoire, comme
 - ▶ les extensions explicites d'espace (cf. ex. de Stack p. 27)
 - ▶ l'utilisation d'un cache
 - ▶ les `listener` (pour la prog. événementielle, notamment pour la prog. graphique)

Fiche 10 : respecter le contrat général en redéfinissant equals

Inutile de redéfinir equals quand :

- chaque instance de la classe est intrinsèquement unique (par ex. Thread)
- il n'y a pas besoin d'effectuer de tests d'“égalité logique”
- la classe de base a déjà redéfini equals, et le comportement de la classe de base est approprié pour cette classe
- la classe est `private` ou visible seulement du package, et vous êtes certain que equals ne sera jamais appelé

Remarque

Attention ! Rédiger une méthode equals n'est pas trivial. Il est important de respecter le contrat, tel que spécifié dans la javadoc de Object. Pour les 1ères implémentations il est conseillé de s'appuyer sur la génération automatique d'un squelette par un IDE.

Voir Effective Java (EJ) pp.37–49

Fiche 11 : toujours redéfinir hashCode lorsqu'on redéfinit equals

Respect du *contrat général* du hashCode (cf. Javadoc de Object)

hashCode *doit* être redéfini dans toute classe qui redéfinit equals

faute de quoi des classes comme HashMap ou HashSet risquent de ne pas fonctionner correctement

Contrat général de hashCode (adapté des spécification de Object)

- lorsque le hashCode d'un même objet est invoqué de façon répétée dans une application il doit invariablement retourner la même valeur dans la mesure où aucune des informations utilisées dans equals n'est modifiée
- si deux objets distincts sont logiquement égaux par equals alors l'appel du hashCode sur ces 2 objets *doit retourner la même valeur entière*
- si deux objets distincts sont logiquement *différents* par equals alors il n'est *pas* nécessaire que l'appel du hashCode sur les deux objets retourne des valeurs entières distinctes. Néanmoins, produire des hashCode distincts pour des objets différents par equals améliore les performances

Problème

En cas de non redéfinition de hashCode le problème majeur vient de la violation du 2ème point : des objets égaux doivent avoir même hashCode

Exemple

```
1  Map<PhoneNumber, String> m = new HashMap<>();  
   m.put(new PhoneNumber(33, 612345678), "Kristin");
```

Q : Que devrait retourner la requête `m.get(new PhoneNumber(33, 612345678))` ?

Exemple

```
Map<PhoneNumber, String> m = new HashMap<>();  
2  m.put(new PhoneNumber(33, 612345678), "Kristin");
```

Q : Que devrait retourner la requête `m.get(new PhoneNumber(33, 612345678))` ?

Réponse

devrait retourner "Kristin" mais retourne en fait null

Exemple

```
Map<PhoneNumber, String> m = new HashMap<>();  
2 m.put(new PhoneNumber(33, 612345678), "Kristin");
```

Q : Que devrait retourner la requête `m.get(new PhoneNumber(33, 612345678))` ?

Réponse

devrait retourner "Kristin" mais retourne en fait null

Question

Pourquoi ?

Exemple

```
Map<PhoneNumber, String> m = new HashMap<>();  
2 m.put(new PhoneNumber(33, 612345678), "Kristin");
```

Q : Que devrait retourner la requête `m.get(new PhoneNumber(33, 612345678))` ?

Réponse

devrait retourner "Kristin" mais retourne en fait null

Question

Pourquoi ?

Réponse...

Parce que *deux objets distincts* sont créés par le put, puis par le get

Comment rédiger un bon hashCode ?

Commencer par s'appuyer sur la génération automatique par un IDE...

Une bonne fonction de hashage tend à produire des hashcodes différents pour des instances inégales

Comment rédiger un bon hashCode ?

Une recette simple

cf. EJ p.51

Fiche 12 : toujours redéfinir toString

- rend le débogage plus confortable, ainsi que l'utilisation des systèmes qui utilisent toString comme les IDE
- en pratique, toString doit présenter toutes les informations intéressantes relatives à l'objet

Fiche 14 : envisager d'implémenter Comparable

L'implémentation de `Comparable` indique à Java que la classe en question est dotée d'un *ordre naturel*, qui permet notamment des appels du genre :

```
Arrays.sort(a);
```

- rend la classe naturellement compatible avec les nombreux algorithmes, notamment sur les collections, fournis par l'API
- s'assurer du respect du *contrat général* tel que spécifié dans la javadoc
- il est notamment recommandé, mais pas requis, de s'assurer de la cohérence de `compareTo` et `equals`, de telle sorte que

```
1    (x.compareTo(y) == 0) == (x.equals(y))
```

Fiche 15 : minimiser l'accessibilité des classes et des membres

- **les attributs d'instance** des classes publiques ne devraient que **très rarement être publics** (cf. Fiche 16)
- il est **mauvais** pour une classe d'avoir un **attribut tableau statique final public**, ou un accesseur qui retourne un tel attribut

Remarque

*Attention ! le point 2 est une source fréquente de **trous de sécurité**.*

Fiche 16 : dans les classes publiques utiliser des *accesseurs* plutôt que des attributs publics

Propriété

Une classe immuable est une classe dont les instances ne peuvent pas être modifiées. Donc toute l'information contenue dans un tel objet est fixée pour toute la vie de l'objet, et ne peut changer en aucun cas.

Exemple

Java fournit plusieurs classes immuables, comme `String`, les enveloppes de types primitifs (`Boolean`, `Integer`, ...), `BigInteger` et `BigDecimal`.

Les classes immuables sont moins sujettes aux erreurs et plus sécurisées.

Ce qu'induit l'immuabilité :

- Les objets immuables sont simples (un objet immuable ne peut être que dans un seul état, celui de sa création)
- les objets immuables peuvent être partagés, et partager leur composants internes
- les objets immuables font de très bons blocs de construction pour d'autres objets, que ces derniers soient mutables ou non
- les objets immuables ne peuvent, par définition, jamais être dans un état incohérent, même en cas d'échec
- l'inconvénient majeur des classes immuables est qu'elles requièrent un objet distinct pour chaque valeur
- Une classe devrait être immuable à moins qu'il y ait une vraie raison pour la rendre mutable (en gros, commencer par éviter de fournir systématiquement un `setter` pour tout `getter`)
- si une classe ne peut pas être immuable, limiter sa mutabilité
- les constructeurs ne devraient créer que des objets entièrement initialisés

Pour rendre une classe immuable, suivre les 5 règles suivantes :

- ❶ ne fournir aucune méthode qui modifie l'état de l'objet (connues sous le nom de *mutateurs*)
- ❷ assurer que la classe ne puisse pas être dérivée (la déclarer `final`)
- ❸ définir tous les attributs `final`
- ❹ définir tous les attributs `private`
- ❺ assurer un accès exclusif à tout composant mutable (s'assurer que les clients de la classe ne peuvent pas obtenir les références d'éventuels attributs mutables)

Voir l'exemple de la classe `Complex` dans EJ p. 81.

Fiche 18 : favoriser la composition plutôt que l'héritage

- Utilisé de façon inappropriée, l'héritage peut fragiliser un logiciel
- l'héritage est sûr dans les cas suivants :
 - ▶ dans un même package, quand classe de base ET classe dérivée sont sous le contrôle des mêmes programmeurs
 - ▶ quand la classe de base a spécifiquement été conçue pour être dérivée (cf. Fiche 19)

Quand et pourquoi ne pas hériter

- Il est dangereux d'hériter de classes concrètes ordinaires entre différents packages, parce que
- dans un tel contexte (héritage hors package) *l'héritage viole l'encapsulation*, contrairement à l'appel de méthode, parce que
- la classe dérivée dépend du fonctionnement interne de la classe de base pour son propre fonctionnement

Fiche 20 : préférer les interfaces aux classes abstraites

- En java, 2 mécanismes d'implémentation d'un type abstrait : classe abstraite et interface
- Différence majeure : une classe abstraite doit être *dérivée*

Remarque

Rappel : en java, pas d'héritage multiple possible, ce qui contraint sévèrement l'utilisation d'une classe abstraite pour la définition de type

- toute classe qui satisfait les conditions requises peut implémenter une interface
- une classe peut implémenter plusieurs interface

Question

Quelles sont les "conditions requises" auxquelles il est fait référence plus haut ?

- Les classes existantes peuvent facilement être adaptées pour les faire implémenter une nouvelle interface
- les interfaces sont idéales pour définir des fonctionnalités optionnelles, en compléments de celle du “type primaire”
- les interfaces permettent la construction de systèmes non-hiérarchiques de types

Fiche 28 : préférer les listes aux tableaux

Les tableaux diffèrent des types génériques sur un aspect important :

- les tableaux sont *covariants*, tandis que les génériques sont *invariants*

Définition (Covariance)

Si Sub est un type dérivé de Super alors le type tableau Sub[] est un type dérivé du type tableau Super []

Définition (Invariance)

Quels que soient Type1 et Type2 2 types distincts, List<Type1> n'est ni un type dérivé ni un type de base de Type2

Exemple

ce morceau de code est légal, mais échoue à l'exécution :

```
1  Object[] objectArray = new Long[1];  
   objectArray[0] = "Je ne rentre pas !" // Leve ArrayStoreException
```

Tandis que ce code ne l'est pas :

```
   // Attention, ne compile pas !  
2  List<Object> obj = new ArrayList<Long>(); // types incompatibles  
   obj.add("Je ne rentre pas !");
```

Remarque

Il existe une autre différence importante entre tableaux et génériques, mais qui dépasse le cadre de ce cours. Voir EJ(3) pp. 126–130 pour plus de détails)

Fiche 29 : favoriser les types génériques

Envisager de définir vos propres types génériques, par exemple dans vos propres structures de données (voir EJ(3) pour les motivations)

Fiche 34 : utiliser Enum plutôt que des constantes int

Rappel

Un *type énuméré* est un type dont le domaine de valeur est un ensemble fixe de constantes (comme les saisons de l'année, les planètes du système solaire, les valeurs d'un jeu de carte, etc.)

Lorsqu'un type `enum` peut être défini, **éviter l'utilisation du *pattern enum***, qui consiste à définir un ensemble de *constantes nommées*, et qui est **particulièrement inefficace** et présente de **nombreux inconvénients**

Exemple (Le pattern enum, très inefficace !)

```
1  public static final int APPLE_FUJI = 0;
   public static final int APPLE_PIPPIN = 1;
3  public static final int APPLE_GRANNY_SMITH = 2;

5  public static final int ORANGE_NAVEL = 0;
   public static final int ORANGE_TEMPLE = 1;
7  public static final int ORANGE_BLOUD = 2;
```

- aucune vérification de type (on peut mélanger pommes et oranges, torchons et serviettes)
- très peu d'expressivité
- parce que les valeurs entières sont compilées, et donc "en dur" dans les programmes clients du type, si les valeurs entières changent, tous les programmes clients doivent être recompilés
- pas de moyen simple d'associer une chaîne de caractère à chaque valeur entière

Remarque

Attention, le pattern String enum, variante du int enum, est pire encore....

Les enum Java sont très sophistiqués : voir EJ(3) pour apprendre à bien les utiliser

Exemple

On retrouve communément les cas suivants :

- une valeur d'index doit être positive
- une référence d'objet doit être non-nulle
- etc.

- À défaut de vérifier la validité, plus l'erreur induite interviendra tard, plus elle sera difficile à détecter
- la méthode `Objects.requireNonNull`, à partir de Java 7, est pratique et facile à utiliser ; il n'est donc plus nécessaire de vérifier manuellement les références nulles

```
1    this.strategy = Objects.requireNonNull(strategy, "message d'erreur strategy null");
```

Fiche 50 : faire des copies défensives lorsqu'il y en a besoin

- Programmer *défensivement*, en partant du principe que les programmes clients de votre classe feront tout pour détruire ses invariants
- créer les copies défensives des objets mutables *avant* de vérifier la validité des paramètres (Fiche 49), et effectuer la vérification sur les copies plutôt que sur les originaux
- se méfier de la méthode `clone` pour faire une copie d'un paramètre dont le type peut être dérivé par un programme tiers

Remarque

Voir EJ(3) pour plus de détails, et de nombreux exemples d'exploitation de trous de sécurité divers qui peuvent être évités par des copies défensives

Fiche 52 : utiliser la surcharge judicieusement

Le programme suivant prétend vouloir classer des collections selon qu'elles sont des Set, List, ou un autre type de Collection :

```
1 public class CollectionClassifier {
2     public static String classify(Set<?> s) {
3         return "Set";
4     }
5
6     public static String classify(List<?> l) {
7         return "List";
8     }
9
10    public static String classify(Collection<?> c) {
11        return "Collection inconnue";
12    }
13
14    public static void main(String[] args) {
15        Collection<?>[] collections = { new HashSet<String>(),
16            new ArrayList<BigInteger>(),
17            new HashMap<String, String>().values() };
18
19        for (Collection<?> c : collections) {
20            System.out.println(classify(c));
21        }
22    }
23 }
```

- On peut s'attendre à ce que ce programme affiche "Set", suivi de "List" et enfin "Collection inconnue"
- pourtant il affiche en fait 3 fois "Collection inconnue"..

Question

Pourquoi ?

Solution

- Parce que la méthode `classify` est *surchargée*, et que
- **le choix de quelle méthode surchargée est appelé s'effectue au moment de la compilation**
- or pour les 3 itération de la boucle, le type du paramètre est toujours le même : `Collection<?>`,
- donc la seule méthode applicable à la compilation est `classify(Collection<?>`
- même si le type dynamique du paramètre est différent à l'exécution

- Éviter les utilisations ambiguës de la surcharge
- *éviter en particulier de combiner surcharge et redéfinition*
- préférer donner des noms différents plutôt que surcharger

Fiche 56 : rédiger la javadoc pour tous les éléments exposés d'API

Voir EJ(3) pp. 254–260

Fiche 57 : minimiser la portée des variables locales

- Certains vieux langage, comme le C (avant C99), rendaient obligatoire la déclaration des variables locale en début de bloc
- par la force de l'habitude, on peut être tenté de conserver cette pratique en Java
- **cette habitude mérite d'être cassée**

- La technique la plus efficace pour minimiser la portée d'une variable locale consiste à la déclarer lors de sa première utilisation
 - ▶ meilleure lisibilité (évite les risques de confusions si déclaration trop tôt)
 - ▶ déclaration trop tôt peut aussi signifier une fin de portée trop tardive (en fin de bloc)
- quasiment toutes les déclarations de variable locale devraient être associées à une initialisation (s'il manque de l'information pour initialiser, c'est probablement que la déclaration peut attendre)
- garder les méthodes courtes et spécialisées

Fiche 58 : préférer les boucles for-each aux boucles for traditionnelles

Boucle for traditionnelle pour itérer sur un tableau :

```
1 // Pas la meilleure facon d'iterer sur un tableau !
  for (int i = 0 ; i < a.length ; i++) {
3   ... // faire qq chose avec a[i]
  }
```

Boucle for traditionnelle pour itérer sur une Collection :

```
    // Pas la meilleure facon d'iterer sur une Collection !
2  for (Iterator<Element> i = c.iterator() ; i.hasNext() ;) {
    Element e = i.next();
4    ... // faire qq chose avec e
  }
```

Problème

- l'iterator et la variable d'index `i` sont du bruit : seuls les éléments sont utiles
- ce bruit introduit autant de sources d'erreurs possibles
- les 2 boucles sont très différentes syntaxiquement, et dépendent (trop) du contenant (tableau ou collection) sans que ce soit nécessaire

Solution

L'instruction "for-each" (officiellement "for amélioré", pour *enhanced for*) résout tous ces problèmes

La meilleure façon d'itérer sur les tableaux et les collections :

```
1  for (Element e : elements) {  
    ... // faire qq chose avec e  
3  }
```

mais ce n'est pas toujours possible...

Fiche 59 : connaître et utiliser les bibliothèques

- L'utilisation d'une bibliothèque standard permet de bénéficier de l'expertise des programmeurs qui l'ont écrite, et de l'expérience de ceux qui l'ont utilisée avant vous
- ne pas perdre son temps sur le développement de solutions adhoc à des problèmes qui ne sont qu'accessoirement reliés à votre application (i.e. se concentrer sur l'essentiel)
- les performances d'une bibliothèque standard ont tendance à s'améliorer au cours du temps, sans aucun effort de votre part
- les bibliothèques standard ont tendance à avoir plus de fonctionnalités avec le temps
- l'utilisation de bibliothèques standard contribue à standardiser votre propre code, en le rendant plus lisible, maintenable et réutilisable

Recommandation

L'API java est trop volumineuse pour étudier toute la javadoc de Java9, mais **tout programmeur devrait connaître les bases de `java.lang`, `java.util`, et `java.io` et de leurs sous-paquetages**

Fiche 63 : attention aux performances de la concaténation de String

Problème

La concaténation à répétition de chaînes de caractères (+) pour concaténer n chaînes nécessite un temps quadratique en n

- Conséquence (malheureuse) de l'immuabilité du type String
- lorsque 2 chaînes sont concaténées, le contenu des 2 est copié

Exemple

```
1 // utilisation inappropriée de la concatenation - mauvaises performances
2 public String statement() {
3     String result = "";
4     for (int i = 0 ; i < numElem(); i++) {
5         result += lineForElem(i); // concatenation de String
6     }
7     return result;
8 }
```

Les performances sont abyssales lorsque le nombre d'éléments (`numElem()`) est grand

Recommandation

Pour des performances acceptables, utiliser un `StringBuilder` au lieu de `String` pour stocker la chaîne en construction

```
public String statement() {  
2   StringBuilder b = new StringBuilder(numElem() * LINE_WIDTH);  
   for (int i = 0 ; i < numElem() ; i++) {  
4       b.append(lineForElem(i));  
       }  
6   return b.toString();  
}
```

Remarque

Même si les performances ont été grandement améliorées depuis Java 6, la différence entre les 2 méthodes est toujours dramatique :

si `numElem` retourne 100 et `lineForElem` retourne une chaîne de 80 caractères, la seconde méthode est 6,5 fois plus rapide que la première [sur la machine de Joshua Bloch]

Fiche 64 : faire référence aux objets par leur interface

Fiche 68 : adopter les conventions de nommage généralement acceptées

Fiche 69 : n'utiliser les exceptions que pour les conditions exceptionnelles

Fiche 85 : préférer les alternatives à la sérialisation Java

Sérialisation

Moyen en Java pour encoder des objets sous la forme de flux binaires (*sérialiser*), et de reconstruire des objets à partir de leur encodage (*désérialiser*).

Problème

la sérialisation est un mécanisme très risqué, potentiellement source de failles de sécurité

Anecdote

En 2016, la compagnie de métro de San Francisco a été victime d'une célèbre attaque de type ransomware qui exploitait une faille de sécurité liée à la sérialisation : le système de collecte des paiements des voyages est resté planté pendant 2 jours

Recommandation

- la meilleure protection possible est de ne *pas* utiliser la sérialisation, tout simplement
- ne jamais désérialiser des données dans lesquelles vous n'avez pas confiance

Plan

- 1 Introduction
- 2 Écointformatique : écoconception de logiciels et services numériques
- 3 Analyse des exigences
- 4 Modélisation objet
- 5 Structures de données
- 6 Java efficace (d'après *Effective Java*, Joshua Bloch)
- 7 Écoconception web : les 115 bonnes pratiques

Section 8

Écoconception web : les 115 bonnes pratiques

Écoconception web : les 115 bonnes pratiques

TODO : LISTE DES BONNES PRATIQUES
TODO : PHOTO COUVERTURE BOUQUIN

Quelques outils

- checklist des 115 bonnes pratiques (par ex. à utiliser pour la SAÉ) : voir [ametice](#)
- Référentiel du numérique écoresponsable ([ecoresponsable.numerique.gouv.fr](#))
- Ecoindex
- GreenIT-analysis

Constat



















- **70%** des **fonctionnalités** demandées par l'utilisateur ne sont **pas essentielles**
- **45%** ne sont **jamais utilisées**

Geste clé d'écoconception

- Épuré au maximum la couverture et la profondeur fonctionnelle

À noter que ...

Les *bonnes pratiques* de spécification sont peu nombreuses, mais **incontournables**.

B	Bonne pratique n°1		
	Retenir uniquement les fonctionnalités essentielles		
22 septembre	PRIORITY     	MISE EN ŒUVRE    	IMPACT ÉCOLOGIQUE    
	RESSOURCES ÉCONOMISÉES     		
document est la propriété exclusive de Jean-Philippe Prost (lambda.doe@gmail.com) - jeudi 22 septembre 2022 à 14h02	<p>Plusieurs études démontrent que 70 % des fonctionnalités demandées par les utilisateurs ne sont pas essentielles et que 45 % ne sont jamais utilisées. En réduisant la couverture et la profondeur fonctionnelle, on diminue les besoins en ressources informatiques (cycles CPU, bande passante, serveurs, etc.), ce qui réduit d'autant les impacts environnementaux associés à la dette technique. À niveau ergonomique constant, plus l'application est sobre fonctionnellement, plus elle sera simple à utiliser. Il faut donc centrer l'application sur le besoin essentiel de l'utilisateur. On peut détecter une fonctionnalité non essentielle au moment de l'analyse de l'expression du besoin. La méthode MoSCoW, des ateliers, des wireframes ou des prototypes avec tests utilisateurs permettent de vérifier l'utilité d'une fonctionnalité en amont de son développement.</p>		
	<div>Exemple</div> <p>Les succès récents du Web - Google, Twitter, WhatsApp, Pinterest, Instagram, etc. - fournissent un seul service et misent sur une grande sobriété fonctionnelle. Au moment de l'analyse de l'expression du besoin. Respectez le principe YAGNI (<i>You Ain't Gonna Need It</i>) de l'<i>extreme programming</i> : développez quand vous avez effectivement besoin d'une fonctionnalité, pas lorsque vous imaginez en avoir besoin.</p>		
	<div>Test</div> <p>Le nombre de fonctionnalités dont l'utilité n'a pas été vérifiée avec un panel d'utilisateurs avant développement est égal à 0 %.</p>		

Quantifier précisément le besoin

PRIORITÉ	MISE EN ŒUVRE	IMPACT ÉCOLOGIQUE
    	   	    

RESSOURCES ÉCONOMISÉES



Les « dimensions » de chaque fonctionnalité doivent être définies précisément et dans leur ensemble. Il peut s'agir d'un taux de compression pour les images, du temps de réponse maximal pour une requête HTTP, du nombre d'éléments dans une liste, etc.

Plus les dimensions et exigences associées à chaque fonctionnalité collent au métier, plus on évite la surqualité. La logique doit donc être inversée par rapport aux habitudes actuelles. Si une quantification n'est pas précisée, c'est le niveau de qualité ou la quantité minimale qui est proposé.

Exemple

En l'absence de précision, le nombre d'items d'une liste est limité à 5 éléments ou au nombre maximum affichable sur le plus petit écran cible de l'application.

Gain potentiel : en jouant sur le nombre d'items affichés sur la page de résultats de son moteur de recherche Bing, Microsoft Research a démontré qu'il était possible de réduire jusqu'à 80 % l'infrastructure physique (nombre de serveurs) sous-jacente.

En utilisant par défaut une résolution de vidéo acceptable (480 p) plutôt que maximale, on réduit la bande passante utilisée pour la plupart des utilisateurs (qui ne changeront pas la valeur par défaut), tout en laissant la possibilité aux autres d'augmenter la résolution au besoin.

Test

Le nombre de fonctionnalités avec des dimensions supérieures au besoin est égal à 0.

FRONT

SPÉCIFICATION

45

Supprimer les fonctionnalités non utilisées

PRIORITÉ	MISE EN ŒUVRE	IMPACT ÉCOLOGIQUE
    	  	    
RESSOURCES ÉCONOMISÉES     		

Éliminer les fonctionnalités non utilisées revient à :

- mesurer l'utilisation des fonctionnalités en production ;
- piloter l'usage des fonctionnalités et supprimer celles qui ne sont pas assez utilisées ou qui ont perdu de la valeur.

Le fait de supprimer des fonctionnalités allège le poids de l'application, son impact en production et sa maintenance.

Comment supprimer une fonctionnalité ?

- En la désactivant : sur le principe du *feature flipping*, empêcher qu'elle soit utilisée avec un flag.
- En la désinstallant : supprimer au maximum le code utilisé puis refactorer le code restant.

En fonction du coût environnemental et économique d'une suppression, l'une ou l'autre solution sera retenue.

Exemple

Un site e-commerce utilise deux listes différentes : une liste d'achats et une liste de favoris. Lors de la refonte du site, au lieu de rester iso fonctionnel, l'équipe mesure la pertinence des fonctionnalités. La mesure montre que la liste de favoris est devenue désuète. Elle est donc supprimée, ainsi que toutes les données en base.

Test

Le nombre de fonctionnalités peu utilisées présentes en production est inférieur ou égal à 10 %.

Privilégier une approche « mobile first »

Ce document est la propriété exclusive de Jean-Philippe Prost (lambda.doe@gmail.com) - jeudi 22 septembre

PRIORITÉ	MISE EN ŒUVRE	IMPACT ÉCOLOGIQUE
   	   	    
RESSOURCES ÉCONOMISÉES  		

Lorsque le contexte le permet, privilégier l'approche « mobile first » qui consiste à concevoir un site/service en ligne pour les terminaux mobiles, et à élargir sa couverture fonctionnelle pour de plus grands écrans uniquement si l'apport fonctionnel/ergonomique est justifié. A défaut de « mobile first », opter alors pour le chargement adaptatif. Cette approche consiste à sélectionner les ressources (y compris JS et CSS) les plus adaptées au contexte d'utilisation (taille de l'écran/de la fenêtre, densité de pixels, qualité du réseau etc.), si possible côté serveur. On évite ainsi de consommer inutilement des ressources : bande passante, mémoire vive du terminal, etc.

Exemple

Côté serveur, on peut utiliser les client hints, ou à défaut l'identifiant du navigateur couplé à un tableau de capacités des navigateurs (*user agent sniffing*).

Côté client, les media queries (notamment dans les attributs `media` des `<link>` pour la sélection de feuilles de styles CSS), les attributs `srcset` et `sizes` des ``, les sous-éléments `<source>` des `<picture>`, `<video>` et `<audio>` pourront être utiles. Les mêmes informations peuvent être récupérées par des API JavaScript pour éventuellement charger du code et/ou du contenu complémentaire dynamiquement.

Test

Le nombre de conceptions ne s'appuyant pas sur une approche mobile first est inférieur ou égal à 1.

Bonne pratique n° 5

Optimiser le parcours utilisateur

PRIORITÉ

MISE EN ŒUVRE

IMPACT ÉCOLOGIQUE

RESSOURCES ÉCONOMISÉES

Optimiser le parcours utilisateur consiste à diminuer le temps passé par l'utilisateur sur ses usages les plus fréquents. Dans un premier temps, cibler les parcours les plus fréquents, puis optimiser leur usage : diminuer le nombre d'étapes et d'actions, supprimer l'inutile, identifier les cas d'échecs, optimiser les temps de réponse. Un parcours est bien conçu lorsque le programme se comporte exactement comme l'utilisateur l'avait imaginé. A minima, sonder en observant son entourage utilisant le service est un bon moyen d'identifier les points de friction - situations ou interactions qui contribuent à dégrader l'expérience utilisateur et à ralentir le parcours - des utilisateurs. Les tests utilisateurs permettent d'aller plus en profondeur dans la recherche de ces points de friction. Le temps passé par l'utilisateur sur son terminal est le deuxième poste en termes d'impacts environnementaux.

Exemple

- Proposer, pour un site de grande distribution, une nouvelle commande sur la base du contenu de la précédente.
- Acheter sans inscription sur un site d'e-commerce.
- Copier-coller son RIB directement plutôt que le télécharger puis le transférer.
- Mettre en avant les champs ou les filtres les plus utilisés.

Test

Le nombre de points de friction est égal à 0.

Éviter les animations JavaScript/CSS



PRIORITÉ



MISE EN ŒUVRE



IMPACT ÉCOLOGIQUE

RESSOURCES ÉCONOMISÉES



Les animations Javascript/CSS peuvent être très coûteuses en termes de cycles CPU et de consommation mémoire. Elles déclenchent toutes une action de type `(re)paint/(re)flow` très coûteuse en ressources. Il faut donc éviter au maximum les animations, et ne les utiliser que lorsqu'elles sont indispensables.

Si vous ne pouvez pas vous passer d'une animation, limitez-vous aux propriétés CSS `opacity` et `transform`, ainsi qu'aux fonctions associées `translate`, `rotate` et `scale` de `transform`. Ces deux propriétés sont automatiquement optimisées par le navigateur, et l'animation peut être prise en charge par le processeur graphique (GPU). Le site <https://csstriggers.com> liste les actions sur le DOM déclenchées par une animation.

Pour que le navigateur puisse réduire au minimum les ressources consommées par l'animation, vous pouvez le prévenir qu'une animation va avoir lieu grâce à l'instruction `will-change`. Pour en savoir plus, consultez le site <https://web.dev/animations-guide/>.

Vous pouvez aussi laisser le choix à vos utilisateurs, via les préférences de leurs navigateurs et la media query `prefers-reduced-motion`, de jouer ou non l'animation. Cette dernière est jouée seulement si l'utilisateur n'a pas défini de préférence.

Exemple


















```
css.box { will-change: transform, opacity;}
```

Test

Le nombre d'animations JS/CSS par page est inférieur ou égal à 2.

Limiter le recours aux carrousels

Ce document est la propriété exclusive de Jean-Philippe Prost (lambda.doe@gmail.com) - jeudi 22 septembre

PRIORITÉ	MISE EN ŒUVRE	IMPACT ÉCOLOGIQUE			
   	   	   			
RESSOURCES ÉCONOMISÉES     					
<p>Le carrousel, souvent utilisé pour mettre en avant de multiples contenus, montre rapidement ses limites en matière de conversion, avec un impact négatif sur l'expérience utilisateur. Sa présence implique plusieurs points critiques :</p> <ul style="list-style-type: none"> - un alourdissement du poids des pages par la présence de CSS et JavaScript dédié, mais également par le contenu présent sur chaque écran du carrousel ; - une complexité en termes d'assurance qualité web et d'accessibilité numérique ; - une utilisation plus importante des ressources processeurs dans le cas d'un carrousel automatique. <p>Limiter au maximum l'utilisation des carrousels en privilégiant du contenu statique mis à jour régulièrement. Dans le cas contraire :</p> <ul style="list-style-type: none"> - mettre en place un contrôle simple et complet du composant (arrêt, écran suivant/précédent) ; - préférer un chargement progressif. 					
<p>Exemple</p> <p>Les animations de certains attributs CSS impliquent un repaint/reflow des navigateurs qui demande des ressources machines. Pour limiter cela, préférez l'animation des carrousels par la propriété CSS transform.</p>					
<p>Test</p> <p>Le nombre de carrousels présents sur la page est inférieur ou égal à 1.</p>					
<table> <tr> <td>FRONT</td> <td>CONCEPTION</td> <td>55</td> </tr> </table>			FRONT	CONCEPTION	55
FRONT	CONCEPTION	55			

Avoir un titre de page et une metadescription pertinents

PRIORITÉ



MISE EN ŒUVRE



IMPACT ÉCOLOGIQUE



RESSOURCES ÉCONOMISÉES



Le titre de page `<h1>`, ainsi que son équivalent `<title>`, adjoints à une balise `<meta name="description">` pertinente doivent être parfaitement en accord avec le contenu de la page associée.

Cette optimisation répond à l'intention de recherche de l'utilisateur, lui évitant une visite inutile (des allers-retours pour trouver l'information recherchée). Elle raccourcit aussi son temps de parcours au sein des moteurs de recherche.

Soigner la qualité des éléments `<title>` et `<meta name="description">` permet également de :

- d'indexer correctement les pages de contenus par les moteurs de recherche et les annuaires ;
- d'assurer l'identification du site et du contenu de la page d'atterrissage ;
- de garantir la compréhension avant l'accès au contenu ;
- de qualifier la lecture du contenu ou de l'information demandée au sein de la page associée.

Exemple

Dans le cadre d'un catalogue de vêtements de sports en ligne et de la page présentant les survêtements, on aura un `<title>` du type « Nom-Boutique > Vêtements > sport > survêtement » et une description du type « Achetez en ligne nos survêtements au prix le plus bas »

Test

Le nombre de titres de pages non repris dans la balise `<title>` et sans le nom du site est inférieur à 1.

Favoriser un design simple, épuré et adapté au Web



PRIORITÉ



MISE EN ŒUVRE



IMPACT ÉCOLOGIQUE

RESSOURCES ÉCONOMISÉES



Tout design d'interface ou webdesign doit être réfléchi en amont, en prenant en compte les points suivants :

- les besoins de l'utilisateur (voir la fiche « Optimiser le parcours utilisateur », page 50) ;
- les heuristiques d'ergonomie (Bastien et Scapin, Nielsen, etc.) ;
- les contraintes techniques ;
- les bonnes pratiques d'écoconception ;
- les bonnes pratiques d'accessibilité.

Privilégiez un design simple et épuré, réalisable uniquement en HTML5 et CSS3.

Exemple

Certains sites contiennent des images encadrées, non contrastées et non lisibles (RGAA), ce qui crée une surcharge mentale inutile (2.2. Densité informationnelle de Scapin et Bastien). Téléchargées, ces images ne sont pourtant pas visibles sur mobile (écoconception). On peut parfois soulever l'incohérence entre signalétique et colorimétrie (1.2.2. Groupement/Distinction par le format de Scapin et Bastien).

Test

Le nombre de pages dont le design est plus chargé que nécessaire est égal à 0.

FRONT

CONCEPTION

57

Préférer la saisie assistée à l'autocomplétion

PRIORITÉ



MISE EN ŒUVRE



IMPACT ÉCOLOGIQUE



RESSOURCES ÉCONOMISÉES



L'autocomplétion suggère à l'utilisateur des résultats correspondant à sa recherche pendant sa saisie. À chaque caractère saisi, une requête est envoyée au serveur pour récupérer les résultats appropriés. Ce qui peut être coûteux.

Préférez la saisie assistée qui consiste à guider l'utilisateur par un ensemble d'informations et d'indices gérés localement, ce qui réduit les échanges avec le serveur.

Minimisez l'impact de l'autocomplétion avec des optimisations simples : délai de quelques dixièmes de secondes entre l'ajout d'un caractère et la requête, limitation du nombre de résultats affichés, etc. Pour minimiser les allers-retours, si la donnée proposée à l'utilisateur est en assez petite quantité, vous pouvez l'inclure directement dans votre code HTML et utiliser l'élément natif `<datalist>`.

Exemple

L'élément `<datalist>` un aller-retour avec le serveur.

```
<label for="ice-cream-choice">Choose a flavor:</label>
<input list="ice-cream-flavors"
      id="ice-cream-choice" name="ice-cream-choice" />
<datalist id="ice-cream-flavors">
  <option value="Chocolate">
  <option value="Coconut">
</datalist>
```

Test

Le nombre de champs en autocomplétion est inférieur ou égal à 20 %.












FRONT

CONCEPTION

59

Mettre en cache les données calculées souvent utilisées

Ce document est la propriété exclusive de Jean-Philippe Prost (lambda.doe@gmail.com) - jeudi 22 septembre 2022 14:00

PRIORITÉ	MISE EN ŒUVRE	IMPACT ÉCOLOGIQUE
   	  	   
RESSOURCES ÉCONOMISÉES		
<p>Lorsque des calculs de valeurs ou de données sont coûteux en ressources, il convient de les mettre en cache si les valeurs demeurent inchangées, afin de ne pas réitérer ces opérations.</p> <p>Les systèmes de cache de type <i>key-value store</i> sont prévus pour stocker ces données. Généralement montés entièrement en mémoire vive (RAM), ils génèrent d'importantes économies de cycles CPU si les données calculées sont très souvent sollicitées.</p>		
Exemple		
<p>Le nombre de contenus (par exemple des produits) appartenant à une catégorie est calculé alors qu'il n'est pas mis à jour de manière très fréquente. Mettre en cache, pour chaque catégorie, son nombre de contenus permet de gagner du cycle CPU. Les jetons d'accès en OAuth2 sont associés à une date d'expiration. Mettre en cache le jeton et son délai d'expiration évite des appels inutiles au serveur d'autorisation et de devoir revalider le jeton.</p>		
Test		
<p>Le nombre de données peu volatiles, demandant un calcul, accédées plusieurs fois et non mises dans un système de cache est égal à 0.</p>		
FRONT	CONCEPTION	61

Éviter le transfert de grandes quantités de données

PRIORITÉ



MISE EN ŒUVRE



IMPACT ÉCOLOGIQUE



RESSOURCES ÉCONOMISÉES



Les systèmes de gestion de base de données sont conçus et optimisés pour répondre efficacement aux traitements de grandes quantités de données. Dans le cas de traitements avec une logique plus ou moins complexe, vous devriez les réaliser au plus près de la donnée afin de :

- limiter la bande passante en raison du transfert de données non traitées ;
- profiter des optimisations de la base de données sur la manipulation des données ;
- alléger le cycle CPU côté serveur backend voire frontend.

Exemple

Dans le cas de requêtes complexes avec un nombre important de données et de l'utilisation d'un système de gestion de base de données relationnelles (SGBDR), il est conseillé d'utiliser des procédures stockées car :

- une procédure stockée économise au serveur l'interprétation de la requête puisqu'elle est précompilée ;
 - une procédure stockée est moins gourmande en bande passante puisqu'il y a moins d'informations échangées entre le serveur et le client.
- Tous les SGBDR récents (SQL Server, MySQL, PostgreSQL, etc.) prennent en charge les procédures stockées.

Test

Le nombre de traitements avec une grande quantité de données exécutés en dehors du serveur de base de données est inférieur ou égal à 1.

Favoriser les pages statiques

Ce document est la propriété exclusive de Jean-Philippe Prost (lambda.doe@gmail.com) - jeudi 22 septembre

PRIORITÉ	MISE EN ŒUVRE	IMPACT ÉCOLOGIQUE
   	  	   

RESSOURCES ÉCONOMISÉES



Si une page n'est modifiée que deux fois par an, préférer des pages statiques, construites en dehors du CMS. Cela permet d'économiser des cycles CPU et de la bande passante.

L'utilisation d'un système de gestion de contenu dynamique requiert en effet de charger les différentes couches logicielles pour servir le contenu demandé par l'internaute : le serveur HTTP, le serveur d'applications, le système de stockage du contenu (base de données), éventuellement les systèmes de cache associés, etc. En revanche, un fichier statique est directement lu et renvoyé à l'internaute par le serveur HTTP ou le serveur de cache, sans solliciter le serveur d'applications ou la base de données.

Exemple

Voici quelques exemples d'actions à éviter pour garantir le bon fonctionnement de la navigation rapide dans l'historique :

- les actions lorsqu'on quitte la page (événements unload ou beforeunload, leur préférer pagehide si c'est vraiment nécessaire) ;
- les liens qui ouvrent de nouveaux onglets/fenêtres sans `rel="noopener"` ou `rel="noreferrer"` ;
- de laisser des connexions (IndexedDB, `fetch()` ou XMLHttpRequest, Web Sockets, etc.) ouvertes quand l'utilisateur quitte la page. Utiliser les événements pageshow et/ou pagehide pour réinitialiser les éléments qui le nécessitent.

Test

Le nombre de pages dynamiques est inférieur ou égal à 25 %.

FRONT

CONCEPTION

63

Bonne pratique n°20

Afficher des pages d'erreur statiques

PRIORITÉ

MISE EN ŒUVRE

IMPACT ÉCOLOGIQUE

RESSOURCES ÉCONOMISÉES

Les pages d'erreur (40x, 50x) doivent être les plus légères possibles et même idéalement inexistantes. En effet, lorsque le navigateur demande une ressource qui n'existe pas (image, feuille de styles CSS, fichier JavaScript, etc.) ou que le serveur renvoie une erreur, la page d'erreur renvoyée peut être plus lourde que la ressource ou la page demandée.

De plus, certains CMS exécutent leur routine de recherche de contenu (dans la base de données) pour tenter de trouver une ressource demandée. Par conséquent, du code serveur est exécuté, le serveur de base de données est sollicité et la génération dynamique de la page HTML est exécutée. Tout ce processus aboutit à un gaspillage de cycles CPU, de mémoire vive et de bande passante.

Exemple

Éviter les pages 404 dynamiques, personnalisées en fonction du contenu de l'URL. Préférer une page 404 statique.

Test

Le nombre de pages d'erreur dynamiques est inférieur ou égal à 0.

Ne se connecter à une base de données que si nécessaire

PRIORITÉ



MISE EN ŒUVRE



IMPACT ÉCOLOGIQUE



RESSOURCES ÉCONOMISÉES



Quel que soit le système de base de données, l'ouverture d'une connexion est un processus coûteux en ressources pour le client et le serveur :

- allocation de mémoire et accès disque pour les buffers ;
- allers-retours réseaux pour le protocole de connexion ;
- coût CPU induit.

La bonne pratique qui est usuellement utilisée est la mise en place d'un pool de connexions. Ce dernier permet d'optimiser la gestion des connexions ainsi que les performances. Cependant, sa configuration n'est pas forcément une opération triviale (cela nécessite de superviser le comportement de celui-ci pour trouver le bon paramétrage). Ceci dit, chaque fois que l'application peut éviter un accès à la base de données, faites-le !

Exemple

HikariCP est un pool de connexions JDBC solide et performant. Il est intégré dans SpringBoot. Dans les cas où il n'y a pas de pool de connexions, réutiliser une connexion et ne pas ouvrir/fermer une nouvelle connexion à chaque requête.

Test

Le nombre de connexions à une base de données pour requêter, stocker une donnée non nécessaire à l'utilisation du service est égal à 0.

Créer une architecture applicative modulaire

PRIORITÉ



MISE EN ŒUVRE



IMPACT ÉCOLOGIQUE



RESSOURCES ÉCONOMISÉES



L'architecture modulaire popularisée par les logiciels open source apporte souvent une plus grande capacité à monter en charge, des coûts réduits de maintenance corrective et évolutive, ainsi qu'un code plus efficient.

Si la couverture fonctionnelle du site web ou du service en ligne peut être amenée à évoluer, mieux vaut implémenter les fonctionnalités de base dans un noyau et les compléter au besoin par des modules. Ces derniers peuvent rassembler des fonctions appartenant à un même domaine métier. Cela permet de les développer indépendamment des autres domaines métiers, ainsi que les partager à d'autres applications. Cette approche est valable à tous les niveaux de granularité, pour un développement sur-mesure comme pour le choix d'un serveur HTTP ou d'un CMS.

Exemple

Les logiciels open source les plus efficaces, comme Nginx, Apache, MySQL ou PHP, reposent sur cette architecture modulaire. Côté backend, le découpage en microservices permet d'apporter un niveau de modularité pour des services HTTP. Il faudra néanmoins porter une attention particulière sur la granularité du découpage pour éviter un effet contre-productif (ajout d'une complexité d'interface entre les services, augmentation globale des ressources informatiques).

Test

Le nombre d'architectures non modulaires est égal à 0.

[BACK](#)
[CONCEPTION](#)

73

Utiliser la version la plus récente du langage

PRIORITÉ



MISE EN ŒUVRE



IMPACT ÉCOLOGIQUE



RESSOURCES ÉCONOMISÉES



Les différents langages côté serveur (PHP, Ruby, Java, etc.) sont régulièrement améliorés par les différentes communautés. Chaque nouvelle version a régulièrement apporté son lot de gains en matière de performances, de gestion de mémoire, de stabilité et comble des failles de sécurité.

Il est donc conseillé d'utiliser la version la plus récente du langage pour bénéficier de ses apports.

Exemple

Les versions 7 et 8 de PHP ont apporté un gros gain de performance. De plus, avec le JIT (*Compilation Just in Time*), PHP 8.0 peut compiler des parties du code et le stocker en mémoire. En plus d'améliorer les performances, le JIT optimise la gestion mémoire.

Test

Le nombre de versions majeures de retard sur la dernière version stable du langage est inférieur ou égal à 1.

document est la propriété exclusive de Jean-Philippe Prost (lambda.doe@gmail.com) - jeudi 22 septembre 2022 16:06

Bonne pratique n°30

Fournir une alternative textuelle aux contenus multimédias

PRIORITÉ	MISE EN ŒUVRE	IMPACT ÉCOLOGIQUE
   	   	    
RESSOURCES ÉCONOMISÉES 		

Le texte, même mis en forme en HTML/CSS, utilise beaucoup moins de bande passante que des formats multimédias comme l'audio ou la vidéo. Fournir aux utilisateurs une alternative textuelle à ces contenus leur permet, s'ils le souhaitent, de lire plutôt que d'écouter ou de visionner, et donc de transférer moins de données.

Si cette alternative textuelle a elle-même une taille importante, elle peut ne pas être chargée par défaut, mais suite à une action utilisateur. Cette pratique est également bénéfique pour l'accessibilité : les malentendants pourront lire le contenu et y auront donc accès, de même pour les malvoyants, si le texte inclut une description des éléments vidéo qui ne sont que visibles.

Cette pratique est également bénéfique pour le référencement, les moteurs de recherche pouvant plus facilement analyser le texte que l'audio et la vidéo (voir aussi « Éviter la lecture automatique des vidéos et des sons », page 156).

Exemple

Une vidéo de 30 minutes va généralement avoir un poids de 500 Mo, un podcast de la même durée fera 30 Mo et l'équivalent texte moins d'1 Mo.

Test

Le nombre de fichiers multimédias sans alternative textuelle est inférieur ou égal à 10 %.

Bonne pratique n°31

Fournir une CSS print

PRIORITÉ	MISE EN ŒUVRE	IMPACT ÉCOLOGIQUE
  	   	  
RESSOURCES ÉCONOMISÉES 		

Outre le service apporté à l'internaute, cette feuille de styles réduit le nombre de pages imprimées, et donc indirectement l'empreinte écologique du site web. La plus dépouillée possible, cette feuille de styles masque le header, le footer, le menu, la sidebar et supprime toutes les images sauf celles du contenu, etc.

Exemple

Cette CSS print « nettoie » la page affichée à l'écran afin de proposer une impression épurée :

```
body {
    background-color : #fff; font-family : Serif; font-size : 15pt;
}
#page {
    margin : 0;
    border : none;
}
#banner, #menuright, #footer {
    display : none;
}
h1#top {
    margin : 0;
    padding : 0;
    text-indent : 0; line-height : 25pt; font-size : 25pt;
} (...)
```

Test

Le nombre de CSS print manquantes est inférieur ou égal à 1.

document est la propriété exclusive de Jean-Philippe Prost (lambda.doe@gmail.com) - jeudi 22 septembre 2022 16:06

Fournir une alternative textuelle aux contenus multimédias



PRIORITY



MISE EN ŒUVRE



IMPACT ÉCOLOGIQUE

RESSOURCES ÉCONOMISÉES



Le texte, même mis en forme en HTML/CSS, utilise beaucoup moins de bande passante que des formats multimédias comme l'audio ou la vidéo. Fournir aux utilisateurs une alternative textuelle à ces contenus leur permet, s'ils le souhaitent, de lire plutôt que d'écouter ou de visionner, et donc de transférer moins de données.

Si cette alternative textuelle a elle-même une taille importante, elle peut ne pas être chargée par défaut, mais suite à une action utilisateur. Cette pratique est également bénéfique pour l'accessibilité : les malentendants pourront lire le contenu et y auront donc accès, de même pour les malvoyants, si le texte inclut une description des éléments vidéo qui ne sont que visibles.

Cette pratique est également bénéfique pour le référencement, les moteurs de recherche pouvant plus facilement analyser le texte que l'audio et la vidéo (voir aussi « Éviter la lecture automatique des vidéos et des sons », page 156).

Exemple

Une vidéo de 30 minutes va généralement avoir un poids de 500 Mo, un podcast de la même durée fera 30 Mo et l'équivalent texte moins d'1 Mo.

Test

Le nombre de fichiers multimédias sans alternative textuelle est inférieur ou égal à 10 %.

Fournir une CSS print



PRIORITY



MISE EN ŒUVRE



IMPACT ÉCOLOGIQUE

RESSOURCES ÉCONOMISÉES



Outre le service apporté à l'internaute, cette feuille de styles réduit le nombre de pages imprimées, et donc indirectement l'empreinte écologique du site web. La plus dépolluée possible, cette feuille de styles masque le header, le footer, le menu, la sidebar et supprime toutes les images sauf celles du contenu, etc.

Exemple

Cette CSS print « nettoie » la page affichée à l'écran afin de proposer une impression épurée :

```
body {
  background-color : #fff; font-family : Serif; font-size : 15pt;
}
#page {
  margin : 0;
  border : none;
}
#banner, #menuright, #footer {
  display : none;
}
h1#top {
  margin : 0;
  padding : 0;
  text-indent : 0; line-height : 25pt; font-size : 25pt;
} (...)
```

Test

Le nombre de CSS print manquantes est inférieur ou égal à 1.

Favoriser les polices standards

PRIORITÉ	MISE EN ŒUVRE	IMPACT ÉCOLOGIQUE
   	  	   
RESSOURCES ÉCONOMISÉES   		

Préférez les polices standards, déjà présentes sur l'ordinateur de l'utilisateur. Elles sont listées sur le site <https://systemfontstack.com/>. Si vous ajoutez une police particulière pour votre service numérique, même si c'est une API font (Google Fonts, Adobe Fonts, etc.), vous devez :

- héberger vous-même la font ;
- utiliser un faux-gras via la propriété CSS `-webkit-text-stroke` ;
- préférer une variable font si vous avez besoin de plus d'une font ;
- optimiser la font en allégeant des caractères non nécessaires.

Exemple

Ici la font Montserrat n'est téléchargée que si l'utilisateur n'a pas défini de préférence. A contrario, la font Arial sera utilisée sans téléchargement de Montserrat.

```
@media (prefers-reduced-data: no-preference) {
  @font-face {
    font-family: Montserrat;
    font-style: normal;
    font-weight: 400;
    src: url('fonts/montserrat-latin-regular.
wo#2');
  }
}
body {
  font-family: Montserrat, Arial;
}
```

Test

Le nombre de polices téléchargées est inférieur ou égal à 2.

Ne pas afficher les documents à l'intérieur des pages

PRIORITÉ	MISE EN ŒUVRE	IMPACT ÉCOLOGIQUE
  		  
RESSOURCES ÉCONOMISÉES 		

Certains rédacteurs de contenu éprouvent parfois le besoin d'intégrer des documents à l'intérieur des pages, en visualisation directe. Cela nécessite un développement spécifique pour une fonctionnalité malvenue.

En effet, le chargement d'un document, à l'intérieur d'un contenu, est non voulu par l'utilisateur si son système en permet l'affichage. De plus, à chaque consultation de la page, le fichier est téléchargé pour être affiché.

Exemple

Pour être affiché, un fichier de traitement de texte devra, par exemple, appeler un logiciel adapté. Or si ce logiciel n'est pas installé sur le poste de l'utilisateur, le fichier ne pourra pas être lu sans un développement spécifique coûteux. Il est donc préférable d'insérer un lien de téléchargement du document au sein de votre page afin que seuls les utilisateurs concernés le téléchargent.

Test

Le nombre de documents affichés dans un contenu est égal à 0.

Limiter le nombre de CSS

PRIORITÉ	MISE EN ŒUVRE	IMPACT ÉCOLOGIQUE
   	   	   
RESSOURCES ÉCONOMISÉES  		

Nous préconisons de limiter le nombre de CSS pour ne pas multiplier les requêtes HTTP et pour simplifier le rendu côté navigateur. Utilisez une feuille de styles commune pour tous les éléments communs, quel que soit l'affichage, et un fichier par résolution cible ou media query. Attention, il faut intégrer ces fichiers directement dans le code HTML et non avec des directives `@import` dans les fichiers CSS eux-mêmes. En découpant de la sorte, le terminal de l'utilisateur choisira ce qui lui correspond et dépriorisera le chargement des fichiers dont il n'a pas besoin. Le cas échéant, des fichiers CSS pour certains composants peuvent être chargés en fonction du contexte.

Exemple

```
<link rel='stylesheet' id='css-css' href='communs.css' type='text/css' media='all' />
<link rel='stylesheet' id='css-xs-css' href='petits-ecrans.css' type='text/css' media='(max-width: 959px)' />
<link rel='stylesheet' id='css-sm-css' href='tablettes.css' type='text/css' media='(min-width: 768px)' />
<link rel='stylesheet' id='css-mdlg-css' href='grands-ecrans.css' media='(min-width: 960px)' />
<link rel='stylesheet' id='css-print-css' href='print.css' type='text/css' media='print' />
```

Test

Le nombre de fichiers CSS est inférieur ou égal à 10.

FRONT	RÉALISATION	81
-------	-------------	----

Découper les CSS

PRIORITÉ	MISE EN ŒUVRE	IMPACT ÉCOLOGIQUE
   	   	   
RESSOURCES ÉCONOMISÉES  		

Employez un ensemble de CSS plutôt qu'une seule et appelez uniquement les CSS utiles en fonction du contexte. Cette méthode permet de limiter le poids de la page lors du premier téléchargement, donc d'économiser de la bande passante et de réduire la charge CPU.

Exemple

Découper les CSS en fonction de la logique fonctionnelle :

- layout ;
- content ;
- module x ;
- module y ;
- etc.

Dans le cas d'un site fonctionnellement riche, cela permettra d'exclure toutes les CSS des modules non utilisés. Le nombre de CSS doit rester raisonnable, plus pour des questions de maintenabilité (profit) que de réduction d'impacts environnementaux (planet), dans la mesure où les CSS générales (layout et content dans notre exemple) seront concaténées en un seul fichier. Les CSS complémentaires (ici, module x et module y) seront téléchargées en fonction du contexte (page, fonctionnalités...).

Test

Le nombre de bibliothèques CSS est compris entre 2 et 5.

82	RÉALISATION	FRONT
----	-------------	-------

Remplacer les boutons officiels de partage des réseaux sociaux

PRIORITÉ	MISE EN ŒUVRE	IMPACT ÉCOLOGIQUE
   	   	   
RESSOURCES ÉCONOMISÉES  		

Les principaux réseaux sociaux tels que Facebook, Twitter, Pinterest, etc., fournissent des plug-ins à installer sur une page web pour y afficher un bouton « Partager » et un compteur de « J'aime ». Ces bibliothèques JavaScript sont souvent lourdes à télécharger et génèrent beaucoup de requêtes. **Préférez des liens directs, en HTML, vers les pages de partage.** On peut générer ces liens à la main (voir ci-dessous) ou via un outil tel que Share Link Generator (<https://www.sharelinkgenerator.com/>).

Exemple

On peut ajouter un bouton qui ouvre un pop-up de partage comme le font les boutons officiels, par exemple avec le code suivant :

```
html<button type = "button" onclick = "window.
open('https://www.facebook.com/ sharer/sharer.
php?u=XXXXX', '', 'menubar = no, toolbar = no,
resizable = yes, scrollbars = yes, height = 500,
width = 700')">Je partage cette page sur Facebook</
button>
```

Pour aller plus loin, consultez la page : <https://www.nuweb.fr/736>.

Test

Le nombre de bibliothèques externes est égal à 0.

FRONT

RÉALISATION 105

Valider les pages auprès du W3C

PRIORITÉ	MISE EN ŒUVRE	IMPACT ÉCOLOGIQUE
  	    	 
RESSOURCES ÉCONOMISÉES 		

Vérifier que le code HTML des pages est bien formé. Dans le cas contraire, le navigateur corrigera dynamiquement un certain nombre d'éléments pour afficher au mieux les pages posant problème. Ces corrections dynamiques consomment inutilement des ressources à chaque chargement des pages concernées.

Exemple

Utiliser le validateur du W3C (World Wide Web Consortium) pour vérifier que les pages sont bien valides et que le code HTML est correctement formé : <https://validator.w3.org> ou <https://validator.w3.org/nu/> pour filtrer les erreurs.

Test

Le nombre d'erreurs critiques est égal à 0.

FRONT




RÉALISATION 107

document est la propriété exclusive de Jean-Philippe Prost (lambda.doe@gmail.com) - jeudi 22 septembre





Bonne pratique n°62

Optimiser la taille des cookies




PRIORITÉ




MISE EN ŒUVRE



IMPACT ÉCOLOGIQUE



RESSOURCES ÉCONOMISÉES



Un cookie permet de maintenir un état entre le navigateur de l'internaute et le serveur web distant grâce à une donnée partagée. Le cookie est une chaîne de caractères transférée dans chaque requête HTTP. Il faut donc optimiser au maximum sa taille et la supprimer dès que sa présence n'est plus obligatoire.

Exemple

On peut supprimer un cookie lorsqu'il n'est plus utile en précisant une durée d'expiration nulle ou négative, de la manière suivante :

```
|| Set-Cookie: user_mvariable=; Max-Age=0
```

Voir la RFC 6265 de l'IETF (*Internet Engineering Task Force*) pour en savoir plus sur les cookies : <https://datatracker.ietf.org/doc/html/rfc6265>.





Test

Le nombre de cookies qui ne sont plus utiles ou sont non optimisés est égal à 0.





Bonne pratique n°63

Choisir un format de données adapté





PRIORITÉ







MISE EN ŒUVRE



IMPACT ÉCOLOGIQUE



RESSOURCES ÉCONOMISÉES



Le type de données utilisé pour manipuler et stocker une donnée a un impact significatif sur la consommation mémoire et les cycles processeurs nécessaires lors des manipulations en base de données, au niveau du serveur d'applications et même dans le navigateur (manipulation via JavaScript), ainsi que sur l'espace de stockage nécessaire. Choisir un mauvais type de données entraîne un gaspillage de mémoire (par exemple, si vous stockez de toutes petites données dans une colonne prévue pour stocker de grosses quantités de données) et des problèmes de performance. Les choix du type de données et de son dimensionnement doivent donc être fondés sur l'analyse d'un échantillon représentatif de données.

Exemple

Dans le cas d'un établissement de formation, la taille du champ permettant de stocker le nombre d'élèves doit être basé sur une étude statistique. On peut ainsi déterminer s'il est possible d'utiliser un TINYINT (1 octet, jusqu'à 127) plutôt qu'un SMALLINT (2 octets, jusqu'à 32 767). Dans tous les cas, le choix par défaut d'un INT (4 octets, jusqu'à 2 147 483 647) voire un BIGINT (8 octets) est une aberration (que nous rencontrons malheureusement tous les jours lors de nos audits...).

Gain potentiel : jusqu'à 8 fois moins de stockage. La consommation en cycles processeurs est réduite dans les mêmes proportions.

Test

Le nombre de champs de la base dont le format est inadéquat est inférieur ou égal à 15 %.

document est la propriété exclusive de Jean-Philippe Prost (lambda.doe@gmail.com) - jeudi 22 septembre

Stocker les données statiques localement

PRIORITÉ    	MISE EN ŒUVRE   	IMPACT ÉCOLOGIQUE    
RESSOURCES ÉCONOMISÉES  		

Avec le support désormais généralisé sur tous les navigateurs des bases de données clé-valeur (IndexedDB, Web Storage), et la mise en cache dans le Cache Storage API, il est possible de stocker localement des données structurées statiques.

L'intérêt du stockage local est double. D'une part, on évite les allers-retours inutiles avec le serveur, ce qui économise des ressources et du temps de réponse. D'autre part, comme les données sont locales, il est plus facile et plus rapide de les manipuler au sein de l'interface. Le gain potentiel est la réduction de la charge serveur, donc du nombre d'équipements nécessaires (de leur empreinte environnementale et économique), des serveurs HTTP jusqu'aux serveurs de base de données.

Exemple

Par exemple pour stocker le nom de l'utilisateur vous pouvez utiliser l'instruction suivante :

```
localStorage.setItem('name', 'nom_utilisateur');
```

Il suffit ensuite d'utiliser les deux lignes suivante pour le retrouver ultérieurement :

```
var myName = localStorage.getItem('name');  
myName
```

Test

Le nombre de données statiques non stockées localement est inférieur ou égal à 25 %.

Bien choisir son thème et limiter les extensions dans un CMS

PRIORITÉ  	MISE EN ŒUVRE  	IMPACT ÉCOLOGIQUE   
RESSOURCES ÉCONOMISÉES  		

Lors de l'utilisation d'un CMS, le choix du thème est primordial. Lorsque vous installez ce dernier, des extensions sont ajoutées, parfois même un constructeur de pages. Les fonctionnalités fournies dépassent souvent vos besoins. Chaque extension ajoute ses données en base, ses fichiers CSS et JavaScript, sans compter le nombre de fichiers stockés sur votre serveur pour faire fonctionner le tout.

Selon la qualité de développement, les assets supplémentaires peuvent être lourds et/ou redondants. Cela provoque une :

- augmentation du nombre de requêtes :
- augmentation du poids des pages :
- augmentation du temps de traitement des requêtes internes :
- complexification de l'administration de vos contenus.

Exemple

Un thème tout-en-un ou qui intègre un constructeur de pages peut ajouter une fonctionnalité pour mettre des carrousels dans vos articles ou encore ultra personnaliser vos boutons d'action. Il peut aussi intégrer des témoignages ou créer un portfolio alors que vous n'avez pas du tout prévu d'en intégrer dans vos contenus. Attention également aux incompatibilités de certaines extensions. Préférez un thème simple contenant l'essentiel, puis ajoutez une fonctionnalité réfléchie lorsqu'elle est nécessaire.

Test

Le nombre d'extensions est inférieur ou égal à 10.