

INF5153 – Génie logiciel : conception

Conception concurrente

Jacques Berger

# Objectifs

Introduire les problèmes de conception liés à la concurrence

# Prérequis

Patrons de conception

# Copies

Il existe 2 façons de copier une instance d'un objet :

- deep-copy

- shallow-copy

# Copies

## Deep-copy

On crée une nouvelle instance pour chaque membre qui n'est pas d'un type natif

Copie complète de l'objet et de ses membres

# Copies

## Shallow-copy

On copie les membres de type natif; pour les objets, uniquement les références des membres sont copiées dans la nouvelle instance

Copie superficielle de l'objet

# Threads

Les threads sont des séquences d'instructions qui sont exécutées en parallèle

Les threads partagent la même mémoire, et d'autres ressources, ce qui entraîne des problèmes de concurrence

# Concurrence

La concurrence survient lorsque 2 threads veulent accéder à la même ressource mais qu'un seul thread ne peut y accéder à la fois

La concurrence aura un impact sur la conception d'un logiciel



# Environnement

## Single-thread

Un seul thread existant

## Multi-thread

Plusieurs threads en parallèle, le nombre peut varier durant l'exécution

# Conception

Dans un environnement single-thread, il existe un couplage temporel entre les instructions

Dans un environnement multi-thread, le moment où est exécuté une instruction est beaucoup moins prévisible, il faut donc découpler (d'un point de vue temporel) une série d'instructions avec une autre

# Conception

Implique souvent un changement majeur dans la conception d'un logiciel

# Thread-safe

Le thread-safety devient une caractéristique importante pour notre conception et notre réalisation

Dans le code concurrent, il faut utiliser des classes et des fonctions thread-safes

Sécuritaires dans un environnement multi-thread

# SRP

## Single Responsibility Principle

Gérer la concurrence, c'est une responsabilité en soi

Le code de gestion de concurrence devrait être dans un objet distinct

# Encapsulation

Toujours utiliser des variables avec la plus petite portée possible

Plus la portée est grande, plus le risque de concurrence est élevé

Bien encapsuler les données et limiter l'accès aux données partagées

# Copies

Lorsque des données doivent être partagées d'une classe à une autre, faire des copies des données

Évite les modifications concurrentes

On favorise le deep-copy dans ce contexte

# Indépendance

Les threads peuvent communiquer entre eux et se synchroniser

Pour communiquer, ils doivent partager des données et ce partage entraîne un problème de concurrence potentiel

Autant que possible, éviter la communication entre threads



# Thread-safe

Vos classes et fonctions utilisées dans des threads doivent être thread-safe

Les bibliothèques utilisées dans les threads doivent être thread-safe

Cette information est généralement présente dans la documentation des classes

Documentez vos classes en conséquence

# Thread-safe

Faire des fonctions réentrantes

Éviter les données statiques ou globales

# Mutabilité

Un objet immuable est un objet qu'on ne peut pas modifier après son instantiation

Si on désire le modifier, il faut obligatoirement créer une nouvelle instance

# Mutabilité

Exemple de la classe String en Java

Un objet immuable est automatiquement thread-safe

# Fonctions synchronisées

Les fonctions qui tentent de se synchroniser avec d'autres threads devraient être aussi courtes que possible

# Fonctions synchronisées

Mot-clé synchronized en Java

Implémente un mutex

Accès en série

# Fonctions synchronisées

Peut provoquer des blocages à l'exécution si le positionnement des blocs synchronisés n'est pas bien réfléchi

La conception est importante pour prévenir ces problèmes difficiles à réparer

# Patron

Producteur-consommateur

Des threads produisent des tâches qui sont placées dans une liste

Des threads consomment ces tâches et les réalisent, libérant de l'espace dans la liste

La taille de la liste est bornée



# Pool

Dans certains langages de programmation, comme le Java, la création d'un thread est une opération coûteuse en temps d'exécution et en mémoire

Afin d'optimiser ce processus, on tente de réutiliser les threads une fois leur tâche complétée

# Pool

Un pool de threads est un contenant où l'on place les threads inactifs

Lorsqu'on doit exécuter une tâche en parallèle, on prend un thread dans le pool et il exécute la tâche

Lorsque la tâche est terminée, le thread retourne dans le pool

# Plus loin...

Clean Code  
Robert C. Martin  
Prentice-Hall, 2008  
Chapitre 13