EL AYADI OTHMANE 3B1

AMPHI 11 CPOA

PROGRAMATION ORIENTEE DONNEES

&

C++ MODERNE

Object-Relationel Mapping :

Il y a plusieurs sortes de données :

-Données dans une base données

-Données dans chaque langage (exemple : une variable, un objet etc... Ce sont des = données)

La différence :

-Java = Objet

-Base de donnes = relations

Faire des requêtes ou exécuter des requêtes en Java « pure » :

Lire les cases une par une puis mettre dans une variable le resultat et ensuite appeler un constructeur d'une classe etc... = bcp à écrire

Librairie pour faciliter :

➔ Relation entre objet et relation Object Relationel Mapping.

Hibernate :

➔ Annoter chaque champ que vous déclarez dans une bd avec le nom d'une colonne dans une bd ==> explicite la relation entre une classe et une table.

HQL :

➔ Hibernate Query Language

Langage de requête qui reprend le princpie de la lib Hibernate. Ce langage totalement orientée objet comprenant des notions d'héritage, de polymorphisme et d'association.

Remarque : Le temps perdu

Base de données très bien mais dans une base de données il y a des problèmes un étudiant dans une bd correspond à une ligne dans la base de données une ligne est donc une place

| ETUD |

| id | nom | diplôme |

Structure de case chaque case on a des données et pour accéder à une case (pointeur) on nomme la case qu'on veut mais pas la donnée

Différence : PLOP (PLace Oriented Programming) pas des donnée mais d'une place exemple : etud.getNom()

Distance importante entre code et données -> en java on ne peut pas écrire un littéral un étudiant il faut passer par un constructeur. (Pas comme JSon à voir dans le chapitre suivant)

```
Map stuff = new HashMap();

stuff.put("a", 1);

stuff.put("b", 2); {"a": 1, "b": 2}
```

.java

Place -> Stuff Données {'a':1 .....}

Une valeur pure n'as pas de mutation possible -> 42 = 42 même dans le temps elle ne changera pas dans le temps

| ETUD |

|42|TOM|DUT|

➔ Modification du diplôme

| ETUD |

|42|TOM|Licence|

! Ici on change la référence se seras la seule chose qui dois muter !

La référence change mais pas la donnée on recréer donc une donnée

➔ Principe d'immuabilité

VALEUR PURES :

Toute propriété pour une valeur pure alors elle restera la même peu importe le temps.

Exemple :

42 = 42 même dans le temps elle ne changera pas !

Avantage :

Plus facile à tester suffit de tester 42 une fois pas la peine de tester plusieurs fois.

Test génératif c'est un jeu de test ou on demande à notre machine de créer

Automatiquement les jeux de test

C'est-à-dire : Générer des valeurs aléatoires et pas des mutations possibles aléatoire ou des donnes aléatoires

JSON

JSON-> java script object notation

Expression littérale d'un objet pas de constructeur différence de Java

Sert a faire communiquer des donnes d'une Machine A a une machine B

HTML -> d'une machine -> humain

Permet d'avoir des valeurs pures

```
{
 "type": "success",
 "value":
 {
 "id": 166,
 "joke": "Chuck Norris doesn't play god.
 Playing is for children.",
 "categories": []
 }
}
```

.json

C++ MODERNE :

"moderne" = C++11, C++14, C++17

-> standard

=> Adaptation par le compilateur mais pas communauté.

Innovation -> travailler avec des objets

Auto

➔ Déduction automatique d'un type avec auto

Exemple :

```
Auto i = 10 ; // -> déduction d'un int

Auto d = 5.5 ;  //-> déduction d'un double

Auto s = "toto"   //-> déduction d'un string
```

New ? :

SmartPointer-> pas besoin de détruire à la main des données

Méthode ancienne :

```
cercle* p = new cercle( 42 );
vector<cercle*> v = load_cercles();
. . .
Delete p
```

Chaque new il faut faire un delete à la fin de l'utilisation de l'objet.

Méthode moderne :

```
auto p = make_shared<cercle>( 42 );
vector< shared_ptr<cercle> > v = load_cercles2();
```
.cpp

make_shared-> on créer quelque chose sur le tas mais pas sur la pile c'est que l'on veux

Partager avec quelqu'un d'autre mais garder l'adresse en mémoire même si la fonction est terminer

On créer alors un sharedPointer de type smartPointer

-> Son principe est d'encapsuler un pointeur et garder la référence, on n'a pas besoins de mettre de delete à la fin de la fonction.

STL->

Standard template librairie => contient des collections (HashMapp, List, Vector).

Lambdas->

Définir un objet de fonction anonyme à l'emplacement où il est appelé ou passé comme argument à une fonction.

Les expressions lambda sont généralement utilisées pour encapsuler quelques lignes de code qui sont passées à des algorithmes ou méthodes asynchrones