UTOPIOS
TRANSFORMATION DIGITALE INTELLIGENTE

QT5 C++

Présentation QT5

- Qt est une bibliothèque logicielle orientée objet (API) développée en C++ par Qt Development Frameworks, filiale de Digia.
- Qt fournit un ensemble de classes décrivant des éléments graphiques (widgets) et des éléments non graphiques : accès aux données (fichier, base de données), connexions réseaux (socket), gestion du multitâche (thread), XML, etc
- Qt permet la portabilité des applications (qui n'utilisent que ses composants) par simple recompilation du code source. Les environnements supportés sont les Unix (dont Linux), Windows et Mac OS X

Présentation QT5

- Qt est principalement dédiée au développement d'interfaces graphiques en fournissant des éléments prédéfinis appelés widgets (pour windows gadgets).
- Les widgets peuvent être utilisés pour créer ses propres fenêtres et boîtes de dialogue complètement prédéfinies (ouverture/enregistrement de fichiers, progression d'opération, etc).
- Les interactions avec l'utilisateur sont gérées par un mécanisme appelé signal/slot. Ce mécanisme est la base de la programmation événementielle des applications basées sur Qt
- Le programme sera principalement défini par ses réactions aux différents événements qui peuvent se produire, c'est-à-dire des changements d'état, par exemple l'incrémentation d'une liste, un mouvement de souris ou un appui sur une touche de clavier etc.
- La programmation événementielle est architecturée autour d'une boucle principale fournie et divisée en deux sections : la première section détecte les événements, la seconde les gère.
- Pour chaque évènement à gérer, il faut lui associer une action à réaliser (le code d'une fonction ou méthode) : c'est le gestionnaire d'évènement (handler)

Présentation QT5 API

- L'API Qt est constituée de classes aux noms préfixés par un Q et dont chaque mot commence par une majuscule (QLineEdit, QLabel, ...).
- L'ensemble des classes est basé sur l'héritage. La classe QObject est la classe mère de toutes les classes Qt.
- En dérivant de QObject, un certain nombre de spécificités Qt sont hérités, notamment : le mécanisme signal/slot pour la communication entre objets, une gestion simplifiée de la mémoire, etc.
- Les objets Qt (ceux héritant de QObject) peuvent s'organiser d'eux-mêmes sous forme d'arbre d'objets. Ainsi, lorsqu'une classe est instanciée, on peut lui définir un objet parent.
- Pour bénéficier des spécificités de Qt, il faudra hériter de QObject ou d'une classe fille de QObject

Présentation QT5 Modules

- Depuis la version 4, Qt sépare sa bibliothèque en modules. Un module regroupe un ensemble de classes Qt. Les principaux modules :
  - QtCore : pour les fonctionnalités non graphiques utilisées par les autres modules.
  - QtWidgets : pour les composants graphiques.
  - QtNetwork : pour la programmation réseau.
  - QtSql : pour l'utilisation de base de données SQL.
  - QtXml : pour la manipulation et la génération de fichiers XML.

Présentation QT5 IDE

- Qt Creator est l'environnement de développement intégré dédié à Qt et facilite la gestion d'un projet Qt. Son éditeur de texte offre les principales fonctions que sont la coloration syntaxique, le complètement, l'indentation, etc... Qt Creator intègre en son sein les outils Qt Designer et Qt Assistant. Il intègre aussi un mode débuggage.
- JetBrains CLion

La classe QApplication

- Pour créer une application Qt graphique (GUI), il faut instancier un objet de la classe QApplication (et QCoreApplication pour les applications non graphiques) :
  - La classe QApplication (qui hérite de QCoreApplication) fournit une boucle principale d'événements pour les applications Qt : tous les événements du système seront traités et expédiés.
  - C'est sa méthode exec() qui exécute la boucle d'attente des événements jusqu'à la fermeture du dernier objet de l'application.
  - La classe QApplication gère également l'initialisation et la finalisation de l'application, ainsi que ses paramètres.
  - L'instance de QApplication doit être créée avant tout objet graphique.

La classe QWidget

- La classe QWidget fournit la capacité de base d'affichage à l'écran et de gestion des événements. Elle est la classe mère de toutes les classes servant à réaliser des interfaces graphiques.

- Les widgets :
  - sont crées "cachés" (hide/show)
  - sont capable de se "peindre" (paint/repaint/update)
  - sont capable de recevoir les évènements souris, clavier
  - sont tous rectangulaires (x,y,width,height)
  - sont initialisés par défaut en coordonnées 0,0
  - sont ordonnés suivant l'axe z (la profondeur)
  - peuvent avoir un widget parent et des widgets enfants

Les Widgets prédéfinis

- Qt fournit des widgets prédéfinis permettant de composer ses propres applications graphiques
  - QLabel
  - QLineEdit
  - QTextEdit
  - QSpinBox
  - QDateEdit, QtimeEdit, QDateTimeEdit
  - QGroupBox, QRadioButton
  - QCheckBox
  - QPushButton
  - QMessageBox

  ...

Exercice 1

- Le but de cet exercice est de réaliser la partie graphique d'une application qui permet de créer un formulaire Pour ajouter des contacts.
- Chaque contact possède un nom, prénom, age, téléphone, age

Positionnement des widgets

Vous pouvez gérer le placement vos widgets de deux manières :

- avec un positionnement absolu (généralement déconseillé car il pose des problèmes de résolution d'écran, de redimensionnement, ...)
- avec un positionnement relatif

Qt fournit un système de disposition (layout) pour l'organisation et le positionnement automatique des widgets enfants dans un widget. Ce gestionnaire de placement permet l'agencement facile et le bon usage de l'espace disponible

Les classes QLayout

Qt inclut un ensemble de classes QxxxLayout qui sont utilisés pour décrire la façon dont les widgets sont disposés dans l'interface utilisateur d'une application
Toutes les classes héritent de la classe abstraite QLayout.

- QHBoxLayout : boîte horizontale
- QVBoxLayout : boîte verticale
- QGridLayout : grille
- QFormLayout : formulaire

Les classes QLayout

Les gestionnaires de disposition (les classes QxxxLayout) simplifient le travail de positionnement

- on peut ajouter des widgets dans un layout

```
void QLayout::addWidget(QWidget *widget)
```

- on peut ajouter des layouts dans un layout

```
void QLayout::addLayout(QLayout *layout)
```

- on peut associer un layout à un widget qui devient alors le propriétaire du layout et parent des widgets inclus dans le layout

```
void QWidget::setLayout (QLayout *layout)
```

Exercice 2

- En utilisant les différents layout, structurer le formulaire d'ajout de contact.

signal / Slot

La programmation évènementielle des applications Qt est basée sur un mécanisme appelé signal /slot :

- un signal est émis lorsqu'un événement particulier se produit. Les classes de Qt possèdent de nombreux signaux prédéfinis mais vous pouvez aussi hériter de ces classes et leur ajouter vos propres signaux.
- un slot est une fonction qui va être appelée en réponse à un signal particulier. De même, les classes de Qt possèdent de nombreux slots prédéfinis, mais il est très courant d'hériter de ces classes et de créer ses propres slots afin de gérer les signaux qui vous intéressent.
- L'association d'un signal à un slot est réalisée par une connexion avec l'appel connect()
- Les signaux et les slots sont faiblement couplés : une classe qui émet un signal ne sait pas (et ne se soucie pas de) quels slots vont recevoir ce signal. De la même façon, un objet interceptant un signal ne sait pas quel objet a émis le signal
- Une connexion signal/slot peut être supprimée par la méthode disconnect()

signal / Slot

Il est possible de créer ses propres signaux et slots.
Pour déclarer un signal personnalisé, on utilise le mot clé signals dans la déclaration de la classe et il faut savoir qu'un signal n'a :

- pas de valeur de retour (donc void)
- pas de définition de la méthode (donc pas de corps {})
  Pour émettre un signal, on utilise emit :

```
emit nomDuSignal( parametreDuSignal );
```

Un signal peut être connecté à un autre signal. Dans ce cas, lorsque le premier signal est émis, il entraîne l'émission du second.
L'émission d'un signal peut être "automatique" par exemple lorsqu'on appui sur un bouton, le signal existe déjà dans la classe utilisée
(QPushButton).

signal / Slot

Les slots personnalisés se déclarent et se définissent comme des méthodes private, protected ou public.
On utilise le mot clé slots dans la déclaration de la classe. Les slots étant des méthodes normales, ils peuvent être appelés directement comme toute méthode.

- Un signal peut être connecté à plusieurs slots. Attention : les slots seront activés dans un ordre arbitraire.
- Plusieurs signaux peuvent être connectés à un seul slot

Exercice 3

- Dans la continuté de l'exercice 2, on shouaite afficher le resulat du formulaire dans une boite de dialogue à la validation du formulaire.

Les Widgets pour afficher des collections

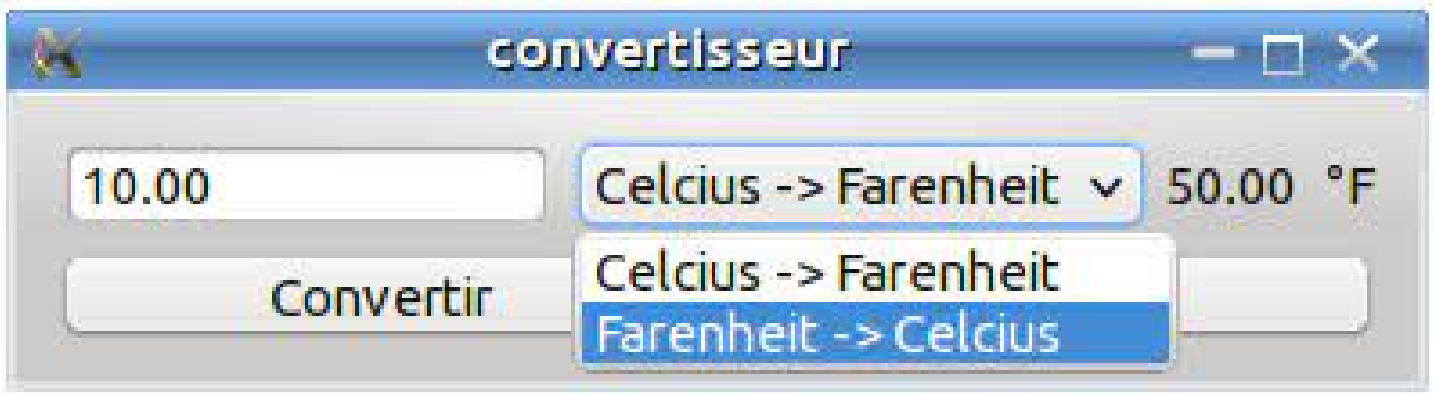
- Qt fournit des widgets prédéfinis pour afficher des collections :
- QListView
- QTreeView
- QTableView

Exercice 4

- Afficher la liste des contacts dans un widget de collection.
- Ajouter un bouton pour supprimer un contact de la collection.

Exercice Rappel

- Réaliser une application de convertisseur comme Celcius -> Farenheit.



Le composant QTSQL

- Qt SQL est un module de Qt, une bibliothèque de programmation C++, qui fournit une interface abstraite vers les bases de données SQL.
- Qt SQL permet d'interagir avec les bases de données SQL de plusieurs manières:
  - QSqlQuery : C'est la classe principale que vous utiliserez pour exécuter des requêtes SQL et parcourir les résultats. Vous pouvez l'utiliser pour exécuter des commandes SQL directes ou pour préparer et exécuter des requêtes paramétrées.
  - QSqlDatabase : Cette classe représente une connexion à une base de données. Vous pouvez l'utiliser pour ouvrir et fermer des connexions, commencer et terminer des transactions, et accéder à des informations sur la base de données.
  - QSqlDriver : Cette classe est une interface abstraite pour les pilotes de base de données SQL. Chaque type de base de données a son propre pilote qui implémente cette interface. Vous n'interagirez généralement pas directement avec cette classe, mais elle est utile pour comprendre comment Qt SQL fonctionne en interne.
  - QSqlError : Cette classe contient des informations sur une erreur qui s'est produite lors de l'exécution d'une opération SQL.

Le composant QTSQL

- Qt SQL prend en charge plusieurs types de bases de données SQL, y compris SQLite, MySQL, PostgreSQL et ODBC.

```
#include <QSqlDatabase>
#include <QSqlQuery>
//...
    QSqlDatabase db = QSqlDatabase::addDatabase("QSQLITE");
    db.setDatabaseName(":memory:");
    if (!db.open()) {
        // Gérer l'erreur d'ouverture de la base de données
    }
    QSqlQuery query;
    if (!query.exec("CREATE TABLE people (id INTEGER PRIMARY KEY, name TEXT)")) {
        // Gérer l'erreur de création de la table
    }
    if (!query.exec("INSERT INTO people (name) VALUES ('John Doe')")) {
        // Gérer l'erreur d'insertion de données
    }
    if (!query.exec("SELECT * FROM people")) {
        // Gérer l'erreur de sélection de données
    }
    while (query.next()) {
        int id = query.value(0).toInt();
        QString name = query.value(1).toString();
        // Utiliser les données
```

Le composant QTSQL

- Les requêtes paramétrées (aussi appelées requêtes préparées) sont un moyen d'optimiser les performances et de renforcer la sécurité de vos requêtes SQL. Elles vous permettent de spécifier les commandes SQL à exécuter, puis de fournir les paramètres de ces commandes séparément
Les requêtes paramétrées offrent plusieurs avantages:

- Performance: La base de données peut compiler la requête une fois et la réutiliser plusieurs fois avec différents paramètres. Cela peut être beaucoup plus rapide que de compiler une nouvelle requête chaque fois, surtout si vous exécutez la même requête de nombreuses fois avec différents paramètres.
- Sécurité: En utilisant des requêtes paramétrées, vous pouvez éviter les injections SQL, car les paramètres sont envoyés à la base de données séparément de la requête elle-même et sont toujours traités comme des données littérales, pas comme une partie de la commande SQL.
- Confort: Les requêtes paramétrées gèrent automatiquement l'échappement et le formatage des données, ce qui peut rendre votre code plus propre et plus facile à lire. Vous n'avez pas à vous soucier de l'échappement des caractères spéciaux dans vos données, car cela est géré pour vous.

Le composant QTSQL

```
QSqlDatabase db = QSqlDatabase::addDatabase("QSQLITE");
    db.setDatabaseName(":memory:");

    if (!db.open()) {
        // Gérer l'erreur d'ouverture de la base de données
    }

    QSqlQuery query;
    query.prepare("INSERT INTO people (name) VALUES (:name)");

    QStringList names = {"John Doe", "Jane Doe", "Jim Doe"};
    for (const QString &name : names) {
        query.bindValue(":name", name);
        if (!query.exec()) {
            // Gérer l'erreur d'insertion de données
        }
    }
```

Le composant QTSQL *Remarque*

- Dans SQLite, lorsque vous insérez une nouvelle ligne dans une table qui a une colonne INTEGER PRIMARY KEY, SQLite génère automatiquement un nouvel ID pour cette ligne. Vous pouvez récupérer cet ID en utilisant la fonction lastInsertId de QSqlQuery juste après l'exécution de la requête d'insertion.

```
QVariant id = query.lastInsertId();
    if (id.isValid()) {
        qDebug() << "The last inserted ID was:" << id.toInt();
    } else {
        // Gérer l'erreur de récupération de l'ID
    }
```

Le composant QTSQL Exercice 5

- En partant de l'exercice 4, ajoutez la possibilité de :
  - enregister un contact dans une base de données
  - supprimer un contact à partir d'une base de données
  - rechercher les contacts d'une base de données.

Création des customs widgets

- Créer un widget personnalisé (custom widget) dans Qt vous permet de concevoir un contrôle d'interface utilisateur qui répond spécifiquement à vos besoins. Cela peut être aussi simple qu'un widget qui combine plusieurs widgets Qt standard, ou aussi complexe qu'un widget qui effectue son propre dessin personnalisé.

Création des customs widgets

```
#include <QWidget>
#include <QLabel>
#include <QPushButton>
#include <QVBoxLayout>
class MyCustomWidget : public QWidget
{
    Q_OBJECT
public:
    MyCustomWidget(QWidget *parent = nullptr)
        : QWidget(parent)
    {
        QVBoxLayout *layout = new QVBoxLayout(this);
        label = new QLabel("Cliquez sur le bouton ci-dessous", this);
        button = new QPushButton("Cliquez moi", this);

        layout->addWidget(label);
        layout->addWidget(button);

        connect(button, &QPushButton::clicked, this, &MyCustomWidget::onButtonClicked);
    }
signals:
    void buttonClicked();
private slots:
    void onButtonClicked() {
        label->setText("Le bouton a été cliqué !");
        emit buttonClicked();
    }
private:
    QLabel *label;
    QPushButton *button;
};
```

Application Multi-pages avec StackedWidget

- QStackedWidget est une classe Qt qui fournit un empilement de widgets dont un seul peut être visible à la fois. C'est comme une pile de cartes dont vous ne pouvez voir que la carte supérieure. C'est utile pour créer des interfaces utilisateur à plusieurs pages, où chaque page est un widget différent

1. Création d'un QStackedWidget
2. Ajout de widgets à QStackedWidget avec la méthode addWidget
3. Changement du widget visible avec les méthodes :setCurrentWidget(), setCurrentIndex() ou setCurrentWidget()
    - Vous pouvez également utiliser les méthodes next() et previous() pour parcourir les widgets dans l'ordre dans lequel ils ont été ajoutés.
4. Récupération du widget actuel avec la méthode currentWidget()

Sujet de TP 1

Système de Gestion de Bibliothèque
Votre mission est de créer une application de gestion de bibliothèque simple. Cette application doit être capable de gérer des livres et des utilisateurs.
Spécifications de base :

- Les livres doivent avoir un titre, un auteur et un identifiant unique.
- Les utilisateurs doivent avoir un nom, un email et un identifiant unique.
- Les utilisateurs peuvent emprunter et rendre des livres.
- Un livre ne peut être emprunté que par un seul utilisateur à la fois.
- L'application doit garder une trace de qui a emprunté quel livre et quand.

Spécifications techniques :

- Utilisez Qt et C++ pour l'interface utilisateur de l'application.
- Utilisez SQLite pour stocker les données de l'application.
- Utilisez le modèle DAO pour gérer l'accès aux données.

MVC en QT5

Principe de base de MVC

- L'architecture de type modèle-vue-contrôleur se divise en trois parties :

1. Modèle (Model) : Contient la logique métier. Il représente les données et les règles qui régissent ces données.
2. Vue (View) : Il s'agit de l'interface utilisateur. C'est ce que l'utilisateur voit et avec quoi il interagit.
3. Contrôleur (Controller) : Fait le lien entre le modèle et la vue. Il reçoit les événements d'entrée de l'utilisateur via la vue, les traite et les applique au modèle.

MVC en QT5

Principe de base de MVC

- Qt n'utilise pas exactement le patron de conception MVC, mais plutôt le MV (Modèle/Vue). Dans Qt, le rôle du contrôleur est principalement assuré par les objets de la vue.
- Dans Qt, le modèle est représenté par une sous-classe de QAbstractItemModel. Les méthodes de cette classe permettent de lire des données, et les signaux permettent de notifier la vue lorsque les données changent.
- Qt fournit plusieurs classes de vue par défaut, comme QListView, QTableView et QTreeView
- Dans Qt, le rôle du contrôleur est souvent assuré par les vues et les délégués. Par exemple, QTableView permet à l'utilisateur de sélectionner, d'éditer et de trier les données.

MVC en QT5

Exercice 1 : Créer une application de gestion de tâches

- Votre tâche est de créer une petite application qui permet à l'utilisateur de gérer une liste de tâches. Chaque tâche doit avoir un nom et une date d'échéance. L'utilisateur doit être capable d'ajouter, de modifier et de supprimer des tâches.
- Ajoutez une colonne "Complétée" à votre table, qui contient une valeur booléenne indiquant si la tâche a été complétée ou non. Affichez cette information dans votre vue et permettez à l'utilisateur de la modifier.

MVC en QT5

Exercice 2 :

Reprendre la vue N° 1 du sujet TP 1 en MVC