UNIVERSITY OF LIVERPOOL

# Application Development with C++ (ELEC362)

## Lecture 14: Building the first application

mihasan@liverpool.ac.uk

## Previous lecture

- Justification for learning Qt was given.
- Basic structure of Qt was explained.
- An introduction to Qt Creator was given.
- Deploying an application using Qt was explained.

## This lecture

- What is covered in this lecture?

  1. Introduction to the concept of applications building using Qt.

  2. Introduction of some of frequently used classes (application frames and texts).

- Why it is covered?

  To understand an application can be built from the classes in Qt.

- How are topics covered in this lecture:

  Building a sample application.

## The concept

- The main question is: how can an application be built from a library of classes?
- Analogous question: how can an electronic device be built from a storage cabinet of electronic components?

Components / objects
Terminals / member functions & data

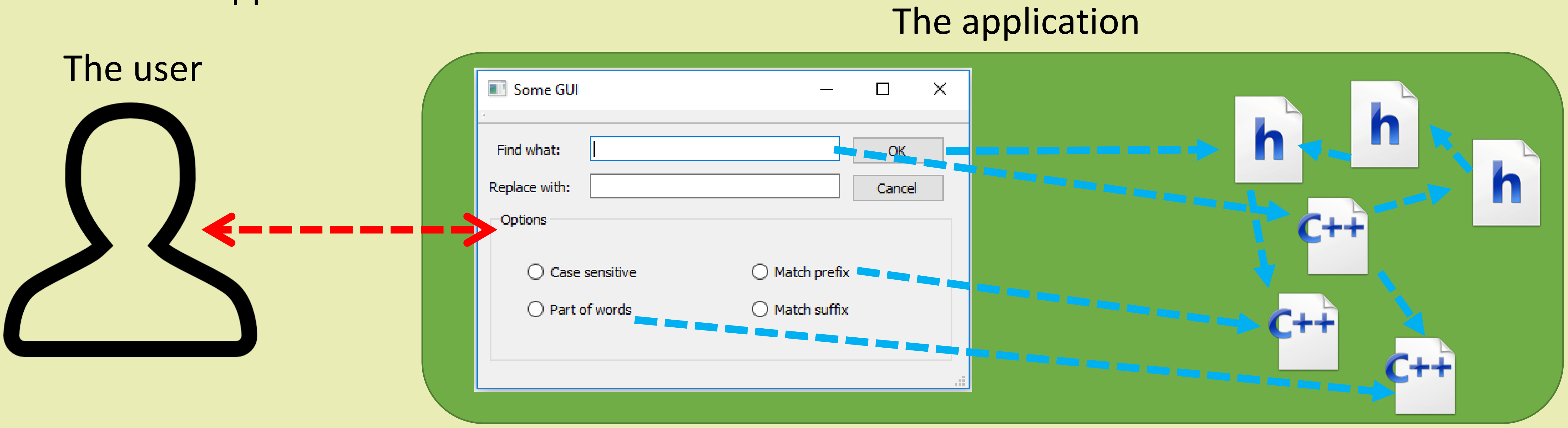
Circuits / applications

Devices / GUIs

Cabinet / library
Drawer / class

# How do GUI applications work?

- GUI applications work as follows:

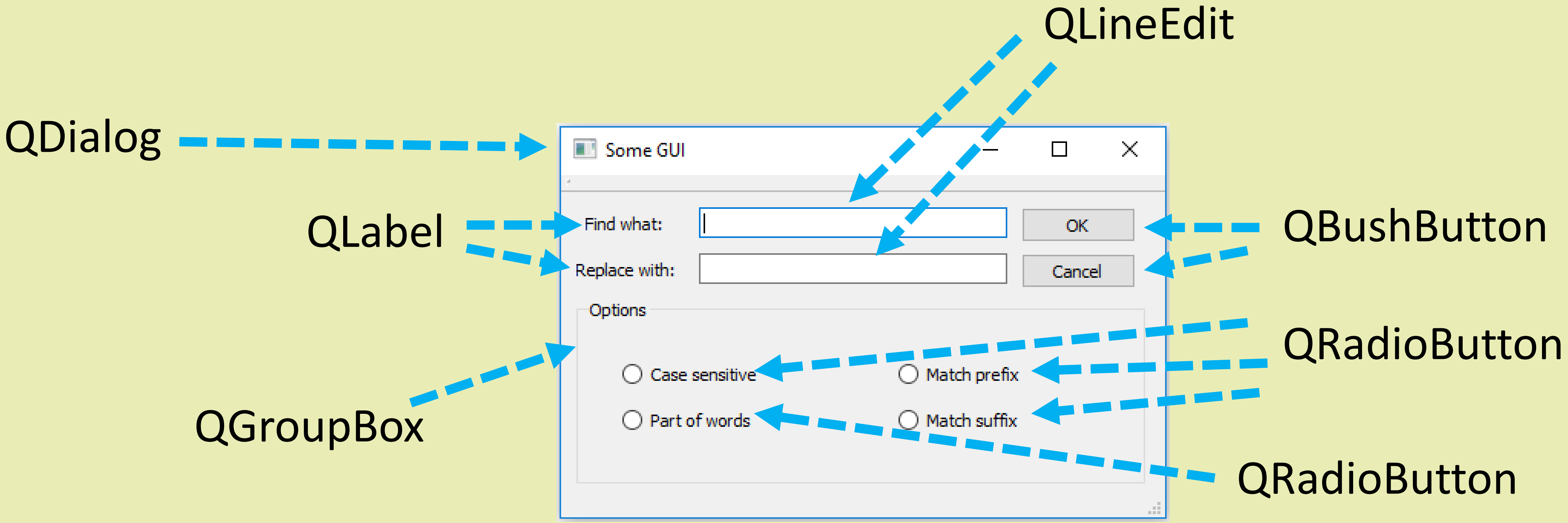


What the application developers do is: 1. Design the GUI using objects from the library.

2. Connect the objects to one another in a functional way.

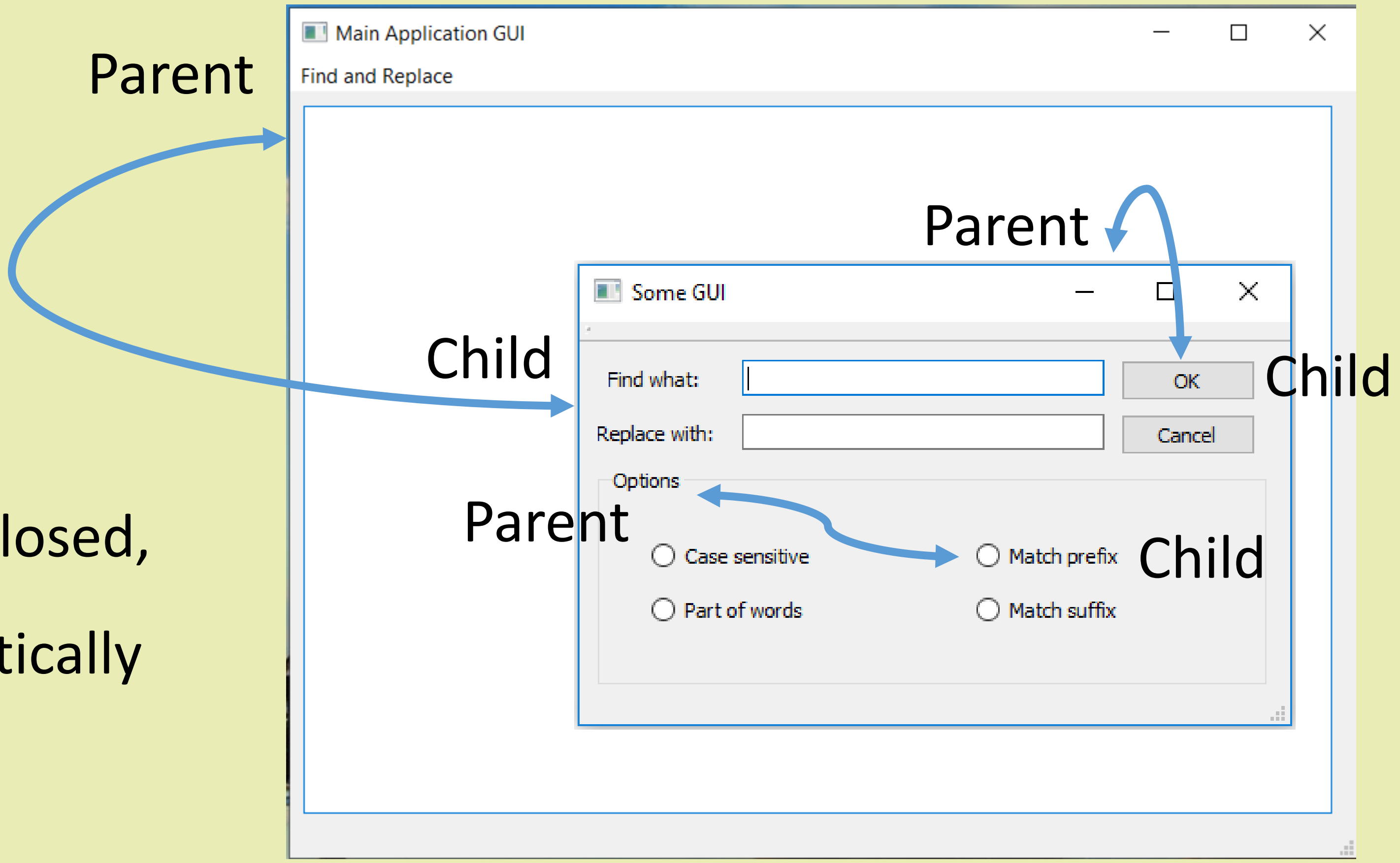
# Widgets

- When using Qt, every visual item in any window is a "widget" (stands for window gadget).
- Every widget is an object of a class but the opposite is not true!



# Parent and Child Widgets

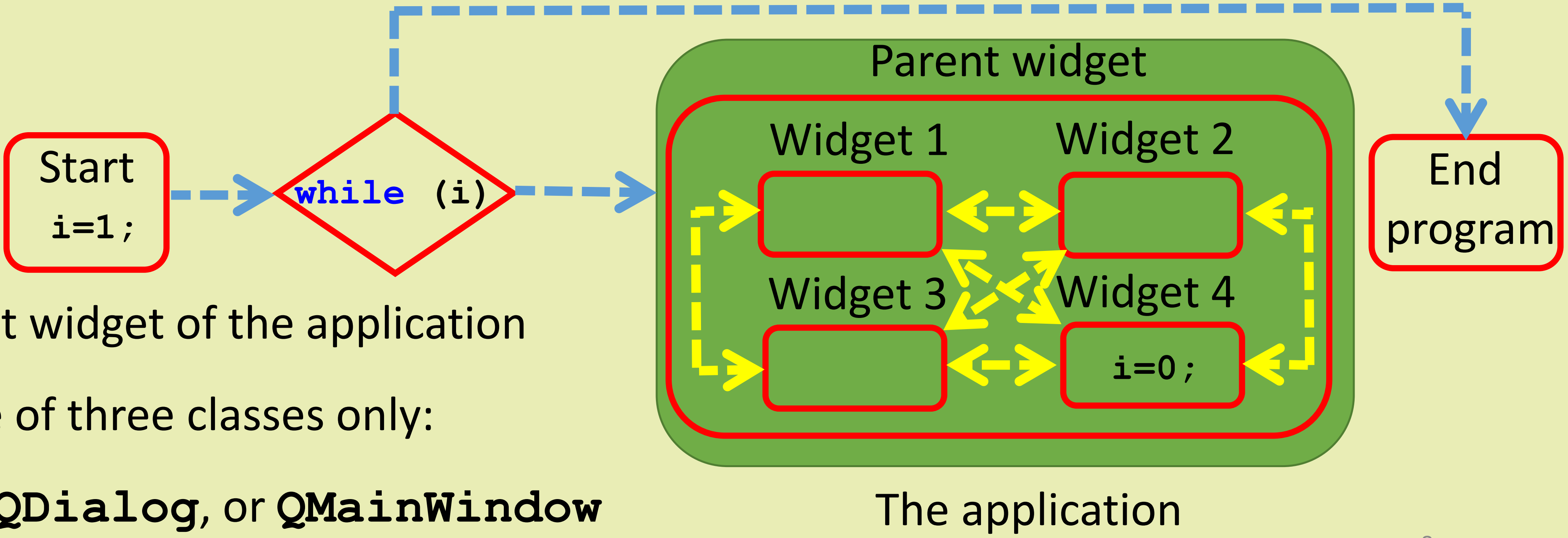
- In a complex application where there are multiple widgets and multiple windows, the widgets can be parent and/or child widgets.
- Parent-Child widgets is not inheritance.
- When a parent widget is closed, all its children are automatically closed or deleted.



# Event loop in Qt Application

- All widgets in an application must be encapsulated in one parent widget.
- Qt creator automatically creates an event loop that

  displays an instance of the parent widget.



- The parent widget of the application

  can be one of three classes only:

**`QWidget`**, **`QDialog`**, or **`QMainWindow`**

## `QWidget` Class

- This class is the base class for all user interface classes.
- Reference: https://doc.qt.io/qt-5/qwidget.html
- Its member data / functions control the following:

➢ Setting the outline of the widget (its shape, design, colours, size policy, etc..)

➢ Settings for mouse tracking, tool tips, curser type, etc..

- When used as a parent widget, it generate an empty user interface.

## `QMainWindow` Class

- This class provides a framework for applications where the user interacts with one weidget most of the time (called the central widget).
- Reference: [https://doc.qt.io/qt-5/qmainwindow.html](https://doc.qt.io/qt-5/qmainwindow.html)
- Its member data / functions control the following:

➢ Docking options for child widgets

➢ Options for menus and toolbars.

- When used as a parent widget, it creates a widget with menu bar and status bar templates (suitable for apps with menus and toolbars).

## `QDialog` Class

- This class is used to create windows for breif interaction with user. It is suitable for applications where the user interacts with most widgets equally.
- Reference: https://doc.qt.io/qt-5/qdialog.html
- Its member data / functions control the following:

➢ Modality of the window (whether the window freezes the app until closed or not).

➢ It can provide a return value when the window is closed.

- When used as a parent widget, it generate an empty user interface. It is suitable for apps where there are buttons and many input/output widgets

# A sample application: Word counter

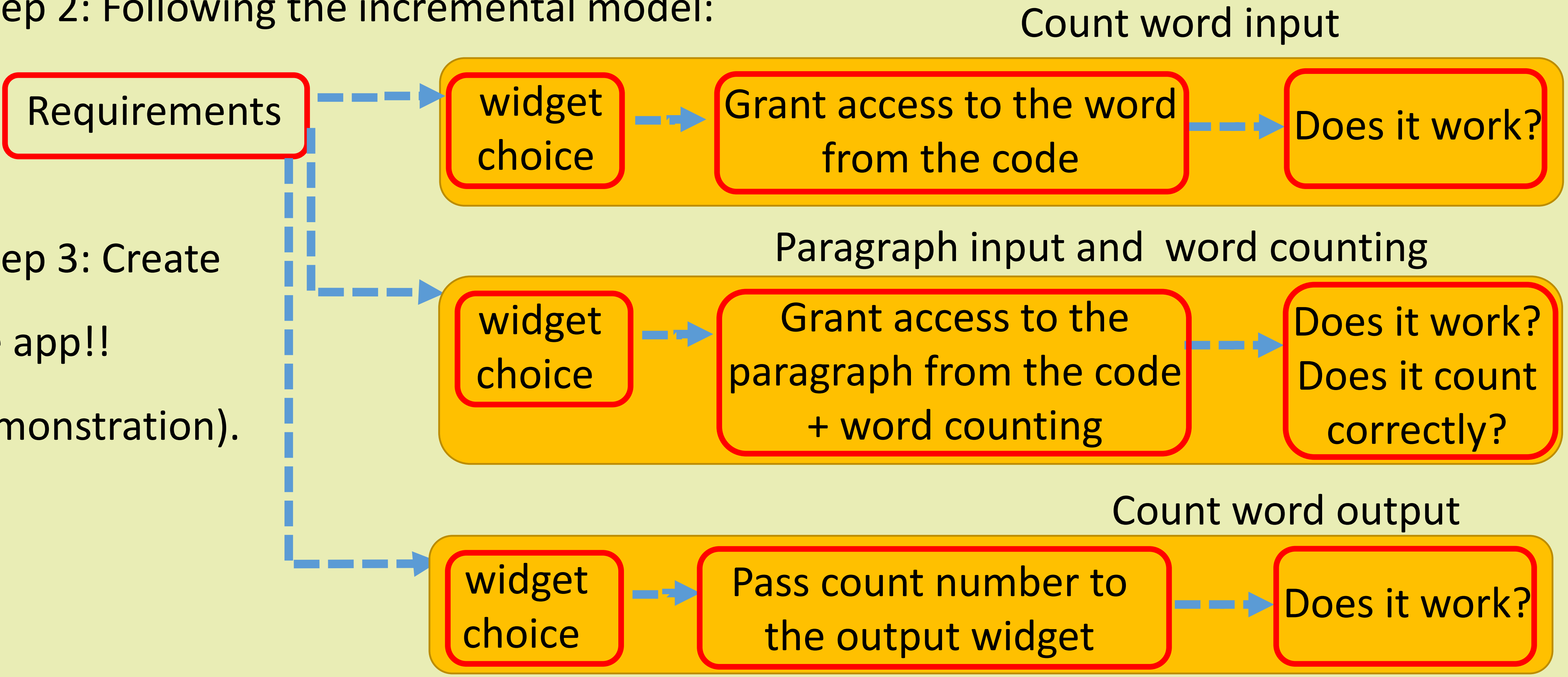
- The task: Design and implement a Qt based GUI application that counts the occurrence of a given word in a typed paragraph. The user will provide both the paragraph and the word to be counted.

- Step 1: Requirement analysis: the application must be able to:

1. Allow the user to input a paragraph or text to be scanned for the counted word (input).
2. Allow the user to specify the word to be counted (input).
3. Allows the user to trigger the counter (user action).
4. Display to the user the count number (output).

# A sample application: Word counter

- Step 2: Following the incremental model:


Requirements
Count word input
widget choice
Grant access to the word from the code
Does it work?
Paragraph input and word counting
widget choice
Grant access to the paragraph from the code + word counting
Does it work? Does it count correctly?
Count word output
widget choice
Pass count number to the output widget
Does it work?

- Step 3: Create the app!! (demonstration).

Good practice note: Write a comment in every block to state when it is called

# Guess the function trick!!

- Every class has 10s of functions, how can we know what function we need easily?
- When looking for a function in the documentation, ask three quesitions:

1. What is the expected return datatype? (important when getting data from object)
2. What does the function do (in simple words)?
3. What are the datatypes of its input parameters? (important when giving data to object)

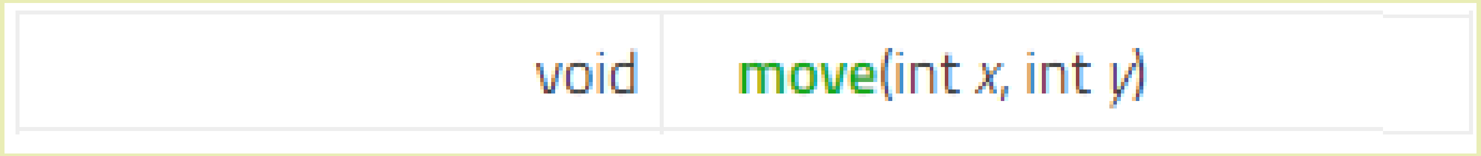
| | |
|---|---|
| Answer1 | Answer2 (Answer 3) |

- Think of multiple answers per question, most of the time you can guess the function from one answer only!

## Guess the function trick!!

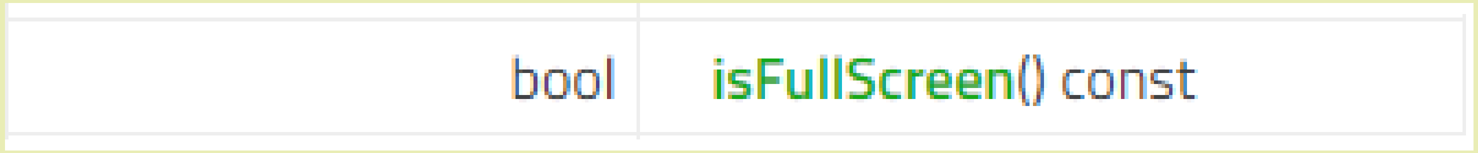
Few notes to improve your guess:

- A return type **`void`** is the most probable when the function returns nothing, or the function is used to modify something.

Example: 

```
void    move(int x, int y)
```

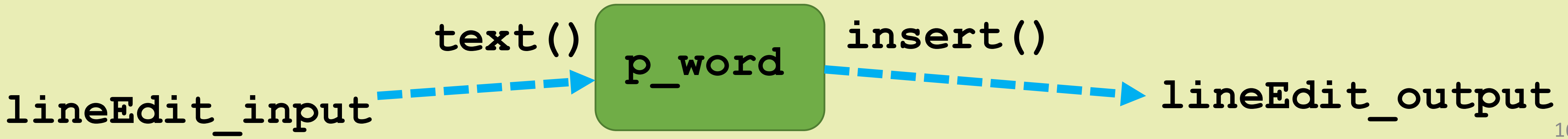
- A function has no paramters is the most probable if it returns a propertie of the object. Or if the function means only one thing.

Example: 

```
bool    isFullScreen() const
```

## Connecting objects in the application

- Every window in the application is defined by a pointer, for the parent window the pointer has the name **`ui`**
- Every widget we add in Qt Designer is also defined by a pointer, which has a name shown to the right of the screen (example: **`lineEdit`** is a pointer to **`QLineEdit`**)
- The objects in the code can interact through variables we define in the header file, such variables are used to read the data from a widget, process it, and output it to another widget, example:



## **`QDebug`** Class

- This class is used whenever the developer needs to write out debugging or tracing information to a device, file, string or console.
- Reference: https://doc.qt.io/qt-5/qdebug.html
- Its most wide use is as follows (assuming x is a variable in the code):

```
qDebug() << “The value of x: “<<x;
```

- This class mimics **`cout`** and **`cin`**, which are used primarily for debugging when developing a GUI.

## Word counter – Further improvements

- Verfication on the application level is done, what about validation?

- Try counting how many “h“ in “Hello, How are you?“

- Try counting how many “Hello”, then how many “are” in the same sentence.

- Try to move the edge of the application.

- Validation requires a lot of further work! That is why software testing is a job on its own!

## Summary

- The concept of developing a GUI application from classes in a library is discussed.
- The concept of widgets, parent and child relation were introduced.
- The first application has been built.
- The object-object connection and the user interface-object connections were discussed.
- Classes introduced: **`QWidget, QDialog, QMainWindow, QDebug, QLineEdit, QBushButton, QPlainTextEdit, QString`**