Hello this is Symeon Huang.

Welcome back to learning Qt5.

In this video we're going to talk about fundamentals.

We're going to take a look at Qt event loop, Qt object and it’s memory management.Right, if you are ready,let’s go back to our HelloWorld

project that we created in the last video. If you remember, in the end we don't return

0.

Instead we return “a.exec”.

Now what “exec” does?

According to the Help

“exec” will enter the main event loop and wait until the exit() is called.

It's a bit confusing.

But if we build and then run this application.

What we find out that the application started but it's never exited.

Why is that?

Because the application now is in the main event loop.It's waiting for any new events andwould execute

all the functions you..,

you previously defined for the event handlers.

In this case since this is just a very simple application, it's basically just waiting for any new event.

But we don't really have any handler to handle the events.

So basically the application is just running and nothing will happen.

If you do not want this behavior, as in this case,you set the return a.exec() entering the main event loop.

We can just return zero.

In this case the event loop would never be entered,hence, the application will exit.

Let's try it out.

As you can see the application has started and exited.

Now let's take a look at

Qt object and its memory management.

A Q\_OBJECT macro must appear here to declare that MyParentClass class is actually a Qt object class that will be managed by Qt meta object system.

In this case I've already created two new classes one called MyParentClass.

Another one is called MyChild.Note here the parent and child don't refer to the parent and child in the

inheritance or C++ class.

What it basically means is I'm going initialize them in the main.cpp. and then use MyParentClass

as the parent of MyChild.

Now I have already created destructive functions.

And they were simply output a string.

Let's include them in the main.cpp

And then initialize them.

Here as you can see I use the parent pointer to initialize MyChild.As you all know in C++, if you

initialize an object in the heap, you must delete them.

Release all the memory you just allocated.

Or in this case, of course, I have to delete parent.

Do you have to delete child?

You must, of course, because you initialized child in the heap.

But in Qt don’t really need it because MyChild and MyParentClass are both Qt object derived classes.

and the parent is the parent of MyChild.

What does that mean?

That means when the parent gets released it will automatically release all children.

You don't believe.Let’s run this application, as you can see MyParentClass destructor is called.

And immediately MyChild destructor is also called.

So amazing isn’t it? So in this case, you don't really have to worry about the memory management yourself, as long as you assign

a parent, once the parent is deleted

and released all it’s children will also be released.Note, the parent pointer is exclusive.

That means once you assigned the parent, you cannot add parent.

But you could change the parent.

Now somebody would wonder what happens if I also want to delete child, just for safety reasons.

Well in this case,Qt actually provide a mechanism to let you call delete multiple times.

Not call delete like this.But use a member function called deleteLater().

Now I'm going to call deleteLater() from child again as well.

As you know if I call the plain delete like this, it will crash. But if I call deleteLater(), it will not. Let’s find out.

Aha the application does not crash.

But if you want to use deleteLater().You must make sure that that event loop is entered.

Otherwise this function will not do anything.

As you can see from the documentation.It says this will schedule this object for deletion.hashes

will be pushed into the event queue for execution.If there's no such event loop running, the event would

never be triggered.

The function will never be triggered.

In which case, parent and child would not be deleted.

So here I change return zero again back to a.exec().

No let's build.

And try it out again.

As you can see the destructors are both called.

But the application has not exited yet.

Now you might wonder.

I think I would like the application to quit once the parent is deleted.

How we’re going to do that?

It's actually quite straight forward.

You can include QCoreApplication header.

In the destructor.

there is a static function called quit().

If you call quit().

It will exit the application with the return code 0.Now let’s try it out. See if our application will exit or not.

Aha, perfect! both parent and child got deleted.

And also the application exited as well.

Now let's go back to see here.We have one deleteLater().What happens if I

call multiple times.Actually I may also want to delete for some reason, I don’t know, I might want to delete MyParent.

For some reason I may wante to delete my parent. Hence, I say delete MyParent.

Also, let's go back to the documentation.

That deleteLater() would just simply schedule these objects for deletion.Every time when Qt execute this

deletion, it will actually check if the object is deleted or not.

Once this has been deleted,

no further action will

actually be taken. Hence, it’s totally thread-safe.

As you can see, they both get deleted and no crashes happen.In the next video we're going to talk about

command-line processing to enrich your command-line applications to take in arguments with Qt.