



# GUI Application #1



## 핵심 정리

### 예제 Project

Project Name	Window1
Project Type	Empty qmake Project
Module	QT += widgets
Source	main.cpp

GUI 관련 많은 클래스가  
“**widgets**” 모듈에 있음.



## 핵심 정리

- 핵심 1. GUI Application 을 위한 event loop  
⇒ “**QApplication**” 사용

QCoreApplication

**Console Application** 을 위한  
event loop 제공

**QApplication**

**GUI Application** 을 위한  
event loop 제공

“**widgets**” 모듈에 있음.

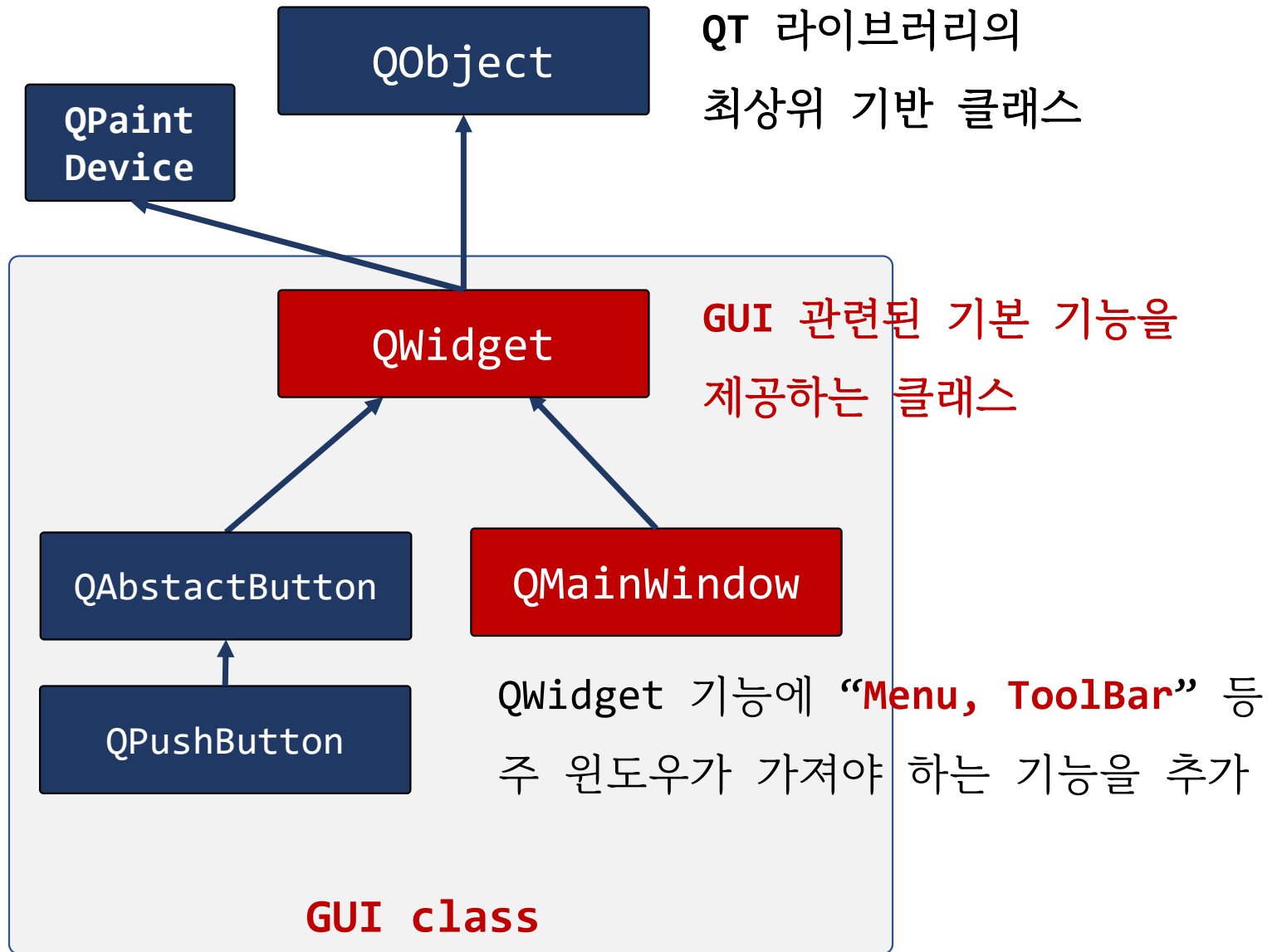
**.pro 파일에 QT += widgets**



## 핵심 정리

### ● 핵심 2. 윈도우 생성하는 방법

⇒ **QWidget** 또는 **QMainWindow** 클래스 사용





## 핵심 정리

- 윈도우의 다양한 “속성을 변경” 하려면
  - ⇒ QWidget 의 “다양한 멤버 함수를 사용”하면 된다.
  - ⇒ 인터넷에서 “**QWidget** ....” 로 검색하면 다양한 정보(소스)를 얻을 수 있다.



# GUI Application #2



## 핵심 정리

### 예제 Project

Project Name	Window2
Project Type	Empty qmake Project
Module	QT += widgets
Source	main.cpp



## 핵심 정리

- 프로그램에서 윈도우를 만든 경우

⇒ 윈도우에서 발생하는 다양한 이벤트(마우스, 키보드 등)를 처리해야 한다.

⇒ 또한, 윈도우 위에 다양한 자식 윈도우(button, slider 등의 컨트롤)을 만들고, 컨트롤에서 나오는 이벤트로 처리하는 코드를 작성해야 한다.

- QWidget, QMainWindow 를 직접 사용하지 말고 파생 클래스를 만들어서 사용하는 것이 관례.





## 핵심 정리



- ① 기반 클래스가 가진 다양한 “**가상함수를 override**” 할 수 있고.
- ② 생성자에서 자식 윈도우(button, slider 등의 컨트롤)를 생성.
- ③ 자식 윈도우(컨트롤)에서 나오는 이벤트로 처리하는 코드를 작성.

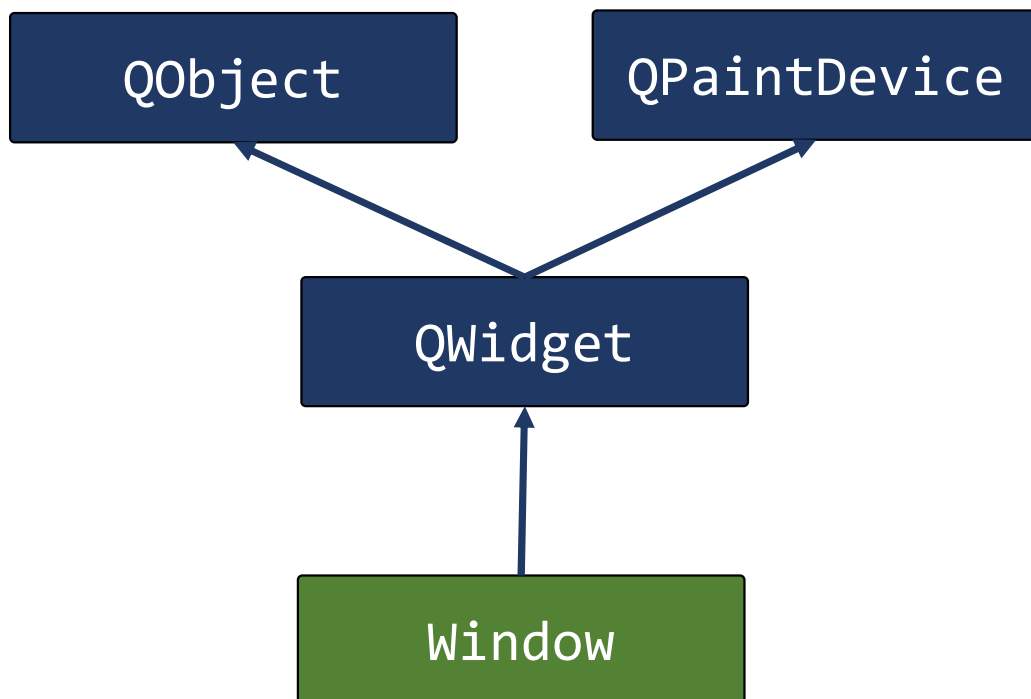


## 핵심 정리

- 기반 클래스의 가상함수 **override**

⇒ 윈도우 위에서 마우스를 누르거나 키보드를 누르면 약속된 가상함수가 호출된다.

⇒ Qt Creator 의 “**Refactor**” 기능을 사용하면 편리하게 추가할 수 있다.



Mouse, Keyboard 이벤트 관련 자세한 내용은 해당 주제를 다루는 강좌 참고



## 핵심 정리

- 자식 윈도우(컨트롤) 만들기
  - ⇒ Window 생성자에서 자식 윈도우 생성
- 컨트롤의 종류와 사용법
  - ⇒ “**Control 강의**” 참고
- 컨트롤의 위치, 크기 관리 하는 방법
  - ⇒ “**Layout 강의**” 참고
- 컨트롤에서 발생하는 이벤트를 처리하는 방법
  - ⇒ “**signal slot 강의**” 참고



# signal slot #1



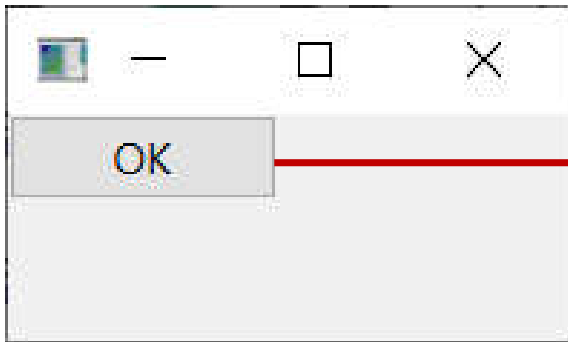
## 핵심 정리

### 예제 Project

Project Name	<b>SIGNAL_SLOT1</b>
Project Type	Empty qmake Project
Module	<b>QT += widgets</b>
Source	<b>main.cpp</b>



## 핵심 정리



```
class Window
```

```
void btn_clicked()  
{  
}
```

- 버튼을 **click** 할 때 **Window** 멤버 함수를 호출
  - ⇒ 버튼 객체의 “**clicked** 라는 **signal**” 을  
Windows 객체의 “**btn\_clicked** 라는 **slot**” 과  
연결
- 이번 강의에서는
  - ⇒ 따라하기 식으로 코드를 완성.
  - ⇒ 사용법을 먼저 익히고.
  - ⇒ 이어지는 강의에서 좀더 자세히 설명.



## 핵심 정리

### ● Signal Slot 을 사용하려면

```
class Window : public QWidget
```

```
{  
  1 Q_OBJECT
```

클래스 선언 1번째 줄에  
Q\_OBJECT 매크로 추가

```
    QPushButton* btn;
```

```
public:
```

```
    Window()
```

```
{
```

```
    btn = new QPushButton("OK", this);
```

signal - slot 연결

```
  3
```

```
    QObject::connect(btn, SIGNAL(clicked()),  
                      this, SLOT(btn_clicked()));
```

```
}
```

```
  2
```

```
public slots:
```

```
    void btn_clicked() { qDebug(__func__); }
```

```
};
```

버튼 signal 과 연결할  
slot 함수 작성

### ● 핵심!!

⇒ “클래스 선언은 반드시 헤더 파일에 작성” 되어야 한다.



## 핵심 정리

- 소스에 이상이 없는데, 컴파일 에러가 나오는 경우  
⇒ “Build” 메뉴에서 “**Clean**” 을 실행한후  
다시 빌드



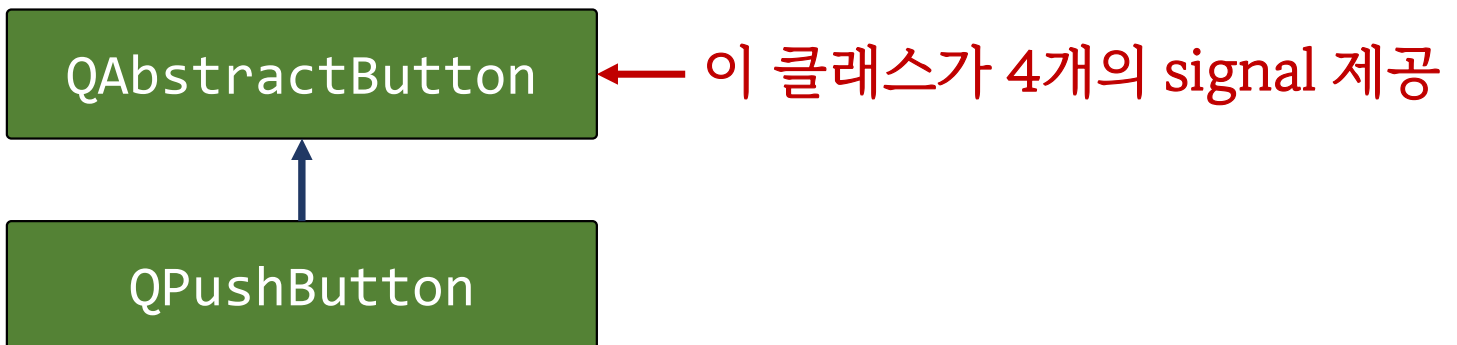


## 핵심 정리

### ● QPushButton 의 signal

<b><u>toggled</u></b> (bool <i>checked</i> )	checkable button 에서 check 상태가 변경될 때 발생
<b><u>pressed</u></b> ()	버튼위에서 마우스 버튼을 누를 때 발생
<b><u>released</u></b> ()	버튼위에서 마우스 버튼을 놓거나, 누른 상태로 커서가 버튼을 벗어날 때
<b><u>clicked</u></b> (bool <i>checked = false</i> )	버튼을 <b>Click</b> 할 때 발생

- QPushButton 에는 어떤 시그널이 있는지 알고 싶다면  
⇒ QT DOCUMENTATION 참고





## 핵심 정리

- **slot**

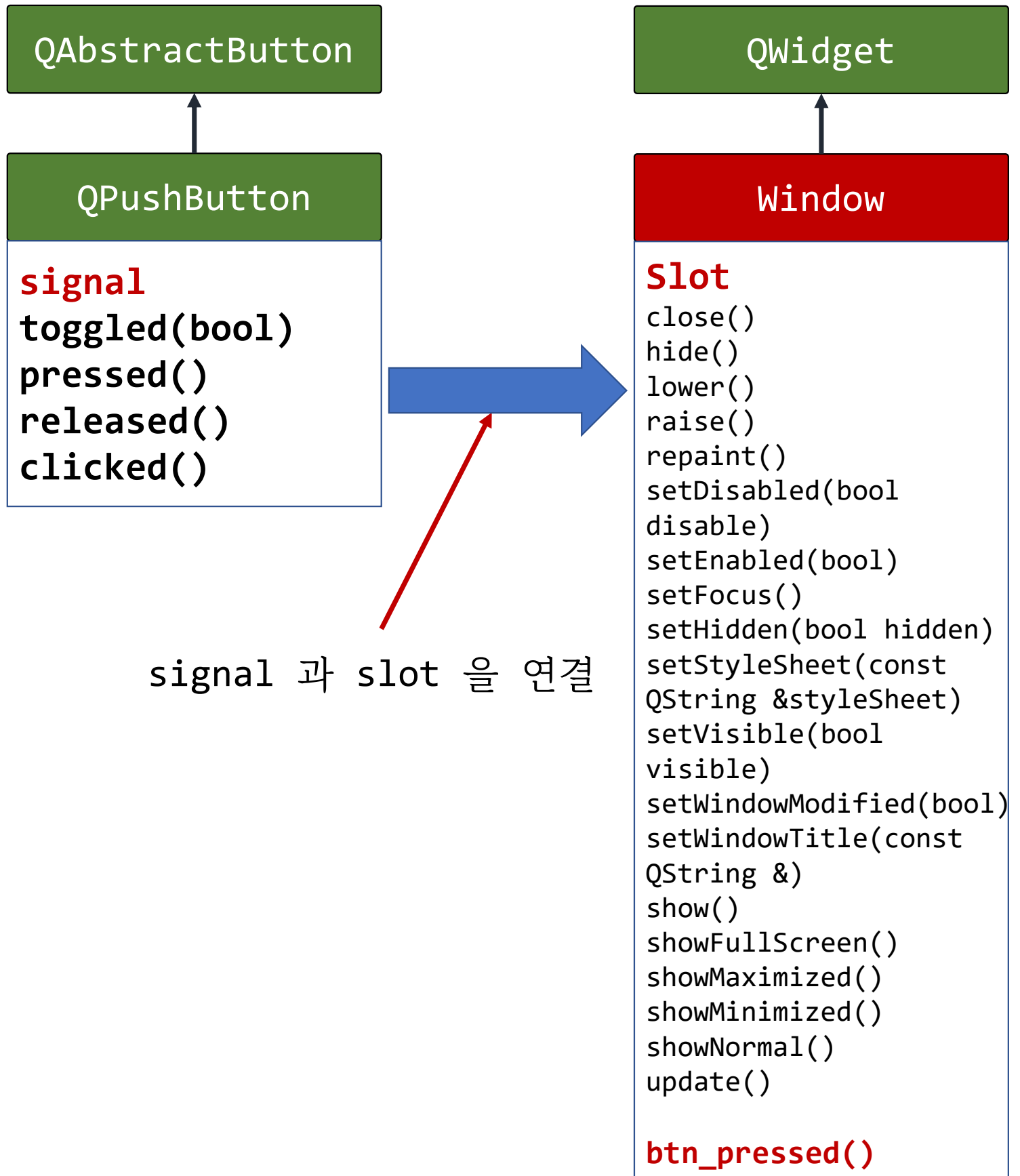
- ⇒ public 멤버 함수 이므로 사용자가 원할 때 직접 호출 할 수 있다.
- ⇒ 다른 객체의 signal 과 연결할 수도 있다.

- **QWidget 의 close() 멤버 함수**

- ⇒ Window 를 닫을 때 사용
- ⇒ slot 으로 선언되어 있기 때문에 signal 과 연결가능



## 핵심 정리





# signal slot implementation



## 핵심 정리

### 예제 Project

Project Name	<b>SIGNAL_SLOT_IMPL</b>
Project Type	<b>Empty qmake Project</b>
Module	
Source	<b>signal_slot.h</b> <b>main.cpp</b>

⇒ GUI Application 이 아닌 Console Application




## 핵심 정리

### InfiniteCounter

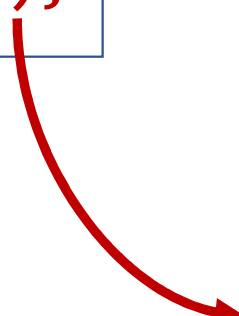
```
void run()
{
    1초마다 카운트증가
}
void valueChanged(int);
```

시그널 발생(emit)



### PrintValue

```
void print(int)
{
}
```





## 핵심 정리

- 임의 클래스가 **signal slot** 을 사용하려면
  - ① 클래스 선언은 헤더 파일로 작성되어야 한다.
  - ② “**QObject** 로 부터 상속” 받아야 한다.  
(또는, QWidget 등 QObject 로 부터 파생된 클래스로 부터)
  - ③ 클래스 선언 제일 앞에 “**Q\_OBJECT** 매크로 추가”



## 핵심 정리

- 멤버 함수를 **slot** 으로 만들려면

⇒ `public` 접근 지정자 뒤에 “**slots**” 을 표기 한다

```
class PrintValue : public QObject
{
    Q_OBJECT

public slots:
    void print(int value) { ... }
};
```





## 핵심 정리

- 클래스에 **signal** 을 추가 하려면

```
class InfiniteCounter
{
public:
    void run()
    {
        int cnt = 0;

        while(1)
        {
            QThread::sleep(1);
            2 cnt++;
            emit valueChanged(cnt);
        }
    }
}
```

이순간 시그널 발생

1

```
signals:
    void valueChanged(int);
};
```

선언만 작성



using signal slot



## 핵심 정리

### 예제 Project

Project Name	USING_SIGNAL_SLOT
Project Type	Empty qmake Project
Module	QT += widgets
Source	window.h main.cpp



## 핵심 정리

### ● **signal slot** 을 연결하는 방법

**SIGNAL()**, **SLOT()**

매크로 사용

**slot** 함수만 연결 가능

멤버 함수 포인터 사용

**slot** 이 아니어도 연결가능  
람다표현식도 사용가능



# Meta Object System



## 핵심 정리

### 예제 Project

Project Name	MOS
Project Type	Empty qmake Project
Source	main.cpp



## 핵심 정리

### ● C++ 언어의 역사와 버전

C++ 언어의 탄생

**1983**

1차 공식 표준화

1998

새로운 C++ 표준화

C++11/14/17/20/23

다양하고 강력한 문법과 라이브러리가 추가됨.

### ● QT 의 탄생과 Meta Object System

⇒ C++ 언어가 공식 표준화 되기 전인 **1995**년에 QT 탄생.

⇒ C++ 언어 자체의 능력이 많이 부족했던 시절

⇒ QT 는 당시 C++의 언어적 부족함을 해결하기 위해 “**Meta Object System**” 이라는 개념을 도입



### MOS (Meta Object System)

사용자가 만든 C++ 코드

MOC (Meta Object Compiler)

가 먼저 컴파일

사용자가 만든 C++ 코드  
+

MOC가 추가한 C++ 코드

1. signal & slot

2. RTTI

3. dynamic property

지원하기 위한 코드

C++ 컴파일러

실행 파일





- **MOS(Meta Object System) 를 사용하려면**

- ① 클래스 선언은 헤더 파일에 작성
  - ② QObject 로 부터 상속 받아야 하고
  - ③ 클래스 선언 제일 위쪽에 Q\_OBJECT 매크로를 넣는다.
- ⇒ QT Creator 의 “**Add Class**” 를 사용하면 편리하게 추가 가능.
- ⇒ 멤버 함수 추가도 “**Refactor**” 를 사용하면 편리하다.



## 핵심 정리

- MOS가 제공하는 3가지 기술

① **signal slot**

② **RTTI**

③ **dynamic property**

- **RTTI**

⇒ 실행시간에 객체의 타입을 조사하는 기술

⇒ C++ 표준 문법이 제공하는 기술도 있지만,  
QT 자체기술도 제공

```
Sample sam;
```

```
const QMetaObject* mo = sam.metaObject();
```

```
qDebug() << mo->className();
```