Вопросы по Qt.

1. Объект класса QApplication осуществляет контроль и управление приложением. Любая использующая Qt-программа с графическим интерфейсом должна создавать только один объект этого класса, и он должен быть создан до использования операций, связанных с пользовательским интерфейсом.
Приложение запускается вызовом QApplication::exec(). С его запуском приводится в действие цикл обработки событий, определенный в классе QCoreApplication, являющимся базовым для QApplication. Этот цикл передает получаемые от системы события на обработку соответствующим объектам. Он продолжается до тех пор, пока либо не будет вызван статический метод QCoreApplication::exit(), либо не закроется окно последнего элемента управления.

2. **Объектная модель Qt** подразумевает, что все построено на объектах. Класс **QObject** - основной, базовый класс. Большинство классов Qt являются его наследниками. Классы, имеющие сигналы и слоты, должны быть унаследованы от этого класса.
Сигналы и **слоты** — это средства, позволяющие эффективно производить обмен информацией о событиях, вырабатываемых объектами.

Механизм сигналов и слотов.
3. ***Сигналы*** (signals) — методы, которые в состоянии осуществлять пересылку сообщений.

1. Сигналы определяются в классе, как и обычные методы, только без реализации (перед их определением должно стоять слово signal:).
2. Сигналы не должны возвращать каких-либо значений, и поэтому перед именем метода всегда должно стоять void.
3. Не имеет смысла определять сигналы как private, protected или public, т. к. они играют роль вызывающих методов.
4. Выслать сигнал можно при помощи ключевого слова emit.
5. Сигналы также имеют возможность высылать информацию, передаваемую в параметре.

```
class MySignal : public QObject {
Q_OBJECT
public:
	void sendSignal(){
		emit sendString("Information");
	}
signals:
	void sendString(const QString&);
};
```

4. ***Слоты*** (slots) — это методы, которые присоединяются к сигналам.

1. Они являются обычными методами. Самое большое их отличие состоит в возможности принимать сигналы.
2. Они определяются в классе как public, private или protected. Соответственно, перед каждой группой слотов должно стоять: private slots:, protected slots: или public slots:. Слоты могут быть виртуальными (однако не рекомендуется делать их виртуальными).
3. Есть небольшие ограничения, отличающие обычные методы от слотов. В слотах нельзя использовать параметры по умолчанию или определять слоты как static.

```
class MySlot : public QObject {
Q_OBJECT
public:
	MySlot();
public slots:
```

```
	void slot() {
		qDebug() << "I'm a slot";
	}
};
```

5. Соединение объектов осуществляется при помощи статического метода connect(), который определен в классе QObject. В общем виде, вызов метода connect() выглядит следующим образом:

```
QObject::connect(
	const QObject*		sender,
	const char*		signal,
	const QObject*		receiver,
	const char*		slot,
	Qt::ConnectionType	type = Qt::AutoConnection
);
```

```
MyClass::MyClass() : QObject()
{
...
	connect(pSender, SIGNAL(signalMethod(int)),
		pReceiver, SLOT(slotMethod(int))
	);
...
}
```

6. MOC, макрос Q_OBJECT.

Qt расширяет язык C++ дополнительными ключевыми словами.
Проблема расширения языка C++ решена в Qt с помощью специального препроцессора MOC (Meta Object Compiler, метаобъектный компилятор).
Его задача — создавать для заголовочных файлов дополнительные cpp-файлы, подлежащие компиляции и присоединению их объектного кода к исполняемому коду программы. Для того чтобы MOC мог распознать классы, нуждающиеся в подобной переработке, такой класс должен содержать макрос Q_OBJECT.

Макрос Q_OBJECT должен располагаться сразу на следующей строке после ключевого слова class с определением имени класса. После макроса не должно стоять точки с запятой. Внедрять макрос в определение класса имеет смысл в тех случаях, когда созданный класс использует механизм сигналов и слотов или если ему необходима информация о свойствах.

Класс, содержащий сигналы и слоты, должен быть унаследован от класса QObject или от класса, унаследованного от этого класса.

7. Объектная иерархия.

Иерархия — порядок подчинённости низших звеньев к высшим, организация их в структуру типа дерево (Википедия).
Конструктор класса QObject:

```
QObject(QObject* pobj = 0);
```

в качестве параметра получает указатель на другой объект класса QObject или унаследованного от него класса. Это указатель на объект-предок. Благодаря этому параметру существует возможность создания иерархий объектов.
При уничтожении созданного объекта (при вызове его деструктора) все присоединенные к нему объекты-потомки уничтожаются автоматически. Значит не нужно заботиться об освобождении ресурсов памяти. Именно поэтому необходимо создавать объекты, а особенно объекты неверхнего уровня, динамически, при помощи оператора new, иначе удаление объекта приведет к ошибке при исполнении программы.

8. ***Виджет*** (widget) — это базовый элемент (элемент управления) графического интерфейса пользователя, "строительный материал" для создания интерфейса. Это не просто область, отображаемая на экране, а компонент, способный выполнять различные действия, например реагировать на поступающие сигналы и события или отправлять сигналы другим виджетам. Qt предоставляет полный арсенал виджетов: от кнопок меню до диалоговых окон, необходимых для создания профессиональных приложений.

QWidget унаследован от класса QObject, и может использовать механизм сигналов/слотов и механизм объектной иерархии.
Виджеты без предка называются ***виджетами верхнего уровня*** (top-level widgets) и имеют свое собственное окно. Все виджеты без исключения могут быть виджетами верхнего уровня.
Для отображения виджета на экране вызывается метод **show()**, а для скрытия — метод **hide()**.

Литература.

1. Шлее М. Qt 5.3. Профессиональное программирование на C++. - СПб.: БХВ-Петербург, 2015.
стр. 45-46, 53-70, 77-78, 119-120.