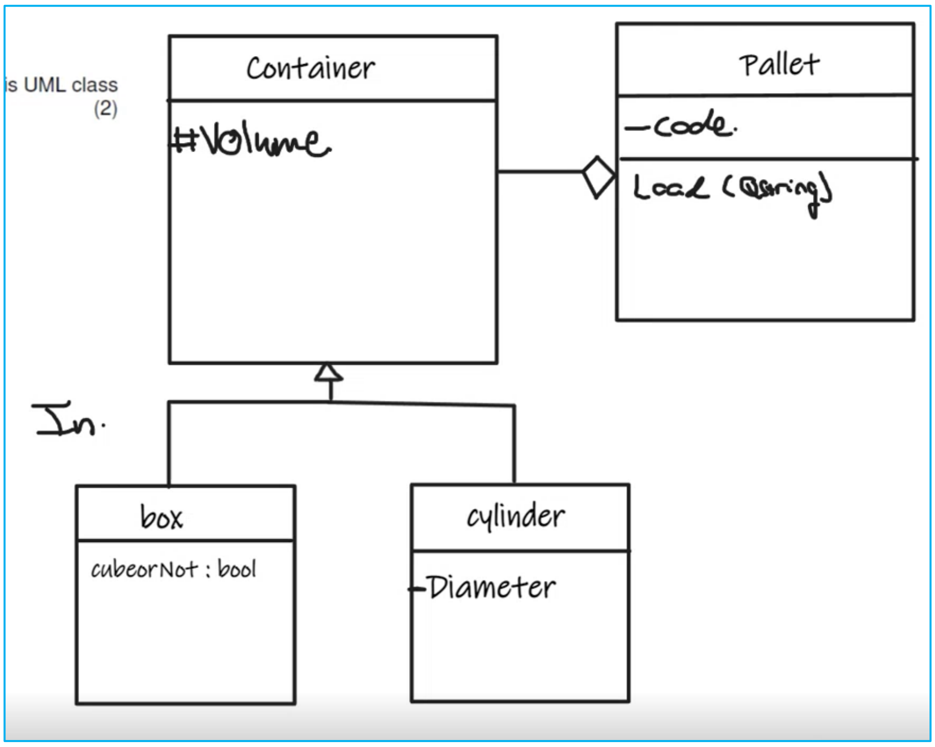
Given

Transporting cargo around the world is essential in ensuring that customers have access to the goods they need and want.

All such items are packaged in some sort of container (which, for the purposes of this scenario, has some volume). Generally, there are two kinds of containers: (i) a box (where we want to know whether it is cube shaped or not), and (ii) a cylinder (where we want to know its diameter). For transport, containers are packed onto pallets, and pallets are then included in a load (where each load will have a code).

1.1 Draw a partial UML class diagram, include:

- the necessary classes
- class attributes
- class relationships



1.2 Aggregation vs composition

Aggregation relationship, since containers can exist outside of the pallet class

1.3 The load code takes the following format

- Year value between 2000 and 2099 (both included)
- Forward slash (/)
- Month value between 01 and 12 (both included)
- Forward slash (/)
- L
- A serial number starting from 1, running up to 9999

Write the regular expression (in quotes) that can be used to check that a load code meets the required criteria. An example of a valid code is 2022/01/L1. Ensure that you use escape characters correctly.

(“20[0-9]{2}/(0|1)[1-2]/L[1-9]{1.4}”)

The intention is to serialise container objects using reflective programming approaches. The idea is to convert all object data in a load to XML and save this data on a network store.

2.1 Major benefits of using a reflective approach in this scenario

- Code reusability
- Gives access to the metaobject and dynamic properties at run-time

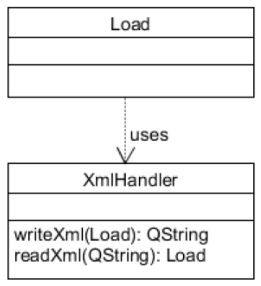
2.2 For a container object that is urgent, the users want to add a property to just this specific object. In such cases, the property name is urgent and its value is a message indicating its priority (such as high priority). Assuming that the classes are set up to allow this ability, write the code to implement this intention for an object named obj.
[Note that this approach can be used to add other optional properties to other objects where necessary.]

```
obj->setProperty(“urgent”, “high priority”);
```

2.3 Consider the requirement to serialise the load class to and from XML.

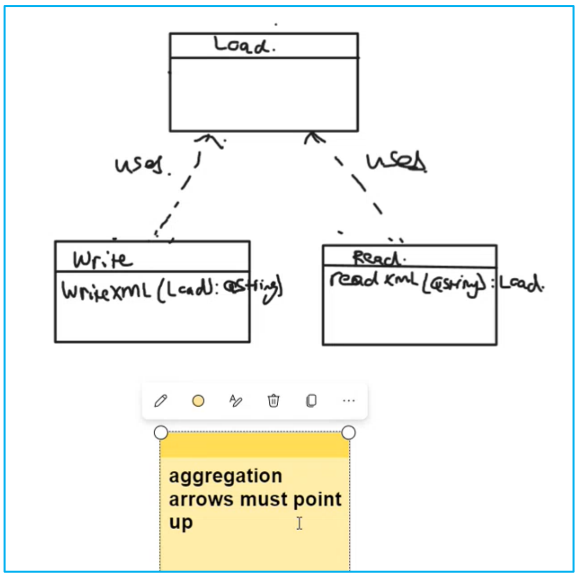
2.3.1 The following UML class diagram for the serialisation has been provided (where the `QString` is the XML text) for some `Load` class.

Given:



It has been argued that this is incorrect. Provide a better UML class diagram.

Rectified:



aggregation arrows must point up

Question regarding XML:

2.3.2 Complete the code by filling in the parts indicated (see **bold** for completed lines)

```
QString XmlHandler::writeXml(Load load)
{
     QString loadCode = load.getCode(); // gets the load code
     QDomElement root = doc.createElement("Load");
     root.setAttribute("code", loadCode);

     //set up pallet tags
     QDomElement palletTag = doc.CreateElement("Pallet");

     // as part of the root load tag
     root.appendChild(palletTag);

     QString classname = mo->className();

     // handle the case where an urgent property has been added
     // to an object; not all such extra properties should be
     // added to the tag, only those named 'urgent'

     foreach(QByteArray dpName, c->dynamicPropertyNames())
     {
       if(dpName == "urgent")
       {
         base.setAttribute(dpname, c->property(dpname).toString());
       }
     }

// add this base tag to its parent tag
palletTag.appendChild(base);

for (int i=1; i<mo->propertyCount(); i++)
{
  QMetaProperty prop = mo->property(i);
  QString propertyName = prop.name();
  QString propertyValue = prop.read(c).toString();

  // create the necessary tags for the property
  QDomElement baseChild = doc.CreateElement(propertyName);
  QDomText baseChildData = doc.createTextNode(propertyValue);

  base.appendChild(baseChild);
  baseChild.appendChild(baseChildData);
}

// the generated XML needs to be sent back to calling function
return doc.toString();
```

2.4 XML text is to be sent over a network

2.4.1 Extend this class definition so that the class can be run as a thread, including all code that would be added to conform to best practice (see **bold** for lines added in)

Serialize Header file implementation:

```
class Serialize : public QObject
{
  Q_OBJECT
public:
  explicit Serialize(Load l); // load object passed to constructor
private:
  Load load;
public slots:
  void doSerialize(); // used to transfer over the network
signals:
  void start(QString&);
  void finish(QString&);
};
```

2.4.2 Complete the following code (that you would expect to find in the client code) that will run an instance of this class as a thread. The code should start the thread and clean up afterwards.

```
Load load;
Serialize* s(new Serialize(load));


QThread *thread = new QThread();

thread->moveToThread(s);

connect(thread, &QThread::start, this, &s::doSerialize);
connect(this, &s::finish, thread, &QThread::quit);
connect(s, SIGNAL(finished()), this, SLOT(deleteLater()));
connect(thread, SIGNAL(finished()), thread, SLOT(deleteLater()));

thread->start();
```

And the QThread Question - Networking:

2.4.3 Finally, write the code for the Serialize::doSerialize() function that gets the XML text serialisation using the XmlHandler class in 2.3.2 (repeated below) and uses UDP to send it over the network using port 55555.

```
Serialize::doSerialize()
{
    XMLHandler xml;
    QUdpSocket * socket = new UdpSocket;
    QUint16 port = 5555;
    QHostAddress host = QHostAddress::Broadcast;
    QByteArray msg = xml.writeXML(load).toLatin1;
    QNetworkDatagram datagram(msg, host, port);
    socket->writeDatagram(datagram);
}
```

MVC Question regarding MyListWidget:

A model-view approach will be used to display a list of load codes. The following class has been proposed (which allows for the use of a memento of instances of the class).

```
class MyListWidget : public QListWidget
{
     public:
     MyListWidget();
     MyListMemento* createMemento();
     void setMemento(MyListMemento* mlm);
};
```

3.1 Given the following code, write the statement that would be used to add the load instance's code to the list widget (see **bold** for line added)

```
MyListWidget* mylist = new MyListWidget;
Load load;

new QListWidgetItem(load, mylist); //value, model.
```

MyListMemento header file:

3.2 Consider the code for the `createMemento()` function.

```
MyListMemento* MyListWidget::createMemento()
{
     MyListMemento *mlm(new MyListMemento);
     QStringList strlist;

     for(int i=0; i<this->count(); i++)
     {
          strlist.append(item(i)->text());
     }

     mlm->setState(strlist);
     return mlm;
}
```

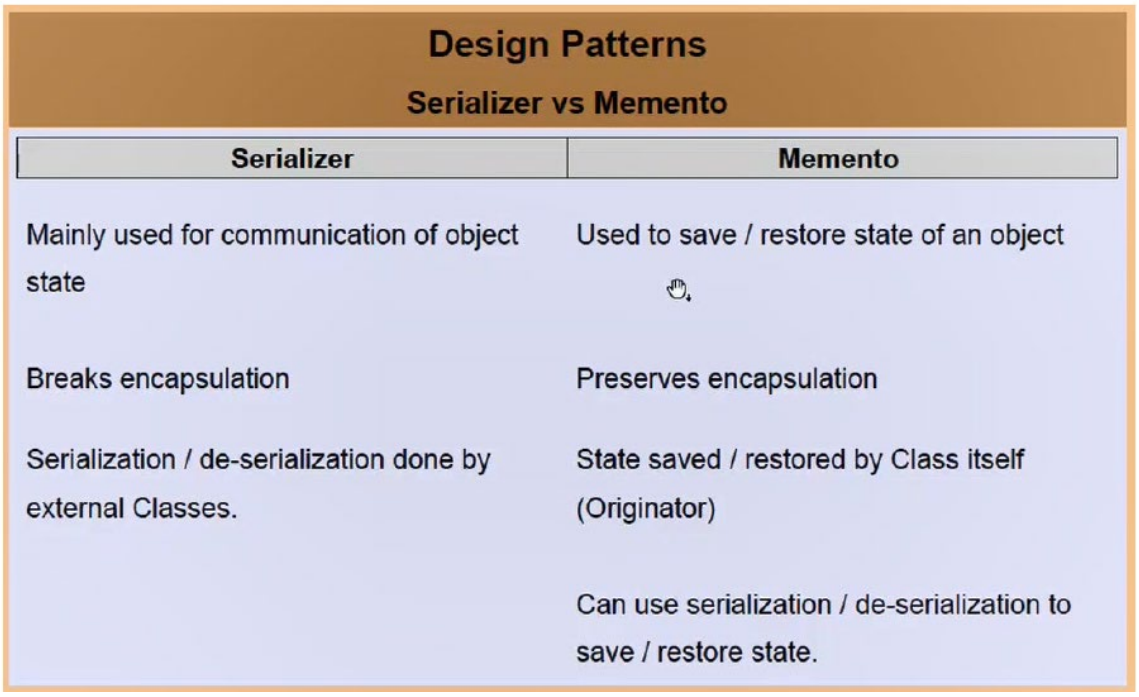
Provide the class definition (that would be expected in the header file) for the `MyListMemento` class.

```
class MyListMemento
{
public:
private:
  friend class MyListWidget;
  MyListMemento();
  void setState(QStringList list);
  QStringList getState():

  QStringList myList;
}
```

3.3 Distinguish between the use of the serialiser and memento design patterns as they have been used in **this scenario**. How do they differ in terms of their ultimate purpose (apart from the fact that the design patterns are being applied to different objects)? Make sure that it is clear which pattern you are referring to in your answer.

MyListWidget (originator) saves and restores the state of the object instance



Design Patterns

Serializer vs Memento

Serializer	Memento
Mainly used for communication of object state	Used to save / restore state of an object
Breaks encapsulation	Preserves encapsulation
Serialization / de-serialization done by external Classes.	State saved / restored by Class itself (Originator)
	Can use serialization / de-serialization to save / restore state.

3.4 What model/view alternatives does Qt provide that could best be used in place of the QListWidget?

QListView, the data is isolated from the view

3.5 Where in the scenario presented in this paper could a factory method design pattern be appropriately used?

The factory method design pattern could be used for the containers, where container would be a super/base class, the box and cylinder would be sub-classes that inherits from the base class and you would have a containerFactory class that creates container instances (using the sub-classes)