THE UNIVERSITY OF ALABAMA IN HUNTSVILLE

Lecture Qt007 Input Validation

Instructor: David J. Coe

CPE 353 – Software Design and Engineering

Department of Electrical and Computer Engineering

THE UNIVERSITY OF ALABAMA IN HUNTSVILLE

Outline

- Motivation
- Input Validation
- **QValidator** Class and Its Descendants
- Hands-On Example: Input Validators
- Hands-On Example: Qt and Object-Oriented Design
- Key Points

Motivation

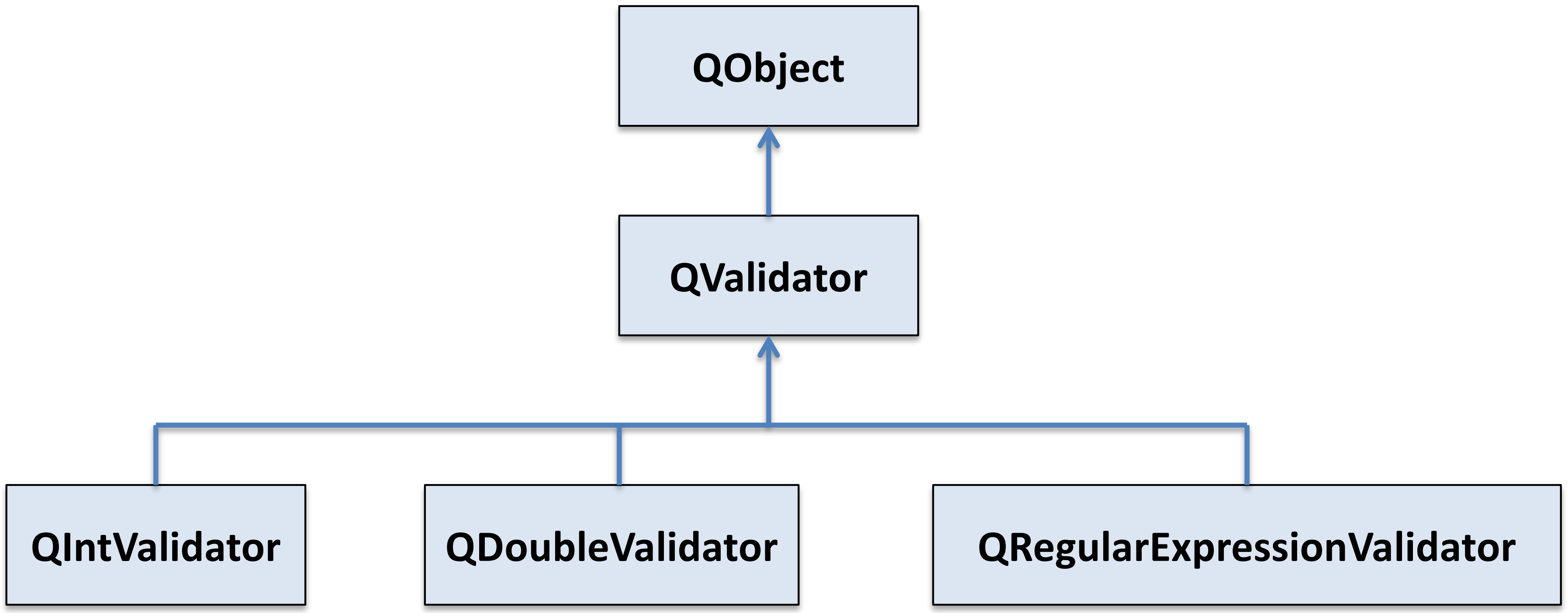
- Users may accidently or deliberately input invalid values
- Your code can be made more robust if you prevent the user from entering values that are clearly invalid
- Qt includes validators that can be used to block invalid inputs

Input Validation

- **QLineEdit** and **QComboBox** classes include a method called ***setValidator(...)*** which may be used to select a validation object that will restrict the user's ability to input inappropriate values

- Validators provided with Qt inherit from the base class **QValidator**

QValidator Class and Its Descendants



Unified-Modeling Language (UML) Class Diagram

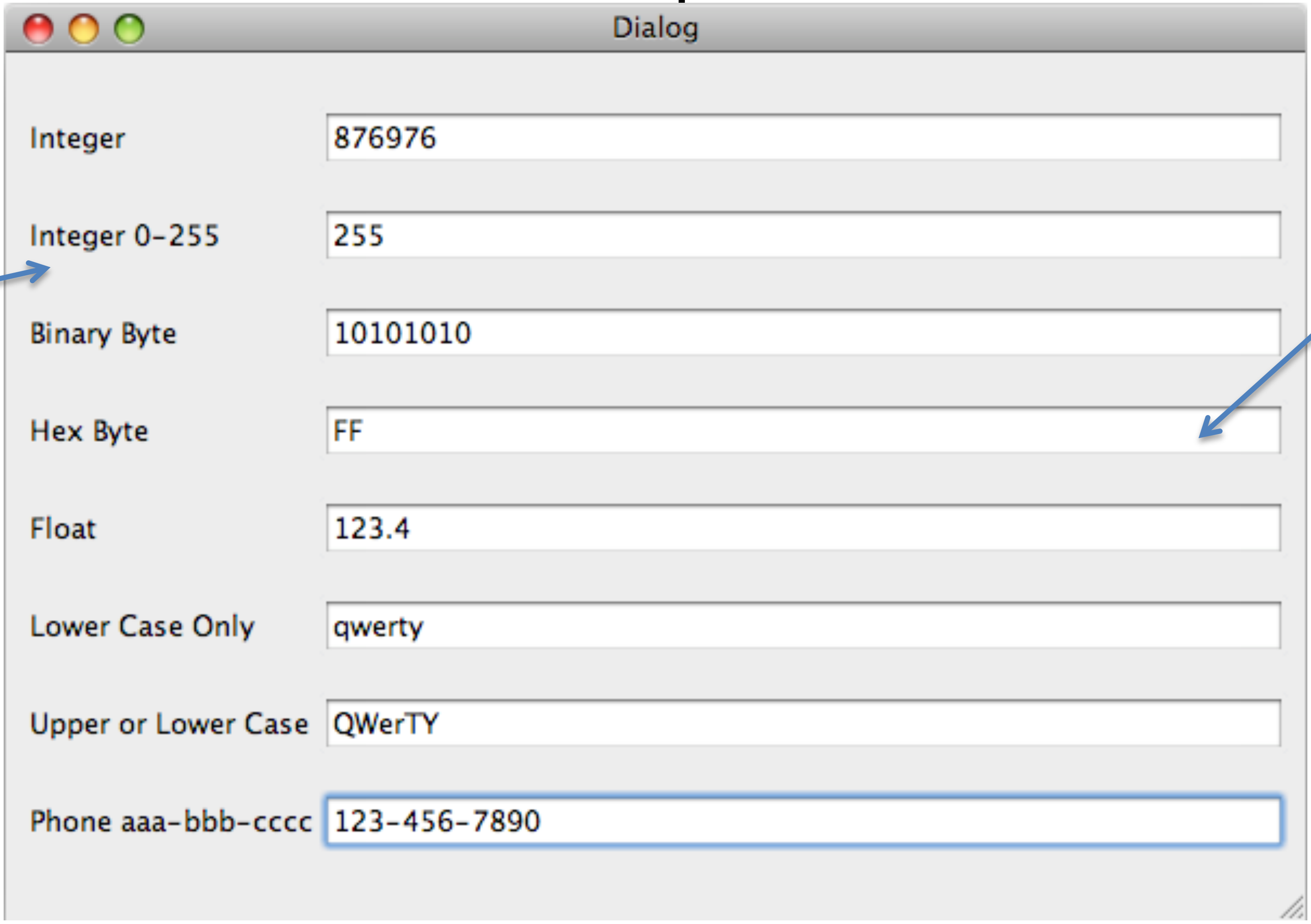
Input QValidator Class and Its Descendants

- **QValidator** includes a ***virtual*** function named ***validate(…)*** which evaluates a **QString** and returns a value of type **QValidate::State**
- Possible values of **QValidate::State** are
  - **QValidator::Invalid**
  - **QValidator::Intermediate**
    - Is input valid if user not finished?
  - **QValidator::Acceptable**
- Derived classes reimplement ***validate(…)***

Hands-On Example: Input Validators

- This example illustrates use of various validator objects to restrict user input

Labels indicating type of input allowed



Integer	876976
Integer 0-255	255
Binary Byte	10101010
Hex Byte	FF
Float	123.4
Lower Case Only	qwerty
Upper or Lower Case	QWerTY
Phone aaa-bbb-cccc	123-456-7890

Line edits accepting user input

Hands-On Example: Input Validators

```
// Standard auto-generated main.cpp

#include <QApplication>
#include "dialog.h"
int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    Dialog w;
    w.show();
    return a.exec();
}
```

Hands-On Example: Input Validators

```
// Standard auto-generated dialog.h

#ifndef DIALOG_H
#define DIALOG_H
#include <QDialog>
namespace Ui
{
    class Dialog;
}
class Dialog : public QDialog
{
    Q_OBJECT

public:
    Dialog(QWidget *parent = 0);
    ~Dialog();

private:
    Ui::Dialog *ui;
};
#endif // DIALOG_H
```

Hands-On Example: Input Validators

```
// Customized dialog.cpp

#include "dialog.h”
#include "ui_dialog.h”
#include <QIntValidator>
#include <QRegularExpressionValidator>
#include <QDoubleValidator>

Dialog::Dialog(QWidget *parent) : QDialog(parent), ui(new Ui::Dialog)
{
  ui->setupUi(this);

  QIntValidator* intValidator =
    new QIntValidator(ui->intLineEdit);
  ui->intLineEdit->setValidator(intValidator);

  QIntValidator* byteintValidator =
    new QIntValidator(0, 255, ui->byteintLineEdit);
  ui->byteintLineEdit->setValidator(byteintValidator);
```

Hands-On Example: Input Validators

THE UNIVERSITY OF ALABAMA IN HUNTSVILLE

```
// Customized dialog.cpp - continued

  QRegularExpressionValidator* binaryValidator =
     new QRegularExpressionValidator(QRegularExpression("[01]{1,8}”),
                                     ui->binaryLineEdit);
  ui->binaryLineEdit->setValidator(binaryValidator);

  QRegularExpressionValidator* hexValidator =
    new QRegularExpressionValidator(
            QRegularExpression("[0-9A-Fa-f]{1,2}"), ui->hexLineEdit);
  ui->hexLineEdit->setValidator(hexValidator);

  QDoubleValidator* floatValidator =
    new QDoubleValidator(-100.0, 100.0, 1, ui->floatLineEdit);
  ui->floatLineEdit->setValidator(floatValidator);

  QRegularExpressionValidator* lowerValidator =
    new QRegularExpressionValidator(
                                QRegularExpression("[a-z]{1,15}"),
                                ui->lowercaseletter);
  ui->lowercaseletter->setValidator(lowerValidator);
```

Hands-On Example: Input Validators

```
// Customized dialog.cpp – continued

  QRegularExpression  upperlowerRegExp("[a-zA-Z]{1,15}");
  QRegularExpressionValidator* upperlowerValidator =
     new QRegularExpressionValidator(upperlowerRegExp,
     ui->upperlowercaseLineEdit);
  ui->upperlowercaseLineEdit->setValidator(upperlowerValidator);

  QRegularExpression  phoneRegExp("[0-9]{3}-[0-9]{3}-[0-9]{4}");
  QRegularExpressionValidator* phoneValidator =
    new QRegularExpressionValidator(phoneRegExp, ui->phoneLineEdit);
  ui->phoneLineEdit->setValidator(phoneValidator);
}

Dialog::~Dialog()
{
  delete ui;
}
```

Lessons Learned: Validators

- QValidator objects can be used to block some undesirable user inputs resulting in a product that is more robust
- One issue with validators is that the undesired inputs are just blocked with no additional feedback given to indicate to the user that the input is deliberately blocked

Example: Qt and Object-Oriented Design

THE UNIVERSITY OF ALABAMA IN HUNTSVILLE

- Suppose we want an integer validator that provides ***audible feedback*** to the user to indicate that input is deliberately blocked
- Currently no Qt integer input validator class provides this sort of audible feedback
- ***Goal: custom beeping integer validator class***
  - Must still accept desired integer inputs and reject undesired inputs as with **QIntValidator**
  - Must ***beep*** to indicate a rejected input

Example:
Qt and Object-Oriented Design

Design Choice #1:

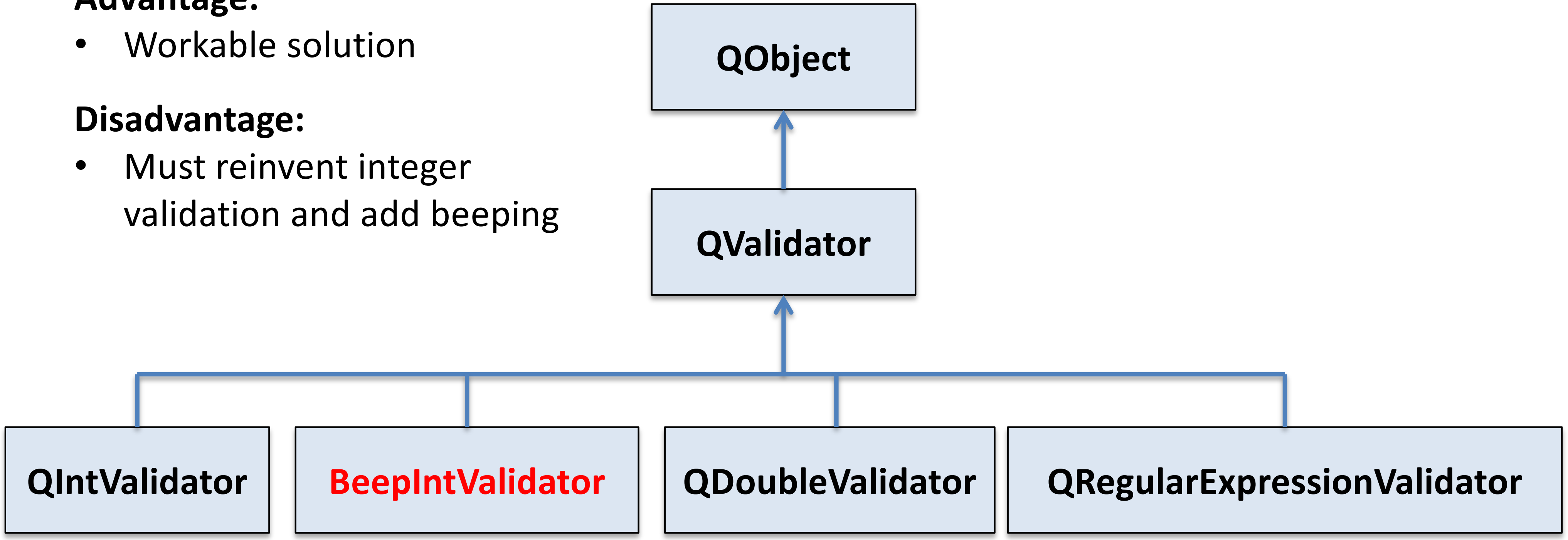
Create a **BeepIntValidator** class that inherits directly from **QValidator**

Advantage:

- Workable solution

Disadvantage:

- Must reinvent integer validation and add beeping



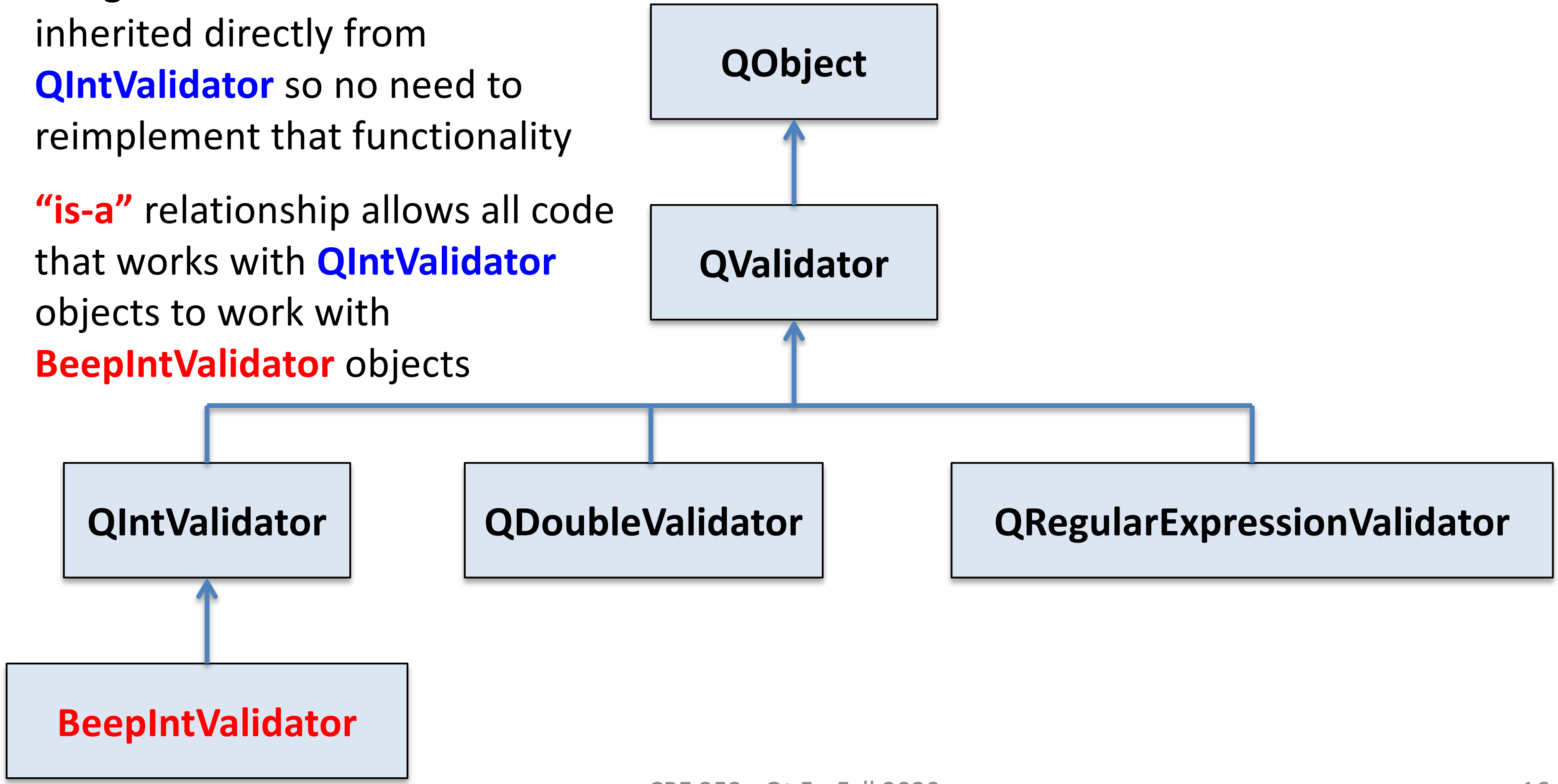
Example:
Qt and Object-Oriented Design

Design Choice #2:

Create a **BeepIntValidator** class that inherits directly from **QIntValidator**

Advantages:

- Integer validation mechanism inherited directly from **QIntValidator** so no need to reimplement that functionality
- **“is-a”** relationship allows all code that works with **QIntValidator** objects to work with **BeepIntValidator** objects

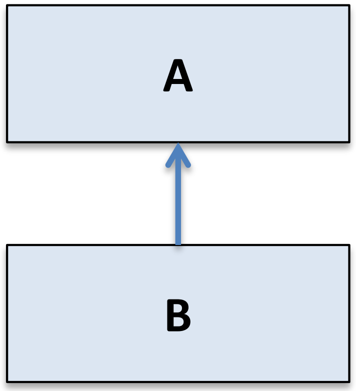


Example:
Qt and Object-Oriented Design

Recall:

Inheritance creates “is-a” relationship

- An object of the derived-class type **B** is also an object of the base-class type **A**



- Example:
  - If **Book** is the base class and **Novel** is the derived class, then

A **Novel** **“is a”** **Book**

but

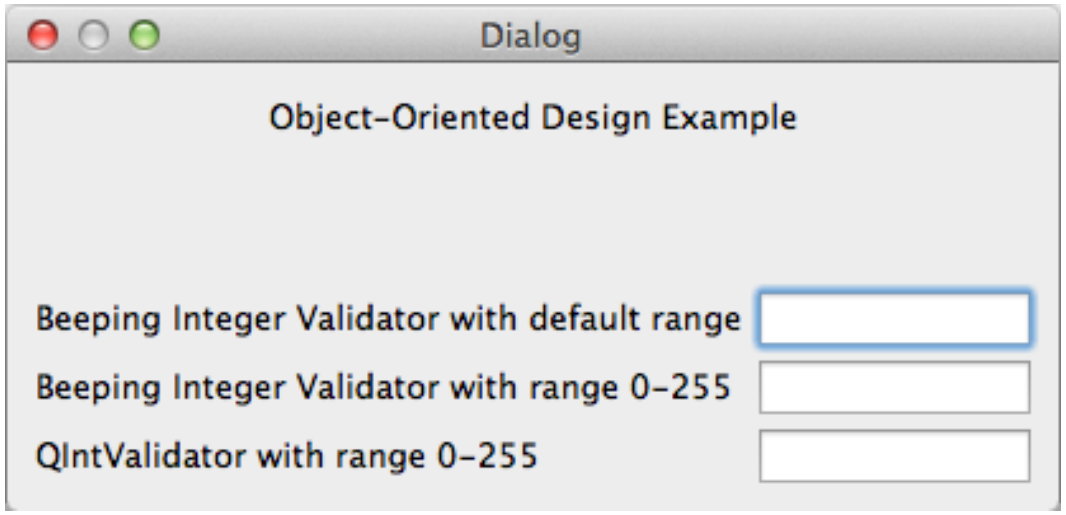
a **Book** is not necessarily a **Novel**

Example: Qt and Object-Oriented Design

```
// Standard auto-generated main.cpp

#include <QApplication>
#include "dialog.h"

int main(int argc, char *argv[])
{
  QApplication a(argc, argv);
  Dialog w;
  w.show();
  return a.exec();
}
```



Object-Oriented Design Example

Beeping Integer Validator with default range

Beeping Integer Validator with range 0-255

QIntValidator with range 0-255

Hands-On Example: Qt and Object-Oriented Design

```
// Standard auto-generated dialog.h

#ifndef DIALOG_H
#define DIALOG_H

#include <QDialog>
namespace Ui
{
    class Dialog;
}
class Dialog : public Qdialog
{
    Q_OBJECT
public:
    Dialog(QWidget *parent = 0);
    ~Dialog();
private:
    Ui::Dialog *ui;
};#endif // DIALOG_H
```

Hands-On Example: Qt and Object-Oriented Design

THE UNIVERSITY OF ALABAMA IN HUNTSVILLE

“is-a” relationship allows me to substitute a BeepIntValidator object wherever I can use a QIntValidator object

```
// Customized dialog.cpp
#include "dialog.h”
#include "ui_dialog.h”
#include <QIntValidator>
#include "beepintvalidator.h”

Dialog::Dialog(QWidget *parent): QDialog(parent), ui(new Ui::Dialog)
{
    ui->setupUi(this);

    BeepIntValidator* intValidator = new BeepIntValidator(ui->intBeepLineEdit);
    ui->intBeepLineEdit->setValidator(intValidator);

    BeepIntValidator* intValidatorRange =
         new BeepIntValidator(0, 255, ui->byteIntBeepLineEdit);
    ui->byteIntBeepLineEdit->setValidator(intValidatorRange);

    QIntValidator* byteIntValidatorRange =
         new QIntValidator(0, 255, ui->byteIntLineEdit);
    ui->byteIntLineEdit->setValidator(byteIntValidatorRange);
}

Dialog::~Dialog()
{
    delete ui;
}
```

Hands-On Example: Qt and Object-Oriented Design

```
// beepintvalidator.h

#include <QIntValidator>

#ifndef BEEPINTVALIDATOR_H
#define BEEPINTVALIDATOR_H

class BeepIntValidator : public QIntValidator
{
public:
  BeepIntValidator( QObject * parent = 0 );

  BeepIntValidator ( int minimum, int maximum, QObject * parent );

  QValidator::State validate ( QString & input, int & pos ) const;
};
#endif // BEEPINTVALIDATOR_H
```

Hands-On Example: Qt and Object-Oriented Design

Code Reuse!!

```
// beepintvalidator.cpp

BeepIntValidator::BeepIntValidator( QObject* parent ) :
        QIntValidator(parent)    // Constructor initializer
{
  /* No additional code required */
}

BeepIntValidator::BeepIntValidator(int minimum, int maximum, QObject* parent) :
        QIntValidator(minimum, maximum, parent)    // Constructor initializer
{
 /* No additional code required */
}


// Virtual method validate must be reimplemented in newly derived class

QValidator::State BeepIntValidator::validate( QString & input, int & pos ) const
{
    QValidator::State  status = QIntValidator::validate(input, pos);

    if (status == QValidator::Invalid)    // Beep if invalid
         QApplication::beep();

    return status;
}
```

Lessons Learned: Object-Oriented Design

- C++ inheritance mechanism facilitates customizing and extending
  - Developer-generated classes
  - C++ class libraries
  - Qt class libraries
- Inheritance also facilitates code reuse
  - Can speed development and reduce the likelihood of injecting defects
  - Code reuse mechanisms include
    - Constructor initializers
    - Use of inherited methods and attributes
- Inheritance establishes the “is-a” relationship

Key Points

- Input validation is critical to development of robust software
- By blocking entry of invalid data values, Qt validation objects simplify the application logic
- The C++ inheritance mechanism may be used to extend and customize the validation mechanism for your application