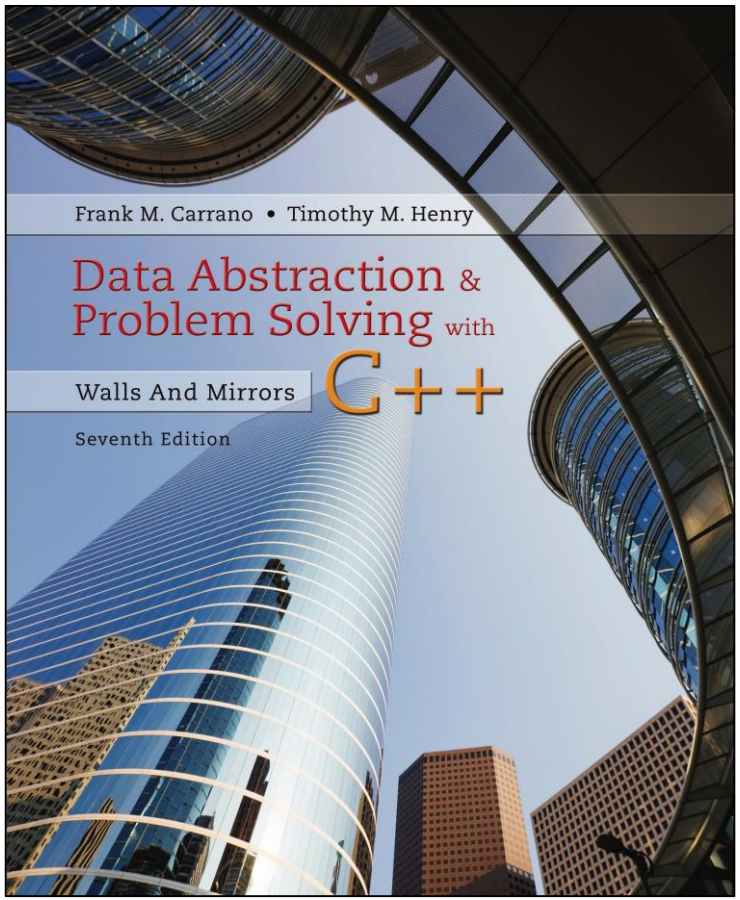
Data Abstraction & Problem Solving with C++: Walls and Mirrors

Seventh Edition



C++ Interlude 1

C++ Classes

Pearson

Copyright © 2017, 2013, 2007 Pearson Education, Inc. All Rights Reserved

A Problem to Solve (1 of 9)

- Consider a video game where a character carries three types of boxes
  - Plain box
  - Toy box
  - Magic box
- Plain box design
  - Get and Set public methods

Pearson

Copyright © 2017, 2013, 2007 Pearson Education, Inc. All Rights Reserved

A Problem to Solve (2 of 9)

Listing C1-1 The header file for the class **PlainBox**

```
1  /** @file PlainBox.h */
2
3  #ifndef PLAIN_BOX_
4  #define PLAIN_BOX_
5
6  // Set the type of data stored in the box
7  typedef double ItemType;
8  // Declaration for the class PlainBox
9  class PlainBox
10 {
11 private:
12    // Data field
13    ItemType item;
14
15 public:
```

Pearson

Copyright © 2017, 2013, 2007 Pearson Education, Inc. All Rights Reserved

A Problem to Solve (3 of 9)

Listing C1-1 [continued]

```
14
15  public:
16     // Default constructor
17     PlainBox();
18
19     // Parameterized constructor
20     PlainBox(const ItemType& theItem);
21
22     // Method to change the value of the data field
23     void setItem(const ItemType& theItem);
24
25     // Method to get the value of the data field
26     ItemType getItem() const;
27  }; // end PlainBox
28  #endif
```

Pearson

Copyright © 2017, 2013, 2007 Pearson Education, Inc. All Rights Reserved

A Problem to Solve (4 of 9)

- Elements of the class
  - Private data fields
  - Constructors, destructors
  - Methods
  - Use of #ifndef, #define, and #endif preprocessor directives
  - Use of initializers
  - Use of typedef
  - Inheritance

Copyright © 2017, 2013, 2007 Pearson Education, Inc. All Rights Reserved

A Problem to Solve (5 of 9)

Listing C1-2 Implementation file for the **PlainBox** class

```
1  /** @file PlainBox.cpp */
2
3  #include "PlainBox.h"
4
5  PlainBox::PlainBox()
6  {
7  }  // end default constructor
8  PlainBox::PlainBox(const ItemType& theItem)
9  {
10    item = theItem;
11 }  // end constructor
12 void PlainBox::setItem(const ItemType& theItem)
13 {
14    item = theItem;
15 }  // end setItem
16
17 ItemType PlainBox::getItem() const
18 {
19    return item;
20 }  // end getItem
```

Pearson

Copyright © 2017, 2013, 2007 Pearson Education, Inc. All Rights Reserved

A Problem to Solve (6 of 9)

Listing C1-3 Template header file for the **PlainBox** class

```
1  /** @file PlainBox.h */
2
3  #ifndef PLAIN_BOX_
4  #define PLAIN_BOX_
5
6  template<class ItemType> // Indicates this is a template definition
7
8  // Declaration for the class PlainBox
9  class  PlainBox
10 {
11 private:
12    // Data field
13    ItemType item;
14 public:
15    // Default constructor
16    PlainBox();
```

Pearson

Copyright © 2017, 2013, 2007 Pearson Education, Inc. All Rights Reserved

A Problem to Solve (7 of 9)

Listing C1-3 [continued]

```
16      PlainBox();
17
18      // Parameterized constructor
19      PlainBox(const ItemType& theItem);
20
21      // Mutator method that can change the value of the data field
22      void setItem(const ItemType& theItem);
23
24      // Accessor method to get the value of the data field
25      ItemType getItem() const;
26  }; // end PlainBox
27
28  #include "PlainBox.cpp" // Include the implementation file
29  #endif
```

Pearson

Copyright © 2017, 2013, 2007 Pearson Education, Inc. All Rights Reserved

A Problem to Solve (8 of 9)

Listing C1-4 Implementation file for the **PlainBox** template class

```
1  /** @file PlainBox.cpp */
2  #include "PlainBox.h"
3
4  template<class ItemType>
5  PlainBox<ItemType>::PlainBox()
6  {
7  }  // end default constructor
8
9  template<class ItemType>
10 PlainBox<ItemType>::PlainBox(const ItemType& theItem) : item(theItem)
11 {
12 }  // end constructor
13
```

Pearson

Copyright © 2017, 2013, 2007 Pearson Education, Inc. All Rights Reserved

A Problem to Solve (9 of 9)

Listing C1-4 [continued]

```
14  template<class ItemType>
15  void PlainBox<ItemType>::setItem(const ItemType& theItem)
16  {
17     item = theItem;
18  }  // end setItem
19
20  template<class ItemType>
21  ItemType PlainBox<ItemType>::getItem() const
22  {
23     return item;
24  }  // end getItem
```

Pearson

Copyright © 2017, 2013, 2007 Pearson Education, Inc. All Rights Reserved

Base Classes and Derived Classes (1 of 3)

- Use **PlainBox** as a base class, or superclass
- The **ToyBox** class is the derived class, or subclass, of the **PlainBox**
- Derived class inherits
  - All the members of its base class,
  - (Except the constructors and destructor)

Pearson

Copyright © 2017, 2013, 2007 Pearson Education, Inc. All Rights Reserved

Base Classes and Derived Classes (2 of 3)

Listing C1-5 Template header file for the class **ToyBox**

```
1  /** @file ToyBox.h */
2
3  #ifndef TOY_BOX_
4  #define TOY_BOX_
5  #include "PlainBox.h"
6
7  enum Color {BLACK, RED, BLUE, GREEN, YELLOW, WHITE};
8
9  template<class ItemType>
10 class ToyBox : public PlainBox<ItemType>
11 {
12 private:
13    Color boxColor;
14
15 public:
16    ToyBox();
17    ToyBox(const Color& theColor);
18    ToyBox(const ItemType& theItem, const Color& theColor);
19    Color getColor() const;
20 }; // end ToyBox
21 #include "ToyBox.cpp"
22 #endif
```

Pearson

Copyright © 2017, 2013, 2007 Pearson Education, Inc. All Rights Reserved

Base Classes and Derived Classes (3 of 3)

Listing C1-6 Implementation file for the class **ToyBox**

```
1  /** @file ToyBox.cpp */
2
3  #include "ToyBox.h"
4
5  template<class ItemType>
6  ToyBox<ItemType>::ToyBox() : boxColor(BLACK)
7  {
8  }  // end default constructor
9
10 template<class ItemType>
11 ToyBox<ItemType>::ToyBox(const Color& theColor) : boxColor(theColor)
12 {
13 }  // end constructor
14
15 template<class ItemType>
16 ToyBox<ItemType>::ToyBox(const ItemType& theItem, const Color& theColor)
17                         : PlainBox<ItemType>(theItem), boxColor(theColor)
18 {
19 }  // end constructor
20
21 template<class  ItemType>
22 Color ToyBox<ItemType>::getColor() const
23 {
24    return boxColor;
25 }  // end getColor
```

Pearson

Copyright © 2017, 2013, 2007 Pearson Education, Inc. All Rights Reserved

Overriding Base-Class Methods (1 of 3)

- You can add as many new members to derived class as desired
- You can redefine inherited methods
  - Called overriding a base-class method.
- A method overrides a base-class method when
  - The two methods have the same name and parameter declarations

Pearson

Copyright © 2017, 2013, 2007 Pearson Education, Inc. All Rights Reserved

Overriding Base-Class Methods (2 of 3)

Listing C1-7 Header file for the class **MagicBox**

```
1  /** @file MagicBox.h */
2
3  #ifndef MAGIC_BOX_
4  #define MAGIC_BOX_
5  #include "PlainBox.h"
6
7  template<class ItemType>
8  class MagicBox: public PlainBox<ItemType>
9  {
10 private:
11    bool firstItemStored;
12
13 public:
14    MagicBox();
15    MagicBox(const ItemType& theItem);
16    void setItem(const ItemType& theItem);
17 }; // end MagicBox
18 #include "MagicBox.cpp"
19 #endif
```

Pearson

Copyright © 2017, 2013, 2007 Pearson Education, Inc. All Rights Reserved

Overriding Base-Class Methods (3 of 3)

Listing C1-8 Implementation file for the class **MagicBox**

```
1  /** @file MagicBox.cpp */
2  #include "MagicBox.h"
3  template<class ItemType>
4  MagicBox<ItemType>::MagicBox(): firstItemStored(false)
5  {
6     // PlainBox constructor is called implicitly.
7     // Box has no magic initially
8  }  // end default constructor
9
10 template<class ItemType>
11 MagicBox<ItemType>::MagicBox(const ItemType& theItem): firstItemStored(false)
12 {
13    // Box has no magic initially
14    setItem(theItem); // Calls MagicBox version of setItem
15    // Box has magic now
16 }  // end constructor
17
18 template<class ItemType>
19 void MagicBox<ItemType>::setItem(const ItemType& theItem)
20 {
21    if (!firstItemStored)
22    {
23       PlainBox<ItemType>::setItem(theItem);
24       firstItemStored = true; // Box has magic now
25    }  // end if
26 }  // end setItem
```

Pearson

Copyright © 2017, 2013, 2007 Pearson Education, Inc. All Rights Reserved

Virtual Methods, Abstract Classes (1 of 2)

- Using keyword **virtual** in front of the prototype
  - Tells the C++ compiler that the code this method executes is determined at runtime
- Pure virtual method
  - Virtual method that has no implementation
- Abstract class
  - Has at least one pure virtual method

Pearson

Copyright © 2017, 2013, 2007 Pearson Education, Inc. All Rights Reserved

Virtual Methods, Abstract Classes (2 of 2)

Listing C1-9 An abstract class that is an interface for the ADT box

```
1  /** @file BoxInterface.h */
2
3  #ifndef BOX_INTERFACE_
4  #define BOX_INTERFACE_
5
6  template <class ItemType>
7  class  BoxInterface
8  {
9  public:
10    virtual void  setItem(const ItemType& theItem) = 0;
11    virtual ItemType getItem() const = 0;
12    virtual ~BoxInterface() {} // C++ Interlude 2 explains virtual destructors
13 }; // end BoxInterface
14 #endif
```

Pearson

Copyright © 2017, 2013, 2007 Pearson Education, Inc. All Rights Reserved

Copyright

This work is protected by United States copyright laws and is provided solely for the use of instructors in teaching their courses and assessing student learning. Dissemination or sale of any part of this work (including on the World Wide Web) will destroy the integrity of the work and is not permitted. The work and materials from it should never be made available to students except by instructors using the accompanying text in their classes. All recipients of this work are expected to abide by these restrictions and to honor the intended pedagogical purposes and the needs of other instructors who rely on these materials.

Pearson

Copyright © 2017, 2013, 2007 Pearson Education, Inc. All Rights Reserved