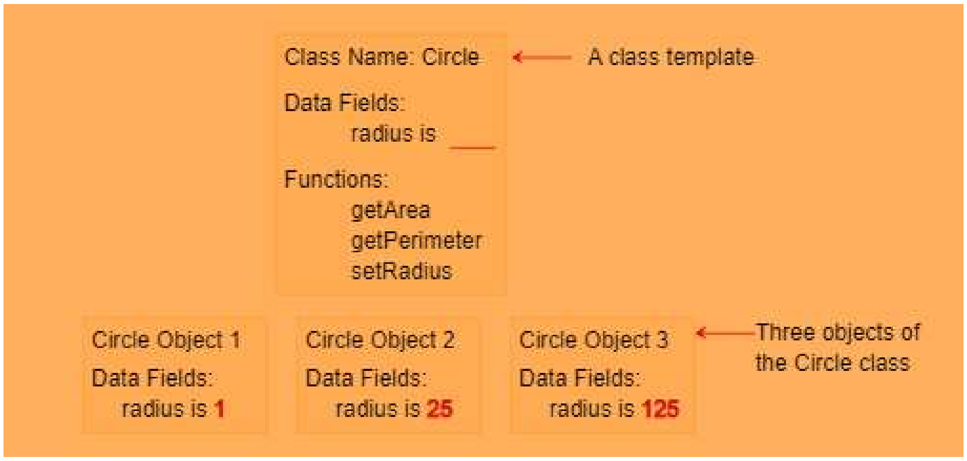
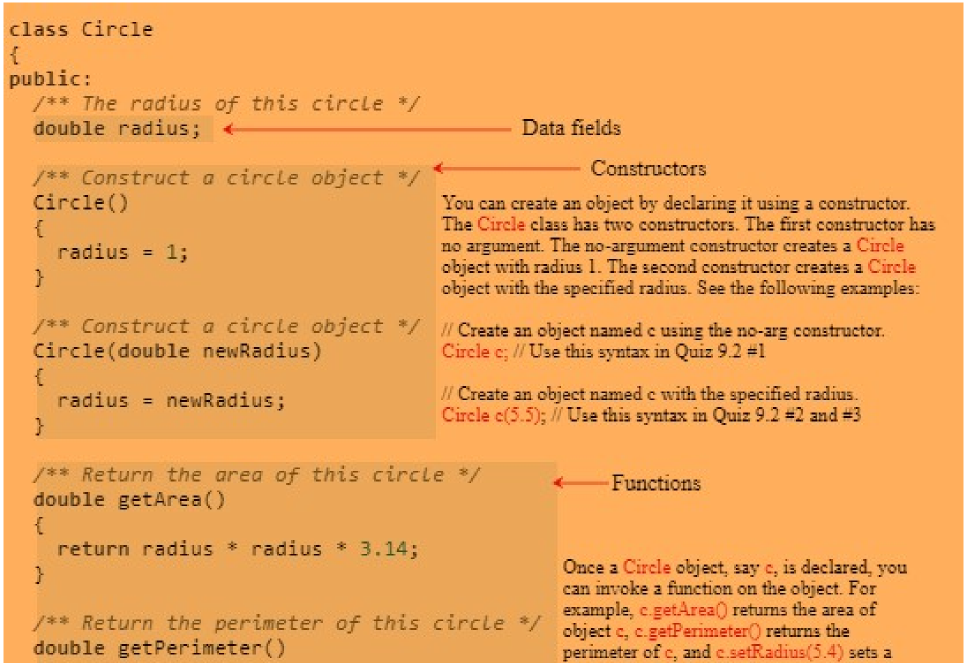
9.2 Defining Classes for Objects

Monday, February 27, 2023 10:09 AM

- Class defines properties and behaviors of objects
- OOP (object oriented pgrming) uses objects which reps entity in real world and can be distinctly id'd
  - State of an object (aka properties aka attributes) repped by data fields w/ current values
  - Behavior of an object (aka actions) defined by functions, to invoke function on an object is asking object to do an action
- Objects of same type defined using common class
- Class is template/blueprint/contract that defines what object's data fields and functions will be
- Object is instance of a class, can make many instances of a class
- Making an instance referred to as instantiation
- Object and instance usually interchangeable
- Classes and objects is like apple pie recipe and apple pie (respectively)
-
- 

- C++ class uses cars to define data fields and functions to define behaviors, class can also make functions of special type (constructors), invoked when new object made
- Constructor is special type of function, can perform any action, but they are designed to perform initializing actions, like initializing the data fields of objects
- 

```
class Circle
{
public:
  /** The radius of this circle */
  double radius;

  /** Construct a circle object */
  Circle()
  {
    radius = 1;
  }

  /** Construct a circle object */
  Circle(double newRadius)
  {
    radius = newRadius;
  }

  /** Return the area of this circle */
  double getArea()
  {
    return radius * radius * 3.14;
  }

  /** Return the perimeter of this circle */
  double getPerimeter()
```

Data fields

Constructors

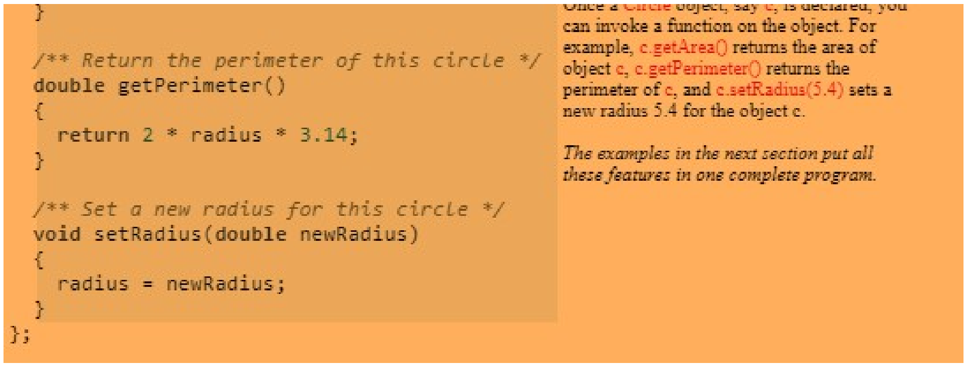
You can create an object by declaring it using a constructor. The Circle class has two constructors. The first constructor has no argument. The no-argument constructor creates a Circle object with radius 1. The second constructor creates a Circle object with the specified radius. See the following examples:

// Create an object named c using the no-arg constructor.
Circle c; // Use this syntax in Quiz 9.2 #1

// Create an object named c with the specified radius.
Circle c(5.5); // Use this syntax in Quiz 9.2 #2 and #3

Functions

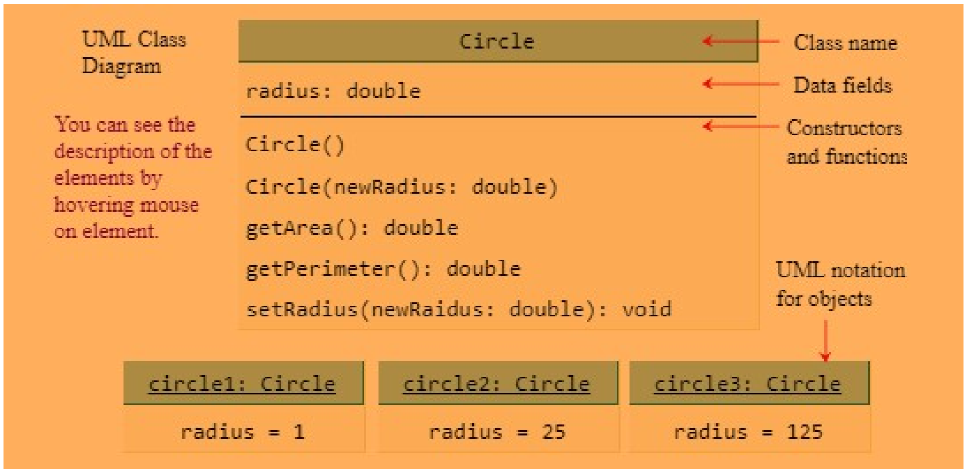
Once a Circle object, say c, is declared, you can invoke a function on the object. For example, c.getArea() returns the area of object c, c.getPerimeter() returns the perimeter of c, and c.setRadius(5.4) sets a



```
  }

  /** Return the perimeter of this circle */
  double getPerimeter()
  {
    return 2 * radius * 3.14;
  }

  /** Set a new radius for this circle */
  void setRadius(double newRadius)
  {
    radius = newRadius;
  }
};
```

Once a Circle object, say c, is declared, you can invoke a function on the object. For example, c.getArea() returns the area of object c, c.getPerimeter() returns the perimeter of c, and c.setRadius(5.4) sets a new radius 5.4 for the object c.

The examples in the next section put all these features in one complete program.

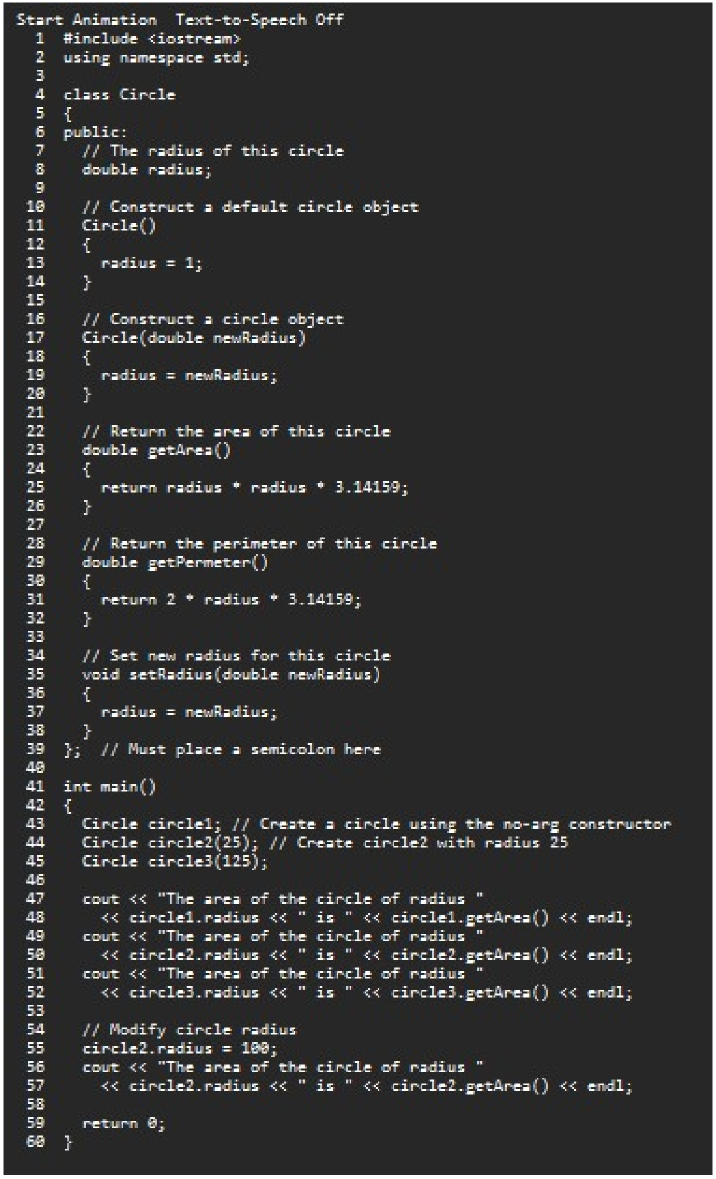
-
- 

You can see the description of the elements by hovering mouse on element.

- Illustration of class and objects can be standardized using UML notation, class diagram
- Data field denoted as :
  dataFieldName: dataFieldType
- Constructor:
  ClassName(parameterName: parameterType)
- Function:
  functionName(parameterName: parameterType): returnType

9.3 Ex: Defining Classes and Creating Objects

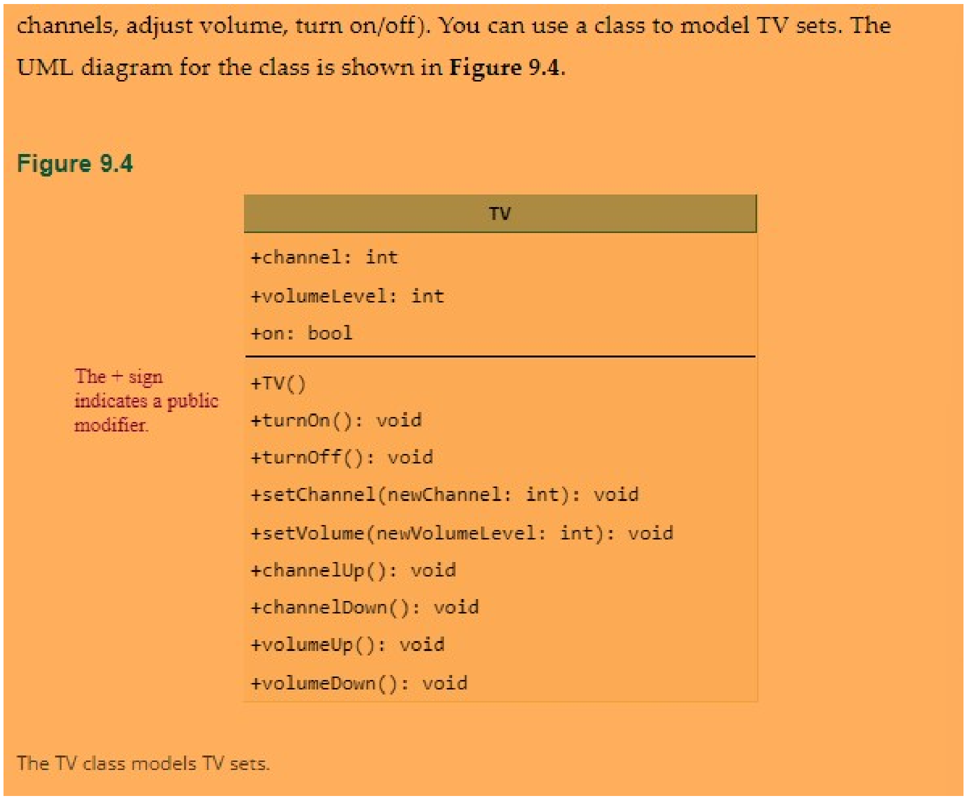
Saturday, March 4, 2023 10:15 AM

- Classes are definitions for objects and objects are created from classes
- This pgrm makes 3 circle objects w/ diff radius'
- 

Start Animation Text-to-Speech Off

```cpp
 1  #include <iostream>
 2  using namespace std;
 3
 4  class Circle
 5  {
 6  public:
 7    // The radius of this circle
 8    double radius;
 9
10    // Construct a default circle object
11    Circle()
12    {
13      radius = 1;
14    }
15
16    // Construct a circle object
17    Circle(double newRadius)
18    {
19      radius = newRadius;
20    }
21
22    // Return the area of this circle
23    double getArea()
24    {
25      return radius * radius * 3.14159;
26    }
27
28    // Return the perimeter of this circle
29    double getPermeter()
30    {
31      return 2 * radius * 3.14159;
32    }
33
34    // Set new radius for this circle
35    void setRadius(double newRadius)
36    {
37      radius = newRadius;
38    }
39  };  // Must place a semicolon here
40
41  int main()
42  {
43    Circle circle1; // Create a circle using the no-arg constructor
44    Circle circle2(25); // Create circle2 with radius 25
45    Circle circle3(125);
46
47    cout << "The area of the circle of radius "
48      << circle1.radius << " is " << circle1.getArea() << endl;
49    cout << "The area of the circle of radius "
50      << circle2.radius << " is " << circle2.getArea() << endl;
51    cout << "The area of the circle of radius "
52      << circle3.radius << " is " << circle3.getArea() << endl;
53
54    // Modify circle radius
55    circle2.radius = 100;
56    cout << "The area of the circle of radius "
57      << circle2.radius << " is " << circle2.getArea() << endl;
58
59    return 0;
60  }
```

- Make sure to place semicolon after class, on ^ pgrm, it's on line 39
- 'public' keyword (ln 6) shows all data fields & constructors & functions can be accessed from the objects of the class
    - If don't use public keyword, then its assumed private, cover later
-



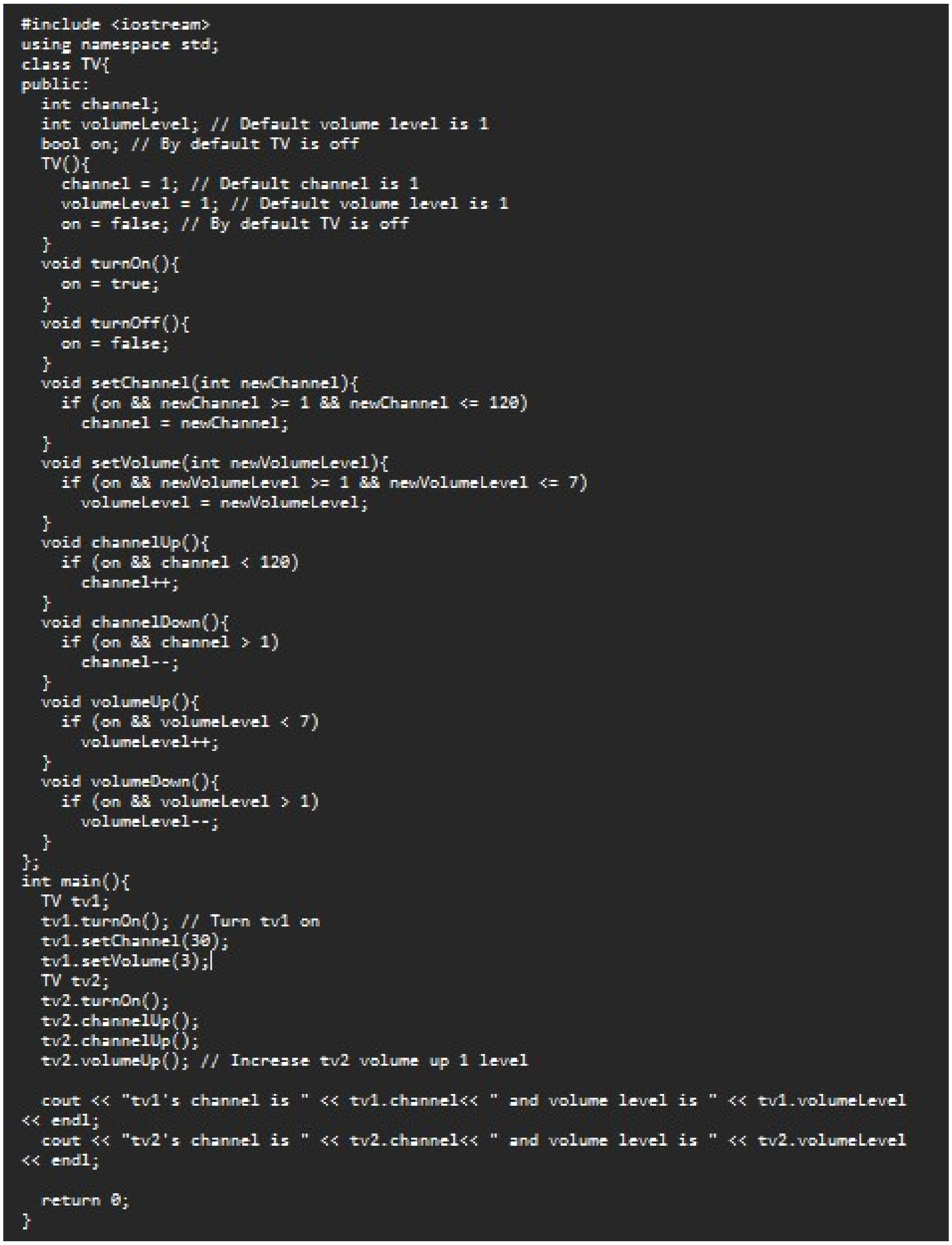
channels, adjust volume, turn on/off). You can use a class to model TV sets. The UML diagram for the class is shown in **Figure 9.4**.

Figure 9.4

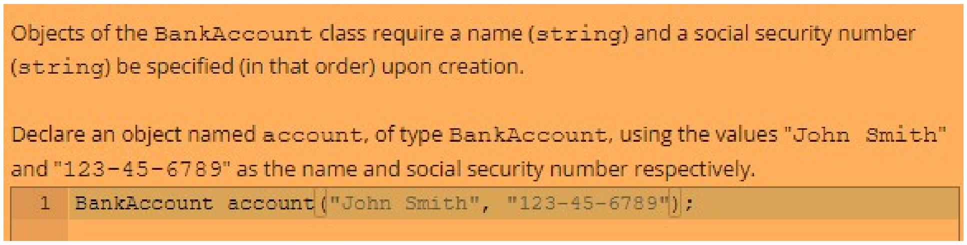
The + sign indicates a public modifier.

```
TV
+channel: int
+volumeLevel: int
+on: bool
+TV()
+turnOn(): void
+turnOff(): void
+setChannel(newChannel: int): void
+setVolume(newVolumeLevel: int): void
+channelUp(): void
+channelDown(): void
+volumeUp(): void
+volumeDown(): void
```

The TV class models TV sets.

- 

```
#include <iostream>
using namespace std;
class TV{
public:
  int channel;
  int volumeLevel; // Default volume level is 1
  bool on; // By default TV is off
  TV(){
    channel = 1; // Default channel is 1
    volumeLevel = 1; // Default volume level is 1
    on = false; // By default TV is off
  }
  void turnOn(){
    on = true;
  }
  void turnOff(){
    on = false;
  }
  void setChannel(int newChannel){
    if (on && newChannel >= 1 && newChannel <= 120)
      channel = newChannel;
  }
  void setVolume(int newVolumeLevel){
    if (on && newVolumeLevel >= 1 && newVolumeLevel <= 7)
      volumeLevel = newVolumeLevel;
  }
  void channelUp(){
    if (on && channel < 120)
      channel++;
  }
  void channelDown(){
    if (on && channel > 1)
      channel--;
  }
  void volumeUp(){
    if (on && volumeLevel < 7)
      volumeLevel++;
  }
  void volumeDown(){
    if (on && volumeLevel > 1)
      volumeLevel--;
  }
};
int main(){
  TV tv1;
  tv1.turnOn(); // Turn tv1 on
  tv1.setChannel(30);
  tv1.setVolume(3);
  TV tv2;
  tv2.turnOn();
  tv2.channelUp();
  tv2.channelUp();
  tv2.volumeUp(); // Increase tv2 volume up 1 level

  cout << "tv1's channel is " << tv1.channel<< " and volume level is " << tv1.volumeLevel
<< endl;
  cout << "tv2's channel is " << tv2.channel<< " and volume level is " << tv2.volumeLevel
<< endl;

  return 0;
}
```

- Syntax is important, tv1.turnOn()
-
- 

  Objects of the `BankAccount` class require a name (`string`) and a social security number (`string`) be specified (in that order) upon creation.

  Declare an object named `account`, of type `BankAccount`, using the values "`John Smith`" and "`123-45-6789`" as the name and social security number respectively.

  ```
  BankAccount account("John Smith", "123-45-6789");
  ```
-

9.4 Constructors

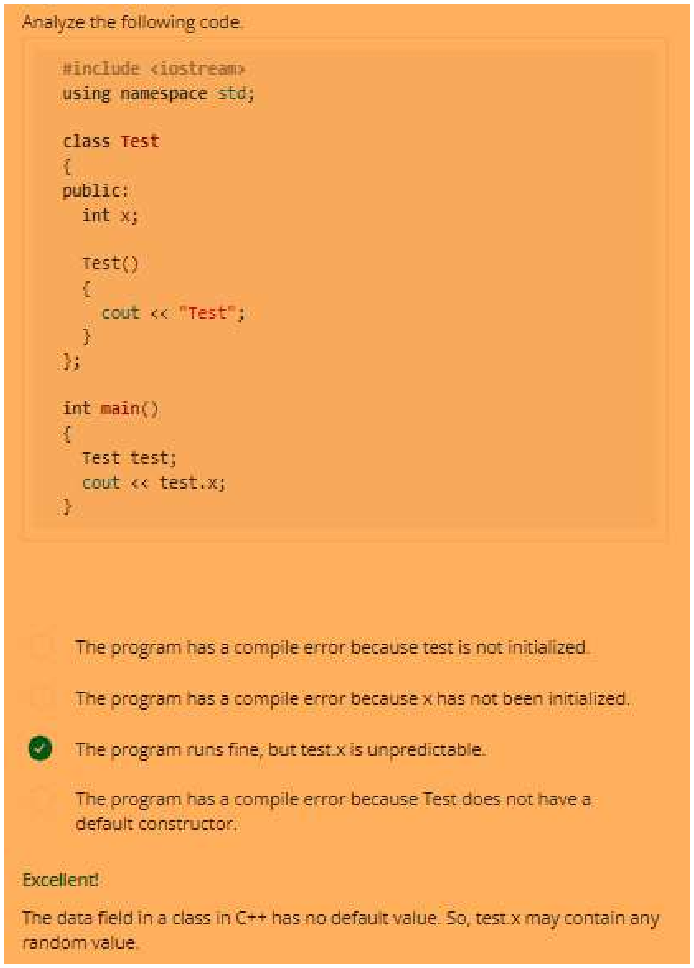
Saturday, March 4, 2023 10:52 AM

- A constructor is invoked to make an object
- They are special types of functions, 3 peculiarities:
  - Must have the same name as the class itself
  - Don't have a return type, not even void
  - Invoked when an object is made, they play role of initializing objects
- Constructor has same name as the defining class
- Like reg functions, can be overloaded (multiple constructors w/ same name, but diff signatures)
- DON'T PUT VOID IN FRONT OF CONSTRUCTOR
- Usually used for initializing data fields, usually w/out initial val
- Can also initialize a member data field when declared (member initializer)
- No-arg or no-argument constructor
- Class w/out constructors, called default constructors, provided auto only if no constructors are explicitly defined in the class

9.5 Constructing & Using Objects

Saturday, March 4, 2023 11:15 AM

- An object's data and functions can be accessed thru the dot (.) operator via the object's name
- Constructor invokes when object made, syntax to make object using no arg constructor is : ClassName objectName;
- Syntax to make an object using a constructor with arguments is : ClassName objectName(arguments);
- In oop, objects member refers to its data fields and functions, new made objects are allocated in the mem
- After object made, data can be accessed and functions invoked using the dot operator(.), aka object member access operator:
  - objectName.dataField references a data field in the object
  - objectName.function(arguments) invokes a function on the object
- Datafield referred to as an instance member variable, aka instance variable, bc it dependent on specific instance
- Instance member function aka instance function bc can invoke only specific instance
- Object which instance function invoked called calling object
- U can use primitive data types to define vars, can also use class names to declare object names, so in a sense, a class is also a data type
- In C++, can use assignment operator = to copy contents from one object to other, by default, each data field of one object is copies to its counterpart in the other object
- Object names are like array names, once object name is declared, it reps an object, can't be reassigned to rep another object, so in a sense, object name is a constant, though contents of object may change, memberwise copy can change object's contents but not its name
- Object w/ data can invoke functions, may think that object is quite large, but naw, data physically stored in object, functions not, since functions shared by all objects of same class, compiler makes just 1 copy for sharing, can find actual size of object using sizeof function
- Usually make named object and later access its members thru its name, occasionally can make object and only use once, if so don't have to name it (anonymous objects)
  - Syntax to make anonymous object using no-arg constructor: ClassName()
  - Syntax to make anonymous object w/ constructor w/ arg is: ClassName(arguments)
-

- 

  Analyze the following code.

```
#include <iostream>
using namespace std;

class Test
{
public:
  int x;

  Test()
  {
    cout << "Test";
  }
};

int main()
{
  Test test;
  cout << test.x;
}
```

  ○ The program has a compile error because test is not initialized.

  ○ The program has a compile error because x has not been initialized.

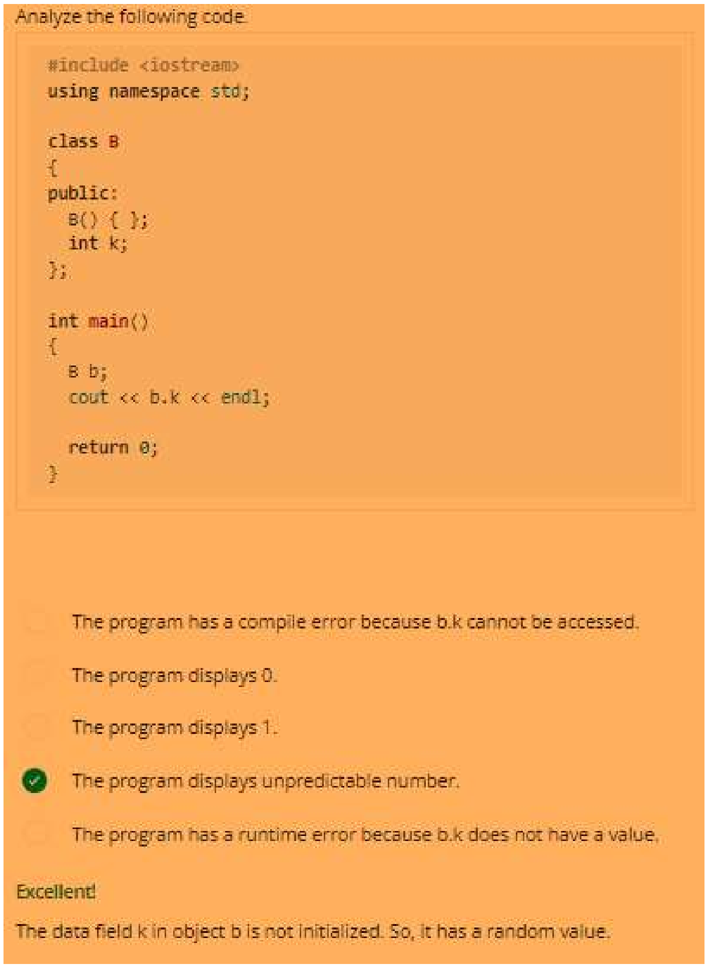
  ✓ The program runs fine, but test.x is unpredictable.

  ○ The program has a compile error because Test does not have a default constructor.

  Excellent!

  The data field in a class in C++ has no default value. So, test.x may contain any random value.

-



Analyze the following code.

```
#include <iostream>
using namespace std;

class B
{
public:
  B() { };
  int k;
};

int main()
{
  B b;
  cout << b.k << endl;

  return 0;
}
```

○ The program has a compile error because b.k cannot be accessed.

○ The program displays 0.

○ The program displays 1.

✓ The program displays unpredictable number.

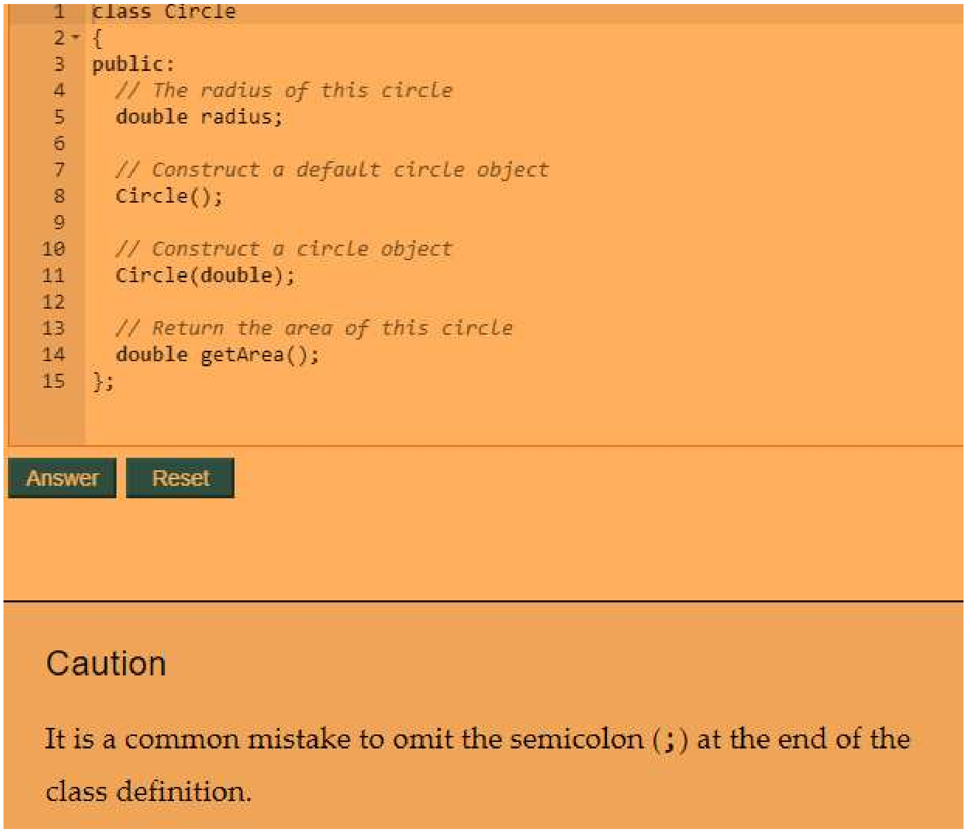
○ The program has a runtime error because b.k does not have a value.

Excellent!

The data field k in object b is not initialized. So, it has a random value.

9.6 Separating Class Definition from implementation

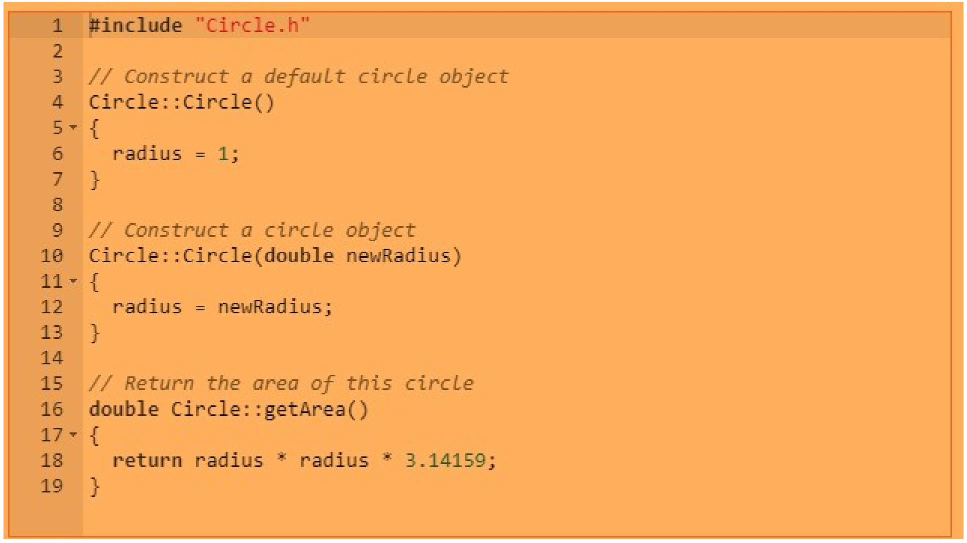
Saturday, March 4, 2023 12:19 PM

- Separating class definition from class implementation makes the class easy to maintain
- Class def describes the contract of the class and the class implementation carries out the contract
- Class def basically lists all data fields, constructor prototypes, and function prototypes
- Class implementation implements the constructors and functions
- Class def and imp can be in 2 sep files, both files should have same name but diff extension names
  - Class def file has extension name .h (h means header)
  - Class imp file has extension name .cpp
- Circle.h
- 

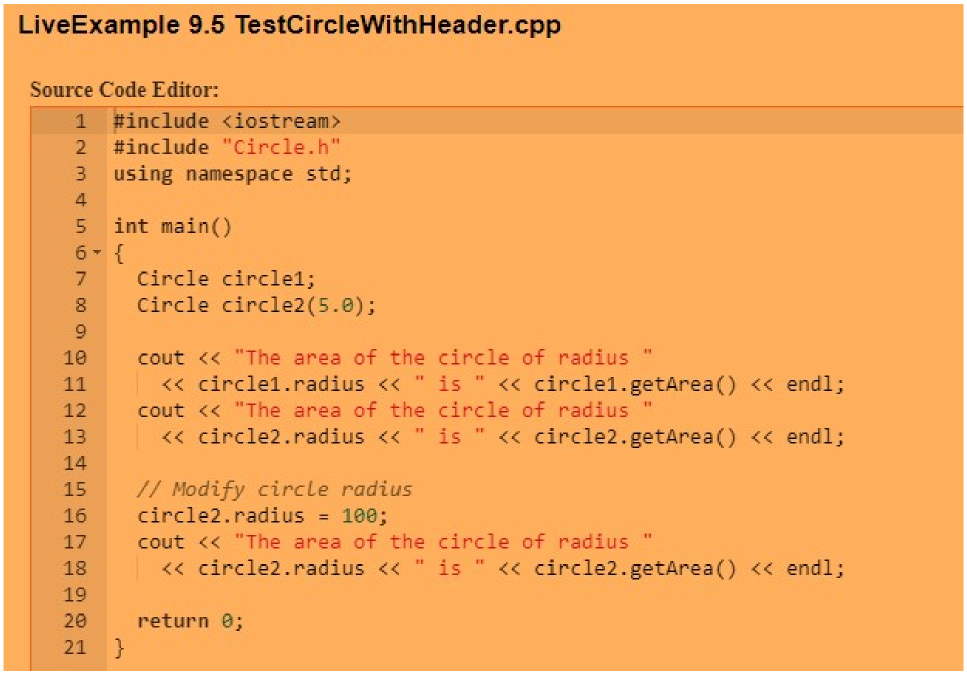
```
class Circle
{
public:
  // The radius of this circle
  double radius;

  // Construct a default circle object
  Circle();

  // Construct a circle object
  Circle(double);

  // Return the area of this circle
  double getArea();
};
```

Caution

It is a common mistake to omit the semicolon (;) at the end of the class definition.

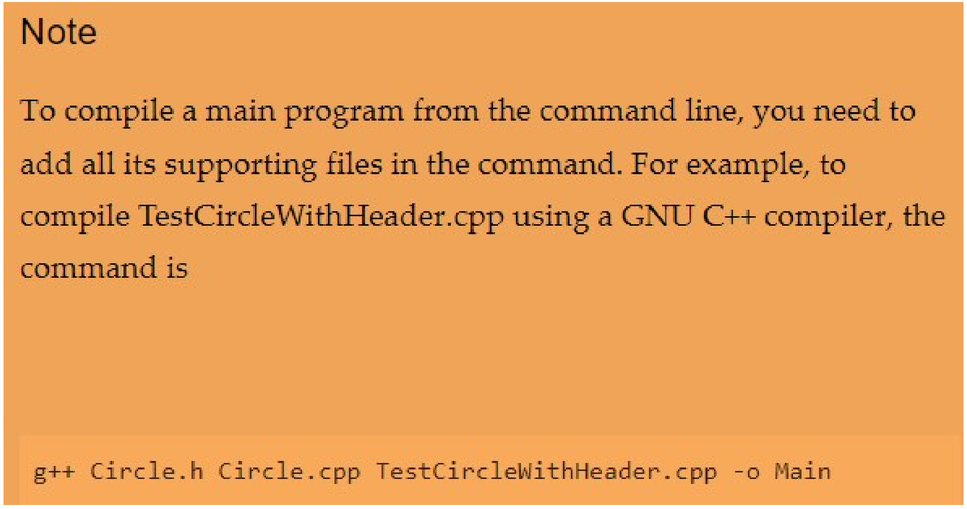
- Just noticed that private has colon right behind it
- Circle.cpp
- 

```
#include "Circle.h"

// Construct a default circle object
Circle::Circle()
{
  radius = 1;
}

// Construct a circle object
Circle::Circle(double newRadius)
{
  radius = newRadius;
}

// Return the area of this circle
double Circle::getArea()
{
  return radius * radius * 3.14159;
}
```

- :: symbol is binary scope resolution operator, specifies scope of class member in a class
- So Circle:: b4 each constructor and function in the Circle class tells compiler that these constructors and functions are defined in the Circle class
- 

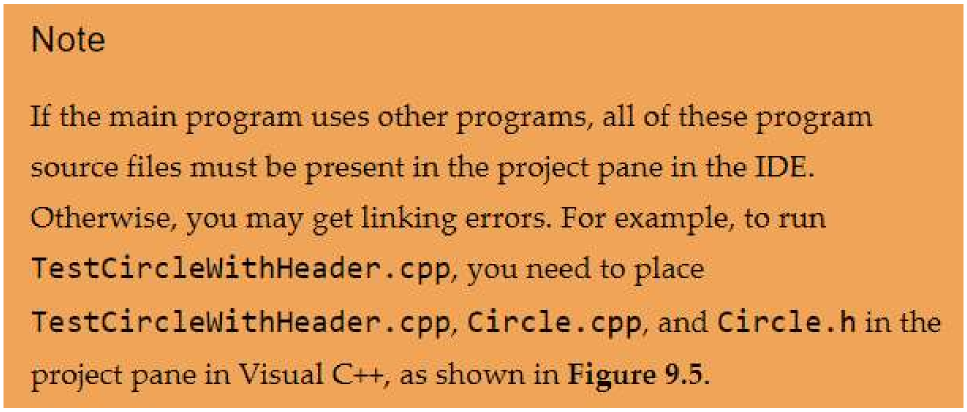
  LiveExample 9.5 TestCircleWithHeader.cpp

  Source Code Editor:

  ```
  #include <iostream>
  #include "Circle.h"
  using namespace std;

  int main()
  {
    Circle circle1;
    Circle circle2(5.0);

    cout << "The area of the circle of radius "
      << circle1.radius << " is " << circle1.getArea() << endl;
    cout << "The area of the circle of radius "
      << circle2.radius << " is " << circle2.getArea() << endl;

    // Modify circle radius
    circle2.radius = 100;
    cout << "The area of the circle of radius "
      << circle2.radius << " is " << circle2.getArea() << endl;

    return 0;
  }
  ```
- 2 benefits for separating class def from implementation
  - ○ Hides implementation from def, free to change implementation, client pgrm that uses the class doesn't need to change as long as the def not changed
  - ○ As software vendor, just provide customer w/ header file and class object code w/out revealing src code for implementing the class, protects intellectual property
  - ○ 

    Note

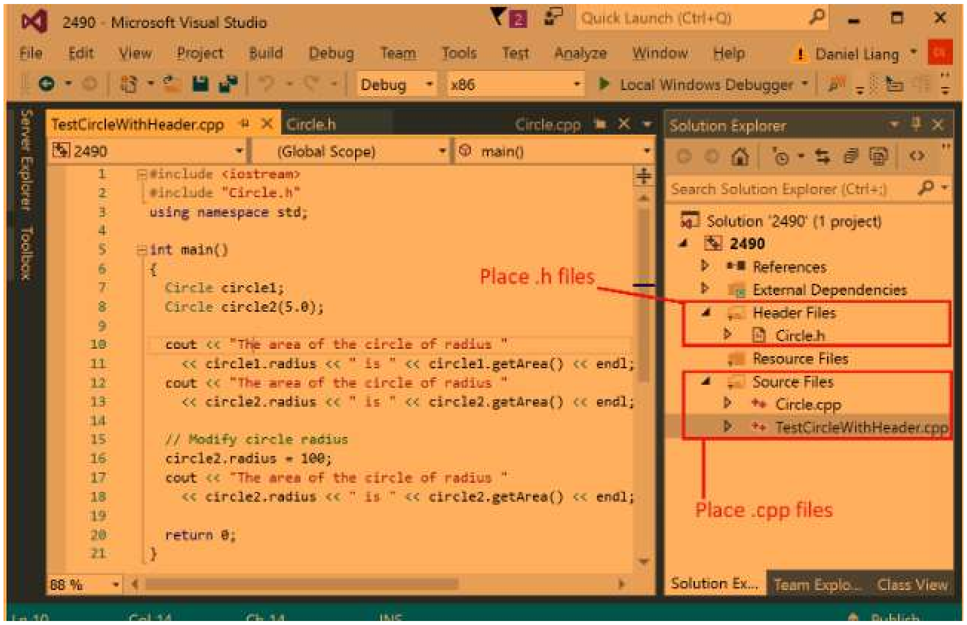
    To compile a main program from the command line, you need to add all its supporting files in the command. For example, to compile TestCircleWithHeader.cpp using a GNU C++ compiler, the command is

    ```
    g++ Circle.h Circle.cpp TestCircleWithHeader.cpp -o Main
    ```
  - ○
  - ○ 

    Note

    If the main program uses other programs, all of these program source files must be present in the project pane in the IDE. Otherwise, you may get linking errors. For example, to run `TestCircleWithHeader.cpp`, you need to place `TestCircleWithHeader.cpp`, `Circle.cpp`, and `Circle.h` in the project pane in Visual C++, as shown in **Figure 9.5**.

○



```cpp
#include <iostream>
#include "Circle.h"
using namespace std;

int main()
{
  Circle circle1;
  Circle circle2(5.0);

  cout << "The area of the circle of radius "
    << circle1.radius << " is " << circle1.getArea() << endl;
  cout << "The area of the circle of radius "
    << circle2.radius << " is " << circle2.getArea() << endl;

  // Modify circle radius
  circle2.radius = 100;
  cout << "The area of the circle of radius "
    << circle2.radius << " is " << circle2.getArea() << endl;

  return 0;
}
```

Place .h files

Place .cpp files

○

-

Which of the following statements are true?

○ C++ allows you to separate class declaration from implementation.

○ The class declaration describes the contract of the class and the class implementation implements the contract.

○ The class declaration and implementation are often in two separate files. Both files should have the same name, but with different extension names.

○ The class declaration file has an extension name .h and the class implementation file has an extension name .cpp.

✓ All of the above.

-

Which of the following statements are true?

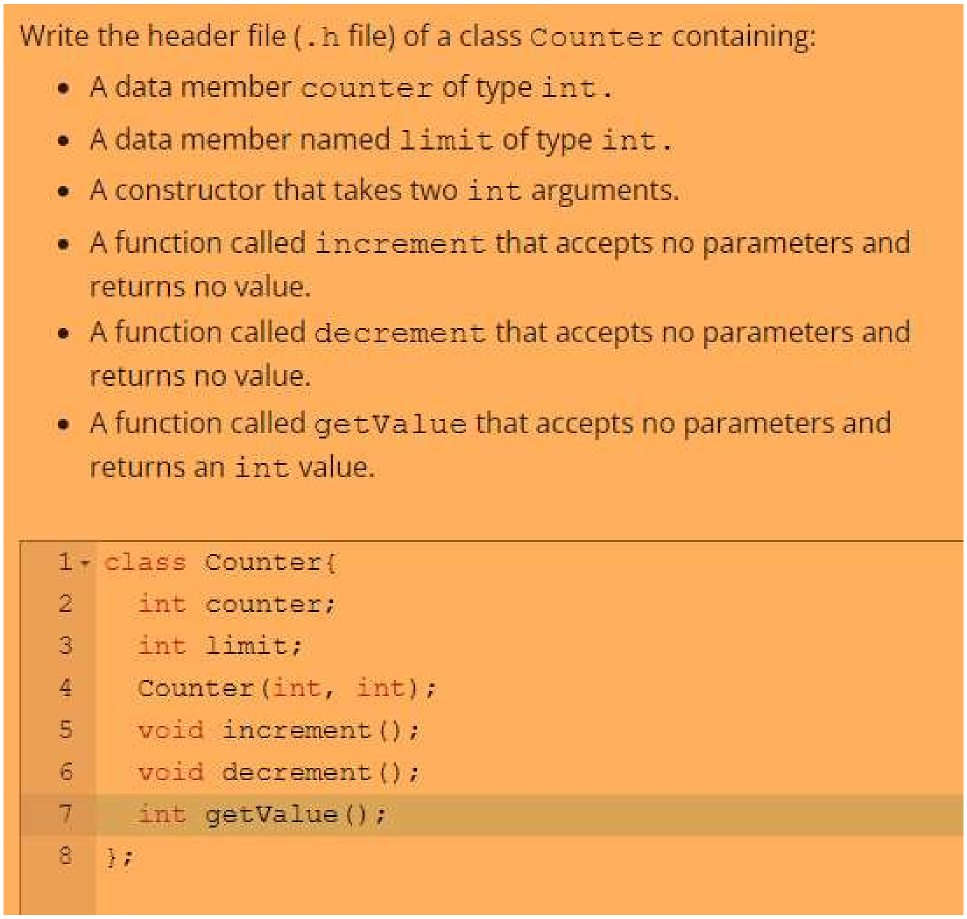
○ The :: symbol is called the scope operator.

○ The binary scope operator can be used as ClassName::member to tell the compiler that a member belongs to a class.

○ The unary scope operator can be used as ::var to tell the compiler that the variable is a global variable.

✓ All of the above.

-

- 

Write the header file (`.h` file) of a class `Counter` containing:

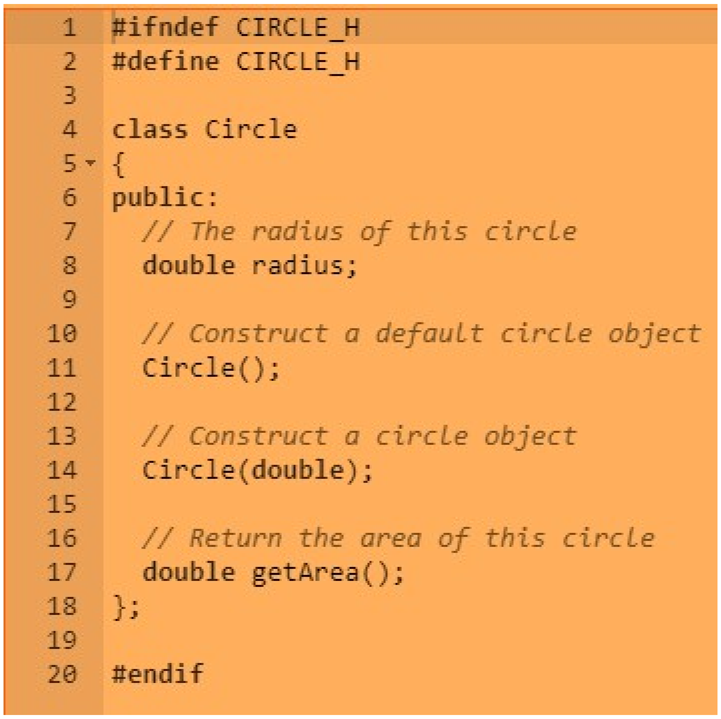
- A data member `counter` of type `int`.
- A data member named `limit` of type `int`.
- A constructor that takes two `int` arguments.
- A function called `increment` that accepts no parameters and returns no value.
- A function called `decrement` that accepts no parameters and returns no value.
- A function called `getValue` that accepts no parameters and returns an `int` value.

```
class Counter{
  int counter;
  int limit;
  Counter(int, int);
  void increment();
  void decrement();
  int getValue();
};
```

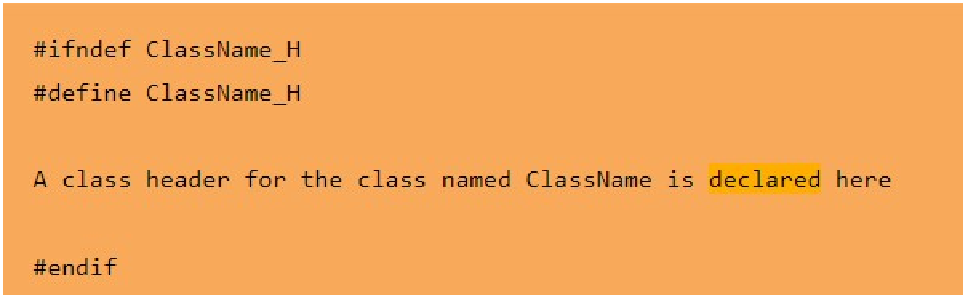
-

9.7 Preventing Multiple Inclusions

Saturday, March 4, 2023 12:31 PM

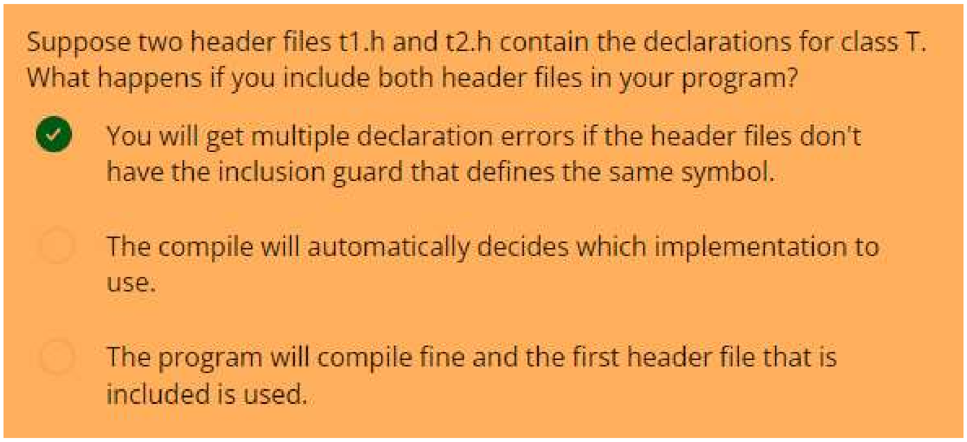
- Inclusion guard prevents header files to be included multiple times
- Common to include same header file in pgrm multiple time
- Preprocessor inserts contents of header file @ pos where header file is included, so if the header file has multiple headers that includes diff definitions, cause multiple-inclusion errors
- #ifndef directive and #define directive can be used to prevent a header file from being included multiple times, aka inclusion guard
-
- 

```
#ifndef CIRCLE_H
#define CIRCLE_H

class Circle
{
public:
  // The radius of this circle
  double radius;

  // Construct a default circle object
  Circle();

  // Construct a circle object
  Circle(double);

  // Return the area of this circle
  double getArea();
};

#endif
```

- # are preprocessor directives, interpreted by C++ preprocessor
- #indef stands for "if not defined"
- #endif indicates end of header file
- To avoid multiple inclusion errors, define class using template:
- 

```
#ifndef ClassName_H
#define ClassName_H

A class header for the class named ClassName is declared here

#endif
```

-

- 

  Suppose two header files t1.h and t2.h contain the declarations for class T. What happens if you include both header files in your program?

  ✓ You will get multiple declaration errors if the header files don't have the inclusion guard that defines the same symbol.

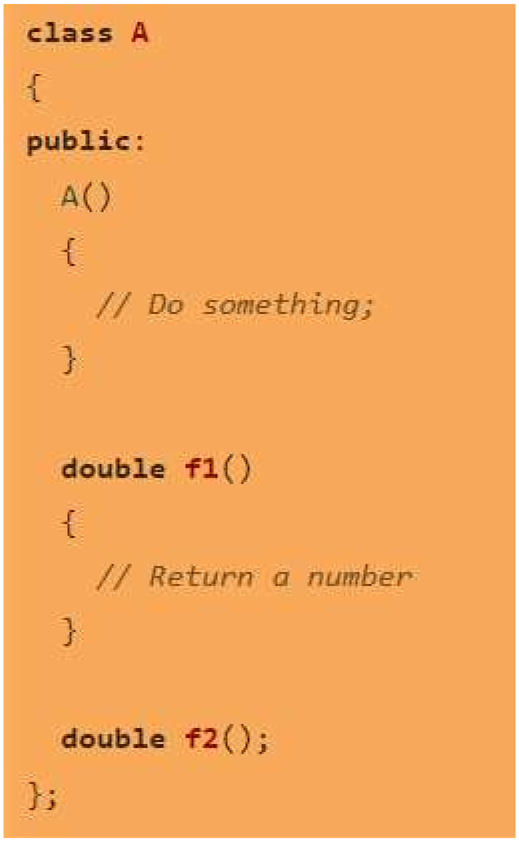
  ○ The compile will automatically decides which implementation to use.

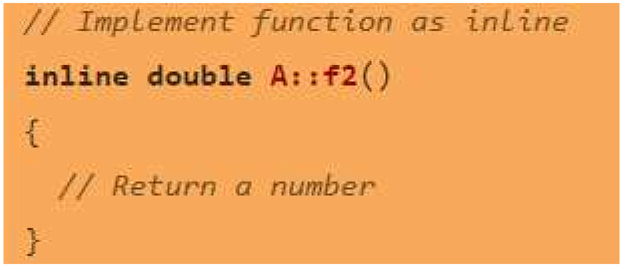
  ○ The program will compile fine and the first header file that is included is used.

-

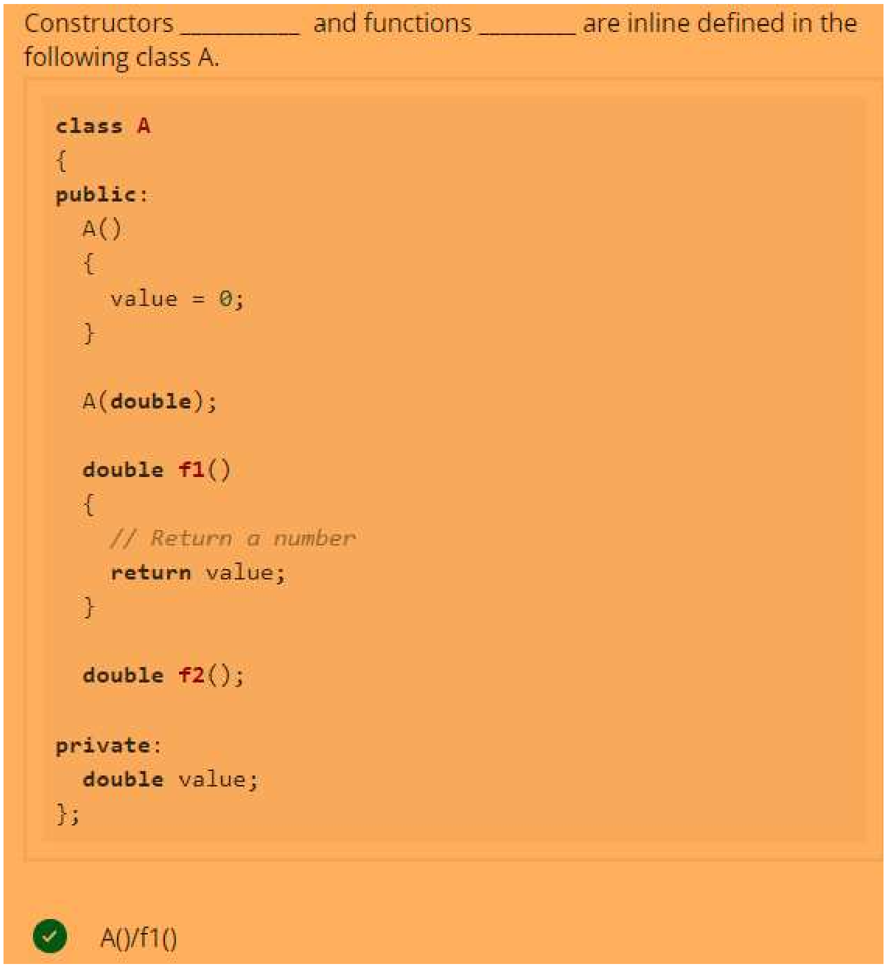
9.8 Inline Functions in Classes

Saturday, March 4, 2023 3:48 PM

- U can define short functions as inline functions, improve performance
- When function implemented inside a class definition, auto becomes inline function, aka inline definition
- 

  ```
  class A
  {
  public:
    A()
    {
      // Do something;
    }

    double f1()
    {
      // Return a number
    }

    double f2();
  };
  ```
- Apparently, function f1 is an inline function, bc dun inside of the class, but function f2 isn't
- Another way to define inline functions for classes is by implementation file
  - Ex: if want to define functionf2 as inline function, do this:
  - 

    ```
    // Implement function as inline
    inline double A::f2()
    {
      // Return a number
    }
    ```
- Use short functions for inline, not long functions
-

- 

Constructors _________ and functions ________ are inline defined in the following class A.

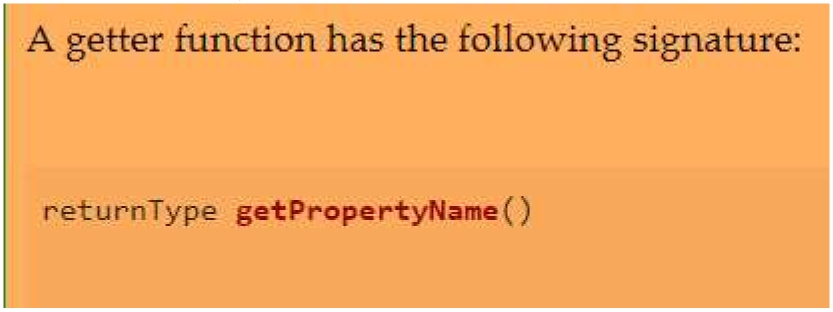
```
class A
{
public:
  A()
  {
    value = 0;
  }

  A(double);

  double f1()
  {
    // Return a number
    return value;
  }

  double f2();

private:
  double value;
};
```

✓ A()/f1()

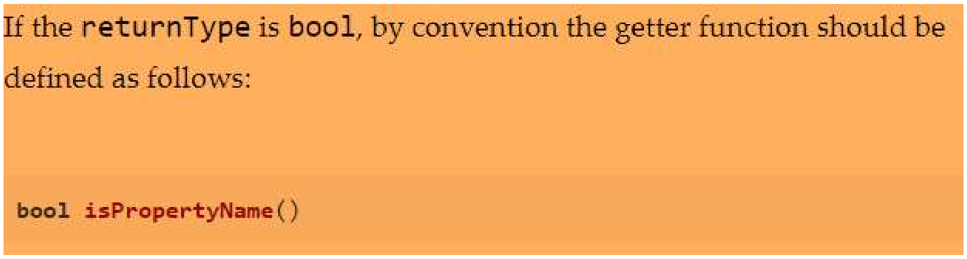
-

9.9 Data Field Encapsulation

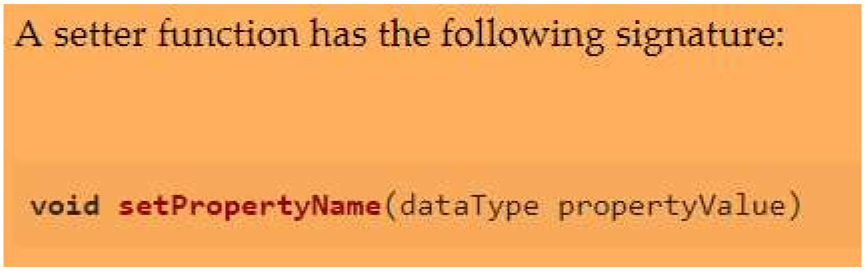
Saturday, March 4, 2023 4:01 PM

- Making data fields private protects data and makes the class easy to maintain
- Don't want to modify vals directly, not good practice
  - Data may be tampered with
  - Make class difficult to keep and vulnerable to bugs
- To prevent changes, declare private data field, use keyword 'private', aka data field encapsulation
- Private data field cant be accessed by object thru direct reference outside the class that defines the private field
- Sometimes need to get/mod data field, to access private data field, give getter function to return the field's cal, and setter function to set a new val
- Getter aka accessor, settor aka mutator
- 

  A getter function has the following signature:

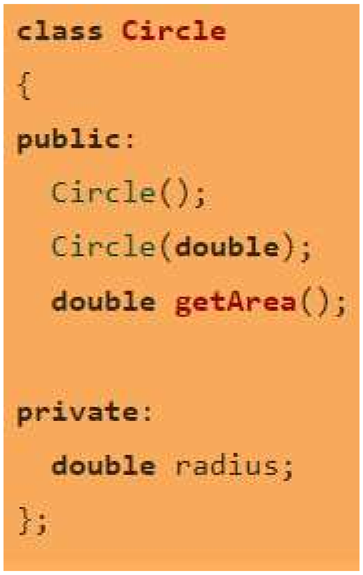
  ```
  returnType getPropertyName()
  ```
- 

  If the `returnType` is `bool`, by convention the getter function should be defined as follows:

  ```
  bool isPropertyName()
  ```
- 

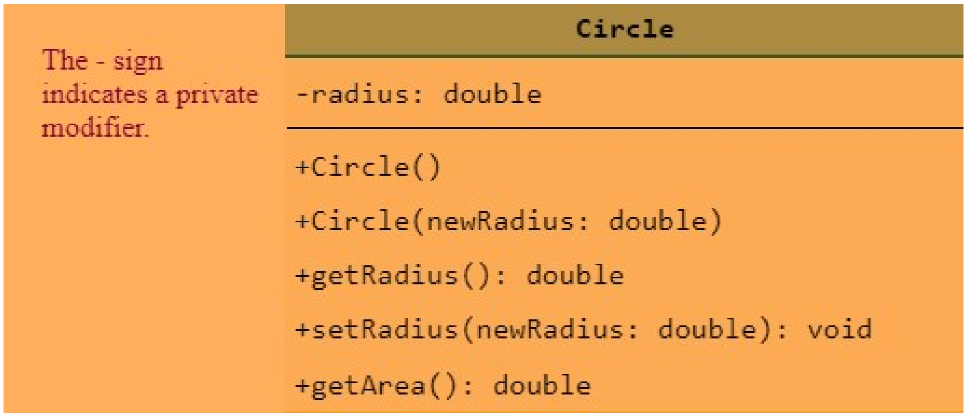
  A setter function has the following signature:

  ```
  void setPropertyName(dataType propertyValue)
  ```
-

- 

  ```
  class Circle
  {
  public:
    Circle();
    Circle(double);
    double getArea();

  private:
    double radius;
  };
  ```

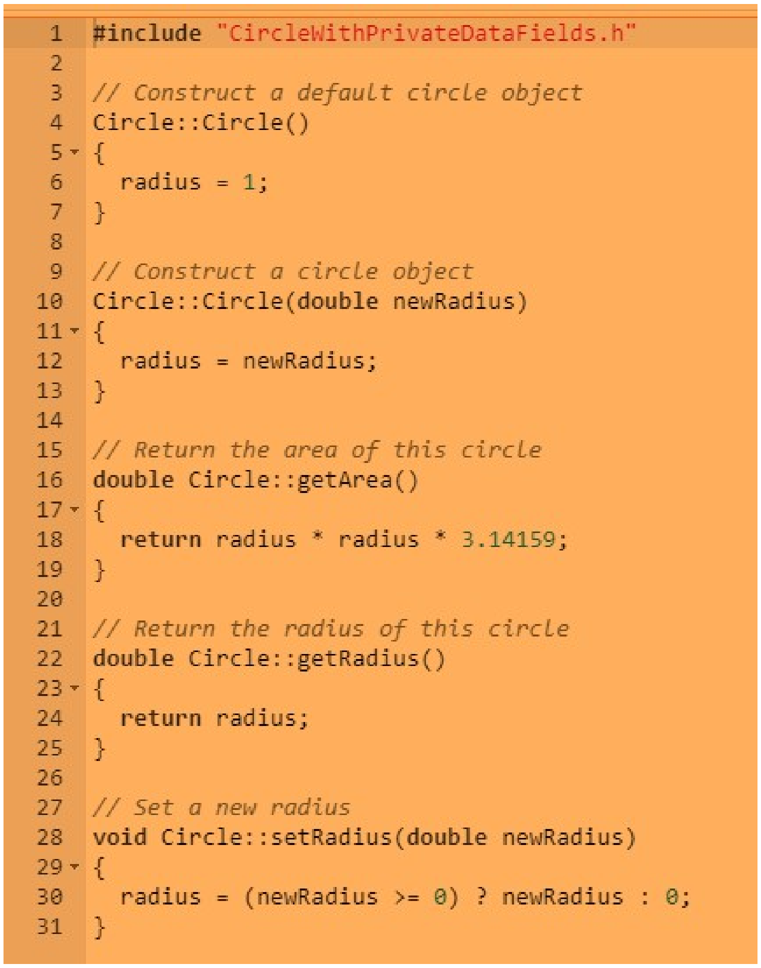
-

- 

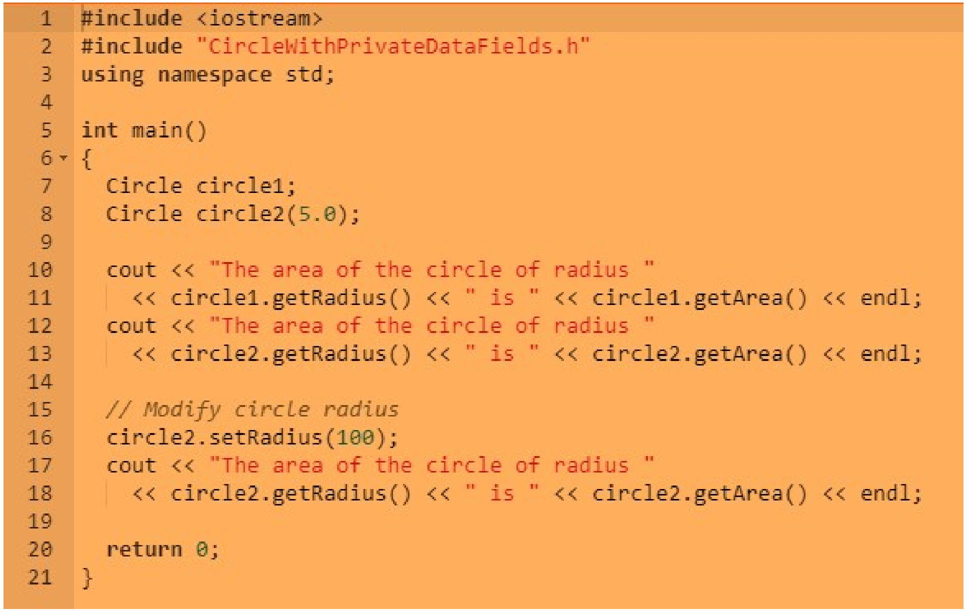
  The - sign indicates a private modifier.

  ```
  Circle
  -radius: double
  +Circle()
  +Circle(newRadius: double)
  +getRadius(): double
  +setRadius(newRadius: double): void
  +getArea(): double
  ```

-

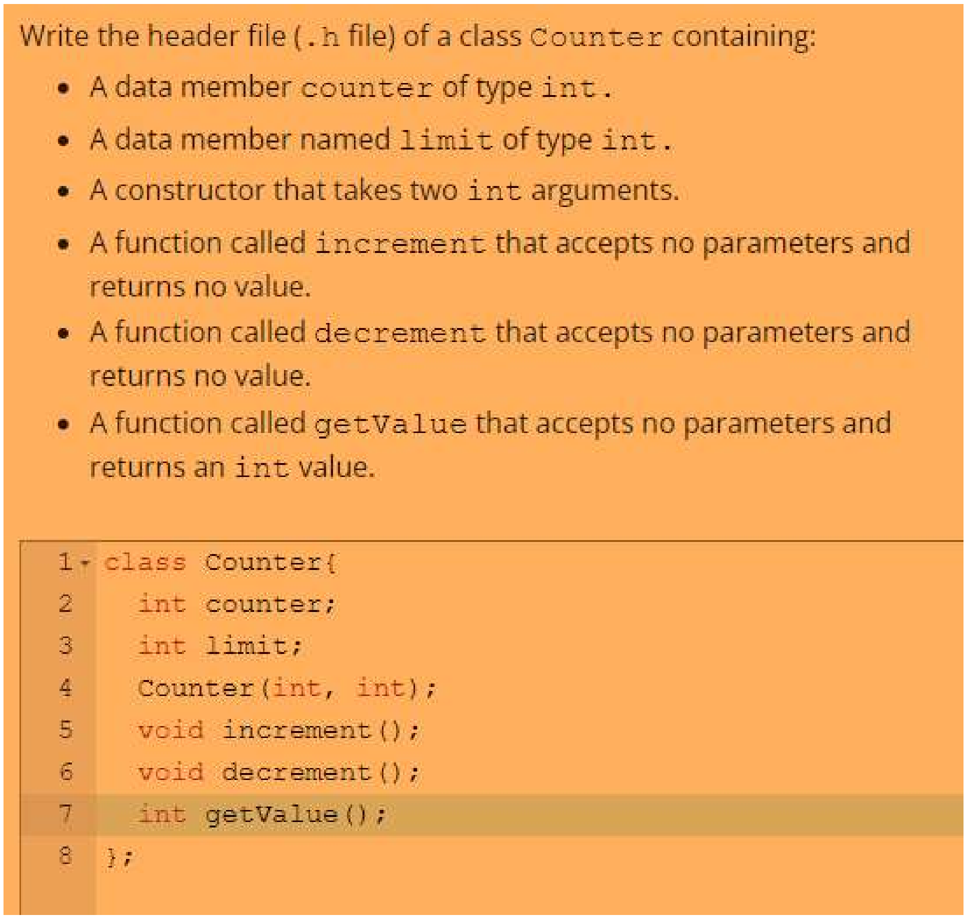
- 

```cpp
#include "CircleWithPrivateDataFields.h"

// Construct a default circle object
Circle::Circle()
{
  radius = 1;
}

// Construct a circle object
Circle::Circle(double newRadius)
{
  radius = newRadius;
}

// Return the area of this circle
double Circle::getArea()
{
  return radius * radius * 3.14159;
}

// Return the radius of this circle
double Circle::getRadius()
{
  return radius;
}

// Set a new radius
void Circle::setRadius(double newRadius)
{
  radius = (newRadius >= 0) ? newRadius : 0;
}
```

- Uses getRadius and setRadius
- 

```cpp
#include <iostream>
#include "CircleWithPrivateDataFields.h"
using namespace std;

int main()
{
  Circle circle1;
  Circle circle2(5.0);

  cout << "The area of the circle of radius "
    << circle1.getRadius() << " is " << circle1.getArea() << endl;
  cout << "The area of the circle of radius "
    << circle2.getRadius() << " is " << circle2.getArea() << endl;

  // Modify circle radius
  circle2.setRadius(100);
  cout << "The area of the circle of radius "
    << circle2.getRadius() << " is " << circle2.getArea() << endl;

  return 0;
}
```

-

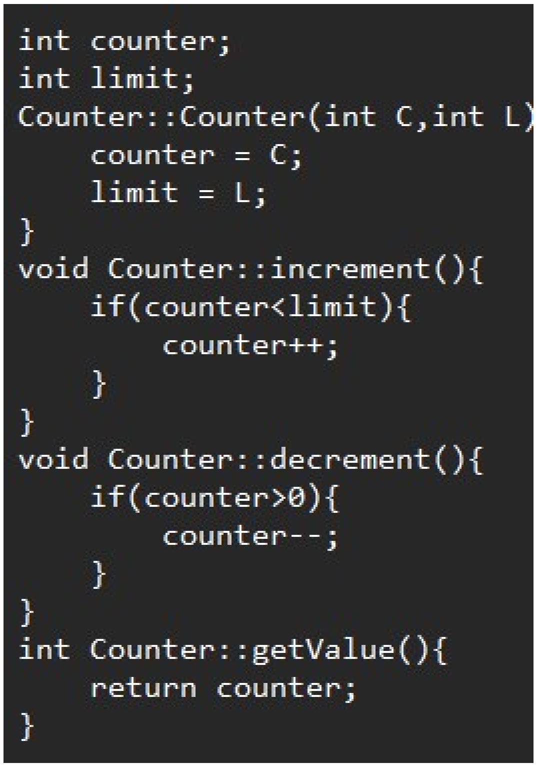
- 

  Write the header file (`.h` file) of a class `Counter` containing:

  - A data member `counter` of type `int`.
  - A data member named `limit` of type `int`.
  - A constructor that takes two `int` arguments.
  - A function called `increment` that accepts no parameters and returns no value.
  - A function called `decrement` that accepts no parameters and returns no value.
  - A function called `getValue` that accepts no parameters and returns an `int` value.

  ```
  class Counter{
    int counter;
    int limit;
    Counter(int, int);
    void increment();
    void decrement();
    int getValue();
  };
  ```

-

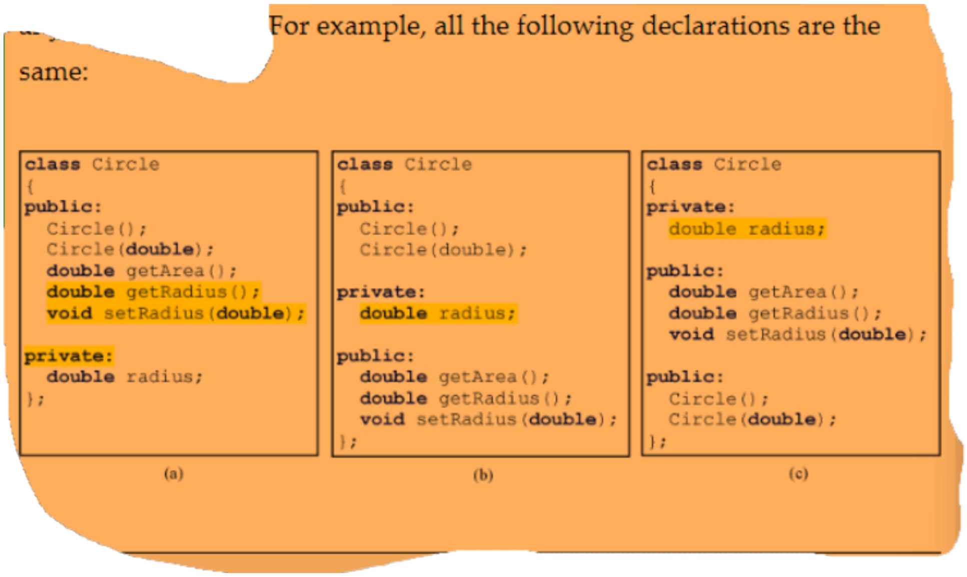
- 

  ```
  int counter;
  int limit;
  Counter::Counter(int C,int L)
      counter = C;
      limit = L;
  }
  void Counter::increment(){
      if(counter<limit){
          counter++;
      }
  }
  void Counter::decrement(){
      if(counter>0){
          counter--;
      }
  }
  int Counter::getValue(){
      return counter;
  }
  ```

-
-

9.10 Scope of Variables

Saturday, March 4, 2023 5:03 PM

- Scope of data fields is entire class, regardless of where data fields are declared
- Glbl vars are declared outside all functions, accessible to all functions in its scope
- Cope of glbl var start from its declaration, end @ end of pgrm
- Local vars are defined inside functions, scope starts from declaration & end @ block that has the var
- Static local vars are permanently stored in the pgrm, can be used in the next call of the function
- Data fields are declared as vars and are accessible to all constructors and functions in the class, data fields and functions can be in any order in a class
-
- 

For example, all the following declarations are the same:

```
class Circle
{
public:
  Circle();
  Circle(double);
  double getArea();
  double getRadius();
  void setRadius(double);

private:
  double radius;
};
```

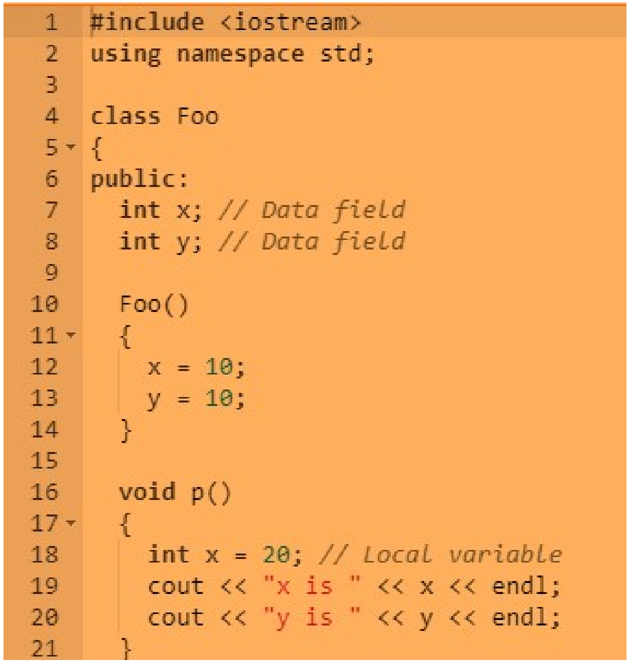
(a)

```
class Circle
{
public:
  Circle();
  Circle(double);

private:
  double radius;

public:
  double getArea();
  double getRadius();
  void setRadius(double);
};
```

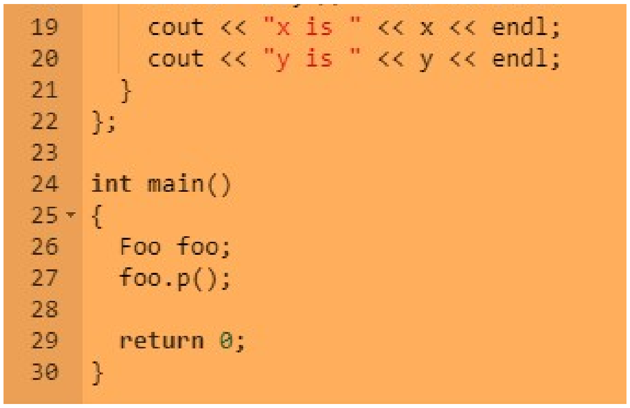
(b)

```
class Circle
{
private:
  double radius;

public:
  double getArea();
  double getRadius();
  void setRadius(double);

public:
  Circle();
  Circle(double);
};
```

(c)

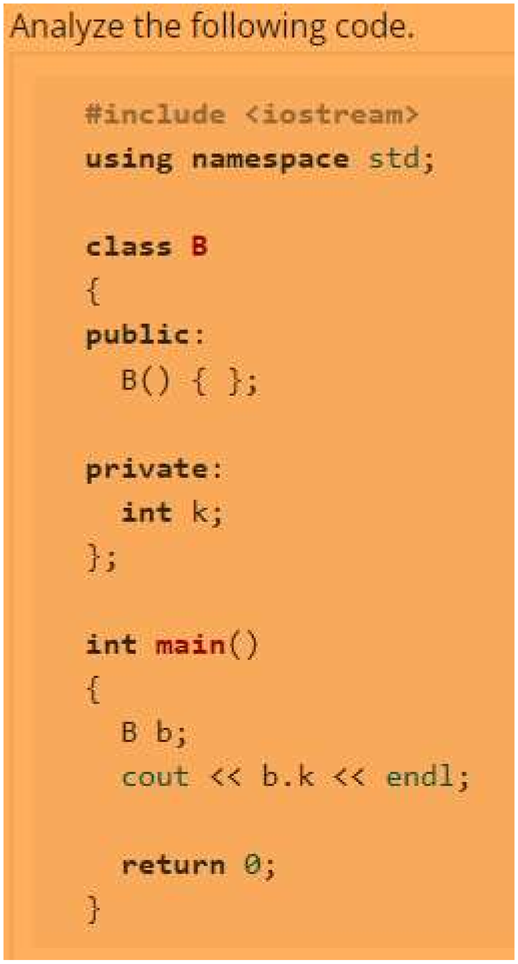
- Common style tho is keep public memb first, then private membs after
- Can declare a var for data field only once, but can declare same var name in a function many times in diff functions
- Local vars declared and used inside a function locally, if local var has same name as data field, local var takes precedence, and data field w/ same name is shadowed
- 

```
#include <iostream>
using namespace std;

class Foo
{
public:
  int x; // Data field
  int y; // Data field

  Foo()
  {
    x = 10;
    y = 10;
  }

  void p()
  {
    int x = 20; // Local variable
    cout << "x is " << x << endl;
    cout << "y is " << y << endl;
  }
```



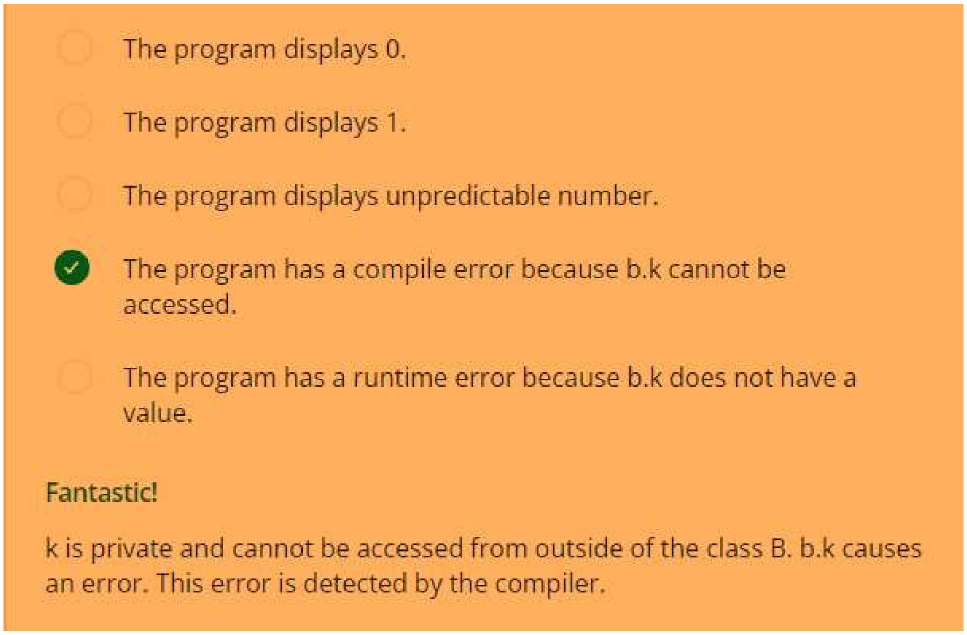
```
19        cout << "x is " << x << endl;
20        cout << "y is " << y << endl;
21      }
22    };
23
24    int main()
25    {
26      Foo foo;
27      foo.p();
28
29      return 0;
30    }
```

- Output say x = 20 and y = 10, this bc
    - X declared as a data field in Foo class, but also defined as local var in function p() w/ initial val of 20, latter x displayed to console in line 19 (using initial val)
    - Y declared as data field, so accessible inside function p()



Analyze the following code.

```
#include <iostream>
using namespace std;

class B
{
public:
  B() { };

private:
  int k;
};

int main()
{
  B b;
  cout << b.k << endl;

  return 0;
}
```



○ The program displays 0.

○ The program displays 1.

○ The program displays unpredictable number.

✓ The program has a compile error because b.k cannot be accessed.

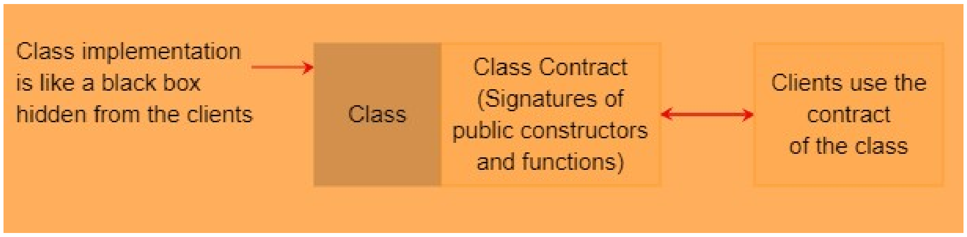
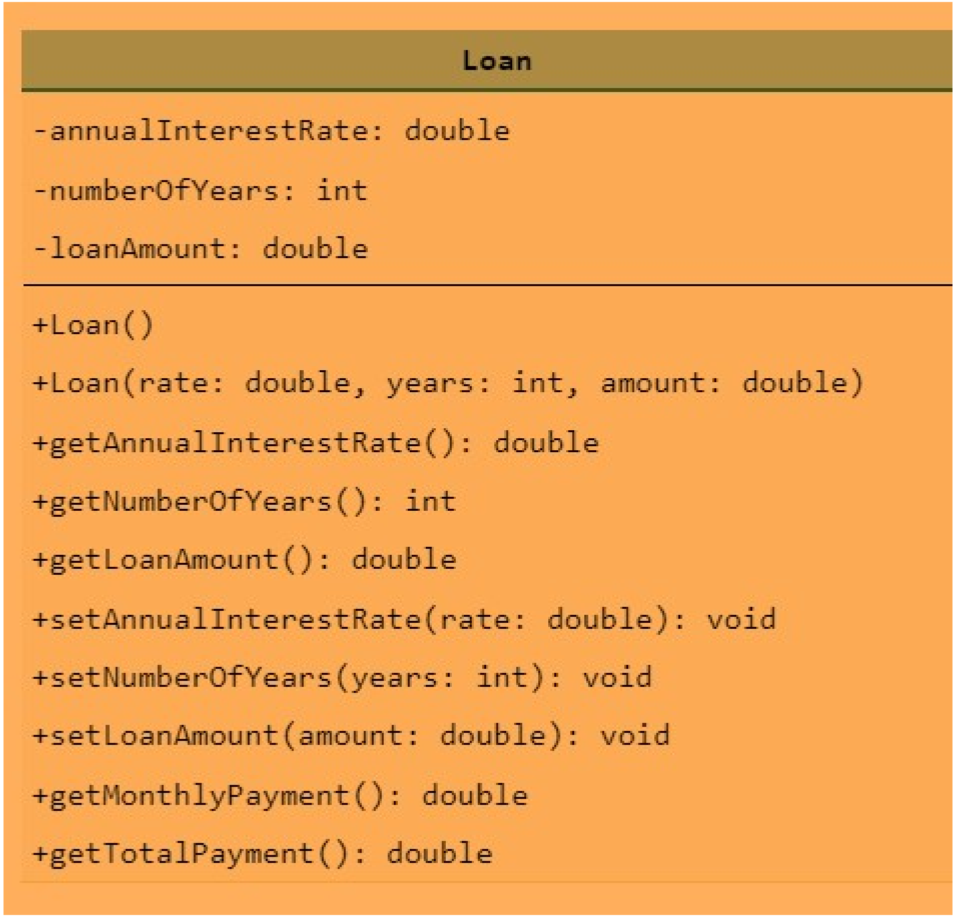
○ The program has a runtime error because b.k does not have a value.

Fantastic!

k is private and cannot be accessed from outside of the class B. b.k causes an error. This error is detected by the compiler.

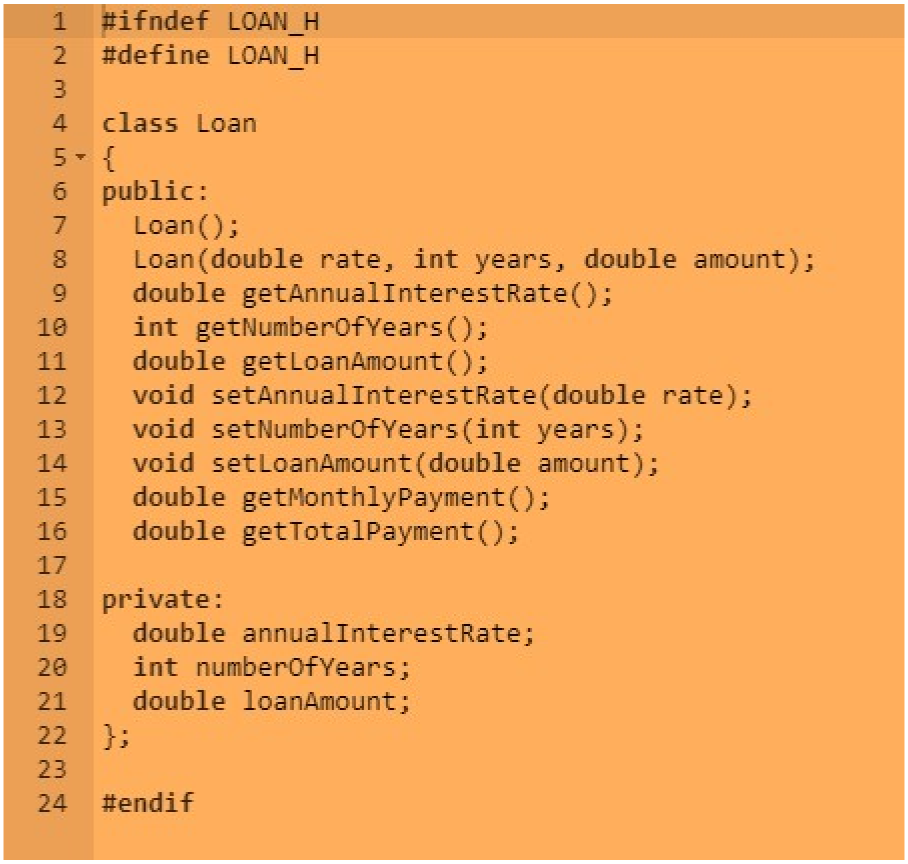
9.11 Class Abstraction and Encapsulation

Saturday, March 4, 2023 5:19 PM

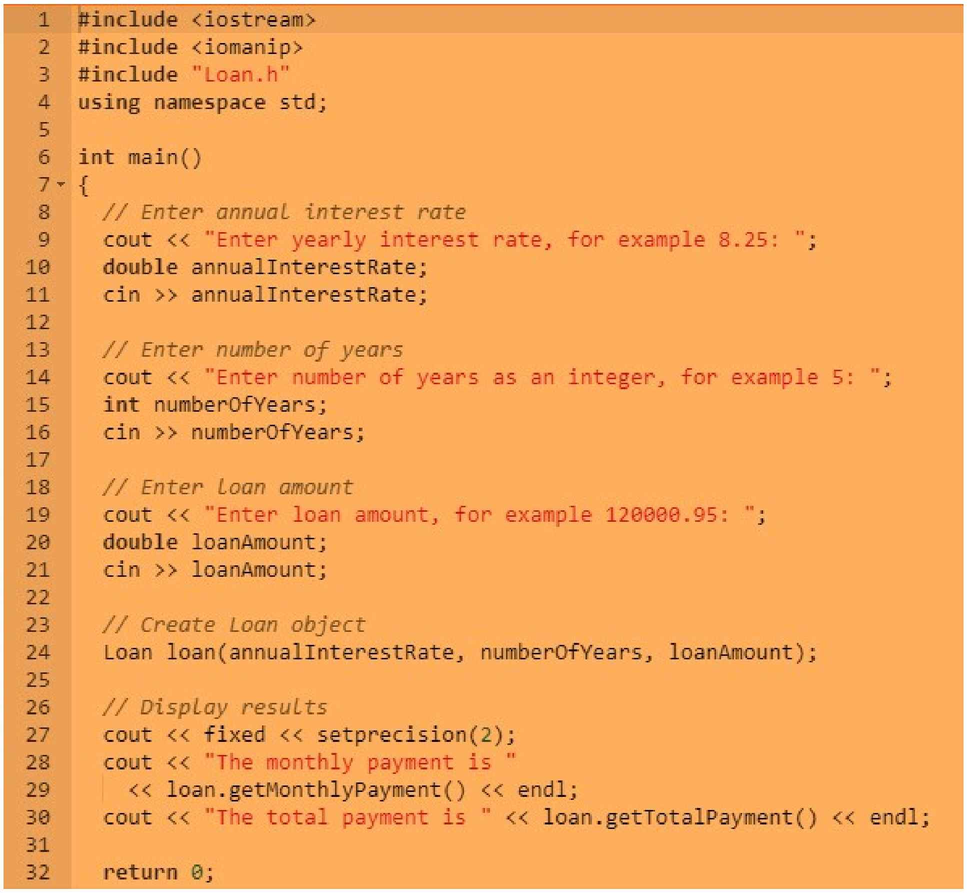
- Class abstraction is separation of class implementation from the use of a class, details of implementation are encapsulated and hidden from the user, aka class encapsulation
- Class abstraction is sep of class implementation from the use of a class
- Creator of a class gives a description of the class and lets the user know how it can be user, collection of constructors and functions that are accessible from outside the class (w/ description of how these members are expected to behave) serve as class's contract
- User of the class don't need know how class implemented, hidden from user
- Aka class encapsulation
-
- 
Class implementation is like a black box hidden from the clients
Class
Class Contract (Signatures of public constructors and functions)
Clients use the contract of the class
- Class abstraction and encapsulation 2 sides of same coin
- 

```
Loan
-annualInterestRate: double
-numberOfYears: int
-loanAmount: double
+Loan()
+Loan(rate: double, years: int, amount: double)
+getAnnualInterestRate(): double
+getNumberOfYears(): int
+getLoanAmount(): double
+setAnnualInterestRate(rate: double): void
+setNumberOfYears(years: int): void
+setLoanAmount(amount: double): void
+getMonthlyPayment(): double
+getTotalPayment(): double
```

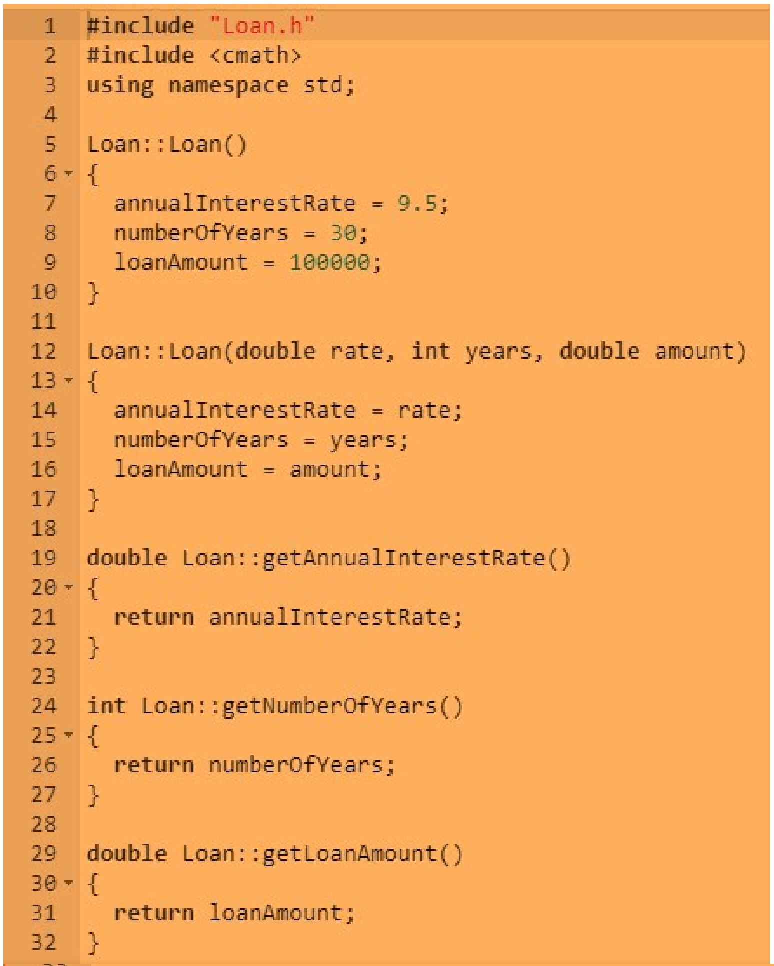
-



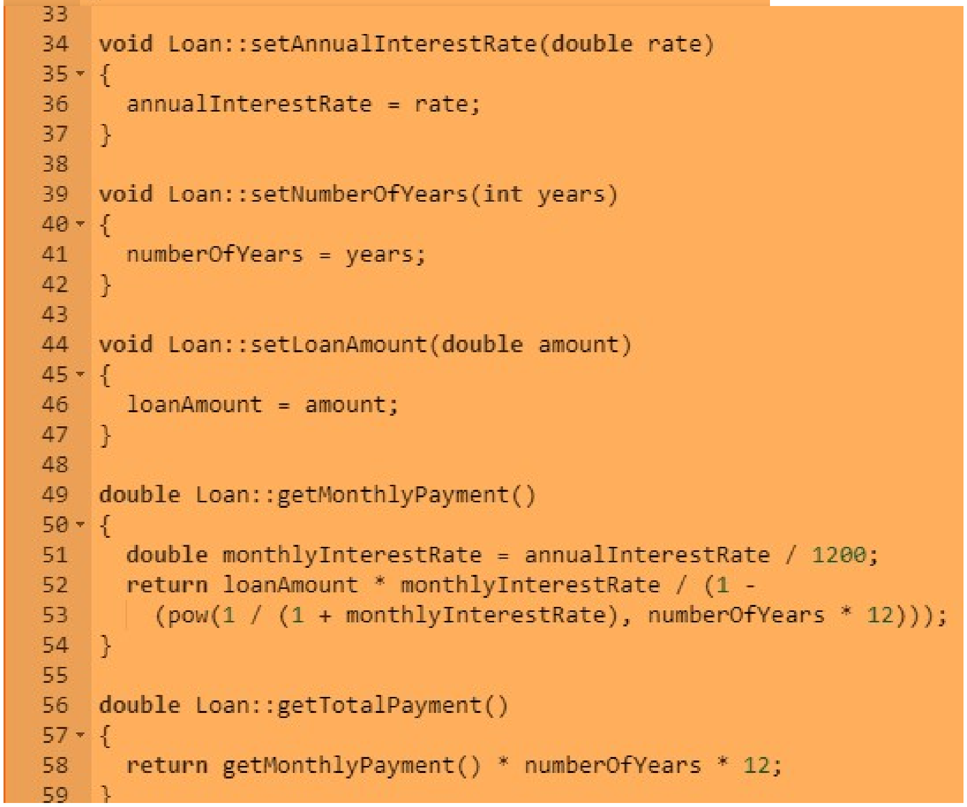
```cpp
#ifndef LOAN_H
#define LOAN_H

class Loan
{
public:
  Loan();
  Loan(double rate, int years, double amount);
  double getAnnualInterestRate();
  int getNumberOfYears();
  double getLoanAmount();
  void setAnnualInterestRate(double rate);
  void setNumberOfYears(int years);
  void setLoanAmount(double amount);
  double getMonthlyPayment();
  double getTotalPayment();

private:
  double annualInterestRate;
  int numberOfYears;
  double loanAmount;
};

#endif
```



```cpp
#include <iostream>
#include <iomanip>
#include "Loan.h"
using namespace std;

int main()
{
  // Enter annual interest rate
  cout << "Enter yearly interest rate, for example 8.25: ";
  double annualInterestRate;
  cin >> annualInterestRate;

  // Enter number of years
  cout << "Enter number of years as an integer, for example 5: ";
  int numberOfYears;
  cin >> numberOfYears;

  // Enter loan amount
  cout << "Enter loan amount, for example 120000.95: ";
  double loanAmount;
  cin >> loanAmount;

  // Create Loan object
  Loan loan(annualInterestRate, numberOfYears, loanAmount);

  // Display results
  cout << fixed << setprecision(2);
  cout << "The monthly payment is "
    << loan.getMonthlyPayment() << endl;
  cout << "The total payment is " << loan.getTotalPayment() << endl;

  return 0;
```

- 

```cpp
#include "Loan.h"
#include <cmath>
using namespace std;

Loan::Loan()
{
  annualInterestRate = 9.5;
  numberOfYears = 30;
  loanAmount = 100000;
}

Loan::Loan(double rate, int years, double amount)
{
  annualInterestRate = rate;
  numberOfYears = years;
  loanAmount = amount;
}

double Loan::getAnnualInterestRate()
{
  return annualInterestRate;
}

int Loan::getNumberOfYears()
{
  return numberOfYears;
}

double Loan::getLoanAmount()
{
  return loanAmount;
}
```

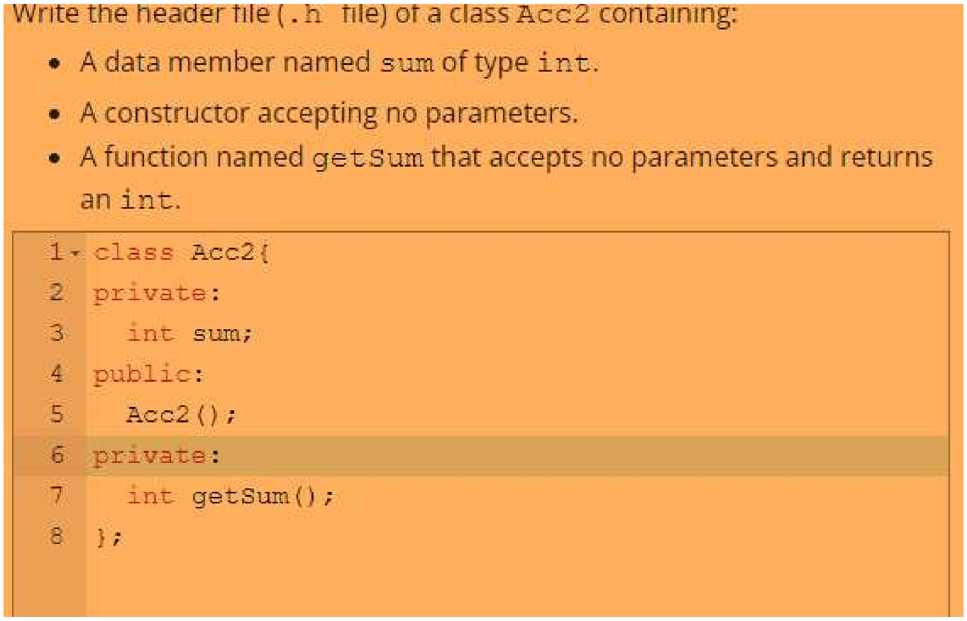
- 

```cpp

void Loan::setAnnualInterestRate(double rate)
{
  annualInterestRate = rate;
}

void Loan::setNumberOfYears(int years)
{
  numberOfYears = years;
}

void Loan::setLoanAmount(double amount)
{
  loanAmount = amount;
}

double Loan::getMonthlyPayment()
{
  double monthlyInterestRate = annualInterestRate / 1200;
  return loanAmount * monthlyInterestRate / (1 -
    (pow(1 / (1 + monthlyInterestRate), numberOfYears * 12)));
}

double Loan::getTotalPayment()
{
  return getMonthlyPayment() * numberOfYears * 12;
}
```

- In UML diagram for Loan class, start writing test pgrm that uses Loan class even though don't know how Loan class implemented, benefits:
  - ○ Demonstrated developing a class and using a class are 2 sep tasks
  - ○ Enables you to skip complex implementation of certain classes w/out interrupting the

sequence of the book

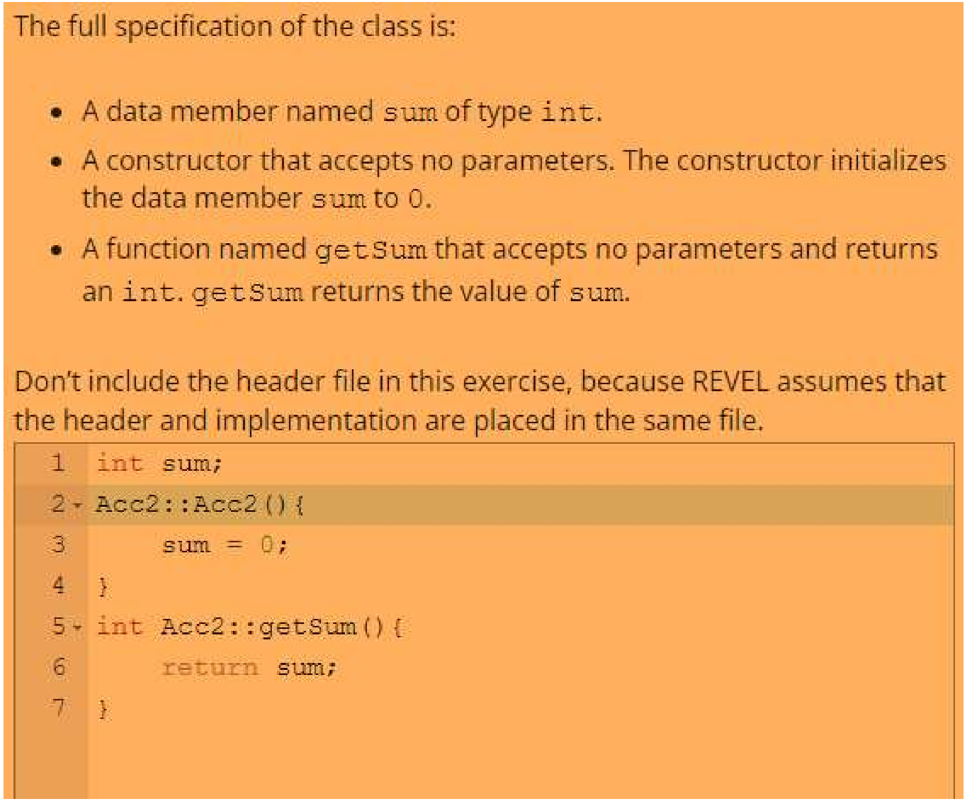
  - Easier to learn how to implement a class if ur familiar w/ class thru using it

- 

Write the header file (.h file) of a class Acc2 containing:

- A data member named sum of type int.
- A constructor accepting no parameters.
- A function named getSum that accepts no parameters and returns an int.

```
class Acc2{
private:
  int sum;
public:
  Acc2();
private:
  int getSum();
};
```

- Idk why I put rando public and private, but I did and I don’t care
- 

The full specification of the class is:

- A data member named sum of type int.
- A constructor that accepts no parameters. The constructor initializes the data member sum to 0.
- A function named getSum that accepts no parameters and returns an int. getSum returns the value of sum.

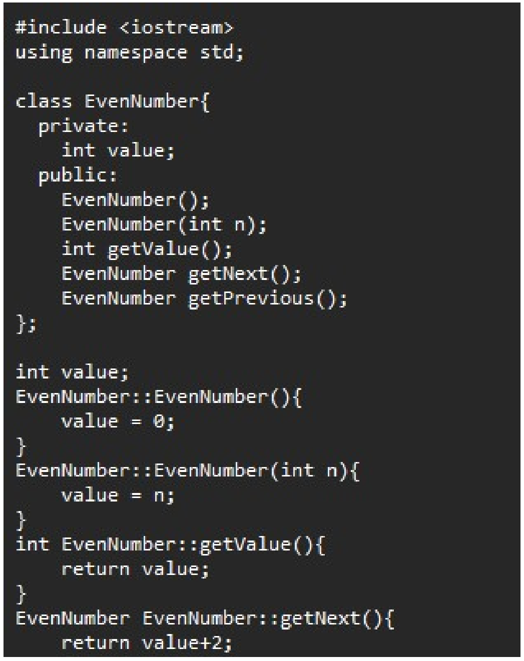
Don’t include the header file in this exercise, because REVEL assumes that the header and implementation are placed in the same file.

```
int sum;
Acc2::Acc2(){
     sum = 0;
}
int Acc2::getSum(){
     return sum;
}
```

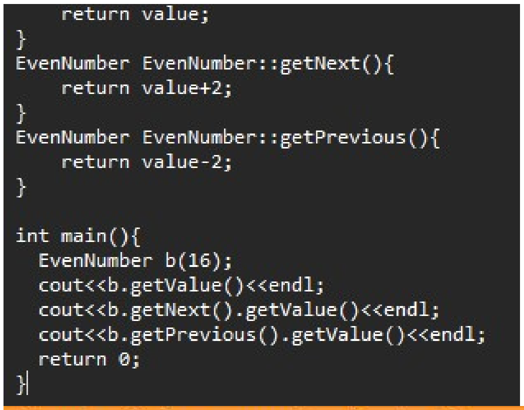
Ch 9 Summary

Saturday, March 4, 2023 5:44 PM

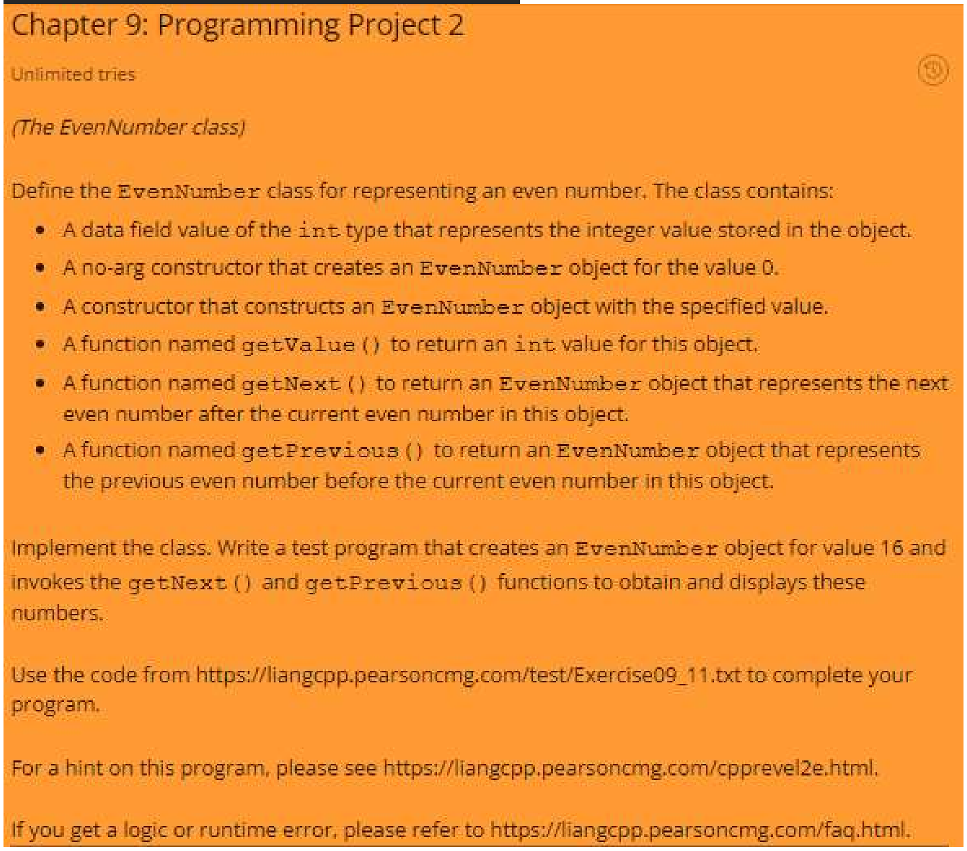
1. A class is a blueprint for objects.
2. A class defines the data fields for storing the properties of objects and provides constructors for creating objects and functions for manipulating them.
3. Constructors must have the same name as the class itself.
4. A non-arg constructor is a constructor that does not have arguments.
5. A class is also a data type. You can use it to declare and create objects.
6. An object is an instance of a class. You use the dot (.) operator to access members of that object through its name.
7. The state of an object is represented by data fields (also known as properties) with their current values.
8. The behavior of an object is defined by a set of functions.
9. The data fields do not have initial values. They must be initialized in constructors.
10. You can separate class definition from class implementation by defining class in a header file and class implementation in a separate file.
11. The C++ #ifndef directive, called inclusion guard, can be used to prevent a header file from being included multiple times.
12. When a function is implemented inside a class definition, it automatically becomes an inline function.
13. Visibility keywords specify how the class, function, and data are accessed.
14. A public function or data is accessible to all clients.
15. A private function or data is accessible only inside the class.
16. You can provide a getter function or a setter function to enable clients to see or modify the data.
17. Colloquially, a getter function is referred to as a getter (or accessor), and a setter function is referred to as a setter (or mutator).
18. A getter function has the signature
    returnType getPropertyName()
19. If the returnType is bool, the getter function should be defined as
    bool isPropertyName().
20. A setter function has the signature
    void setPropertyName(dataType propertyValue)



```
#include <iostream>
using namespace std;

class EvenNumber{
  private:
    int value;
  public:
    EvenNumber();
    EvenNumber(int n);
    int getValue();
    EvenNumber getNext();
    EvenNumber getPrevious();
};

int value;
EvenNumber::EvenNumber(){
    value = 0;
}
EvenNumber::EvenNumber(int n){
    value = n;
}
int EvenNumber::getValue(){
    return value;
}
EvenNumber EvenNumber::getNext(){
    return value+2;
```



```
    return value;
}
EvenNumber EvenNumber::getNext(){
    return value+2;
}
EvenNumber EvenNumber::getPrevious(){
    return value-2;
}

int main(){
  EvenNumber b(16);
  cout<<b.getValue()<<endl;
  cout<<b.getNext().getValue()<<endl;
  cout<<b.getPrevious().getValue()<<endl;
  return 0;
}
```



Chapter 9: Programming Project 2

Unlimited tries

(The EvenNumber class)

Define the `EvenNumber` class for representing an even number. The class contains:

- A data field value of the `int` type that represents the integer value stored in the object.
- A no-arg constructor that creates an `EvenNumber` object for the value 0.
- A constructor that constructs an `EvenNumber` object with the specified value.
- A function named `getValue()` to return an `int` value for this object.
- A function named `getNext()` to return an `EvenNumber` object that represents the next even number after the current even number in this object.
- A function named `getPrevious()` to return an `EvenNumber` object that represents the previous even number before the current even number in this object.

Implement the class. Write a test program that creates an `EvenNumber` object for value 16 and invokes the `getNext()` and `getPrevious()` functions to obtain and displays these numbers.

Use the code from https://liangcpp.pearsoncmg.com/test/Exercise09_11.txt to complete your program.

For a hint on this program, please see https://liangcpp.pearsoncmg.com/cpprevel2e.html.

If you get a logic or runtime error, please refer to https://liangcpp.pearsoncmg.com/faq.html.