Lab 9

Rodrigo Becerril Ferreyra and Ethan Hua

Source Files

`Employee.h`

h Employee.h ×

```h
/**
 * Employee class header file.
 * CECS 275 - Fall 2021
 * @author Ethan Hua
 * @author Rodrigo Becerril Ferreyra
 * @version 1.21
 */

#ifndef EMPLOYEE_H
#define EMPLOYEE_H

#include <string> // Used for storing employee's name and date hired.

/**
 * "Employee" class stores data about a given workplace's employee as an
 * object, intended to be used as a base class for further specific
 * workplace needs.
*/
class Employee
{
    private:
        std::string employeeName;
        int employeeNumber;
        std::string hireDate; // NOTE: Professor recommended use of string.
        void setDefaults();
    public:
        // Constructors
        Employee();
        Employee(std::string);
```

```
30          Employee(std::string,int);
31          Employee(std::string,int,std::string);
32
33          // Mutators
34          void setName(std::string);
35          void setNumber(int);
36          void setDate(std::string);
37
38          // Accessors
39          /**
40           * Accessor member function used to obtain employee's name.
41           * @return Target employee name.
42          */
43          std::string getName() const
44              { return employeeName; }
45          /**
46           * Accessor member function used to obtain employee's numerical ID.
47           * @return Target employee identification number.
48          */
49          int getNumber() const
50              { return employeeNumber; }
51          /**
52           * Accessor member function used to obtain date of employment.
53           * @return Target employee's date they were initially hired.
54          */
55          std::string getDate() const
56              { return hireDate; }
57  };
58
59  #endif
60
```

Employee.cpp

Employee.cpp

```cpp
/**
 * Employee class implementation file.
 * CECS 275 - Fall 2021
 * @author Ethan Hua
 * @author Rodrigo Becerril Ferreyra
 * @version 1
 */

#include "Employee.h"

/**
 * Default class constructor for a no-parameter object creation.
 * Loads pre-defined default values into member variables.
 * @return "Employee" object with generic name, ID, and [current-year] date
*/
Employee::Employee(){
    setDefaults();
}

/**
 * Class constructor for a named employee with no other information.
 * Loads the new name into the corresponding field.
 * @return "Employee" object with custom name and generic ID/hire date.
*/
Employee::Employee(std::string newEmp){
    setDefaults();
    employeeName = newEmp;
}

```

```
30  /**
31   * Class constructor for a named employee with a generated unique ID.
32   * Loads the new employee and their ID into the corresponding fields.
33   * @return "Employee" object with custom name, their ID, and generic date.
34  */
35  Employee::Employee(std::string newEmp, int newID){
36      setDefaults();
37      employeeName = newEmp;
38      setNumber(newID);
39  }
40
41  /**
42   * Class constructor for a known employee, their ID, and when they were hired.
43   * Completely overwrites generic defaults with provided information.
44   * @return "Employee" object containing given name, ID, and hire date.
45  */
46  Employee::Employee(std::string newEmp, int newID, std::string newDate){
47      setDefaults();
48      employeeName = newEmp;
49      setNumber(newID);
50      hireDate = newDate;
51  }
52
53  /**
54   * Mutator member function that overwrites the object's current name
55   * data with a provided string.
56   * @param newEmp Text string containing replacement name
57  */
```

```cpp
void Employee::setName(std::string newEmp){
    employeeName = newEmp;
}

/**
 * Mutator member function that overwrites the object's current ID
 * number with the provided int value, provided it is non-negative.
 * @param newID Replacement ID for overwriting
*/
void Employee::setNumber(int newID){
    if (newID > 0){
        employeeNumber = newID;
    }
}

/**
 * Mutator member function that overwrites the object's currently held
 * date string with a new string intended to represent the date the
 * employee was hired.
 * @param newDate Replacement text string containing hiring date
*/
void Employee::setDate(std::string newDate){
    hireDate = newDate;
}

/**
 * Private function used by constructors to quickly default to given
 * arbitrary values before user's new information is known.
*/
void Employee::setDefaults(){
    employeeName = "John Doe";
    employeeNumber = 123456789;
    hireDate = "January 1, 2021";
}

```

ProductionWorker.h

h ProductionWorker.h

```h
/**
 * Production Worker class header file.
 * CECS 275 - Fall 2021
 * @author Ethan Hua
 * @author Rodrigo Becerril Ferreyra
 * @version 1
 */

#ifndef PRODUCTIONWORKER_H
#define PRODUCTIONWORKER_H

#include <string>
#include "Employee.h"

/**
 * "Production Worker" class acts as an extension of the "Employee"
 * base class. Associates the employee's identifying characteristics
 * with the shift they work (assuming a binary shift schedule) and
 * their wages earned.
*/
class ProductionWorker : public Employee
{
    private:
        int shift;
        double hourWages;
        /*
        NOTE: Chapter 15 Programming Challenges 1 notes that a "shift" of
        1 should correspond to a Day shift and that a "shift" of 2 should
        correspond to a Night shift. This array is used to implement this
        relation by way of indexing.
        */
```

```
32          const std::string shiftHalves[3] = {"None","Day","Night"};
33          void minWage();
34      public:
35          // Constructors
36          /**
37           * Default class constructor for a no-parameter object creation.
38           * Loads pre-defined default values into member variables.
39           * @return Worker with a presumed minimum wage and day shift.
40          */
41          ProductionWorker() : Employee()
42              { minWage(); }
43          ProductionWorker(std::string,int,std::string,int,double);
44
45          // Mutators
46          void setShift(int);
47          void setWages(double);
48
49          // Accessors
50          /**
51           * Accessor member function used to obtain worker's hourly wage.
52           * @return Worker's payments; unformatted raw double.
53          */
54          double getWages() const
55              { return hourWages; }
```

```
56          /**
57           * Accessor member function used to obtain worker's shift as a string.
58           * @return Worker's job shift written out.
59          */
60          std::string getShiftFormat() const
61              { return shiftHalves[shift]; }
62          /**
63           * Accessor member function used to obtain worker's shift as an int.
64           * @return Worker's job shift as a raw int value.
65          */
66          int getShiftNum() const
67              { return shift; }
68  };
69
70  #endif
71
```

ProductionWorker.cpp

ProductionWorker.cpp

```cpp
/**
 * Production Worker class implementation file.
 * CECS 275 - Fall 2021
 * @author Ethan Hua
 * @author Rodrigo Becerril Ferreyra
 * @version 1
 */

#include "ProductionWorker.h"
#include "Employee.h"
#include <string>

/**
 * Class constructor for a known employee (including ID and hire date),
 * their hourly earnings, and their assigned shift.
 * @return "Production Worker" that works at X time that makes Y amount.
*/
ProductionWorker::ProductionWorker(std::string empName, int empID,
                                   std::string hireDate, int empShift,
                                   double empWage)
                : Employee(empName, empID, hireDate){
    minWage();
    setShift(empShift);
    setWages(empWage);
}

/**
 * Mutator member function that attempts to assign the employee with their
 * corresponding shift timing if given a valid shift; otherwise maintains
 * their current shift (or default).
 * @param newShift Integer value of 1 or 2 representing worker's shift.
*/
void ProductionWorker::setShift(int newShift){
    // Verify that the employee's shift is one of two valid values.
    if ( (newShift == 1) || (newShift == 2) ) {
        shift = newShift;
    } else {
        shift = shift;
    }
}

/**
 * Mutator member function that attempts to set the employee to their
 * appropriate hourly compensation assuming that they are being paid
 * a positive amount of money; otherwise maintains their current
 * wages (or default).
 * @param newWage Worker's hourly payment rate.
```

```
48  */
49  void ProductionWorker::setWages(double newWage){
50      // Verify that the employee is actually being paid.
51      if (newWage > 0) {
52          hourWages = newWage;
53      } else {
54          hourWages = hourWages;
55      }
56  }
57
58  /**
59   * Private function used by constructors to quickly default to given
60   * arbitrary values before user's new information is known.
61   */
62  void ProductionWorker::minWage(){
63      // Default assumption is a day shift.
64      shift = 1;
65      // Default assumption is a payment of USD$15.50/hour.
66      hourWages = 15.50;
67  }
68
```

WorkerTest.cpp

WorkerTest.cpp

```cpp
/**
 * Program to generate employee profiles.
 * CECS 275 - Fall 2021
 * @author Ethan Hua
 * @author Rodrigo Becerril Ferreyra
 * @version 1.3
 */

#include <iostream>           // Used for terminal I/O
#include <iomanip>            // Used to format payment output to terminal
#include <string>             // Used for string object creation
#include "Employee.h"         // Base class; employee identification profiles
#include "ProductionWorker.h" // Derived class from "Employee"; work profiles

using namespace std;

// Function Prototypes
void massPrint(const Employee&);
void massPrint(const ProductionWorker&);

int main(){
    // Context
    cout << "Beginning tests. \n";
    cout << "======================================== \n";
    // Employee-class constructor; no parameters (default)
    cout << "Default Employee Constructor: \n";
    Employee defaultTest;
    massPrint(defaultTest);
```

```
29
30      // Employee-class constructor; all parameters
31      cout << "Populated Employee Constructor: \n";
32      Employee fullTest("Johnny Test",605081427,"October 21, 2021");
33      massPrint(fullTest);
34
35      // ProductionWorker-class constructor; no parameters (default)
36      cout << "Default Production Worker Constructor: \n";
37      ProductionWorker defaultWorker;
38      massPrint(defaultWorker);
39
40      // ProductionWorker-class constructor; all parameters (incl. base class)
41      // Note the order flows from base class to derived class.
42      cout << "Populated Production Worker Constructor: \n";
43      ProductionWorker defWork("Big Bob",963485712,"October 22, 2021",2,16.75);
44      massPrint(defWork);
45
46      // ProductionWorker-class constructor; all parameters (incl. base class)
47      // Deliberate use of a known, invalid wage to show default override
48      cout << "Production Worker Test (Negative Wage): \n";
49      ProductionWorker noMons("Al Portland",275720172,
50                              "September 21, 2021",2,-24.84);
51      massPrint(noMons);
52
```

```
53      // ProductionWorker-class constructor; all parameters (incl. base class)
54      // Deliberate use of a known, invalid shift to show default override
55      cout << "Production Worker Test (Invalid Shift): \n";
56      ProductionWorker timeBreak("Thieu-Thanh Do",883368649,
57                                 "July 29, 2021",982,31.84);
58      massPrint(timeBreak);
59
60      cout << "Terminating... \n";
61      return 0;
62  }
63
64  /**
65   * Overloaded function variant used to output formatted and distinct block
66   * of all variables inherent to an Employee. Constant pass by address
67   * used as the program is only reading from object.
68   * @param target  Employee-class object to be read out.
69  */
70  void massPrint(const Employee &target){
71      // Access all private variables
72      string targetName = target.getName();
73      int targetNumber = target.getNumber();
74      string targetDate = target.getDate();
75      // Terminal output
76      cout << "======================================== \n";
77      cout << "Employee Name : " << targetName << "\n";
78      cout << "Employee ID   : " << targetNumber << "\n";
79      cout << "Employee Hired: " << targetDate << "\n";
80      cout << "======================================== \n";
81  }
```

```
82
83  /**
84   * Overloaded function variant used to output formatted and distinct block
85   * of all variables inherent to a Production Worker. Constant pass by address
86   * used as the program is only reading from object.
87   * @param target  ProductionWorker-class object to be read out.
88  */
89  void massPrint(const ProductionWorker &target){
90      // Access all private variables + inherited variables
91      string targetName = target.getName();
92      int targetNumber = target.getNumber();
93      string targetDate = target.getDate();
94      string targetShiftType = target.getShiftFormat();
95      int targetShiftNum = target.getShiftNum();
96      double targetPay = target.getWages();
97      // Terminal output
98      cout << "======================================== \n";
99      cout << "Employee Name : " << targetName << "\n";
100     cout << "Employee ID   : " << targetNumber << "\n";
101     cout << "Employee Hired: " << targetDate << "\n";
102     // Provide both name/string of shift type & corresponding value
103     cout << "Employee Shift: " << targetShiftType << "/"
104          << targetShiftNum << "\n";
105     // getWages() does not automatically format value
106     cout << "Employee's Pay: USD$" << fixed << setprecision(2)
107          << targetPay << "\n";
108     cout << "======================================== \n";
109 }
110
```

Output of Worker test:

```
Beginning tests.
=======================================
Default Employee Constructor:
=======================================
Employee Name : John Doe
Employee ID   : 123456789
Employee Hired: January 1, 2021
=======================================
Populated Employee Constructor:
=======================================
Employee Name : Johnny Test
Employee ID   : 605081427
Employee Hired: October 21, 2021
=======================================
Default Production Worker Constructor:
=======================================
Employee Name : John Doe
Employee ID   : 123456789
Employee Hired: January 1, 2021
Employee Shift: Day/1
Employee's Pay: USD$15.50
=======================================
Populated Production Worker Constructor:
```

```
=======================================
Employee Name : Big Bob
Employee ID   : 963485712
Employee Hired: October 22, 2021
Employee Shift: Night/2
Employee's Pay: USD$16.75
=======================================
Production Worker Test (Negative Wage):
=======================================
Employee Name : Al Portland
Employee ID   : 275720172
Employee Hired: September 21, 2021
Employee Shift: Night/2
Employee's Pay: USD$15.50
=======================================
Production Worker Test (Invalid Shift):
=======================================
Employee Name : Thieu-Thanh Do
Employee ID   : 883368649
Employee Hired: July 29, 2021
Employee Shift: Day/1
Employee's Pay: USD$31.84
=======================================
Terminating...
```

BasicShape.h

h BasicShape.h

```
/**
 * BasicShape class header file.
 * CECS 275 - Fall 2021
 * @author Rodrigo Becerril Ferreyra
 * @author Ethan Hua
 * @version 1
 */

#ifndef BASICSHAPE_H
#define BASICSHAPE_H

/**
 * The BasicShape class provides an abstract base class for other types of
 * shapes. Note that this class cannot be instantiated due to the pure
 * virtual function included in this class.
 */
class BasicShape
{
    private:
        double area;

    public:
        /**
         * Getter function.
         * @return The area of the shape.
         */
        double getArea() const {return area;}

        /**
         * Setter function.
         */
        void setArea(double area) {this->area = area;}

        /**
         * Pure virtual function that calculates the area of a shape and stores
         * it in the area member.
         */
        virtual void calcArea() = 0;
};

#endif // BASICSHAPE_H

```

Circle.h

h Circle.h

```h
/**
 * Circle class header file.
 * CECS 275 - Fall 2021
 * @author Rodrigo Becerril Ferreyra
 * @author Ethan Hua
 * @version 1
 */

#ifndef CIRCLE_H
#define CIRCLE_H

#include "BasicShape.h"

/**
 * The Circle class extends the BasicShape class.
 */
class Circle : public BasicShape
{
    private:
        long centerX;
        long centerY;
        double radius;

    public:
        /**
         * Initializes the object's private members and calculates the area.
         * @param centerX The x-coordinate of the center of the circle.
         * @param centerY The y-coordinate of the center of the circle.
         * @param radius The radius of the circle.
         */
        Circle(long centerX, long centerY, double radius);

        // center getter methods
        long getCenterX() const {return centerX;}
        long getCenterY() const {return centerY;}

        /**
         * Calculates the area of the circle and stores it in the inherited
         * area member.
         */
        void calcArea();
};

#endif // CIRCLE_H

```

Circle.cpp

Circle.cpp

```cpp
/**
 * Circle class implementation file.
 * CECS 275 - Fall 2021
 * @author Rodrigo Becerril Ferreyra
 * @author Ethan Hua
 * @version 1
 */

#include "Circle.h"
#define PI 3.14159

Circle::Circle(long centerX, long centerY, double radius)
{
    this->centerX = centerX;
    this->centerY = centerY;
    this->radius = radius;
    calcArea();
}

void Circle::calcArea()
{
    setArea(PI * radius * radius);
}

```

Rectangle.h

h Rectangle.h ×

```h
/**
 * Rectangle class header file.
 * CECS 275 - Fall 2021
 * @author Rodrigo Becerril Ferreyra
 * @author Ethan Hua
 * @version 1
 */

#ifndef RECTANGLE_H
#define RECTANGLE_H

#include "BasicShape.h"

/**
 * Rectangle class extends Basic shape.
 */
class Rectangle : public BasicShape
{
    private:
        long width;
        long length;
    public:
        /**
         * Initializes width and length of rectangle and calculates the area.
         * @param length The length of the rectangle.
         * @param width The width of the rectangle.
         */
        Rectangle(long length, long width);

        // getter functions
        long getWidth() const {return width;}
        long getLength() const {return length;}

        /**
         * Calculates the area of the rectangle and stores it in the inherited
         * area member.
         */
        void calcArea();
};

#endif//RECTANGLE_H

```

Rectangle.cpp

Rectangle.cpp

```cpp
/**
 * Rectangle implementation header file.
 * CECS 275 - Fall 2021
 * @author Rodrigo Becerril Ferreyra
 * @author Ethan Hua
 * @version 1
 */

#include "Rectangle.h"

Rectangle::Rectangle(long length, long width)
{
    this->length = length;
    this->width = width;
    calcArea();
}

void Rectangle::calcArea()
{
    setArea(length * width);
}
```

ShapesDriver.cpp

ShapesDriver.cpp

```cpp
/**
 * This program tests both Rectangle and Circle classes.
 * CECS 275 - Fall 2021
 * @author Rodrigo Becerril Ferreyra
 * @author Ethan Hua
 * @version 1
 */

#include <iostream>
#include "Circle.h"
#include "Rectangle.h"

int main()
{
    std::cout << "Testing Circle class...\n";
    // create two Circle objects
    Circle circle1(0.0, 0.0, 5.0), circle2(1.0, 1.0, 10.0);

    // get the center coordinates of both circles
    std::cout << "The center of circle1 is located at ("
      << circle1.getCenterX() << ", " << circle1.getCenterY()
      << ").\n";
    std::cout << "The center of circle2 is located at ("
      << circle2.getCenterX() << ", " << circle2.getCenterY()
      << ").\n";
```

```
26
27      // get the area of both circles
28      std::cout << "The area of circle1 is " << circle1.getArea()
29        << " square units.\n";
30      std::cout << "The area of circle2 is " << circle2.getArea()
31        << " square units.\n";
32
33      std::cout << "Circle testing done.\n\nTesting Rectangle class...\n";
34
35      // define two rectangles
36      Rectangle rect1(9, 3), rect2(10, 11);
37
38      // test length/width getter functions
39      std::cout << "Length of rect1: " << rect1.getLength() << " units.\n";
40      std::cout << "Width of rect1: " << rect1.getWidth() << " units.\n";
41      std::cout << "Length of rect2: " << rect2.getLength() << " units.\n";
42      std::cout << "Width of rect2: " << rect2.getWidth() << " units.\n";
43
44      // test area
45      std::cout << "Area of rect1: " << rect1.getArea() << " square units.\n";
46      std::cout << "Area of rect2: " << rect2.getArea() << " square units.\n";
47
48      return 0;
49  }
50
```

Output of Shapes testing:

```
Testing Circle class...
The center of circle1 is located at (0, 0).
The center of circle2 is located at (1, 1).
The area of circle1 is 78.5397 square units.
The area of circle2 is 314.159 square units.
Circle testing done.

Testing Rectangle class...
Length of rect1: 9 units.
Width of rect1: 3 units.
Length of rect2: 10 units.
Width of rect2: 11 units.
Area of rect1: 27 square units.
Area of rect2: 110 square units.
```