PROGRAMMING WITH C++

DTS102TC

JIANJIA WANG

JIANJIA.WANG@XJTLU.EDU.CN

SCHOOL OF AI AND ADVANCED COMPUTING

Xi'an Jiaotong-Liverpool University

西交利物浦大学

CHAPTER 10

OBJECT-ORIENTED THINKING

Xi'an Jiaotong-Liverpool University
西交利物浦大學

OBJECTIVES

- To process strings using the **string** class (§10.2).
- To develop functions with object arguments (§10.3).
- To store and process objects in arrays (§10.4).
- To distinguish between instance and static variables and functions (§10.5).
- To define constant functions to prevent data fields from being modified accidentally (§10.6).
- To explore the differences between the procedural paradigm and object-oriented paradigm (§10.7).
- To design a class for body mass index (§10.7).
- To develop classes for modeling composition relationships (§10.8).
- To design a class for a stack (§10.9).
- To design classes that follow the class-design guidelines (§10.10).

CONSTRUCTING A STRING

You can create an empty string using string's no-arg constructor like this one:

```
string newString;
```

You can create a string object from a string value or from an array of characters. To create a string from a string literal, use a syntax like this one:

```
string newString(stringLiteral);
```

APPENDING A STRING

string	
+append(s: string): string	Appends string s into this string object.
+append(s: string, index: int, n: int): string	Appends n number of characters in s starting at the position index to this string.
+append(s: string, n: int): string	Appends the first n number of characters in s to this string.
+append(n: int, ch: char): string	Appends n copies of character ch to this string.

APPENDING A STRING EXAMPLES

You can use several overloaded functions to add new contents to a string. For example, see the following code:

```
string s1("Welcome");
s1.append(" to C++"); // appends " to C++" to s1
cout << s1 << endl; // s1 now becomes Welcome to C++
string s2("Welcome");
s2.append(" to C and C++", 0, 5); // appends " to C" to s2
cout << s2 << endl; // s2 now becomes Welcome to C
string s3("Welcome");
s3.append(" to C and C++", 5); // appends " to C" to s3
cout << s3 << endl; // s3 now becomes Welcome to C
string s4("Welcome");
s4.append(4, 'G'); // appends "GGGG" to s4
cout << s4 << endl; // s4 now becomes WelcomeGGGG
```

ASSIGNING A STRING

string	
+assign(s[]: char): string	Assigns array of characters or a string s to this string.
+assign(s: string, index: int, n: int): string	Assigns n number of characters in s starting at the position index to this string.
+assign(s: string, n: int): string	Assigns the first n number of characters in s to this string.
+assign(n: int, ch: char): string	Assigns n copies of character ch to this string.

ASSIGNING A STRING EXAMPLES

You can use several overloaded functions to assign new contents to a string. For example, see the following code:

```
string s1("Welcome");
s1.assign("Dallas"); // assigns "Dallas" to s1
cout << s1 << endl; // s1 now becomes Dallas
string s2("Welcome");
s2.assign("Dallas, Texas", 0, 5); // assigns "Dalla" to s2
cout << s2 << endl; // s2 now becomes Dalla
string s3("Welcome");
s3.assign("Dallas, Texas", 5); // assigns "Dalla" to s3
cout << s3 << endl; // s3 now becomes Dalla
string s4("Welcome");
s4.assign(4, 'G'); // assigns "GGGG" to s4
cout << s4 << endl; // s4 now becomes GGGG
```

FUNCTIONS AT, CLEAR, ERASE, AND EMPTY

string	
+at(index: int): char	Returns the character at the position index from this string.
+clear(): void	Removes all characters in this string.
+erase(index: int, n: int): string	Removes n characters from this string starting at position index.
+empty(): bool	Returns true if this string is empty.
+front(): char	Returns the first character in the string.
+back(): char	Returns the last character in the string.

C++11: front() and back() are defined in C++11

FUNCTIONS AT, CLEAR, ERASE, AND EMPTY EXAMPLES

You can use the at(index) function to retrieve a character at a specified index, clear() to clear the string, erase(index, n) to delete part of the string, and empty() to test if a string is empty. For example, see the following code:

```
string s1("Welcome");
cout << s1.at(3) << endl; // s1.at(3) returns c
cout << s1.erase(2, 3) << endl; // s1 is now Weme
s1.clear(); // s1 is now empty
cout << s1.empty() << endl; // s1.empty returns 1
(means true)
```

FUNCTIONS LENGTH, SIZE, CAPACITY, AND C_STR()

string	
+length(): int	Returns the number of characters in this string.
+size(): int	Same as length().
+capacity(): int	Returns the size of the storage allocated for this string.
+c_str(): char[]	Returns a C-string for this string.
+data(): char[]	Same as c_str().

FUNCTIONS LENGTH, SIZE, CAPACITY, AND C_STR() EXAMPLES

```
string s1("Welcome");
cout << s1.length() << endl; // Length is 7
cout << s1.size() << endl; // Size is 7
cout << s1.capacity() << endl; // Capacity is 7
s1.erase(1, 2);
cout << s1.length() << endl; // Length is now 5
cout << s1.size() << endl; // Size is now 5
cout << s1.capacity() << endl; // Capacity is still 7
```

COMPARING STRINGS

Often, in a program, you need to compare the contents of two strings.

You can use the compare function. This function works in the same way as the C-string strcmp function and returns a value greater than 0, 0, or less than 0. For example, see the following code:

```
string s1("Welcome");
string s2("Welcomg");
cout << s1.compare(s2) << endl; // returns -2
cout << s2.compare(s1) << endl; // returns 2
cout << s1.compare("Welcome") << endl; // returns 0
```

OBTAINING SUBSTRINGS

You can obtain a single character from a string using the at function.

You can also obtain a substring from a string using the substr function. For example, see the following code:

```
string s1("Welcome");
cout << s1.substr(0, 1) << endl; // returns W
cout << s1.substr(3) << endl; // returns come
cout << s1.substr(3, 3) << endl; // returns com
```

SEARCHING IN A STRING

You can use the find function to search for a substring or a character in a string.

For example, see the following code:

```
string s1("Welcome to HTML");
cout << s1.find("co") << endl; // returns 3
cout << s1.find("co", 6) << endl; // returns -1
cout << s1.find('o') << endl; // returns 4
cout << s1.find('o', 6) << endl; // returns 9
```

INSERTING AND REPLACING STRINGS

Here are the examples to use the insert and replace functions:

```
string s1("Welcome to HTML");
s1.insert(11, "C++ and ");
cout << s1 << endl; // s1 becomes Welcome to C++ and HTML
string s2("AA");
s2.insert(1, 4, 'B');
cout << s2 << endl; // s2 becomes to ABBBBA
string s3("Welcome to HTML");
s3.replace(11, 4, "C++");
cout << s3 << endl; // returns Welcome to C++
```

STRING OPERATORS

Operator	Description
[]	Accesses characters using the array subscript operators.
=	Copies the contents of one string to the other.
+	Concatenates two strings into a new string.
+=	Appends the contents of one string to the other.
<<	Inserts a string to a stream
>>	Extracts characters from a stream to a string delimited by a whitespace or the null terminator character.
==, !=, <, <=, >, >=	Six relational operators for comparing strings.

CONVERTING NUMBERS TO STRINGS

You can also use the **itoa** function to convert an integer to a string. Sometimes you need to convert a floating-point number to a string.

You can write a function to perform the conversion. However, a simple approach is to use the **stringstream** class in the **<sstream>** header.

```
1  stringstream ss;
2  ss << 3.1415;
3  string s = ss.str();
```

SPLITTING STRINGS

Often you need to extract the words from a string. Assume that the words are separated by whitespaces.

You can use the **stringstream** class discussed in the preceding section to accomplish this task.

ExtractWords.cpp

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

int main()
{
  string text("Programming is fun");
  stringstream ss(text);

  cout << "The words in the text are " << endl;
  string word;
  while (!ss.eof())
  {
    ss >> word;
    cout << word << endl;
  }

  return 0;
}
```

```
The words in the text are
Programming
is
fun
```

ReplaceString.cpp

```
#include <iostream>
#include <string>
using namespace std;
// Replace oldSubStr in s with newSubStr
bool replaceString(string& s, const string&
oldSubStr,
  const string& newSubStr);
int main()
{
  // Prompt the user to enter s, oldSubStr, and
newSubStr
  cout << "Enter string s, oldSubStr, and
newSubStr: ";
  string s, oldSubStr, newSubStr;
  cin >> s >> oldSubStr >> newSubStr;
  bool isReplaced = replaceString(s, oldSubStr,
newSubStr);
  if (isReplaced)
    cout << "The replaced string is " << s << endl;
  else
    cout << "No matches" << endl;
  return 0;
}
```

```
bool replaceString(string& s, const string& oldSubStr,
  const string& newSubStr)
{
  bool isReplaced = false;
  int currentPosition = 0;
  while (currentPosition < s.length())
  {
    int position = s.find(oldSubStr, currentPosition);
    if (position == string::npos) // Mo more matches
      return isReplaced;
    else
    {
      s.replace(position, oldSubStr.length(), newSubStr);
      currentPosition = position + newSubStr.length();
      isReplaced = true; // At least one match
    }
  }

  return isReplaced;
}
```

```
Enter string s, oldSubStr, and newSubStr: Newsubstring
oldstring
compstring
No matches
```

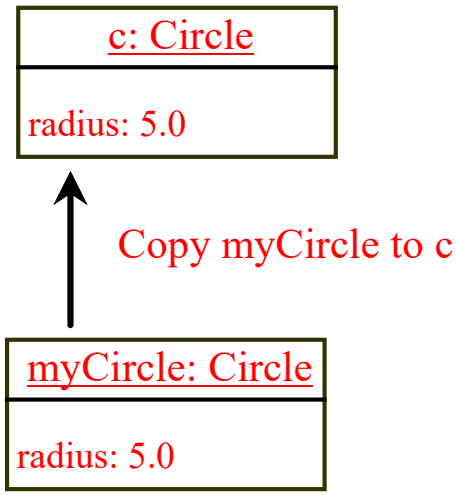
READING STRINGS

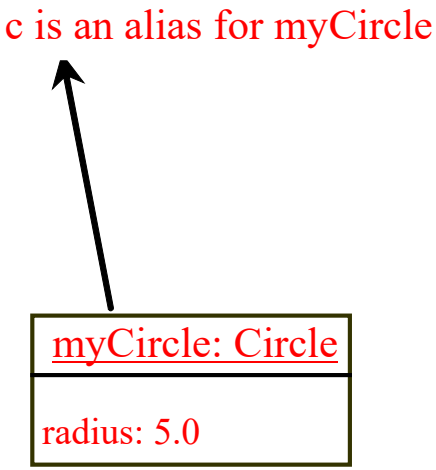
```
string city;
cout << "Enter a city: ";
cin >> city; // Read to array city
cout << "You entered " << city << endl;
```

```
string city;
cout << "Enter a city: ";
getline(cin, city, '\n'); // Same as getline(cin, city)
cout << "You entered " << city << endl;
```

PASSING OBJECTS TO FUNCTIONS

You can pass objects by value or by reference.





PassObjectByValue.cpp

```
#include <iostream>
// CircleWithPrivateDataFields.h is defined in Listing 9.9
#include "CircleWithPrivateDataFields.h"
using namespace std;

void printCircle(Circle c)
{
  cout << "The area of the circle of "
    << c.getRadius() << " is " << c.getArea() << endl;
}

int main()
{
  Circle myCircle(5.0);
  printCircle(myCircle);

  return 0;
}
```

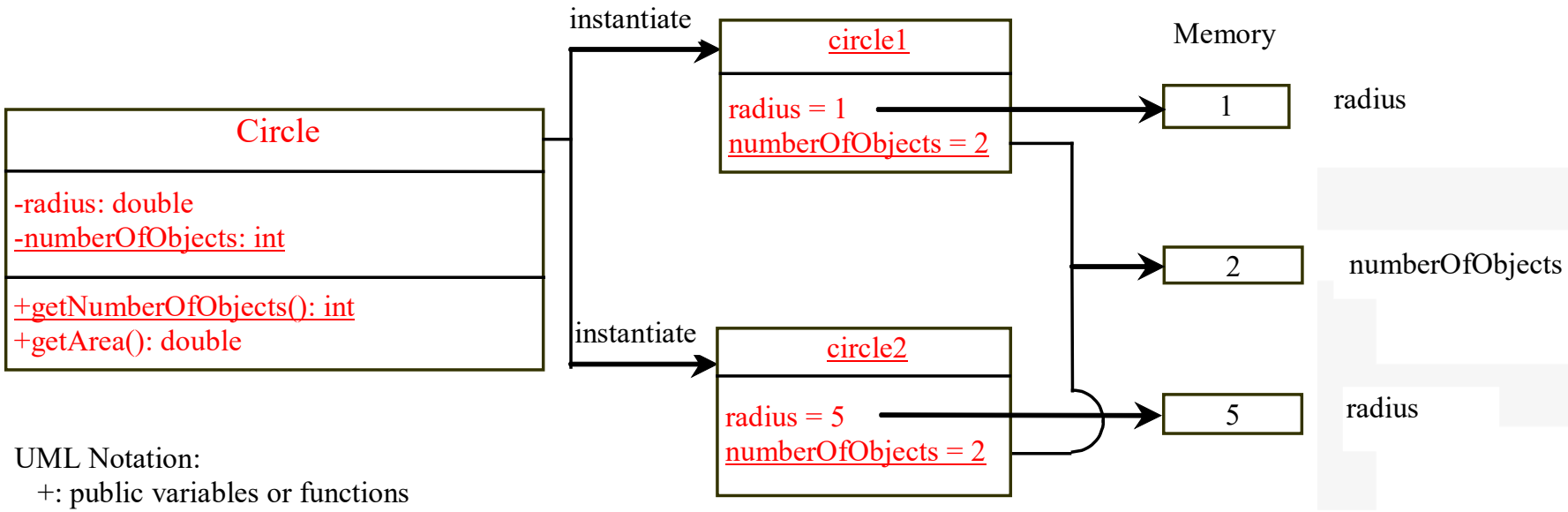
```
huakangleedeMacBook-Pro-7:Downloads huakanglee$ g++ -o PassObjectByValue PassObjectByVal
ue.cpp
Undefined symbols for architecture x86_64:
  "Circle::getArea()", referenced from:
      printCircle(Circle) in PassObjectByValue-a8f1e1.o
  "Circle::getRadius()", referenced from:
      printCircle(Circle) in PassObjectByValue-a8f1e1.o
  "Circle::Circle(double)", referenced from:
      _main in PassObjectByValue-a8f1e1.o
ld: symbol(s) not found for architecture x86_64
clang: error: linker command failed with exit code 1 (use -v to see invocation)
huakangleedeMacBook-Pro-7:Downloads huakanglee$ g++ -o PassObjectByValue PassObjectByVal
ue.cpp CircleWithPrivateDataFields.cpp
huakangleedeMacBook-Pro-7:Downloads huakanglee$ ./PassObjectByValue
The area of the circle of 5 is 78.5397
huakangleedeMacBook-Pro-7:Downloads huakanglee$ _
```

PassObjectByReference.cpp

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

void printCircle(Circle& c)
{
  cout << "The area of the circle of "
    << c.getRadius() << " is " << c.getArea() << endl;
}

int main()
{
  Circle myCircle(5.0);
  printCircle(myCircle);

  return 0;
}
```

```
huakangleedeMacBook-Pro-7:Downloads huakanglee$ g++ -o PassObjectByReference PassObjectB
yReference.cpp
Undefined symbols for architecture x86_64:
  "Circle::getArea()", referenced from:
      printCircle(Circle&) in PassObjectByReference-6b2cf1.o
  "Circle::getRadius()", referenced from:
      printCircle(Circle&) in PassObjectByReference-6b2cf1.o
  "Circle::Circle(double)", referenced from:
      _main in PassObjectByReference-6b2cf1.o
ld: symbol(s) not found for architecture x86_64
clang: error: linker command failed with exit code 1 (use -v to see invocation)
huakangleedeMacBook-Pro-7:Downloads huakanglee$ g++ -o PassObjectByReference PassObjectB
yReference.cpp CircleWithPrivateDataFields.cpp
huakangleedeMacBook-Pro-7:Downloads huakanglee$ ./PassObjectByReference
The area of the circle of 5 is 78.5397
huakangleedeMacBook-Pro-7:Downloads huakanglee$
```

INSTANCE AND STATIC MEMBERS



UML Notation:
+: public variables or functions
-: private variables or functions
underline: static variables or functions

TestCircleWithStaticDataFields.cpp

```
#include <iostream>
#include "CircleWithStaticDataFields.h"
using namespace std;
int main(){
  cout << "Number of circle objects created: "
    << Circle::getNumberOfObjects() << endl;
  Circle circle1;
  cout << "The area of the circle of radius "
    << circle1.getRadius() << " is " << circle1.getArea() << endl;
  cout << "Number of circle objects created: "
    << Circle::getNumberOfObjects() << endl;
  Circle circle2(5.0);
  cout << "The area of the circle of radius "
    << circle2.getRadius() << " is " << circle2.getArea() << endl;
  cout << "Number of circle objects created: "
      << Circle::getNumberOfObjects() << endl;
  circle1.setRadius(3.3);
  cout << "The area of the circle of radius "
    << circle1.getRadius() << " is " << circle1.getArea() << endl;
  cout << "circle1.getNumberOfObjects() returns "
    << circle1.getNumberOfObjects() << endl;
  cout << "circle2.getNumberOfObjects() returns "
    << circle2.getNumberOfObjects() << endl;
  return 0;
}
```

```
[huakangleedeMacBook-Pro-7:Downloads huakanglee$ g++ -o TestCir]
 cleWithStaticDataFields TestCircleWithStaticDataFields.cpp Cir
 cleWithStaticDataFields.cpp
[huakangleedeMacBook-Pro-7:Downloads huakanglee$ ./TestCircleWi]
 thStaticDataFields
Number of circle objects created: 0
The area of the circle of radius 1 is 3.14159
Number of circle objects created: 1
The area of the circle of radius 5 is 78.5397
Number of circle objects created: 2
The area of the circle of radius 3.3 is 34.2119
circle1.getNumberOfObjects() returns 2
circle2.getNumberOfObjects() returns 2
huakangleedeMacBook-Pro-7:Downloads huakanglee$
```

CircleWithStaticDataFields.h

```
#ifndef CIRCLE_H
#define CIRCLE_H

class Circle
{
public:
  Circle();
  Circle(double);
  double getArea();
  double getRadius();
  void setRadius(double);
  static int getNumberOfObjects();

private:
  double radius;
  static int numberOfObjects;
};

#endif
```

CircleWithStaticDataFields.cpp

```
#include "CircleWithStaticDataFields.h"
int Circle::numberOfObjects = 0;
// Construct a circle object
Circle::Circle(){
  radius = 1;
  numberOfObjects++;
}
// Construct a circle object
Circle::Circle(double newRadius){
  radius = newRadius;
  numberOfObjects++;
}
// 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;
}
// Return the number of circle objects
int Circle::getNumberOfObjects(){
  return numberOfObjects;
}
```

USE CLASS NAME

Use ClassName::functionName(arguments) to invoke a static function and ClassName::staticVariable. This improves readability because the user can easily recognize the static function and data in the class.

INSTANCE OR STATIC?

How do you decide whether a variable or function should be instance or static?

A variable or function that is dependent on a specific instance of the class should be an instance variable or function.

A variable or function that is not dependent on a specific instance of the class should be a static variable or function.

For example, every circle has its own radius. Radius is dependent on a specific circle.

Therefore, radius is an instance variable of the Circle class. Since the getArea function is dependent on a specific circle, it is an instance function.

Since numberOfObjects is not dependent on any specific instance, it should be declared static.

CONSTANT MEMBER FUNCTIONS

You can use the const keyword to specify a constant parameter to tell the compiler that the parameter should not be changed in the function.

C++ also enables you to specify a constant member function to tell the compiler that the function should not change the value of any data fields in the object.

To do so, place the const keyword at the end of the function header.

CircleWithConstantMemberFunctions.h

```
#ifndef CIRCLE_H
#define CIRCLE_H

class Circle
{
public:
  Circle();
  Circle(double);
  double getArea() const;
  double getRadius() const;
  void setRadius(double);
  static int getNumberOfObjects();

private:
  double radius;
  static int numberOfObjects;
};

#endif
```

CircleWithConstantMemberFunctions.cpp

```
#include "CircleWithConstantMemberFunctions.h"
int Circle::numberOfObjects = 0;
// Construct a circle object
Circle::Circle(){
  radius = 1;
  numberOfObjects++;
}
// Construct a circle object
Circle::Circle(double newRadius){
  radius = newRadius;
  numberOfObjects++;
}
// Return the area of this circle
double Circle::getArea() const{
  return radius * radius * 3.14159;
}
// Return the radius of this circle
double Circle::getRadius() const{
  return radius;
}
// Set a new radius
void Circle::setRadius(double newRadius){
  radius = (newRadius >= 0) ? newRadius : 0;
}
// Return the number of circle objects
int Circle::getNumberOfObjects(){
  return numberOfObjects;
}
```

OBJECT-ORIENTED THINKING

The module introduced fundamental programming techniques for problem solving using loops, functions, and arrays.

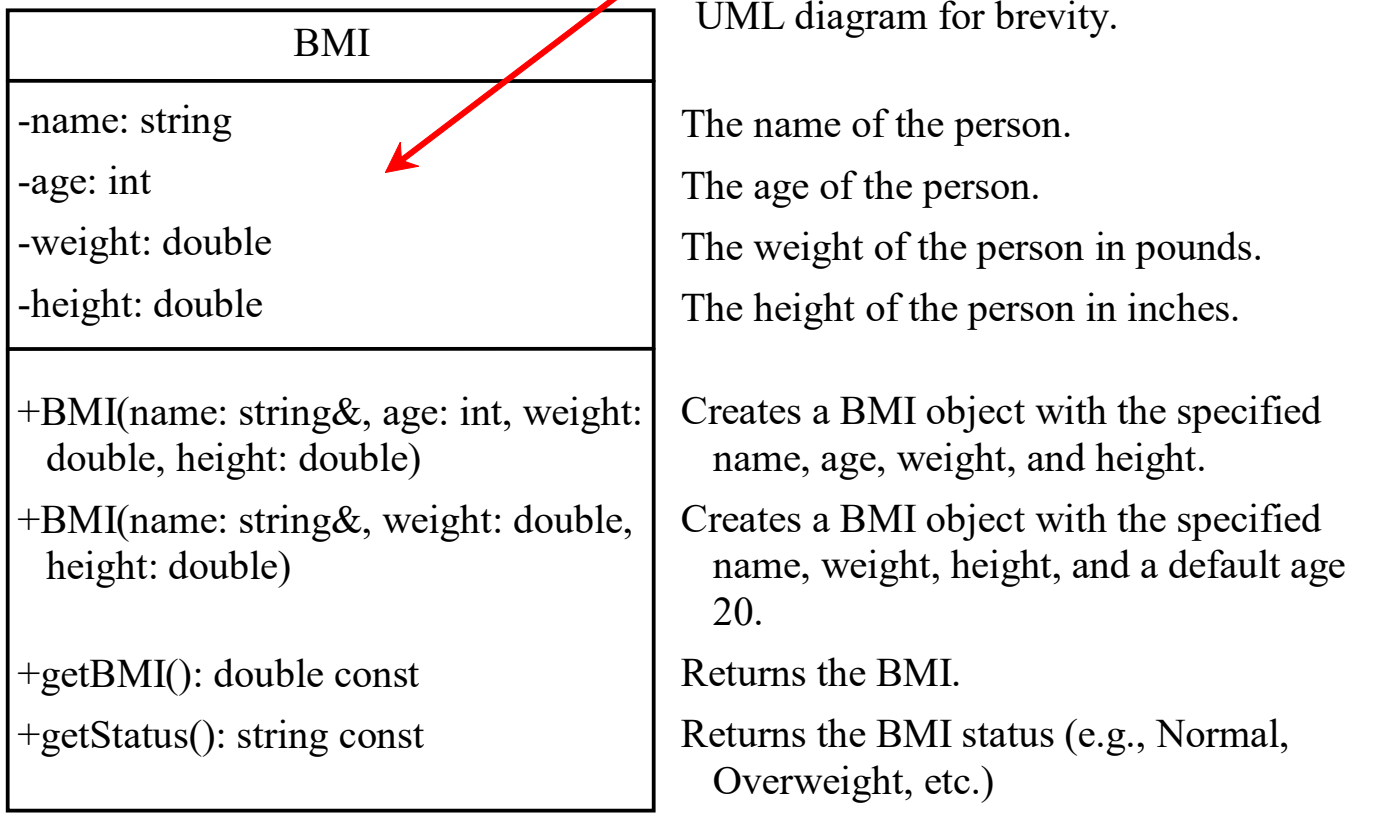
The study of these techniques lays a solid foundation for object-oriented programming.

Classes provide more flexibility and modularity for building reusable software.

This section improves the solution for a problem introduced in Chapter 3 using the object-oriented approach.

From the improvements, you will gain insight on the differences between the procedural programming and object-oriented programming and see the benefits of developing reusable code using objects and classes.

THE BMI CLASS



The get functions for these data fields are provided in the class, but omitted in the UML diagram for brevity.

The name of the person.

The age of the person.

The weight of the person in pounds.

The height of the person in inches.

Creates a BMI object with the specified name, age, weight, and height.

Creates a BMI object with the specified name, weight, height, and a default age 20.

Returns the BMI.

Returns the BMI status (e.g., Normal, Overweight, etc.)

UseBMIClass.cpp

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

int main()
{
  BMI bmi1("John Doe", 18, 145, 70);
  cout << "The BMI for " << bmi1.getName() << " is "
    << bmi1.getBMI() << " " << bmi1.getStatus() << endl;

  BMI bmi2("Susan King", 215, 70);
  cout << "The BMI for " << bmi2.getName() << " is "
    << bmi2.getBMI() << " " + bmi2.getStatus() << endl;

  return 0;
}
```

```
huakangleedeMacBook-Pro-7:Downloads huakanglee$ g++ -o Us
eBMIClass UseBMIClass.cpp BMI.cpp
huakangleedeMacBook-Pro-7:Downloads huakanglee$ ./UseBMIC
lass
The BMI for John Doe is 20.8051 Normal
The BMI for Susan King is 30.849 Obese
huakangleedeMacBook-Pro-7:Downloads huakanglee$ _
```

BMI.h

```
#ifndef BMI_H
#define BMI_H
#include <string>
using namespace std;
class BMI
{
public:
  BMI(const string& newName, int
newAge,
    double newWeight, double
newHeight);
  BMI(const string& newName, double
newWeight, double newHeight);
  double getBMI() const;
  string getStatus() const;
  string getName() const;
  int getAge() const;
  double getWeight() const;
  double getHeight() const;
private:
  string name;
  int age;
  double weight;
  double height;
};
#endif
```

BMI.cpp

```
#include <iostream>
#include "BMI.h"
using namespace std;
BMI::BMI(const string& newName, int
newAge,
  double newWeight, double
newHeight){
  name = newName;
  age = newAge;
  weight = newWeight;
  height = newHeight;}
BMI::BMI(const string& newName,
double newWeight, double
newHeight){
  name = newName;
  age = 20;
  weight = newWeight;
  height = newHeight;}
double BMI::getBMI() const{
  const double
KILOGRAMS_PER_POUND =
0.45359237;
  const double METERS_PER_INCH =
0.0254;
  double bmi = weight *
KILOGRAMS_PER_POUND /
    ((height * METERS_PER_INCH) *
(height * METERS_PER_INCH));
  return bmi;
}
string BMI::getStatus() const
{
  double bmi = getBMI();
  if (bmi < 18.5)
    return "Underweight";
  else if (bmi < 25)
    return "Normal";
  else if (bmi < 30)
    return "Overweight";
  else
    return "Obese";
}
string BMI::getName() const
{
  return name;
}
int BMI::getAge() const
{
  return age;
}
double BMI::getWeight() const
{
  return weight;
}
double BMI::getHeight() const
{
  return height;
}
```

DESIGNING A CLASS: COHESION

- A class should describe a single entity or a set of similar operations.
- A single entity with too many responsibilities can be broken into several classes to separate responsibilities.

DESIGNING A CLASS: CONSISTENCY

- Follow standard programming style and naming conventions.
- Choose informative names for classes, data fields, and functions.
- A popular style in C++ is to place the data declaration after the functions, and place constructors before functions.

DESIGNING A CLASS: ENCAPSULATION

- A class should use the private modifier to hide its data from direct access by clients. This makes the class easy to maintain.
- Provide a get function only if you want the field to be readable, and provide a set function only if you want the field to be updateable.
- A class should also hide functions not intended for client use. Such functions should be defined as private.

DESIGNING A CLASS: CLARITY

- Users can incorporate classes in many different combinations, orders, and environments.
- Therefore, you should design a class that imposes no restrictions on what or when the user can do with it, design the properties in a way that lets the user set them in any order and with any combination of values, and design functions independently of their order of occurrence.
- For example, the Loan class contains the functions setLoanAmount, setNumberOfYears, and setAnnualInterestRate. The values of these properties can be set in any order.

DESIGNING A CLASS: COMPLETENESS

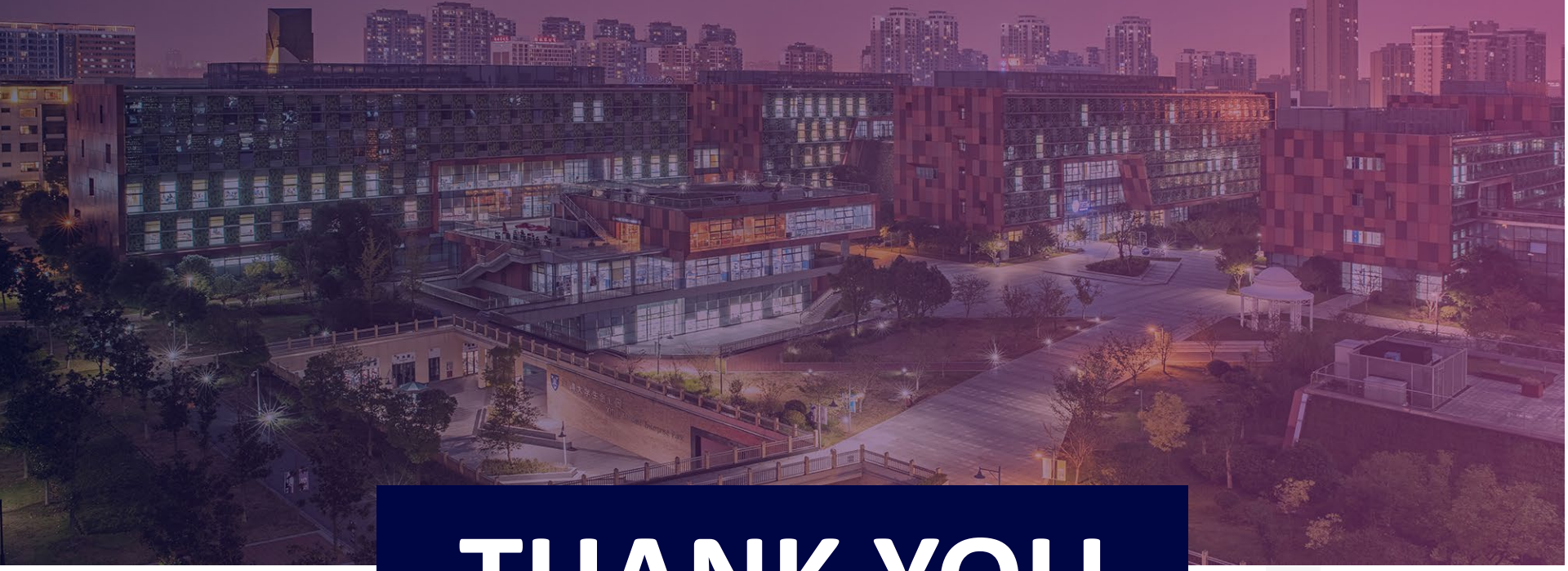
- Classes are designed for use by many different customers.
- In order to be useful in a wide range of applications, a class should provide a variety of ways for customization through properties and functions.
- For example, the string class contains more than 20 functions that are useful for a variety of applications.

DESIGNING A CLASS: INSTANCE VS. STATIC

- A variable or function that is dependent on a specific instance of the class should be an instance variable or function.
- A variable that is shared by all the instances of a class should be declared static.
- For example, the variable numberOfObjects in Circle in Listing 10.9, is shared by all the objects of the Circle class, and therefore is declared static.
- A function that is not dependent on a specific instance should be declared as a static function. For instance, the getNumberOfObjects function in Circle is not tied to any specific instance, and therefore is declared as static functions.

SUMMARIZATION

- Assigning String
- Function Length
- String Operators
- String Objects
- Constant Function
- Stack Class



THANK YOU

VISIT US

WWW.XJTLU.EDU.CN

FOLLOW US

Xi'an Jiaotong-Liverpool University
西交利物浦大学