Java programming for C/C++ developers Part II

Lecture 3

Dr. Tamer ABUHMED

Java Programming Course (SWE2023)

College of Computing

Outline

- Introduction to Class
- Primitive Type Values vs. Class Type Values
- Instance Variables and Methods
- Class Constructor
- Full Java Example
- Common Class differences between C++ and Java

A Class Is a Type

- A class is a special kind of programmer-defined type, and variables can be declared of a class type
- A value of a class type is called an object or an instance of the class
  - If A is a class, then the phrases “X is of type A," “X is an object of the class A," and “X is an instance of the class A" mean the same thing
- A class determines the types of data that an object can contain, as well as the actions it can perform

Example

football

tennis ball

rugby ball

Instances of the class Ball

Class Ball

Fields:

Color, Size, Shape

Methods:

Set_Ball_Color()

Set_Ball_Size()

Set_Ball_Shape()

Primitive Type Values vs. Class Type Values

- A primitive type value is a single piece of data
- A class type value or object can have multiple pieces of data, as well as actions called *methods*
  - ✓All objects of a class have the same methods
  - ✓All objects of a class have the same pieces of data (i.e., name, type, and number)
  - ✓For a given object, each piece of data can hold a different value

The Contents of a Class Definition

- A class definition specifies the data items *(fields)* and methods that all of its objects will have
- These data items and methods are sometimes called members of the object
- Data items are called fields or instance variables
- Instance variable declarations and method definitions can be placed in any order within the class definition

The **new** Operator

- An object of a class is named or declared by a variable of the class type:

```
ClassName  classVar;
```

- The **new** operator must then be used to create the object and associate it with its variable name:

```
classVar = new ClassName();
```

- These can be combined as follows:

```
ClassName classVar = new ClassName();
```

Instance Variables and Methods

- Instance variables can be defined as in the following two examples
  - Note the `public` modifier (for now):

```
public String  instanceVar1;
public int  instanceVar2;
```

- In order to refer to a particular instance variable, preface it with its object name as follows:

```
objectName.instanceVar1
objectName.instanceVar2
```

Constructors

- A *constructor* is a special kind of method that is designed to initialize the instance variables for an object:

  ```
  public ClassName(anyParameters){code}
  ```

  - A constructor must have the same name as the class
  - A constructor has no type returned, not even `void`
  - Constructors are typically overloaded

Constructors

- A constructor is called when an object of the class is created using **`new`**

  ```
  ClassName objectName = new ClassName(anyArgs);
  ```

  - The name of the constructor and its parenthesized list of arguments (if any) must follow the **`new`** operator
  - This is the **only** valid way to invoke a constructor: a constructor cannot be invoked like an ordinary method
- If a constructor is invoked again (using **`new`**), the first object is discarded and an entirely new object is created
  - If you need to change the values of instance variables of the object, use mutator methods instead

You Can Invoke Another Method in a Constructor

- The first action taken by a constructor is to create an object with instance variables
- Therefore, it is legal to invoke another method within the definition of a constructor, since it has the newly created object as its calling object
  - For example, mutator methods can be used to set the values of the instance variables
  - It is even possible for one constructor to invoke another

Class Example 1/2

Class Name

Fileds

mutator method

method

```
public class Time1
{
   private int hour; // 0 - 23
   private int minute; // 0 - 59
   private int second; // 0 - 59

   // set a new time value using universal time; throw an
   // exception if the hour, minute or second is invalid
   public void setTime( int h, int m, int s )
   {
      // validate hour, minute and second
      if ( ( h >= 0 && h < 24 ) && ( m >= 0 && m < 60 ) &&
         ( s >= 0 && s < 60 ) )
      {
         hour = h;
         minute = m;
         second = s;
      } // end if


         else
            throw new IllegalArgumentException(
               "hour, minute and/or second was out of range" );
      } // end method setTime

      // convert to String in universal-time format (HH:MM:SS)
      public String toUniversalString()
      {
         return String.format( "%02d:%02d:%02d", hour, minute, second );
      } // end method toUniversalString

      // convert to String in standard-time format (H:MM:SS AM or PM)
      public String toString()
      {
         return String.format( "%d:%02d:%02d %s",
            ( ( hour == 0 || hour == 12 ) ? 12 : hour % 12 ),
            minute, second, ( hour < 12 ? "AM" : "PM" ) );
      } // end method toString
   } // end class Time1
```

Class Example 2/2

Class Name

Main method

```java
   public class Time1Test
   {
     public static void main( String[] args )
     {
       // create and initialize a Time1 object
       Time1 time = new Time1(); // invokes Time1 constructor

       // output string representations of the time
       System.out.print( "The initial universal time is: " );
       System.out.println( time.toUniversalString() );
       System.out.print( "The initial standard time is: " );
       System.out.println( time.toString() );
       System.out.println(); // output a blank line

       // change time and output updated time
       time.setTime( 13, 27, 6 );
       System.out.print( "Universal time after setTime is: " );
       System.out.println( time.toUniversalString() );
       System.out.print( "Standard time after setTime is: " );
       System.out.println( time.toString() );
       System.out.println(); // output a blank line

    // attempt to set time with invalid values
    try
    {
      time.setTime( 99, 99, 99 ); // all values out of range
    } // end try
    catch ( IllegalArgumentException e )
    {
      System.out.printf( "Exception: %s\n\n", e.getMessage() );
    } // end catch

    // display time after attempt to set invalid values
    System.out.println( "After attempting invalid settings:" );
    System.out.print( "Universal time: " );
    System.out.println( time.toUniversalString() );
    System.out.print( "Standard time: " );
    System.out.println( time.toString() );
  } // end main
} // end class Time1Test
```

CLASS Object Oriented Programming

C++ vs. JAVA

Class

- Class is a template used to create objects
- A class consists of
  - a collection of *fields*, or *variables*
  - all the operations (called *methods*) that can be performed on those fields
  - can be *instantiated*
- A class describes objects and operations defined on those objects
- In Java everything belongs to a class and there are no global methods.
- Class methods in C++ can be defined either inside or outside of the class, but Java methods are only allowed to be defined inside of the class.

Class (C++ vs. JAVA)

- In Java Class, fields can be given initial values at the same time as they are declared; but direct assignments to fields cannot be done in C++;
- In C++, all field initialization must be performed using the constructor.
- C++ [stack-based objects] and Java use "." to accessing instance methods of objects but operator "->" also is used in C++ when accessing objects allocated in the heap
- Java class (and array) types are REFERENCE TYPES

Class (C++ vs. JAVA) 1/5

C++

```
class MyClass {
 public:
 void MyMethod();
 void MyInlineMethod() {}
};
void MyClass::MyMethod() {};
int main() {
 MyClass stack;
 stack.MyMethod();
 MyClass* heap = new MyClass();
 heap->MyMethod();
 delete heap; }
```

Java

```
class MyClass {
 public void MyMethod() {}

 public static void  main(String[] a)
 {
 MyClass c = new MyClass();
 c.MyMethod();
 }
}
```

Class (C++ vs. JAVA) 2/5

C++

```
class Car{          // Declares class Car
 int x;
public:
Car(): x(0) {}// Constructor for Car initializes
                  //  x to 0. If the initializer were
               //  omitted, the variable would not
                  //  be initialized to a specific
                 //  value.

  int Wheels (int i) { // Member function
     return 3*i + x;
   }
};
```

Java

```
class Car{             // Defines class Car
private int x;         // Member variable,
// normally variables are declared as
// private to enforce encapsulation
//initialized to 0 by default
 public Car() {  // Constructor for Car
    }
 public int Wheels(int i) {
// Member method
       return 3*i + x;
    }
}
```

Class (C++ vs. JAVA) 3/5

C++	Java
`Car a;`	`Car a;`
`// declares a to be a Car object value,`	`// declares a to be a reference to a Car object`
`// initialized using the default constructor.`	`a = new Car();`
`// Another constructor can be used as`	`// initializes using the default constructor`
`Car a(args);`	`// Another constructor can be used as`
	`Car a = new Car(args);`
`a.x = 5; // modifies the object a`	`a.x = 5; // modifies the object reference`
`cout << a.x << endl;`	`System.out.println(a.x);`
`// outputs 0, because b is`	`// outputs 0, because b points to`
`// a different object than a`	`// a different object than a`

Class (C++ vs. JAVA) 4/5

C++

```
Car *c;
// declares c to be a pointer to a
// car object (initially
// undefined; could point anywhere)

Car *d = c;
// binds d to reference the same object as c
c->x = 5;
// modifies the object referenced by c
c.Wheels (5);// invokes Car::Wheels() for a
c-> Wheels(5);
// invokes Car:: Wheels() for *c
```

Java

```
Car c;
// declares c to be a reference to a Car
// object (initially null if c is a class member;
// it is necessary to initialize c before use
// if it is a local variable)
Car d = c;
// binds d to reference the same object as c
c.x = 5;
// modifies the object referenced by c
c.Wheels (5); //invokes Car.Wheels()
c.Wheels(5); //invokes Car.Wheels()
```

Class (C++ vs. JAVA) 5/5

C++	Java

```cpp
const Car *a; // it is not possible to
                    //modify the object
                  // pointed to by a through a


Car *const b = new Car();
// a declaration of a "const" pointer
b = new Car ();
//ILLEGAL, it is not allowed to re-bind it
b->x = 5;
// LEGAL, the object can still be modified
```

```java
final Car a; // a declaration of a "final"
            // it is possible to modify the object,
        // but the reference will constantly point
           // to the first object assigned to it
final Car b = new Car();
// a declaration of a "final" reference
b = new Car();
// ILLEGAL, it is not allowed to re-bind it
b.x = 5;
// LEGAL, the object can still be modified
```

Summery

- Introduction to Class
- Primitive Type Values vs. Class Type Values
- Instance Variables and Methods
- Class Constructor
- Full Java Example
- Common Class differences between C++ and Java