EXAMPLE 49: UNDERSTAND THE BASIC IDEA OF INHERITANCE

EXAMPLE 49: UNDERSTAND THE BASIC IDEA OF INHERITANCE

A SHAPE CLASS

```
class Shape
{
private:
  string shapeType;
public:
  Shape()
  {
    cout << "SHAPE: Inside the default constructor" << endl;
  }

  Shape(string s) : shapeType(s)
  {
    cout << "SHAPE: Inside the 1-argument constructor" << endl;
  }

  Shape(const Shape& rhs)
  {
    cout << "SHAPE: Inside the copy constructor" << endl;
    shapeType = rhs.shapeType;
  }
  ~Shape()
  {
    cout << "SHAPE: Inside the destructor" << endl;
  }

  Shape& operator=(const Shape &rhs)
  {
    cout << "SHAPE: Inside the assignment operator" << endl;
    // 1. check for self-assignment
    if (this != &rhs)
      {
        shapeType = rhs.shapeType;
      }
    // 5.  Return *this
    return *this;
  }

};
```

A CIRCLE CLASS

```
class Circle : public Shape
{
private:
  float radius;
public:
  Circle() : radius(0.0)
  {
    cout << "CIRCLE: Inside the default constructor" << endl;
  }

  Circle(string s,float r) : Shape(s), radius(r)
  {
    cout << "CIRCLE: Inside the 1-argument constructor" << endl;
  }

  Circle(const Circle& rhs)
  {
    cout << "CIRCLE: Inside the copy constructor" << endl;
    radius =rhs.radius;
  }
  ~Circle()
  {
    cout << "CIRCLE: Inside the destructor" << endl;
  }

  Circle& operator=(const Circle &rhs)
  {
    cout << "CIRCLE: Inside the assignment operator" << endl;
    // 1. check for self-assignment
    if (this != &rhs)
      {
        Shape::operator=(rhs);
        radius =rhs.radius;
      }
    // 5.  Return *this
    return *this;
  }

};
```

A SIMPLE MAIN FUNCTION THAT CREATES SHAPE AND CIRCLE OBJECTS

```
int main()
{
  //Shape s("Some shape");
  cout << "Start of program" << endl;
  cout << endl << "Instantiate object c1" << endl;
  Circle c1("Some Shape", 3.5);
  cout << endl << "Instantiate object c2" << endl;
  Circle c2("Some other shape", 10.2);
  cout << endl << "Assign c1=c2" << endl;
  c1 = c2;
  cout << endl << "Instantiate object c3 (copy constructor)" << endl;
  Circle c3 = c1;
  cout << endl << "End of program" << endl;
}
```

A SHAPE CLASS

```
class Shape
{
private:
  string shapeType;
public:
  Shape()
  {
    cout << "SHAPE: Inside the default constructor" << endl;
  }

  Shape(string s) : shapeType(s)
  {
    cout << "SHAPE: Inside the 1-argument constructor" << endl;
  }

  Shape(const Shape& rhs)
  {
    cout << "SHAPE: Inside the copy constructor" << endl;
    shapeType = rhs.shapeType;
  }
  ~Shape()
  {
    cout << "SHAPE: Inside the destructor" << endl;
  }

  Shape& operator=(const Shape &rhs)
  {
    cout << "SHAPE: Inside the assignment operator" << endl;
    // 1. check for self-assignment
    if (this != &rhs)
      {
        shapeType = rhs.shapeType;
      }
    // 5.  Return *this
    return *this;
  }

};
```

A REALLY SIMPLE CLASS, WITH 1 PRIVATE MEMBER VARIABLE

IT PRINTS MESSAGES INSIDE ITS CONSTRUCTORS, DESTRUCTOR AND ASSIGNMENT OPERATOR

EXAMPLE 49: UNDERSTAND THE BASIC IDEA OF INHERITANCE

A SHAPE CLASS

```
class Shape
{
private:
  string shapeType;
public:
  Shape()
  {
    cout << "SHAPE: Inside the default constructor" << endl;
  }

  Shape(string s) : shapeType(s)
  {
    cout << "SHAPE: Inside the 1-argument constructor" << endl;
  }

  Shape(const Shape& rhs)
  {
    cout << "SHAPE: Inside the copy constructor" << endl;
    shapeType = rhs.shapeType;
  }
  ~Shape()
  {
    cout << "SHAPE: Inside the destructor" << endl;
  }

  Shape& operator=(const Shape &rhs)
  {
    cout << "SHAPE: Inside the assignment operator" << endl;
    // 1. check for self-assignment
    if (this != &rhs)
      {
        shapeType = rhs.shapeType;
      }
    // 5.  Return *this
    return *this;
  }

};
```

A CIRCLE CLASS

```
class Circle : public Shape
{
private:
  float radius;
public:
  Circle() : radius(0.0)
  {
    cout << "CIRCLE: Inside the default constructor" << endl;
  }

  Circle(string s,float r) : Shape(s), radius(r)
  {
    cout << "CIRCLE: Inside the 1-argument constructor" << endl;
  }

  Circle(const Circle& rhs)
  {
    cout << "CIRCLE: Inside the copy constructor" << endl;
    radius =rhs.radius;
  }
  ~Circle()
  {
    cout << "CIRCLE: Inside the destructor" << endl;
  }

  Circle& operator=(const Circle &rhs)
  {
    cout << "CIRCLE: Inside the assignment operator" << endl;
    // 1. check for self-assignment
    if (this != &rhs)
      {
        Shape::operator=(rhs);
        radius =rhs.radius;
      }
    // 5.  Return *this
    return *this;
  }

};
```

A SIMPLE MAIN FUNCTION THAT CREATES SHAPE AND CIRCLE OBJECTS

```
int main()
{
  //Shape s("Some shape");
  cout << "Start of program" << endl;
  cout << endl << "Instantiate object c1" << endl;
  Circle c1("Some Shape", 3.5);
  cout << endl << "Instantiate object c2" << endl;
  Circle c2("Some other shape", 10.2);
  cout << endl << "Assign c1=c2" << endl;
  c1 = c2;
  cout << endl << "Instantiate object c3 (copy constructor)" << endl;
  Circle c3 = c1;
  cout << endl << "End of program" << endl;
}
```

A CIRCLE CLASS

```
class Circle : public Shape
{
private:
  float radius;
public:
  Circle() : radius(0.0)
  {
    cout << "CIRCLE: Inside the default constructor" << endl;
  }

  Circle(string s,float r) : Shape(s), radius(r)
  {
    cout << "CIRCLE: Inside the 1-argument constructor" << endl;
  }

  Circle(const Circle& rhs)
  {
    cout << "CIRCLE: Inside the copy constructor" << endl;
    radius =rhs.radius;
  }
  ~Circle()
  {
    cout << "CIRCLE: Inside the destructor" << endl;
  }

  Circle& operator=(const Circle &rhs)
  {
    cout << "CIRCLE: Inside the assignment operator" << endl;
    // 1. check for self-assignment
    if (this != &rhs)
      {
        Shape::operator=(rhs);
        radius =rhs.radius;
      }
    // 5.  Return *this
    return *this;
  }

};
```

THIS IS MORE INTERESTING

A CIRCLE CLASS

```
class Circle : public Shape
```

```
{
private:
  float radius;
public:
  Circle() : radius(0.0)
  {
    cout << "CIRCLE: Inside the default constructor" << endl;
  }

  Circle(string s,float r) : Shape(s), radius(r)
  {
    cout << "CIRCLE: Inside the 1-argument constructor" << endl;
  }

  Circle(const Circle& rhs)
  {
    cout << "CIRCLE: Inside the copy constructor" << endl;
    radius =rhs.radius;
  }
  ~Circle()
  {
    cout << "CIRCLE: Inside the destructor" << endl;
  }

  Circle& operator=(const Circle &rhs)
  {
    cout << "CIRCLE: Inside the assignment operator" << endl;
    // 1. check for self-assignment
    if (this != &rhs)
      {
        Shape::operator=(rhs);
        radius =rhs.radius;
      }
    // 5.  Return *this
    return *this;
```

NOTE THE C++ SYNTAX FOR SPECIFYING INHERITANCE

A CIRCLE CLASS

```
class Circle : public Shape
{
private:
  float radius;
public:
  Circle() : radius(0.0)
  {
    cout << "CIRCLE: Inside the default constructor" << endl;
  }

  Circle(string s,float r) : Shape(s), radius(r)
  {
    cout << "CIRCLE: Inside the 1-argument constructor" << endl;
  }

  Circle(const Circle& rhs)
  {
    cout << "CIRCLE: Inside the copy constructor" << endl;
    radius =rhs.radius;
  }
  ~Circle()
  {
    cout << "CIRCLE: Inside the destructor" << endl;
  }

  Circle& operator=(const Circle &rhs)
  {
    cout << "CIRCLE: Inside the assignment operator" << endl;
    // 1. check for self-assignment
    if (this != &rhs)
      {
        Shape::operator=(rhs);
        radius =rhs.radius;
      }
    // 5.  Return *this
    return *this;
```

REMEMBER WE SAID THERE ARE 3 TYPES OF INHERITANCE? PUBLIC IS THE MOST COMMON BY FAR

A CIRCLE CLASS

```
Circle& operator=(const Circle &rhs)
 {
   cout << "CIRCLE: Inside the assignment operator" << endl;
   // 1. check for self-assignment
   if (this != &rhs)
     {
      Shape::operator=(rhs);
       radius =rhs.radius;
     }
   // 5.  Return *this
   return *this;
 }
```

IN THE ASSIGNMENT OPERATOR, CALL THE BASE CLASS ASSIGNMENT OPERATOR AS WELL!

A CIRCLE CLASS

```
Circle& operator=(const Circle &rhs)
  {
    cout << "CIRCLE: Inside the assignment operator" << endl;
    // 1. check for self-assignment
    if (this != &rhs)
      {
        Shape::operator=(rhs);
        radius =rhs.radius;
      }
    // 5.  Return *this
    return *this;
  }
```

IN THE ASSIGNMENT OPERATOR, CALL THE BASE CLASS ASSIGNMENT OPERATOR AS WELL!

A CIRCLE CLASS

CONSTRUCTORS, DESTRUCTOR, ASSIGNMENT OPERATOR PRINT MESSAGE WHEN CALLED

```
class Circle : public Shape
{
private:
  float radius;
public:
  Circle() : radius(0.0)
  {
    cout << "CIRCLE: Inside the default constructor" << endl;
  }

  Circle(string s,float r) : Shape(s), radius(r)
  {
    cout << "CIRCLE: Inside the 1-argument constructor" << endl;
  }

  Circle(const Circle& rhs)
  {
    cout << "CIRCLE: Inside the copy constructor" << endl;
    radius =rhs.radius;
  }
  ~Circle()
  {
    cout << "CIRCLE: Inside the destructor" << endl;
  }

  Circle& operator=(const Circle &rhs)
  {
    cout << "CIRCLE: Inside the assignment operator" << endl;
    // 1. check for self-assignment

    if (this != &rhs)
      {
        radius =rhs.radius;
      }
    // 5.  Return *this

    return *this;
  }
```

EXAMPLE 49: UNDERSTAND THE BASIC IDEA OF INHERITANCE

A SHAPE CLASS

```
class Shape
{
private:
  string shapeType;
public:
  Shape()
  {
    cout << "SHAPE: Inside the default constructor" << endl;
  }

  Shape(string s) : shapeType(s)
  {
    cout << "SHAPE: Inside the 1-argument constructor" << endl;
  }

  Shape(const Shape& rhs)
  {
    cout << "SHAPE: Inside the copy constructor" << endl;
    shapeType = rhs.shapeType;
  }
  ~Shape()
  {
    cout << "SHAPE: Inside the destructor" << endl;
  }

  Shape& operator=(const Shape &rhs)
  {
    cout << "SHAPE: Inside the assignment operator" << endl;
    // 1. check for self-assignment
    if (this != &rhs)
      {
        shapeType = rhs.shapeType;
      }
    // 5.  Return *this
    return *this;
  }

};
```

A SIMPLE MAIN FUNCTION THAT CREATES SHAPE AND CIRCLE OBJECTS

```
int main()
{
  //Shape s("Some shape");
  cout << "Start of program" << endl;
  cout << endl << "Instantiate object c1" << endl;
  Circle c1("Some Shape", 3.5);
  cout << endl << "Instantiate object c2" << endl;
  Circle c2("Some other shape", 10.2);
  cout << endl << "Assign c1=c2" << endl;
  c1 = c2;
  cout << endl << "Instantiate object c3 (copy constructor)" << endl;
  Circle c3 = c1;
  cout << endl << "End of program" << endl;
}
```

A CIRCLE CLASS

```
class Circle : public Shape
{
private:
  float radius;
public:
  Circle() : radius(0.0)
  {
    cout << "CIRCLE: Inside the default constructor" << endl;
  }

  Circle(string s,float r) : Shape(s), radius(r)
  {
    cout << "CIRCLE: Inside the 1-argument constructor" << endl;
  }

  Circle(const Circle& rhs)
  {
    cout << "CIRCLE: Inside the copy constructor" << endl;
    radius =rhs.radius;
  }
  ~Circle()
  {
    cout << "CIRCLE: Inside the destructor" << endl;
  }

  Circle& operator=(const Circle &rhs)
  {
    cout << "CIRCLE: Inside the assignment operator" << endl;
    // 1. check for self-assignment

    if (this != &rhs)
      {
        radius =rhs.radius;
      }
    // 5.  Return *this

    return *this;
  }

};
```

A SIMPLE MAIN FUNCTION THAT CREATES SHAPE AND CIRCLE OBJECTS

```
int main()
{

  cout << "Start of program" << endl;
  cout << endl << "Instantiate object c1" << endl;
  Circle c1("Some Shape", 3.5);
  cout << endl << "Instantiate object c2" << endl;
  Circle c2("Some other shape", 10.2);
  cout << endl << "Assign c1=c2" << endl;
  c1 = c2;
  cout << endl << "Instantiate object c3 (copy constructor)" << endl;
  Circle c3 = c1;
  cout << endl << "End of program" << endl;
}
```

THIS IS MOST INTERESTING

A SIMPLE MAIN FUNCTION THAT CREATES SHAPE AND CIRCLE OBJECTS

```
int main()
{

  cout << "Start of program" << endl;
  cout << endl << "Instantiate object c1" << endl;
  Circle c1("Some Shape", 3.5);
  cout << endl << "Instantiate object c2" << endl;
  Circle c2("Some other shape", 10.2);
  cout << endl << "Assign c1=c2" << endl;
  c1 = c2;
  cout << endl << "Instantiate object c3 (copy constructor)" << endl;
  Circle c3 = c1;
  cout << endl << "End of program" << endl;
}
```

THIS IS MOST INTERESTING

A SIMPLE MAIN FUNCTION THAT CREATES SHAPE AND CIRCLE OBJECTS

```
int main()
{

  cout << "Start of program" << endl;
  cout << endl << "Instantiate object c1" << endl;
  Circle c1("Some Shape", 3.5);
```

```
Vitthals-MacBook-Pro:~ vitthalsrinivasan$ g++ -Wall Example49.cpp
Vitthals-MacBook-Pro:~ vitthalsrinivasan$ ./a.out
 Start of program

 Instantiate object c1
SHAPE: Inside the 1-argument constructor
CIRCLE: Inside the 1-argument constructor
```

CIRCLE CONSTRUCTOR WAS CALLED AFTER SHAPE CONSTRUCTOR

A SIMPLE MAIN FUNCTION THAT CREATES SHAPE AND CIRCLE OBJECTS

CIRCLE CONSTRUCTOR WAS CALLED
AFTER SHAPE CONSTRUCTOR

THIS HAPPENED BY DEFAULT, WE HAD TO
DO NOTHING TO MAKE THIS HAPPEN

BUT WITH THE ASSIGNMENT OPERATOR,
WE NEED TO EXPLICITLY ENSURE THIS!

A SIMPLE MAIN FUNCTION THAT CREATES SHAPE AND CIRCLE OBJECTS

```
int main()
{
Assign c1=c2
CIRCLE: Inside the assignment operator
SHAPE: Inside the assignment operator
  Circle c2("Some other shape", 10.2);
  cout << endl << "Assign c1=c2" << endl;
  c1 = c2;
  cout << endl << "Instantiate object c3 (copy constructor)" << endl;
  Circle c3 = c1;
  cout << endl << "End of program" << endl;
}
```

THIS ONLY HAPPENED BECAUSE WE SETUP THE ASSIGNMENT OPERATOR CORRECTLY!

THIS ONLY HAPPENED BECAUSE WE SETUP THE ASSIGNMENT OPERATOR CORRECTLY!

```
Circle& operator=(const Circle &rhs)
 {
    cout << "CIRCLE: Inside the assignment operator" << endl;
    // 1. check for self-assignment
    if (this != &rhs)
       {
        Shape::operator=(rhs);
         radius =rhs.radius;
       }
    // 5.  Return *this
    return *this;
 }
```

IN THE ASSIGNMENT OPERATOR, CALL THE BASE CLASS ASSIGNMENT OPERATOR AS WELL!

A SIMPLE MAIN FUNCTION THAT CREATES SHAPE AND CIRCLE OBJECTS

```
int main()
{
Assign c1=c2
CIRCLE: Inside the assignment operator
SHAPE: Inside the assignment operator
  Circle c2("Some other shape", 10.2);
  cout << endl << "Assign c1=c2" << endl;
  c1 = c2;
  cout << endl << "Instantiate object c3 (copy constructor)" << endl;
  Circle c3 = c1;
  cout << endl << "End of program" << endl;
}
```

THIS ONLY HAPPENED BECAUSE WE SETUP THE ASSIGNMENT OPERATOR CORRECTLY!

A SIMPLE MAIN FUNCTION THAT CREATES SHAPE AND CIRCLE OBJECTS

```
int main()
{

  cout << "Start of program" << endl;
  cout << endl << "Instantiate object c1" << endl;
  Circle c1("Some Shape", 3.5);
  cout << endl << "Instantiate object c2" << endl;
  Circle c2("Some other shape", 10.2);
  cout << endl << "Assign c1=c2" << endl;
  c1 = c2;
  cout << endl << "Instantiate object c3 (copy constructor)" << endl;
  Circle c3 = c1;
  cout << endl << "End of program" << endl;
}
```

THIS IS MOST INTERESTING

A SIMPLE MAIN FUNCTION THAT CREATES SHAPE AND CIRCLE OBJECTS

```
End of program
CIRCLE: Inside the destructor
SHAPE: Inside the destructor
CIRCLE: Inside the destructor
SHAPE: Inside the destructor
CIRCLE: Inside the destructor
SHAPE: Inside the destructor
```

```
    cout << endl << "End of program" << endl;
}
```

PROGRAM EXIT TRIGGERS DESTRUCTOR CALLS

A SIMPLE MAIN FUNCTION THAT CREATES SHAPE AND CIRCLE OBJECTS

```
End of program
CIRCLE: Inside the destructor
SHAPE: Inside the destructor
CIRCLE: Inside the destructor
SHAPE: Inside the destructor
CIRCLE: Inside the destructor
SHAPE: Inside the destructor
```

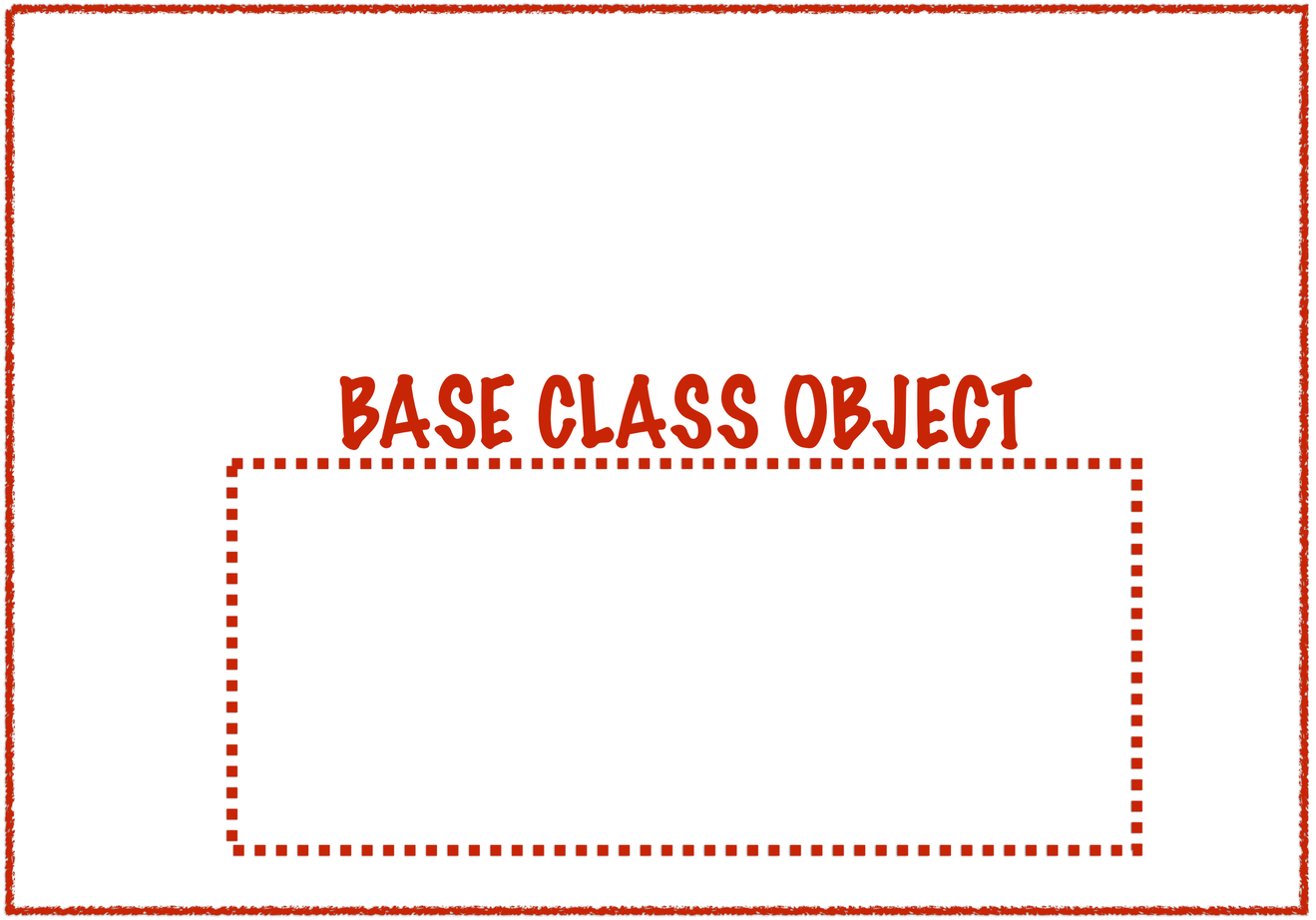
```
    cout << endl << "End of program" << endl;
}
```

WITH EACH CIRCLE DESTRUCTOR CALLING THE SHAPE DESTRUCTOR IMPLICITLY (AGAIN NOTHING WE NEEDED TO DO!)

"BASE CLASS" AND "DERIVED CLASS"

REMEMBER THESE TERMS, THEY ARE VERY IMPORTANT!

ORDER OF CONSTRUCTOR CALLS



ORDER OF DESTRUCTOR CALLS