EXAMPLE 55: MULTIPLE INHERITANCE, AND A DIAMOND HIERARCHY

EXAMPLE 55: MULTIPLE INHERITANCE, AND A DIAMOND HIERARCHY

C++ ALLOWS A CLASS TO INHERIT FROM MULTIPLE BASE CLASSES

THIS IS CALLED MULTIPLE INHERITANCE, AND IS NOT ALLOWED IN JAVA*

(IN JAVA YOU CAN INHERIT FROM MULTIPLE INTERFACES, BUT ONLY 1 BASE CLASS)

C++ ALLOWS A CLASS TO INHERIT FROM MULTIPLE BASE CLASSES

THIS IS CALLED MULTIPLE INHERITANCE, AND IS NOT ALLOWED IN JAVA*

(IN JAVA YOU CAN INHERIT FROM MULTIPLE INTERFACES, BUT ONLY 1 BASE CLASS)

MULTIPLE INHERITANCE LEADS TO SOME TRICKY SITUATIONS, WHICH IS WHY JAVA IS N'T TOO KEEN ON IT

EXAMPLE 48: BASICS OF INHERITANCE: LET'S GET ONE CLASS TO INHERIT FROM ANOTHER

WE HAVE A CLASS REPRESENTING A SHAPE

WE NOW NEED TO CREATE A CLASS REPRESENTING A CIRCLE

CLEARLY A CIRCLE IS-A SHAPE

RECAP

CLEARLY A CIRCLE IS-A SHAPE

WE HAVE A CLASS REPRESENTING A SHAPE

WE NOW NEED TO CREATE A CLASS REPRESENTING A CIRCLE

INHERITANCE IS PERFECT FOR MODELLING IS-A RELATIONSHIPS

RECAP

INHERITANCE IS PERFECT FOR MODELLING IS-A RELATIONSHIPS

WE HAVE A CLASS REPRESENTING A SHAPE

WE NOW NEED TO CREATE A CLASS REPRESENTING A CIRCLE

CLEARLY A CIRCLE IS-A SHAPE

THE CIRCLE CLASS SHOULD "INHERIT FROM" THE SHAPE CLASS

RECAP

IS-A RELATIONSHIP

REAL-WORLD RELATIONSHIP

WE HAVE A CLASS REPRESENTING A SHAPE

WE NOW NEED TO CREATE A CLASS REPRESENTING A CIRCLE

CLEARLY A CIRCLE IS A SHAPE

CODE RELATIONSHIP

THE CIRCLE CLASS SHOULD "INHERIT FROM" THE SHAPE CLASS

RECAP

THE CIRCLE CLASS SHOULD "INHERIT FROM" THE SHAPE CLASS

WHAT EXACTLY DOES THIS MEAN?

EVERY OBJECT OF THE CIRCLE CLASS WILL HAVE INSIDE IT AN OBJECT OF THE SHAPE CLASS

RECAP

THE CIRCLE CLASS SHOULD "INHERIT FROM" THE SHAPE CLASS

WHAT EXACTLY DOES THIS MEAN?

EVERY OBJECT OF THE CIRCLE CLASS WILL HAVE INSIDE IT AN OBJECT OF THE SHAPE CLASS

WHEN THE CIRCLE IS BEING CONSTRUCTED/ DESTRUCTED, THE SHAPE OBJECT NEEDS TO BE CONSTRUCTED/DESTRUCTED TOO

RECAP

THE CIRCLE CLASS SHOULD "INHERIT FROM" THE SHAPE CLASS

WHAT EXACTLY DOES THIS MEAN?

EVERY OBJECT OF THE CIRCLE CLASS WILL HAVE INSIDE IT AN OBJECT OF THE SHAPE CLASS

WHEN THE CIRCLE IS BEING CONSTRUCTED/DESTRUCTED, THE SHAPE OBJECT NEEDS TO BE CONSTRUCTED/DESTRUCTED TOO

THE CIRCLE OBJECT CAN DIRECTLY ACCESS SOME (NOT ALL) OF ITS INNER SHAPE OBJECT

RECAP

THE CIRCLE CLASS SHOULD "INHERIT FROM" THE SHAPE CLASS

WHAT EXACTLY DOES THIS MEAN?

EVERY OBJECT OF THE CIRCLE CLASS WILL HAVE INSIDE IT AN OBJECT OF THE SHAPE CLASS

WHEN THE CIRCLE IS BEING CONSTRUCTED/DESTRUCTED, THE SHAPE OBJECT NEEDS TO BE CONSTRUCTED/DESTRUCTED TOO

THE CIRCLE OBJECT CAN DIRECTLY ACCESS SOME (NOT ALL) OF ITS INNER SHAPE OBJECT

RECAP

THE CIRCLE CLASS SHOULD "INHERIT FROM" THE SHAPE CLASS

WHAT EXACTLY DOES THIS MEAN?

THERE IS A LOT GOING ON - LET'S TAKE IT ONE AT A TIME

THE OBJECTS OF THE CIRCLE CLASS WILL HAVE ITS OWN OBJECT OF THE SHAPE CLASS

WHEN THE CIRCLE IS BEING CONSTRUCTED/DESTRUCTED, THE SHAPE OBJECT NEEDS TO BE CONSTRUCTED/DESTRUCTED TOO

THE CIRCLE OBJECT CAN DIRECTLY ACCESS SOME (NOT ALL) OF ITS INNER SHAPE OBJECT

RECAP

THE CIRCLE CLASS SHOULD "INHERIT FROM" THE SHAPE CLASS

WHAT EXACTLY DOES THIS MEAN?

EVERY OBJECT OF THE CIRCLE CLASS WILL HAVE INSIDE IT AN OBJECT OF THE SHAPE CLASS

WHEN THE CIRCLE IS BEING CONSTRUCTED/DESTRUCTED, THE SHAPE OBJECT NEEDS TO BE CONSTRUCTED/DESTRUCTED TOO

THE CIRCLE OBJECT CAN DIRECTLY ACCESS SOME (NOT ALL) OF ITS INNER SHAPE OBJECT

RECAP

EVERY OBJECT OF THE CIRCLE CLASS WILL HAVE INSIDE IT AN OBJECT OF THE SHAPE CLASS

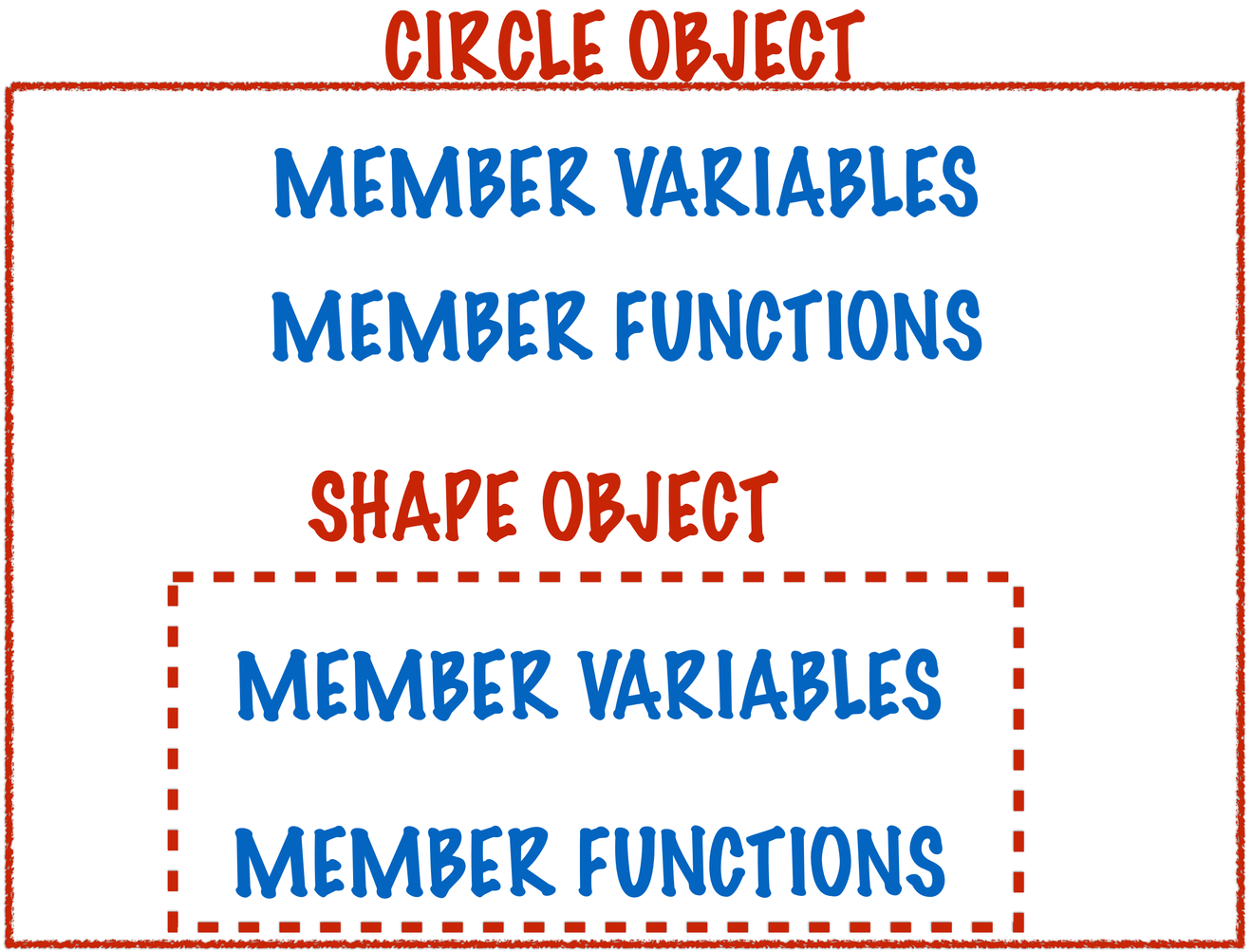
SHAPE OBJECT

MEMBER VARIABLES

MEMBER FUNCTIONS

RECAP

EVERY OBJECT OF THE CIRCLE CLASS WILL HAVE INSIDE IT AN OBJECT OF THE SHAPE CLASS



RECAP

C++ ALLOWS A CLASS TO INHERIT FROM MULTIPLE BASE CLASSES

THIS IS CALLED MULTIPLE INHERITANCE, AND IS NOT ALLOWED IN JAVA*

(IN JAVA YOU CAN INHERIT FROM MULTIPLE INTERFACES, BUT ONLY 1 BASE CLASS)

MULTIPLE INHERITANCE LEADS TO SOME TRICKY SITUATIONS, WHICH IS WHY JAVA IS N'T TOO KEEN ON IT

MULTIPLE INHERITANCE LEADS TO SOME TRICKY SITUATIONS, WHICH IS WHY JAVA IS N'T TOO KEEN ON IT

FOR INSTANCE, IT IS POSSIBLE FOR A DERIVED CLASS OBJECT TO END UP WITH 2 COPIES OF A BASE CLASS OBJECT INSIDE IT

MULTIPLE INHERITANCE LEADS TO SOME TRICKY SITUATIONS, WHICH IS WHY JAVA IS N'T TOO KEEN ON IT

FOR INSTANCE, IT IS POSSIBLE FOR A DERIVED CLASS OBJECT TO END UP WITH 2 COPIES OF A BASE CLASS OBJECT INSIDE IT

MULTIPLE INHERITANCE LEADS TO SOME TRICKY SITUATIONS, WHICH IS WHY JAVA IS N'T TOO KEEN ON IT

FOR INSTANCE, IT IS POSSIBLE FOR A DERIVED CLASS OBJECT TO END UP WITH 2 COPIES OF A BASE CLASS OBJECT INSIDE IT

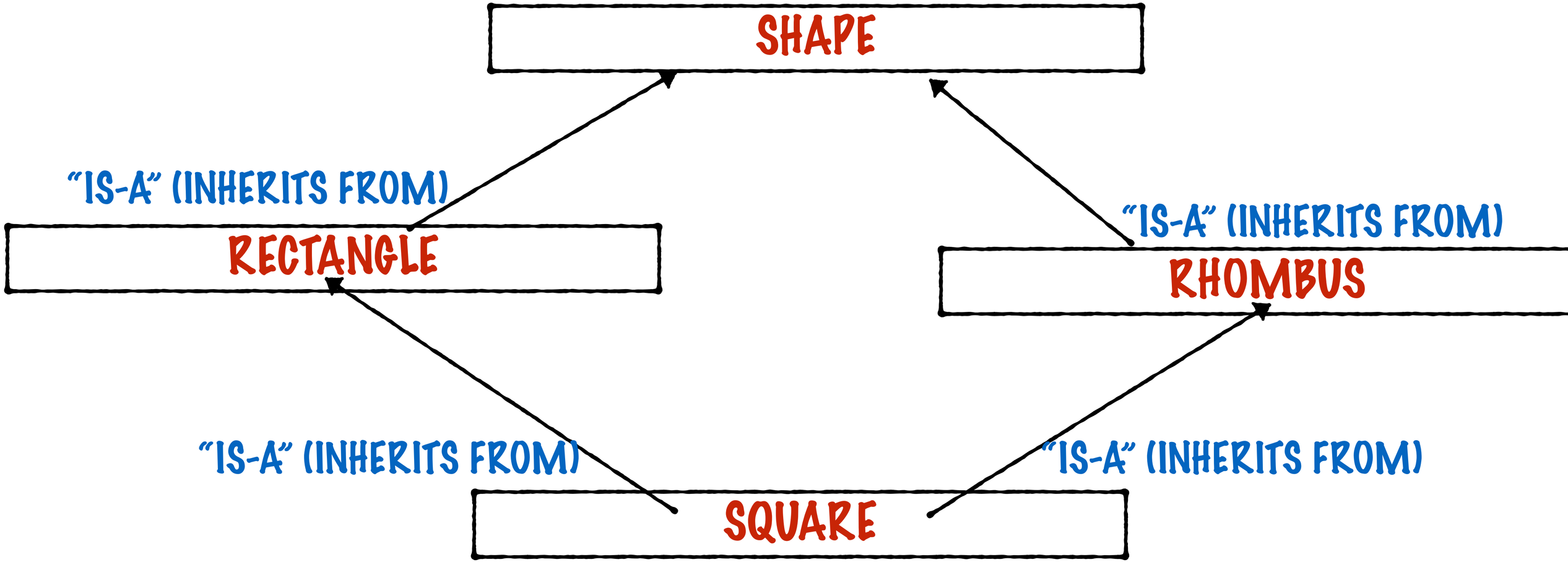
MULTIPLE INHERITANCE LEADS TO SOME TRICKY SITUATIONS, WHICH IS WHY JAVA IS N'T TOO KEEN ON IT

FOR INSTANCE, IT IS POSSIBLE FOR A DERIVED CLASS OBJECT TO END UP WITH 2 COPIES OF A BASE CLASS OBJECT INSIDE IT

FOR INSTANCE, IT IS POSSIBLE FOR A DERIVED CLASS OBJECT TO END UP WITH 2 COPIES OF A BASE CLASS OBJECT INSIDE IT

THERE IS NOTHING INHERENTLY WRONG WITH THIS, BUT IT GETS COMPLICATED TO WORK WITH

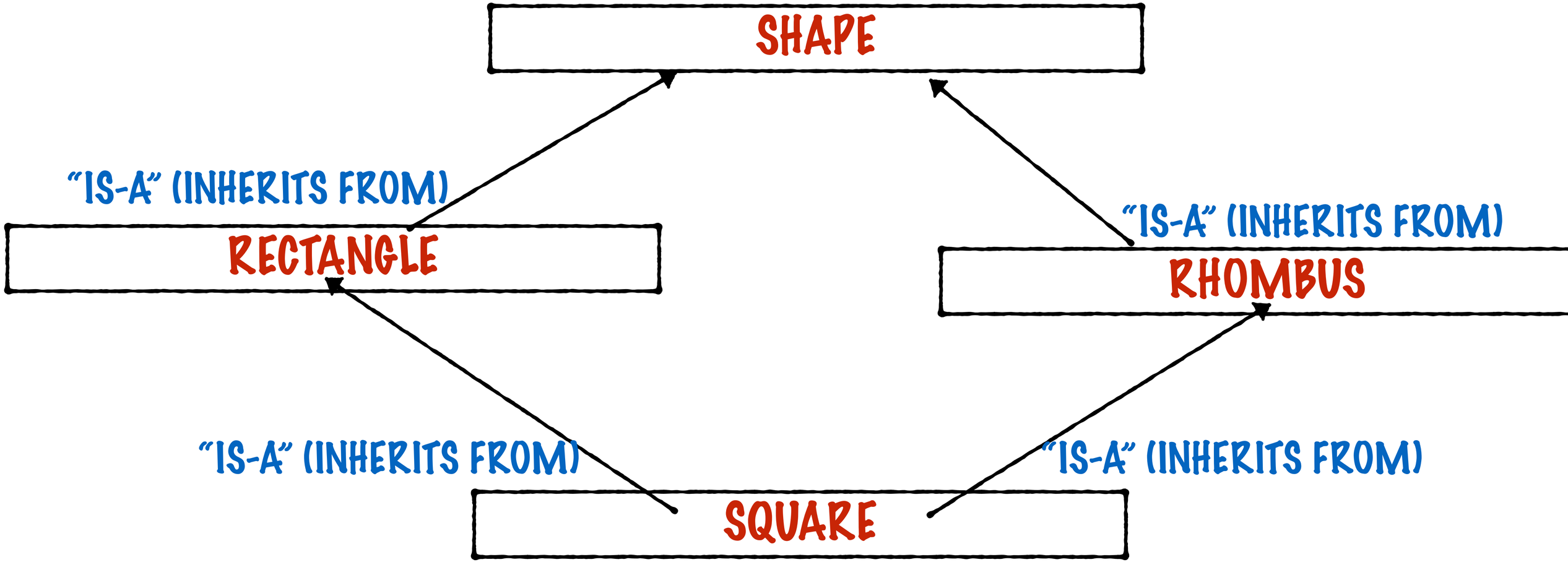
FOR INSTANCE, IT IS POSSIBLE FOR A DERIVED CLASS OBJECT TO END UP WITH 2 COPIES OF A BASE CLASS OBJECT INSIDE IT



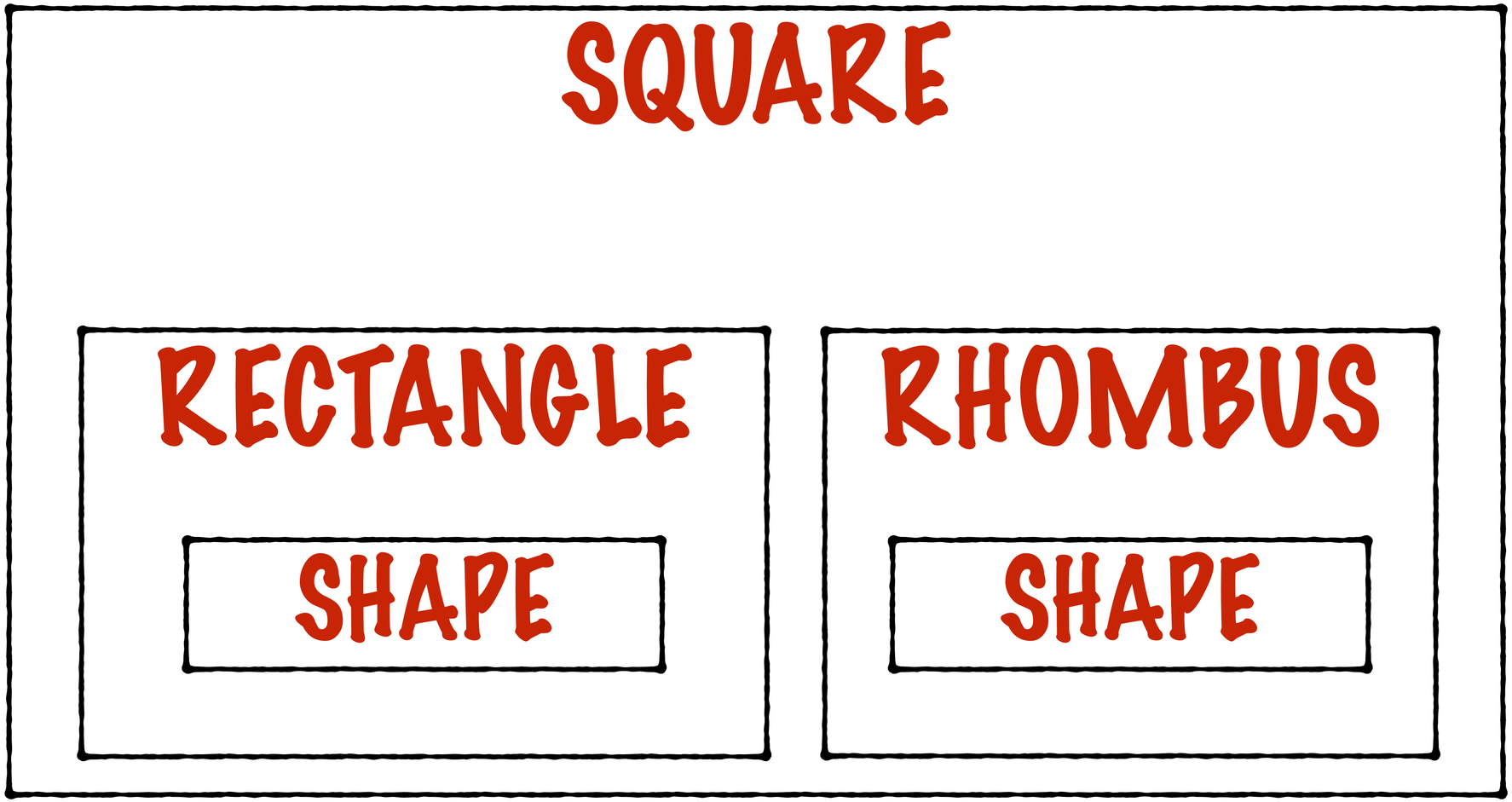
NOW THIS IS ALL PERFECTLY REASONABLE FROM A REAL-WORLD POINT-OF-VIEW..

..BUT THIS DIAMOND-SHAPED CLASS DIAGRAM LEAVES THE SQUARE OBJECT WITH 2 COPIES OF THE SHAPE OBJECT

..BUT THIS DIAMOND-SHAPED CLASS DIAGRAM LEAVES THE SQUARE OBJECT WITH 2 COPIES OF THE SHAPE OBJECT



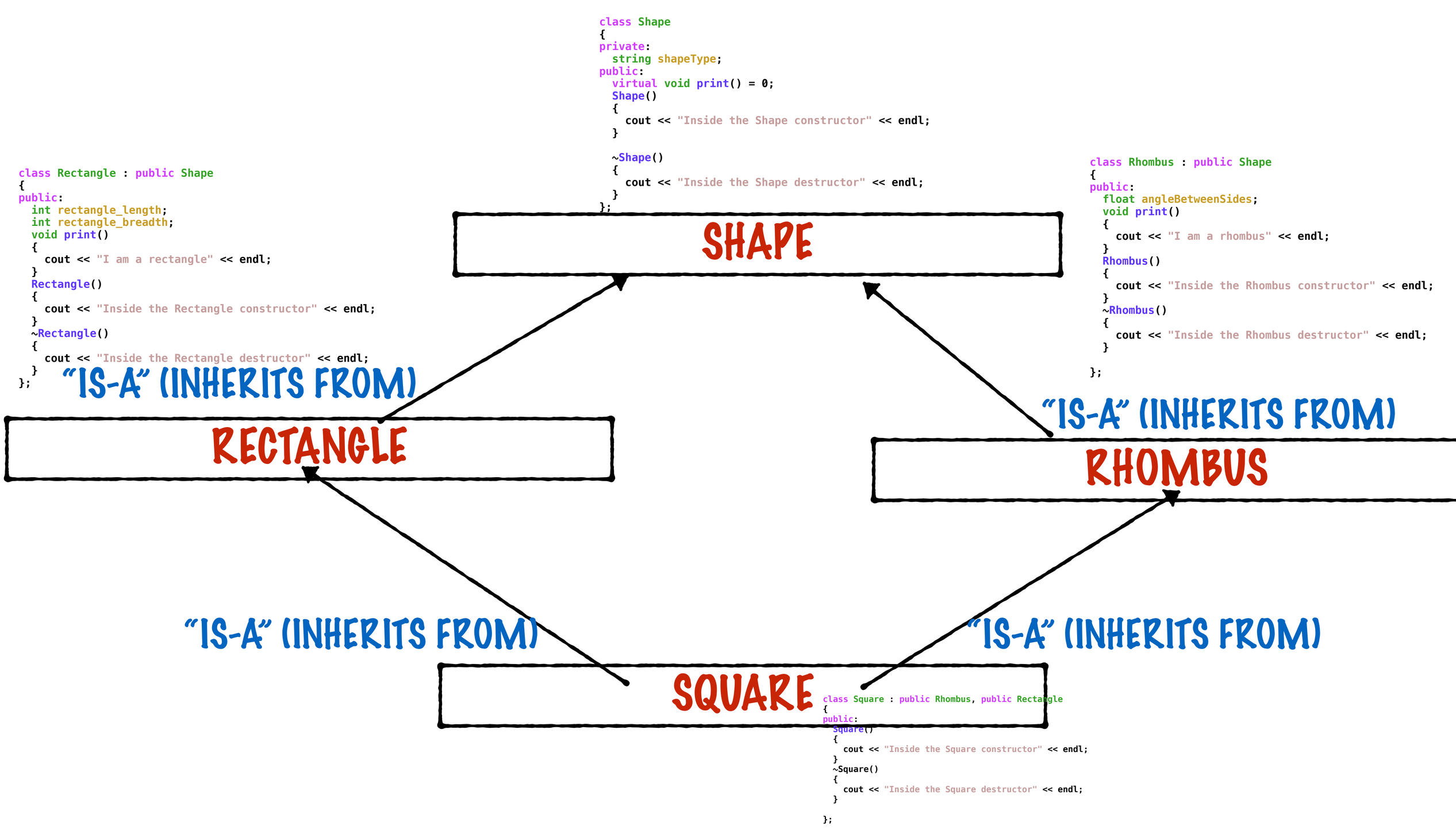
THIS DIAMOND-SHAPED CLASS DIAGRAM LEAVES THE SQUARE OBJECT WITH 2 COPIES OF THE SHAPE OBJECT



LAYOUT OF AN OBJECT OF CLASS SQUARE

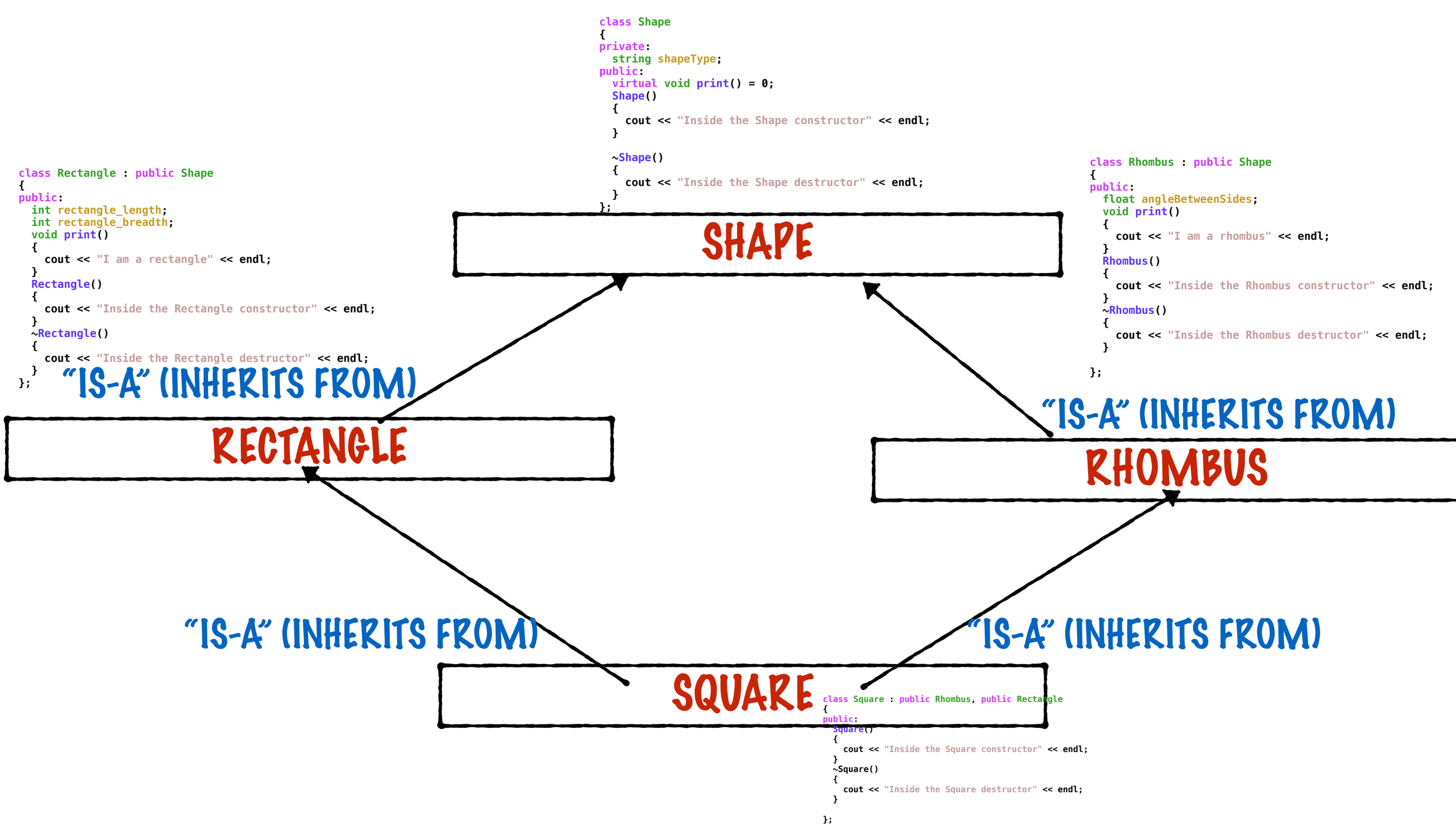
NOW IF YOU WANT TO CALL A METHOD OF THE SHAPE CLASS, YOU HAVE EXPLICITLY SPECIFY WHICH OF THE 2 VERSIONS YOU ARE REFERRING TO

DO SO USING THE SCOPE RESOLUTION OPERATOR.

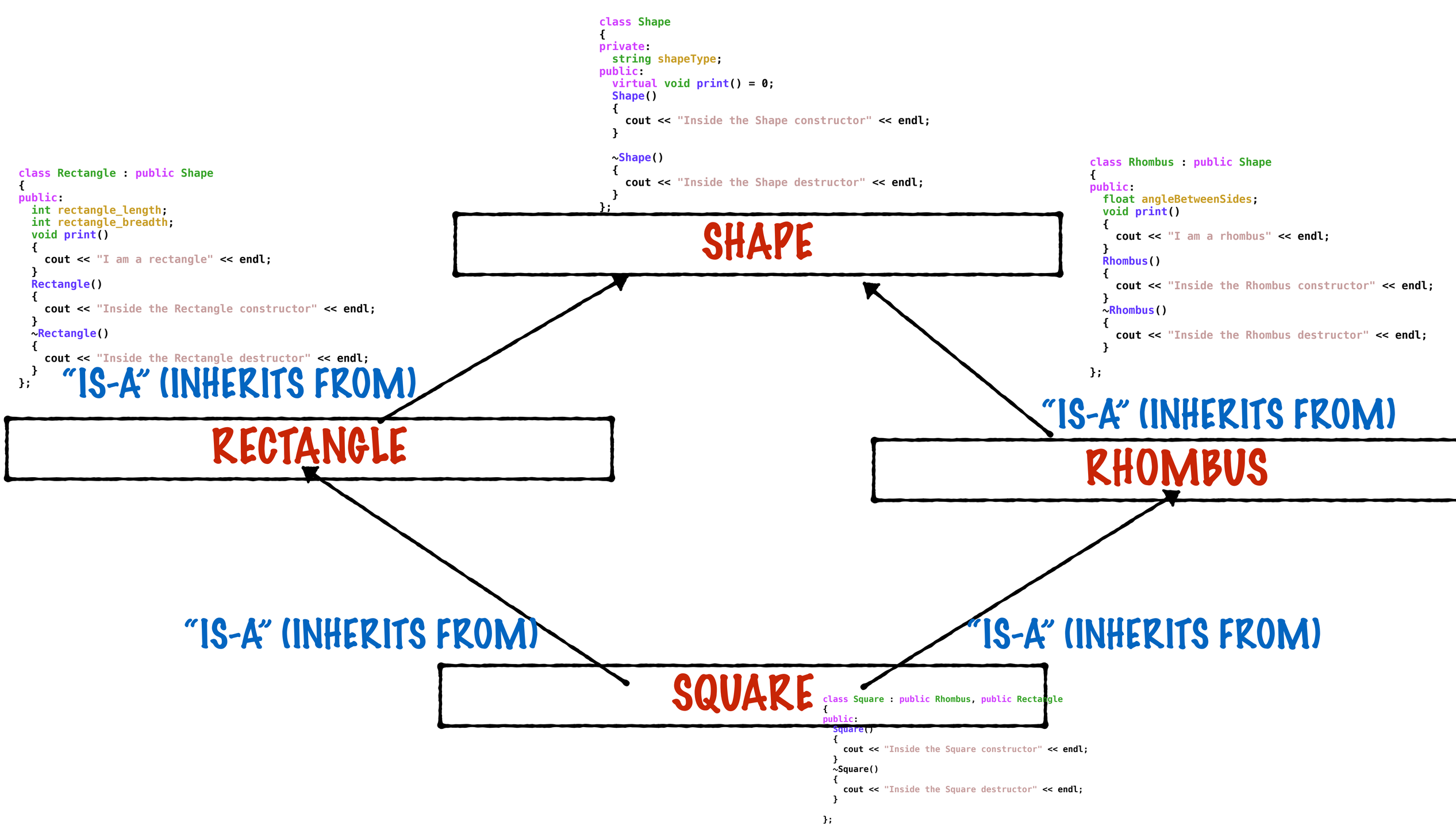

class Shape
{
private:
string shapeType;
public:
virtual void print() = 0;
Shape()
{
cout << "Inside the Shape constructor" << endl;
}
~Shape()
{
cout << "Inside the Shape destructor" << endl;
}
};
SHAPE
class Rectangle : public Shape
{
public:
int rectangle_length;
int rectangle_breadth;
void print()
{
cout << "I am a rectangle" << endl;
}
Rectangle()
{
cout << "Inside the Rectangle constructor" << endl;
}
~Rectangle()
{
cout << "Inside the Rectangle destructor" << endl;
}
};
"IS-A" (INHERITS FROM)
RECTANGLE
class Rhombus : public Shape
{
public:
float angleBetweenSides;
void print()
{
cout << "I am a rhombus" << endl;
}
Rhombus()
{
cout << "Inside the Rhombus constructor" << endl;
}
~Rhombus()
{
cout << "Inside the Rhombus destructor" << endl;
}
};
"IS-A" (INHERITS FROM)
RHOMBUS
"IS-A" (INHERITS FROM)
"IS-A" (INHERITS FROM)
SQUARE
class Square : public Rhombus, public Rectangle
{
public:
Square()
{
cout << "Inside the Square constructor" << endl;
}
~Square()
{
cout << "Inside the Square destructor" << endl;
}
};

```
class Shape
{
private:
   string shapeType;
public:
   virtual void print() = 0;
   Shape()
   {
      cout << "Inside the Shape constructor" << endl;
   }

   ~Shape()
   {
      cout << "Inside the Shape destructor" << endl;
   }
};
```

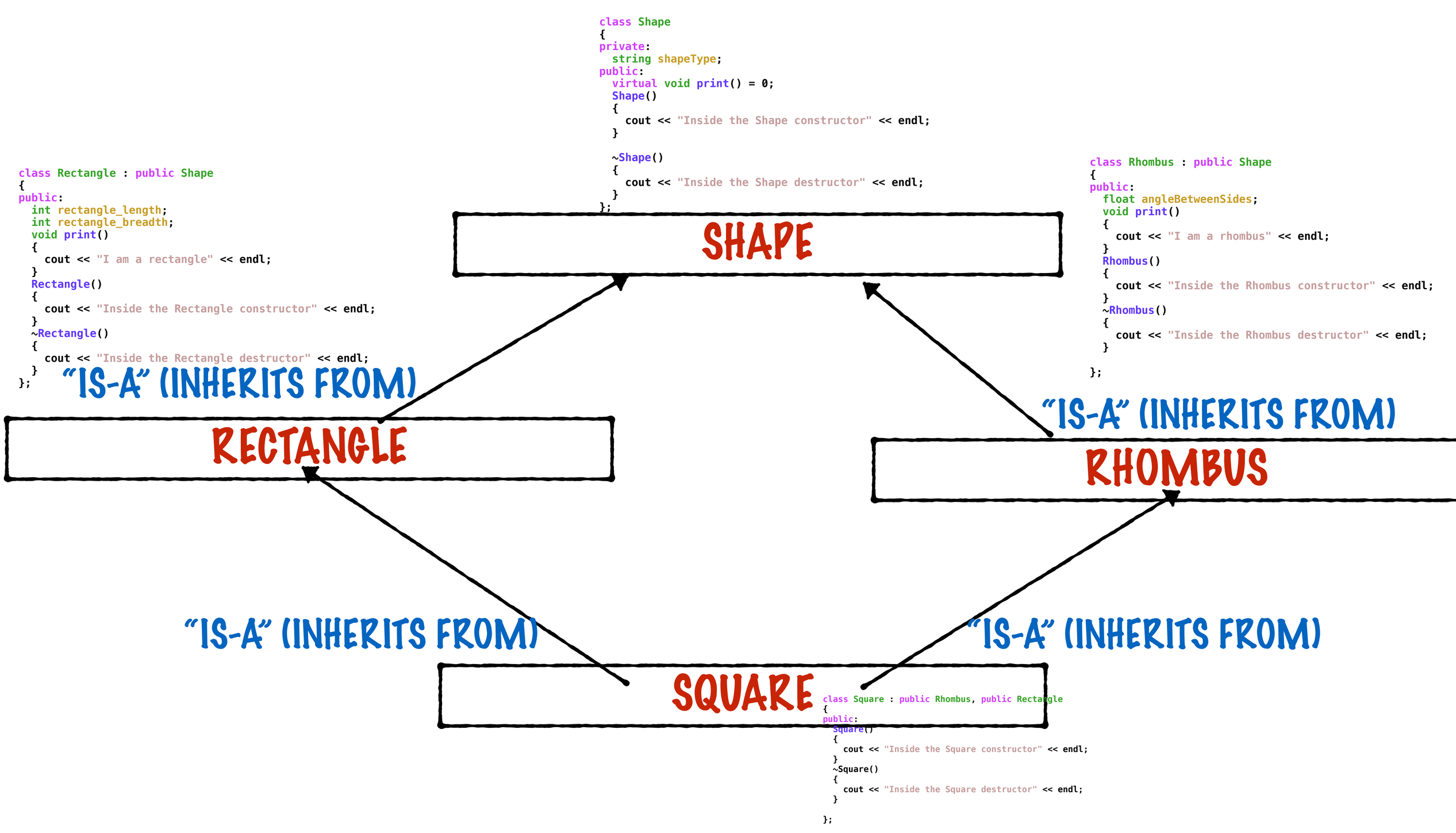

class Shape
{
private:
string shapeType;
public:
virtual void print() = 0;
Shape()
{
cout << "Inside the Shape constructor" << endl;
}
~Shape()
{
cout << "Inside the Shape destructor" << endl;
}
};
SHAPE
class Rectangle : public Shape
{
public:
int rectangle_length;
int rectangle_breadth;
void print()
{
cout << "I am a rectangle" << endl;
}
Rectangle()
{
cout << "Inside the Rectangle constructor" << endl;
}
~Rectangle()
{
cout << "Inside the Rectangle destructor" << endl;
}
};
"IS-A" (INHERITS FROM)
RECTANGLE
class Rhombus : public Shape
{
public:
float angleBetweenSides;
void print()
{
cout << "I am a rhombus" << endl;
}
Rhombus()
{
cout << "Inside the Rhombus constructor" << endl;
}
~Rhombus()
{
cout << "Inside the Rhombus destructor" << endl;
}
};
"IS-A" (INHERITS FROM)
RHOMBUS
"IS-A" (INHERITS FROM)
"IS-A" (INHERITS FROM)
SQUARE
class Square : public Rhombus, public Rectangle
{
public:
Square()
{
cout << "Inside the Square constructor" << endl;
}
~Square()
{
cout << "Inside the Square destructor" << endl;
}
};

```
class Rectangle : public Shape
{
public:
   int rectangle_length;
   int rectangle_breadth;
   void print()
   {
      cout << "I am a rectangle" << endl;
   }
   Rectangle()
   {
      cout << "Inside the Rectangle constructor" << endl;
   }
   ~Rectangle()
   {
      cout << "Inside the Rectangle destructor" << endl;
   }
};
```


class Shape
{
private:
string shapeType;
public:
virtual void print() = 0;
Shape()
{
cout << "Inside the Shape constructor" << endl;
}
~Shape()
{
cout << "Inside the Shape destructor" << endl;
}
};
SHAPE
class Rectangle : public Shape
{
public:
int rectangle_length;
int rectangle_breadth;
void print()
{
cout << "I am a rectangle" << endl;
}
Rectangle()
{
cout << "Inside the Rectangle constructor" << endl;
}
~Rectangle()
{
cout << "Inside the Rectangle destructor" << endl;
}
};
"IS-A" (INHERITS FROM)
RECTANGLE
class Rhombus : public Shape
{
public:
float angleBetweenSides;
void print()
{
cout << "I am a rhombus" << endl;
}
Rhombus()
{
cout << "Inside the Rhombus constructor" << endl;
}
~Rhombus()
{
cout << "Inside the Rhombus destructor" << endl;
}
};
"IS-A" (INHERITS FROM)
RHOMBUS
"IS-A" (INHERITS FROM)
"IS-A" (INHERITS FROM)
SQUARE
class Square : public Rhombus, public Rectangle
{
public:
Square()
{
cout << "Inside the Square constructor" << endl;
}
~Square()
{
cout << "Inside the Square destructor" << endl;
}
};

```
class Rhombus : public Shape
{
public:
  float angleBetweenSides;
  void print()
  {
    cout << "I am a rhombus" << endl;
  }
  Rhombus()
  {
    cout << "Inside the Rhombus constructor" << endl;
  }
  ~Rhombus()
  {
    cout << "Inside the Rhombus destructor" << endl;
  }

};
```


class Shape
{
private:
string shapeType;
public:
virtual void print() = 0;
Shape()
{
cout << "Inside the Shape constructor" << endl;
}
~Shape()
{
cout << "Inside the Shape destructor" << endl;
}
};
SHAPE
class Rectangle : public Shape
{
public:
int rectangle_length;
int rectangle_breadth;
void print()
{
cout << "I am a rectangle" << endl;
}
Rectangle()
{
cout << "Inside the Rectangle constructor" << endl;
}
~Rectangle()
{
cout << "Inside the Rectangle destructor" << endl;
}
};
"IS-A" (INHERITS FROM)
RECTANGLE
class Rhombus : public Shape
{
public:
float angleBetweenSides;
void print()
{
cout << "I am a rhombus" << endl;
}
Rhombus()
{
cout << "Inside the Rhombus constructor" << endl;
}
~Rhombus()
{
cout << "Inside the Rhombus destructor" << endl;
}
};
"IS-A" (INHERITS FROM)
RHOMBUS
"IS-A" (INHERITS FROM)
"IS-A" (INHERITS FROM)
SQUARE
class Square : public Rhombus, public Rectangle
{
public:
Square()
{
cout << "Inside the Square constructor" << endl;
}
~Square()
{
cout << "Inside the Square destructor" << endl;
}
};

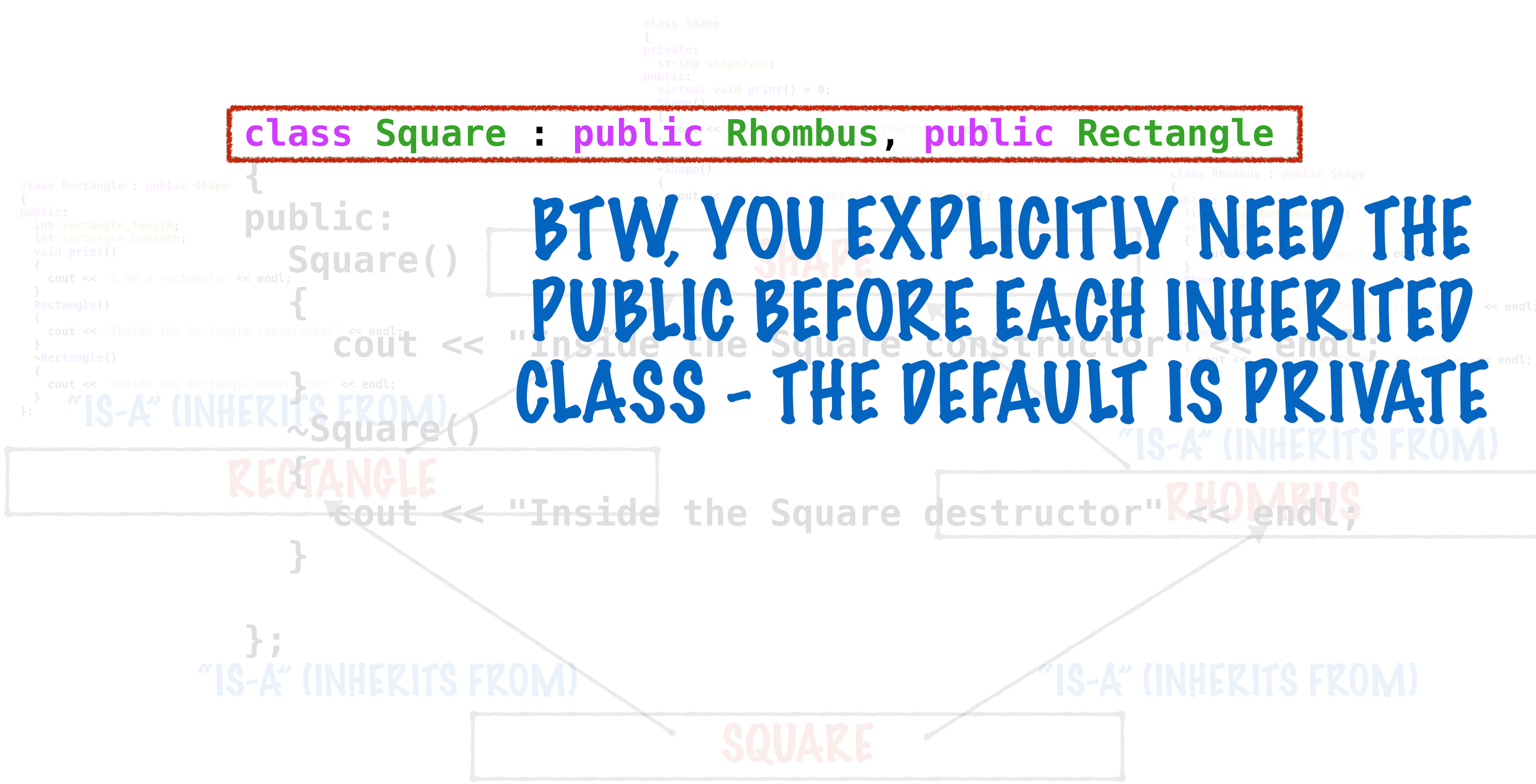
```
class Square : public Rhombus, public Rectangle
{
public:
    Square()
    {
        cout << "Inside the Square constructor" << endl;
    }
    ~Square()
    {
        cout << "Inside the Square destructor" << endl;
    }

};
```

DOES NOT IMPLEMENT THE PRINT METHOD!
NEED TO USE ONE OF THE BASE CLASS VERSIONS

```
class Square : public Rhombus, public Rectangle
```

BTW, YOU EXPLICITLY NEED THE PUBLIC BEFORE EACH INHERITED CLASS - THE DEFAULT IS PRIVATE



```
class Square : public Rhombus, public Rectangle
{
public:
    Square()
    {
        cout << "Inside the Square constructor" << endl;
    }
    ~Square()
    {
        cout << "Inside the Square destructor" << endl;
    }

};
```

DOES NOT IMPLEMENT THE PRINT METHOD!
NEED TO USE ONE OF THE BASE CLASS VERSIONS

DOES NOT IMPLEMENT THE PRINT METHOD! NEED TO USE ONE OF THE BASE CLASS VERSIONS

```
Square s1;
s1.print();
```

```
Vitthals-MacBook-Pro:~ vitthalsrinivasan$ g++ -Wall Example55.cpp
Example55.cpp:80:6: error: member 'print' found in multiple base classes of different types
  s1.print();
     ^
Example55.cpp:46:8: note: member found by ambiguous name lookup
  void print()
       ^
Example55.cpp:28:8: note: member found by ambiguous name lookup
  void print()
       ^
1 error generated.
```

DOES NOT IMPLEMENT THE PRINT METHOD!
NEED TO USE ONE OF THE BASE CLASS VERSIONS

```
Square s1;
s1.Rectangle::print();
s1.Rhombus::print();
```

NOW IF YOU WANT TO CALL A METHOD OF THE SHAPE CLASS, YOU HAVE
EXPLICITLY SPECIFY WHICH OF THE 2 VERSIONS YOU ARE REFERRING TO

DO SO USING THE SCOPE RESOLUTION OPERATOR.

DOES NOT IMPLEMENT THE PRINT METHOD!
NEED TO USE ONE OF THE BASE CLASS VERSIONS

```
Square s1;
s1.Rectangle::print();
s1.Rhombus::print();
```

RECTANGLE VERSION OF PRINT

NOW IF YOU WANT TO USE A METHOD IN A BASE CLASS, YOU HAVE
EXPLICITLY SPECIFY WHICH OF THE 2 VERSIONS YOU ARE REFERRING TO

DO SO USING THE SCOPE RESOLUTION OPERATOR.

DOES NOT IMPLEMENT THE PRINT METHOD!
NEED TO USE ONE OF THE BASE CLASS VERSIONS

```
Square s1;
s1.Rectangle::print();
```

```
s1.Rhombus::print();
```

RHOMBUS VERSION OF PRINT

NOW IF YOU WANT TO CALL A METHOD FROM A BASE CLASS, YOU HAVE
EXPLICITLY SPECIFY WHICH OF THE 2 VERSIONS YOU ARE REFERRING TO

DO SO USING THE SCOPE RESOLUTION OPERATOR.