

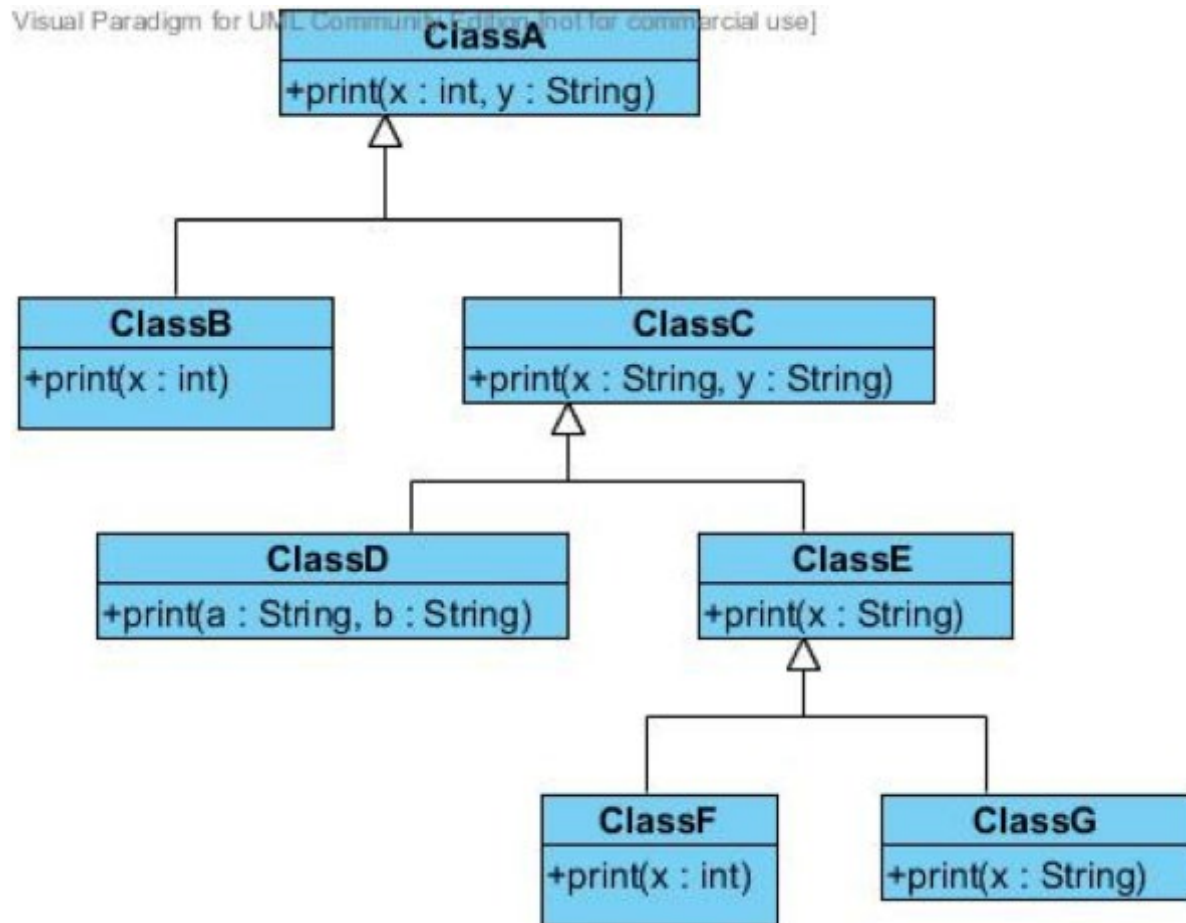
# TUTORIAL 5

---

Inheritance & Polymorphism

# Q1. Inheritance

**Q1** Given the following class hierarchy diagram in the figure





superclass



subclass

# Q1 Given the following class hierarchy diagram in the figure

- Assume that `ClassF z = new ClassF();`
- Which class' `print()` method will be used for each of the message below :

(a) `z.print(9)`

- **Class F**

(b) `z.print(2,"Cx2002")`

- **Class A**

(c) `z.print("Object")`

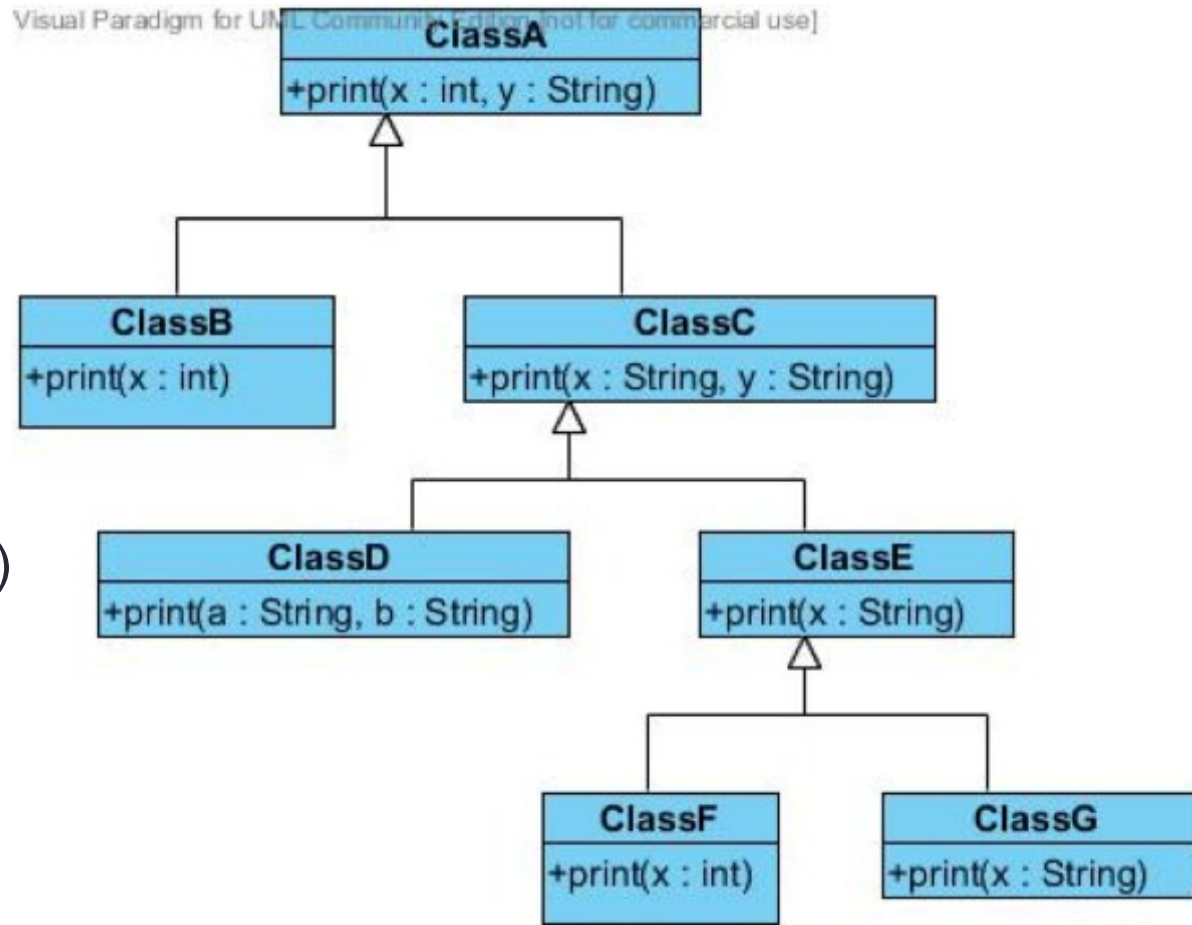
- **Class E**

(d) `z.print("OODP", "Java")`

- **Class C**

(e) `z.print("OODP", 2002)`

- **Compile Error!**



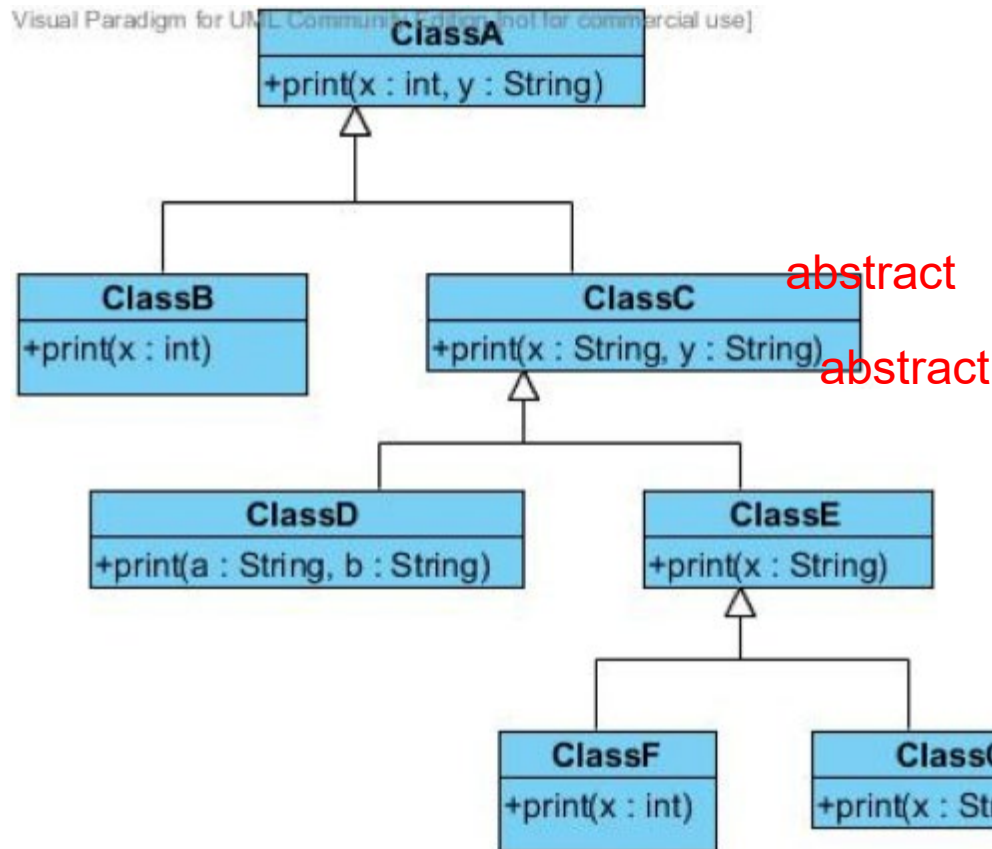
## Q2. Inheritance & Polymorphism

abstract, upcasting, downcasting

## Q2

Using Figure 1, and assuming all print methods just print out the contents of the its parameter values, answer the following

Q2(a) if the method `print(String, String)` in class `ClassC` is declared as abstract, describe what will happen and how to resolve it.

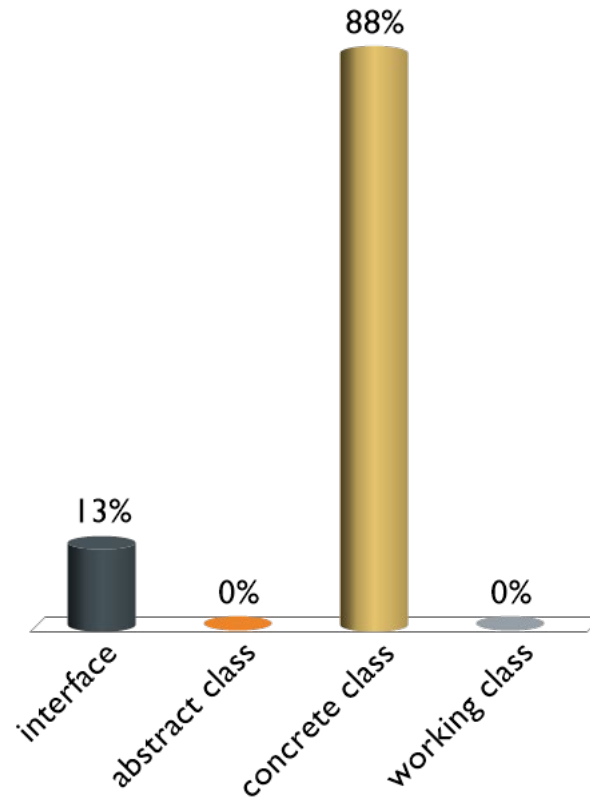




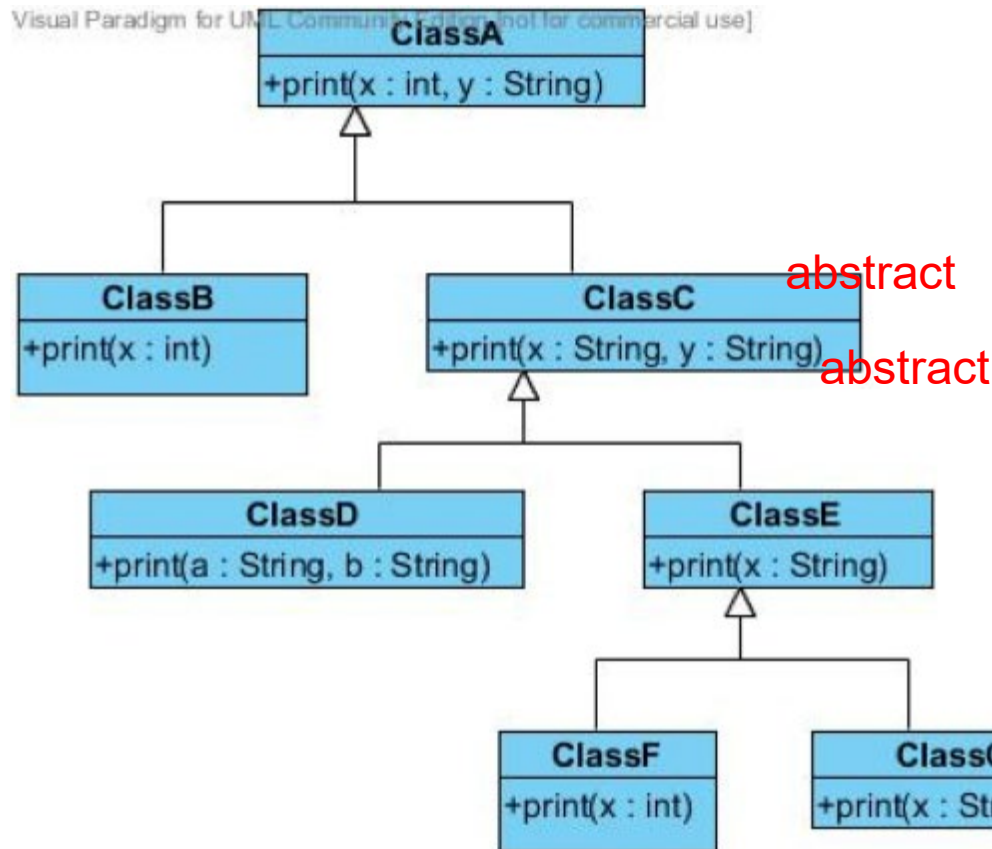
# Abstract method and class

When all methods are implemented in a class, this class is known as

1. interface
2. abstract class
- 😊 3. concrete class
4. working class

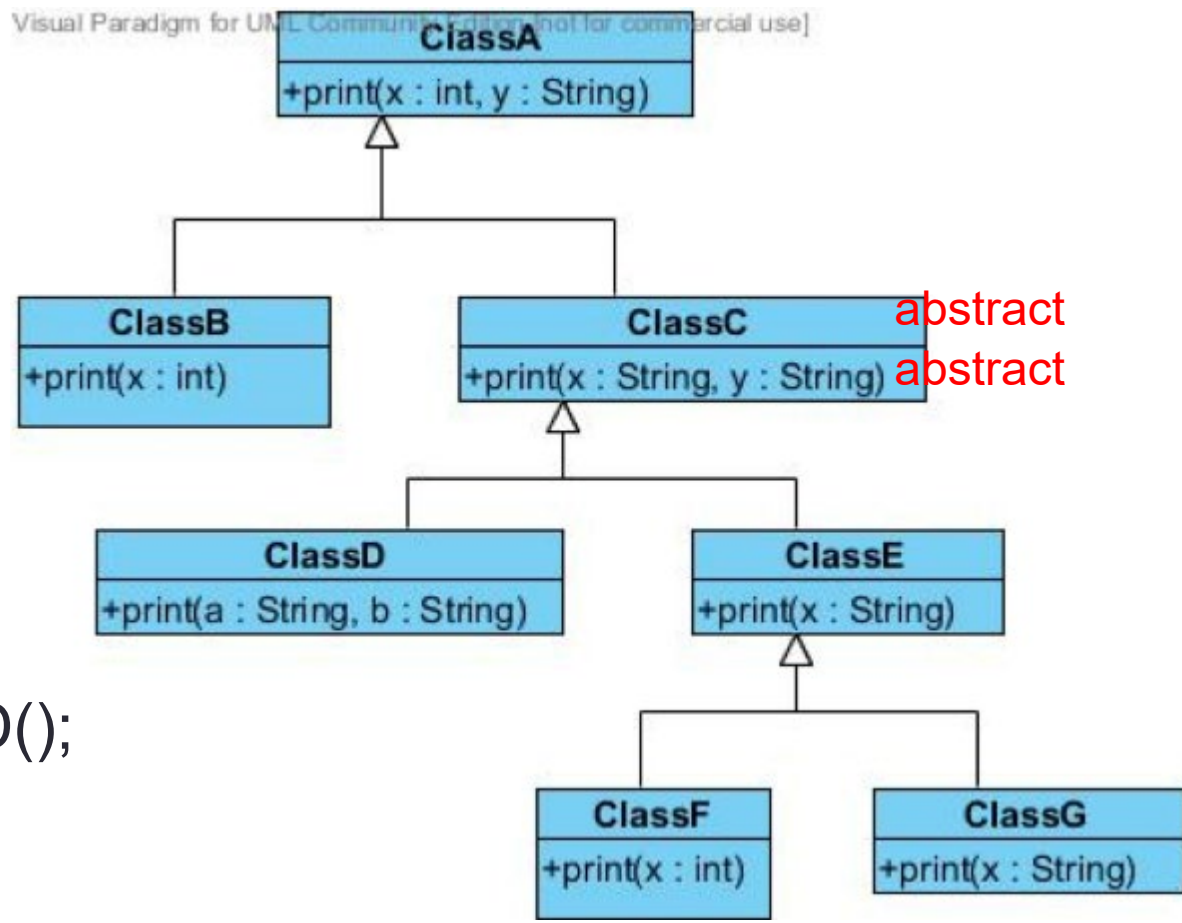


Q2(a) if the method `print(String, String)` in class `ClassC` is declared as abstract, describe what will happen and how to resolve it.



- If **ClassE** implements the abstract method `print(String,String)`, then all solved.
- Else **ClassE** needs to be declared as abstract (pass it on...) and **F** and **G** need to implement the abstract method

• (b) After resolving (a), what will be the outcome of the following codes :



• `ClassC c = new ClassD();`  
  
`c.print("hello","there");`

Polymorphism  
type casting

declared type    object type

**Class\_NameA** Object\_Variable\_Name = new *Class\_NameB*()

Class\_NameA:

*Class\_NameB*

<reference type>

object type

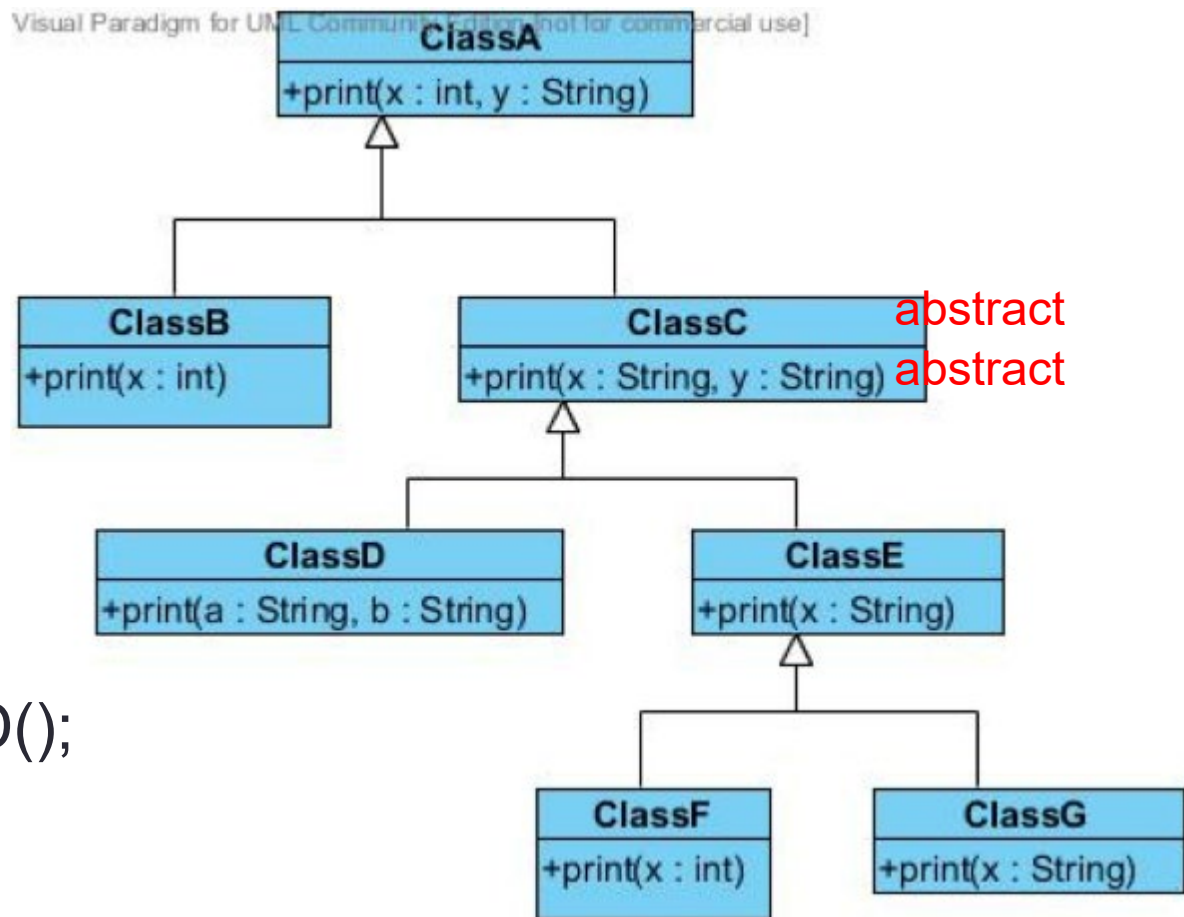
<declared type>

<variable type>

Example:

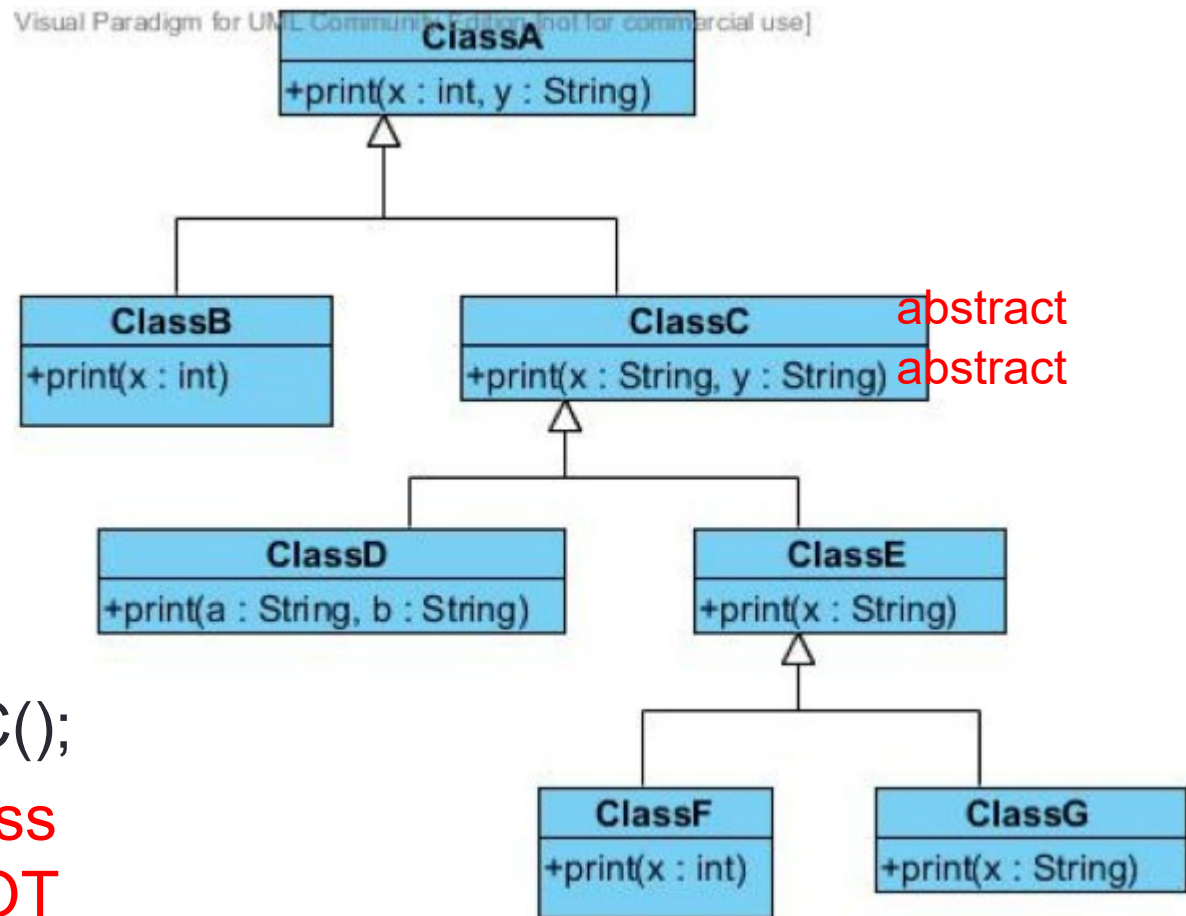
**Animal** a1 = new *Cat*();

- (b) After resolving (a), what will be the outcome of the following codes :



- `ClassC c = new ClassD();`  
`//upcast OK,`  
`c.print("hello", "there");`  
`// using ClassD method`

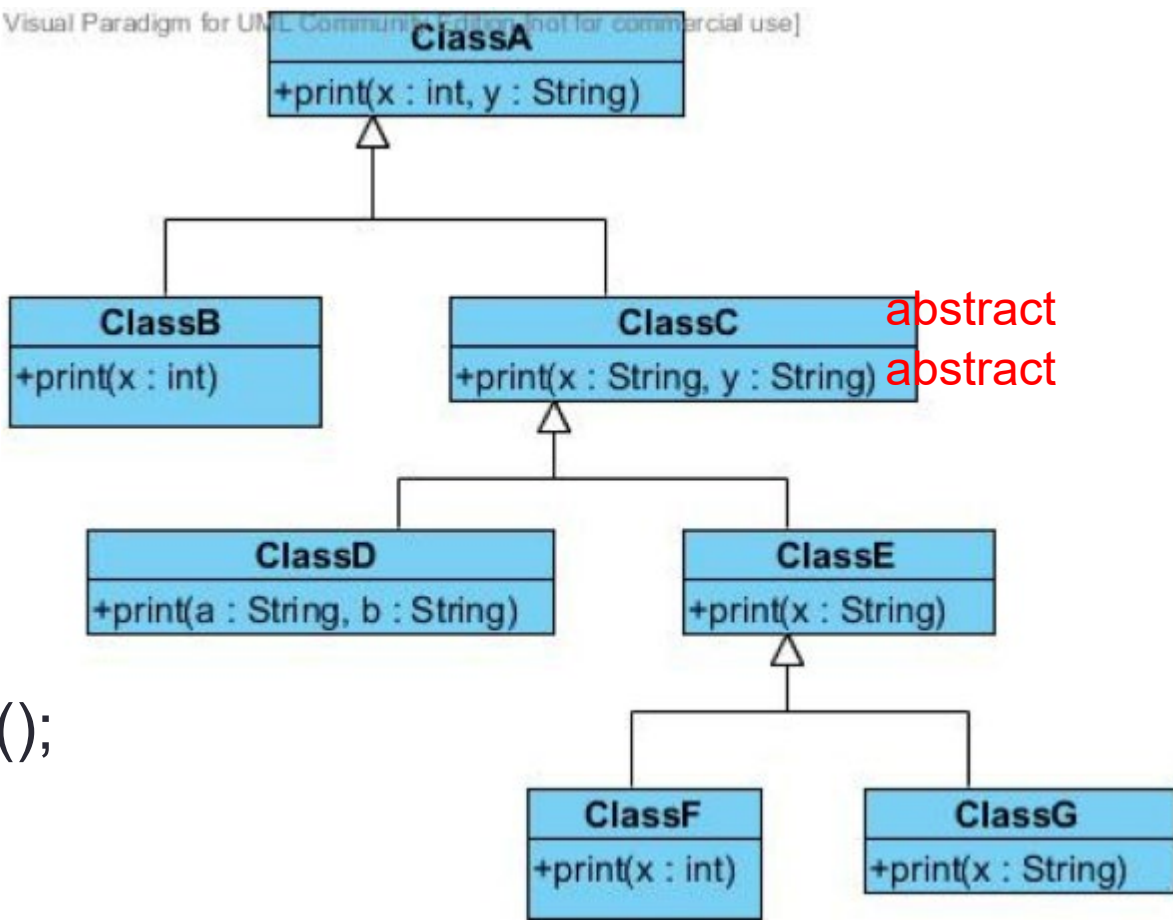
- (b) After resolving (a), what will be the outcome of the following codes :



- `ClassA a = new ClassC();`  
`// ClassC is abstract class`  
`// abstract class CANNOT`  
`// instantiate obj`  
`a.print(1,"there");`

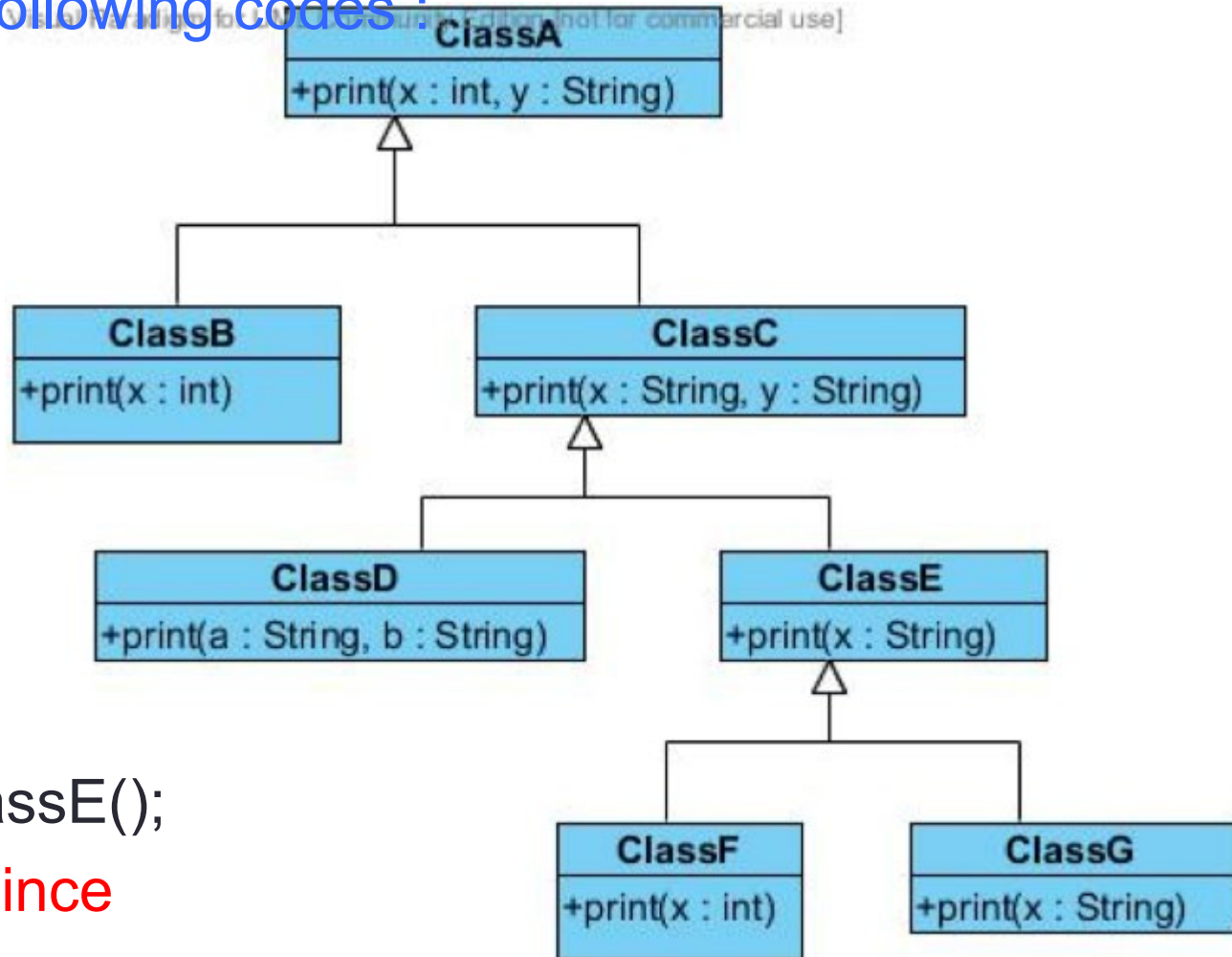


- (b) After resolving (a), what will be the outcome of the following codes :



- ClassA a = new ClassF();  
//upcast OK,  
a.print("hello", "there");  
// compile error since ClassA  
// has no print(string,string) method

- (c) Assume all classes are concrete classes, what will be the outcome of the following codes :



- ClassB b = new ClassE();  
// Error : not upcast since  
// different family line  
b.print("hello");

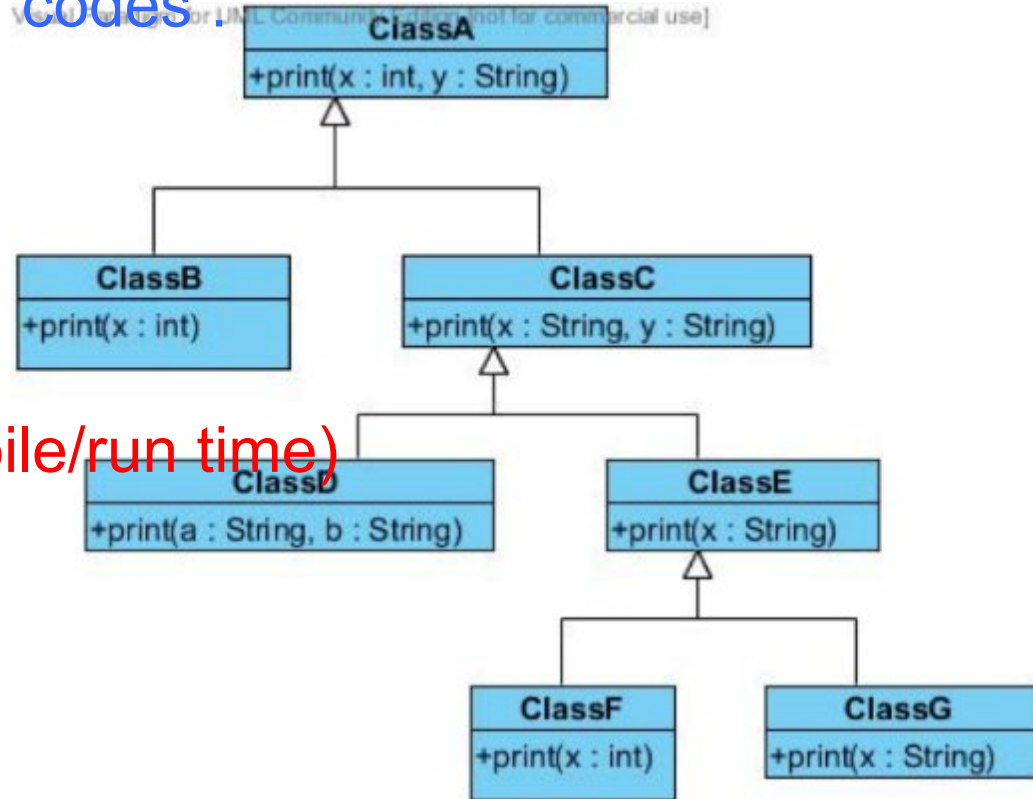
- (c) Assume all classes are concrete classes, what will be the outcome of the following codes :

• `ClassA a = new ClassF();`

`// upcast ok (compile ok)`

`a.print(12, "there");`

`// use ClassA method (compile/run time)`



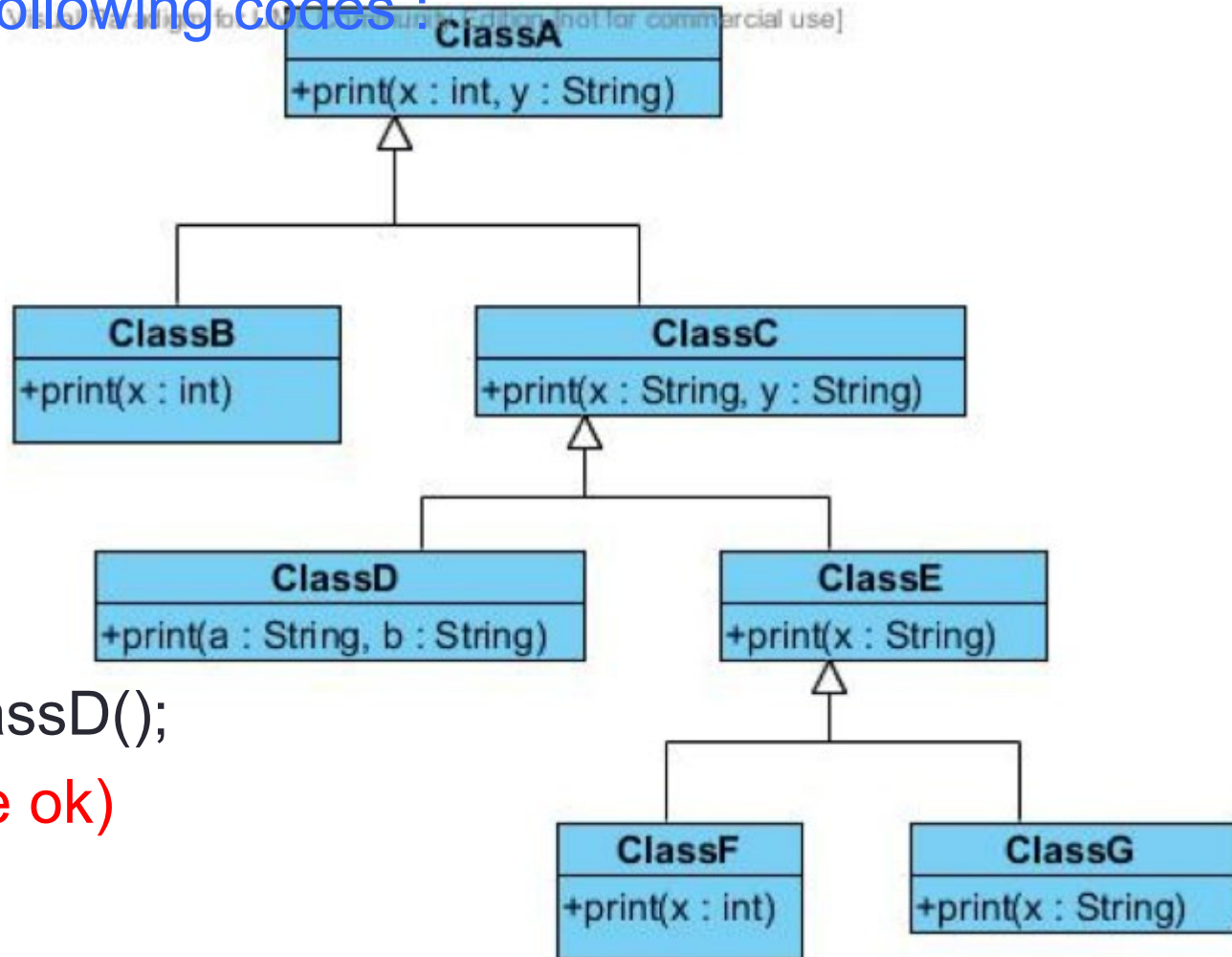
• `ClassA a = new ClassF();`

`// upcast ok (compile ok)`

`a.print(88);`

`//compile error, ClassA doesn't have print(int) method`

- (c) Assume all classes are concrete classes, what will be the outcome of the following codes :



- ClassC c = new ClassD();  
// upcast ok (compile ok)  
ClassE e = c;  
// compile Error:  
// explicitly downcast  
// ClassE e=(ClassE)c;

# Polymorphism Downcasting



# What is the output of the following program?

```
class A {  
    A()  
    System.out.println("A");  
}  
  
class C {  
    C()  
    System.out.println("C");  
}  
  
class E extends C {  
    E()  
    System.out.println("E");  
}  
  
class G extends E {  
    G()  
    System.out.println("G");  
}  
  
public class Test {  
    public static void main(String[] args) {  
        A a = new A();  
        C c = new C();  
        E e = new E();  
        G g = new G();  
    }  
}
```

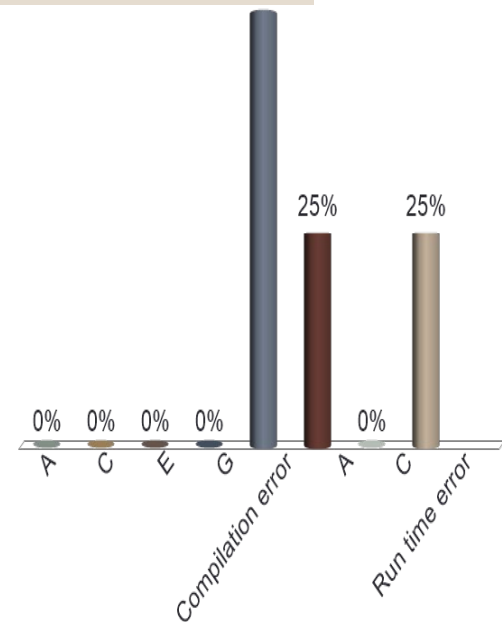
Exception in thread "main"

java.lang.ClassCastException: downcasting.C

cannot be cast to downcasting.G

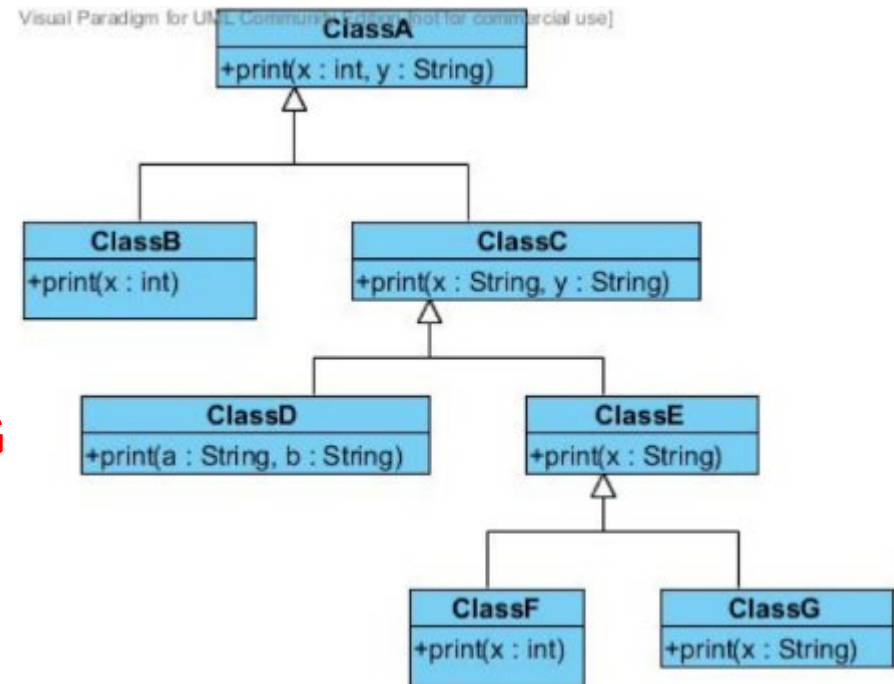
at downcasting.Test.main(Test.java:6)

- A. A
- B. C
- C. E
- D. G
- E. Compilation error
- ✓ F. A  
C  
Run time error



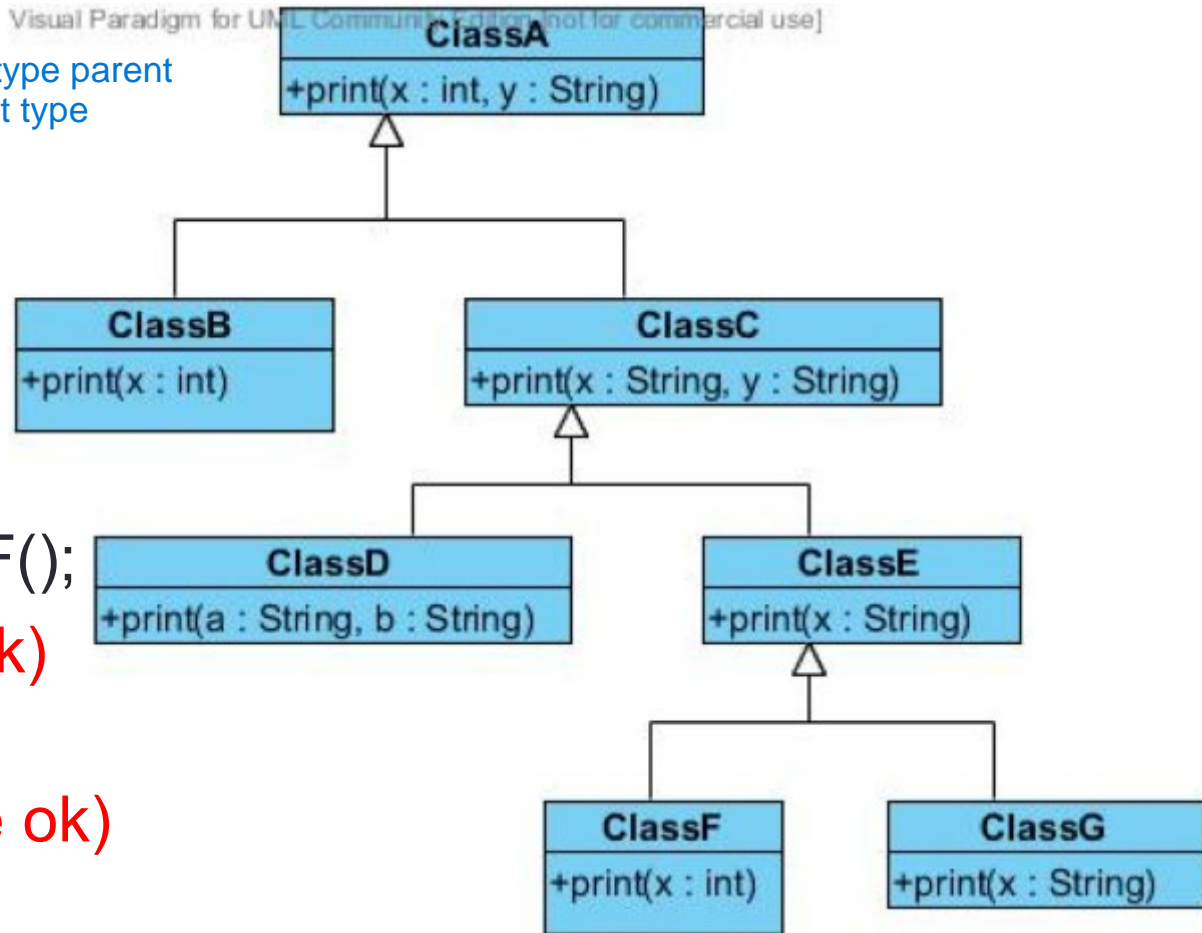
- `ClassA a = new ClassC();`  
// upcast ok (compile ok)  
`ClassG g = (ClassG)a;`  
// downcast ok (compile ok)  
// RuntimeError : Object is ClassC  
// cannot be cast to downcasting.G

`g.print("hello");`  
//compile ok





Visual Paradigm for UML Community Edition [not for commercial use]



valid if downcast type parent  
of previous upcast type

downcasting

runtime error

upcasting

compile time error

use methods from upcast classes only

```
• ClassA a = new ClassF();  
// upcast ok, (compile ok)  
ClassC f = (ClassC)a;  
// downcast ok (compile ok)  
f.print(88,"there");
```

// compile ok, inherit from ClassA. Runtime ok since a is  
ClassF object and ClassF to ClassC is upcasting

## Q3. Benefits of Polymorphism

- Q3** • Given the Java code for a Polygon class. Two subclasses, Rectangle and Triangle, are derived from Polygon class

```
public class Polygon {  
    public enum KindofPolygon{POLY_PLAIN,POLY_RECT,POLY_TRIANG};  
    protected String name;  
    protected float width, height;  
    protected KindofPolygon polytype;  
    public Polygon(String theName,float theWidth,float theHeight){  
        name = theName;  
        width = theWidth;  
        height = theHeight;  
        polytype = KindofPolygon.POLY_PLAIN; }  
    public KindofPolygon getPolytype() {  
        return polytype; }  
    public void setPolytype(KindofPolygon value) {  
        polytype = value; }  
    public String getName() { return name; }  
    public float calArea() { return 0; }  
    public void printWidthHeight( ) {  
        System.out.println("Width = "+width+" Height = "+height); }  
}
```

(i) Write the code for the Rectangle and Triangle subclass

```
public class Rectangle extends Polygon {  
    public Rectangle(String theName, float theWidth, float theHeight)  
        super(theName, theWidth, theHeight) ;  
        this.polytype = KindofPolygon.POLY_RECT;  
    }  
    public float calArea() { return width * height; }  
}
```

```
public class Triangle extends Polygon {  
    public Triangle(String theName, float theWidth, float theHeight {  
        super(theName, theWidth, theHeight) ;  
        this.polytype = KindofPolygon.POLY_TRIANG;  
    }  
    public float calArea() { return 0.5f * width * height; }  
}
```

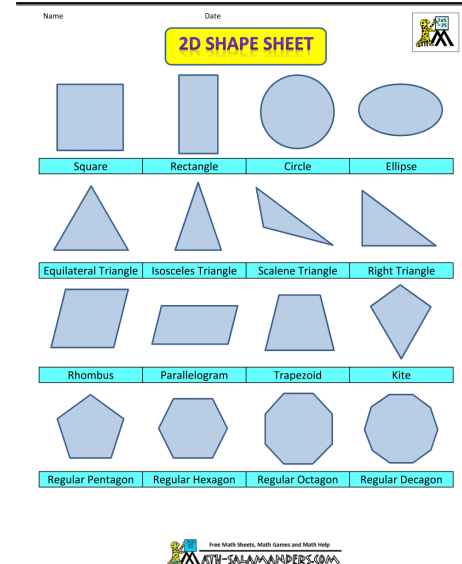
(ii) Write a TestPolygon class to have a overloaded method **printArea(...)** which will calculate and printout the area of the polygon type passed as argument, ie **printArea(Rectangle r)** and **printArea(Triangle t)**.

```
public class TestPolygon {  
    public static void printArea(Rectangle rect) {  
        float area = rect.calArea( );  
        System.out.println("The area of the Rectangle"  
        + " is " + area);  
    }  
    public static void printArea(Triangle tri) {  
        float area = tri.calArea( );  
        System.out.println("The area of the Triangle"  
        + " is " + area);  
    }  
}
```

```
//the method for comparing the areas of two Rectangles
public static boolean equalArea(Rectangle fig1, Rectangle fig2){
return fig1.calArea() == fig2.calArea();
}

//the method for comparing the areas of a Rectangle and a Circle
public static boolean equalArea(Rectangle fig1, Circle fig2){
return fig1.calArea() == fig2.calArea();
}

//the method for comparing the areas of two Circles
public static boolean equalArea(Circle fig1, Circle fig2){
return fig1.calArea() == fig2.calArea();
}
```



**256 combinations**

**Permutations**

(iii) Write the main( ) function to demonstrate **static binding** of all printArea methods.[Hints : have overloaded printArea methods for each Polygon subclass].

What is the impact on the program when a new subclass of Polygon is introduced?

```
public static void main(String[] args ) {  
    Rectangle rect1 = new Rectangle("Rect1", 3.0f, 4.0f);  
    printArea(rect1); // static binding  
    rect1.printWidthHeight();  
  
    Triangle trianl1= new Triangle("Trianl1", 3.0f, 4.0f);  
    printArea(trianl1); // static binding  
    trianl1.printWidthHeight();  
}
```

(iv) Repeat part (ii) for **dynamic binding** of `printArea( )`.

- [Hints : have a single `printArea` method, regardless of which `Polygon` subclass].

```
public class TestPolygon {
    public static void printArea(Polygon poly) {
        float area = poly.calArea( );
        System.out.println("The area of the "+
            poly.getPolytype() +" is "+area);
    }
    public static void main(String[] args ) {
        Rectangle rect1 = new Rectangle("Rect1", 3.0f, 4.0f);
        printArea(rect1);
        rect1.printWidthHeight();

        Triangle triangl= new Triangle("Triangl", 3.0f, 4.0f);
        printArea(triangl);
        triangl.printWidthHeight(); }
}
```



```
public class FigureAbstractWhyGood {  
    public static void main( String[] args )  
    {  
        Figure f1 = new Rectangle( "Red" , 10 , 10 );  
        Figure f2 = new Rectangle( "YELLOW" , 8 , 7 );  
        Figure f3 = new Circle( "Orange" , 7.8 );  
        Figure f4 = new Circle( "BLUE" , 6.2);  
    }  
}
```

```
System.out.println("The two Figures f1 and f2 have the same area? " +  
    equalArea(f1,f2));
```

```
...  
}
```

One general method only

```
//the method for comparing the areas of the two figures  
public static boolean equalArea(Figure fig1, Figure fig2){  
    return fig1.findArea() == fig2.findArea();  
}
```

```
}  
}
```

(v) Modify the **Polygon** code so that any of its subclasses **must** include a **calArea()** member method. Suggest reason(s) why this requirement would be appropriate in this case.

- Make the calArea method an abstract method and make Polygon class an abstract class. ie,  
    public abstract class Polygon {  
        ....  
        Public abstract float calArea();  
    }
- This will 'enforce' all derived classes of Polygon to implement its own calArea.
- It is not appropriate for Polygon class to have a default implementation as different polygons have different formula used for calculating area. Enforcing the implementation of the method ensures that calArea will be implemented appropriately for all Polygon subclasses.