E0-245: ASP

Lecture 6: OOPs Inheritance and Polymorphism

Dipanjan Gope

Questions from Lecture 5

- Static function: is an object instantiated?
- Can there be multiple main in same package?

Static function call: object instantiated?

```
1
2  public class TutorialMain
3  {
4
5      /**
6       * @param args
7       */
8      public static void main(String args[]){
9          //Rectangle r = new Rectangle();
10         Rectangle.example();
11         System.out.println("Done");
12     }
13 }
```

```
public class Rectangle
{
    public Rectangle() { v = new Vector<Float>(100000000);}

    public int getNumNodes() { return numNodes; }
    public static void example() {System.out.println("Static"); }

    public int numNodes=4;
    private Vector<Float> v;
}
```

javaw.exe	Dipanjan Gope	00	0:00:00	15,520 K	8,904 K	Java(TM) Platform SE binary

```
public class TutorialMain
{

    /**
     * @param args
     */
    public static void main(String args[]){
        Rectangle r = new Rectangle();
        Rectangle.example();
        System.out.println("Done");
    }
}
```

```
public class Rectangle
{
    public Rectangle() { v = new Vector<Float>(100000000);}

    public int getNumNodes() { return numNodes; }
    public static void example() {System.out.println("Static"); }

    public int numNodes=4;
    private Vector<Float> v;
}
```

javaw.exe	Dipanjan Gope	00	0:00:00	408,440 K	401,800 K	Java(TM) Platform SE binary

Can there be multiple mains in package?

- Yes, the mainfest.mf file will resolve entry point

```java
public class TutorialMain
{

    /**
     * @param args
     */
    public static void main(String args[]){
            main(122);
            main('f');
            main("hello java");
        }

        public static void main(int i){
            System.out.println("Overloaded main()"+i);
        }

        public static void main(char i){
            System.out.println("Overloaded main()"+i);
        }

        public static void main(String str){
            System.out.println("Overloaded main()"+str);
        }
}
```

Function Overloading

```java
public class Rectangle
{
    public Rectangle() {;}

    public int getNumNodes() { return numNodes; }
    public int numNodes=4;

    public static void main(String[] args)
    {
        System.out.println("Rectangle");
    }

}
```

Why is main a static?

- Can call method without instantiation of class (making the object)

Other Questions

Question: What if the main method is declared as private?
Question: What if the static modifier is removed from the signature of the main method?
Question: What if I write static public void instead of public static void?
Question: What if I do not provide the String array as the argument to the method?

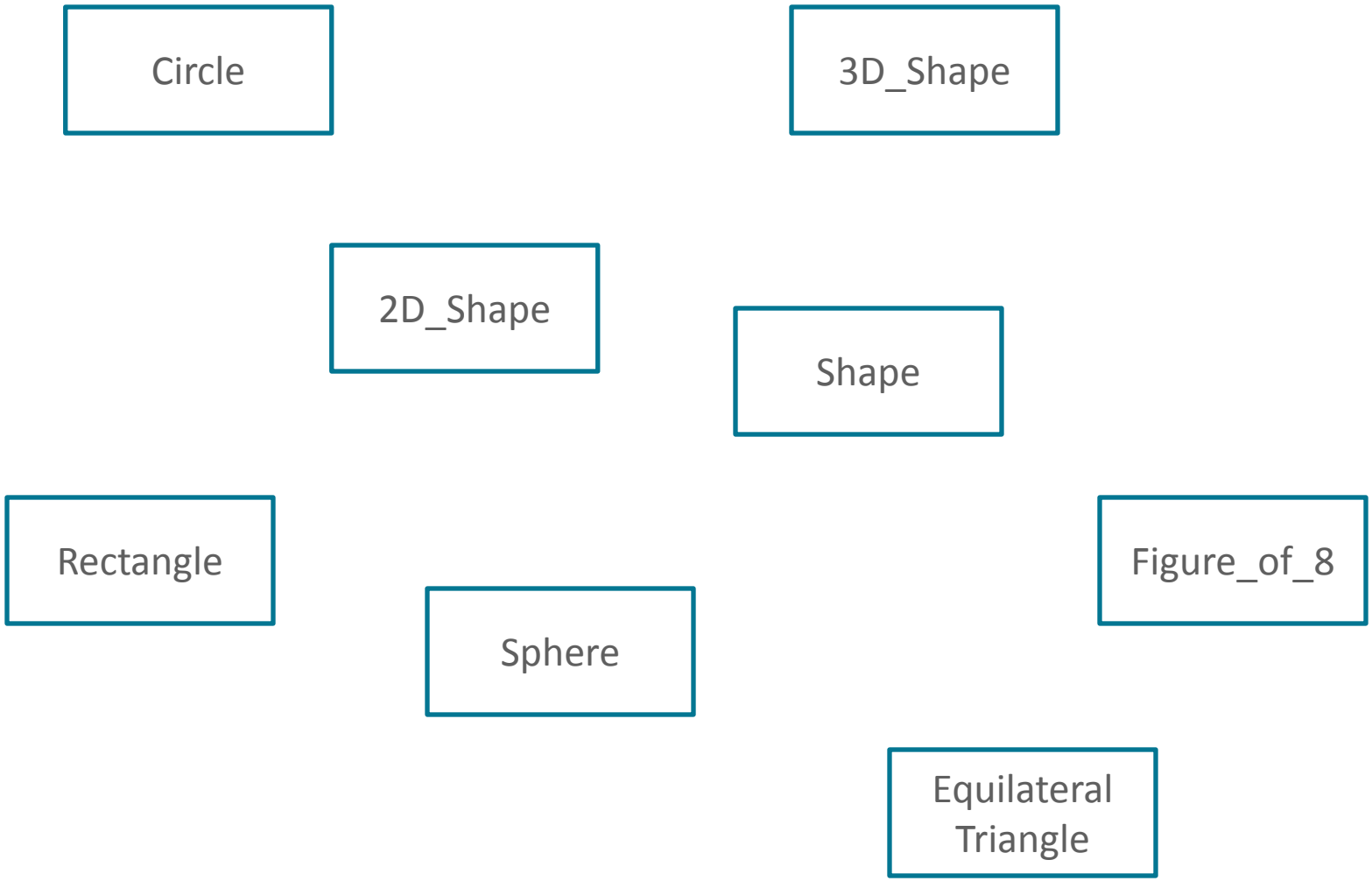
http://www.allapplabs.com/interview_questions/java_interview_questions_2.htm#q1

Module 1: OOPs and DS

JAVA and C++ mainly: (others C# and Objective-C)

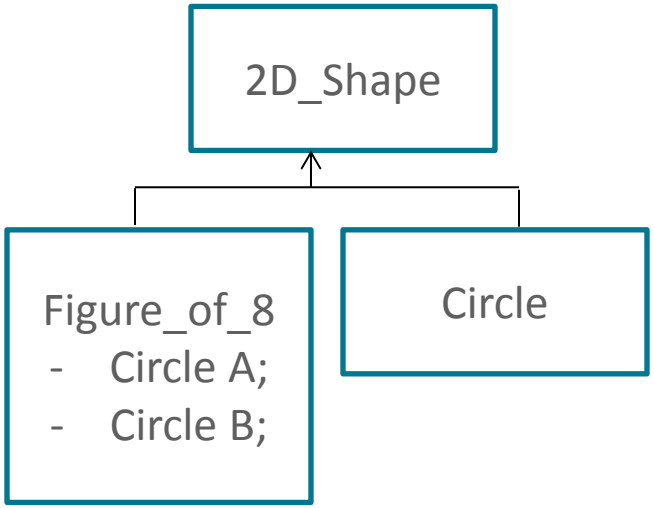
- Object oriented programing: Classes and objects
- Inheritance, polymorphism, abstract class, const
- Templates and generics
- Data structures
- Standard library, JCF, STL
- Complexity analysis
- Multithreading and synchronization
- Good programming styles

Inherited Class: Geometry Tool-box



Inherited vs Member class (common)

- IS-A vs HAS-A



Base Class

Derived Class

Inheritance Notation

```cpp
class Shape
{
public:
    Shape::Shape() {;}
    Shape::~Shape() {;}

protected:

};
```

```cpp
class TwoD_shape : public Shape
{
public:
    TwoD_shape::TwoD_shape() {;}
    TwoD_shape::~TwoD_shape() {;}

protected:

};
```

```cpp
class iRectangle : public TwoD_shape
{
public:
    iRectangle::iRectangle() {;}
    iRectangle::~iRectangle() {;}

protected:
};
```

C++

```java
public class Shape {

    public Shape()
    {
        System.out.println("Inside constructor of Shape");
    }
}
```

```java
public class TwoD_Shape extends Shape {

    public TwoD_Shape()
    {
        System.out.println("Inside constructor of TwoD Shape");
    }

}
```

```java
public class iRectangle extends TwoD_Shape{

    public iRectangle()
    {
        System.out.println("Inside constructor of iRectangle");
    }
}
```

JAVA

Special Inheritance properties

Access	public	protected	private
Same class	yes	yes	yes
Derived classes	yes	yes	no
Outside classes	yes	no	no

C++

Type of Inheritance:

When deriving a class from a base class, the base class may be inherited through **public, protected** or **private** inheritance. The type of inheritance is specified by the access-specifier as explained above.

We hardly use **protected** or **private** inheritance, but **public** inheritance is commonly used. While using different type of inheritance, following rules are applied:

- **Public Inheritance:** When deriving a class from a **public** base class, **public** members of the base class become **public** members of the derived class and **protected** members of the base class become **protected** members of the derived class. A base class's **private** members are never accessible directly from a derived class, but can be accessed through calls to the **public** and **protected** members of the base class.
- **Protected Inheritance:** When deriving from a **protected** base class, **public** and **protected** members of the base class become **protected** members of the derived class.
- **Private Inheritance:** When deriving from a **private** base class, **public** and **protected** members of the base class become **private** members of the derived class.

C++

http://www.tutorialspoint.com/cplusplus/cpp_inheritance.htm

JAVA does not allow multiple inheritance

Virtual Functions (C++)

```
class Shape
{
public:
    Shape::Shape()
    {
        std::cout << "Inside constructor of Shape" << std::endl;
    }
    Shape::~Shape() {;}

public:
    virtual void typeOfShape()
    {
        std::cout << "General Shape" << std::endl;;
    }

protected:

};

class iRectangle : public TwoD_shape
{
public:
    iRectangle::iRectangle()
    {
        std::cout << "Inside constructor of iRectangle" << std::endl;;
    }
    iRectangle::~iRectangle() {;}

public:
    virtual void typeOfShape()
    {
        std::cout << "Rectangular Shape" << std::endl;
    }

protected:
};
```

```
class TwoD_shape : public Shape
{
public:
    TwoD_shape::TwoD_shape()
    {
        std::cout << "Inside constructor of TwoD shape" << std::endl;
    }
    TwoD_shape::~TwoD_shape() {;}

public:
    virtual void typeOfShape()
    {
        std::cout << "2D Shape" << std::endl;;
    }

otected:
```

POLYMORPHISM

By default methods in JAVA are virtual functions

Virtual Functions (JAVA)

```
class Shape
{
public:
    Shape::Shape()
    {
        std::cout << "Inside constructor of Shape" << std::endl;
    }
    Shape::~Shape() {;}

public:
    virtual void typeOfShape()
    {
        std::cout << "General Shape" << std::endl;;
    }

protected:

};
```

```
public class TwoD_Shape extends Shape {

    public TwoD_Shape()
    {
        System.out.println("Inside constructor of TwoD Shape");
    }

    public void typeOfShape()
    {
        System.out.println("2D Shape");
    }
}
```

```
public class iRectangle extends TwoD_Shape{

    public iRectangle()
    {
        System.out.println("Inside constructor of iRectangle");
    }

    public void typeOfShape()
    {
        System.out.println("Rectangular Shape");
    }
}
```

JAVA final keyword:
- Variable
- Method
- Class

Pure Virtual Functions (C++)

- virtual void typeOfShape() = 0;
- Shape s; // compile error

- Shape: Abstract class or an interface

Interface in JAVA

- Contains methods (without body) and final static variables (constants)
- Cannot instantiate an interface

- public interface transformCoordinates

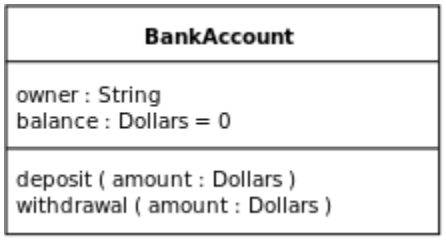
```
{
        void rotate(double degree);
        void translate(Vector<Double> direction);
};
```

Implements and Extends

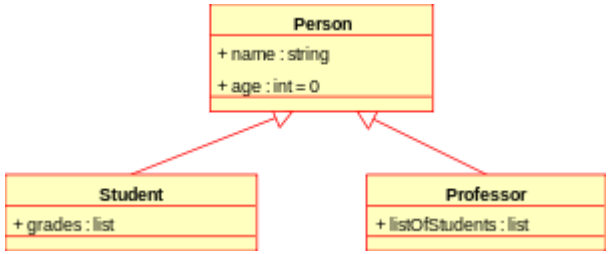
- Class can extend one class
- Class can implement multiple interfaces
- Interface can extend multiple interfaces
- Interface CANNOT implement an interface

Unified Modeling Language

http://en.wikipedia.org/wiki/Class_diagram



3 Layer Class Diagram



Inheritance

+	Public
-	Private
#	Protected
/	Derived (can be combined with one of the others)
~	Package

Conventions