NANYANG TECHNOLOGICAL UNIVERSITY

CZ/CE2002: OBJECT ORIENTED DESIGN & PROGRAMMING

Tutorials

(Partial Suggested Solutions for students)

SCHOOL OF COMPUTER ENGINEERING
NANYANG TECHNOLOGICAL UNIVERSITY

Tutorial 3 : Class Methods & Inheritance

2. The Java code is given as follows:

```
public class Point {
      protected int x, y;

      public int getX()  {return x; }
      public int getY()  {return y; }

      public Point() { x = 0; y = 0; }
      public Point(int x, int y) { this.x = x; this.y = y; }
      public void setPoint(int x, int y) { this.x = x; this.y = y; }

      public String toString(){
            return "[" + x + "," + y + "]";
      }
}

public class Circle extends Point {
      private double radius;

      public Circle() { radius = 1; }
      public Circle(double radius) { this.radius = radius;}
      // note the use of super
      public Circle(double radius, int a, int b) {
            super(a,b);
            this.radius = radius;
      }

      public double getRadius() { return radius; }
      public void setRadius(double radius) { this.radius = radius;}

      public double area() {
            return Math.PI * Math.pow(radius,2);
      }
      public String toString(){
            return "Circle of radius " + radius + " at point [" + x +
            "," + y + "]";
      }
}

public class Cylinder extends Circle {
      private double height;

      public Cylinder() { height = 1; }
      public Cylinder(double h) { height = h; }
      // note the use of super
      public Cylinder(double h, double r) {
            super(r) ;
            height = h;
      }
      public Cylinder(double h, double r, int a, int b)  {
            super(r,a,b);
            height = h;
      }

      public double getHeight() { return height; }
      public void setHeight(double height) { this.height = height; }

      // area
      public double area() {
            // note the use of super
```

```
            return 2 * (super.area()+
                  Math.PI*super.getRadius()* height);
      }
      //volume
      public double volume() { return super.area() * height; }

      public String toString() {
            return "Cylinder of height " + height + ", radius " +
            getRadius() + " at point [" + x + "," + y + "]";
      }
}

public class Test{
      public static void main(String args[]){
      Circle testCircle1 = new Circle();
      Circle testCircle2 = new Circle(35, 2, 4);
      Cylinder testCylinder1 = new Cylinder();
      Cylinder testCylinder2 = new Cylinder(8, 20, 3, 3);

      System.out.println("Area of Circle 1 = " + testCircle1.area());

      System.out.println("Area of Circle 2 = " + testCircle2.area());

      System.out.println("Surface Area of Cylinder 1 = " +
                  testCylinder1.area());

      System.out.println("Surface Area of Cylinder 2 = " +
                  testCylinder2.area());

      System.out.println("Volume of Cylinder 1 = " +
            testCylinder1.volume());

      System.out.println("Volume of Cylinder 2 = " +
            testCylinder2.volume());
      }
}

Alternative will be to use Object Composition (has-a relationship)
rather than inheritance (is-a relationship)

public class Circle {
      protected Point centre = new Point();
      protected radius;
       ……
}

Students will revisit this again in Lab 4
```

Tutorial 5 : Inheritance & Polymorphism

Given the following class hierarchy diagram in Figure 1:

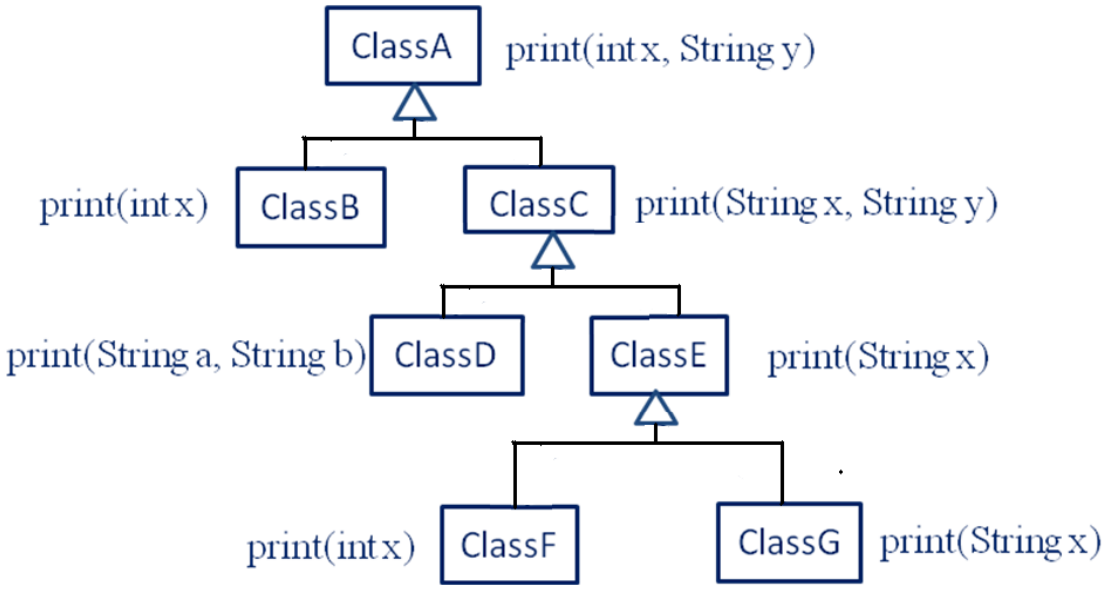

Figure 1

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

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

   Sol:

   ClassC need to be declared as abstract class.
   If ClassE implements the abstract method `print(String,String)` then all solved.
   Else ClassE need to be declared as abstract (pass it on…) and ClassF and ClassG need to implement abstract method.

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

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

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

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

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

(i)

```
ClassC  c = new ClassD(); // upcast ok (compile ok)
ClassE e = c;     //  compile  Error  :  downcast  need  to
//explicitly cast ie ClassE e = (ClassE)c;
```

(ii)

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

(iii)

```
ClassA  a = new ClassF();// upcast ok (compile ok)
a.print(12, "there");  // use ClassA method (compile ok)
a.print(88);                  // compile error, ClassA does
//not have print(int) method.
```

(iv)

```
ClassA  a = new ClassC();     // upcast ok  (compile ok)
*ClassG  g  = (ClassG)a;      // downcast ok (compile ok)
// *RuntimeError : Object is ClassC – downcasting!
// a(ClassC) not an instanceOf ClassG
g.print("hello");        // compile ok
```

(v)

```
ClassA  a = new ClassC();     // upcast ok  (compile ok)
*ClassG  g  = (ClassG)a;      // downcast ok (compile ok)
// *RuntimeError : Object is ClassC – downcasting!
// a(ClassC) not an instanceOf ClassG
g.print("hello","there");            // compile ok
```

(vi)

```
ClassA  a = new ClassF();// upcast ok (compile, runtime ok)
ClassC f = (ClassC)a;// downcast ok (compile, runtime ok)
f.print(88,"there");    // compile Ok, inherit fr ClassA.
//Runtime  Ok  since  a  is  ClassF  object  and  ClassF  to
//ClassC is upcasting
```

3.

(i)

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

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

// (iii)
      public static void main(String[] args ) {

            Rectangle rect1 = new Rectangle("Rect1", 3.0f, 4.0f);
            printArea(rect1);  // static binding
            rect1.printWidthHeight();

            Triangle triang1= new Triangle("Triang1", 3.0f, 4.0f);
            printArea(triang1); // static binding
            triang1.printWidthHeight();
      }
}

(iv)

public class TestPolygon {
                                    // dynamic binding
      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 triang1= new Triangle("Triang1", 3.0f, 4.0f);
            printArea(triang1);
            triang1.printWidthHeight();
      }
}
```

(v)

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.