Polymorphism (cont.)

In the previous lectures we discussed Polymorphism in Java. In this lecture we will see **types of polymorphism**. There are two types of polymorphism in java:

1- **Static Polymorphism** also known as compile time polymorphism.
2- **Dynamic Polymorphism** also known as runtime polymorphism.

- Method Overloading in Java – This is an example of ***compile time(or static polymorphism)*** and it has been discussed previously.
- Method Overriding in Java – This is an example of ***run time (or dynamic polymorphism).***

Recall the Rules for Method Overriding

- The method signature i.e. method name, parameter list and return type must match exactly.
- The overridden method can widen the accessibility but not narrow it, i.e. if it is private in the base class, the child class can make it public but not vice versa.

Example 1 (Static Polymorphism):

Method overloading is one of the way java supports static polymorphism. Here we have two definitions of the same method add() which add method would be called is determined by the parameter list at the compile time. That is the reason this is also known as compile time polymorphism.

```
public class Calculator  {
   int add(int a, int b)    {    return a+b;     }
   int  add(int a, int b, int c)      {  return a+b+c;  } }

public class Demo {
  public static void main(String args[])    {
           Calculator obj = new Calculator ();
      System.out.println(obj.add(10, 20));
      System.out.println(obj.add(10, 20, 30));    }}
```

The Output will be:

```
30
60
```

Dynamic Polymorphism means the ability of an object to take on many forms. The most common use of polymorphism in OOP occurs when a parent class reference is used to refer to a child class object. It is one of the core concepts of object-oriented programming (OOP). It is important to satisfy two conditions:

1- The classes must be part of the same inheritance hierarchy.
2- The classes must support the same required methods.

Example 2 (Dynamic polymorphism):

```
public class A {
public void doIt(){   }
…….
}
 public classB extends  A {
public void doIt(){   }
…….
}
public classC extends B {
public void doIt(){   }
…….
}
public class Main {
public static void main(String [] args){
A x=new B();
x.doIt();
……
}
}
```

Given classes A, B, and C where B extends A and C extends B and where all classes implement the instance method void doIt(). A reference variable is instantiated as "A x = new B();" and then x.doIt() is executed. What version of the doIt() method is actually executed and why?

The version of the doIt() method that's executed is the one in the ***B class because of dynamic binding.***

Dynamic binding basically means that the method implementation that is actually called is determined at run-time, not at compile-time. Hence the term ***dynamic binding***. Although x is of type A, because it references an object of class B, the version of the doIt() method that will be called is the one that exists in B.

Example 3 (Dynamic Polymorphism and Dynamic binding):

```
public class ABC{
  public void myMethod(){
        System.out.println("Overridden Method");    } }

public class XYZ extends ABC{

  public void myMethod(){
        System.out.println("Overriding Method");   }}

public class Main{
public static void main(String args[]){

ABC obj = new ABC();
obj.myMethod();
// This would call the myMethod() of parent class ABC

XYZ obj = new XYZ();
obj.myMethod();
// This would call the myMethod() of child class XYZ

ABC obj = new XYZ();
obj.myMethod();
// This would call the myMethod() of child class XYZ   } }
```

Example 4 (Trace):

```
public class Animal{
  public void move(){ System.out.println("Animals can move"); } }

class Dog extends Animal{
  public void move(){    super.move(); // invokes the super class method
    System.out.println("Dogs can walk and run"); }}

public class TestDog{
  public static void main(String args[]){

Animal b = new Dog();
    b.move();//Runs the method in Dog class   }}
```

This would produce following result:

```
Animals can move
Dogs can walk and run
```

Example5 (Writing a program):

Define a super class called Shape that has the following members:

String name;
Shape(String aName)
String getName()
float calculateArea()

Derive two classes Circle and Square from class Shape. The class Circle has the following members:

int radius;
Circle(String aName)
float calculateArea()

While the Square class has the following members:

int side;
Square(String aName) ;
public float calculateArea() ;

Write a Main class to define two objects (one circle and one square) based on one reference of type shape and print their areas.

```
public class Shape {
private String name;
public Shape(String aName) {name=aName;}
public String getName() {return name;}
public float calculateArea () {return 0.0f;}
}
public class Circle extends Shape {
private int radius;
public Circle(String aName) {
super(aName);
radius = 3;}
public float calculateArea() {
float area;
area = (float) (3.14 * radius * radius);
return area; }
}
public class Square extends Shape {
private int side;
public Square(String aName) {
super(aName);
side = 5; }
public float calculateArea() {
int area;
area = side * side;
return area; } }
```

```
public class Main{
public static void main(String argv[]) {
Shape obj = new Circle(”Circle C”);
System.out.println("The area of " + obj.getName() + " is " + obj.calculateArea()+"
sq. cm.\n");
obj = new Square(”Square S”);
System.out.println("The area of " + obj.getName() + " is " + obj.calculateArea()+"
sq. cm.\n");
} }
```

The output will be:

```
The area of Circle C is 28.26 sq. cm.
The area of Square S is 25.0 sq. cm.
```

Recall the previous example and re-write the Main class to define an array of shape type that has two elements one is circle object and the other is square object …then print their areas. **(Example 6):**

```
public class Shape {
private String name;
public Shape(String aName) {name=aName;}
public String getName() {return name;}
public float calculateArea () {return 0.0f;}
}
public class Circle extends Shape {
private int radius;
public Circle(String aName) {
super(aName);
radius = 3;}
public float calculateArea() {
float area;
area = (float) (3.14 * radius * radius);
return area; }
}
public class Square extends Shape {
private int side;
public Square(String aName) {
super(aName);
side = 3; }
public float calculateArea() {
int area;
area = side * side;
return area; } }
```

```
public class Main{
public static void main(String argv[]) {
Circle c = new Circle(”Circle C”);
Square s = new Square(”Square S”);
Shape shapeArray[] = {c, s};
for (int i=0; i<shapeArray.length; i++) {

System.out.println("The area of " + shapeArray[i].getName() + " is " +
shapeArray[i].calculateArea()+" sq. cm.\n");

}}}
```

Two objects, a circle and a square, are created by the first two statements of and iterating through the array, the area of the respective object is produced and main(). Object variables for the circle and square are kept in the array, shapeArray

While the actual routine for the choice of shape in the array elements has to be pre-determined via a switch statement in static binding, switch statement is not required with dynamic binding. Based on the output from the code, it is clear that the appropriate method for responding to the choice in shapeArray has been used:

```
The area of Circle C is 28.26 sq. cm.
The area of Square S is 9 sq. cm.
```

The method has been selected based on the class of the shape referenced in shapeArray at run-time. This is only possible in programming languages that support dynamic binding. With dynamic binding, the variable shapeArray[i] is bound to an object method at run time when the class definition of the shape referenced is known.

The advantage of polymorphism

1- Simplicity

If you need to write code that deals with a family of types, the code can ignore type- specific details and just interact with the base type of the family. Even though the code thinks it is using an object of the base class, the object's class could actually be the base class or any one of its subclasses This makes your code easier for you to write and easier for others to understand.

2- Extensibility

Other subclasses could be added later to the family of types, and objects of those new subclasses would also work with the existing code this means that making Incremental Development.

Example 7 (Recall Example 5)

A new Triangle class was added. The addition of the class does not affect the other parts of the program. A statement was added in main() to create the Triangle object. A statement was added in main() to include the newly created triangle into the shapeArray.

```
class Shape {
...
public static void main(String argv[]) {
Circle c = new Circle("Circle C");
Square s = new Square("Square S");
Triangle t = new Triangle("Triangle T");
Shape shapeArray[] = {c, s, t};
for (int i=0; i<shapeArray.length; i++) {
System.out.println("The area of " + shapeArray[i].getName()
+ " is " + shapeArray[i].calculateArea()+" sq. cm.\n"); }}}

public class Triangle extends Shape {
private int base, height;
Triangle(String aName) {
super(aName);
base = 4; height = 5;}
public float calculateArea() {
float area = 0.5f * base * height;
return area;  }  }
```

Example 8 (writing a program):

Define a base class called Polygon that has:
two integers attributes represent the dimensions of any polygon and set and print methods. Derive three classes Rectangle,Traingle and Parallelgram classes from Polygon class that have the area method and override the print method of the super class. Use these classes to print the areas of the polygons: rectangle which its dimensions are 5 and 4 , triangle which its dimensions are 5 by 6 and parallegram which its dimensions are 7 by 8.

```
  public class Polygon {
    protected int d1,d2;
    protected void set(int a,int b){d1=a;d2=b;}
public void print(){System.out.println(d1+"  "+d2);}    }

public class Rectangle extends Polygon {
private int area(){return(d1*d2);}
    public void print(){
    System.out.println(area());
   System.out.println(d1+"   "+d2);    }}

public class Traingle extends Polygon {
private int area(){return(d1*d2/2);}
    public void print(){
    System.out.println(area());
     System.out.println(d1+"   "+d2); }}

public class Parallegram extends Polygon {
private int area(){return(d1*d2);}
    public void print(){
    System.out.println(area());
   System.out.println(d1+"   "+d2);}}

public class Main {
   public static void main(String[] args) {
    Rectangle p1=new Rectangle();
    p1.set(5,4);     p1.print();
   Traingle p2=new Traingle();
    p2.set(5,6);     p2.print();
    Parallegram p3=new Parallegram();
    P3.set(7,8);     p3.print();
//       P3.set(7,8);     p3.print();   =>7 8 28
}}
```

The output will be the same

```
20
5   4
15
5   6
56
7   8
```

If we change the program to:

```
   public class Polygon {
    protected int d1,d2;
    protected void set(int a,int b){d1=a;d2=b;}
    protected void print(){System.out.println(d1+"   "+d2);}   }

public class Rectangle extends Polygon {
private int area(){return(d1*d2);}
    public void print(){
    System.out.println(area());
    System.out.println(d1+"   "+d2); //super.print();
} }
public class Traingle extends Polygon {
private int area(){return(d1*d2/2);}
    public void print(){      System.out.println(area());
    System.out.println(d1+"   "+d2); //super.print(); }}

public class Parallegram extends Polygon {
private int area(){return(d1*d2);}
    public void print(){      System.out.println(area());
    System.out.println(d1+"   "+d2); //super.print(); } }

public class Main {
     public static void main(String[] args) {

    Polygon p1=new Rectangle();
    p1.set(5,4);        p1.print();

    Polygon p2=new Traingle();
    p2.set(5,6);       p2.print();
    p2=new Parallegram();
    p2.set(7,8);      p2.print();    }}
```

The output :

```
20
5   4
15
5   6
56
7   8
```

The same previous example except that the print method in polygon is deleted

```
public class Polygon {
   protected int d1,d2;
   protected void set(int a,int b){d1=a;d2=b;}   }
```

The compiler will detect an error *WHY?!!*

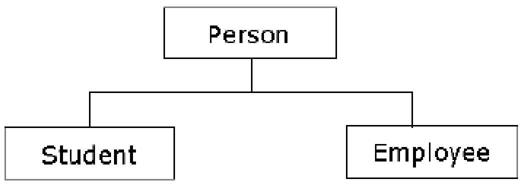
If we put super.print () instead of System.out.println(d1+"   "+d2) ; in print() method in all classes the

output will be :

```
20
5   4
15
5   6
56
7   8
```

H.W. (quize)

- Given the parent class `Person` and the child class `Student`, we add another subclass of `Person` which is `Employee`.
- Below is the class hierarchy



Define the class hierarchy and suggest all members…

```
public class Student {
public String getName(){
System.out.println("Student Name:" + name);
return name;  }  }
public class Employee {
public String getName(){
System.out.println("Employee Name:" + name);
return name;  }}
………
```

```
public static main( String[] args ) {

 Student studentObject = new Student();
 Employee employeeObject = new Employee();
 Person ref = studentObject; //Person ref. points to a
 // Student object
 // getName() method of Student class is called
 String temp= ref.getName();
 System.out.println( temp );
 ref = employeeObject; //Person ref. points to an
 // Employee object
//getName() method of Employee class is called
 String temp = ref.getName();
 System.out.println( temp );  }
```