Inheritance (cont.)

Vũ Thị Hồng Nhạn

(vthnhan@vnu.edu.vn)

Dept. of software engineering, UET

Vietnam National Univ., Hanoi

Static & dynamic binding

Static vs. dynamic binding

- Static binding refers to the execution of a program where **type of object** is determined/known **at compile time**
  - i.e., when compiler executes the code it knows the type of object (class) to which object belongs to

- In case of dynamic binding, the type of object is determined **at run time**
  - Dynamic binding is slower than static binding

Binding

- **Private, final and static modifiers** binds to the class level,
  - so methods and variables use static binding & bonded by compiler
  - while the other methods are bonded during runtime based upon runtime object

- In general, **overloaded methods** are bonded using static binding, while **overridden methods** are bonded using dynamic binding

- Static binding is better performance wise
  - Compiler knows that all such methods cannot be overridden and will always be accessed by objects of local class

example static binding

```
public class NewClass {
    public static class superclass {
        static void print()
        {
            System.out.println("print in superclass.");
        }
    }
    public static class subclass extends superclass {
        static void print()
        {
            System.out.println("print in subclass.");
        }
    }

    public static void main(String[] args)
    {
        superclass A = new superclass();
        superclass B = new subclass();
        A.print();
        B.print();
    }
}
```

Output:
print in superclass.
print in superclass.

Since the print() method of **superclass** is static, compiler knows that it will not be overridden in subclasses
→ compiler knows during compile time which print() method to call and hence no ambiguity

Example dynamic binding

```
public class NewClass {
    public static class superclass {
        void print()
        {
            System.out.println("print in superclass.");
        }
    }

    public static class subclass extends superclass {
        @Override
        void print()
        {
            System.out.println("print in subclass.");
        }
    }

    public static void main(String[] args)
    {
        superclass A = new superclass();
        superclass B = new subclass();
        A.print();
        B.print();
    }
}
```

```
Output:
print in superclass.
print in subclass.
```

- Methods are not static
- During compilation, the compiler has no idea as to which print() has to be called since compiler goes only by referencing variable not by type of object

→ the binding would be delayed to runtime and therefore the corresponding version of print() will be called based on type on object.

Remarks

- Cannot override **static** methods in java
- In a static context, cannot access a **non-static variable**
  - i.e., in a static method, can access only to static variables because non-static variables depend on the existence of an instance of a class.
  - Same for static class (nested static class)

- Final method: cannot be overridden
- Final class: cannot be inherited

Abstract classes in Java

Abstract class & method

- Goal: **hiding** the internal implementation of the features and only **showing** the functionality to the users
- There are some related classes that need **to share** some lines of code, so **these lines of codes** can be put within **abstract class**
  - *this abstract class* should be **extended** by all these **related classes**
- can include **abstract method**
  - But, abstract method must be overridden in the inherited classes

Cannot create an object of an abstract class

```
abstract class Shape {
    protected int x, y;
    Shape(int x1, int y1) {
        x = x1; y = y1;
    }
}
class Circle extends Shape
{    …
}

Shape s = new Shape(10, 10) //  compile error
Shape s = new Circle();
```

Constructor of abstract class is called

when creating an object of inherited class

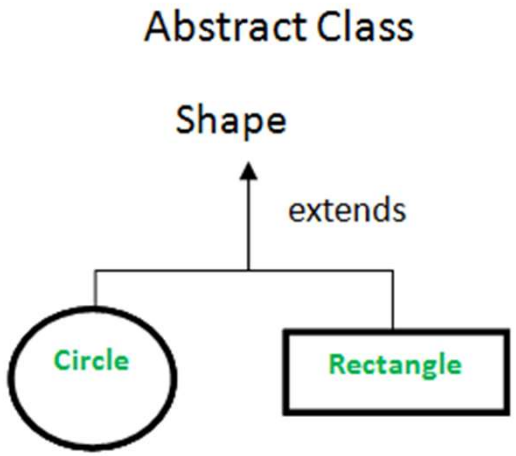
```
abstract class MyBase {
    MyBase() { System.out.println("MyBaseClass Constructor called"); }
    abstract void fun();
}
class DerivedClass extends MyBase {
    DerivedClass(int x) { System.out.println("DerivedClass Constructor called"); }
    void fun() { System.out.println("DerivedClass fun() called"); }
}

class Main {
    public static void main(String args[]) {
       DerivedClass d = new DerivedClass(3);
    }
}
```

Output:

MyBaseClass Constructor called

DerivedClass Constructor called

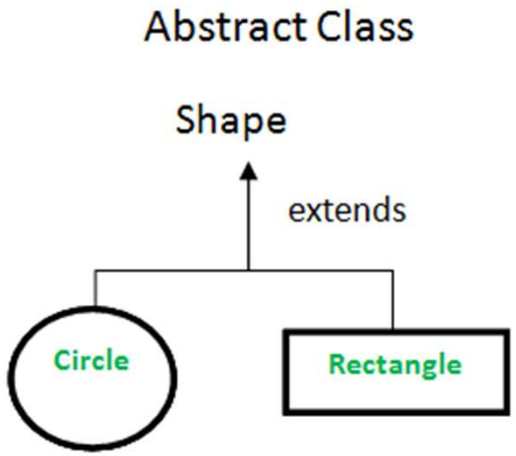
Example




```
import java.io.*;
abstract class Shape  {

    String objectName = " ";

    Shape(String name)     {
        this.objectName = name;
    }

     public void moveTo(int x, int y)     {
        System.out.println(this.objectName + " " + "has been moved to"
                                + " x = " + x + " and y = " + y);
     }

    // abstract methods which will be
    // implemented by its subclass(es)
     abstract public double area();
      abstract  public void draw();
}
```

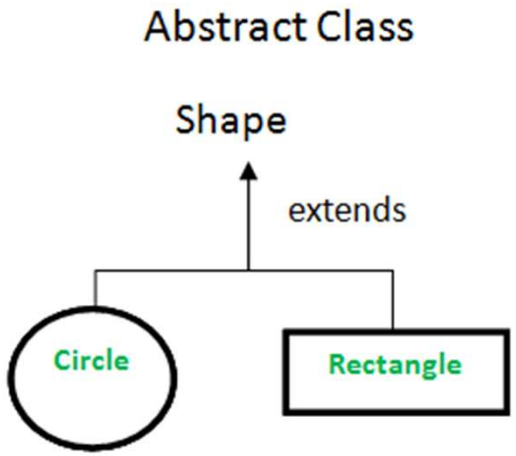
Example...

```
class Rectangle extends Shape{

    int length, width;

    // constructor
    Rectangle(int length, int width, String name) {
        super(name);
        this.length = length;
        this.width = width;
    }

    @Override
    public void draw()      {
        System.out.println("Rectangle has been drawn ");
    }

    @Override
    public double area()      {
        return (double)(length*width);
    }
}
```



Example...

```
class Circle extends Shape {
    double pi = 3.14; int radius;

    //constructor
    Circle(int radius, String name)    {
        super(name);
        this.radius = radius;
    }
    @Override
    public void draw()     {
        System.out.println("Circle has been drawn ");      }

    @Override
    public double area()      {
        return (double)((pi*radius*radius)/2);     }
}
```



Example...

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

        // creating the Object of Rectangle class
        // and using shape class reference.
        Shape rect = new Rectangle(2,3, "Rectangle");
        System.out.println("Area of rectangle: " + rect.area());
        rect.moveTo(1,2);

        System.out.println(" ");

        // creating the Objects of circle class
        Shape circle = new Circle(2, "Cicle");
        System.out.println("Area of circle: " + circle.area());
        circle.moveTo(2,4);
    }
}
```