COSC 121

Computer Programming I

Polymorphism

Part 1/2

Dr. Mostafa Mohamed

Acknowledgement: The slides mainly rely on the textbook of Y. D. Liang titled “Introduction to Java Programming, 10th Ed.”, Pearson Edu. Inc.

Previous Pre-recorded Lecture

Students' led Q/As about the previous lecture:

- Method Overriding
- Accessing class members & constructors using `super` keyword
- The `final` modifier
- Visibility Modifiers Revisited
- The `Object` Class and Its Methods

Outline

Today:

- Multiple classes in one file
- Polymorphism
  - Rule 1: reference of supertype referring to subtype
  - Rule 2: can only access class members to known to reference
  - Rule 3: dynamic binding

Next lecture: *more on polymorphism*

- Generic programming
- `instanceof` operator
- Object casting
- `Object`'s `equals` method

Before we start: a useful tip!

How to create multiple classes in a project?

- Separate .java files
  - Create separate files, a file for each class.
  - If classes will be used by several other classes in your project
  - This is what we have been doing since COSC 111
- Many classes in the same .java file:
  - Logical grouping of classes that **are mostly used within the containing class**.
  - **Option1:** Nested classes
    - define classes (inner classes) within another class (outer class).
  - **Option2:** several classes in a file
    - define **one public class** in a file, and define other classes in the same file outside the public class
      - Other classes can only use default visibility modifier.

```
public class OuterClass{
    ...
    class InnerClass {
        ...
    }
}
```

```
public class C1 {
    ...
}
class C2{
    ...
}
```

The 3rd pillar

Polymorphism

The Three Pillars of OOP

OOP
Encapsulation
Inheritance
Polymorphism

What is 'Polymorphism'?

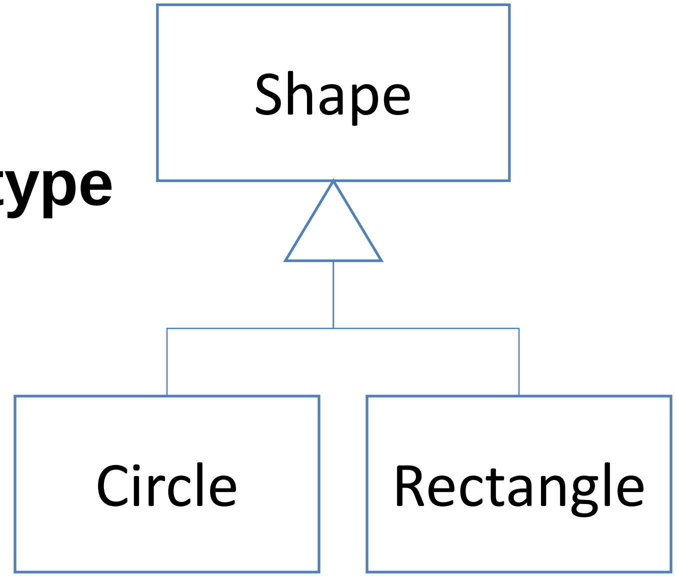
Terminology: A class defines a type. A type of a subclass is called a ***subtype***, and a type of its superclass is a ***supertype***.

- E.g., Circle is a subtype of Shape and Shape is a supertype for Circle.

Polymorphism: the ability of an object to take on "*many forms*".

In Java, a **reference variable of a supertype can refer to any of its subtype objects**.

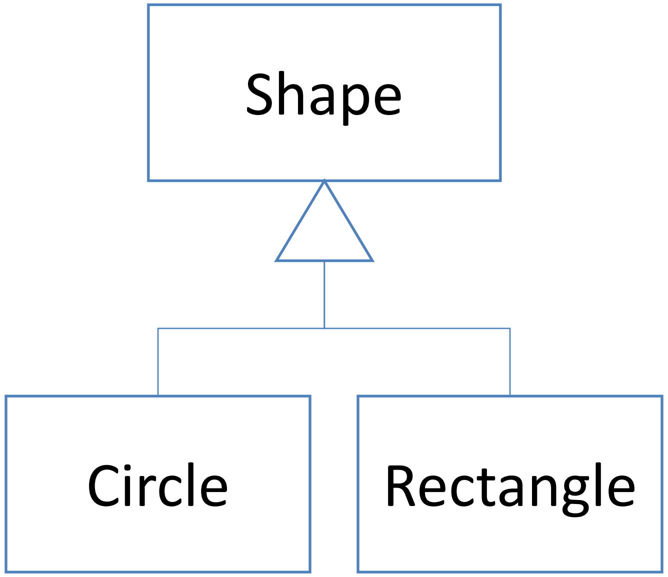
- This allows us to perform a single action (method) in different ways.
  - *more about this shortly*
- Every instance of a subclass is also an instance of its superclass, but not vice versa.
  - e.g., every circle is a shape object, but not every shape object is a circle.



Example

```
Shape s = new Shape();
System.out.println(s);

Circle c = new Circle(1.5);
System.out.println(c);

Rectangle r = new Rectangle(3.1,2.1);
System.out.println(r);
```

```
Shape s = new Shape();
System.out.println(s);

s = new Circle(1.5);
System.out.println(c);

s = new Rectangle(3.1,2.1);
System.out.println(r);
```



```java
public class Shape {
    //attributes
    private String color;
    private boolean filled;
    //constructors
    public Shape(){□
    public Shape(String color, boolean filled) {□
    //methods
    public String getColor() {□
    public void setColor(String color) {□
    public boolean isFilled() {□
    public void setFilled(boolean filled) {□
    public String toString() {□
}
```

```java
public class Circle extends Shape{
    //attributes
    private double radius;

    //constructors
    public Circle(){□
    public Circle(double radius){□
    public Circle(double radius,String color,boolean fil

    //methods
    public double getArea(){□
    public double getPerimeter(){□
    public double getDiameter(){□
    public void printCircle(){□

    public double getRadius(){□
    public void setRadius(double radius){□

    public String toString() {□
}
```

```java
public class Rectangle extends Shape{
    //attributes
    private double width, height;

    //constructors
    public Rectangle() {□
    public Rectangle(double width, double height) {□
    public Rectangle(double width,double height,String color,boolean filled){□

    //methods
    public double getArea(){□
    public double getPerimeter(){□

    public double getWidth() {□
    public void setWidth(double width) {□
    public double getHeight() {□
    public void setHeight(double height) {□

    public String toString() {□
}
```

THE THREE RULES

Rule 1: A reference of a supertype can be used to refer to an object of a subtype.(not vice versa).

Rule 2: You can only access class members known to the reference variable

Rule 3: When invoking a method using a reference variable x, the method in the object referenced by x is executed, regardless of the type of x.

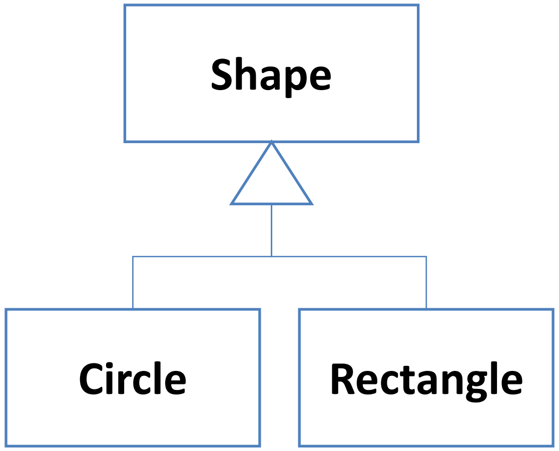
Polymorphism in Java

A reference variable of a supertype can refer to any of its subtype objects, but not vice versa.

- Example1:

All the following statements are valid:

```
Rectange r1 = new Rectangle();
Circle c1 = new Circle();

Shape s1 = new Rectangle();
Shape s2 = r1;
Shape s3 = c1;
```



RULE #1

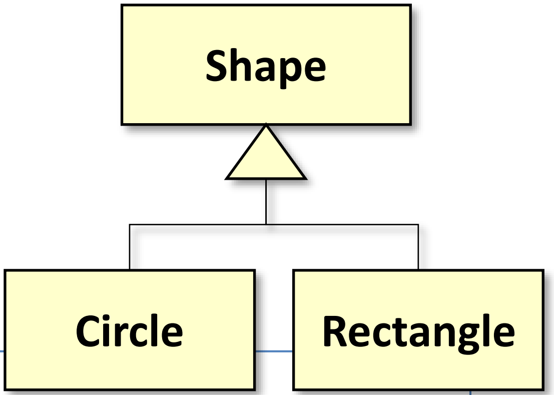
The following statements are **INVALID** :

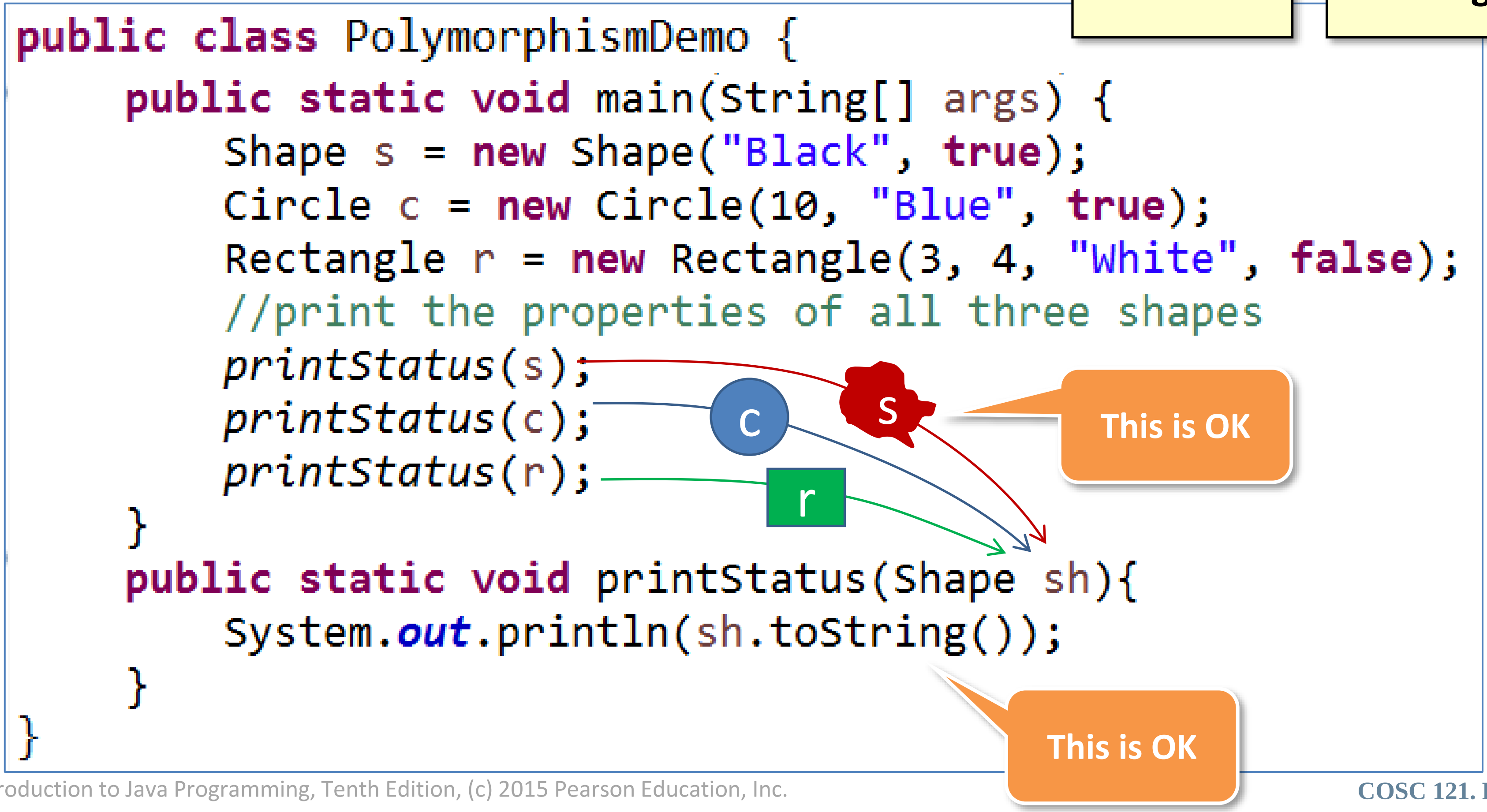
```
Rectangle r2 = new Shape();  //invalid
Rectangle r3 = new Circle(); //invalid
```

Passing References to Methods

Example2:

- we can pass an instance of a subclass to a parameter of its superclass type.



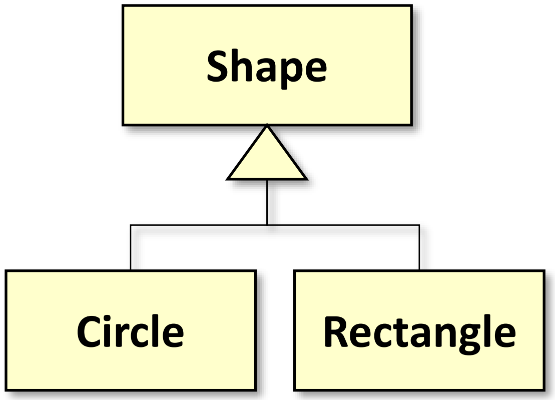


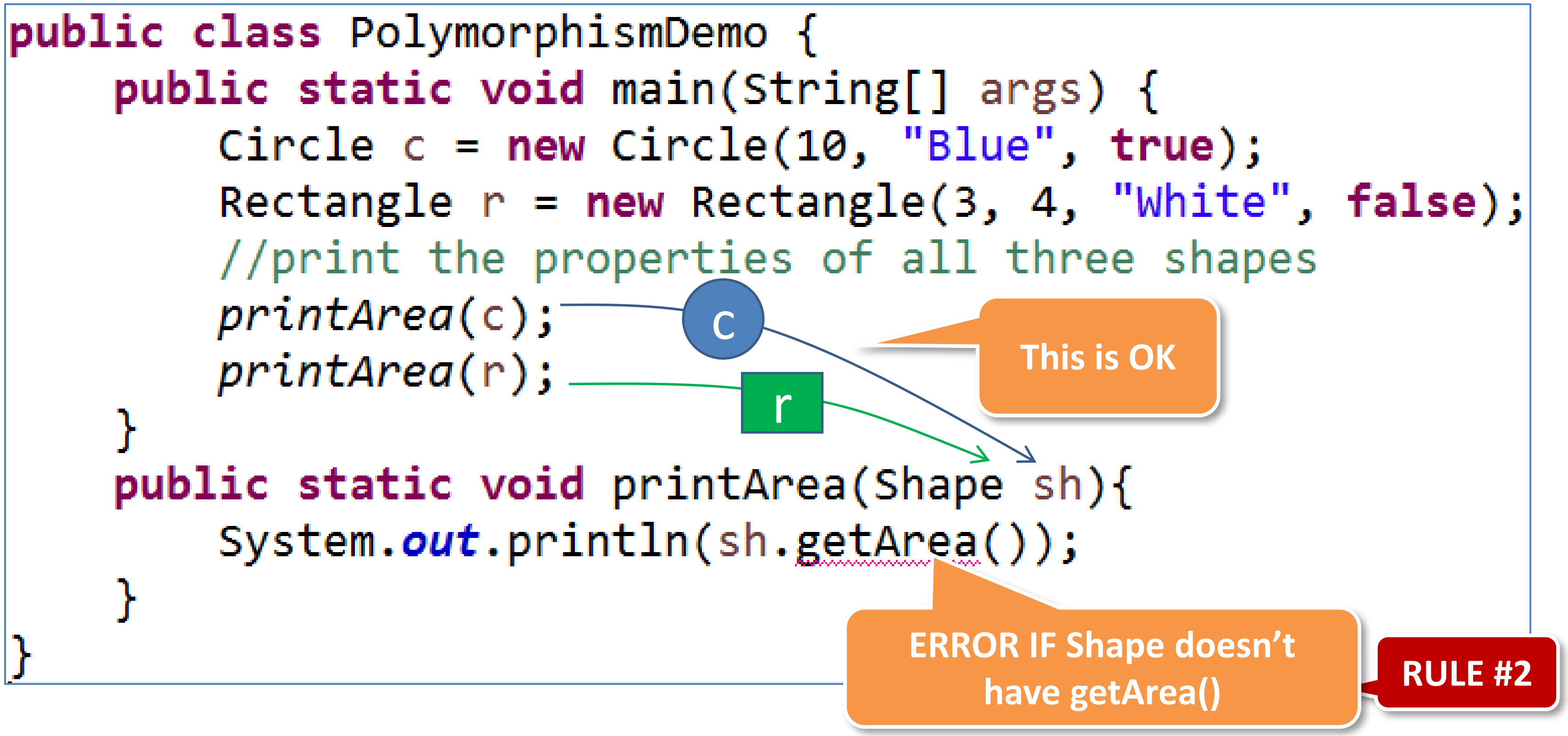
```
public class PolymorphismDemo {
    public static void main(String[] args) {
        Shape s = new Shape("Black", true);
        Circle c = new Circle(10, "Blue", true);
        Rectangle r = new Rectangle(3, 4, "White", false);
        //print the properties of all three shapes
        printStatus(s);
        printStatus(c);
        printStatus(r);
    }
    public static void printStatus(Shape sh){
        System.out.println(sh.toString());
    }
}
```

Passing References to Methods, *cont'd*

Example3:

- we can pass an instance of a subclass to a parameter of its superclass type.





```
public class PolymorphismDemo {
    public static void main(String[] args) {
        Circle c = new Circle(10, "Blue", true);
        Rectangle r = new Rectangle(3, 4, "White", false);
        //print the properties of all three shapes
        printArea(c);
        printArea(r);
    }
    public static void printArea(Shape sh){
        System.out.println(sh.getArea());
    }
}
```

THE THREE RULES

Rule 1: A reference of a supertype can be used to refer to an object of a subtype.(not vice versa).

Rule 2: You can only access class members known to the reference variable

Rule 3: When invoking a method using a reference variable x, the method in the object referenced by x is executed, regardless of the type of x.

THE THREE RULES

Rule 1: A reference of a supertype can be used to refer to an object of a subtype.(not vice versa).

Rule 2: You can only access class members known to the reference variable

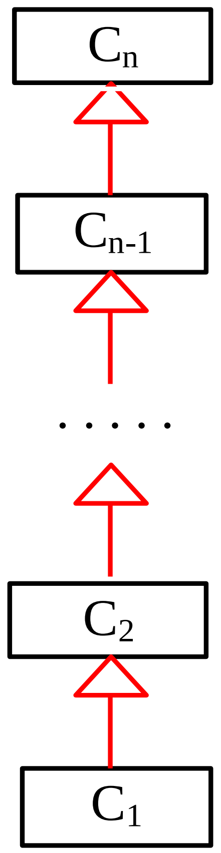
Rule 3: When invoking a method using a reference variable x, the method in the object referenced by x is executed, regardless of the type of x.

Dynamic Binding and Rule #3

Assume:

- C_1 is a subclass of C_2, C_2 is a subclass of C_3, ..., and C_{n-1} is a subclass of C_n.
- An object **obj** is an instance of C_1 (and hence it is aslo an instance of C_2, ..., C_n).

C_n is the Object class



How dynamic binding works?

- If we invoke a method **obj.p()**, the JVM searches the implementation for the method **p()** in C_1, C_2, ..., C_{n-1} and C_n in this order, until it is found. Once an implementation is found, the search stops and the first-found implementation is invoked.

RULE #3

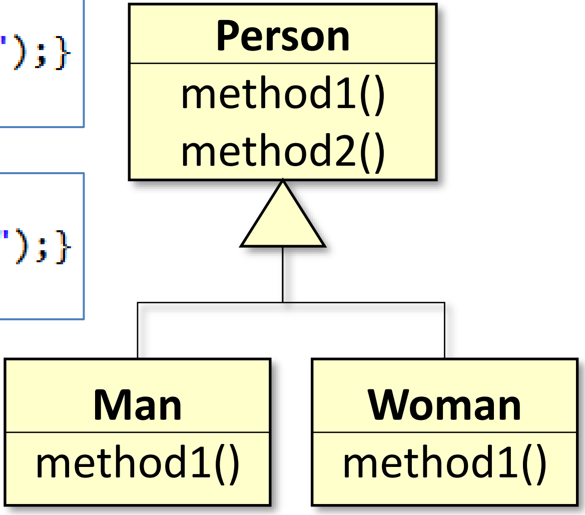
Dynamic Binding, Ex1

```java
public class Person {
    public void method1(){System.out.println("This is person 1.");}
    public void method2(){System.out.println("This is person 2.");}
}
```

```java
public class Man extends Person{
    public void method1(){System.out.println("This is a man.");}
}
```

```java
public class Woman extends Person{
    public void method1(){System.out.println("This is a woman.");}
}
```

```java
public class DynamicBindingTest {
    public static void main(String[] args) {
        Person p1 = new Person();
        Person p2 = new Man();
        Person p3 = new Woman();
        p1.method1();
        p2.method1();
        p3.method1();
        p1.method2();
        p2.method2();
        p3.method2();
    }
}
```



Output

```
This is person 1.
This is a man.
This is a woman.
This is person 2.
This is person 2.
This is person 2.
```

Dynamic Binding, Ex2

When the method m(Object x) is envoked, the argument x's toString method is invoked. x may be an instance of GradStudent, Student, Human, or Object. Classes GradStudent, Student, Human, and Object have their own implementation of the toString method. Which implementation is used will be determined dynamically by the JVM at runtime.

The method **m** takes a parameter of the Object type, which mean you can invoke it with **any object type**.

```java
public class DynamicBindingTest2 {
    public static void main(String[] args) {
        m(new GradStudent());
        m(new Student());
        m(new Human());
        m(new Object());
    }
    public static void m(Object x) {
        System.out.println(x.toString());
    }
}

class Human extends Object {
    public String toString() {return "Human";}
}

class Student extends Human {
    public String toString() {return "Student";}
}

class GradStudent extends Student {
}
```

Output

```
Student
Student
Human
java.lang.Object@5e65ab77
```