COSC 121

Computer Programming II

OOP: Inheritance

Part 2/2

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.

Outline

Previous lecture:

- Intro to inheritance

Today:

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

What can you do in a subclass?

A subclass inherits from a superclass. You can:

- **Use** inherited class members (properties and methods).
- **Add** new class members.
- Methods:
  - **Override** instance methods of the superclass
    - to modify the implementation of a method defined in the superclass
    - the method must be defined in the subclass using the same signature and the same return type as in its superclass.
  - **Hide** static methods of the superclass
    - By writing a new *static* method in the subclass that has the same signature as the one in the superclass.
- Constructors:
  - **Invoke** a superclass constructor from within a subclass constructor
    - either ***implicitly***
    - or ***explicitly*** using the keyword super

Overriding methods

Overriding Methods

Overriding allows a subclass to modify the behavior of an inherited method as needed.

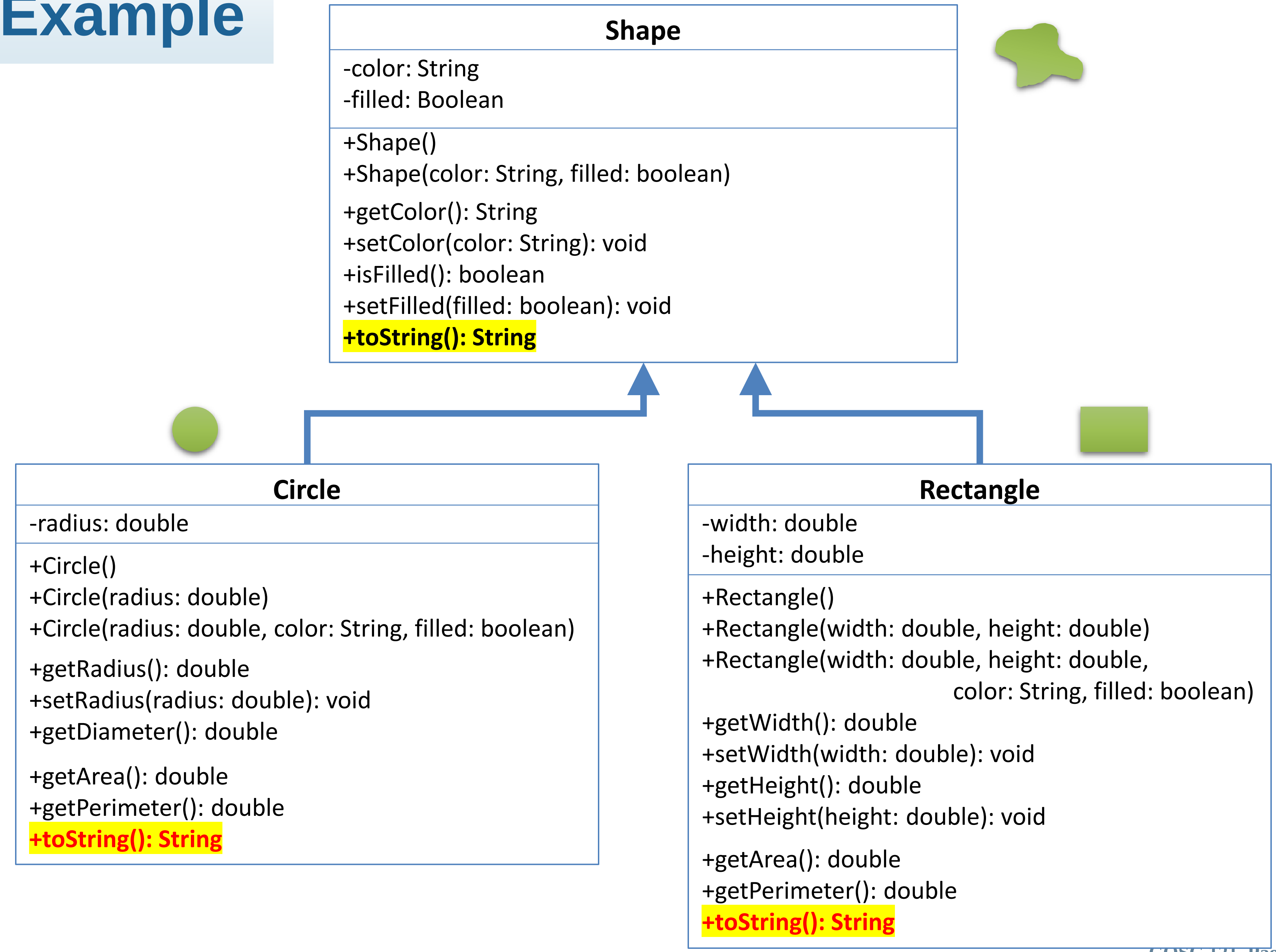
- i.e. provide a different implementation of a method that is already provided by the super-class.

Overriding happens when you implement a method in a subclass that has the same

- signature (name and parameters) ***and***
- return type (or subtype)

as a method in its super-class.

Example


Shape
-color: String
-filled: Boolean
+Shape()
+Shape(color: String, filled: boolean)
+getColor(): String
+setColor(color: String): void
+isFilled(): boolean
+setFilled(filled: boolean): void
+toString(): String
Circle
-radius: double
+Circle()
+Circle(radius: double)
+Circle(radius: double, color: String, filled: boolean)
+getRadius(): double
+setRadius(radius: double): void
+getDiameter(): double
+getArea(): double
+getPerimeter(): double
+toString(): String
Rectangle
-width: double
-height: double
+Rectangle()
+Rectangle(width: double, height: double)
+Rectangle(width: double, height: double, color: String, filled: boolean)
+getWidth(): double
+setWidth(width: double): void
+getHeight(): double
+setHeight(height: double): void
+getArea(): double
+getPerimeter(): double
+toString(): String

Inheritance Example, cont.

```
public class Shape {
  private String color;
  private boolean filled;

  public Shape() {this("White", true);}
  public Shape(String color, boolean filled) {
    setColor(color);
    setFilled(filled);
  }

  public String getColor() {return color;}
  public void setColor(String color) {this.color = color;}
  public boolean isFilled() {return filled;}
  public void setFilled(boolean filled) {this.filled = filled;}

  public String toString() {
    return "Color: " + color + ". Filled: " + filled;
  }
}
```

Inheritance Example, cont.

```
public class Circle extends Shape{
   private double radius;

   public Circle() {this(1);}
   public Circle(double radius) {
    setRadius(radius);
   }

   public void setRadius(double radius) {this.radius = radius;}
   public double getRadius() {return radius;}
   public double getDiameter() {return 2*radius;}

   public double getArea() {return Math.PI * radius * radius;}
   public double getPer() {return 2 * Math.PI * radius;}

   public String toString() {
    return "Color:" + getColor() + ". Filled: " + isFilled() +
             ". Radius: " + radius;
   }
}
```

Inheritance Example, cont.

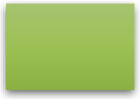
```
public class Rectangle extends Shape{
    private double width, height;

    public Rectangle() {this(1,1);}
    public Rectangle(double width, double height) {
        setWidth(width);
        setHeight(height);
    }

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

    public double getArea() {return width * height;}
    public double getPerimeter() {return 2 * (width+height);}

    public String toString() {
        return "Color:" + getColor() + ". Filled: " + isFilled() +
               ". Width: " + width + "Height: " + height;
    }
}
```



Overriding vs. Overloading

Overridden methods are in different classes related by inheritance; **overloaded** methods can be either in the same class or different classes related by inheritance.

Overridden methods have the same signature and return type; **overloaded** methods have the same name but a different parameter list.

```java
public class Test {
  public static void main(String[] args) {
    A a = new A();
    a.p(10);
    a.p(10.0);
  }
}

class B {
  public void p(double i) {
    System.out.println(i * 2);
  }
}

class A extends B {
  // This method overrides the method in B
  public void p(double i) {
    System.out.println(i);
  }
}
```

```java
public class Test {
  public static void main(String[] args) {
    A a = new A();
    a.p(10);
    a.p(10.0);
  }
}

class B {
  public void p(double i) {
    System.out.println(i * 2);
  }
}

class A extends B {
  // This method overloads the method in B
  public void p(int i) {
    System.out.println(i);
  }
}
```

`this` and `super` keywords

The `this` Keyword

The `this` keyword is the name of a reference that an object can use to refer to itself.

Uses:

- To reference class members within the class.
  - Class members can be referenced from anywhere within the class
  - Examples:
    - this.x = 10;
    - this.amethod(3, 5);
- To enable **a constructor to invoke another constructor** of the same class.
  - A constructor can only be invoked from within another constructor
  - Examples:
    - this(10, 5);

The `super` Keyword

The keyword `super` refers to the superclass of the class in which super appears.

Uses:

- To reference class members in the superclass.
  - Example:
    - super.amethod(3, 5);
    - super.toString();
- To enable **a constructor to invoke another constructor** of the superclass.
  - A constructor can only be invoked from within another constructor
  - Examples:
    - super(10, 5);

Example: this vs super for class members

```
public class Circle extends Shape{
  private double radius;

  public Circle() {this(1);}
  public Circle(double radius) {
   setRadius(radius);
  }

  public void setRadius(double radius) {this.radius = radius;}
  public double getRadius() {return radius;}
  public double getDiameter() {return 2*radius;}

  public double getArea() {return Math.PI * radius * radius;}
  public double getPer() {return 2 * Math.PI * radius;}

  public String toString() {
   return "Color:"+super.getColor()+". Filled: " + super.isFilled()+
          ". Radius: " + this.radius;
  }
}
```

Superclass Constructors

Explicit & implicit calling of superclass constructor

If no constructor is called within a given constructor, Java implicitly calls the super constructor. For example, the following two segments of code are equivalent:

```
class A{
   public A(){
      System.out.print(1);
   }
}

class B extends A{
   public B(){
      System.out.print(2);
   }
}
```

```
class A{
   public A(){
      System.out.print(1);
   }
}

class B extends A{
   public B(){
      super();
      System.out.print(2);
   }
}
```

```
Output of
B b = new B();
is 12
```

CAUTION: You must use the keyword super to call the superclass constructor. Invoking a superclass constructor's name in a subclass causes a syntax error. Java requires that the statement that uses the keyword super appear first in the constructor.

Example

This example is based on the Circle class presented a few slides ago.

```
Circle c = new Circle(1);
System.out.println(c.toString());
```

Output:
```
Color: White. Filled: true. Radius: 1.0
```

In above output, we created a White, Filled circle, although these attributes were not coded in the Circle constructor:

```
public Circle(double radius) {
   setRadius(radius);
}
```

The reason is, the Cicle constructor calls the super constructor by default.

```
public Circle(double radius) {
   super();
   setRadius(radius);
}
```

Constructor Chaining

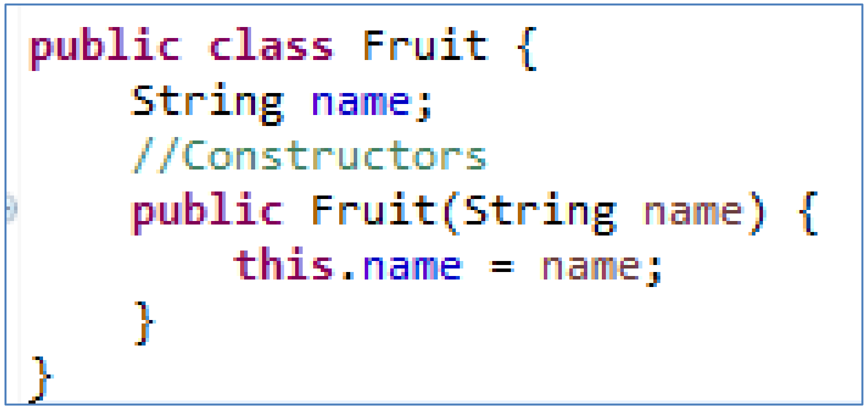
Constructing an instance of a class invokes all the superclasses' constructors along the inheritance chain. This is known as **constructor chaining**.

```java
class Person {
  public Person() {System.out.print(1);}
}
class Employee extends Person {
  public Employee() {
    this(2);
    System.out.print(3);
  }
  public Employee(int n) {System.out.print(n);}
}
class Faculty extends Employee {
  public Faculty() {System.out.print(4);}
}
```

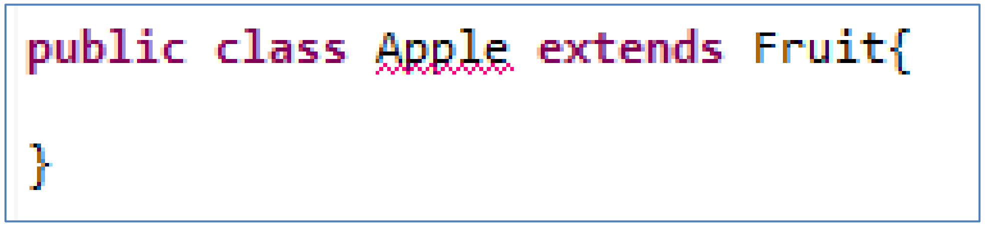
```java
public static void main(String[] args) {
   Faculty f = new Faculty(); //output is 1234
}
```

Example on the Impact of a Superclass without no-arg Constructor

What is wrong with the code below?



```
public class Fruit {
    String name;
    //Constructors
    public Fruit(String name) {
        this.name = name;
    }
}
```



```
public class Apple extends Fruit{

}
```

`final` modifier

The `final` Modifier

A `final` local variable is a constant inside a method.

The `final` class cannot be extended:

```
final class Math {
    ...
}
```

The `final` method cannot be overridden by its subclasses.

Visibility Modifiers Revisited

Visibility Modifiers

Access modifiers are used for controlling levels of access to class members in Java. We shall study two modifiers:

`public,`

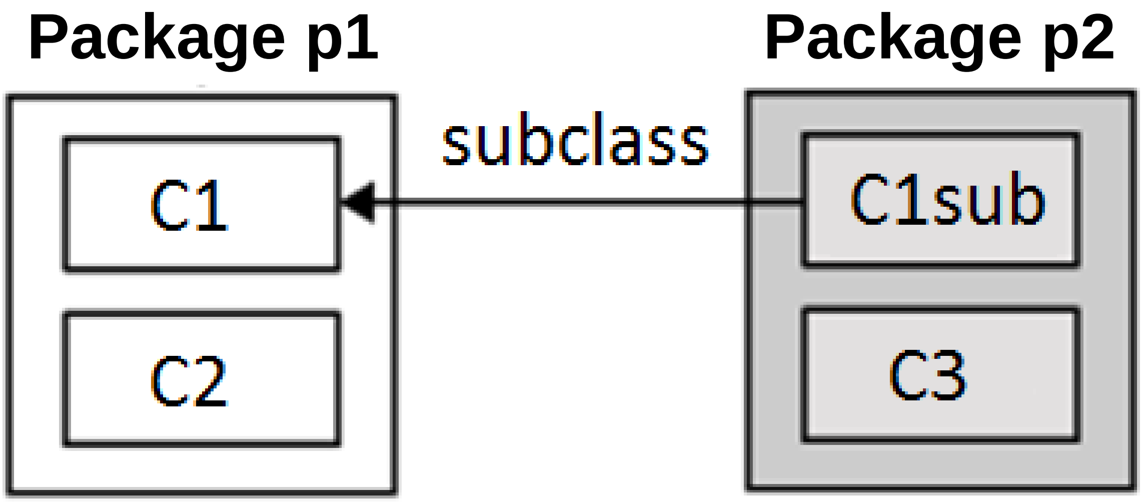
- The class, data, or method is visible to any class in any package.

`Private:`

- The data or methods can be accessed only by the declaring class.

If no access modifier is used, then a class member can be accessed by any class in the same package.

Visibility Modifiers

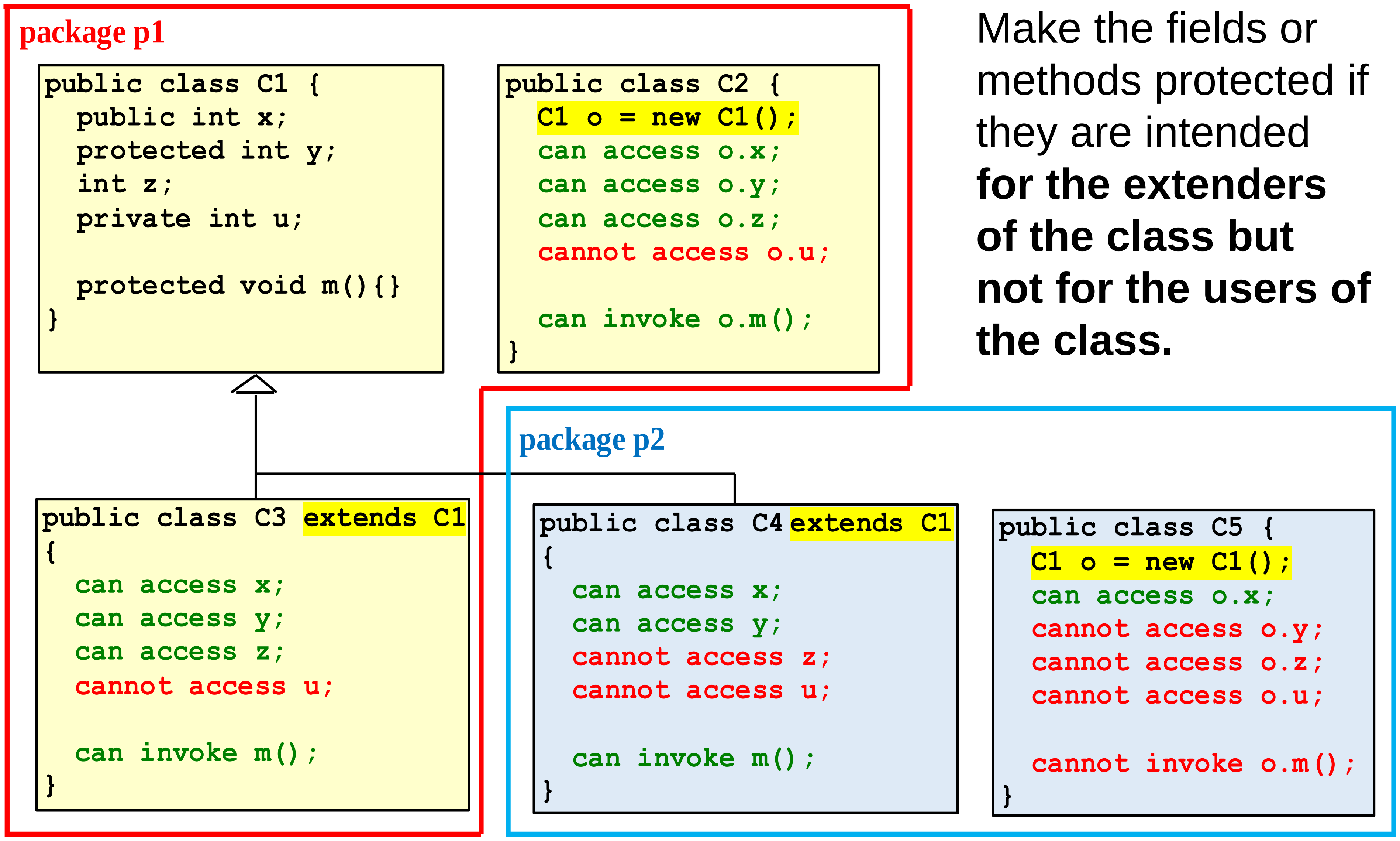


Visibility of a class member in C1

Modifier	C1	C2	C1sub	C3
public	Yes	Yes	Yes	Yes
protected	Yes	Yes	Yes	No
no modifier	Yes	Yes	No	No
Private	Yes	No	No	No

NOTE
Java 9 introduces a new feature: Java modules, which allows for more accessibility levels (e.g. public to module only instead of to all) but we won't discuss it in this class.

Visibility Modifiers



```
public class C1 {
  public int x;
  protected int y;
  int z;
  private int u;

  protected void m(){}
}
```

```
public class C2 {
  C1 o = new C1();
  can access o.x;
  can access o.y;
  can access o.z;
  cannot access o.u;

  can invoke o.m();
}
```

```
public class C3 extends C1
{
  can access x;
  can access y;
  can access z;
  cannot access u;

  can invoke m();
}
```

```
public class C4 extends C1
{
  can access x;
  can access y;
  cannot access z;
  cannot access u;

  can invoke m();
}
```

```
public class C5 {
  C1 o = new C1();
  can access o.x;
  cannot access o.y;
  cannot access o.z;
  cannot access o.u;

  cannot invoke o.m();
}
```

Make the fields or methods protected if they are intended **for the extenders of the class but not for the users of the class.**

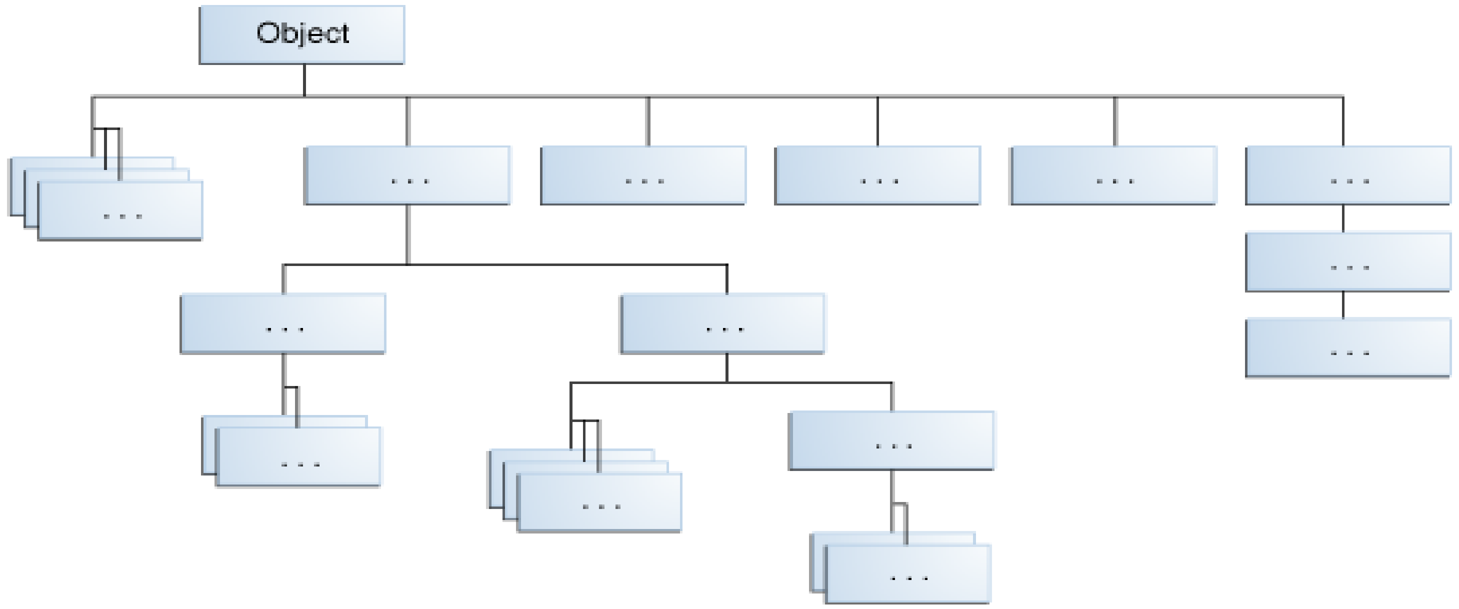
A Subclass Cannot Weaken the Accessibility

A subclass may override a protected method in its superclass and change its visibility to public. However, a subclass cannot weaken the accessibility of a method defined in the superclass. For example, if a method is defined as public in the superclass, it must be defined as public in the subclass.

The `Object` Class and Its Methods

The `Object` class

Classes in Java are descendants of **`java.lang.Object`** class



Source: oracle.com

Several methods are inherited from **`Object`** such as:

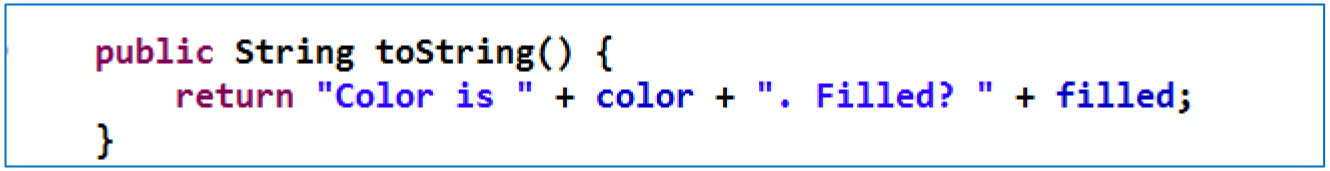
- **`public String toString()`**
  - Returns a string representation of the object.
- **`public boolean equals(Object obj)`**
  - Indicates whether some other object is "equal to" this one
- …

The `toString()` method

The `toString()` method returns a string representation of the object.

Usually you should **override the `toString`** method so that it returns a descriptive string representation of the object.

- For example, the `toString` method in the `Object` class was overridden in the `Shape` class presented earlier as follows:



```
public String toString() {
    return "Color is " + color + ". Filled? " + filled;
}
```