UBC OKANAGAN

COSC 111

Computer Programming I

Inheritance

Dr. Firas Moosvi

Acknowledgement: Slides (provided by Dr. Abdallah Mohammed) mainly rely on the materials prepared by Y. D. Liang for the textbook "Introduction to Java Programming, 10th Ed., Pearson Edu. Inc.".

The Three Pillars of OOP

OOP
Encapsulation
Inheritance
Polymorphism

The 2nd pillar

Inheritance

Inheritance Overview

Inheritance is a mechanism for enhancing and extending existing, working classes.

- In real life, you inherit some of the properties from your parents when you are born. However, you also have unique properties specific to you.
- In Java, a class that extends another class inherits some of its properties (methods, instance variables) and can also define properties of its own.

`extends` is the key word used to indicate when one class is related to another by inheritance.

Syntax: `class` *`subclass`* **`extends`** *`superclass`*

- The ***superclass*** is the existing, parent class.
- The ***subclass*** is the new class which contains the functionality of the superclass plus new variables and methods.
- A subclass may only inherit from ***one*** superclass.

Why use inheritance?

The biggest reason for using inheritance is to ***re-use code.***

- Once a class has been created to perform a certain function it can be re-used in other programs.
- Further, using inheritance the class can be extended to tackle new, more complex problems without having to re-implement the part of the class that already works.

The alternative is copy and paste which is bad, especially when the code changes.

Example:

- in the `Circle` and `Rectangle` classes we implemented a few slides ago, there was a lot of code redundancy (e.g. `setColor()` was exactly repeated).
- A better solution is to have a superclass, e.g. `Shape`, that has the common code and then have `Circle` and `Rectangle` inherit from `Shape`.

What is inherited?

When a subclass inherits (or extends) a superclass:

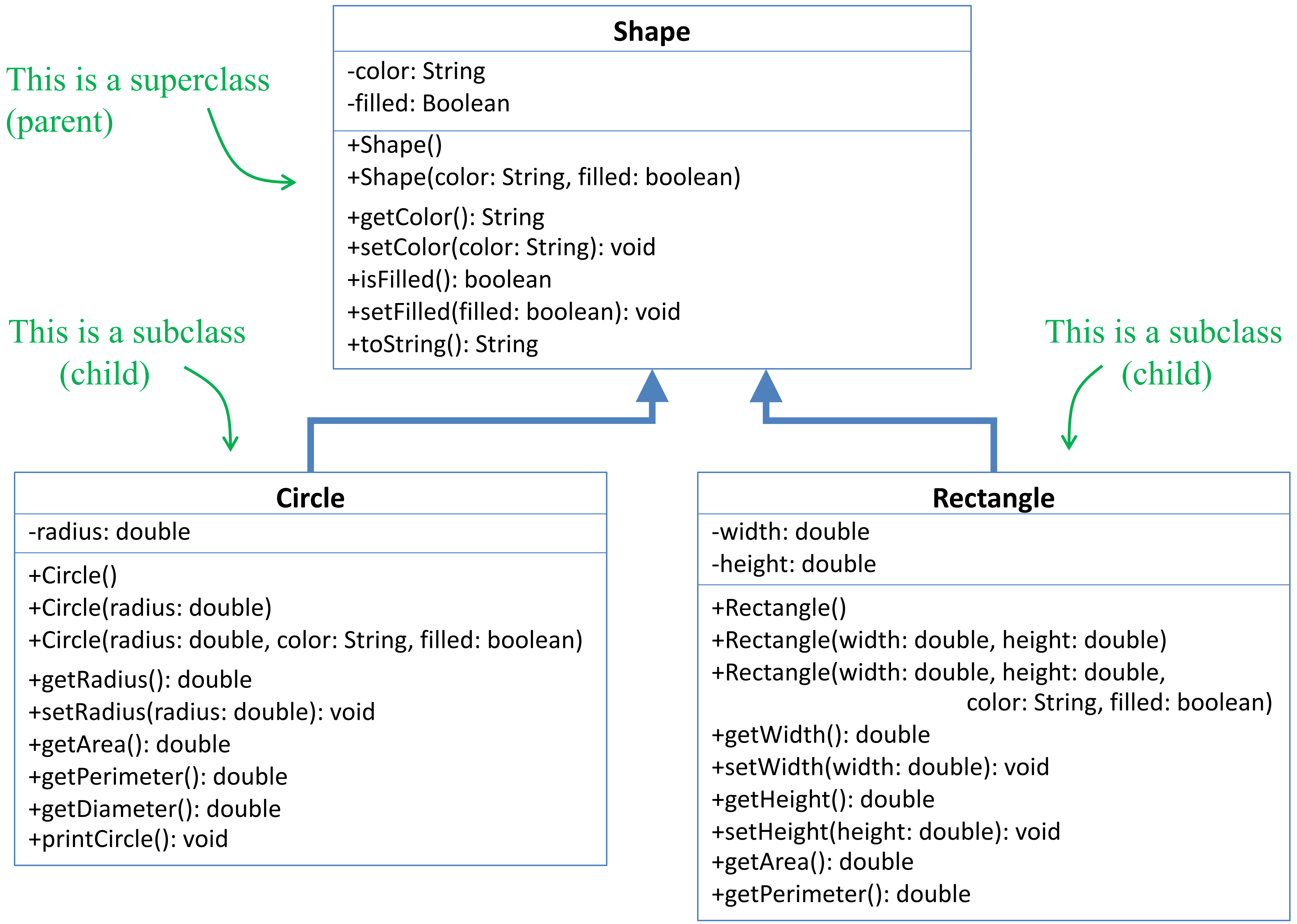
Instance variable inheritance:

- All instance variables of the superclass are inherited by the subclass.
  - However, if a variable is `private`, it can only be accessed using methods defined by the superclass.

Method inheritance:

- All superclass methods are inherited by the subclass, but they may be ***overridden***.

Inheritance Example



This is a superclass (parent)
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
This is a subclass (child)
This is a subclass (child)
Circle
-radius: double
+Circle()
+Circle(radius: double)
+Circle(radius: double, color: String, filled: boolean)
+getRadius(): double
+setRadius(radius: double): void
+getArea(): double
+getPerimeter(): double
+getDiameter(): double
+printCircle(): void
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

Inheritance Example, cont.

Shape is the superclass (parent)

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

    public Shape() {color = "white";}
    public Shape(String color, boolean filled) {
        this.color = color;
        this.filled = 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 is " + color + ". Filled? " + filled;
    }
}
```

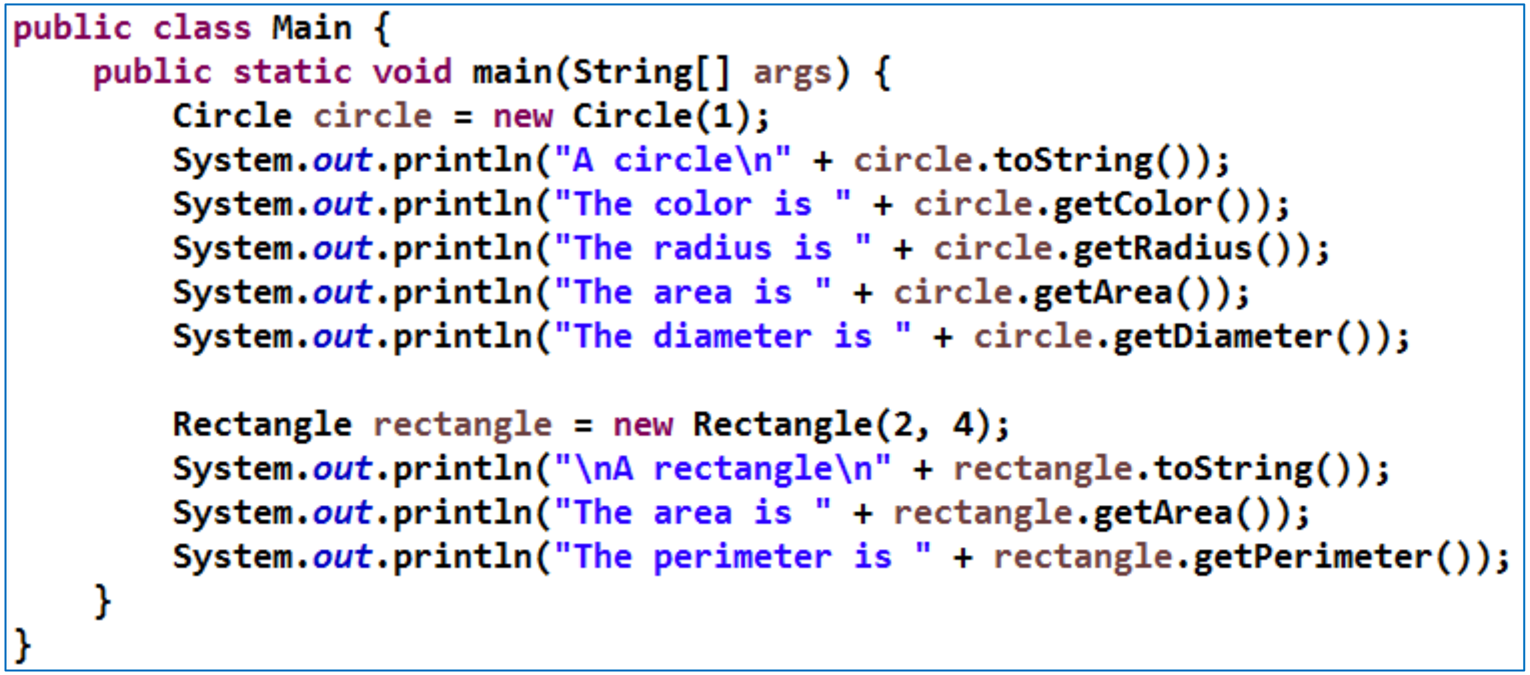
Inheritance Example, cont.

Rectangle is the subclass.
Its parent is Shape

```java
public class Rectangle extends Shape {
    private double width;
    private double height;
    public Rectangle() { this(1.0, 1.0); }
    public Rectangle(double width, double height) {
        this.width = width;
        this.height = height;
    }
    public Rectangle(double w, double h, String color, boolean filled) {
        setWidth(w);
        setHeight(h);
        setColor(color);
        setFilled(filled);
    }
    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); }
}
```

Inheritance Example, cont.

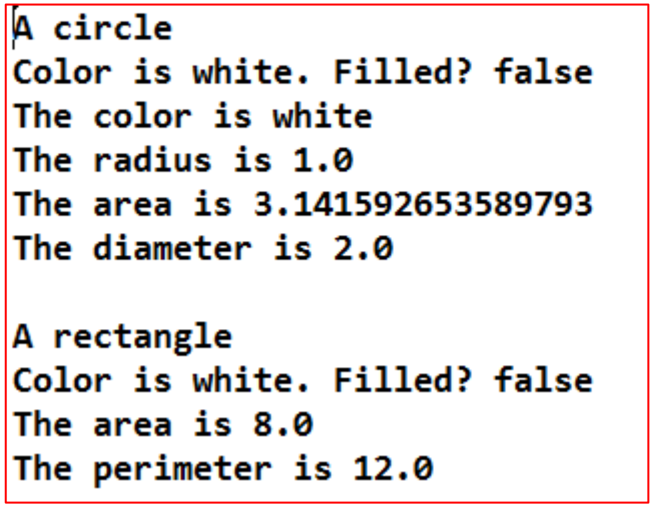
This is a test program!



```java
public class Main {
    public static void main(String[] args) {
        Circle circle = new Circle(1);
        System.out.println("A circle\n" + circle.toString());
        System.out.println("The color is " + circle.getColor());
        System.out.println("The radius is " + circle.getRadius());
        System.out.println("The area is " + circle.getArea());
        System.out.println("The diameter is " + circle.getDiameter());

        Rectangle rectangle = new Rectangle(2, 4);
        System.out.println("\nA rectangle\n" + rectangle.toString());
        System.out.println("The area is " + rectangle.getArea());
        System.out.println("The perimeter is " + rectangle.getPerimeter());
    }
}
```

The output



```
A circle
Color is white. Filled? false
The color is white
The radius is 1.0
The area is 3.141592653589793
The diameter is 2.0

A rectangle
Color is white. Filled? false
The area is 8.0
The perimeter is 12.0
```

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.
- **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.
- **Invoke** a superclass constructor from within a subclass constructor
  - either ***implicitly***
  - or ***explicitly*** using the keyword super

Overriding methods

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);
  }
}
```

Clicker Question

In which OOP pillar does an object acquire all the attributes and behaviors of the parent object?

A. Encapsulation

B. Inheritance

C. Polymorphism

D. None of the above

Clicker Question

What is subclass in java?

A. a class that extends another class

B. a class declared inside a class

C. a class that uses the keyword `sub` in its header

D. Both above.

E. None of the above.

Clicker Question

What is/are the advantage(s) of inheritance in Java?

A. Code re-usability

B. Save development time

C. Class extendibility

D. All of the above

Clicker Question

Which of the following is not inherited?

A. Instance variables

B. Constructors

C. Method

D. Both (B) and (C)

E. None of the above

Clicker Question

What is the output?

A. 0

B. 1

C. 2

D. 3

E. Error

```
class A {
   public int x;
   public void display() {
      System.out.println(x);
   }
}
class B extends A {
   public int y;
   public void display() {
      System.out.println(y);
   }
}

class Test {
   public static void main(String args[]){
      B b = new B();
      b.x = 1;
      b.y = 2;
      b.display();
   }
}
```

Clicker Question

What is the output?

A. 0

B. 1

C. 2

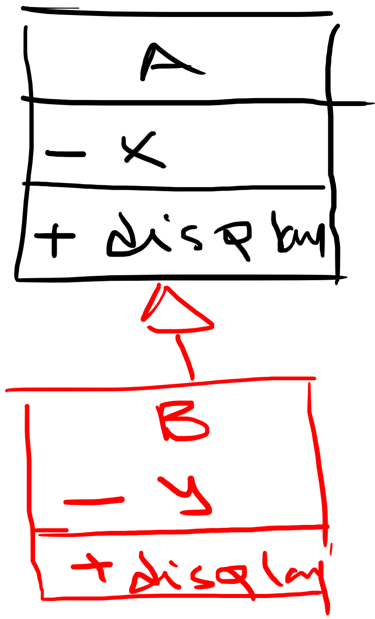
D. 3

E. Error

```
class A {
    private int x;
    public void display() {
        System.out.println(x);
    }
}
class B extends A {
    private int y;
    public void display() {
        System.out.println(x+y);
    }
}

class Test {
    public static void main(String args[]){
        B b = new B();
        b.display();
    }
}
```

Be careful!



`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);

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.

Constructor Chaining

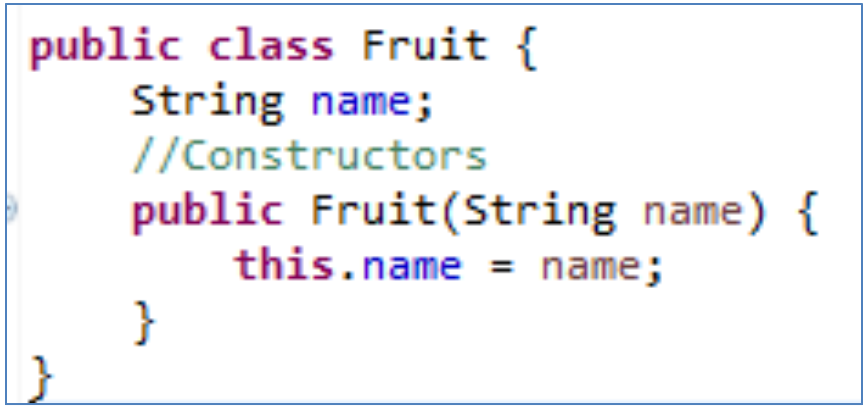
Constructing an instance of a class invokes all the superclasses' constructors along the inheritance chain. This is known as **constructor chaining**.

```
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);}
}
```

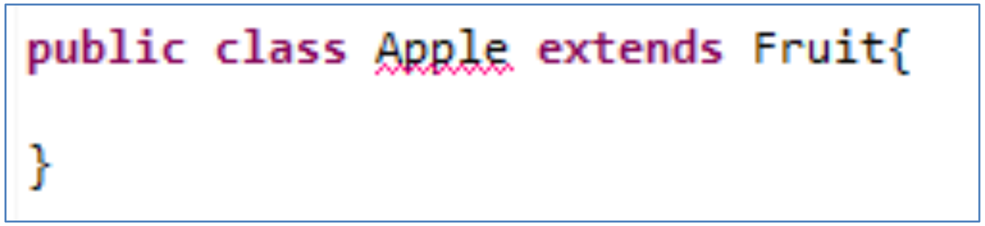
```
public static void main(String[] args) {
   Faculty f = new Faculty(); //output is 1234
}
```

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{

}
```

Clicker Question

Which of these keywords is used to call a no-arg constructor of a superclass named **`Shape`** from its subclass?

A. **`Shape()`**

B. **`this.Shape()`**

C. **`super.Shape()`**

D. **`this()`**

E. **`super()`**

`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.

Clicker Question

Which of the following is FALSE

A. final class cannot be inherited

B. final method can be inherited

C. final method can be overridden

D. final variable cannot be changed.

Clicker Question

Which class cannot be extended?

A. suerclass
B. subclass
C. final class
D. Object class
E. abstract class

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,`

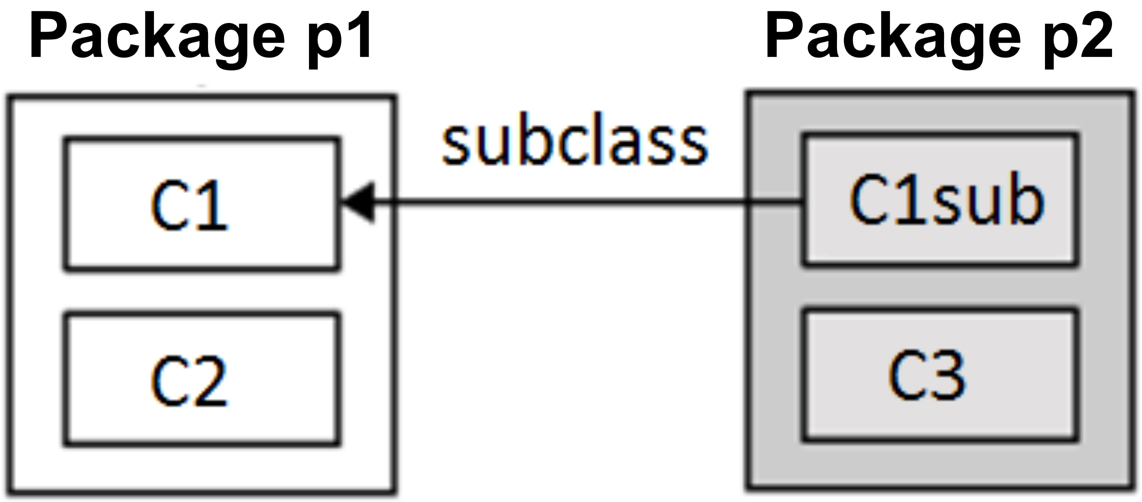
- The class, data, or method is visible to any class in any package.

`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

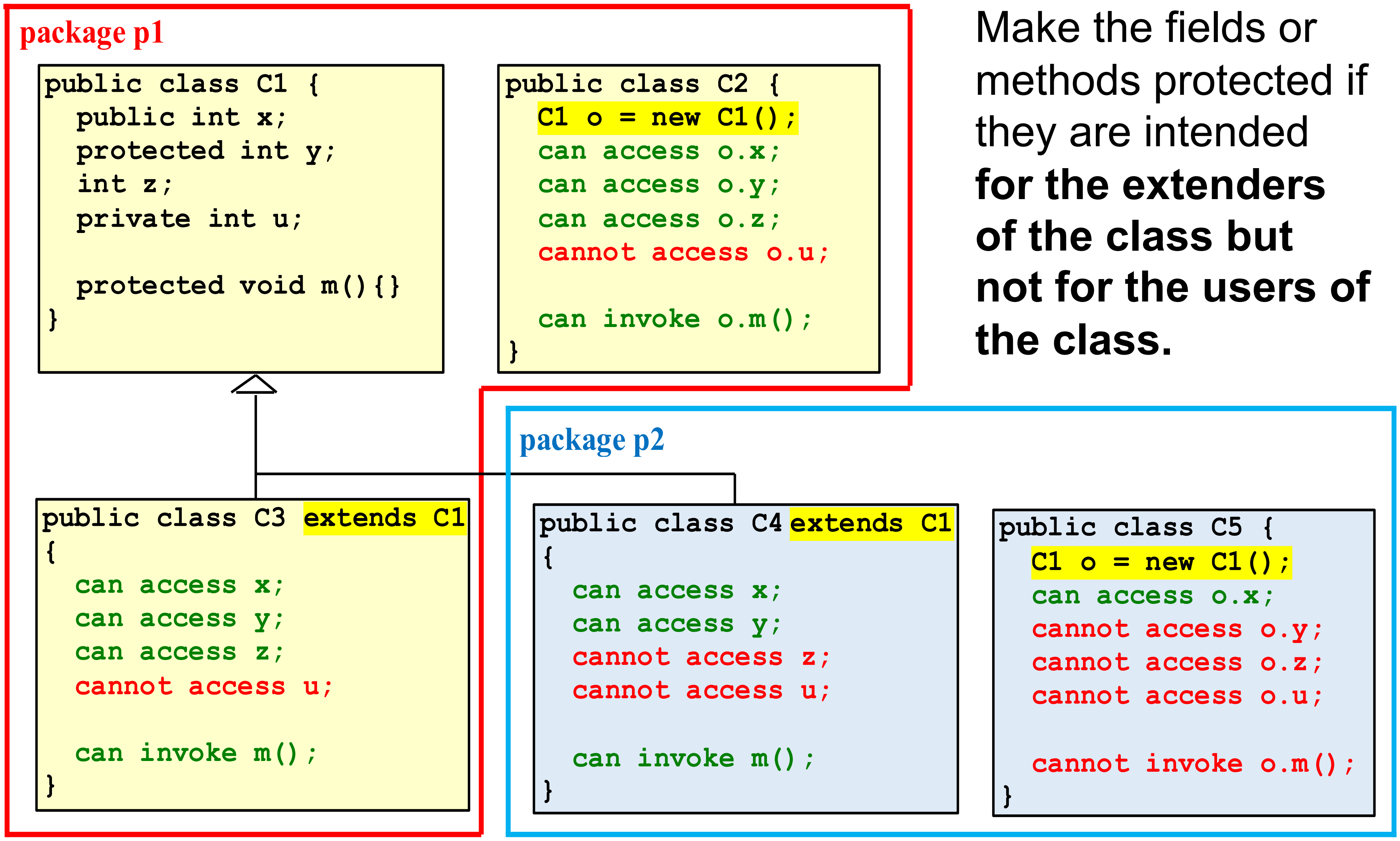


Visibility of a class member in C1

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



package p1

```
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();
}
```

package p2

```
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.**

Clicker Question

What is the output?

A. 0

B. 1

C. 2

D. 01

E. Error

```
class A {
   public int x;
   private int y;
}

class B extends A {
   public void display(){
      super.y = super.x + 1;
      System.out.println(super.x+super.y);
   }
}

class Q {
   public static void main(String args[]){
      B b = new B();
      b.display();
   }
}
```

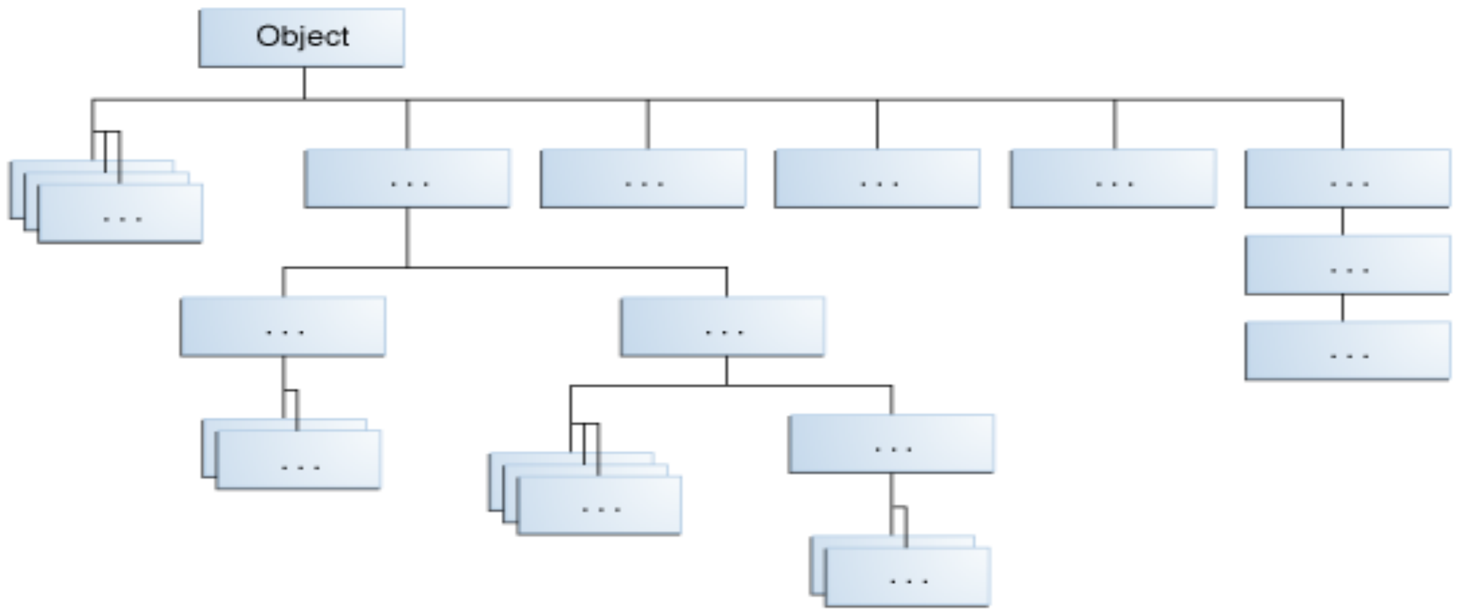
A Subclass Cannot Weaken the Accessibility

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:

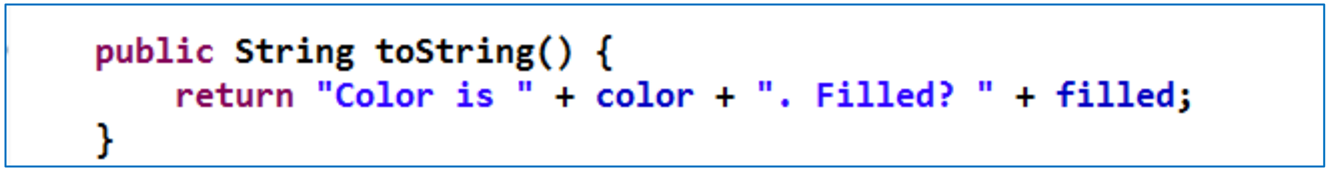
- `public String toString()`
  - Returns a string representation of the object.
- `public boolean equals(Object obj)`
  - Indicates whether some other object is "equal to" this one
- ...

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;
}
```