Object Oriented Programming

Chapter 8

Ayesha Khatun

Lecturer, Green University of Bangladesh

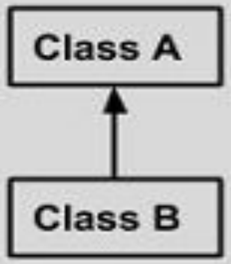
Inheritance

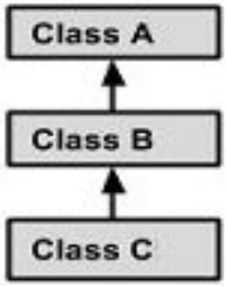
Inheritance is an important pillar of OOP(Object Oriented Programming). It is the mechanism in java by which one class is allow to inherit the features(fields and methods) of another class.

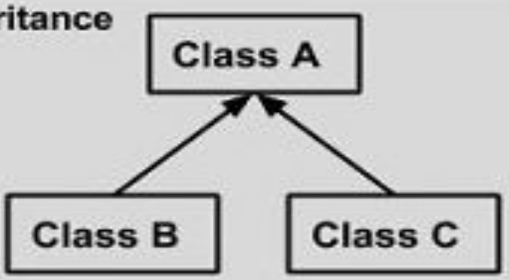
Important terminology:

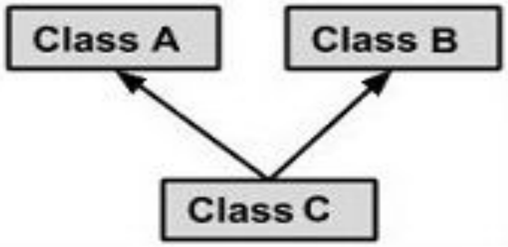
Super Class: The class whose features are inherited is known as super class(or a base class or a parent class).

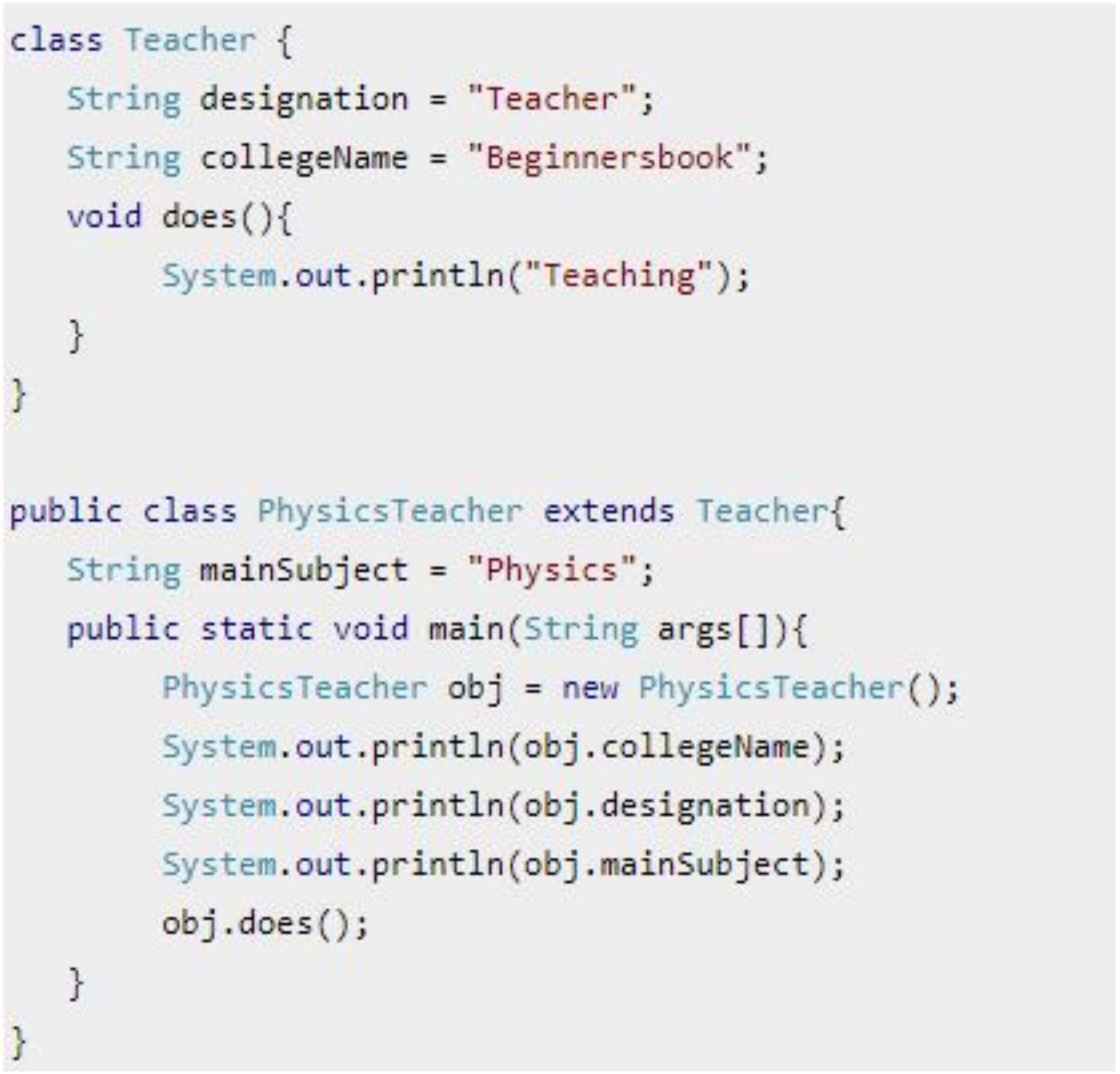
Sub Class: The class that inherits the other class is known as sub class(or a derived class, extended class, or child class). The subclass can add its own fields and methods in addition to the superclass fields and methods.

Single Inheritance 	public class A {
	    ....
	}
	public class B **extends** A {
	    ....
	}

Multi Level Inheritance 	public class A { ....}
	
	public class B **extends** A {....................}
	
	public class C **extends** B {..................... }

Hierarchical Inheritance 	public class A { ....}
	
	public class B **extends** A {....................}
	
	public class C **extends** A {..................... }

Multiple Inheritance 	public class A { ....}
	
	public class B {....................}
	
	public class C **extends** A,B {
	    ....
	} // Java does not support mutiple Inheritance

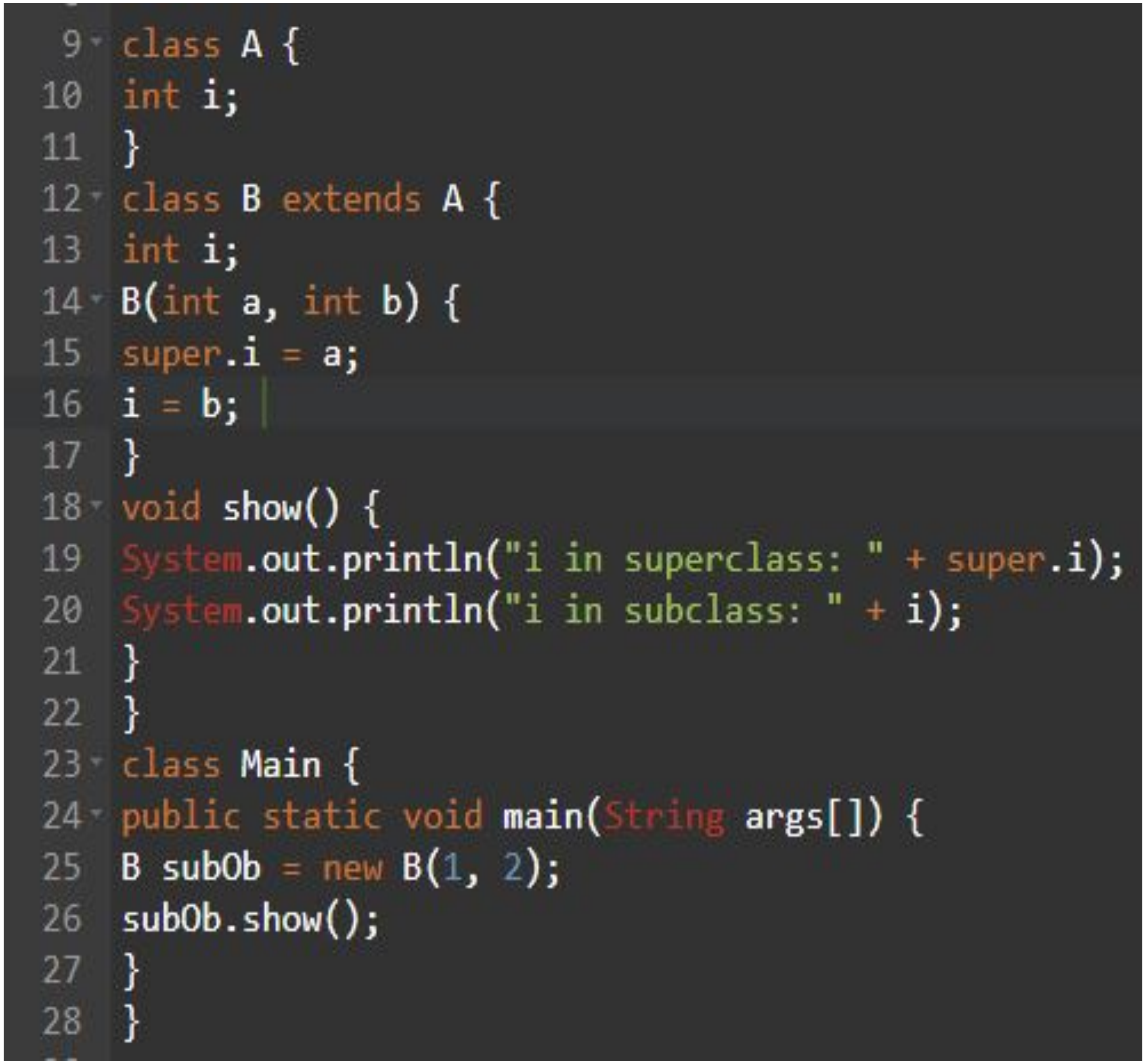


```java
class Teacher {
   String designation = "Teacher";
   String collegeName = "Beginnersbook";
   void does(){
        System.out.println("Teaching");
   }
}

public class PhysicsTeacher extends Teacher{
   String mainSubject = "Physics";
   public static void main(String args[]){
        PhysicsTeacher obj = new PhysicsTeacher();
        System.out.println(obj.collegeName);
        System.out.println(obj.designation);
        System.out.println(obj.mainSubject);
        obj.does();
   }
}
```

Using super

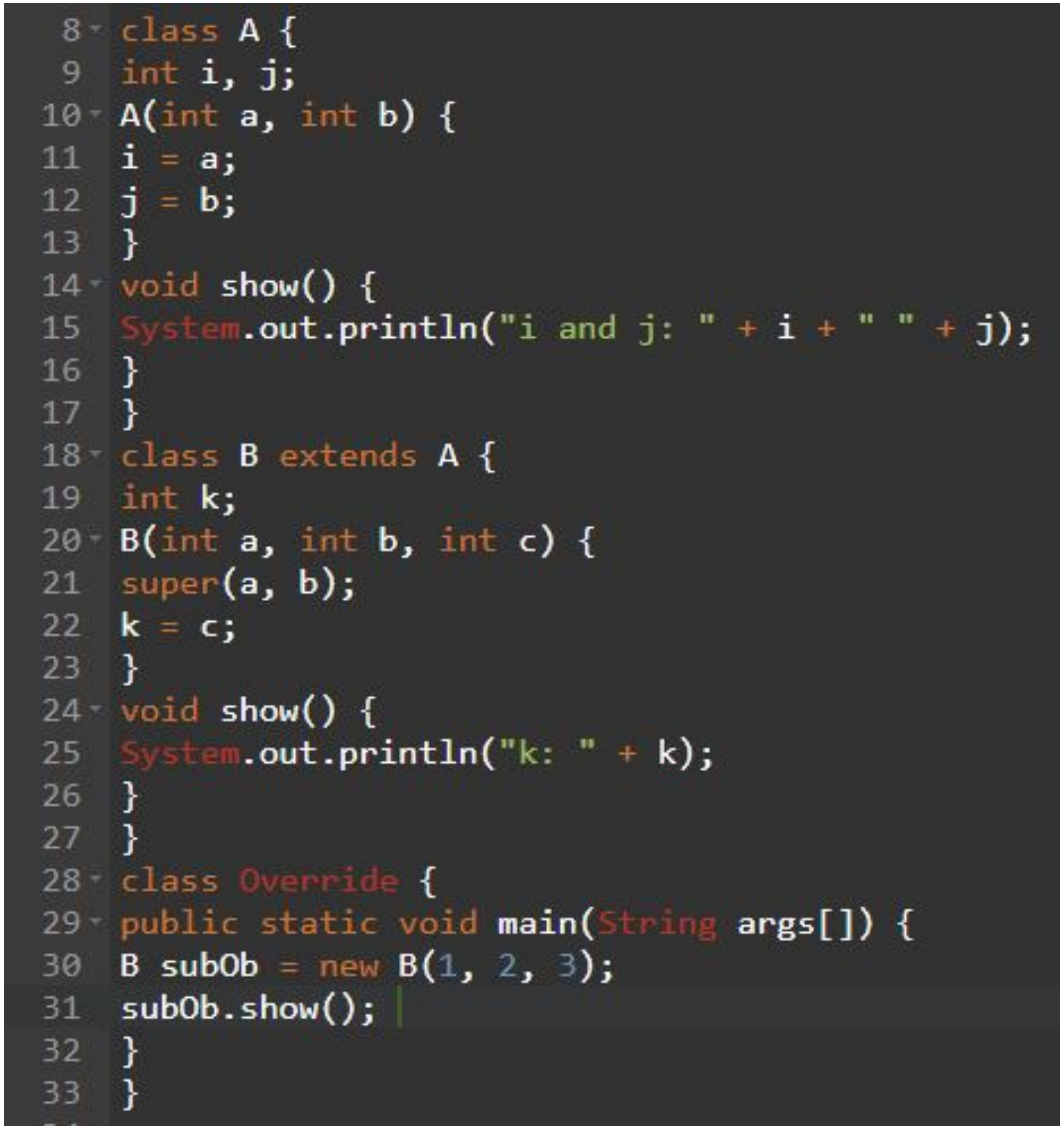
- Whenever a subclass needs to refer to its immediate super class, it can do so by use of the keyword super.
- **super has two general forms. The first calls the super class constructor.**
- **The second is** used to access a member of the super class that has been hidden by a member of a subclass.



```java
class A {
int i;
}
class B extends A {
int i;
B(int a, int b) {
super.i = a;
i = b;
}
void show() {
System.out.println("i in superclass: " + super.i);
System.out.println("i in subclass: " + i);
}
}
class Main {
public static void main(String args[]) {
B subOb = new B(1, 2);
subOb.show();
}
}
```

Method Overriding

- In a class hierarchy, when a method in a subclass has the same name and type signature as a method in its super class, then the method in the sub class is said to *override the method in* the super class. When an overridden method is called from within a subclass.



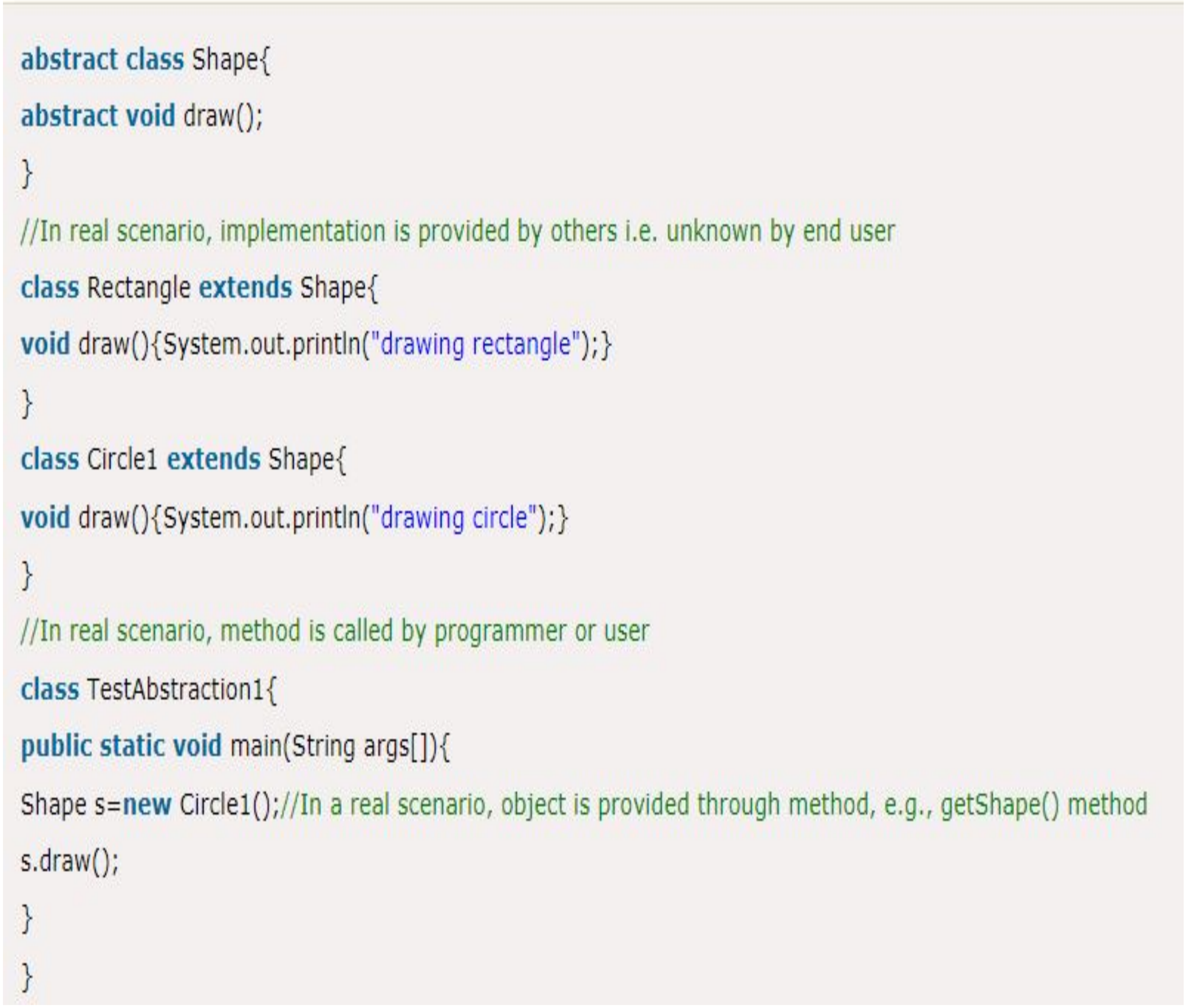
```java
class A {
int i, j;
A(int a, int b) {
i = a;
j = b;
}
void show() {
System.out.println("i and j: " + i + " " + j);
}
}
class B extends A {
int k;
B(int a, int b, int c) {
super(a, b);
k = c;
}
void show() {
System.out.println("k: " + k);
}
}
class Override {
public static void main(String args[]) {
B subOb = new B(1, 2, 3);
subOb.show();
}
}
```

Why Overridden Methods?

175P

Using Abstract Classes

- A class which is declared with the abstract keyword is known as an abstract class in Java. It can have abstract and non-abstract methods (method with the body).



```java
abstract class Shape{
abstract void draw();
}
//In real scenario, implementation is provided by others i.e. unknown by end user
class Rectangle extends Shape{
void draw(){System.out.println("drawing rectangle");}
}
class Circle1 extends Shape{
void draw(){System.out.println("drawing circle");}
}
//In real scenario, method is called by programmer or user
class TestAbstraction1{
public static void main(String args[]){
Shape s=new Circle1();//In a real scenario, object is provided through method, e.g., getShape() method
s.draw();
}
}
```

Using final with Inheritance

1. **Using final to Prevent Overriding**

```
class A {
final void meth() {
System.out.println("This is a final method.");
}
}
class B extends A {
void meth() { // ERROR! Can't override.
System.out.println("Illegal!");
}
}
```

Using final with Inheritance

2. Using final to Prevent Inheritance

```
final class A {
// ...
}
// The following class is illegal.
class B extends A { // ERROR! Can't subclass A
// ...
}
```

Thank You!