INHERITANCE IN JAVA-Part B

-Compiled by Nikahat Mulla

Agenda

- Revision: Single Inheritance
- super and hiding data
- Multilevel Inheritance
- Method overriding rules
- Polymorphism and Types

Single Inheritance

- Single class deriving from a base class
- All members of the base class except private and default are accessible to any subclass from everywhere.
- However, if the subclass is within the same folder (package), all members of the base class except private are accessible inside the subclass.

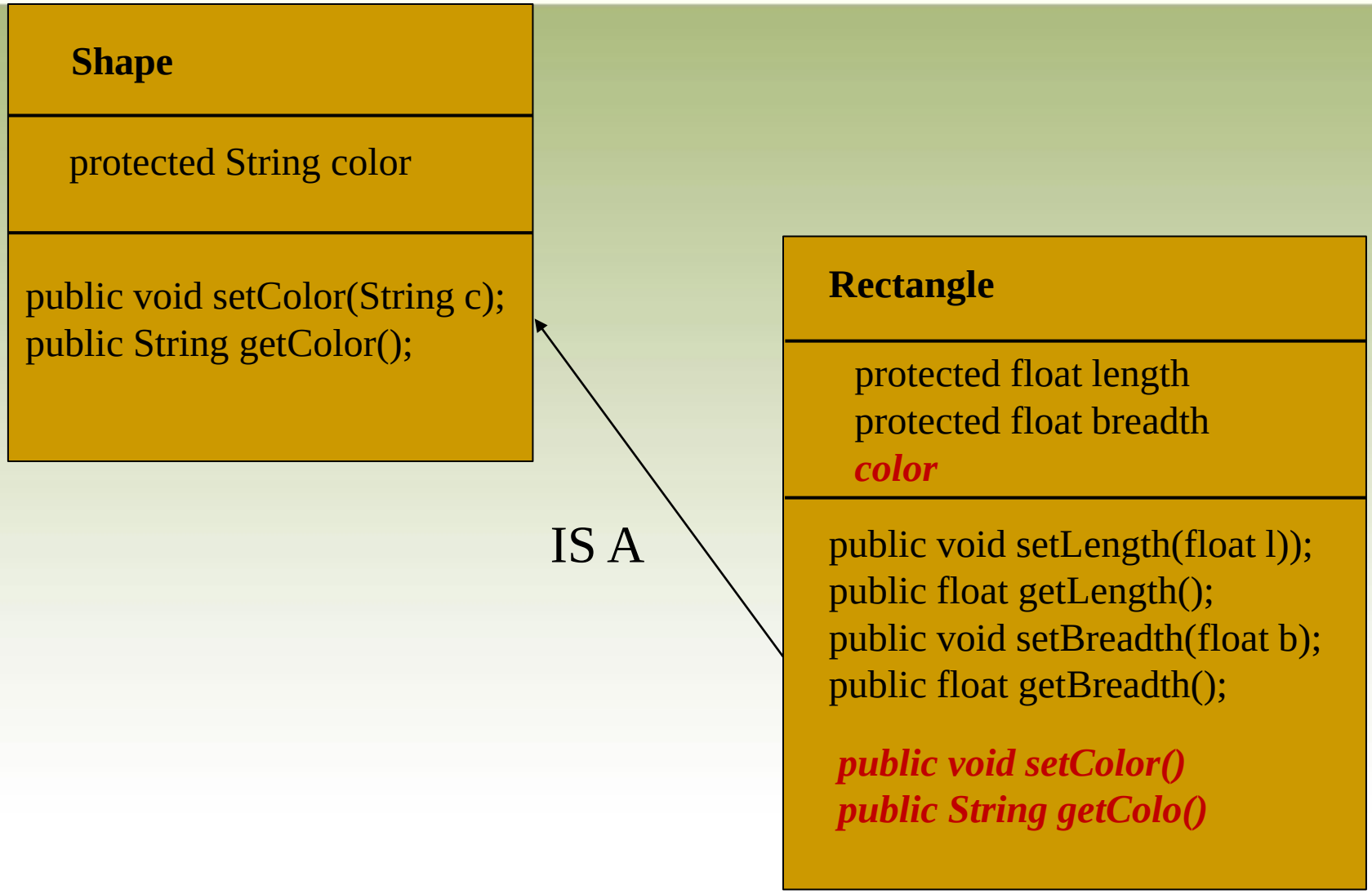
Single Inheritance

- Single Inheritance without constructors
- Single Inheritance with constructors

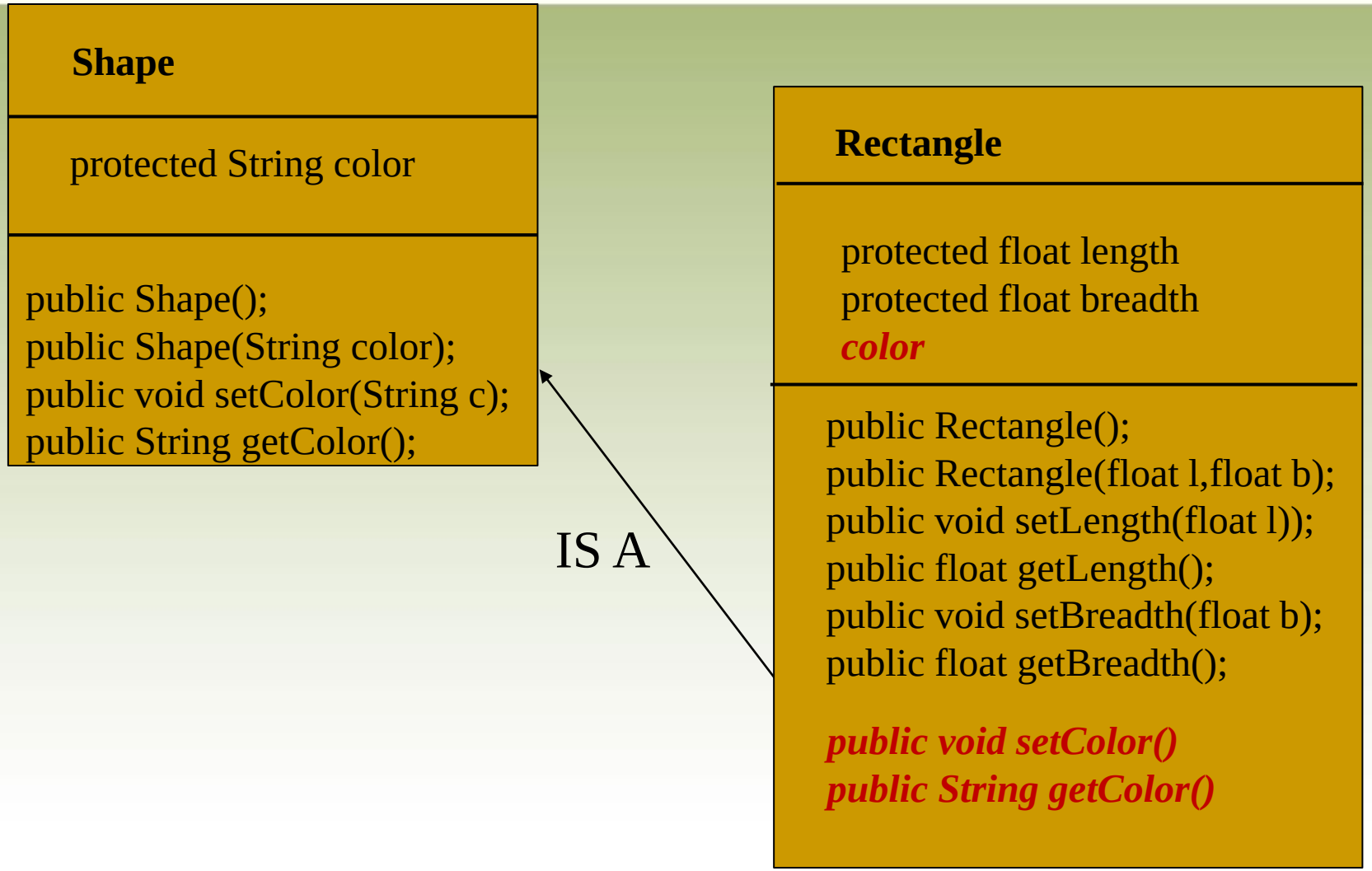
Single Inheritance Example

- Consider a class called Shape with member color of the shape. Consider a class called Rectangle, derived from the Shape class, which adds features length and breadth, add a method to print the rectangle's details and find the area of the rectangle. Use appropriate access modifiers. Let us model it with and without constructors.
- Program: TestRect1.java

Scenario 1: Without Constructors



Scenario 2: With Constructors



Using s*uper* to access hidden members

- *U*sed when member names of the subclass hide members by the same name in the super class

- super.member
  - *member* can be either a method or an instance variable
- Let us consider the previous example and attempt to create an instance variable called color in the Rectangle class: TestRect3.java

Multilevel Inheritance

- Extend the Rectangle class to create a new class Box with feature height. Add the method volume and override the toString() method in all classes.

Types of Polymorphism

- Polymorphism is exhibited using method overloading and method overriding
- There are two types of polymorphism depending upon on the time of binding a method to its implementation.
- Binding happens when a method invocation is bound to an implementation
  - Involves lookup of the method in the class, or one of its parents
  - Both method names & parameters are checked
- Can happen at:
  - Compilation Time (Static Binding->Static/ Compile Time Polymorphism)
  - Execution Time (Dynamic Binding->Dynamic/ Run-time Polymorphism)

Static Binding

- Static binding is done by the compiler
  - When it can determine the type of an object
- Method calls are bound to their implementation immediately

```
    public class MotorBike
    {
        public void revEngine() {…}
    }
MotorBike bike = new MotorBike();
bike.revEngine();
```

Dynamic Binding

- When an object's class cannot be determined at compile time
- JVM (not the compiler) has to bind a method call to its implementation
- Instances of a sub-class can be treated as instances of the parent class
- So the compiler doesn't know its type, just knows its base type

```
public class Mammal { .. . . .  }
public class Dog extends Mammal {  . . . .  }
public class Cat extends Mammal {  . . . .  }
Mammal m;
m = new Dog( );
m.speak(); // Invokes Dog's implementation
m= new Cat();
m.speak(); // Invokes Cat's implementation
```

Type Casting for Reference Types

- We can assign a variable of a certain class type with an instance of that particular class or an instance of any subclass of that class

```
public class Mammal { .. . . .  }
public class Dog extends Mammal {  . . . .  }
```

- When we cast a reference along the class hierarchy in a direction from the root class towards the subclasses, it's a downcast

```
Mammal m=new Dog();
Dog d=(Dog) m; // explicit casting
```

- When we cast a reference along the class hierarchy in a direction from the sub classes towards the root, it's an upcast

```
Mammal m=new Dog();// implicit casting
```

Rules for Method Overriding

1. Method must have same name, same arguments
2. It can broaden visibility in child class but not narrow it down. For example, if a method is protected in the parent class and overridden in the child class, then it can be made protected or public but not private in the child class.
3. Return type of overridden method should be same or if it returns an object its return type in child class can be the child class type.
4. Constructors and private methods cannot be overridden
5. A subclass cannot call the parent class method using its reference, if the method is overridden. Always the child class variant will be called.
6. static and final methods cannot be overridden.

this vs. super

Sr. No.	this	super
1.	this refers to the current object	super refers to the parent class
2.	It is used in methods and constructors when the arguments of these methods have the same name as the data members of the class	It is used to call the methods or refer to the data members of the super class from within methods of the child class
3.	this can also be used in constructors as a constructor call to other constructors and can be based as method parameter	It is used in inheritance to call the constructors of the parent classes from constructors of the base class for instantiating objects in the correct order.

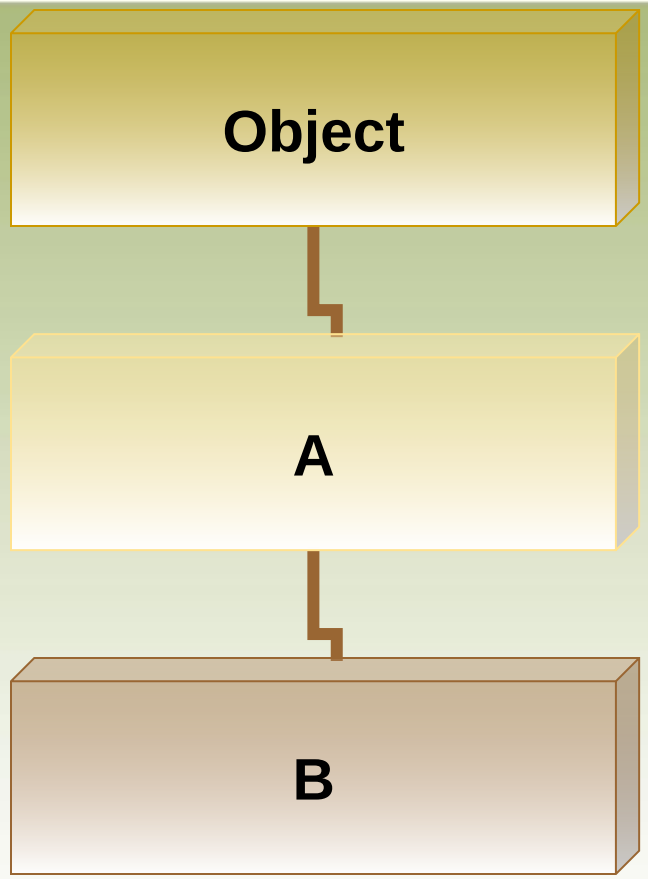
final keyword

- Using *final* to prevent Overriding
  - When a method in the superclass is declared as final, it cannot be overridden in the subclass
- Using *final* to prevent Inheritance
  - When a class is declared as final, it cannot be inherited
- final can also be used to make a data member of constant value. For e.g.

```
final int empid;
Employee(int id){empid=id;}
```

The Object Class

- Object is the base class for all Java classes
- Every class extends this class directly or indirectly
- Present in the package java.lang which is imported by default into all java programs




```
class A
{
        …
}
public class B extends A
{
        …
}
```

The *instanceof* Operator

- Used to check the actual type of an object
- Has two operands:
  - A reference to an object on the left
  - A class name on the right
- Returns true or false based on whether the object is an instance of the named class or any of that class's subclasses

The *instanceof* Operator (Contd…)

- An Example:

```
public class MotorBike extends Vehicle
{
        …
}
```

A class definition

```
Vehicle bike = new MotorBike();
if (bike instanceOf MotorBike)
 {
       -------
}
```

Check whether the bike reference is an instance of MotorBike or not

Thank You