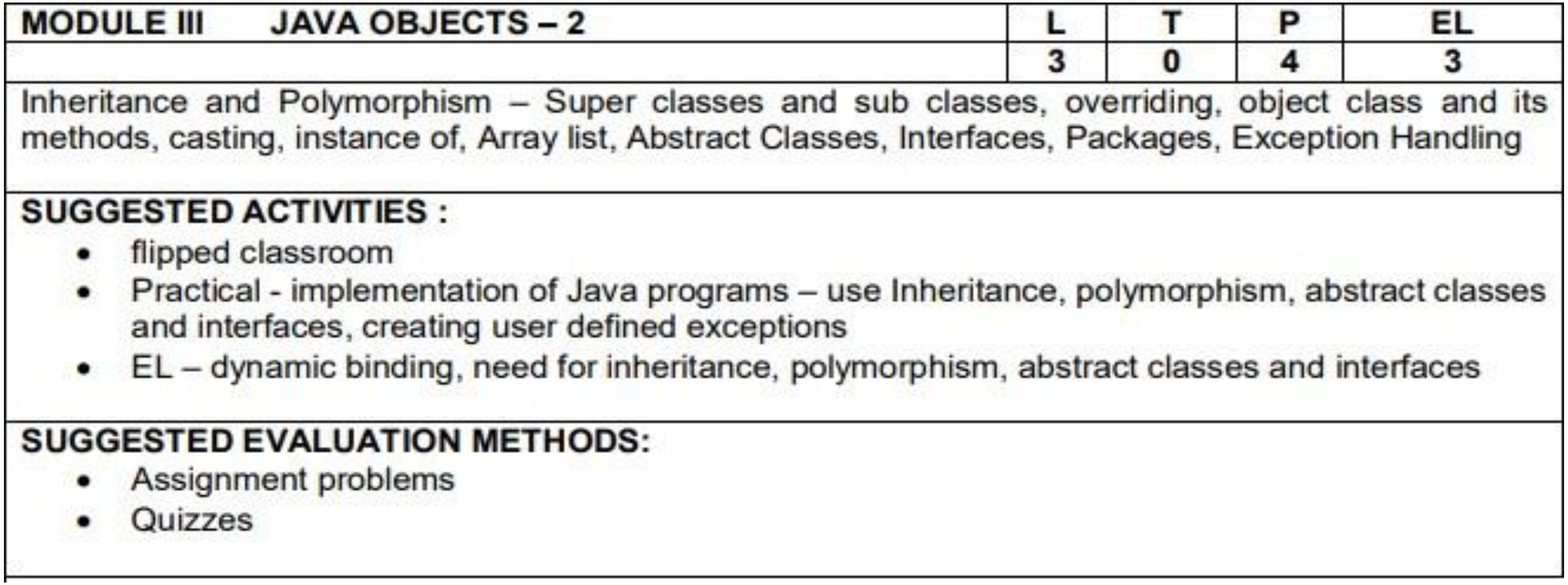
CS6308- Java Programming

V P Jayachitra
Assistant Professor
Department of Computer Technology
MIT Campus
Anna University

Syllabus



MODULE III JAVA OBJECTS – 2	L	T	P	EL
	3	0	4	3

Inheritance and Polymorphism – Super classes and sub classes, overriding, object class and its methods, casting, instance of, Array list, Abstract Classes, Interfaces, Packages, Exception Handling

SUGGESTED ACTIVITIES :

- flipped classroom
- Practical - implementation of Java programs – use Inheritance, polymorphism, abstract classes and interfaces, creating user defined exceptions
- EL – dynamic binding, need for inheritance, polymorphism, abstract classes and interfaces

SUGGESTED EVALUATION METHODS:

- Assignment problems
- Quizzes

Access Modifier

Public: accessible from anywhere

- Visibility: Accessible from any other class in any package.
- Usage:
  - Can be applied to classes, methods, and variables.
  - When a class is marked public, it can be accessed from any other class, regardless of the package.
  - When a method or variable is marked public, it can be accessed from any other class.

Private: accessible only within the same class.

- Visibility: Accessible only within the same class.
- Usage:
  - Can be applied to methods and variables.
  - A private member is not accessible from any other class, including subclasses.
  - Private methods cannot be overridden in subclasses.
  - a subclass cannot override a non-private method and make the new method private.

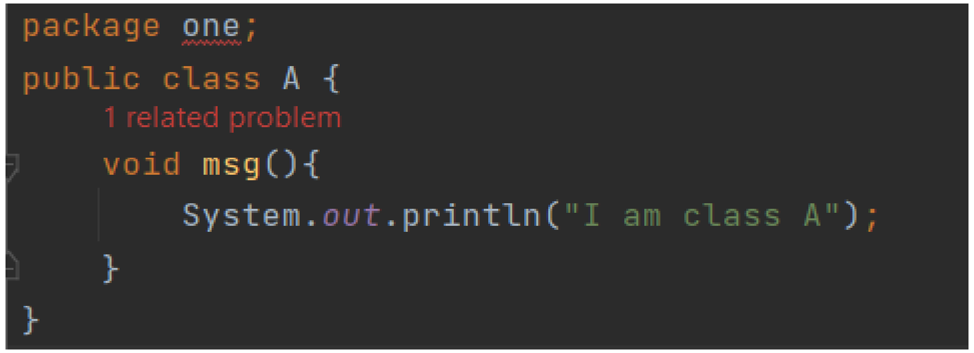
Protected:access within the same package and subclasses

- **Visibility: Accessible within the same class, subclasses, and classes in the same package.**
- **Usage:**
  - **Can be applied to methods and variables.**
  - **A protected member is accessible in subclasses even if they are in different packages.**
  - **A subclass can override a protected method or variable.**

```
/* default access modifier */
//file name:Demo.java
class A {
  void msg(){
                System.out.println(" I am Class A");
            }
}
class Demo{
                public static void main(String args[]){
                                A obj = new A();
                                obj.msg();
   }
}
```

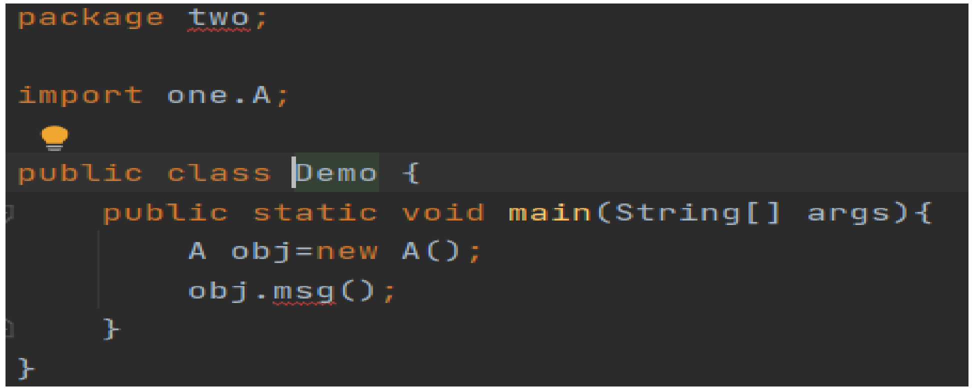
```
I am Class A
```

```
/* default access modifier */
//file name:A.java in package one
```



```
package one;
public class A {
    1 related problem
    void msg(){
        System.out.println("I am class A");
    }
}
```

```
//file name:Demo.java in package two
```



```
package two;

import one.A;

public class Demo {
    public static void main(String[] args){
        A obj=new A();
        obj.msg();
    }
}
```

```
java: msg() is not public in one.A; cannot be accessed from outside package
```

```
/* default access modifier */
//file name:A.java
package one;
class A {
  public void msg(){
               System.out.println(" I am Class A");
          }
}

//file name:Demo.java
package two;
class Demo{
               public static void main(String args[]){
                              A obj = new A();
                              obj.msg();
   }
}
```

```
 I am Class A
```

```
/* private access modifier */
//file name:Demo.java
class A {
  private int n=2;
  void msg(){
               System.out.println(" n:“+n);
          }
}

public class Demo {
               public static void main(String args[]){
               A obj = new A();
               //System.out.println(obj.n); //Compile error
               obj.msg(); //access private value via public method
               }
}
```

```
n:2
```

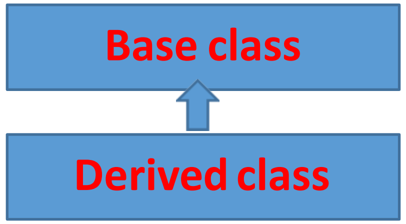
```
/* private access modifier */
class A{
		private A(){//private constructor

				}

		void msg(){
				System.out.println("Class A");
		}
}

public class Demo {
		public static void main(String args[]){
				A obj = new A(); //Compile Time Error
		}
}
```

```
/* private access modifier */
package one;
public class A {
   public void msg1() {
      System.out.println(" Public!");
   }

   private void msg2() {
      System.out.println("Private!");
   }
   protected void msg3(){
      System.out.println("Protected!");
   }
}
```

```
package two;
import one.*;
public class Demo extends A{
   public void msg1() {
      System.out.println("Class A: Public!");     }
   private void msg2() {
      System.out.println("Class A: Private!");     }
   protected void msg3(){
      System.out.println("Class A: Protected!");     }
   public static void main(String[] args){
      A obj=new A();
      obj.msg1();
      obj.msg2(); //ERROR msg2() has private access in one.A
      obj.msg3();
   }
}
```

What is inheritance?

- The mechanism by which one class acquires the properties(attributes) and functionalities(methods) of another class is called **inheritance**.
- Allows subclass of a class to inherit all of its member elements and methods from its superclass as well as creates its own.
- A special key word **extends** is used to implement this mechanism.
- Inheritance is that it expresses an "**is-a**" relationship.
  - An eagle *is a* bird.
  - **is-a relationship**: Each object of the subclass also "is a(n)" object of the superclass.



- Subclass:
  - A class that is derived from another class is called a *subclass* (also a *derived class*, *extended class*, or *child class*).
  - Child class which inherits properties of the parent class and defines its own.
- Superclass:
  - The class from which the subclass is derived is called a *superclass* (also a *base class* or a *parent class*).
  - Parent class with some functionality.

```
class  ClassName
[extends SuperClassName ]
[ implements Interface ] {

            [declaration of member
elements ]

            [ declaration of methods ]
            }
```

```
class Superclass
{
 // Superclass attributes
 // Superclass methods
}
class Subclass extends Superclass
{
 // Subclass attributes
 // Subclass methods
}
```

Inheritance in Java

- Specify one superclass for any subclass
- Java does not support the inheritance of multiple superclasses into a single subclass.
- No class can be a superclass of itself.

Why Inheritance?

Why Inheritance?

- **Code reuse**
- **Extensibility**
- **Ease of modification**
- **Logical structures and grouping**
- **Protected visibility**

Why Inheritance?

- **Code reuse**:
  - Used to eliminate redundant coding
  - To reuse the functionality of a class like reusing function libraries.
  - Smaller derived class definitions
- **Extensibility**
  - new functionality can be added by extending new class and thereby ensures adaptability of the system
- **Ease of modification to common properties and behaviour**
  - Common properties and behaviors can be defined in superclass and shared among subclasses.
  - Hence, modification or changes need to be done one place rather than at multiple places
- **Logical structures and grouping:**
  - Grouping classes and packages enhances readability and maintainability
- **Protected visibility**
  - Protected access modifier provides access to inherited members
  - Promotes encapsulation and security

Modifiers

- **Public** – Accessible by any other class in any package.
- **Private** – Accessible only within the class. Hidden from all sub classes
- **Protected** – Accessible only by classes within the same package and any subclasses in other packages.
    - (For this reason, some choose not to use protected, but use private with accessors)
- Default (No Modifier) – Accessible by classes in the same package but not by classes in other packages.

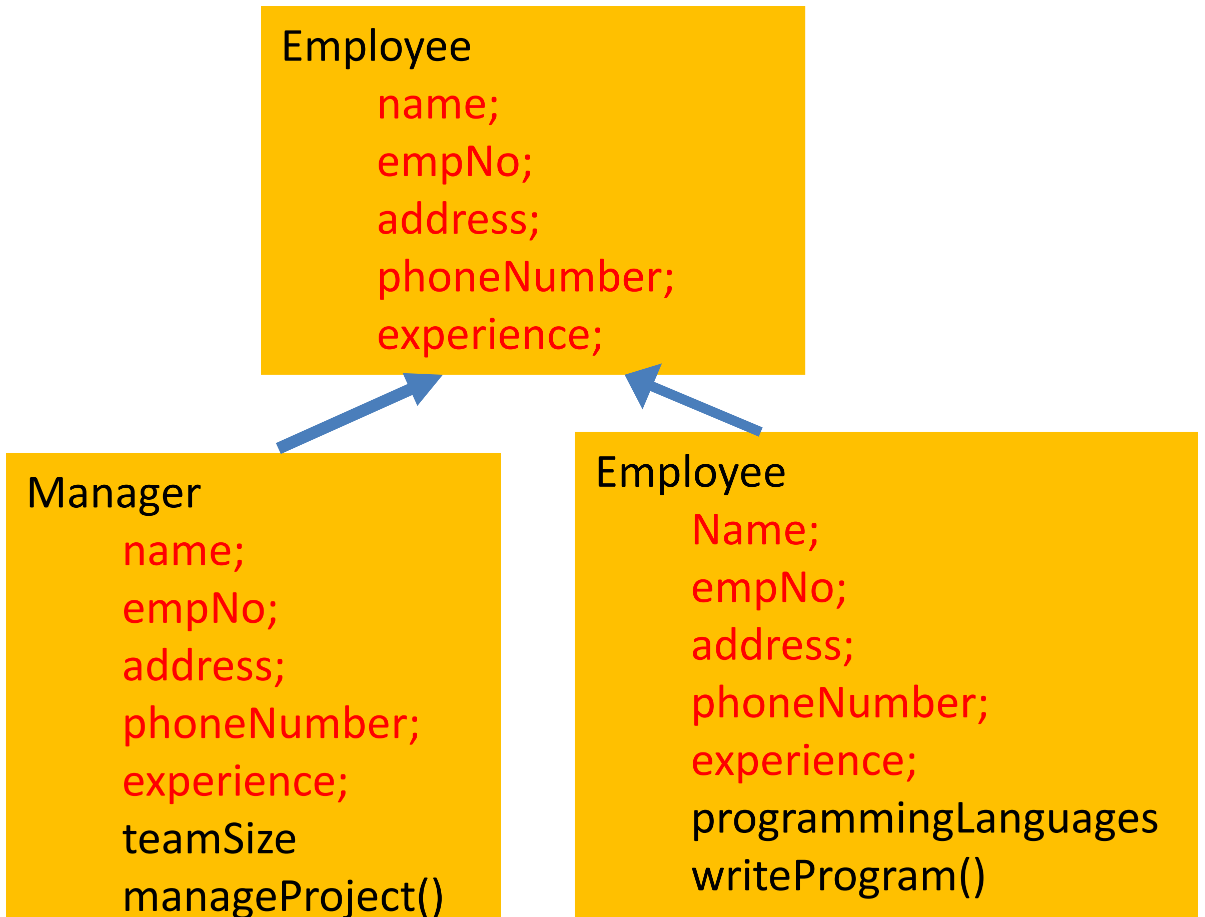
The need to inherit classes

Properties and behavior of a Programmer and a Manager, together with their representations as classes

```
class Programmer {
  String name;
  long empNo;
  String address;
  String phoneNumber;
  int experience;
  String[] ProgrammingLanguages
 void writeProgram()
}
```

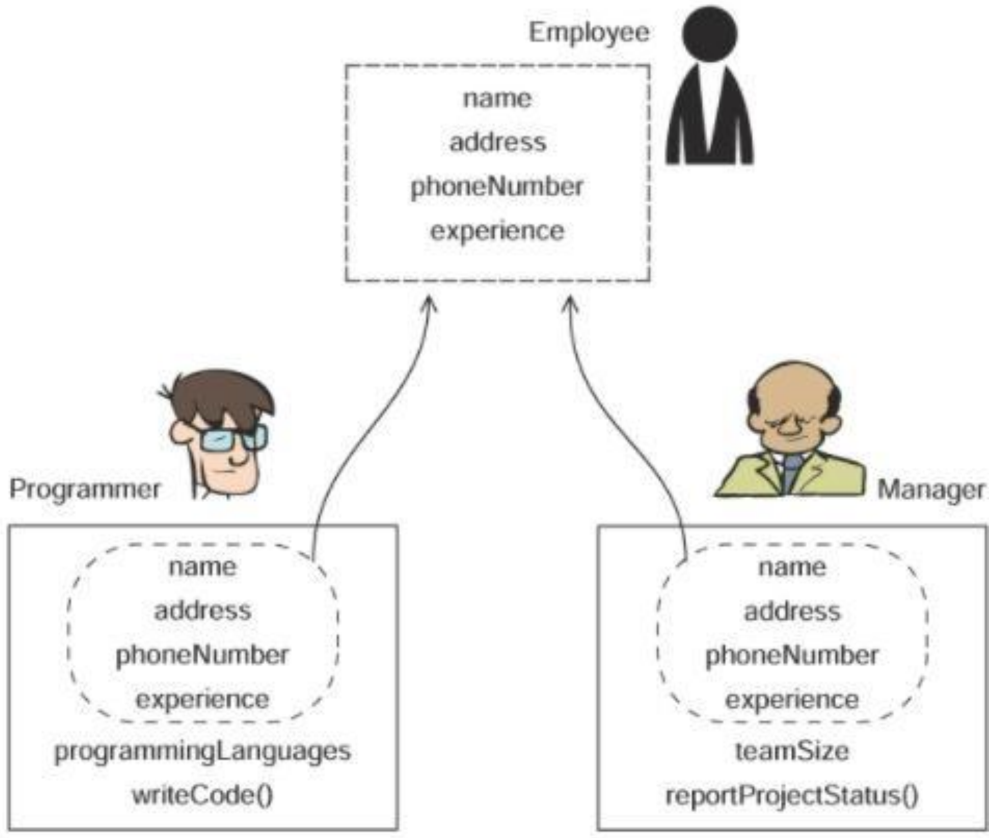
```
class Manager {
  String name;
  long empNo;
  String address;
  String phoneNumber;
  int experience;
  int teamSize
 void manageProject()
}
```

Programmer and Manager have common properties, namely, name, address, phoneNumber, and experience



The need to inherit classes

Identify common properties and behaviors of a Programmer and a Manager, pull them out into a new position, and name it Employee.

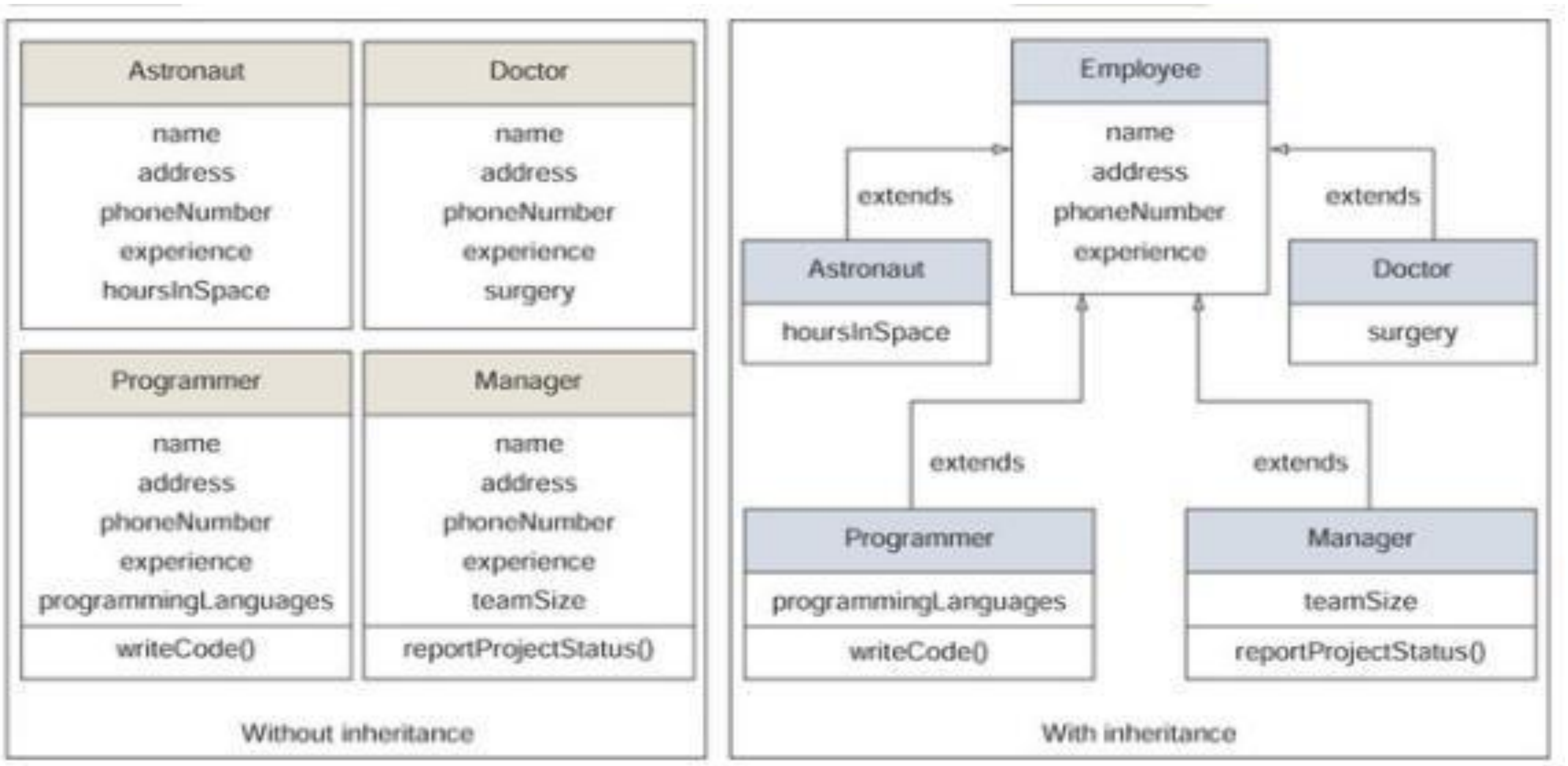


Source: Manning Text book

Programmer and Manager have common properties, namely, name, address, phoneNumber, and experience

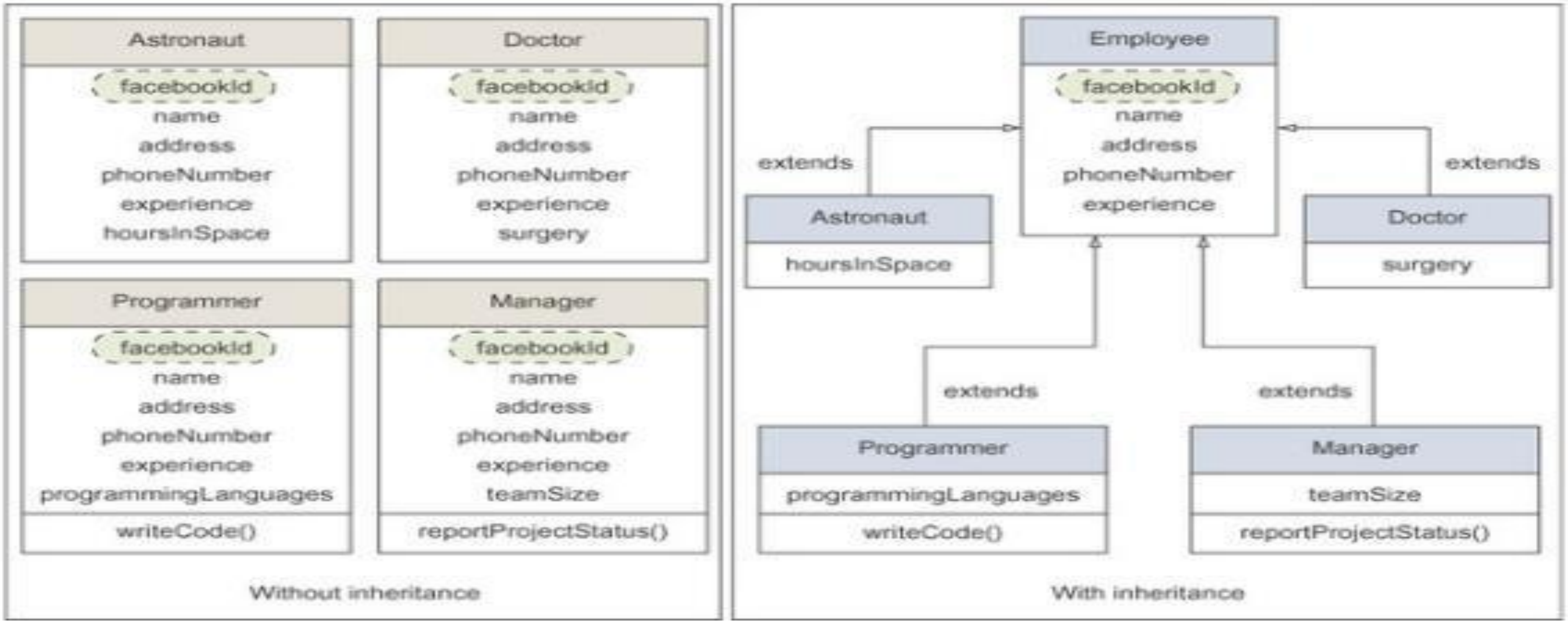
```
class Employee{
 String name;
long empNo
 String address;
String phoneNumber;
 int experience;

}
class Programmer extends Employee{
 String[] programmingLanguages;
  void writeProgram(){
      …
}
}
```

Extensibility: Smaller derived class definitions



Source: Manning Text book

Ease of modification to common properties and behavior



Adding a new property, facebookId, to all classes, with and without the base class Employee

Source: Manning Text book

What all is inherited?

- A derived class method can access
    - All public member functions and fields of base
    - All protected member functions and fields of base
    - All methods and fields of itself
    - Subclass can add new methods and fields.
    - Constructors can be invoked by the subclass
- A subclass or derived class method cannot access
    - Any private methods or fields of base
    - Any protected or private members of any other class
    - Constructors are not inherited

Super

```
super.method(parameters)
 super(parameters);
```

- Constructors are not inherited, even though they have public visibility
- The `super` reference can be used to refer to the parent class, and is used to invoke the parent's constructor
  1. To call a parent's method, use **super.*methodName*(…)**
  2. To call a parent's constructor, use super(some parameter) from the child class' constructor
- still use *this* (super not needed) to access parent's fields: **this.parentVar**

What You Can Do in a Subclass?

- A subclass inherits all of the *public* and *protected* members of its parent, no matter what package the subclass is in.
- If the subclass is in the same package as its parent, it also inherits the *package-private* members of the parent.
  - The inherited fields can be used directly, just like any other fields.
  - You can **declare a field in the subclass with the same name** as the one in the superclass, thus *hiding* it (not recommended).
  - You can **declare new fields in the subclass** that are not in the superclass.
  - The inherited methods can be used directly as they are.
  - You can **write a new *instance* method in the subclass that has the same signature as the one in the superclass, thus *overriding*** it.
  - You can write a **new *static* method in the subclass that has the same signature as the one in the superclass, thus *hiding*** it.
  - You can declare **new methods in the subclass that are not in the superclass**.
  - You can write **a subclass constructor that invokes the constructor of the superclass, either implicitly or by using the keyword super**.

Private Members in a Superclass

- Private Members in a Superclass
- A subclass does not inherit the private members of its parent class.
- However, if the superclass has public or protected methods for accessing its private fields, these can also be used by the subclass.
- A nested class has access to all the private members of its enclosing class—both fields and methods.
- Therefore, a public or protected nested class inherited by a subclass has indirect access to all of the private members of the superclass.

```
/* A simple example of inheritance. */
// superclass.
class A {
        int i, j;
        void showij() {
                System.out.println("i and j: " + i + " " + j);
        }
}
// subclass by extending class A.
class B extends A {
        int k;
        void showk() {
                System.out.println("k: " + k);
        }
        void sum() {
                System.out.println("i+j+k: " + (i + j + k));
        }
}
```

```
class Demo{
        public static void main(String args[]) {
                A superOb = new A();
                B subOb = new B();
                // The superclass may be used by itself.
                superOb.i = 10;
                superOb.j  = 20;
                System.out.println("Contents of superOb: ");
                superOb.showij();
                System.out.println();
        /* The subclass has access to all public members of its superclass. */
                subOb.i = 7;
                subOb.j = 8;
                subOb.k = 9;
                System.out.println("Contents of subOb: ");
                subOb.showij();
                subOb.showk();
                System.out.println();
                System.out.println("Sum of i, j and k in subOb: ");
                subOb.sum();
        }   }
```

OUTPUT:

```
Contents of superOb:
i and j: 10 20
Contents of subOb:
i and j: 7 8
k: 9
Sum of i, j and k in subOb:
i+j+k: 24
```

```java
/* Simple example of access modifier.
This program will not compile. */


// Create a superclass.
class A {
		int i; // default
		private int j; // private to A

		void setij(int x, int y) {
				i = x;
				j = y;
		}
}
```

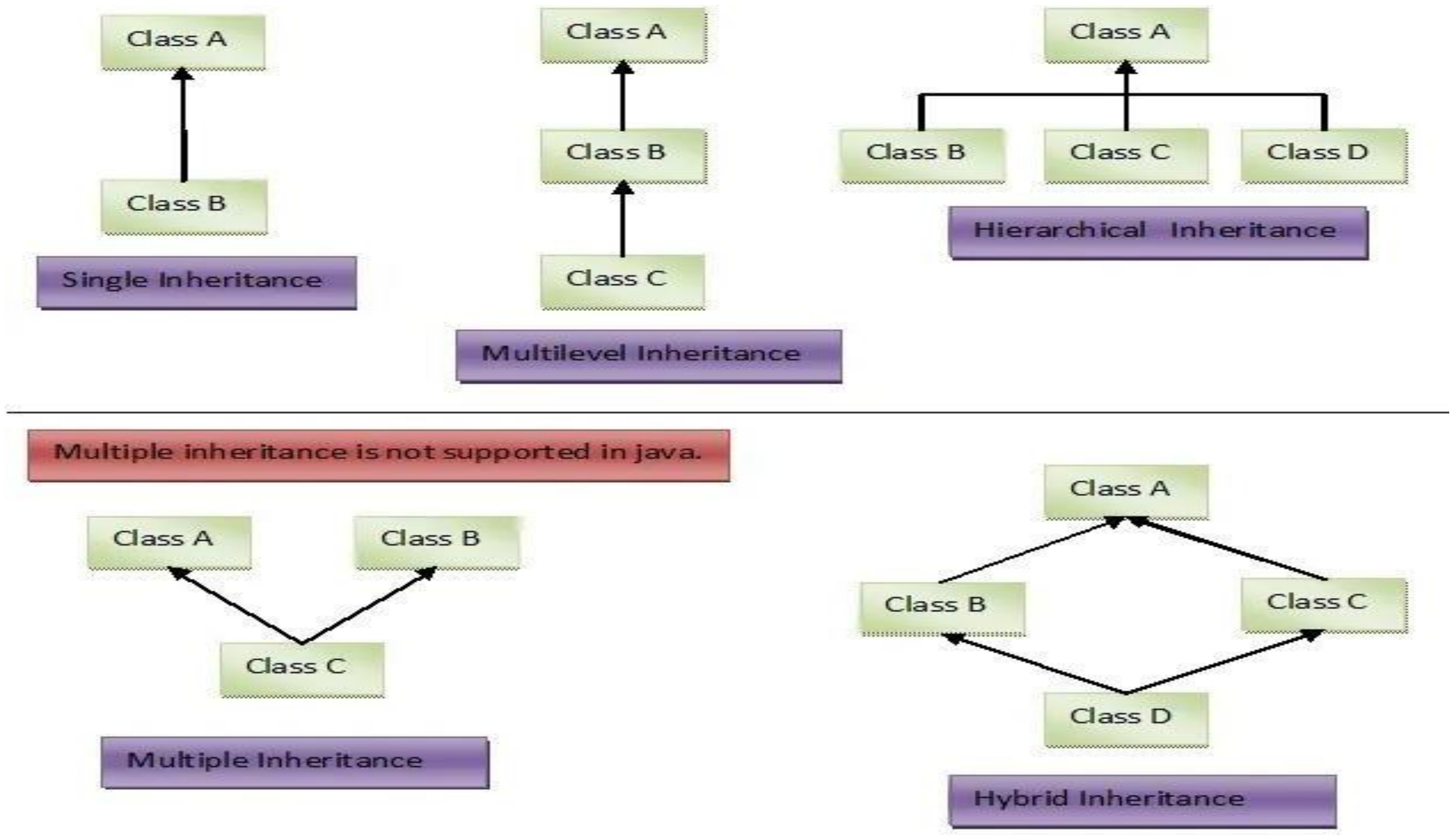
```java
// A's j is not accessible here.
class B extends A {
		int total;
		void sum() {
				total = i + j; // ERROR, j is not accessible here
		}
}
class Demo{
		public static void main(String args[]) {
				B subOb = new B();
				subOb.setij(10, 12);
				subOb.sum();
				System.out.println("Total is " + subOb.total);
		}
}
```

```
/* Example of access modifier with public, private and protected data */
class BaseClass {
        public int x = 10;
        private int y = 10;
        protected int z = 10;
        int a = 10;
        public int getX() {
        return x;
}
public void setX(int x) {
        this.x = x;        }
private int getY() {
        return y;          }
private void setY(int y) {
        this.y = y;        }
protected int getZ() {
        return z;          }
protected void setZ(int z) {
        this.z = z;        }
int getA() {
        return a;          }

        this.a = a;        }
}
```

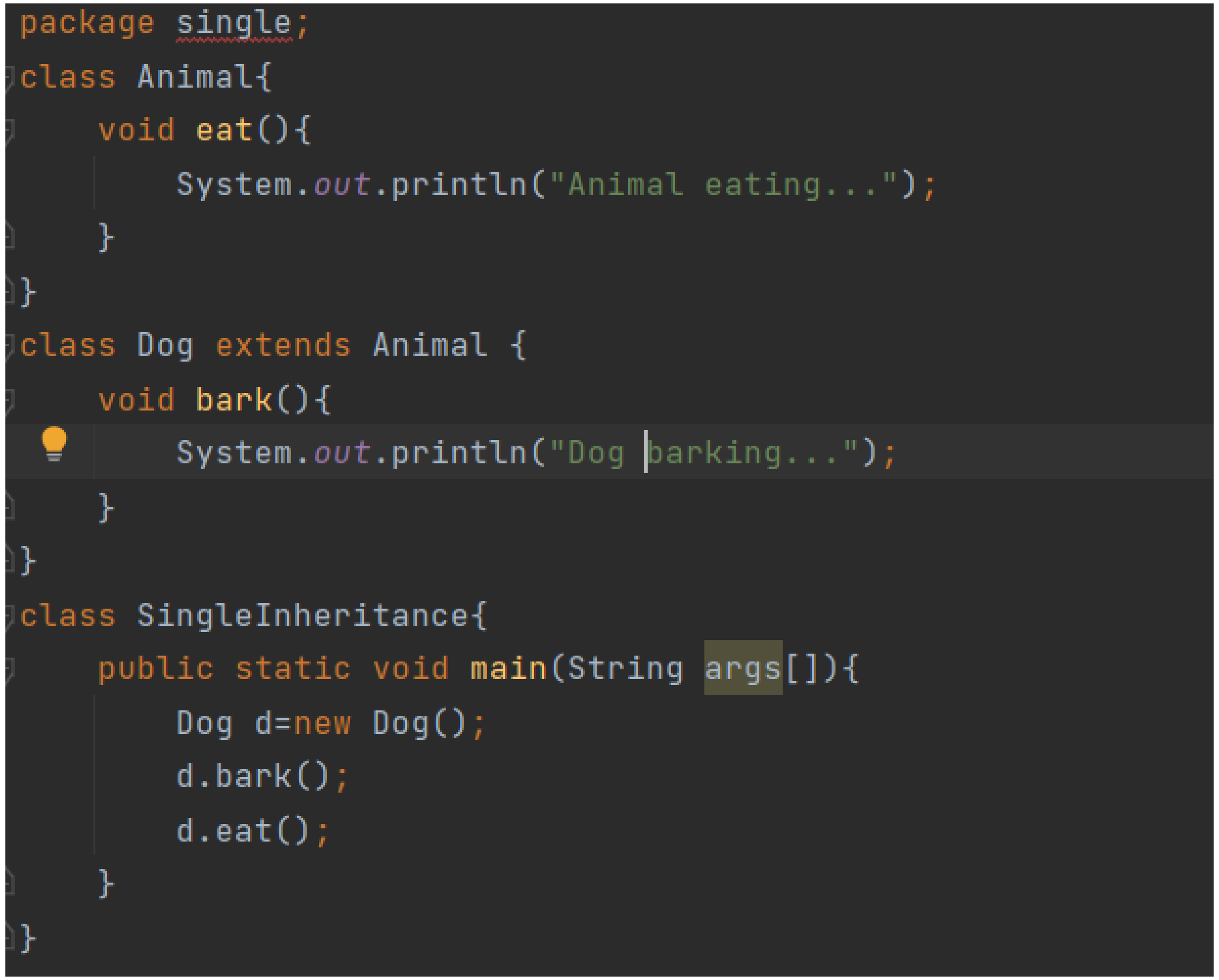
```
OUTPUT:
Value of x is : 10
Value of x is : 20
Value of z is : 10
Value of z is : 30
Value of x is : 10
Value of x is : 20
```

```
public class Demo extends BaseClass {
    public static void main(String args[]) {
            BaseClass Br = new BaseClass();
            Br.z = 0;
            Demo subClassObj = new Demo();
            //Access  Modifiers - Public
            System.out.println("Value of x is : " + subClassObj.x);
            subClassObj.setX(20);
            System.out.println("Value of x is : " + subClassObj.x);
            // Access Modifiers - Private
            // compilaton error as the fields and methods are private

            /* System.out.println("Value of y is : "+subClassObj.y);
            subClassObj.setY(20);
            System.out.println("Value of y is : "+subClassObj.y);*/

            //Access Modifiers - Protected
            System.out.println("Value of z is : " + subClassObj.z);
            subClassObj.setZ(30);
            System.out.println("Value of z is : " + subClassObj.z);
            //Access Modifiers - Default
            System.out.println("Value of x is : " + subClassObj.a);
            subClassObj.setA(20);
            System.out.println("Value of x is : " + subClassObj.a);
    }
}
```

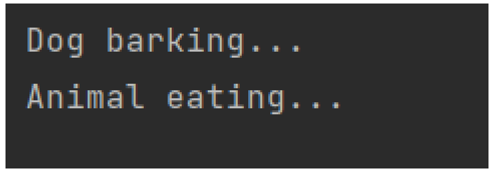
Types of Inheritance



Single Inheritance



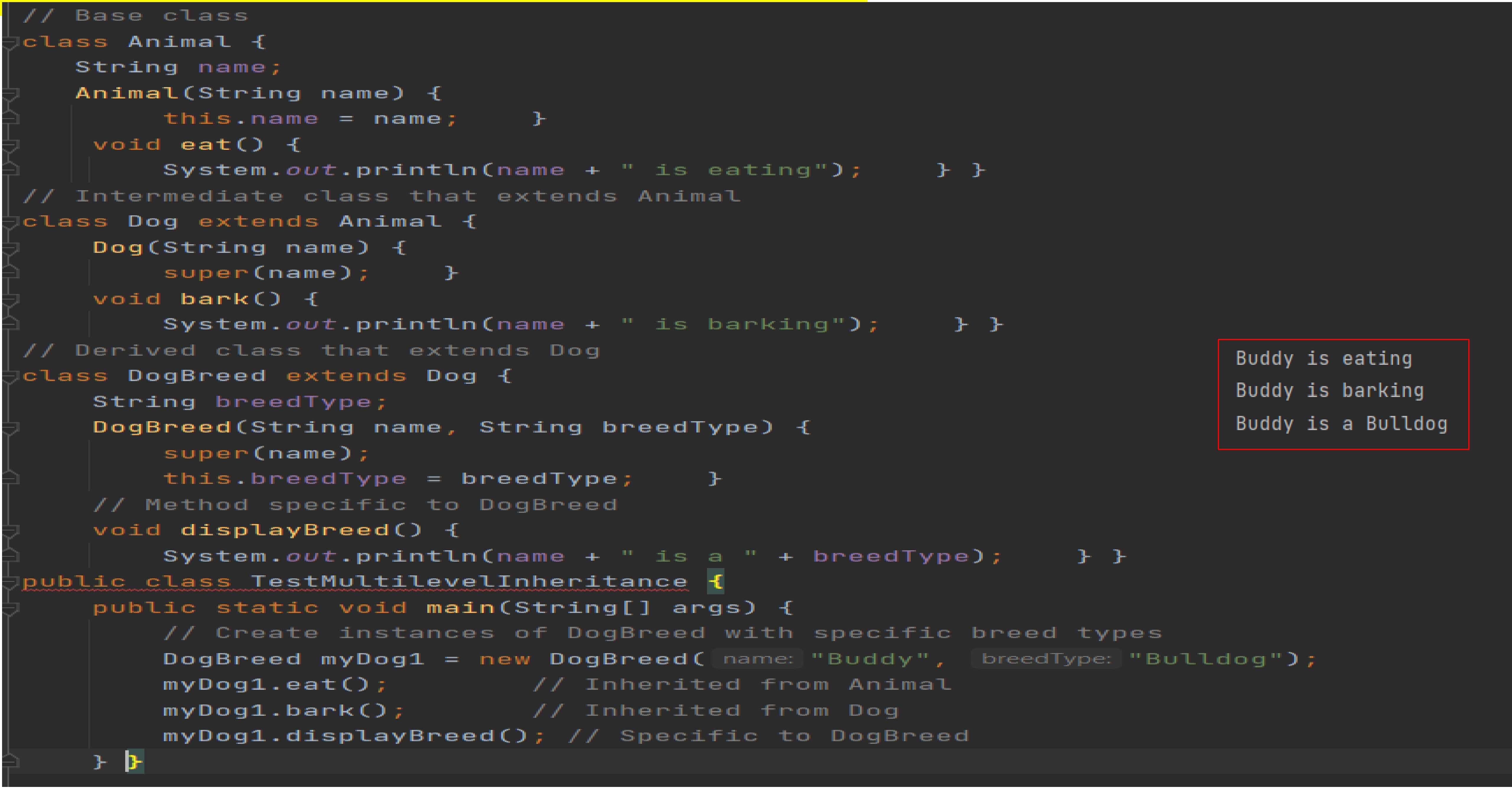
```
package single;
class Animal{
    void eat(){
        System.out.println("Animal eating...");
    }
}
class Dog extends Animal {
    void bark(){
        System.out.println("Dog barking...");
    }
}
class SingleInheritance{
    public static void main(String args[]){
        Dog d=new Dog();
        d.bark();
        d.eat();
    }
}
```



```
Dog barking...
Animal eating...
```

A class (subclass or derived class) inherits from one and only one parent class (superclass or base class).
A subclass inherits fields and methods from a single superclass

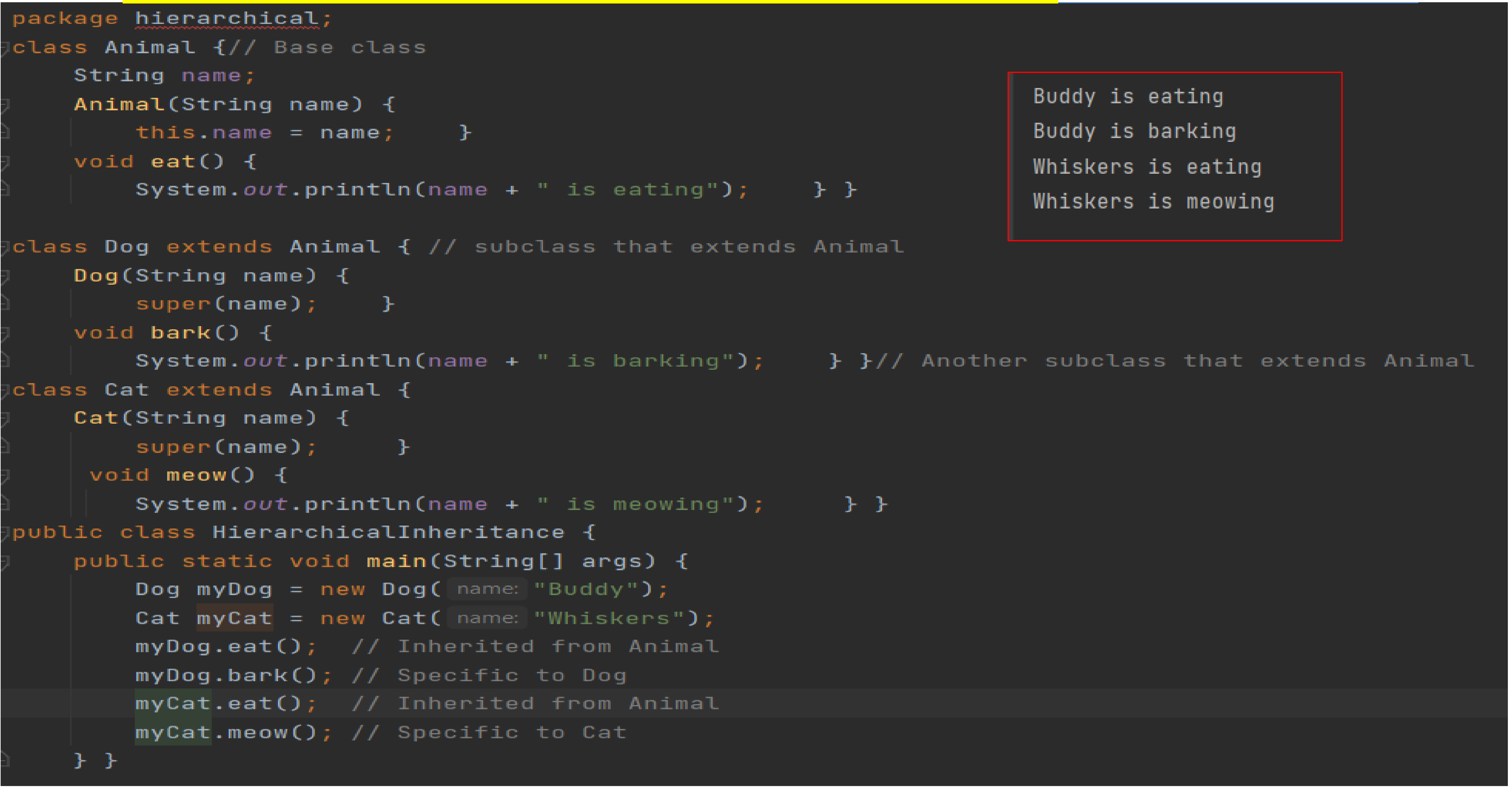
MultiLevel Inheritance



```java
// Base class
class Animal {
   String name;
   Animal(String name) {
        this.name = name;      }
    void eat() {
        System.out.println(name + " is eating");      } }
// Intermediate class that extends Animal
class Dog extends Animal {
    Dog(String name) {
        super(name);      }
    void bark() {
        System.out.println(name + " is barking");      } }
// Derived class that extends Dog
class DogBreed extends Dog {
    String breedType;
    DogBreed(String name, String breedType) {
        super(name);
        this.breedType = breedType;      }
    // Method specific to DogBreed
    void displayBreed() {
        System.out.println(name + " is a " + breedType);      } }
public class TestMultilevelInheritance {
    public static void main(String[] args) {
        // Create instances of DogBreed with specific breed types
        DogBreed myDog1 = new DogBreed( name: "Buddy",  breedType: "Bulldog");
        myDog1.eat();          // Inherited from Animal
        myDog1.bark();         // Inherited from Dog
        myDog1.displayBreed(); // Specific to DogBreed
    } }
```

```
Buddy is eating
Buddy is barking
Buddy is a Bulldog
```

Hierarchical Inheritance



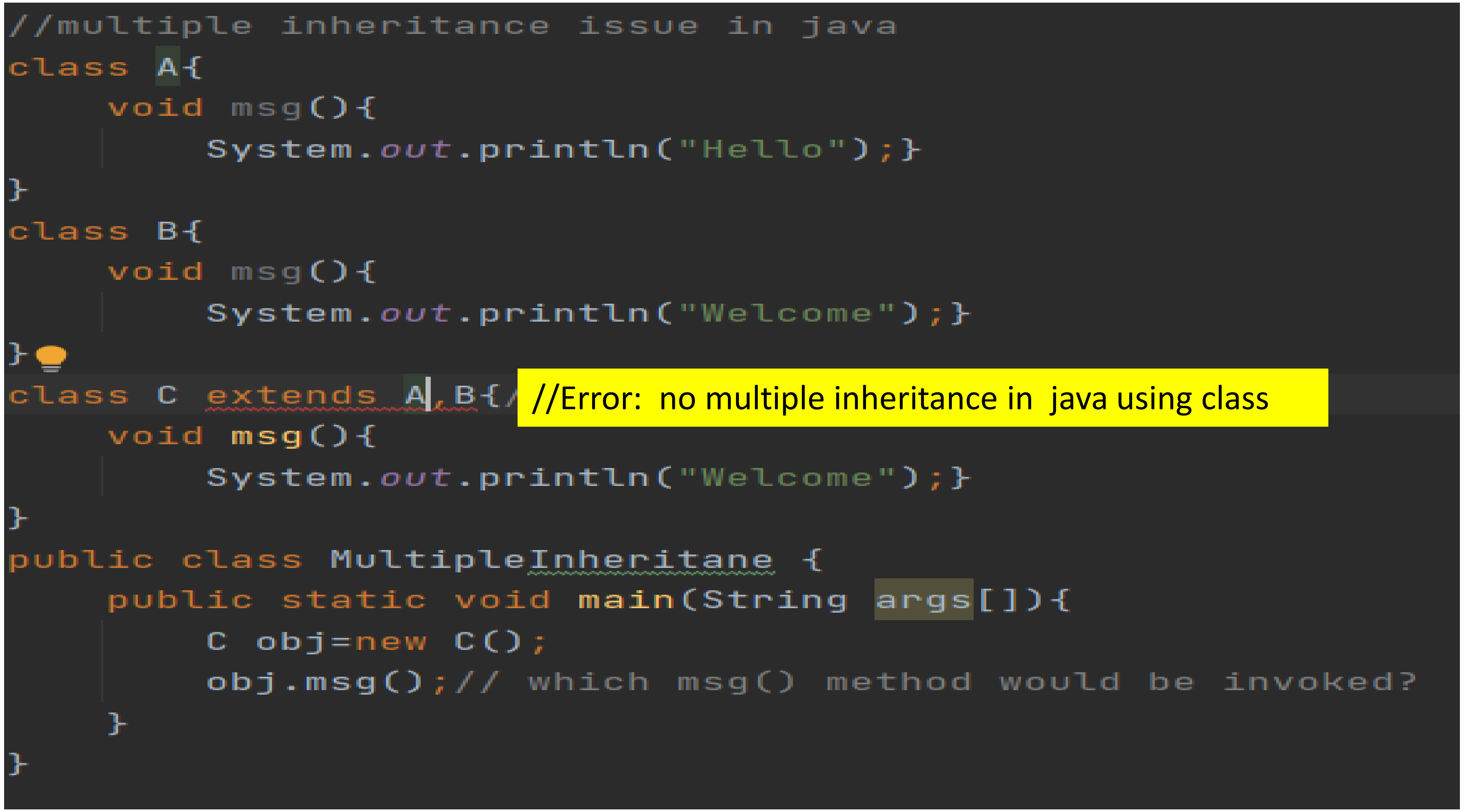
```
package hierarchical;
class Animal {// Base class
    String name;
    Animal(String name) {
        this.name = name;     }
    void eat() {
        System.out.println(name + " is eating");     } }

class Dog extends Animal { // subclass that extends Animal
    Dog(String name) {
        super(name);     }
    void bark() {
        System.out.println(name + " is barking");     } }// Another subclass that extends Animal
class Cat extends Animal {
    Cat(String name) {
        super(name);      }
     void meow() {
        System.out.println(name + " is meowing");      } }
public class HierarchicalInheritance {
    public static void main(String[] args) {
        Dog myDog = new Dog( name: "Buddy");
        Cat myCat = new Cat( name: "Whiskers");
        myDog.eat();  // Inherited from Animal
        myDog.bark(); // Specific to Dog
        myCat.eat();  // Inherited from Animal
        myCat.meow(); // Specific to Cat
    } }
```

```
Buddy is eating
Buddy is barking
Whiskers is eating
Whiskers is meowing
```

Multiple inheritance

- Java supports *single inheritance*, meaning that a derived class can have only one parent class
- *Multiple inheritance* allows a class to be derived from two or more classes, inheriting the members of all parents
- Collisions, such as the same variable name in two parents, have to be resolved
- Java does not support multiple inheritance
- In most cases, the use of interfaces gives us aspects of multiple inheritance without the overhead

Multiple inheritance Issue using class



```
//multiple inheritance issue in java
class A{
    void msg(){
        System.out.println("Hello");}
}
class B{
    void msg(){
        System.out.println("Welcome");}
}
class C extends A,B{/
    void msg(){
        System.out.println("Welcome");}
}
public class MultipleInheritane {
    public static void main(String args[]){
        C obj=new C();
        obj.msg();// which msg() method would be invoked?
    }
}
```

//Error: no multiple inheritance in java using class

Super Keyword

- The super keyword is a reference variable which is used to refer immediate parent class object.
- Whenever the instance of subclass created, an instance of parent class is created implicitly which is referred by super reference variable.

Super keyword

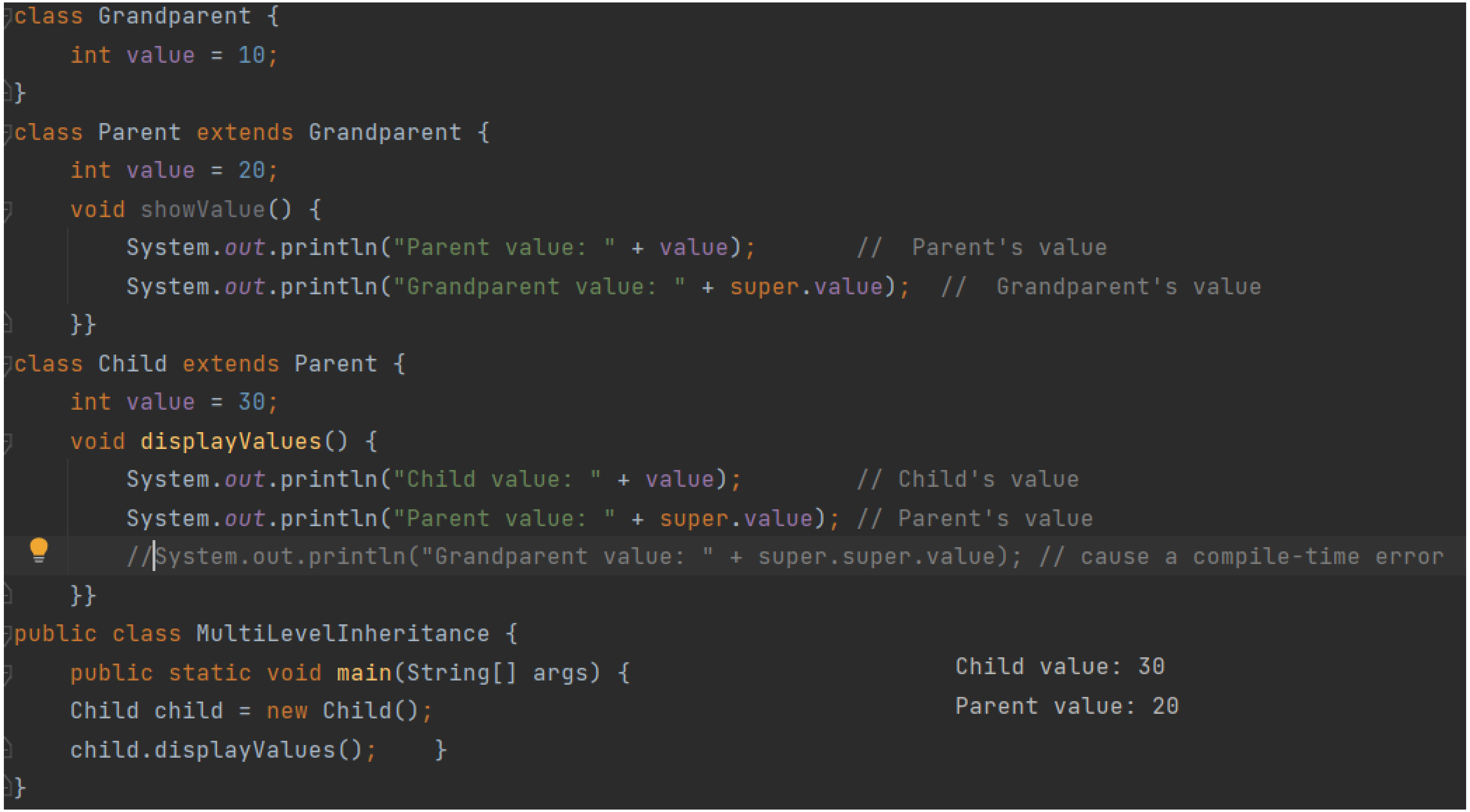
- super can be used to refer instance variable of the immediate parent class. Distinguish between the subclass’s field and the parent class’s field.
- super can be used to invoke immediate parent class method.
- super() can be used to invoke immediate parent class constructor.

Super keyword

```
class Parent {
   int value = 10;
}
class Child extends Parent {
   int value = 20;
   void display() {
//Child class's value
      System.out.println("Child value: " + value);
// Parent class's value
      System.out.println("Parent value: " + super.value);
}
}
```

Super keyword

super cannot be used to directly access variables or methods from a grandparent class or any ancestor beyond the immediate parent class.



```java
class Grandparent {
    int value = 10;
}
class Parent extends Grandparent {
    int value = 20;
    void showValue() {
        System.out.println("Parent value: " + value);        //  Parent's value
        System.out.println("Grandparent value: " + super.value);  //  Grandparent's value
    }}
class Child extends Parent {
    int value = 30;
    void displayValues() {
        System.out.println("Child value: " + value);         // Child's value
        System.out.println("Parent value: " + super.value); // Parent's value
        //System.out.println("Grandparent value: " + super.super.value); // cause a compile-time error
    }}
public class MultiLevelInheritance {
    public static void main(String[] args) {
    Child child = new Child();
    child.displayValues();    }
}
```

```
Child value: 30
Parent value: 20
```

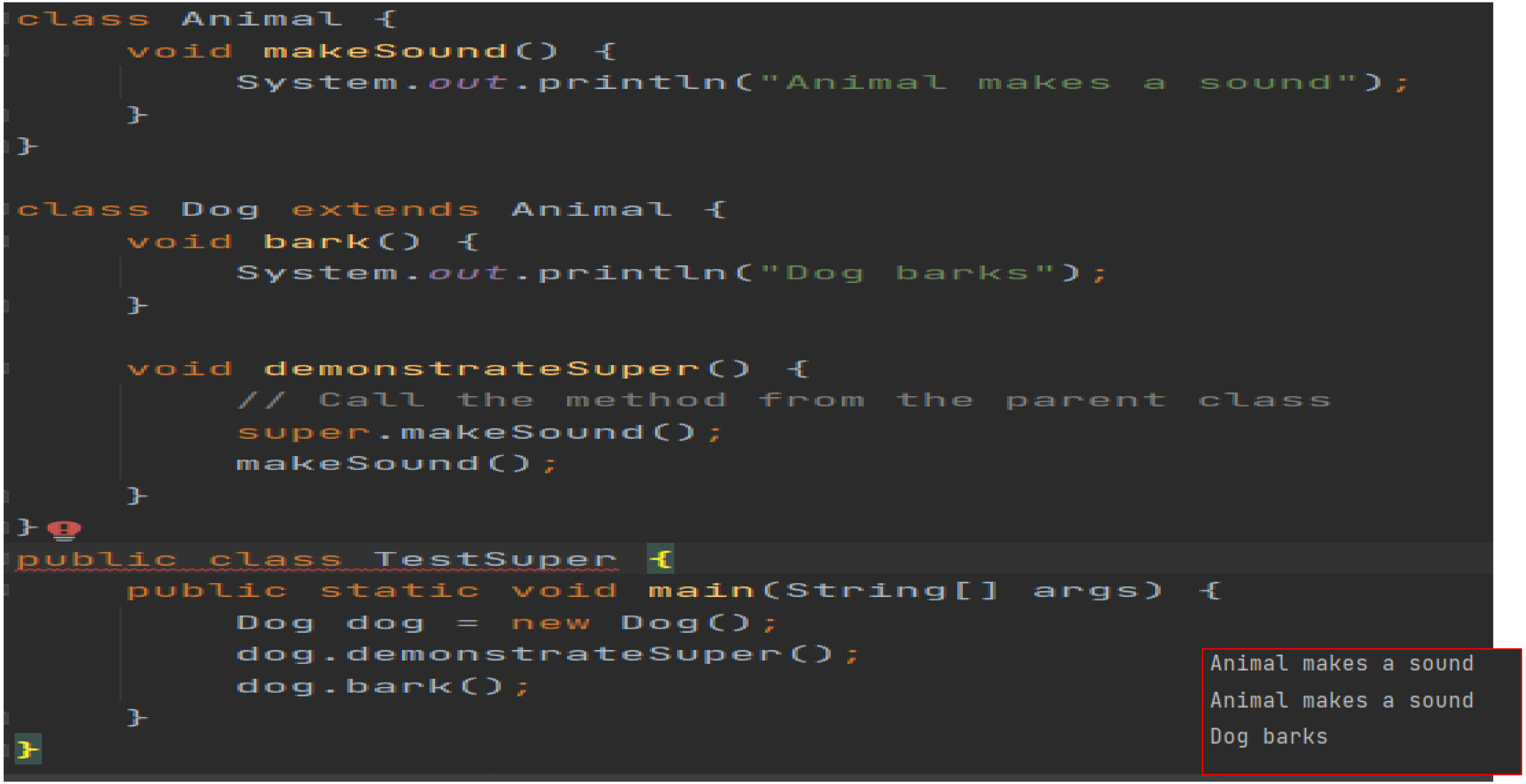
Super keyword

super.methodName() to call the parent class's version of that method from within the subclass. It should be used if subclass contains the same method as parent class.

```
class Animal {
    void makeSound() {
        System.out.println("Animal makes a sound");
    }
}

class Dog extends Animal {
    @Override
    void makeSound() {
        super.makeSound();  // Call Animal's makeSound() method
        System.out.println("Dog barks");
    }
}

public class TestSuperMethod {
    public static void main(String[] args) {
        Dog dog = new Dog();
        dog.makeSound();
    }
}
```

Super keyword

It is still valid to use super.methodName() to call a parent class's method, even if it hasn't been overridden in the subclass



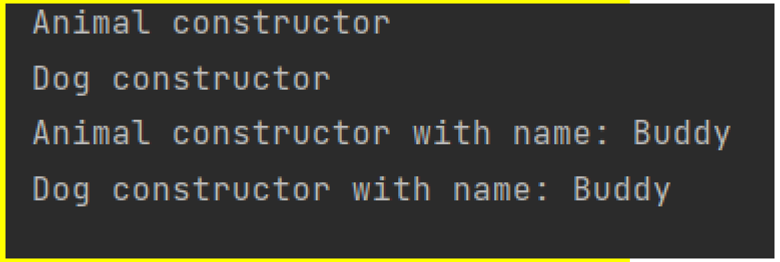
```java
class Animal {
    void makeSound() {
        System.out.println("Animal makes a sound");
    }
}

class Dog extends Animal {
    void bark() {
        System.out.println("Dog barks");
    }

    void demonstrateSuper() {
        // Call the method from the parent class
        super.makeSound();
        makeSound();
    }
}
public class TestSuper {
    public static void main(String[] args) {
        Dog dog = new Dog();
        dog.demonstrateSuper();
        dog.bark();
    }
}
```

```
Animal makes a sound
Animal makes a sound
Dog barks
```

Super keyword

The super() call must be the first statement in the subclass constructor. You cannot place any other statements before it

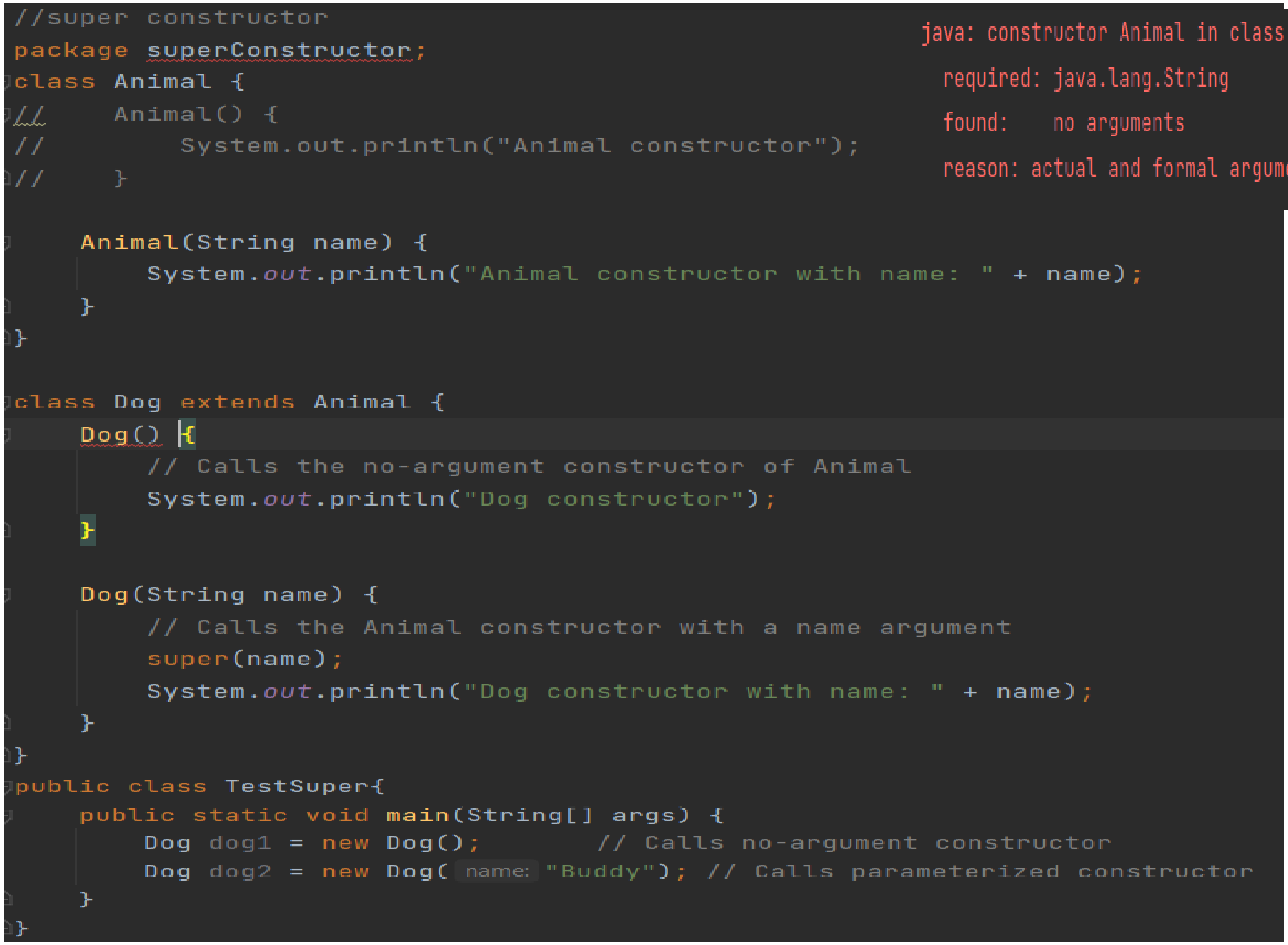
```
class Animal {
   Animal() {
      System.out.println("Animal constructor");
   }
   Animal(String name) {
      System.out.println("Animal constructor with name: " + name);
   }
}
class Dog extends Animal {
   Dog() {// Calls the no-argument constructor of Animal
      System.out.println("Dog constructor");
   }
   Dog(String name) {
      super(name); // Calls the Animal constructor with a name argument
      System.out.println("Dog constructor with name: " + name);
   }
}
public class TestSuperConstructor {
   public static void main(String[] args) {
      Dog dog1 = new Dog();       // Calls no-argument constructor
      Dog dog2 = new Dog("Buddy"); // Calls parameterized constructor
   } }
```



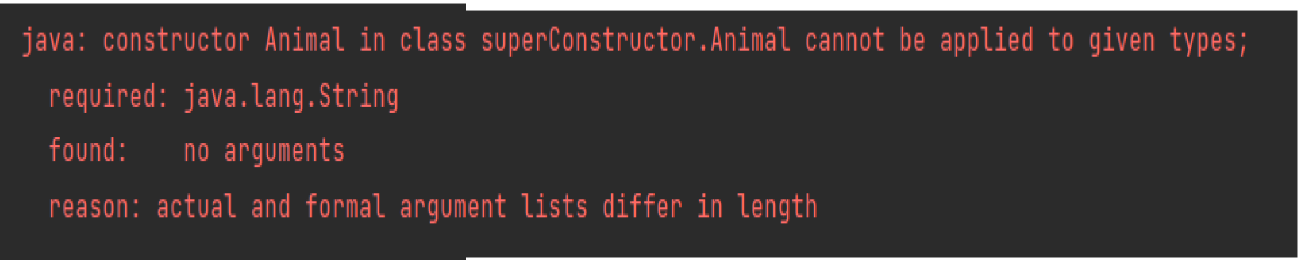
```
Animal constructor
Dog constructor
Animal constructor with name: Buddy
Dog constructor with name: Buddy
```

Super keyword

The super() call must be the first statement in the subclass constructor. You cannot place any other statements before it



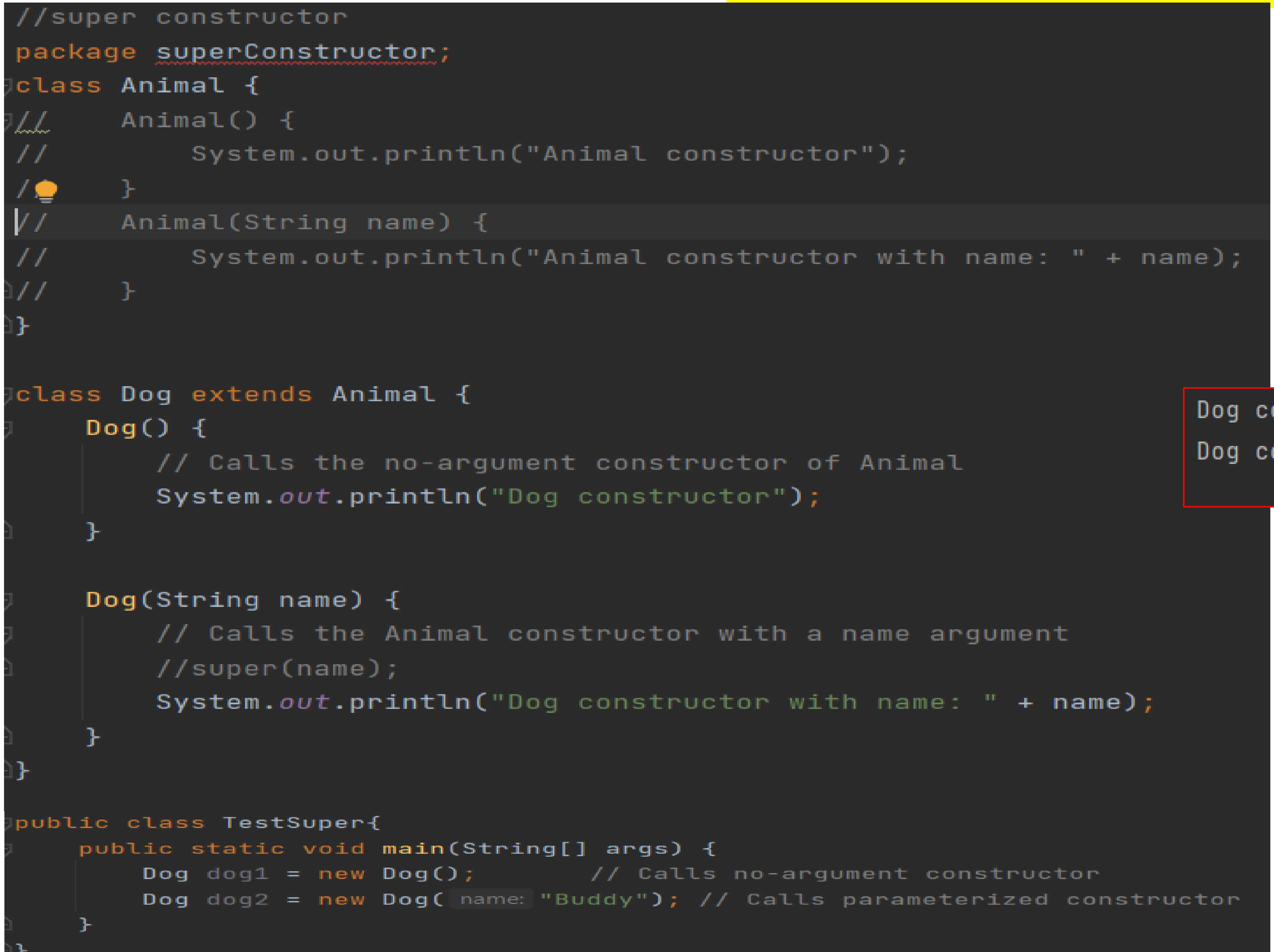
```java
//super constructor
package superConstructor;
class Animal {
//    Animal() {
//        System.out.println("Animal constructor");
//    }

    Animal(String name) {
        System.out.println("Animal constructor with name: " + name);
    }
}

class Dog extends Animal {
    Dog() {
        // Calls the no-argument constructor of Animal
        System.out.println("Dog constructor");
    }

    Dog(String name) {
        // Calls the Animal constructor with a name argument
        super(name);
        System.out.println("Dog constructor with name: " + name);
    }
}
public class TestSuper{
    public static void main(String[] args) {
        Dog dog1 = new Dog();          // Calls no-argument constructor
        Dog dog2 = new Dog( name: "Buddy"); // Calls parameterized constructor
    }
}
```



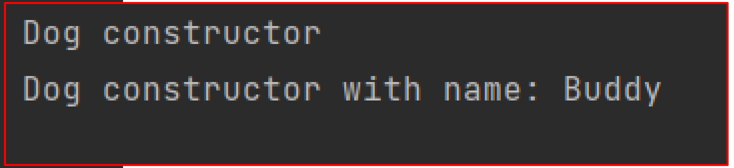
```
java: constructor Animal in class superConstructor.Animal cannot be applied to given types;
  required: java.lang.String
  found:    no arguments
  reason: actual and formal argument lists differ in length
```

Super keyword

The super() call must be the first statement in the subclass constructor. You cannot place any other statements before it



```
//super constructor
package superConstructor;
class Animal {
//    Animal() {
//        System.out.println("Animal constructor");
//    }
//    Animal(String name) {
//        System.out.println("Animal constructor with name: " + name);
//    }
}

class Dog extends Animal {
    Dog() {
        // Calls the no-argument constructor of Animal
        System.out.println("Dog constructor");
    }

    Dog(String name) {
        // Calls the Animal constructor with a name argument
        //super(name);
        System.out.println("Dog constructor with name: " + name);
    }
}

public class TestSuper{
    public static void main(String[] args) {
        Dog dog1 = new Dog();          // Calls no-argument constructor
        Dog dog2 = new Dog( name: "Buddy"); // Calls parameterized constructor
    }
}
```



```
Dog constructor
Dog constructor with name: Buddy
```

Final keyword

The final keyword is used in various context in java.

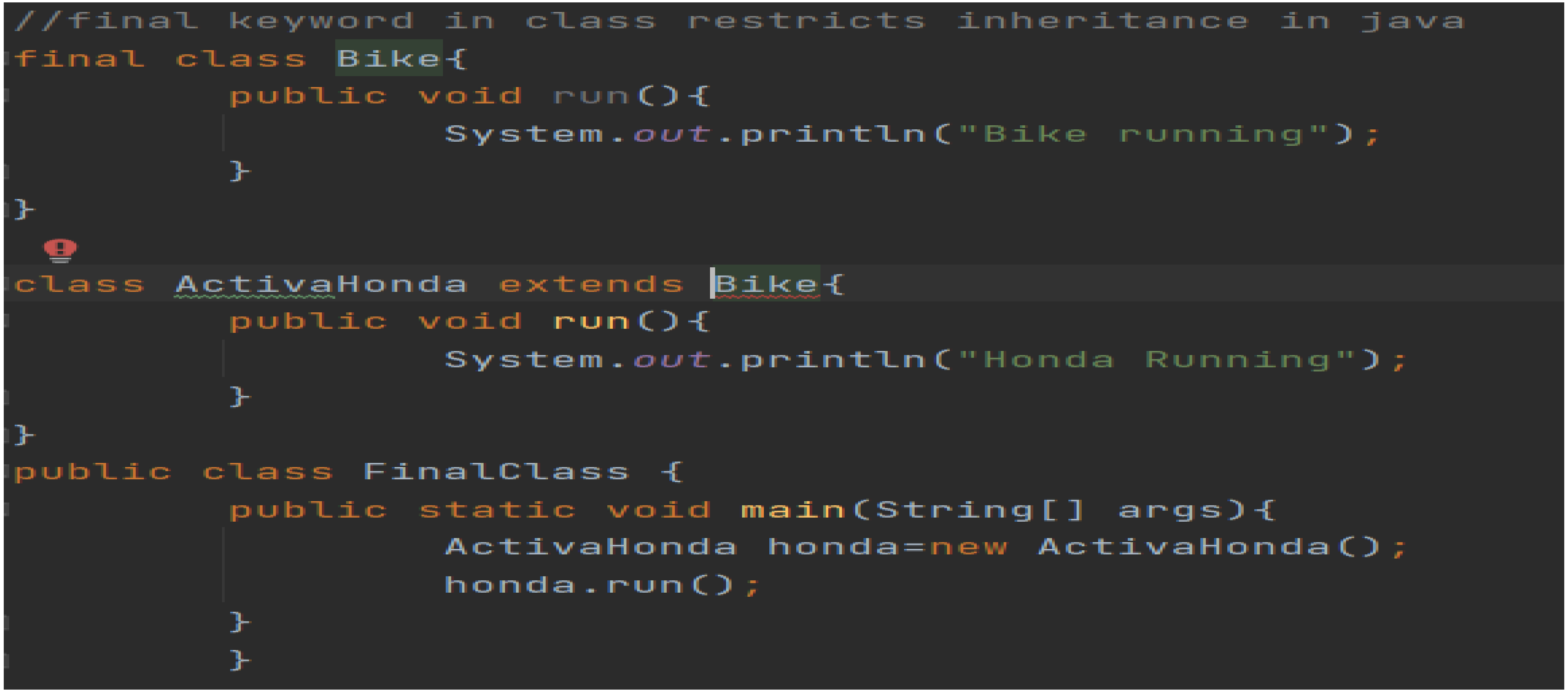
- Class
- Method
- Variable

Final keyword

The final keyword is used in various context in java.

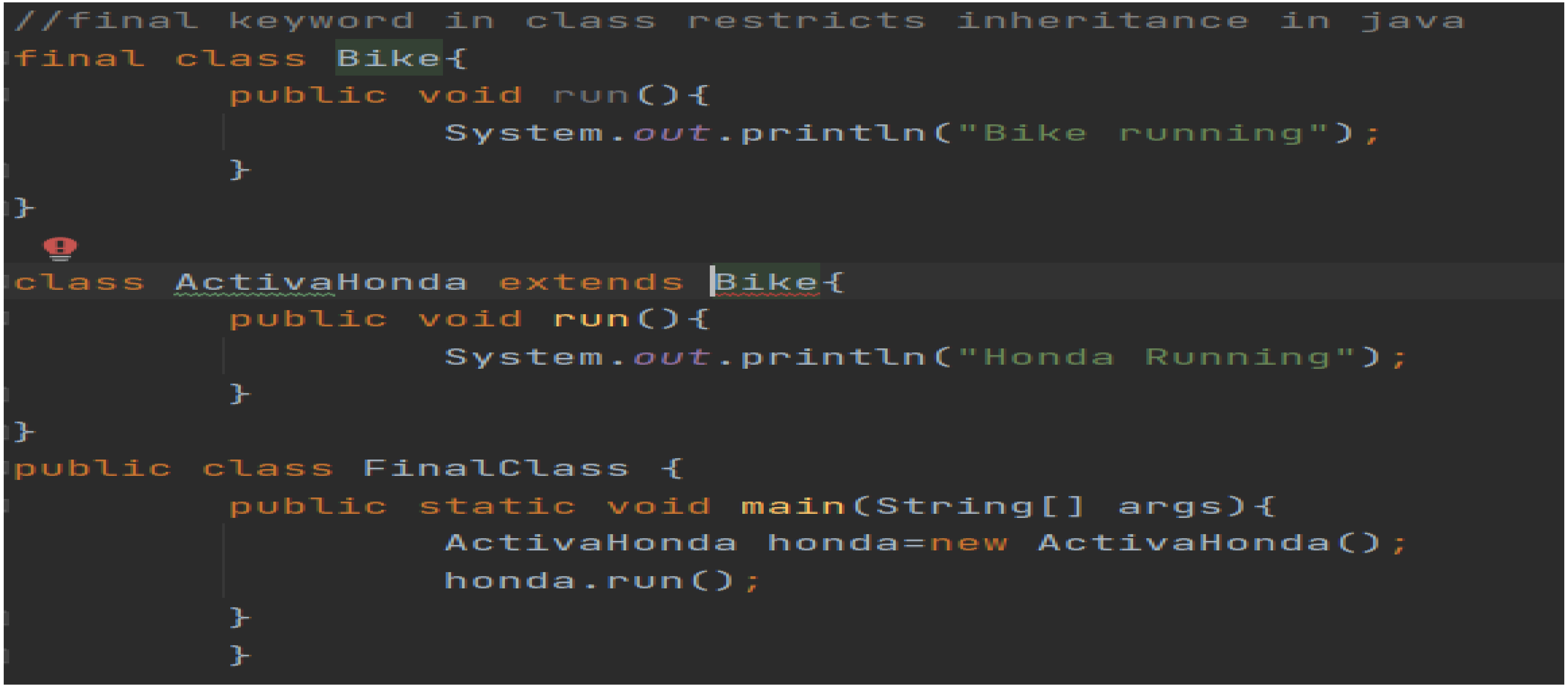
- Class –restrict inheritance
- Method-restrict overriding
- Variable-restrict value change

Final keyword-class

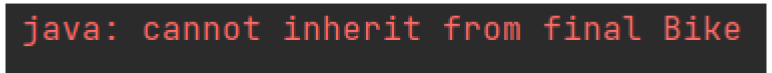


```
//final keyword in class restricts inheritance in java
final class Bike{
        public void run(){
                System.out.println("Bike running");
        }
}

class ActivaHonda extends Bike{
        public void run(){
                System.out.println("Honda Running");
        }
}
public class FinalClass {
        public static void main(String[] args){
                ActivaHonda honda=new ActivaHonda();
                honda.run();
        }
        }
```

Final keyword-class

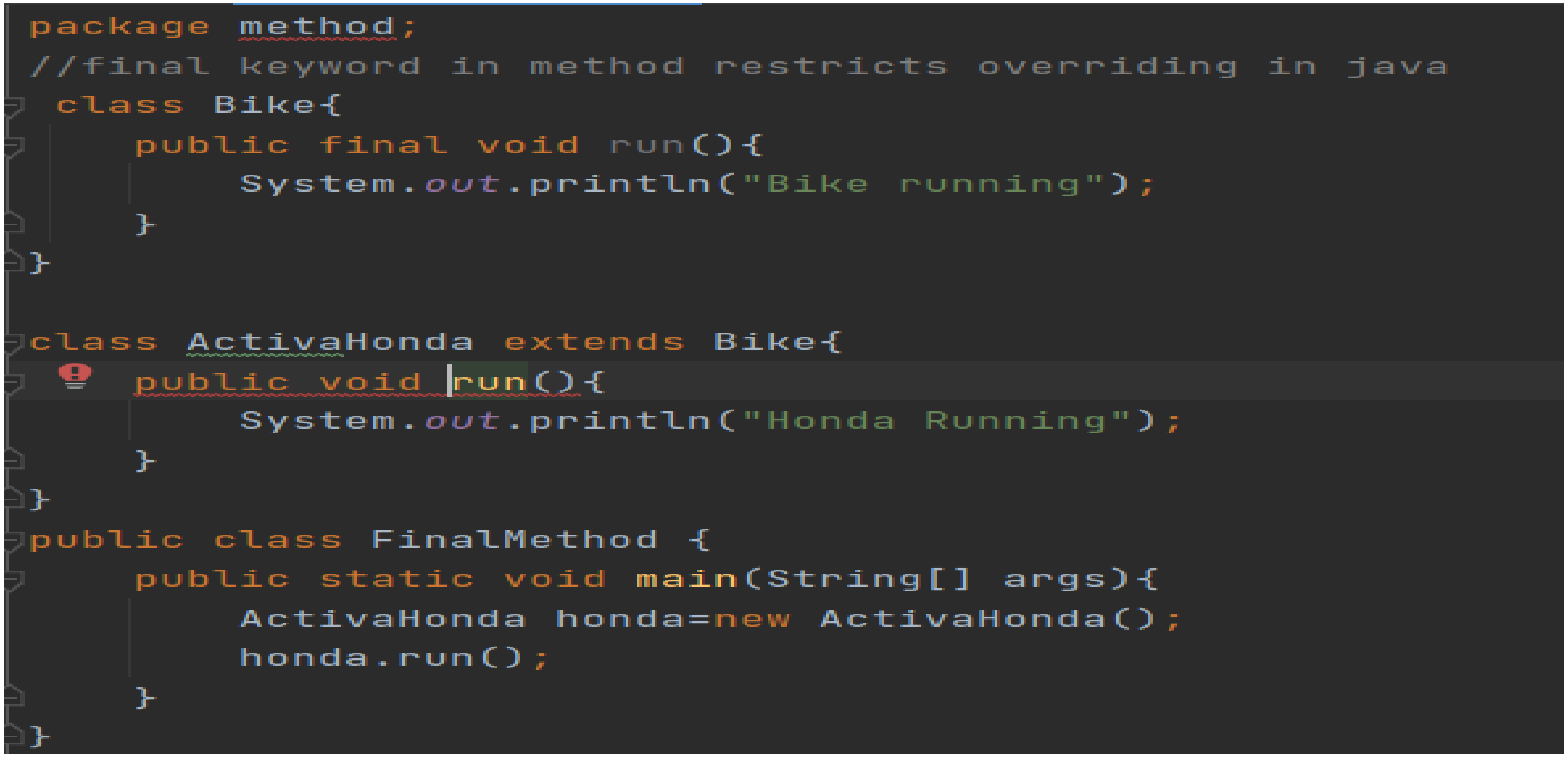


```
//final keyword in class restricts inheritance in java
final class Bike{
        public void run(){
                System.out.println("Bike running");
        }
}

class ActivaHonda extends Bike{
        public void run(){
                System.out.println("Honda Running");
        }
}
public class FinalClass {
        public static void main(String[] args){
                ActivaHonda honda=new ActivaHonda();
                honda.run();
        }
        }
```



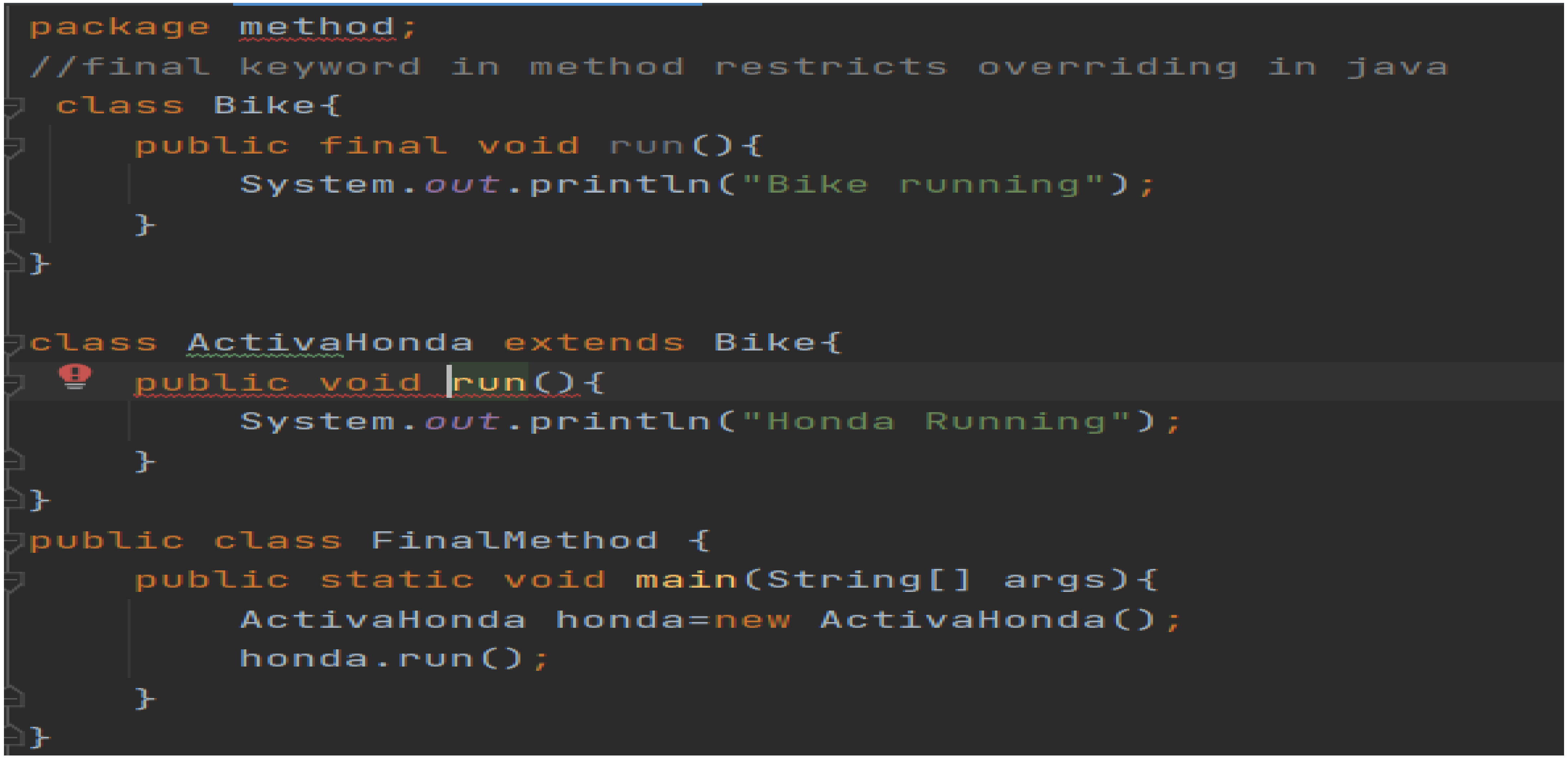
```
java: cannot inherit from final Bike
```

Final keyword-method

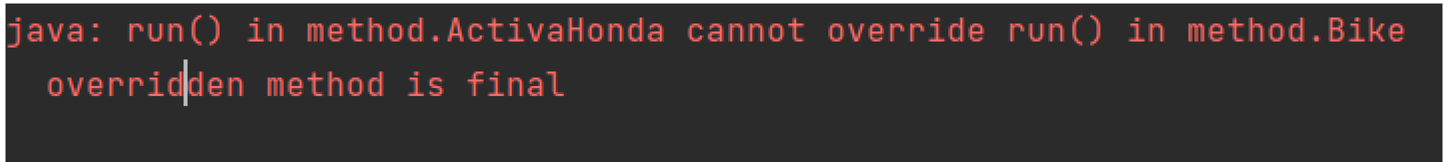


```
package method;
//final keyword in method restricts overriding in java
 class Bike{
    public final void run(){
        System.out.println("Bike running");
    }
}

class ActivaHonda extends Bike{
    public void run(){
        System.out.println("Honda Running");
    }
}
public class FinalMethod {
    public static void main(String[] args){
        ActivaHonda honda=new ActivaHonda();
        honda.run();
    }
}
```

Final keyword-method

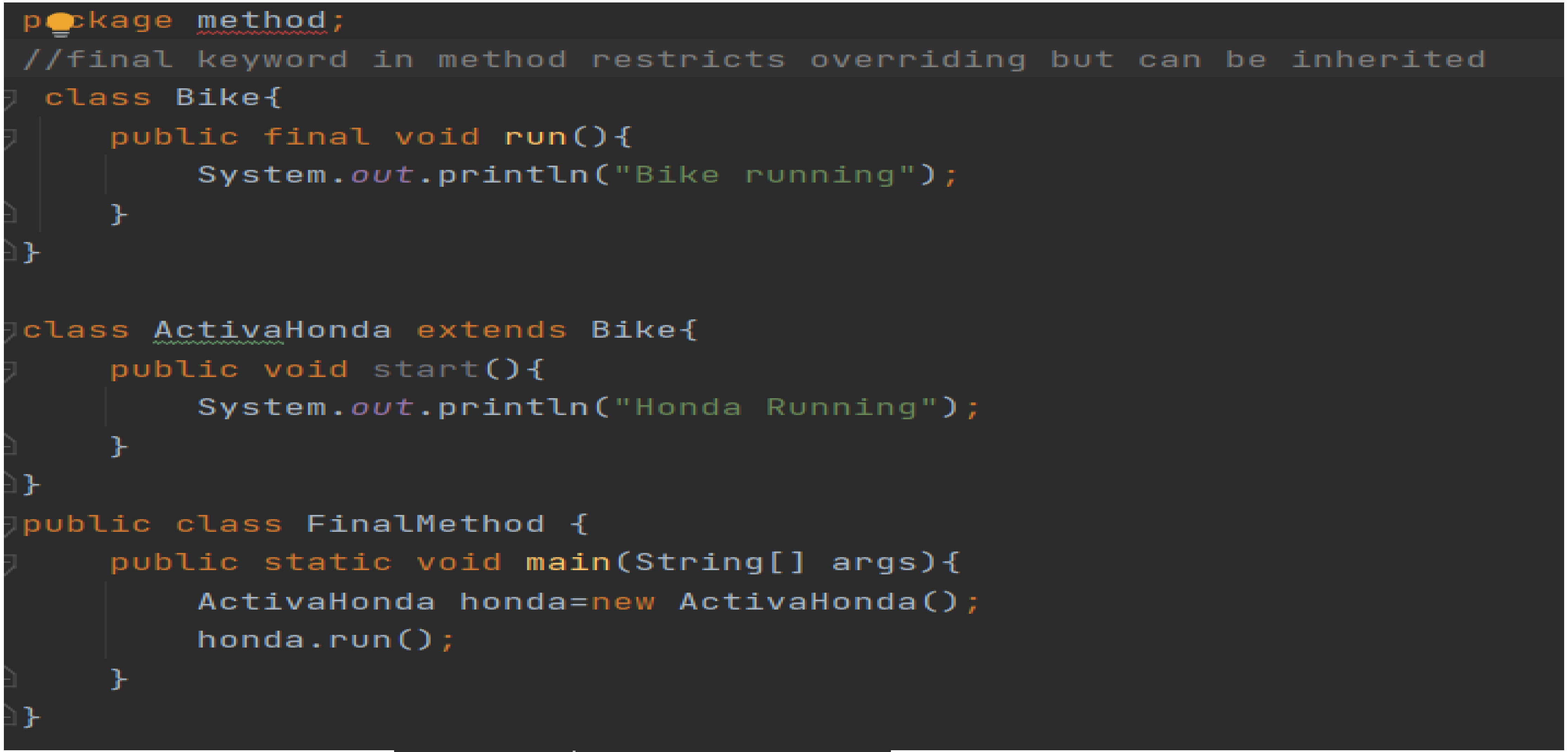


```java
package method;
//final keyword in method restricts overriding in java
 class Bike{
    public final void run(){
        System.out.println("Bike running");
    }
}

class ActivaHonda extends Bike{
    public void run(){
        System.out.println("Honda Running");
    }
}
public class FinalMethod {
    public static void main(String[] args){
        ActivaHonda honda=new ActivaHonda();
        honda.run();
    }
}
```

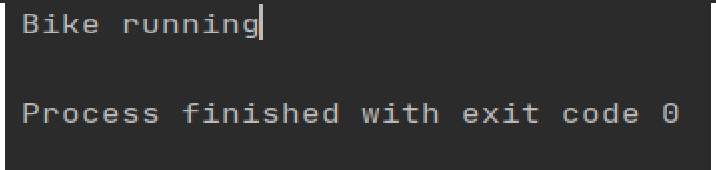


```
java: run() in method.ActivaHonda cannot override run() in method.Bike
  overridden method is final
```

Final keyword-method

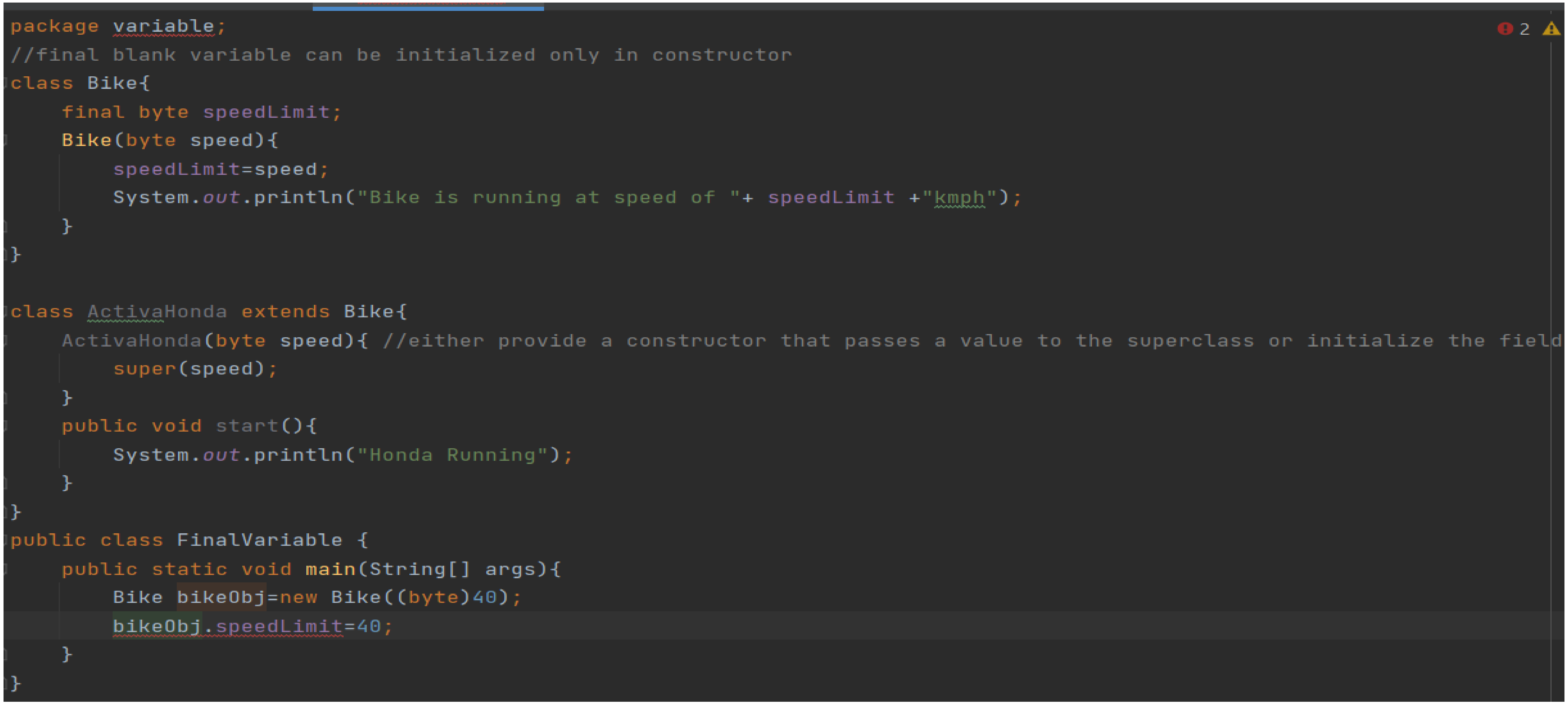


```
package method;
//final keyword in method restricts overriding but can be inherited
 class Bike{
    public final void run(){
        System.out.println("Bike running");
    }
}

class ActivaHonda extends Bike{
    public void start(){
        System.out.println("Honda Running");
    }
}
public class FinalMethod {
    public static void main(String[] args){
        ActivaHonda honda=new ActivaHonda();
        honda.run();
    }
}
```



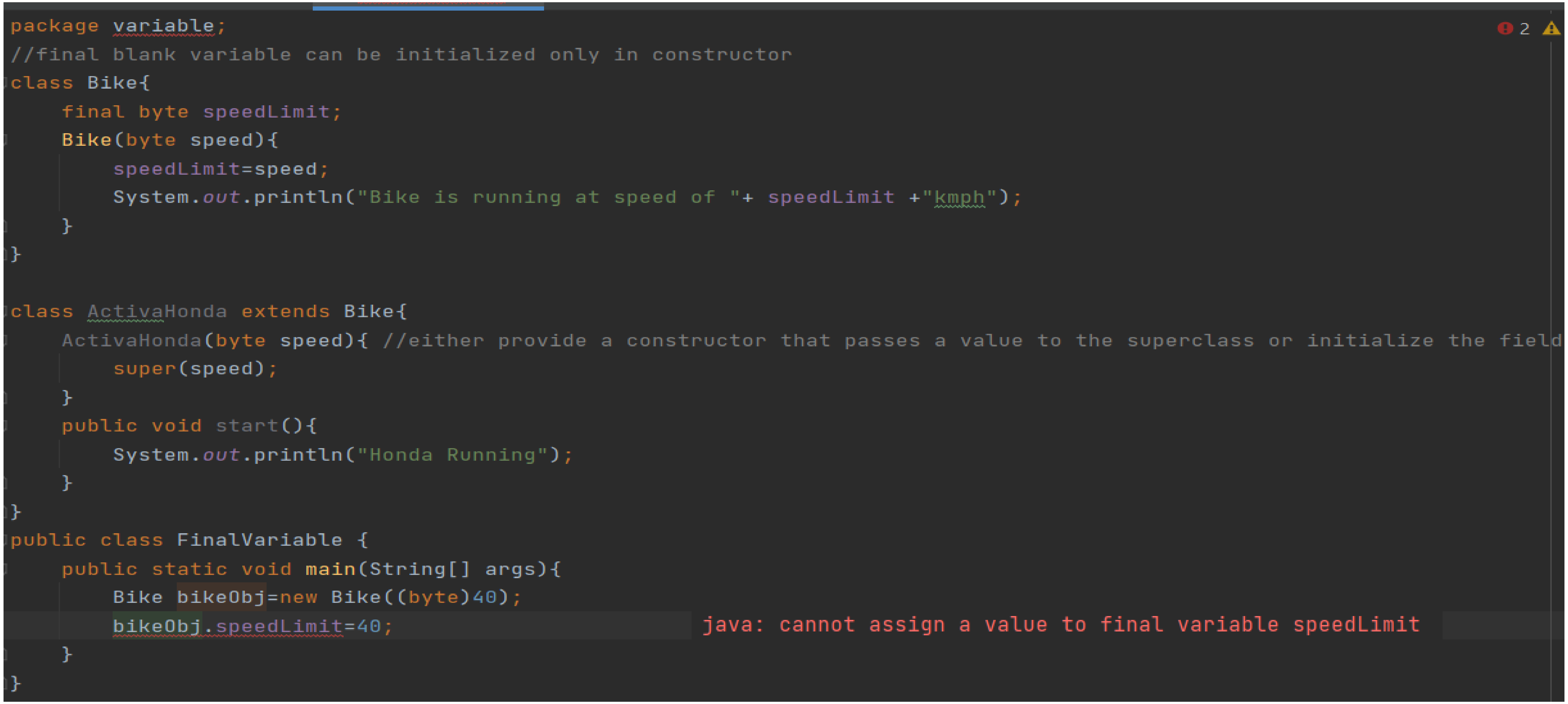
```
Bike running

Process finished with exit code 0
```

Final Variable



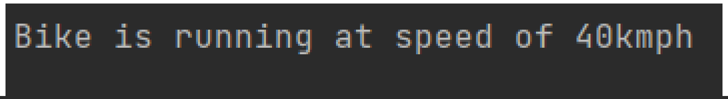
```
package variable;
//final blank variable can be initialized only in constructor
class Bike{
    final byte speedLimit;
    Bike(byte speed){
        speedLimit=speed;
        System.out.println("Bike is running at speed of "+ speedLimit +"kmph");
    }
}

class ActivaHonda extends Bike{
    ActivaHonda(byte speed){ //either provide a constructor that passes a value to the superclass or initialize the field
        super(speed);
    }
    public void start(){
        System.out.println("Honda Running");
    }
}
public class FinalVariable {
    public static void main(String[] args){
        Bike bikeObj=new Bike((byte)40);
        bikeObj.speedLimit=40;
    }
}
```

Final Variable

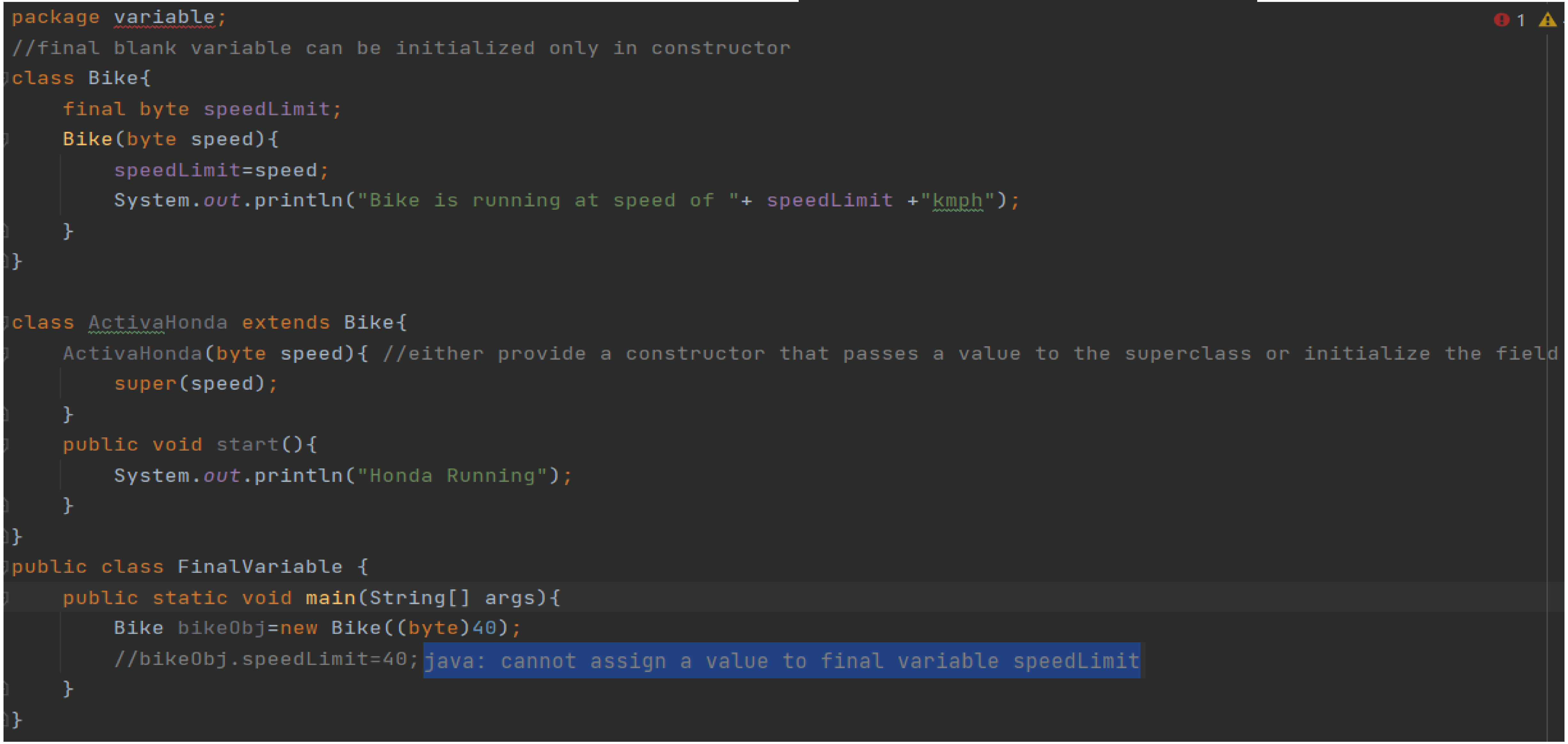


```
package variable;
//final blank variable can be initialized only in constructor
class Bike{
    final byte speedLimit;
    Bike(byte speed){
        speedLimit=speed;
        System.out.println("Bike is running at speed of "+ speedLimit +"kmph");
    }
}

class ActivaHonda extends Bike{
    ActivaHonda(byte speed){ //either provide a constructor that passes a value to the superclass or initialize the field
        super(speed);
    }
    public void start(){
        System.out.println("Honda Running");
    }
}
public class FinalVariable {
    public static void main(String[] args){
        Bike bikeObj=new Bike((byte)40);
        bikeObj.speedLimit=40;              java: cannot assign a value to final variable speedLimit
    }
}
```

Final Variable

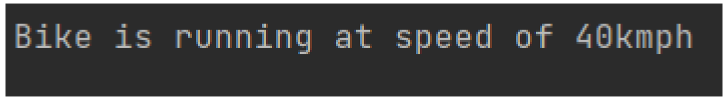


```
Bike is running at speed of 40kmph
```

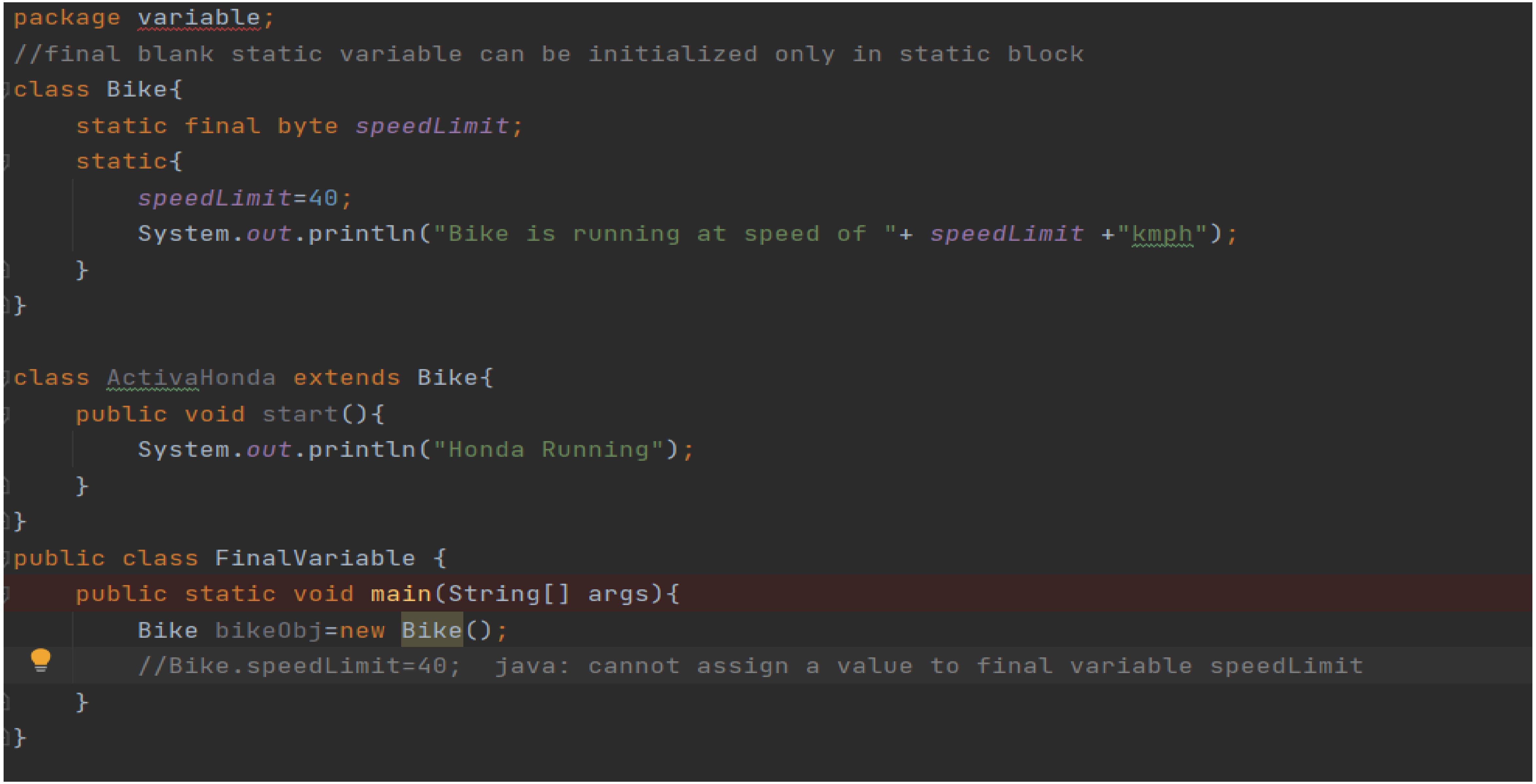


```
package variable;
//final blank variable can be initialized only in constructor
class Bike{
    final byte speedLimit;
    Bike(byte speed){
        speedLimit=speed;
        System.out.println("Bike is running at speed of "+ speedLimit +"kmph");
    }
}

class ActivaHonda extends Bike{
    ActivaHonda(byte speed){ //either provide a constructor that passes a value to the superclass or initialize the field
        super(speed);
    }
    public void start(){
        System.out.println("Honda Running");
    }
}
public class FinalVariable {
    public static void main(String[] args){
        Bike bikeObj=new Bike((byte)40);
        //bikeObj.speedLimit=40;java: cannot assign a value to final variable speedLimit
    }
}
```

Final Variable

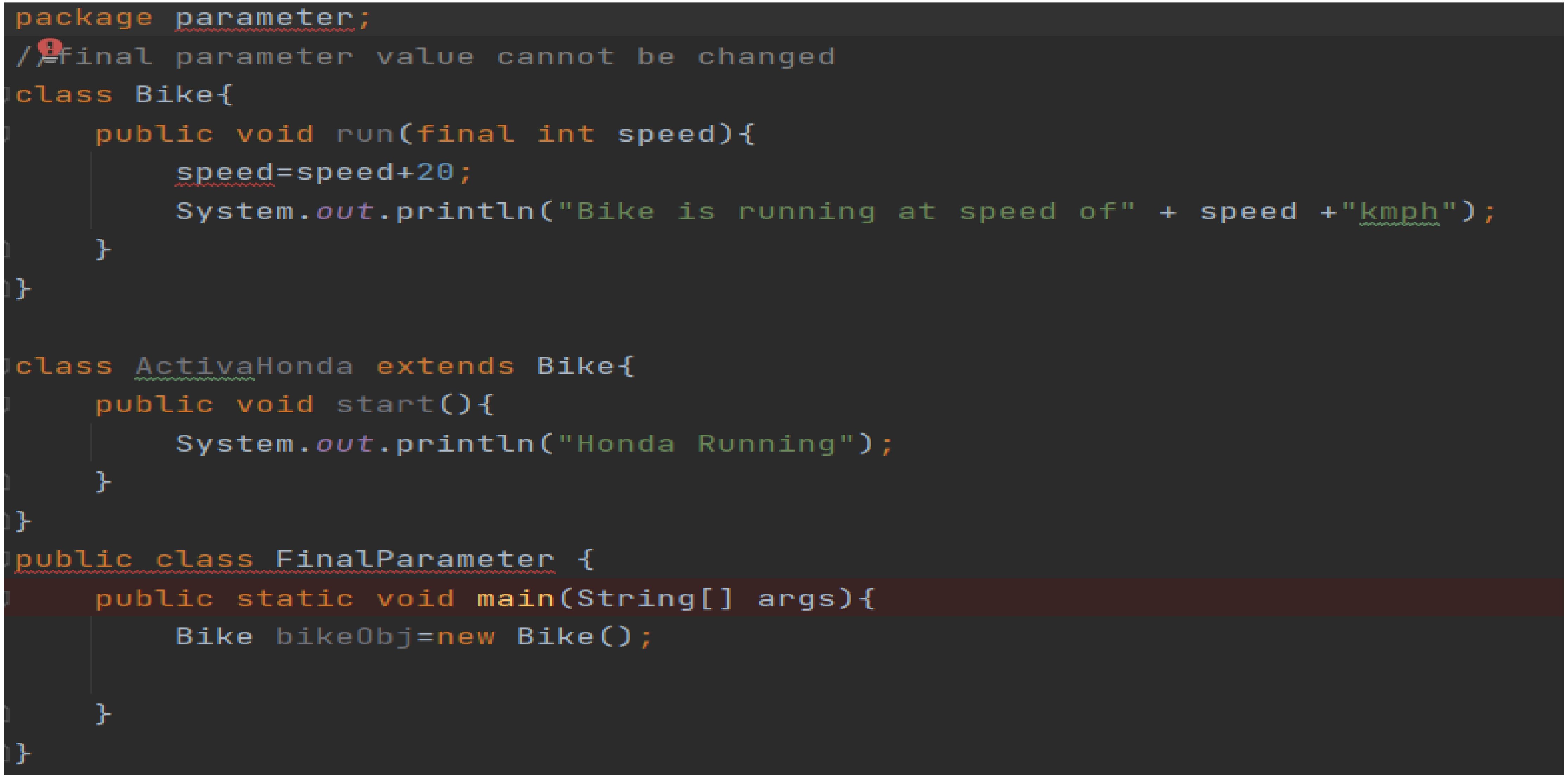


```
Bike is running at speed of 40kmph
```

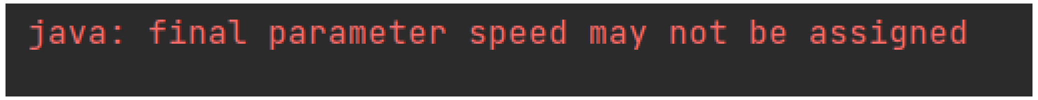


```java
package variable;
//final blank static variable can be initialized only in static block
class Bike{
    static final byte speedLimit;
    static{
        speedLimit=40;
        System.out.println("Bike is running at speed of "+ speedLimit +"kmph");
    }
}

class ActivaHonda extends Bike{
    public void start(){
        System.out.println("Honda Running");
    }
}
public class FinalVariable {
    public static void main(String[] args){
        Bike bikeObj=new Bike();
        //Bike.speedLimit=40;  java: cannot assign a value to final variable speedLimit
    }
}
```

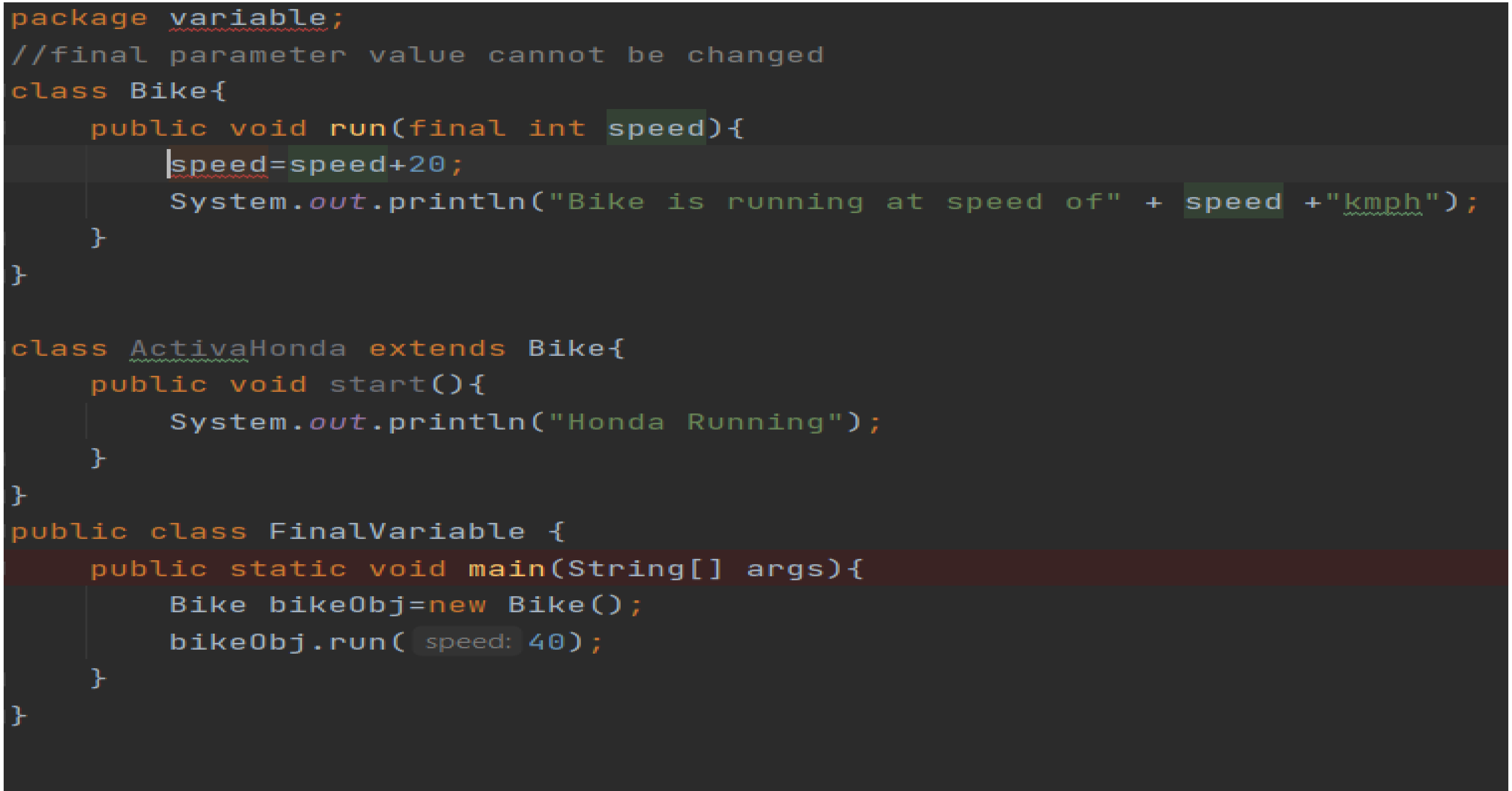
Final parameter


```
package parameter;
//final parameter value cannot be changed
class Bike{
    public void run(final int speed){
        speed=speed+20;
        System.out.println("Bike is running at speed of" + speed +"kmph");
    }
}

class ActivaHonda extends Bike{
    public void start(){
        System.out.println("Honda Running");
    }
}
public class FinalParameter {
    public static void main(String[] args){
        Bike bikeObj=new Bike();

    }
}
```

Final parameter

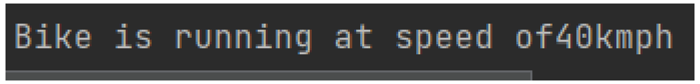


```
java: final parameter speed may not be assigned
```

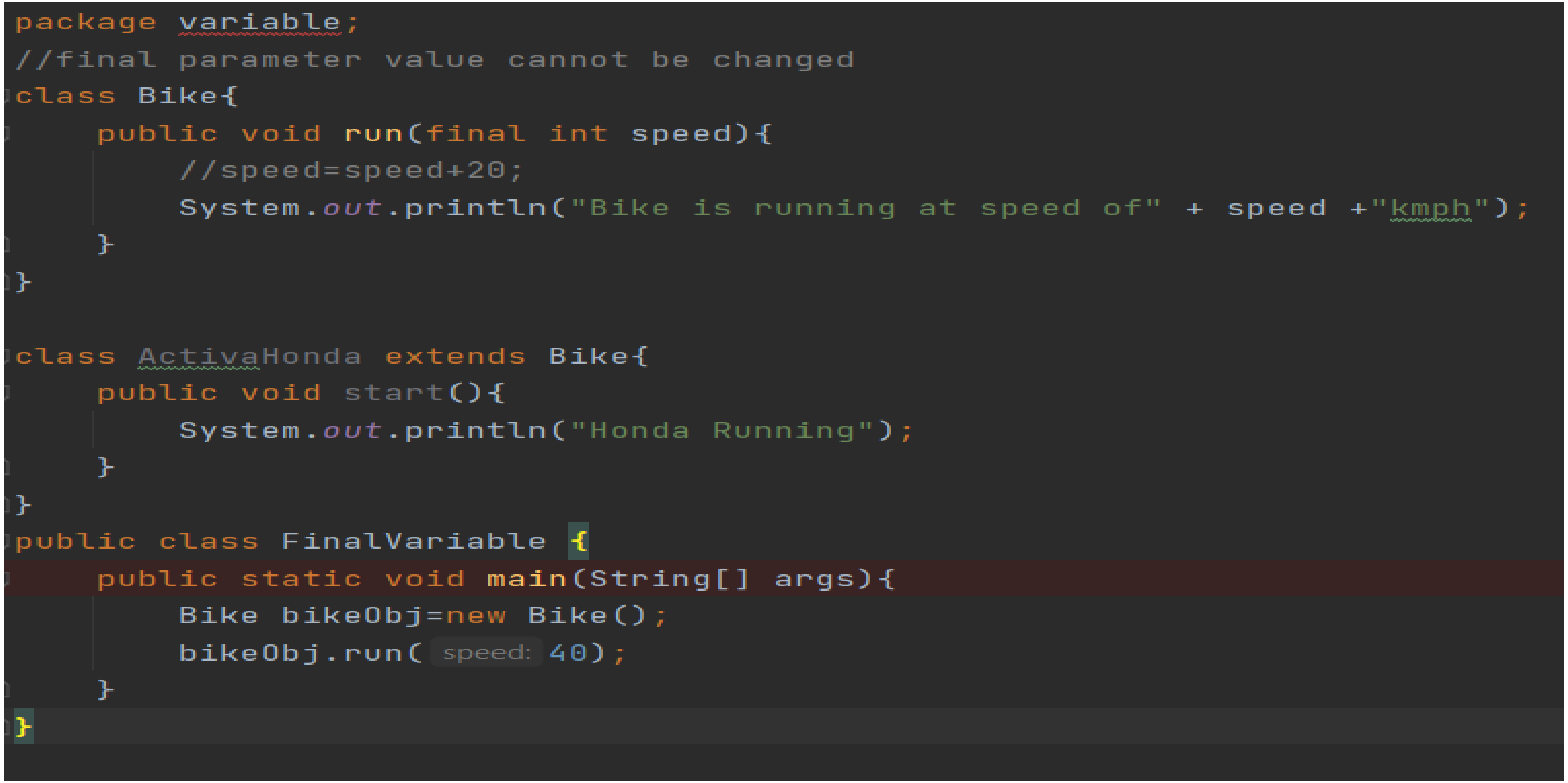


```java
package variable;
//final parameter value cannot be changed
class Bike{
    public void run(final int speed){
        speed=speed+20;
        System.out.println("Bike is running at speed of" + speed +"kmph");
    }
}

class ActivaHonda extends Bike{
    public void start(){
        System.out.println("Honda Running");
    }
}
public class FinalVariable {
    public static void main(String[] args){
        Bike bikeObj=new Bike();
        bikeObj.run( speed: 40);
    }
}
```

Final parameter



```
Bike is running at speed of40kmph
```



```
package variable;
//final parameter value cannot be changed
class Bike{
    public void run(final int speed){
        //speed=speed+20;
        System.out.println("Bike is running at speed of" + speed +"kmph");
    }
}

class ActivaHonda extends Bike{
    public void start(){
        System.out.println("Honda Running");
    }
}
public class FinalVariable {
    public static void main(String[] args){
        Bike bikeObj=new Bike();
        bikeObj.run( speed: 40);
    }
}
```