Unit III

Implementing OOP Concepts (Inheritance)

Tutor: Pema Galey

Learning Outcomes

In this session, you will learn about:

- Basic inheritance
- Relationship of inheritance
- Importance of inheritance
- Types of inheritance
- Method Overriding
- Super Keyword

Introduction to Inheritance

- Inheritance is one of the key features of OOP that allows us to create a new class from an existing class.
- The new class that is created is known as **subclass** (child or derived class) and the existing class from where the child class is derived is known as **superclass** (parent or base class).
- The extends keyword is used to perform inheritance in Java.

Example: Inheritance

```
class Animal {
        // methods and fields
}
// use of extends keyword
// to perform inheritance
class Dog extends Animal {
        // methods and fields of Animal
         // methods and fields of Dog
 }
```

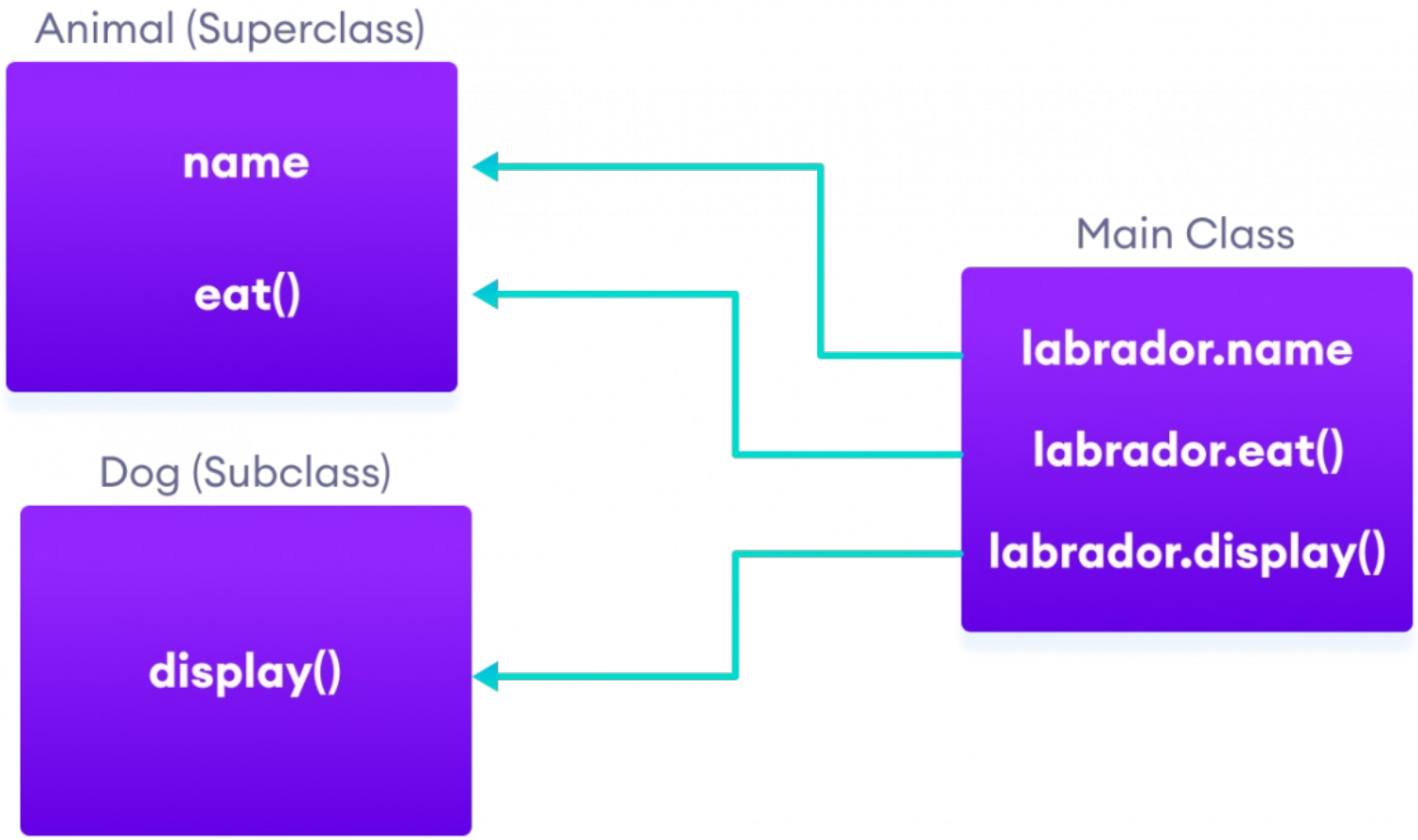
Example: Inheritance

```
class Animal {
        // field and method of the parent class
        String name;
        public void eat() {
         System.out.println("I can eat");
} }
// inherit from Animal
class Dog extends Animal {
         // new method in subclass
        public void display() {
                System.out.println("My name
                        is " + name);
} }
```

```
class Main {
        public static void main(String[] args)
{
                // create an object of the
                //subclass
                 Dog ob = new Dog();
                // access field of superclass
                ob.name = ”Sintu";
                ob.display();
                // call method of superclass
                //using object of subclass
                ob.eat();
} }
```

Output:
My name is Sintu
I can eat

Example: Inheritance



is-a relationship

- Inheritance is an **is-a** relationship. That is, we use inheritance only if there exists an is-a relationship between two classes.
- For example,
  - **Car** is a **Vehicle**
  - **Orange** is a **Fruit**
  - **Surgeon** is a **Doctor**
  - **Dog** is an **Animal**
- Here, **Car** can inherit from **Vehicle**, **Orange** can inherit from **Fruit**, and so on.

protected Members in Inheritance

- If a class includes protected fields and methods, then these fields and methods are accessible from the subclass of the class.

Example: protected Members

```
class Animal {
        protected String name;
        protected void display() {
                System.out.println("I am an animal.");
} }
class Dog extends Animal {
        public void getInfo() {
                System.out.println("My name is " + name);
} }
class Main {
        public static void main(String[] args) {
                Dog labrador = new Dog();
                labrador.name = ”Sintu";
                labrador.display();
                labrador.getInfo();
} }
```

Output:
I am an animal.
My name is Sintu

Why use inheritance?

- The most important use of inheritance in Java is ***code reusability***. The code that is present in the parent class can be directly used by the child class.

- Method overriding is also known as ***runtime polymorphism***. Hence, we can achieve Polymorphism in Java with the help of inheritance.

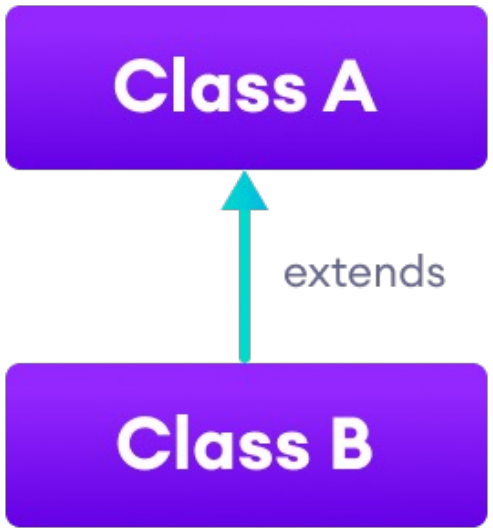
- In general, there are two types of inheritance:
  1. **Single Inheritance**
  2. **Multilevel Inheritance**

Types of Inheritance

- There are five types of inheritance.
  1. Single Inheritance
  2. Multilevel Inheritance
  3. Hierarchical Inheritance
  4. Multiple Inheritance
  5. Hybrid Inheritance

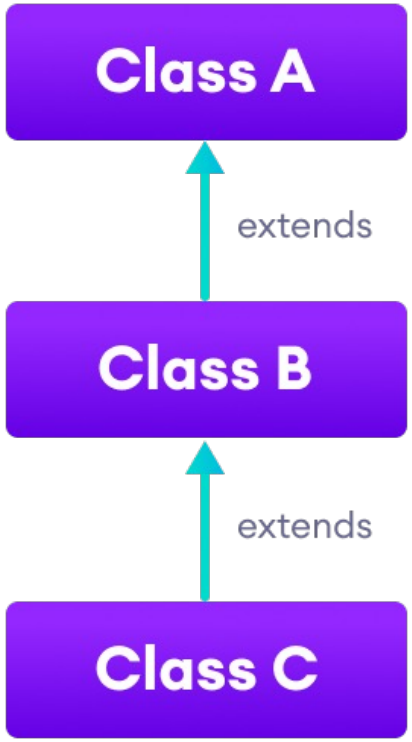
1. Single Inheritance

- In single inheritance, a single subclass extends from a single superclass.



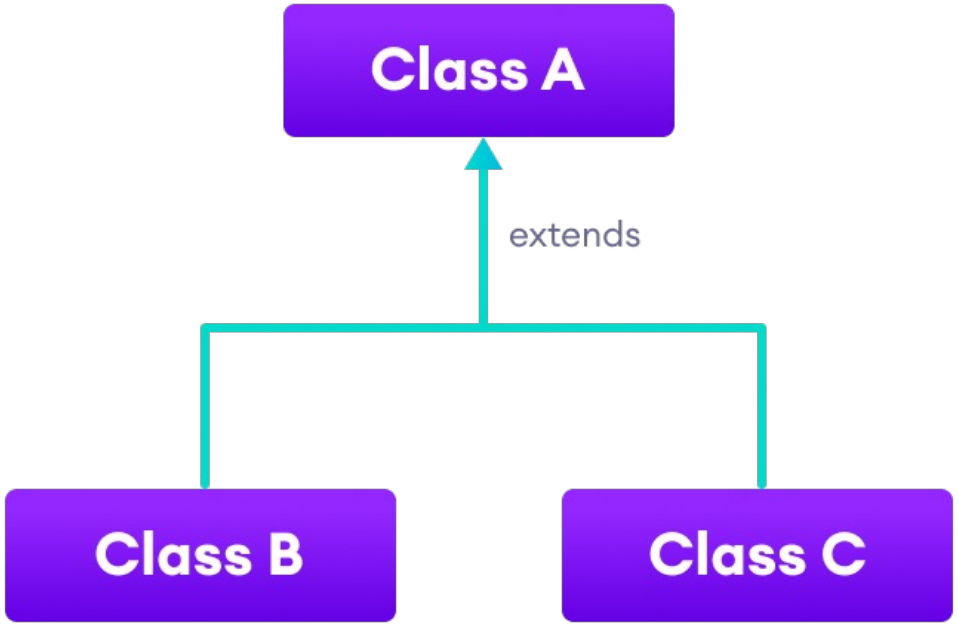
2. Multilevel Inheritance

- In multilevel inheritance, a subclass extends from a superclass and then the same subclass acts as a superclass for another class.



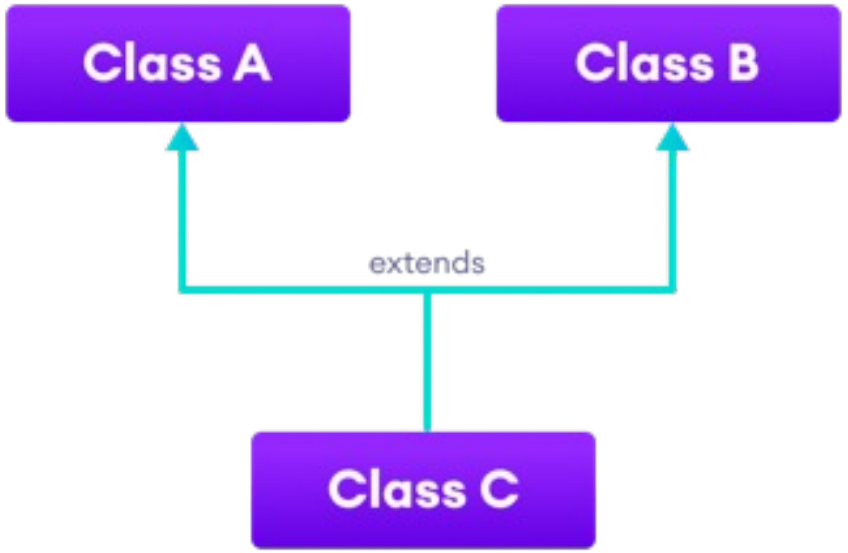
3. Hierarchical Inheritance

- In hierarchical inheritance, multiple subclasses extend from a single superclass.



4. Multiple Inheritance

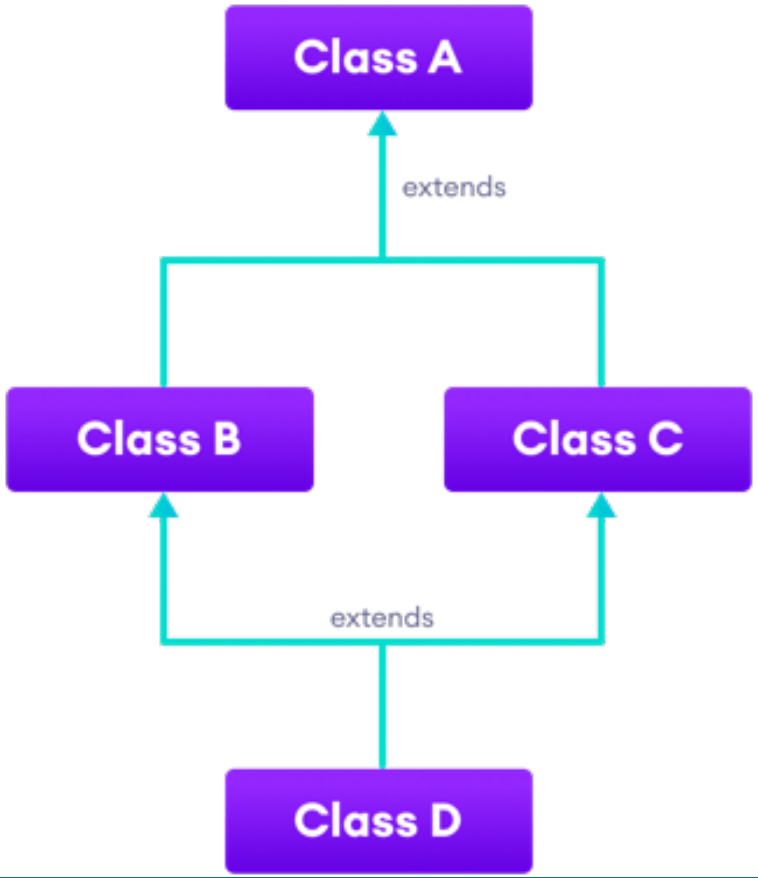
- In multiple inheritance, a single subclass extends from multiple superclasses.



- **Note**: Java doesn't support multiple inheritance. However, we can achieve multiple inheritance using interfaces.

5. Hybrid Inheritance

- Hybrid inheritance is a combination of two or more types of inheritance.



Method Overriding

- Inheritance is an OOP property that allows us to derive a new class (subclass) from an existing class (superclass). The subclass inherits the attributes and methods of the superclass.

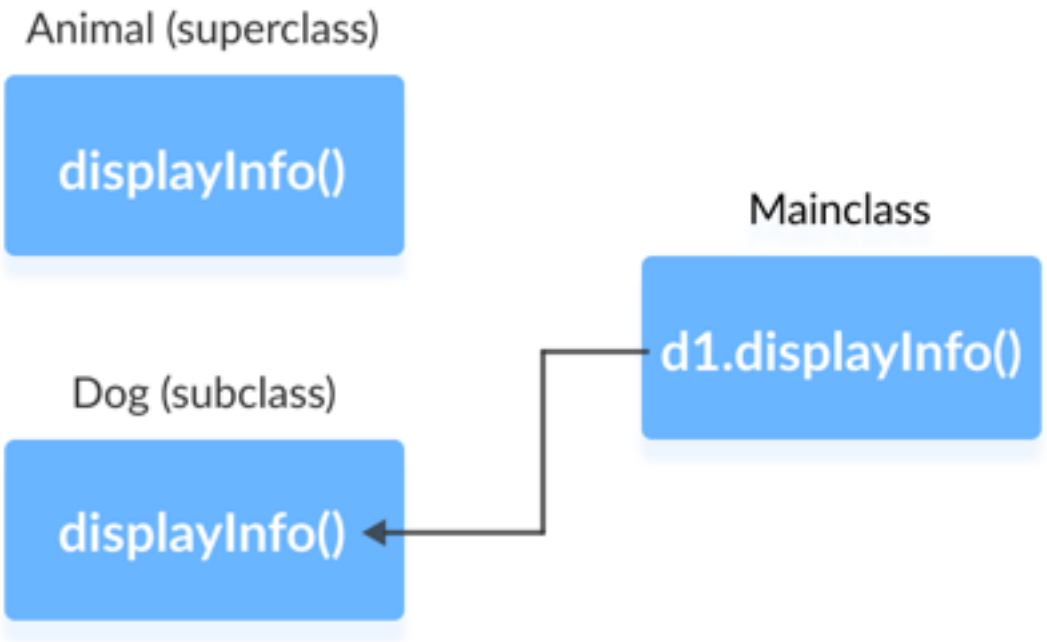
- Now, if the same method is defined in both the superclass and the subclass, then the method of the subclass class overrides the method of the superclass. This is known as method overriding.

Example: Method Overriding

```
class Animal {
        public void displayInfo() {
                System.out.println("I am an animal.");
} }
class Dog extends Animal {
        @Override
        public void displayInfo() {
                System.out.println("I am a dog.");
} }
class Main {
        public static void main(String[] args) {
                Dog d1 = new Dog();
                d1.displayInfo();
        }
 }
```

Output:
I am a dog.

Example: Method Overriding



Note: Notice the use of the @Override annotation. Here, the @Override annotation specifies the compiler that the method after this annotation overrides the method of the superclass. It is not mandatory to use @Override.

Java Overriding Rules

- Both the superclass and the subclass must have the same method name, the same return type and the same parameter list.
- We cannot override the method declared as final and static.
- We should always override abstract methods of the superclass (will be discussed in later classes).

Overriding Abstract Method

- In Java, abstract classes are created to be the superclass of other classes. And, if a class contains an abstract method, it is mandatory to override it.

super Keyword

- **Can we access the method of the superclass after overriding?**
    - Well, the answer is **Yes**. To access the method of the superclass from the subclass, we use the ***super*** keyword.
- It is important to note that constructors in Java are not inherited. Hence, there is no such thing as constructor overriding in Java.
- However, we can call the constructor of the superclass from its subclasses. For that, we use super().

Uses of super keyword

1. To call methods of the superclass that is overridden in the subclass.
2. To access attributes (fields) of the superclass if both superclass and subclass have attributes with the same name.
3. To explicitly call superclass no-argument (default) or parameterized constructor from the subclass constructor.

Example 1: super Keyword

```
class Animal {
        public void displayInfo() {
                System.out.println("I am an animal.");
} }
class Dog extends Animal {
        public void displayInfo() {
                super.displayInfo();
                System.out.println("I am a dog.");
} }
class Main {
        public static void main(String[] args) {
                Dog d1 = new Dog();
                d1.displayInfo();
        }
}
```

Output:
I am an animal.
I am a dog.

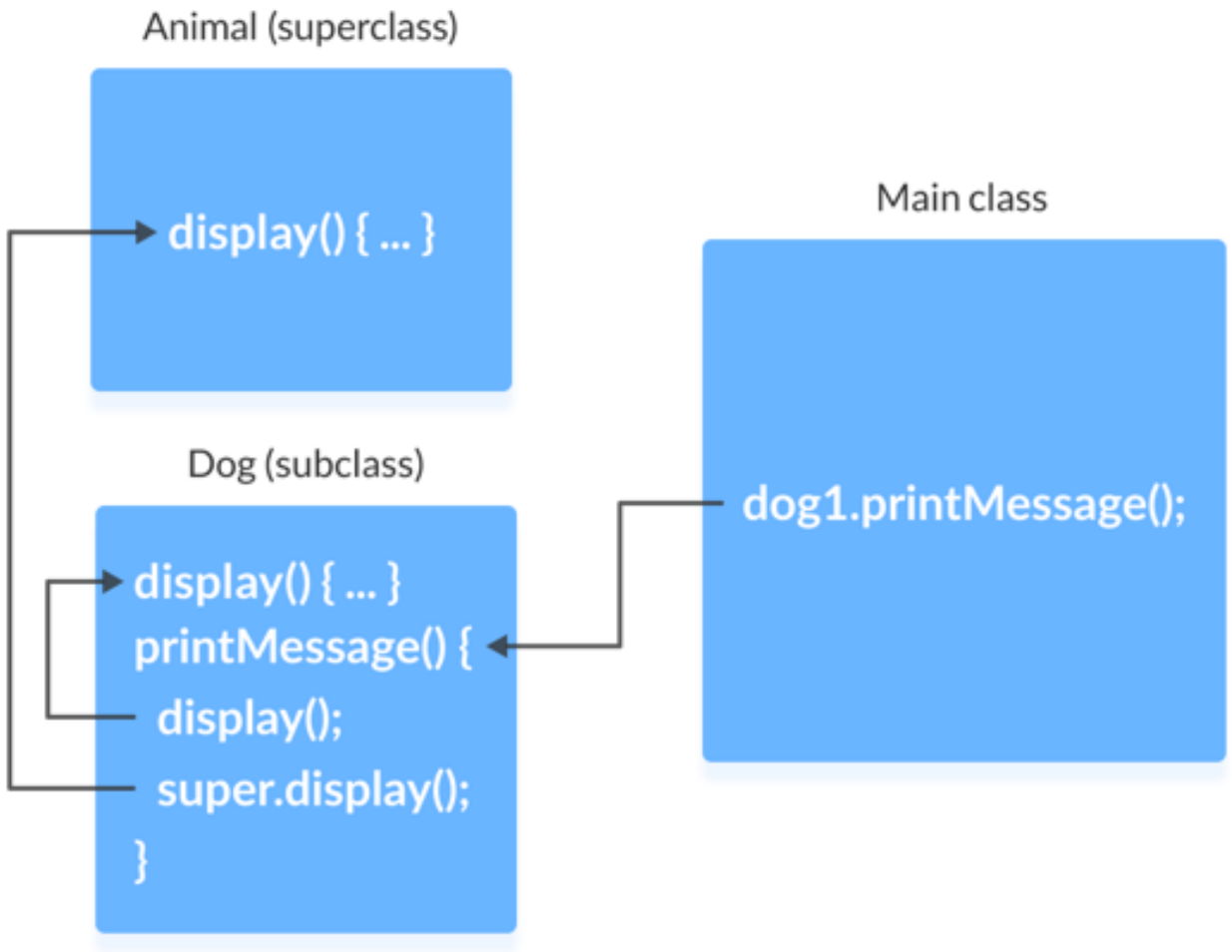
Example 2: super to Call Superclass Method

```
class Animal {
        // overridden method
        public void display(){
                System.out.println("I am an animal");
} }
class Dog extends Animal {
        @Override public void display(){
                System.out.println("I am a dog");
        }
        public void printMessage(){
                // this calls overriding method
                display();
                // this calls overridden method
                super.display();
} }
```

```
class Main {
        public static void main(String[] args) {
                Dog dog1 = new Dog();
                dog1.printMessage();
} }
```

Output:
I am a dog
I am an animal

Example 2: super to Call Superclass Method



Example 3: Access superclass attribute

```
class Animal {
        protected String type="animal";
}
class Dog extends Animal {
        public String type=”dog";
        public void printType() {
                System.out.println("I am a " + type);
                System.out.println("I am an " + super.type);
} }
class Main {
        public static void main(String[] args) {
                Dog dog1 = new Dog();
                dog1.printType();
} }
```

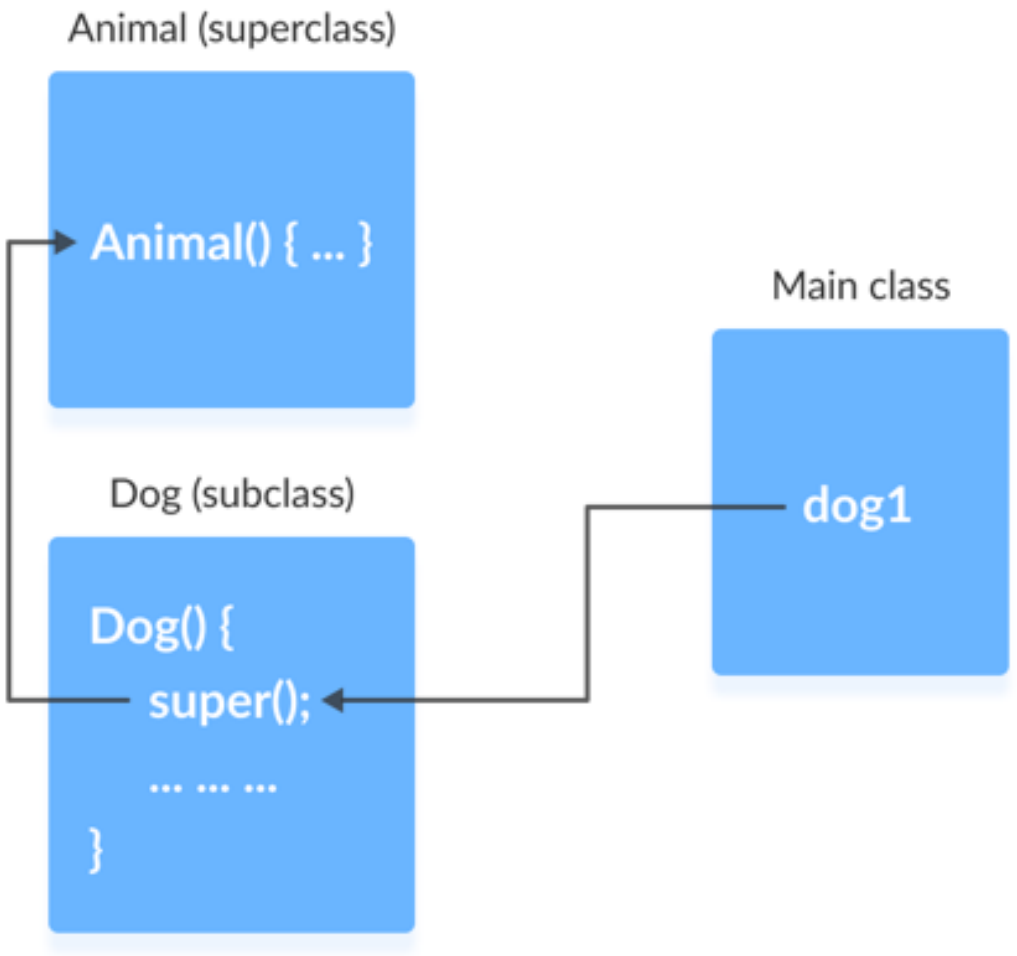
Output:
I am a dog
I am an animal

Example 4: Use of super()

```
class Animal {
        Animal() {
                System.out.println("I am an animal");
} }
class Dog extends Animal {
        Dog() {
                // calling default constructor of the superclass
                super();
                System.out.println("I am a dog");
} }
class Main {
        public static void main(String[] args) {
                Dog dog1 = new Dog();
        }
}
```

Output
I am an animal
I am a dog

Example 4: Use of super()



Thank you!