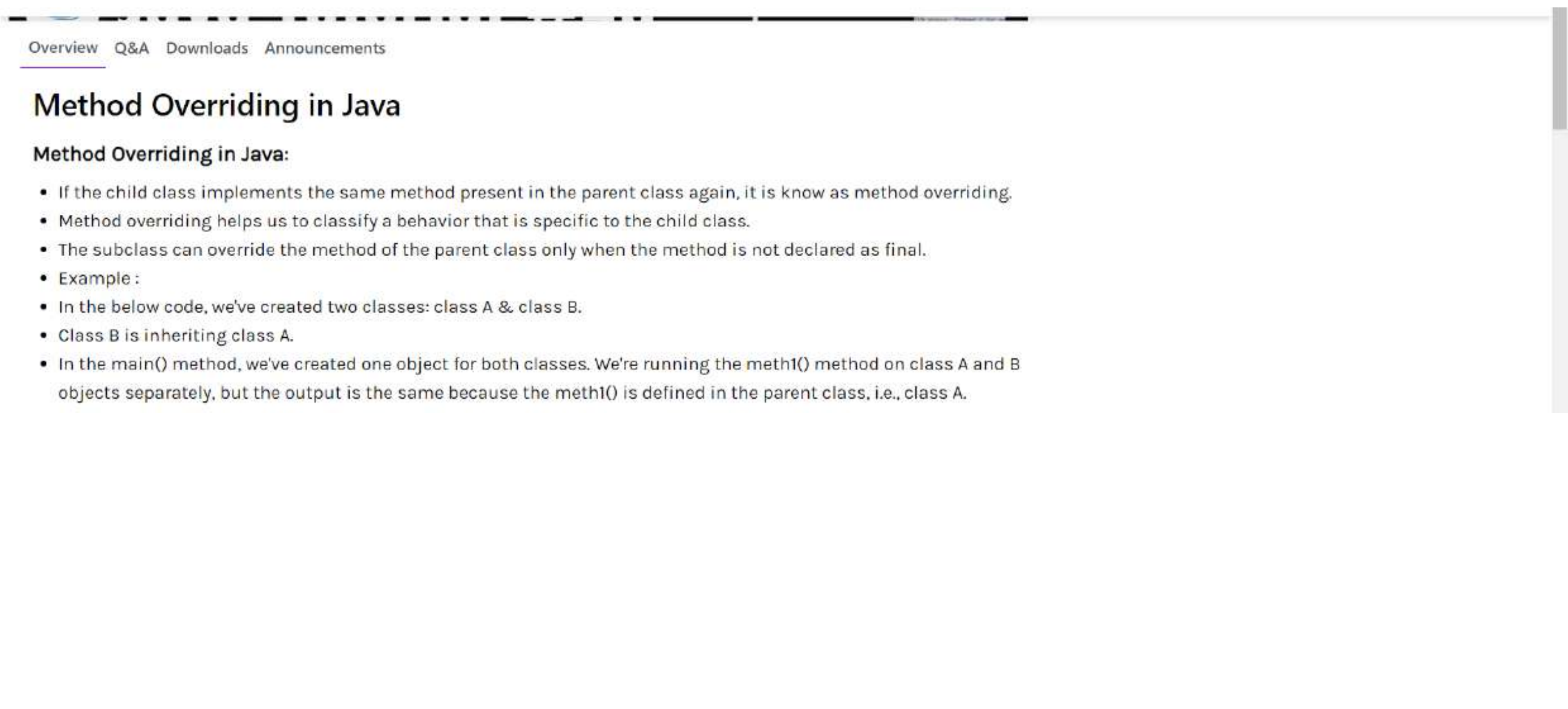
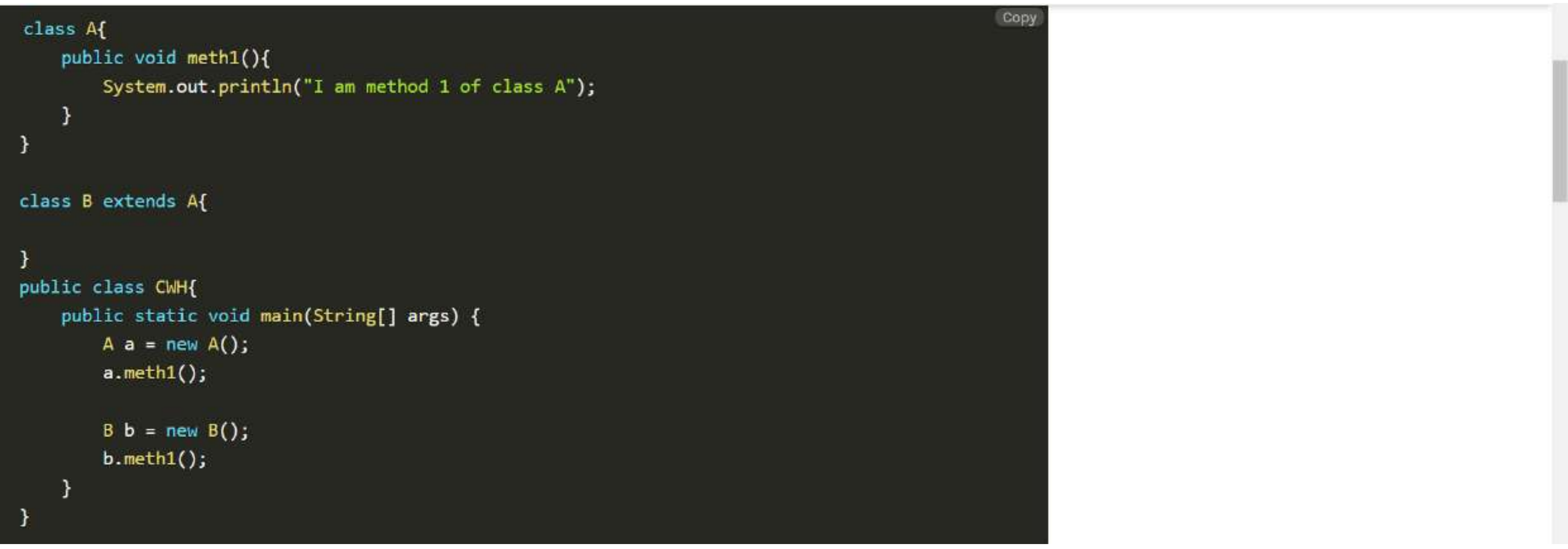


Method Overriding in Java

Method Overriding in Java:

- If the child class implements the same method present in the parent class again, it is know as method overriding.
- Method overriding helps us to classify a behavior that is specific to the child class.
- The subclass can override the method of the parent class only when the method is not declared as final.
- Example :
- In the below code, we've created two classes: class A & class B.
- Class B is inheriting class A.
- In the main() method, we've created one object for both classes. We're running the meth1() method on class A and B objects separately, but the output is the same because the meth1() is defined in the parent class, i.e., class A.



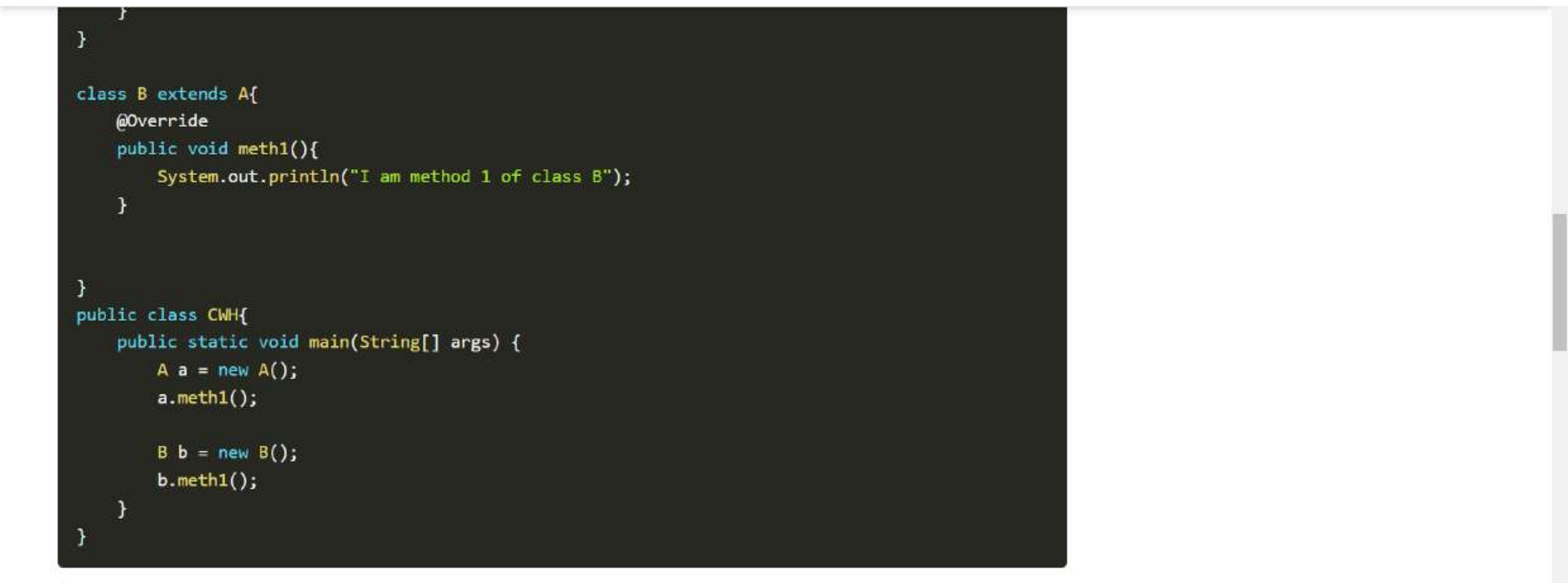
```java
class A{
    public void meth1(){
        System.out.println("I am method 1 of class A");
    }
}

class B extends A{

}
public class CWH{
    public static void main(String[] args) {
        A a = new A();
        a.meth1();

        B b = new B();
        b.meth1();
    }
}
```

Output :

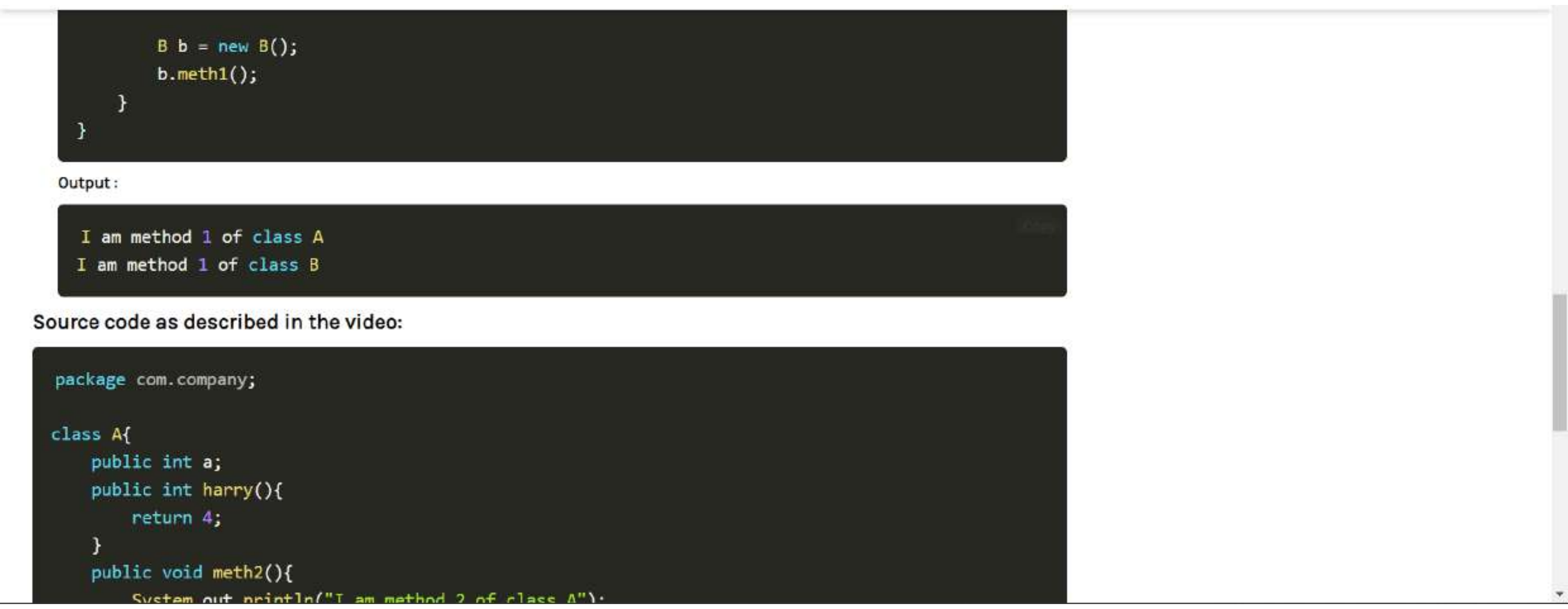
```
I am method 1 of class A
I am method 1 of class A
```

- Now, let's see how we can override the meth1() for class B :

```
class A{
    public void meth1(){
        System.out.println("I am method 1 of class A");
    }
}

class B extends A{
    @Override
    public void meth1(){
        System.out.println("I am method 1 of class B");
    }


}
```



```java
    }
}

class B extends A{
    @Override
    public void meth1(){
        System.out.println("I am method 1 of class B");
    }


}
public class CWH{
    public static void main(String[] args) {
        A a = new A();
        a.meth1();

        B b = new B();
        b.meth1();
    }
}
```



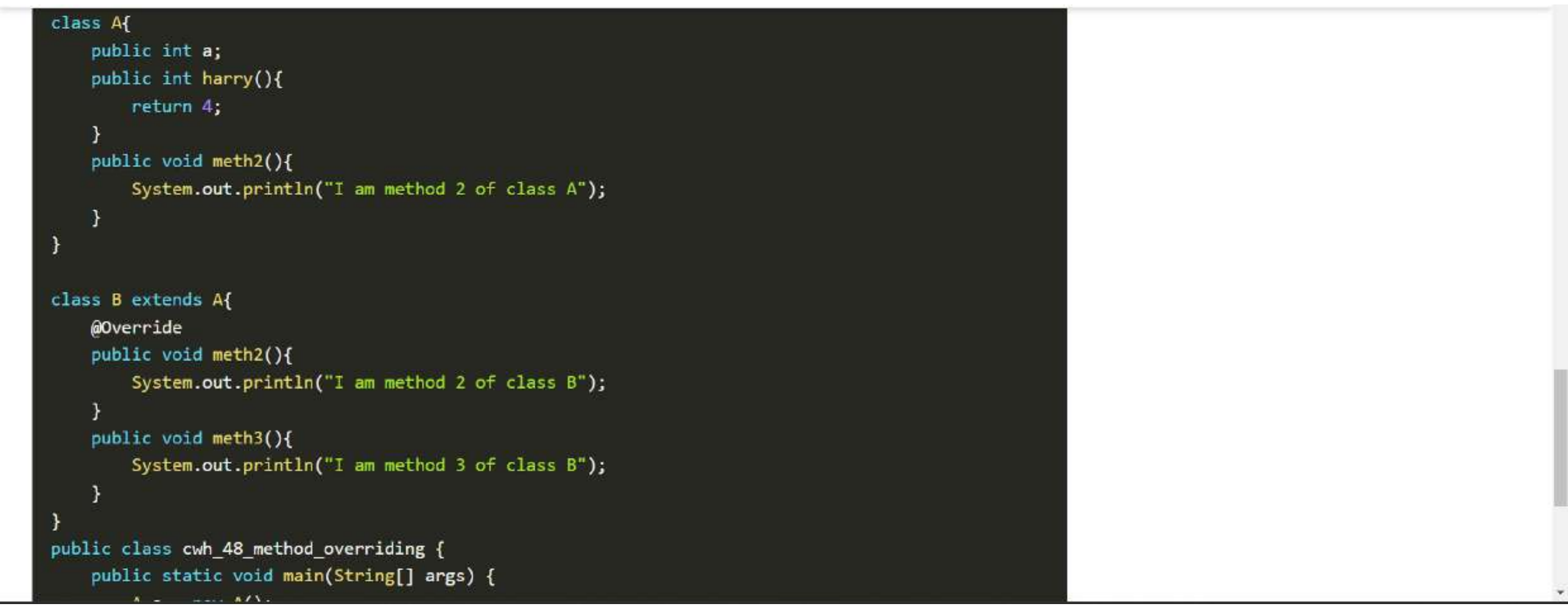
```
        B b = new B();
        b.meth1();
    }
}
```

Output :

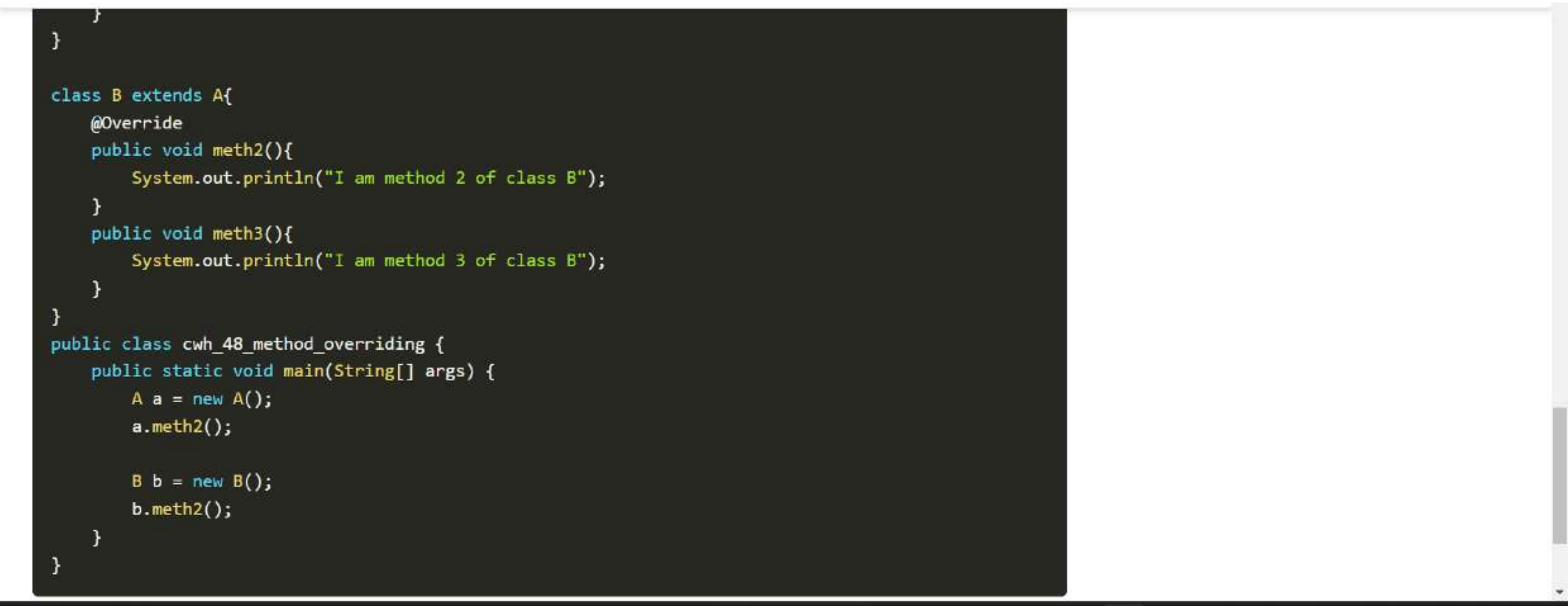
```
 I am method 1 of class A
I am method 1 of class B
```

Source code as described in the video:

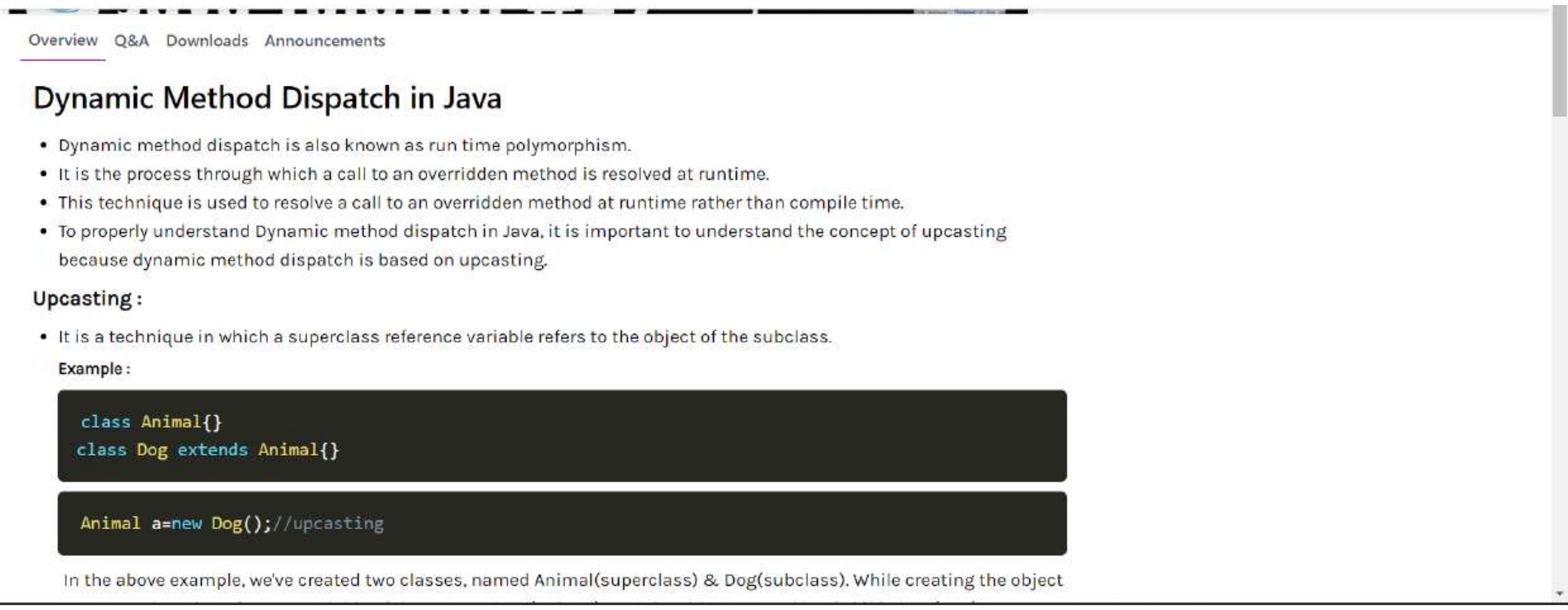
```
package com.company;

class A{
    public int a;
    public int harry(){
        return 4;
    }
    public void meth2(){
        System.out.println("I am method 2 of class A");
```



```java
class A{
    public int a;
    public int harry(){
        return 4;
    }
    public void meth2(){
        System.out.println("I am method 2 of class A");
    }
}

class B extends A{
    @Override
    public void meth2(){
        System.out.println("I am method 2 of class B");
    }
    public void meth3(){
        System.out.println("I am method 3 of class B");
    }
}
public class cwh_48_method_overriding {
    public static void main(String[] args) {
```



```java
    }
}

class B extends A{
    @Override
    public void meth2(){
        System.out.println("I am method 2 of class B");
    }
    public void meth3(){
        System.out.println("I am method 3 of class B");
    }
}
public class cwh_48_method_overriding {
    public static void main(String[] args) {
        A a = new A();
        a.meth2();

        B b = new B();
        b.meth2();
    }
}
```



Dynamic Method Dispatch in Java

- Dynamic method dispatch is also known as run time polymorphism.
- It is the process through which a call to an overridden method is resolved at runtime.
- This technique is used to resolve a call to an overridden method at runtime rather than compile time.
- To properly understand Dynamic method dispatch in Java, it is important to understand the concept of upcasting because dynamic method dispatch is based on upcasting.

Upcasting :

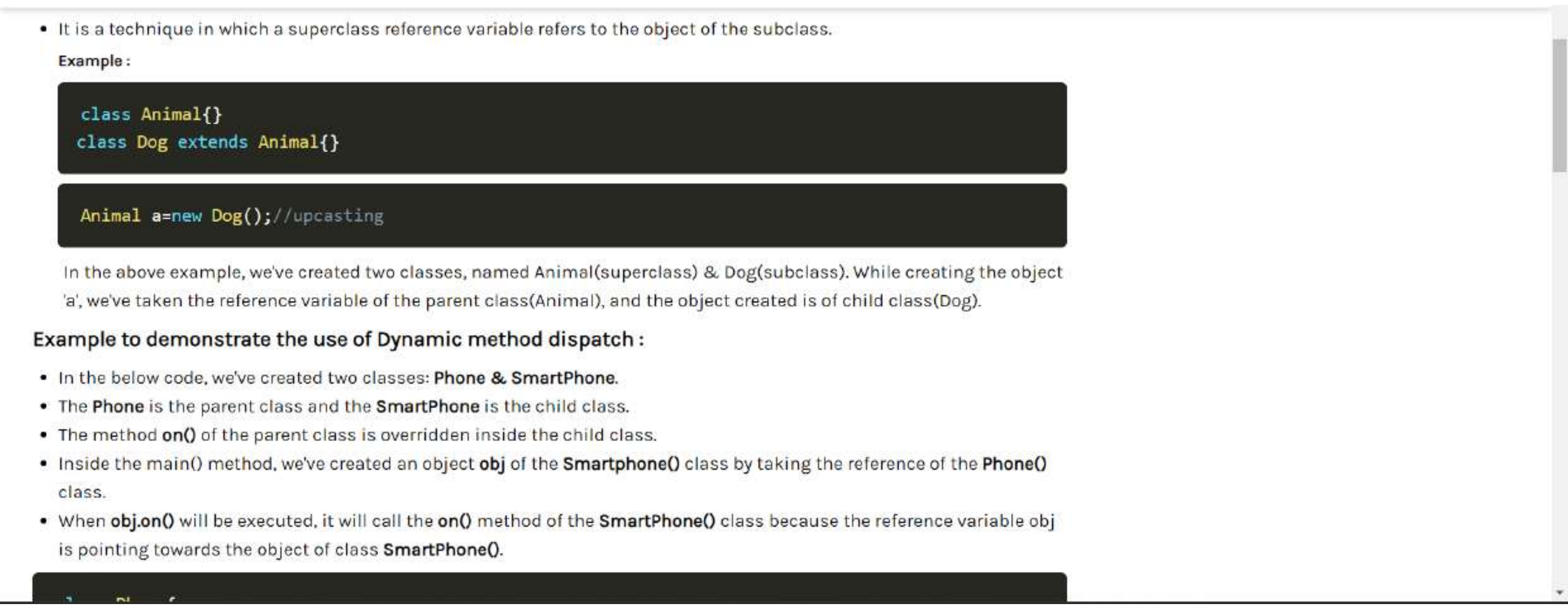
- It is a technique in which a superclass reference variable refers to the object of the subclass.

Example :

```
 class Animal{}
class Dog extends Animal{}
```

```
Animal a=new Dog();//upcasting
```

In the above example, we've created two classes, named Animal(superclass) & Dog(subclass). While creating the object



- It is a technique in which a superclass reference variable refers to the object of the subclass.

  Example :

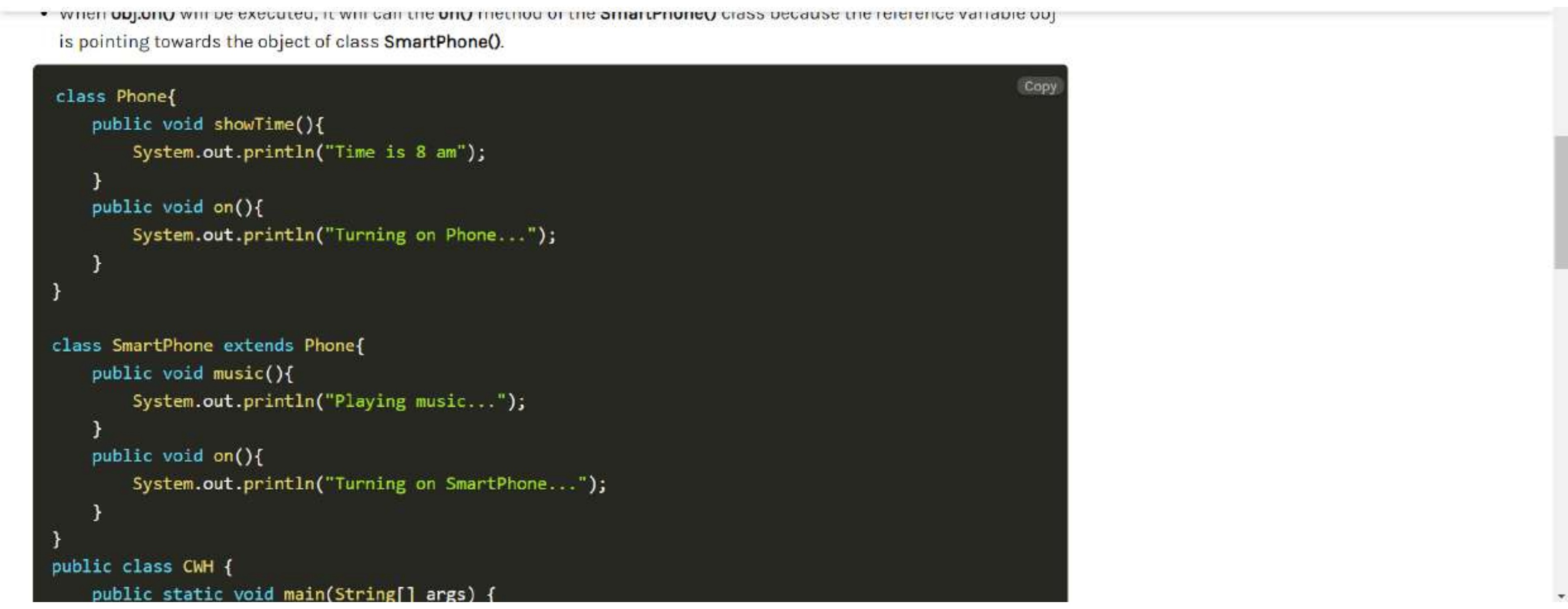
```java
 class Animal{}
class Dog extends Animal{}
```

```java
Animal a=new Dog();//upcasting
```

In the above example, we've created two classes, named Animal(superclass) & Dog(subclass). While creating the object 'a', we've taken the reference variable of the parent class(Animal), and the object created is of child class(Dog).

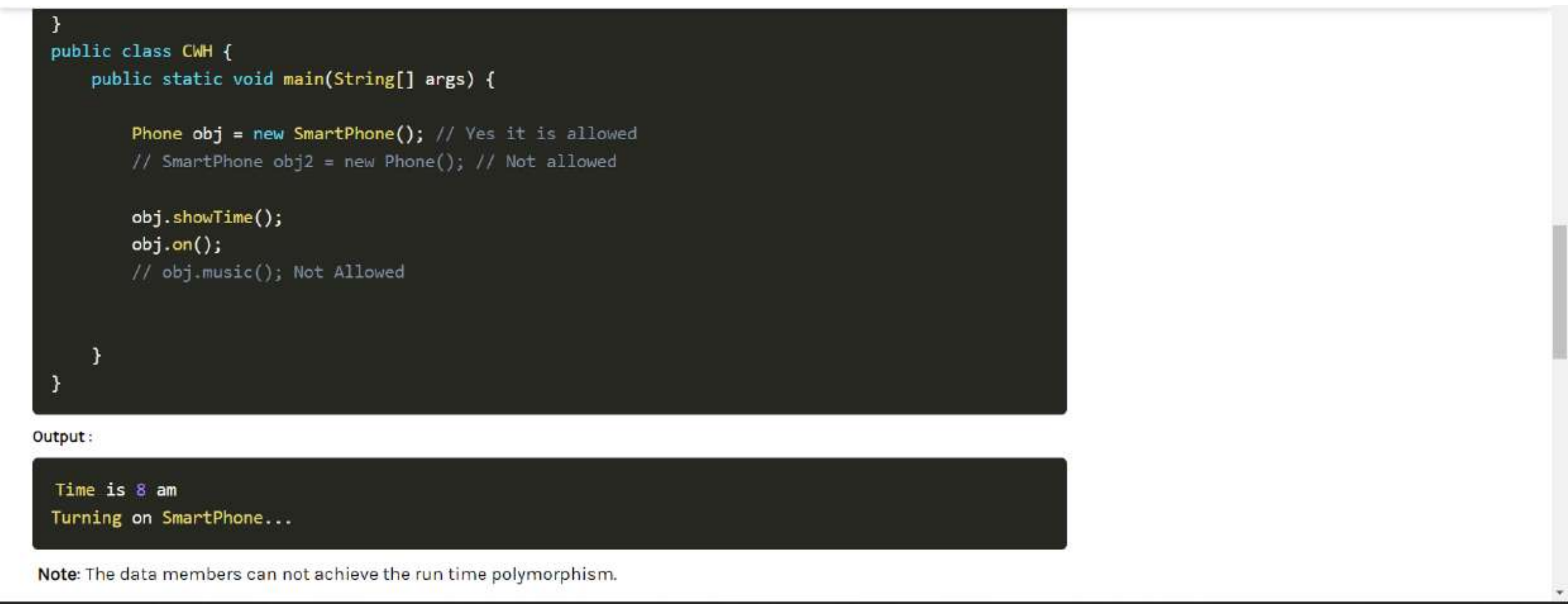
Example to demonstrate the use of Dynamic method dispatch :

- In the below code, we've created two classes: **Phone & SmartPhone.**
- The **Phone** is the parent class and the **SmartPhone** is the child class.
- The method **on()** of the parent class is overridden inside the child class.
- Inside the main() method, we've created an object **obj** of the **Smartphone()** class by taking the reference of the **Phone()** class.
- When **obj.on()** will be executed, it will call the **on()** method of the **SmartPhone()** class because the reference variable obj is pointing towards the object of class **SmartPhone()**.



- When **obj.on()** will be executed, it will call the **on()** method of the **SmartPhone()** class because the reference variable obj is pointing towards the object of class **SmartPhone()**.

```java
class Phone{
    public void showTime(){
        System.out.println("Time is 8 am");
    }
    public void on(){
        System.out.println("Turning on Phone...");
    }
}

class SmartPhone extends Phone{
    public void music(){
        System.out.println("Playing music...");
    }
    public void on(){
        System.out.println("Turning on SmartPhone...");
    }
}
public class CWH {
    public static void main(String[] args) {
```

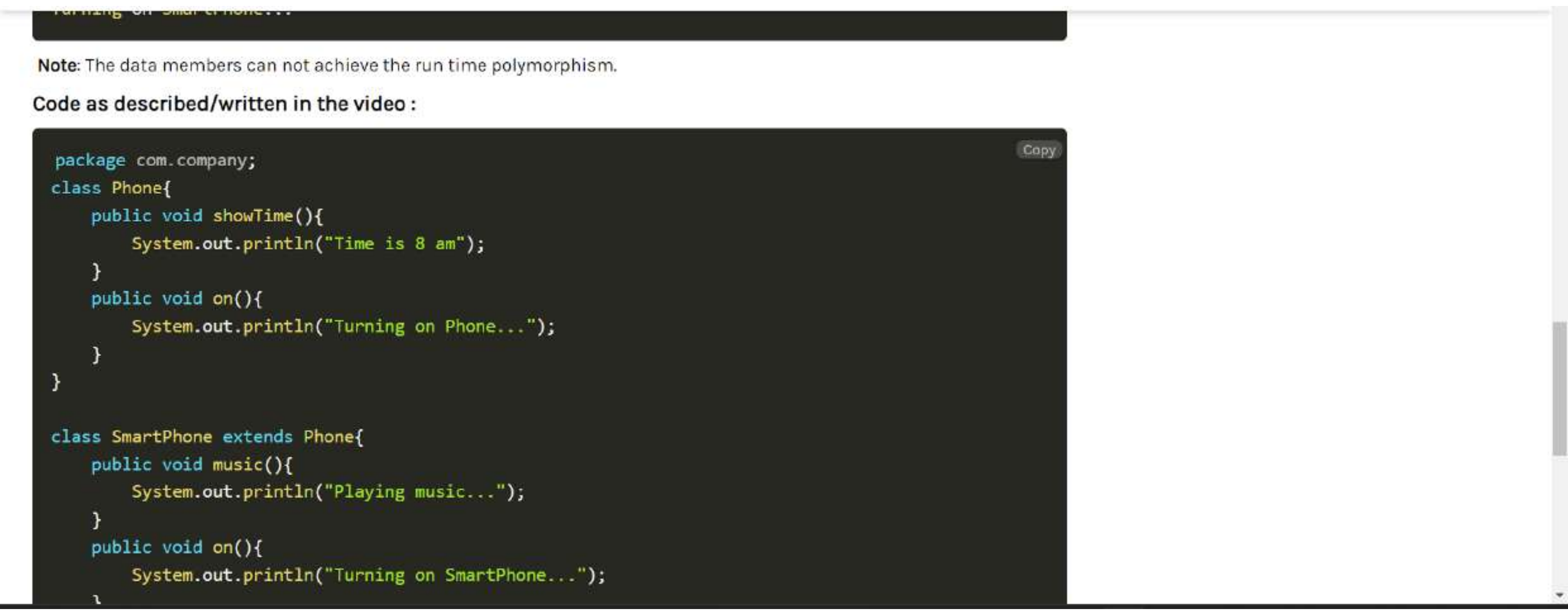


```
}
public class CWH {
    public static void main(String[] args) {

        Phone obj = new SmartPhone(); // Yes it is allowed
        // SmartPhone obj2 = new Phone(); // Not allowed

        obj.showTime();
        obj.on();
        // obj.music(); Not Allowed


    }
}
```

Output :

```
Time is 8 am
Turning on SmartPhone...
```

Note: The data members can not achieve the run time polymorphism.

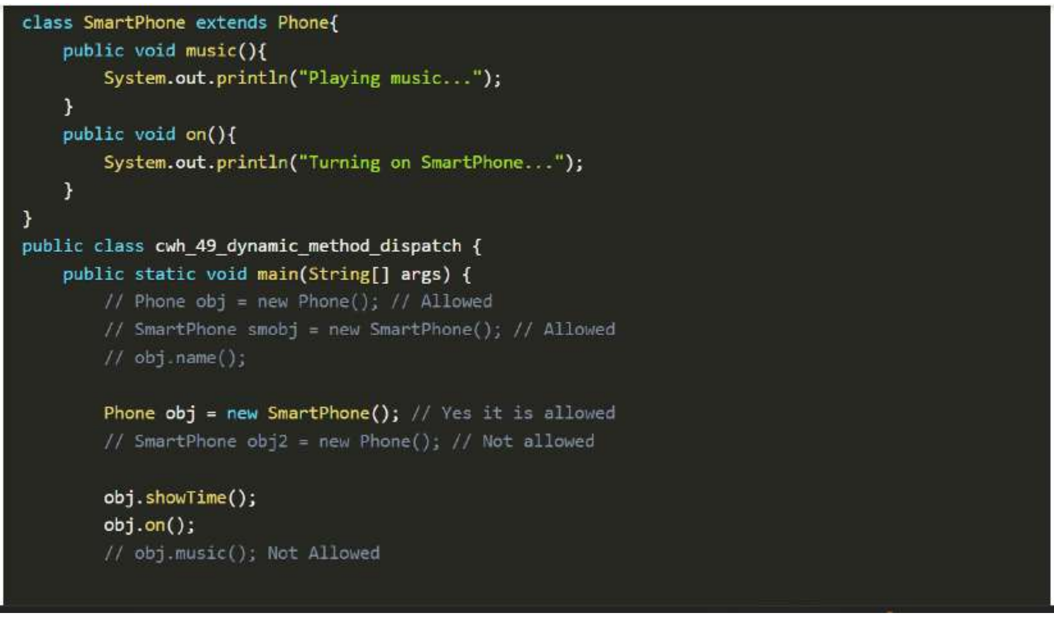


```
Turning on SmartPhone...
```

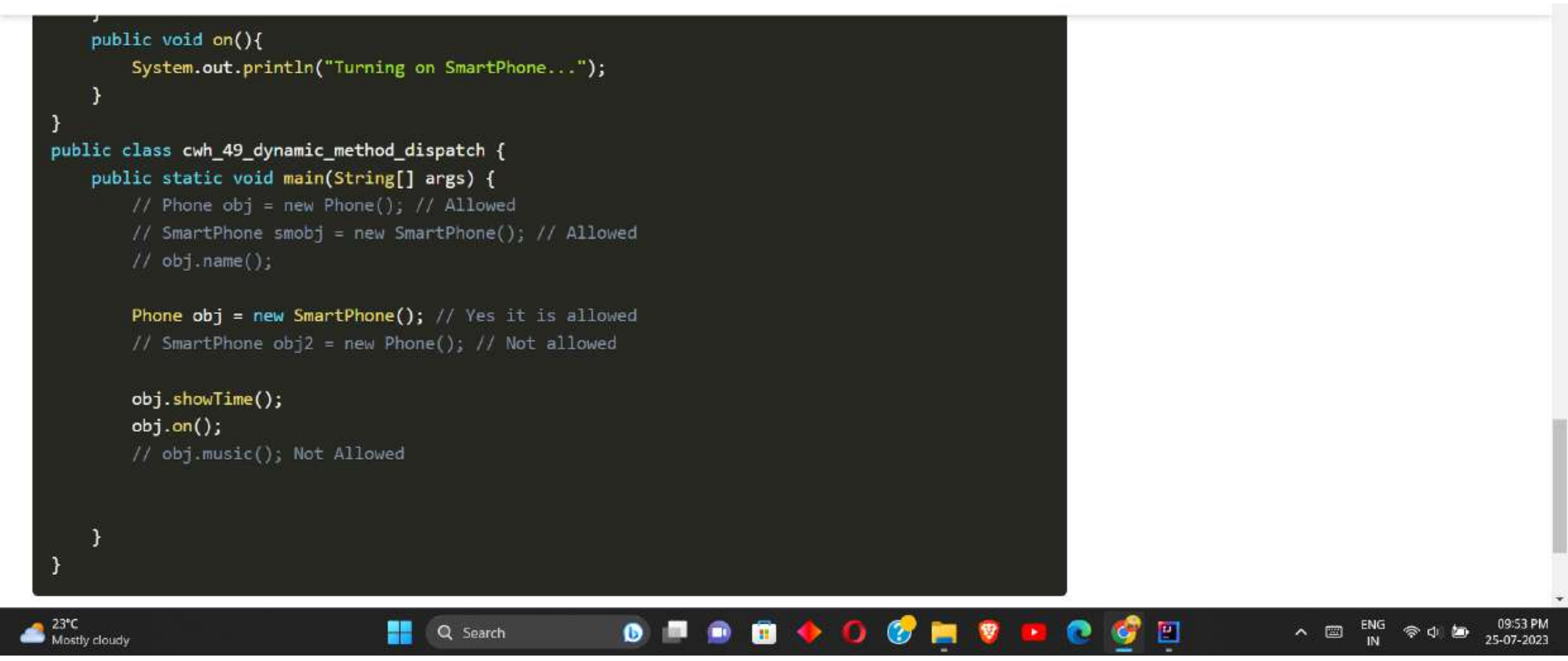
Note: The data members can not achieve the run time polymorphism.

Code as described/written in the video :

```
package com.company;
class Phone{
    public void showTime(){
        System.out.println("Time is 8 am");
    }
    public void on(){
        System.out.println("Turning on Phone...");
    }
}

class SmartPhone extends Phone{
    public void music(){
        System.out.println("Playing music...");
    }
    public void on(){
        System.out.println("Turning on SmartPhone...");
    }
```



```java
class SmartPhone extends Phone{
    public void music(){
        System.out.println("Playing music...");
    }
    public void on(){
        System.out.println("Turning on SmartPhone...");
    }
}
public class cwh_49_dynamic_method_dispatch {
    public static void main(String[] args) {
        // Phone obj = new Phone(); // Allowed
        // SmartPhone smobj = new SmartPhone(); // Allowed
        // obj.name();

        Phone obj = new SmartPhone(); // Yes it is allowed
        // SmartPhone obj2 = new Phone(); // Not allowed

        obj.showTime();
        obj.on();
        // obj.music(); Not Allowed
```



```java
    public void on(){
        System.out.println("Turning on SmartPhone...");
    }
}
public class cwh_49_dynamic_method_dispatch {
    public static void main(String[] args) {
        // Phone obj = new Phone(); // Allowed
        // SmartPhone smobj = new SmartPhone(); // Allowed
        // obj.name();

        Phone obj = new SmartPhone(); // Yes it is allowed
        // SmartPhone obj2 = new Phone(); // Not allowed

        obj.showTime();
        obj.on();
        // obj.music(); Not Allowed


    }
}
```