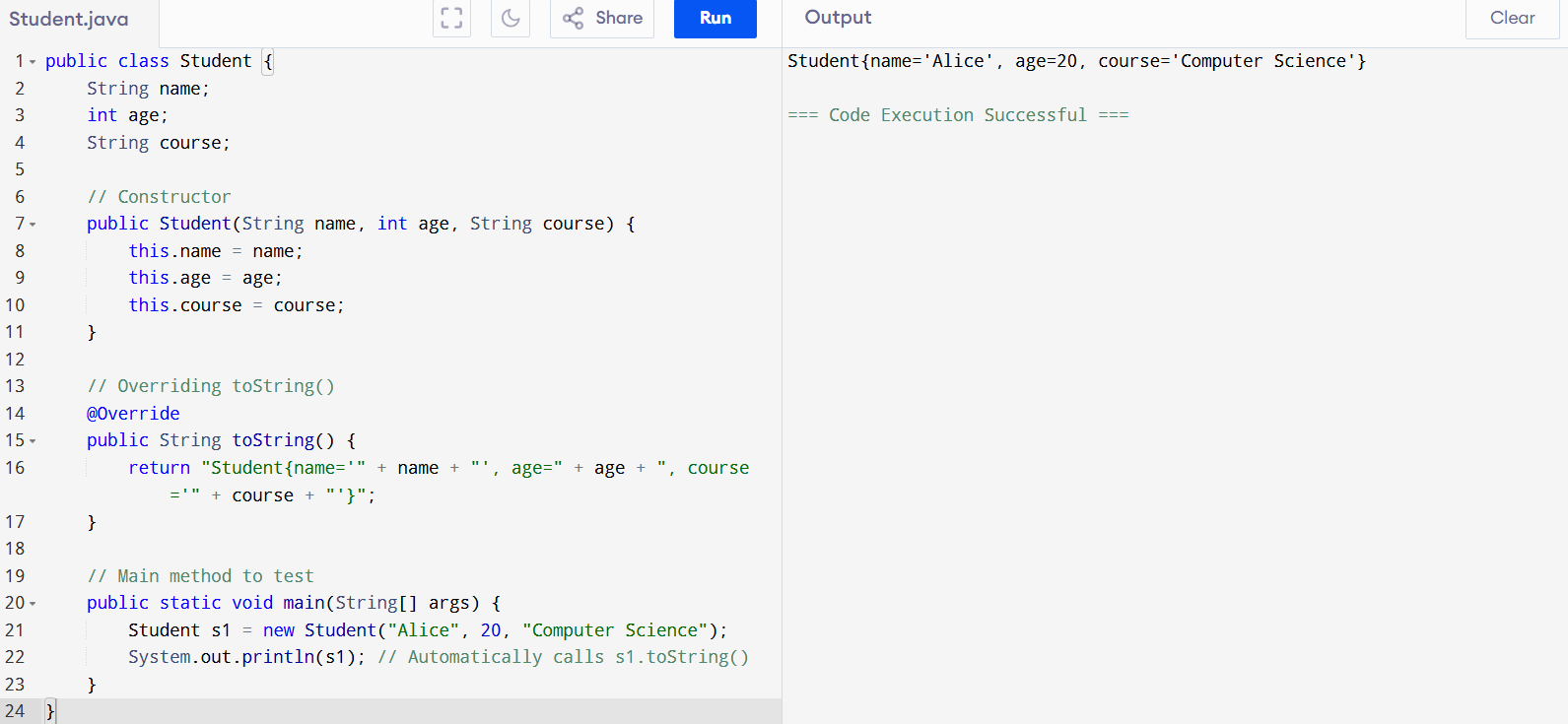
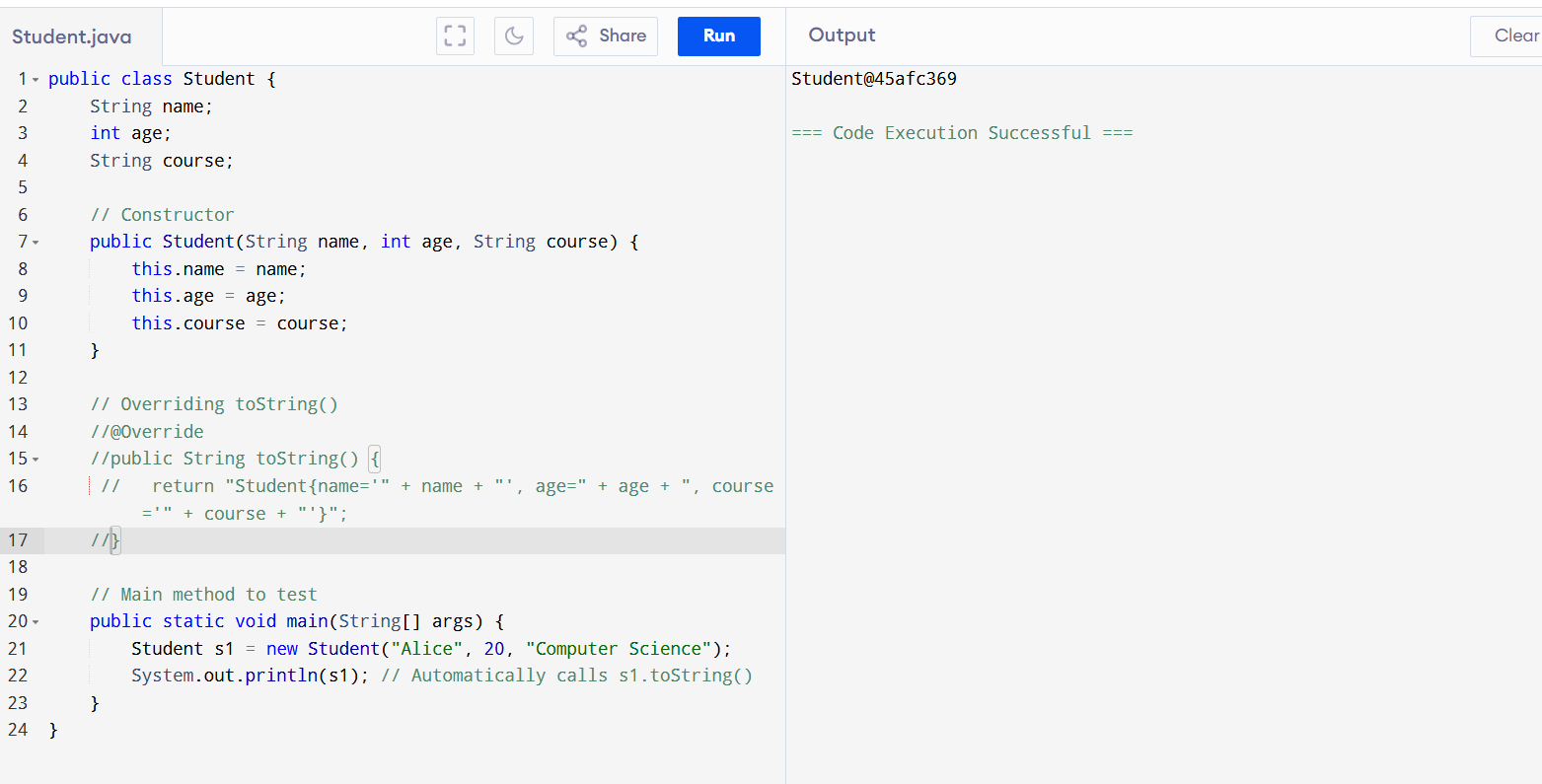
Task-1: With override



Task-2: Without Override:



public class Student {

String name;

int age;

String course;

// Constructor

public Student(String name, int age, String course) {

this.name = name;

this.age = age;

this.course = course;

}

// Overriding toString()

//@Override

//public String toString() {

// return "Student{name='" + name + "', age=" + age + ", course='" + course + "'}";

//}

// Main method to test

public static void main(String[] args) {

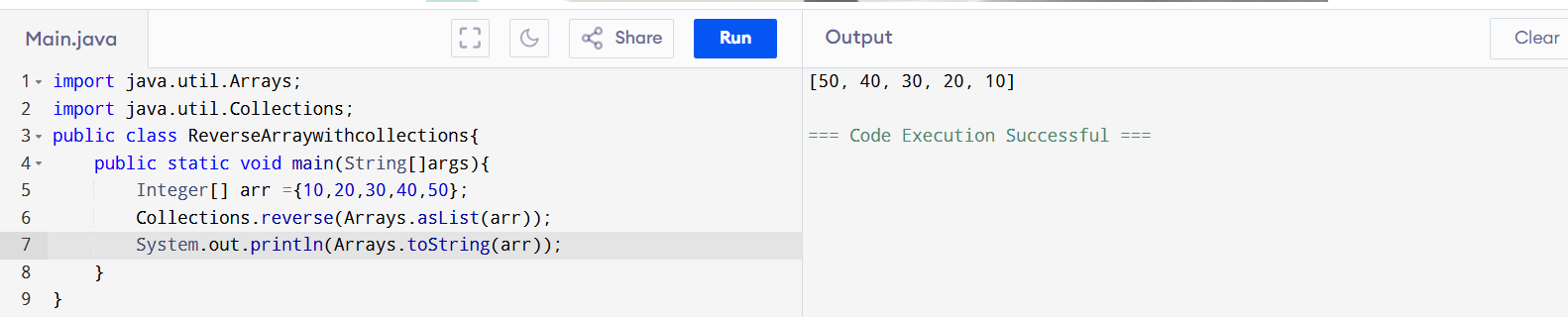
Student s1 = new Student("Alice", 20, "Computer Science");

System.out.println(s1); // Automatically calls s1.toString()

}

}

Task-3:Reverse array



import java.util.Arrays;

import java.util.Collections;

public class ReverseArraywithcollections{

public static void main(String[]args){

Integer[] arr ={10,20,30,40,50};

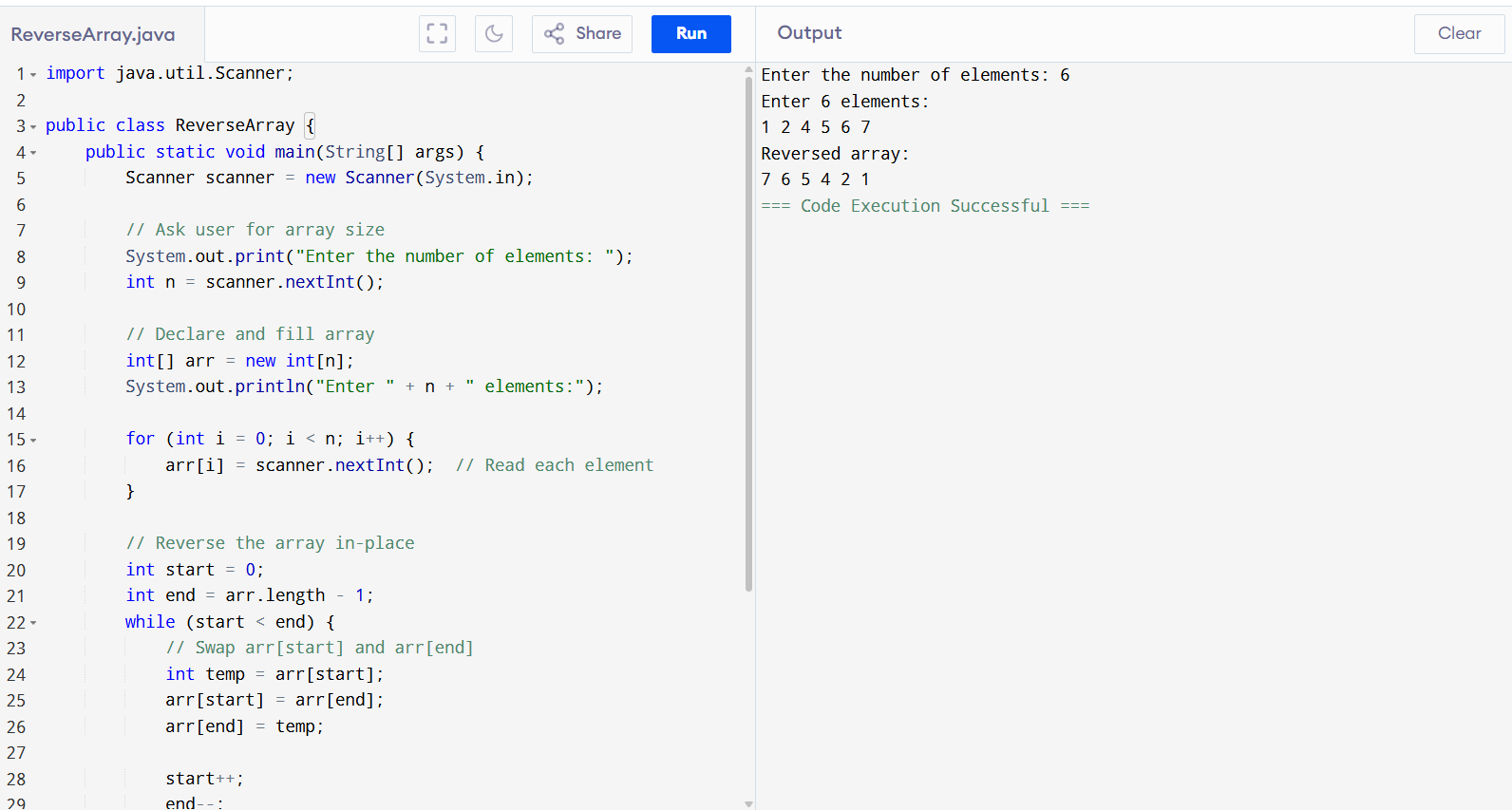
Collections.reverse(Arrays.asList(arr));

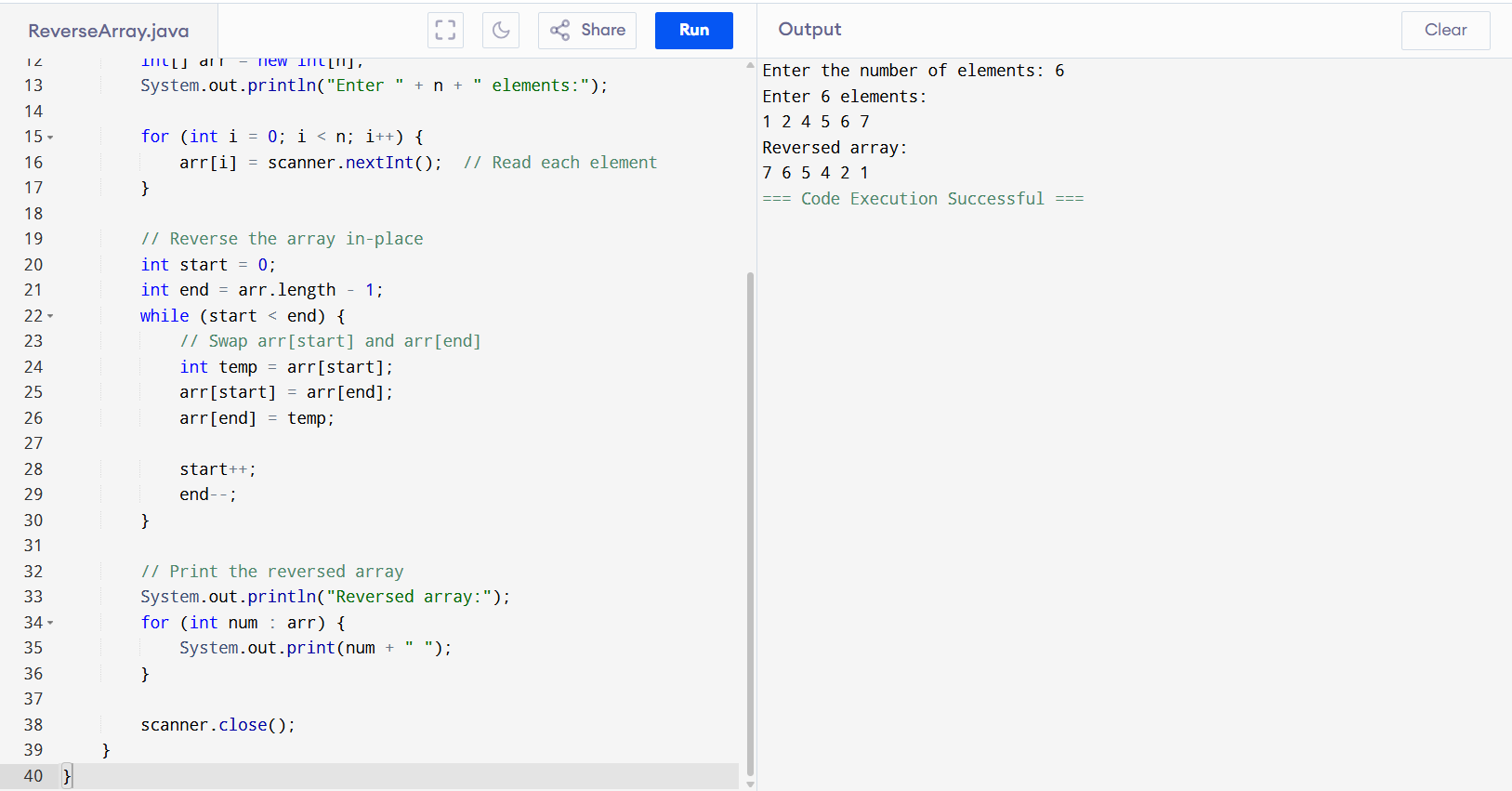
System.out.println(Arrays.toString(arr));

}

}

Reverse Array with inputs:





Task-4: With super class

public class Task04 extends Customer {

void billing(){

String items = "onions";

int cost = 30;

super.items = "Potatoes";

super.cost = 50;

super.purchage\_list();

System.out.println(items);

System.out.println(cost);

System.out.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

System.out.println(super.items);

System.out.println(super.cost);

// return 0;

}

public static void main(String[] args){

Customer cobj =new Customer();

cobj.purchage\_list();

Task033 tobj = new Task033();

tobj.billing();

}

}

class Customer{

int cost = 40;

String items = "Tomatoes";

Customer(){

System.out.println("Constructor called");

}

void purchage\_list(){

System.out.println("cost of tomatoes in Customer class is "+ cost);

}

}

/\*

\* output:

\*

\* Constructor called

cost of tomatoes in Customer class is 40

Constructor called

cost of tomatoes in Customer class is 50

onions

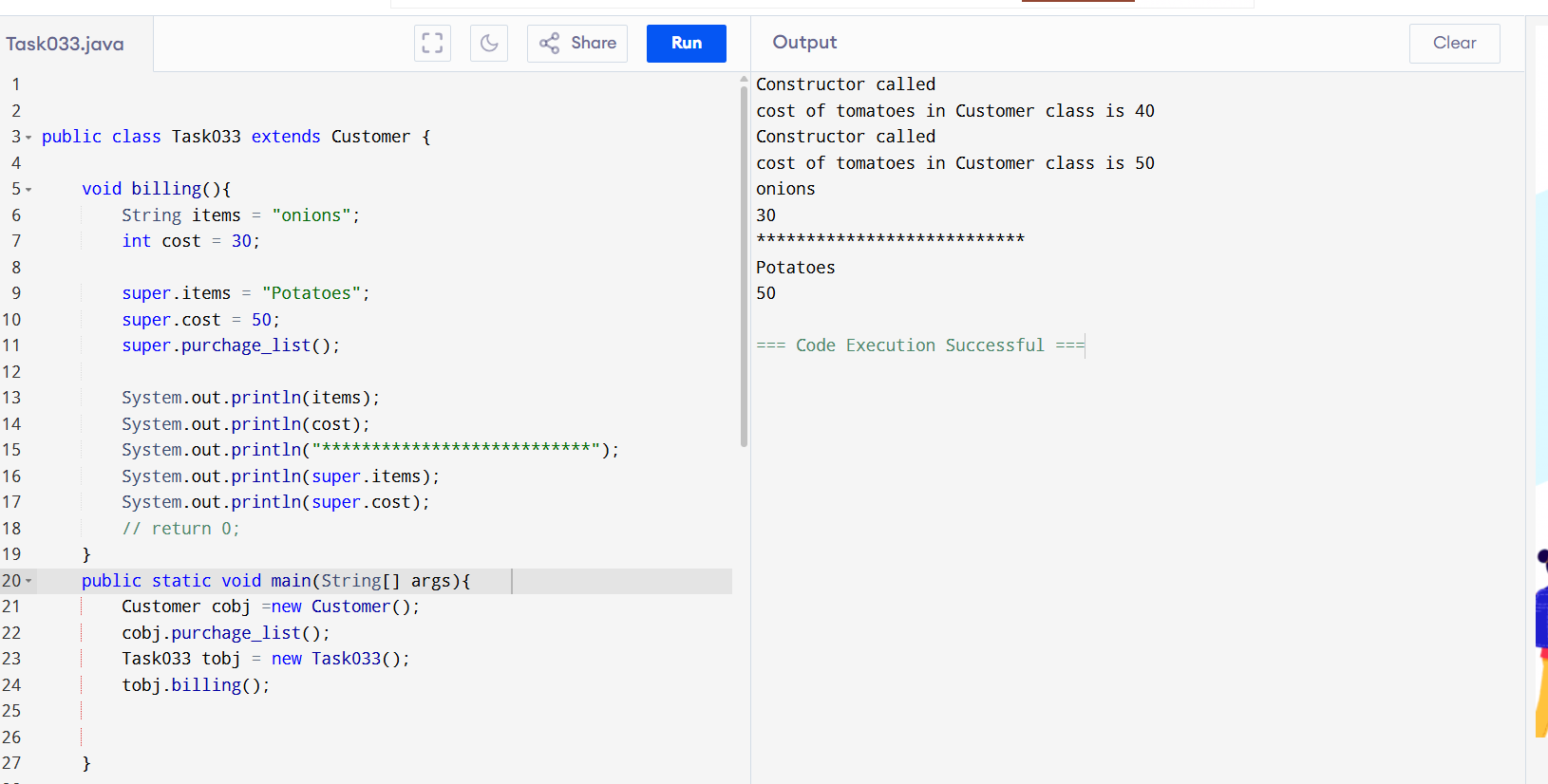
30

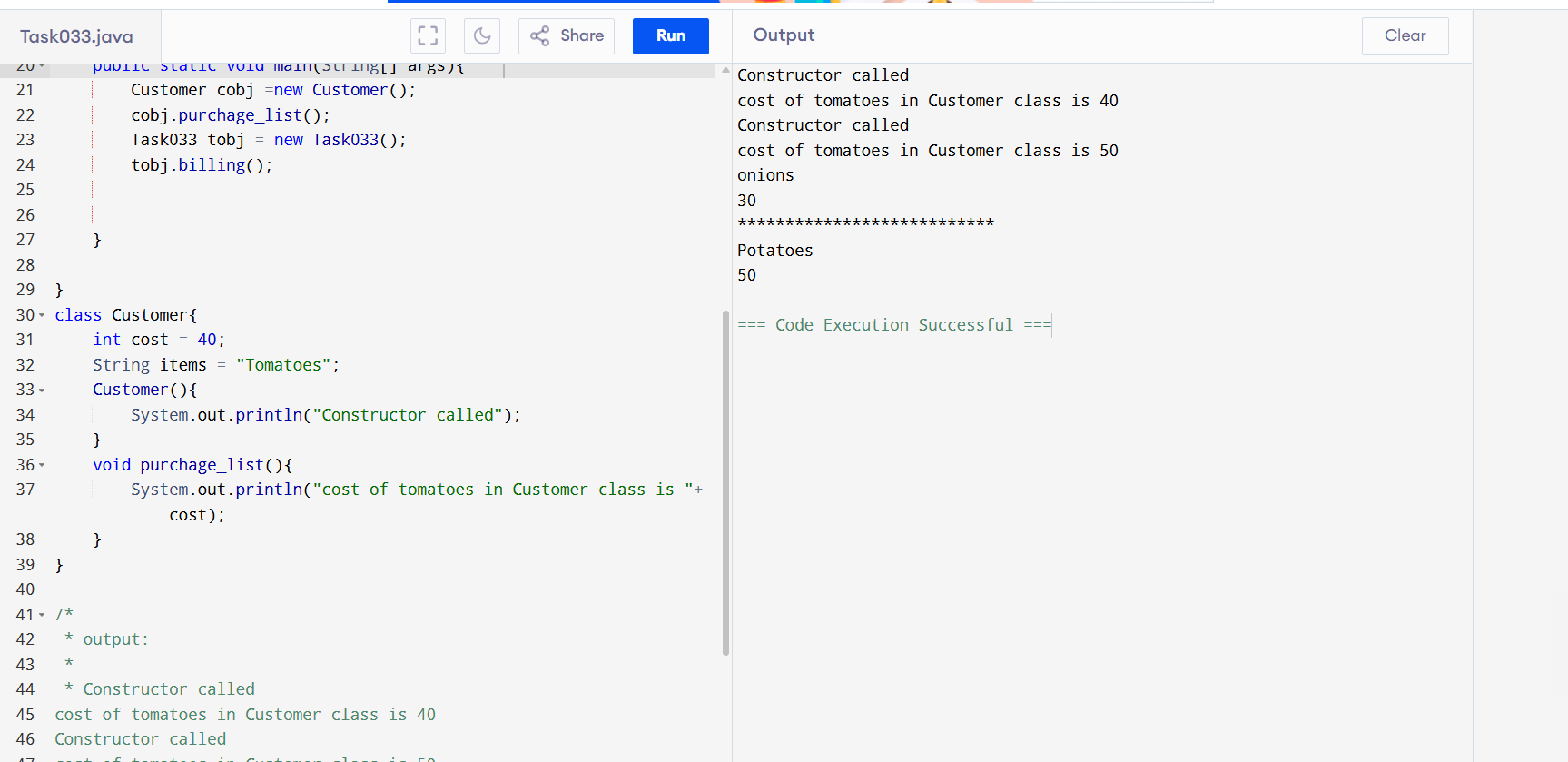
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Potatoes

50

\*/





Task-5:

class Superclass{

int var;

Superclass(int var){

this.var = var;

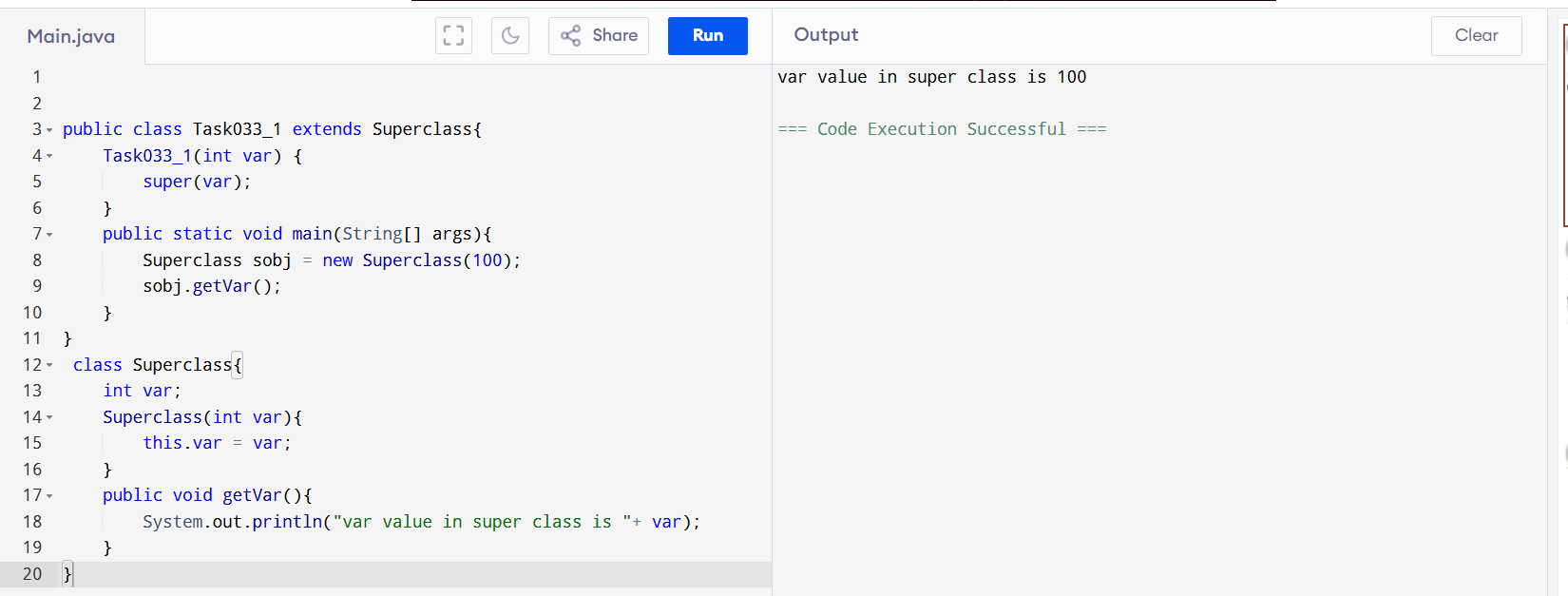
}

public void getVar(){

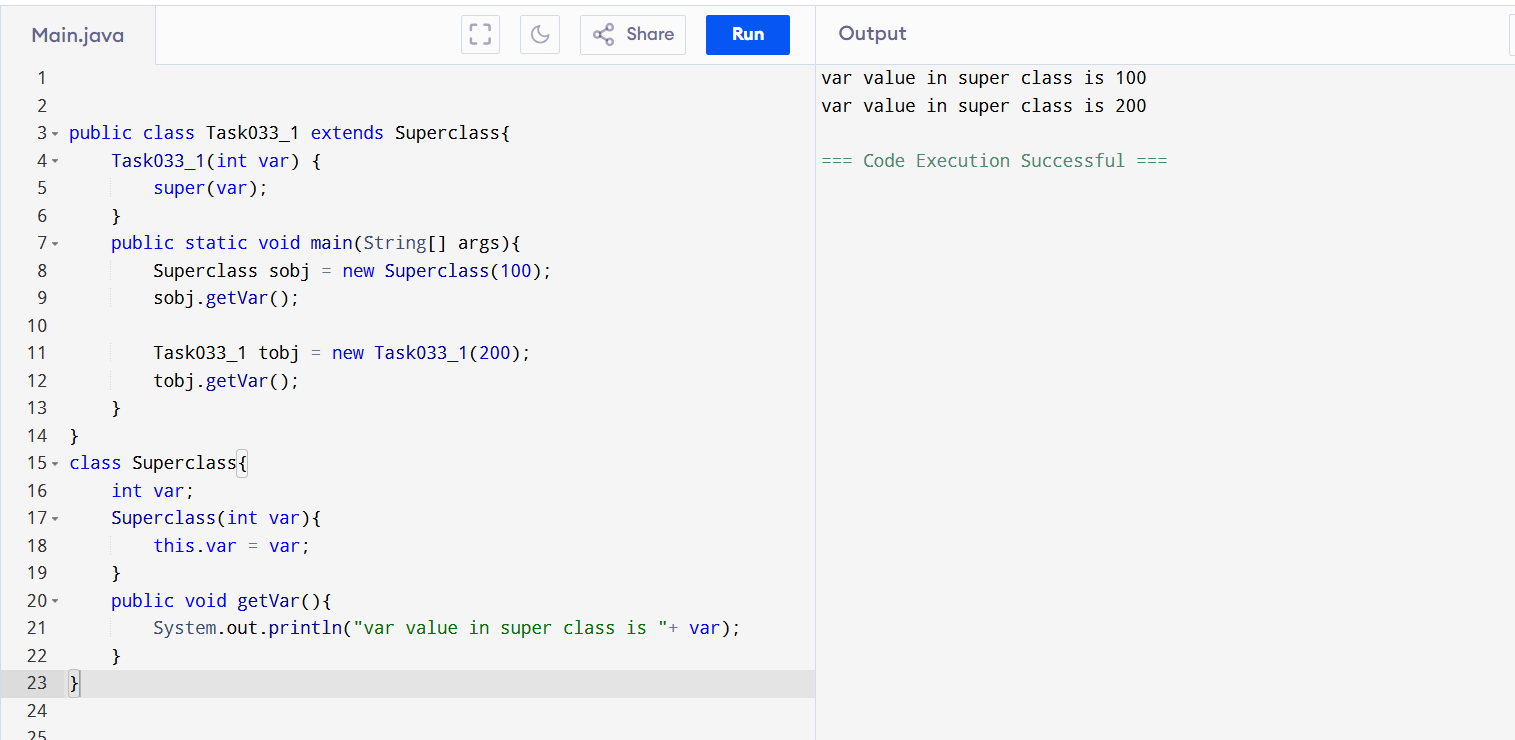
System.out.println("var value in super class is "+ var);

}

}



Task-6:



public class Task033\_1 extends Superclass{

Task033\_1(int var) {

super(var);

}

public static void main(String[] args){

Superclass sobj = new Superclass(100);

sobj.getVar();

Task033\_1 tobj = new Task033\_1(200);

tobj.getVar();

}

}

class Superclass{

int var;

Superclass(int var){

this.var = var;

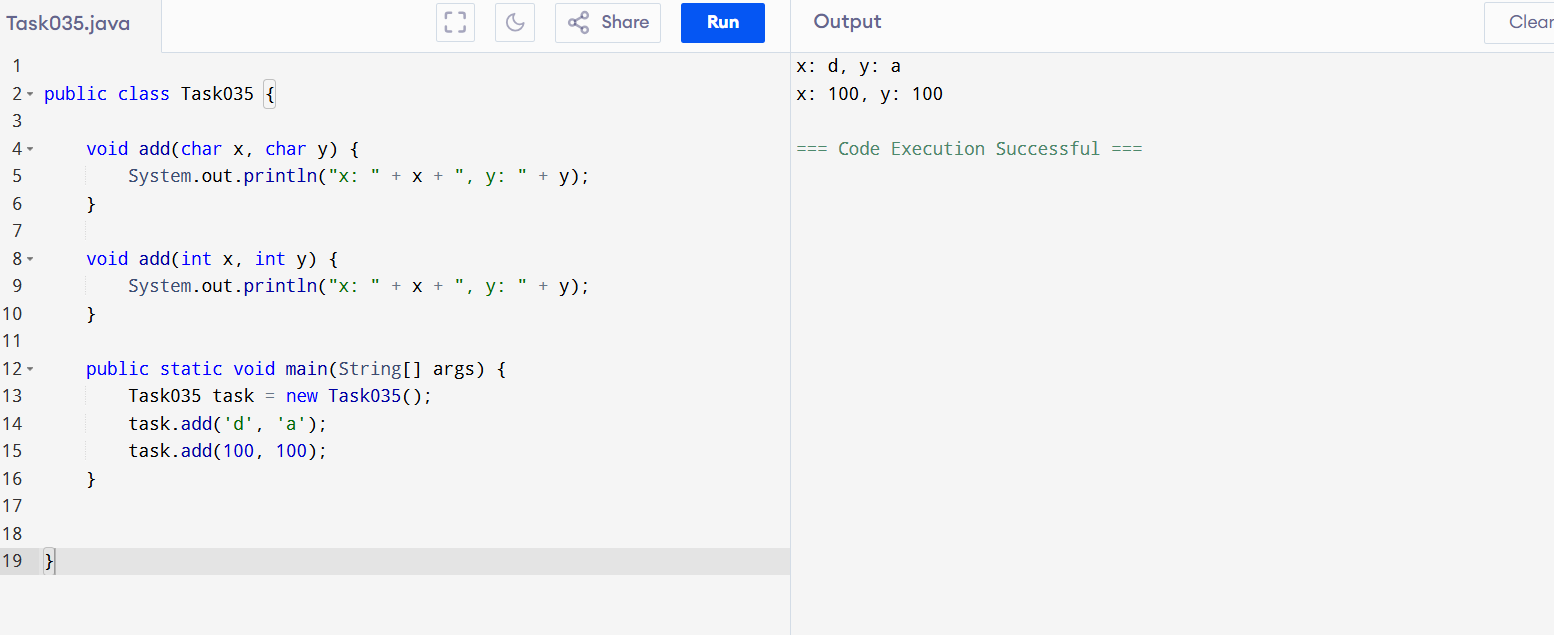
}

public void getVar(){

System.out.println("var value in super class is "+ var);

}

}

Task-8:

public class Task035 {

    void add(char x, char y) {

        System.out.println("x: " + x + ", y: " + y);

    }

    void add(int x, int y) {

        System.out.println("x: " + x + ", y: " + y);

    }

    public static void main(String[] args) {

        Task035 task = new Task035();

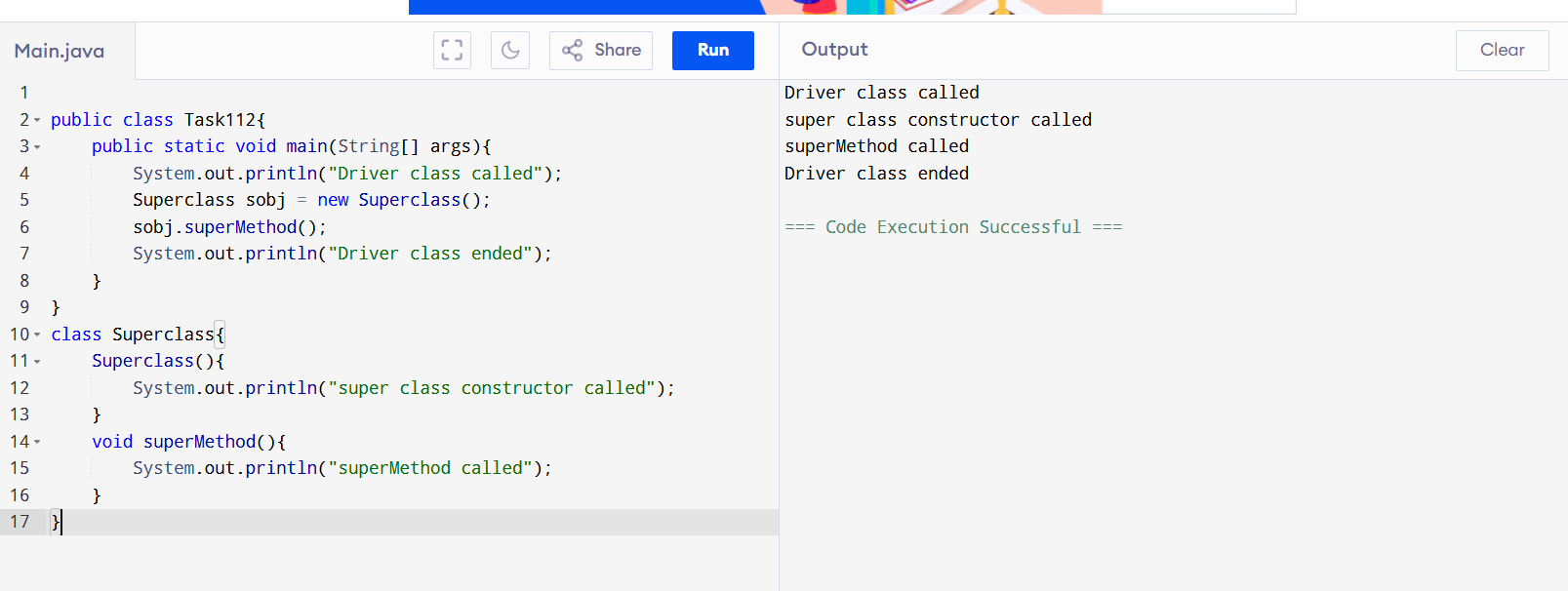
        task.add('d', 'a');

        task.add(100, 100);

    }

}

Task-9:



public class Task112{

public static void main(String[] args){

System.out.println("Driver class called");

Superclass sobj = new Superclass();

sobj.superMethod();

System.out.println("Driver class ended");

}

}

class Superclass{

Superclass(){

System.out.println("super class constructor called");

}

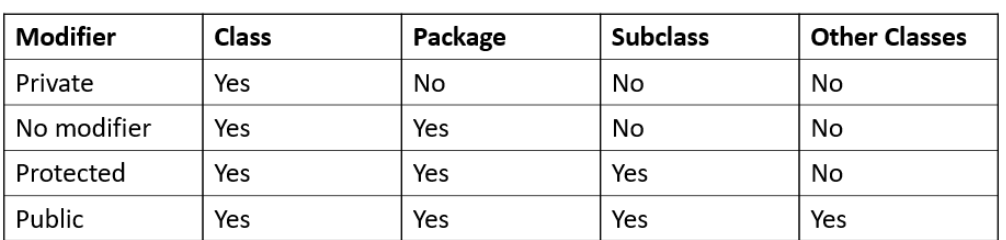
void superMethod(){

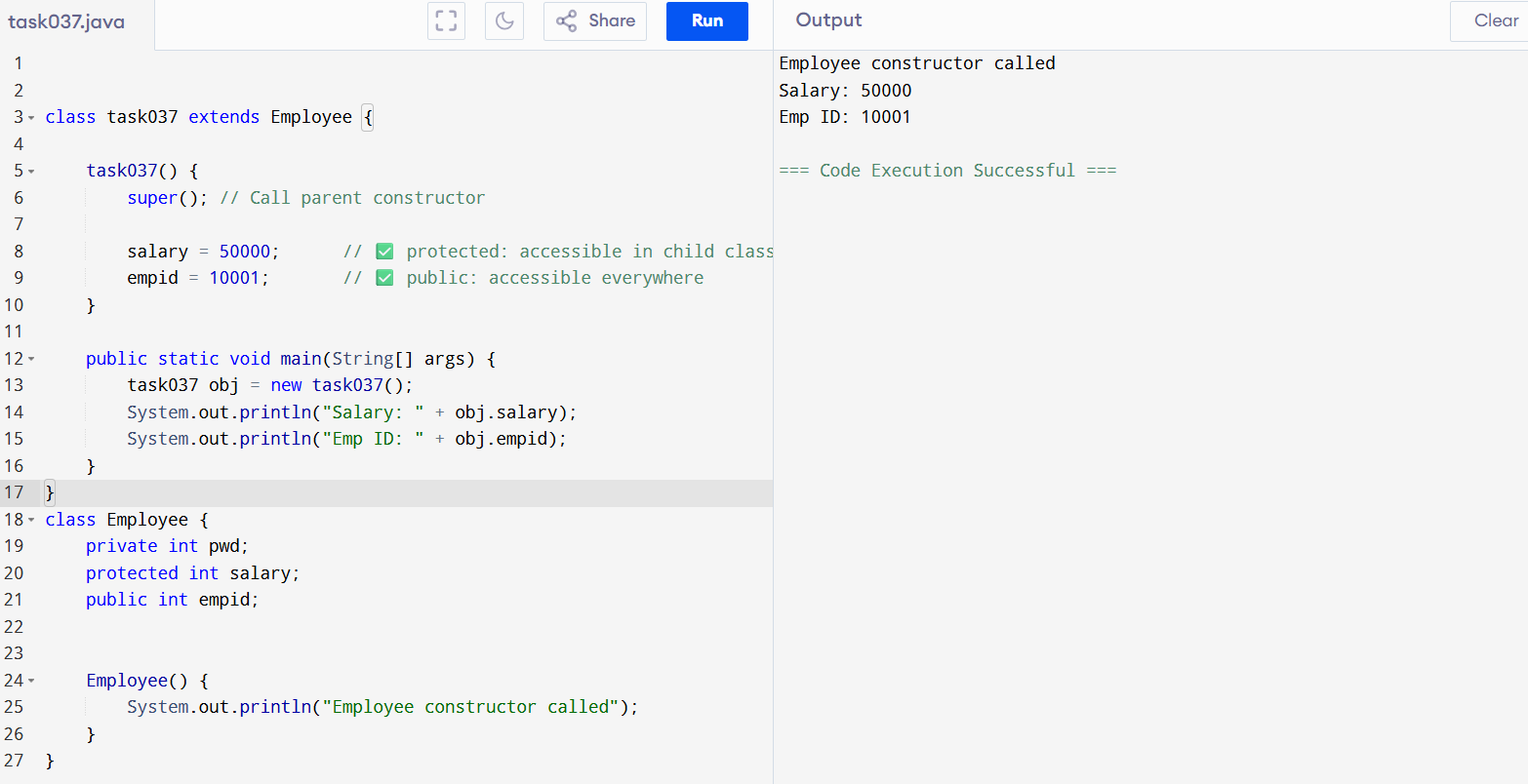
System.out.println("superMethod called");

}

}

**Encapsulation:**





Encapsulation code:

class task037 extends Employee {

task037() {

super(); // Call parent constructor

salary = 50000; // ✅ protected: accessible in child class

empid = 10001; // ✅ public: accessible everywhere

}

public static void main(String[] args) {

task037 obj = new task037();

System.out.println("Salary: " + obj.salary);

System.out.println("Emp ID: " + obj.empid);

}

}

class Employee {

private int pwd;

protected int salary;

public int empid;

Employee() {

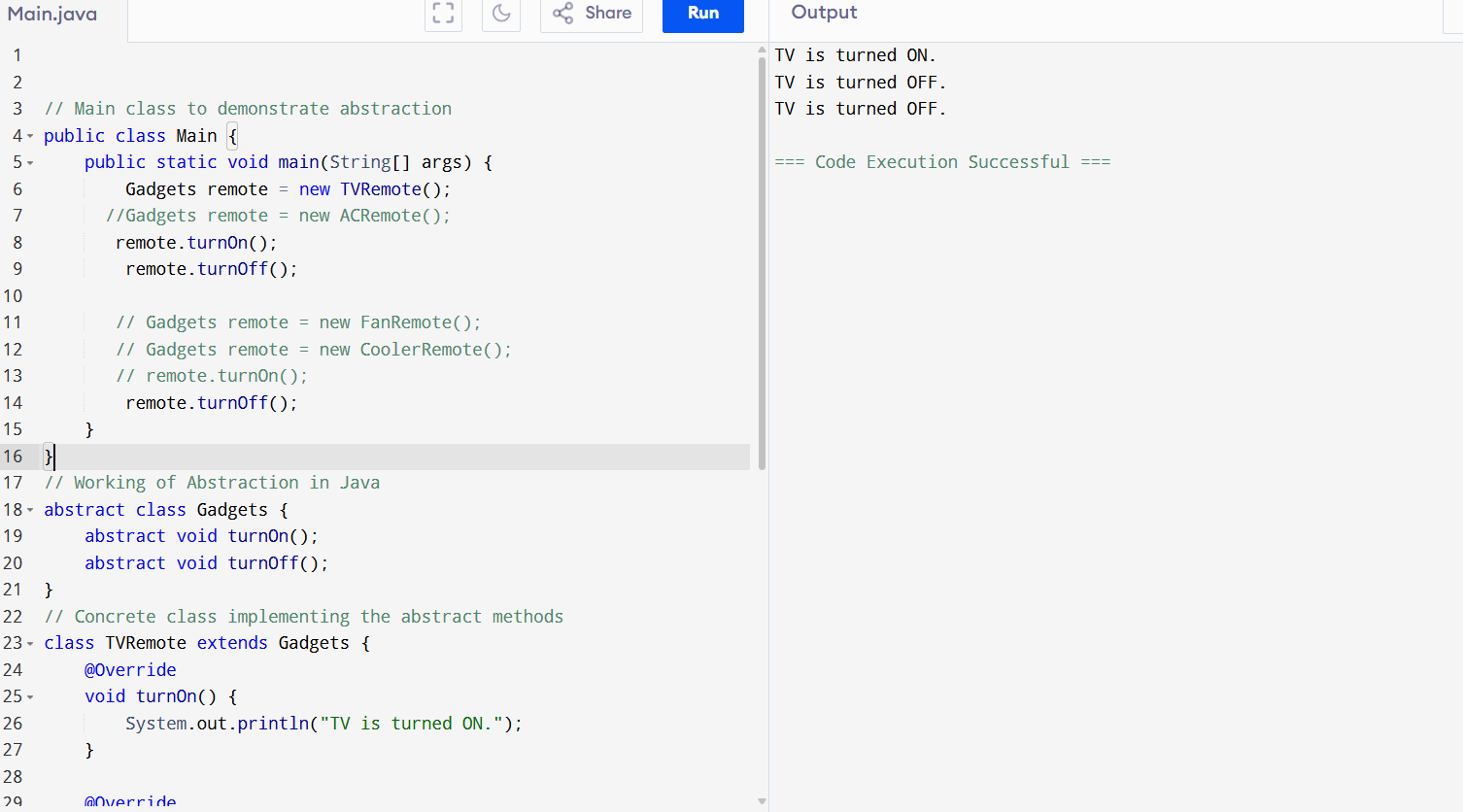
System.out.println("Employee constructor called");

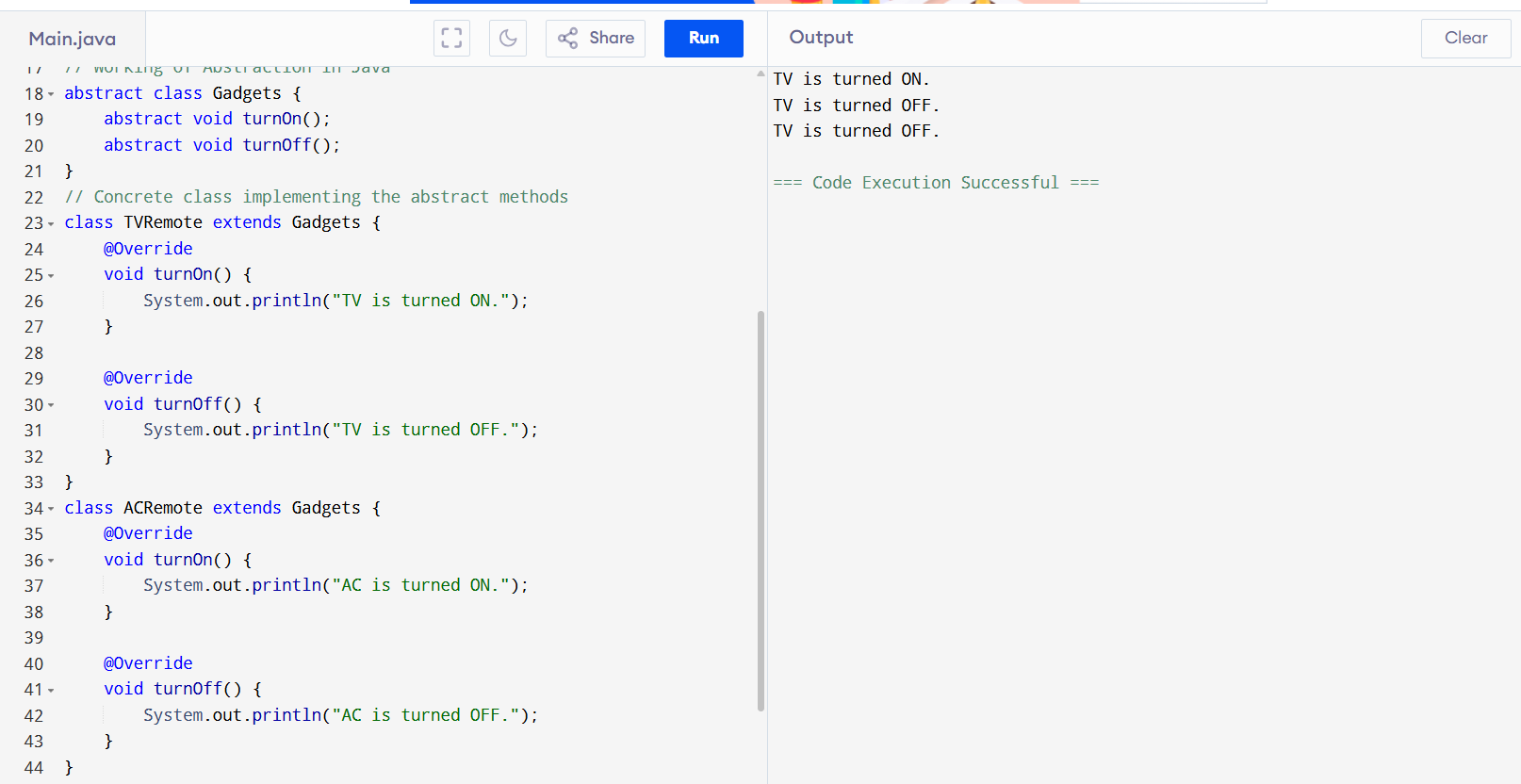
}

}

**Task-10**

**// Working of Abstraction in Java:**

****

****

// Main class to demonstrate abstraction

public class Main {

public static void main(String[] args) {

Gadgets remote = new TVRemote();

//Gadgets remote = new ACRemote();

remote.turnOn();

remote.turnOff();

// Gadgets remote = new FanRemote();

// Gadgets remote = new CoolerRemote();

// remote.turnOn();

remote.turnOff();

}

}

// Working of Abstraction in Java

abstract class Gadgets {

abstract void turnOn();

abstract void turnOff();

}

// Concrete class implementing the abstract methods

class TVRemote extends Gadgets {

@Override

void turnOn() {

System.out.println("TV is turned ON.");

}

@Override

void turnOff() {

System.out.println("TV is turned OFF.");

}

}

class ACRemote extends Gadgets {

@Override

void turnOn() {

System.out.println("AC is turned ON.");

}

@Override

void turnOff() {

System.out.println("AC is turned OFF.");

}

}

**Final and constant Difference:**

**Final:**

- Initialized at runtime

- Immutable after being set

- Should be initialized before the constructor called

**Constant:**

- Initialized at compile time

- Immutable forever

- Should be strictly initialized during declaration.