Day 8 - 07th June 2025

Task 21: Home Task

**Example:** This example demonstrates how to initialize an array and traverse it using a for loop to print each element.

public class Main {

   public static void main(String[] args)

   {

​

       // initializing array

       int[] arr = { 1, 2, 3, 4, 5 };

​

       // size of array

       int n = arr.length;

​

       // traversing array

       for (int i = 0; i < n; i++)

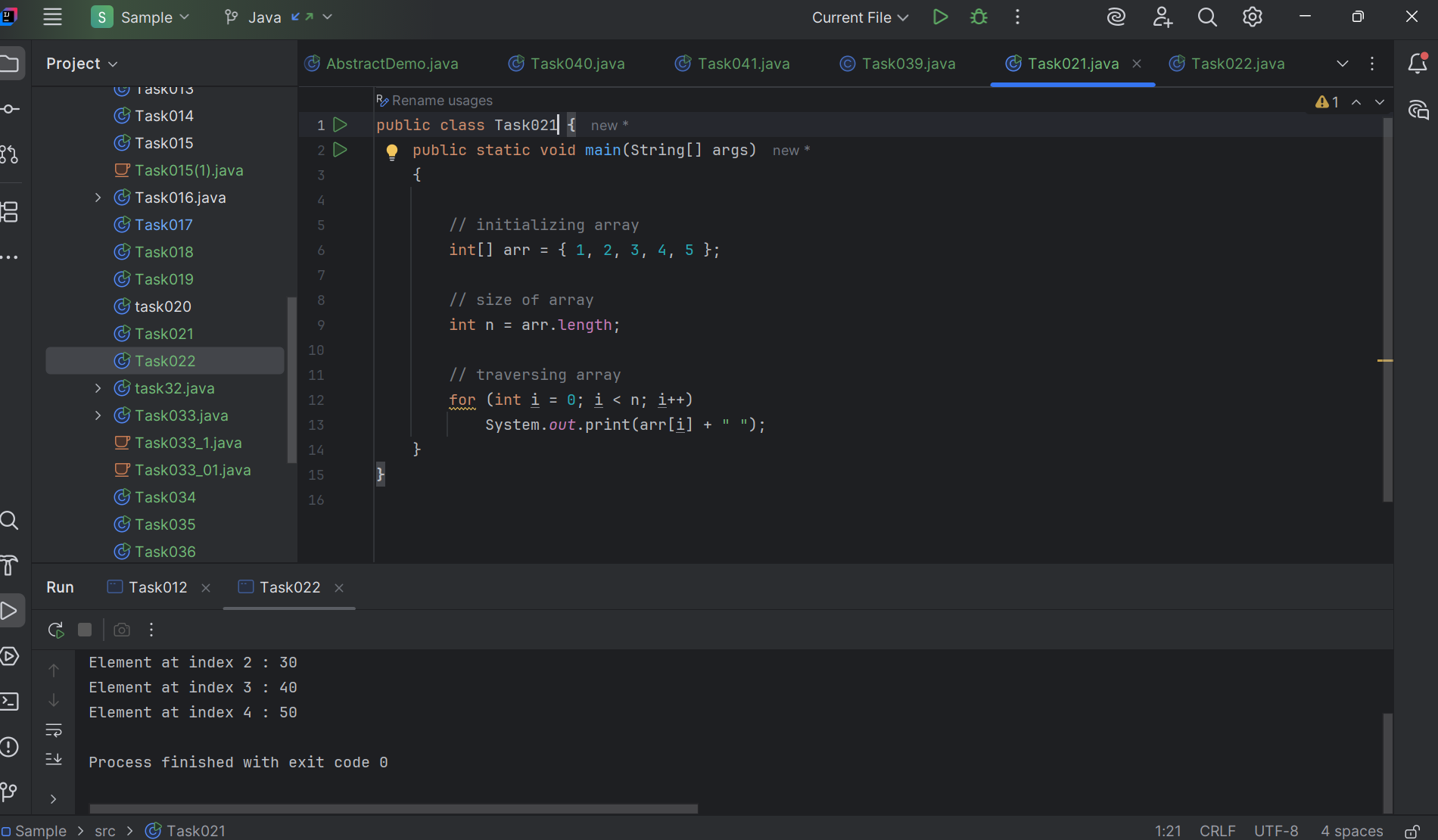
           System.out.print(arr[i] + " ");

   }

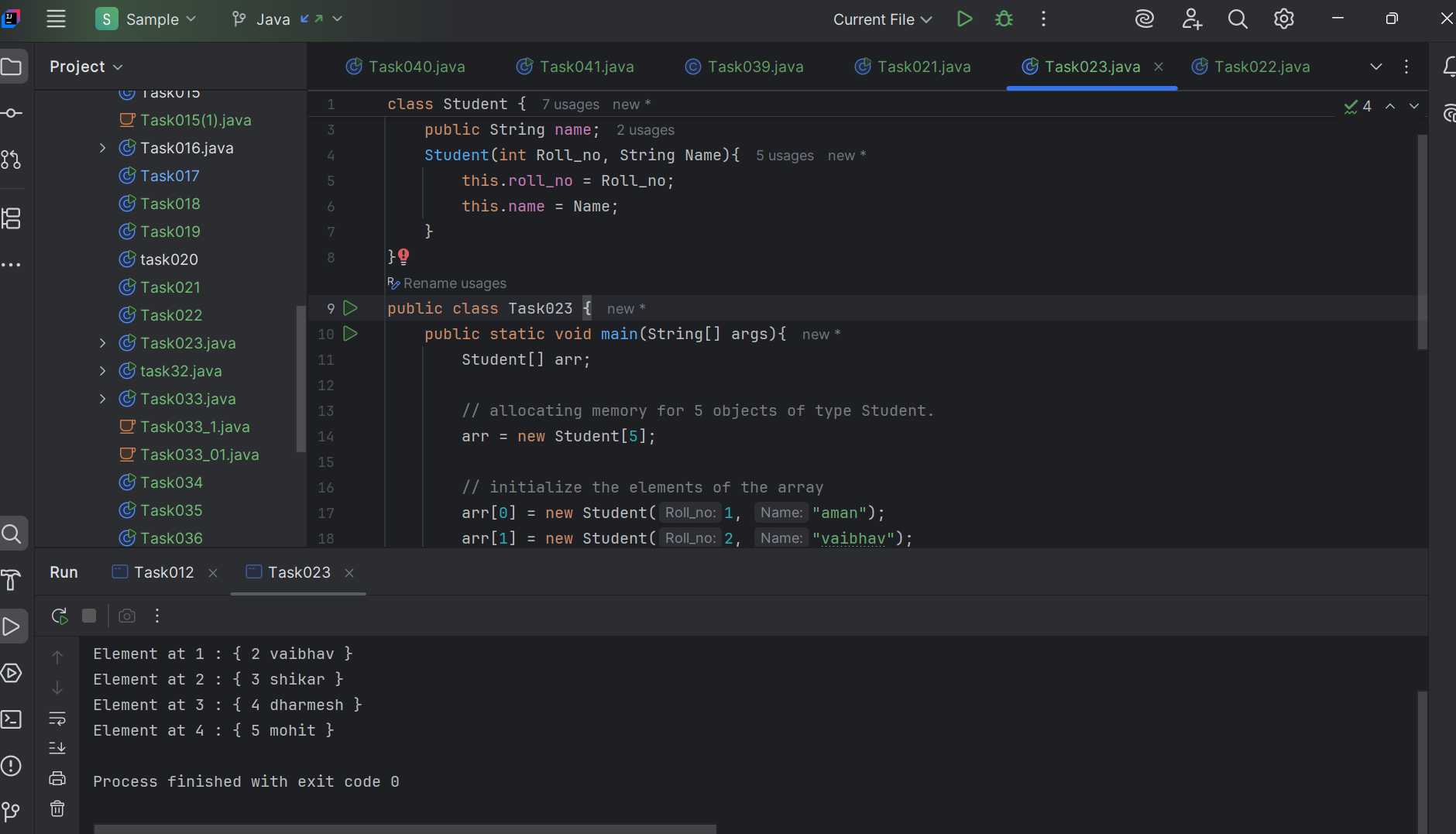
}

**Output**

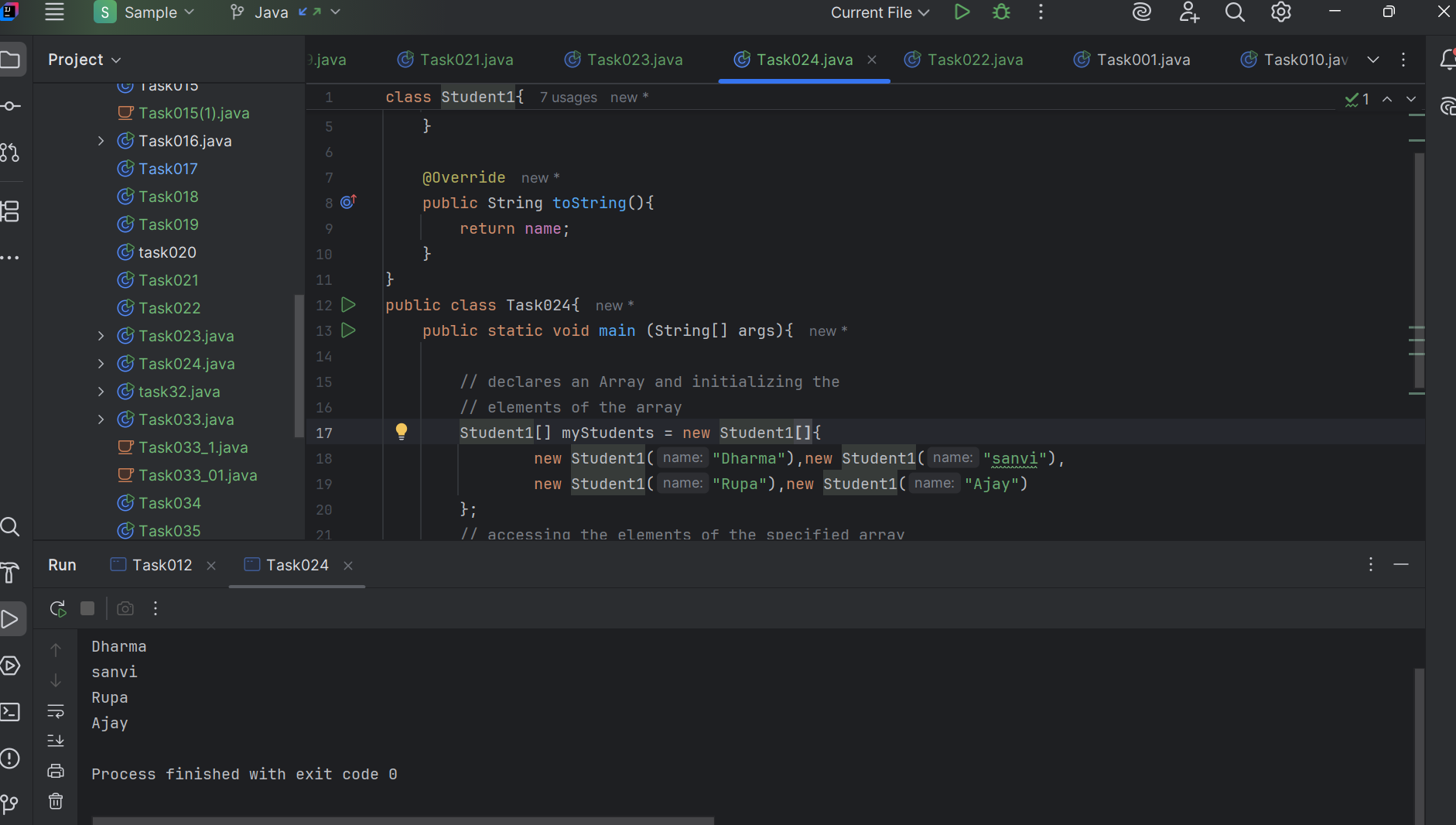
1 2 3 4 5



Task 23



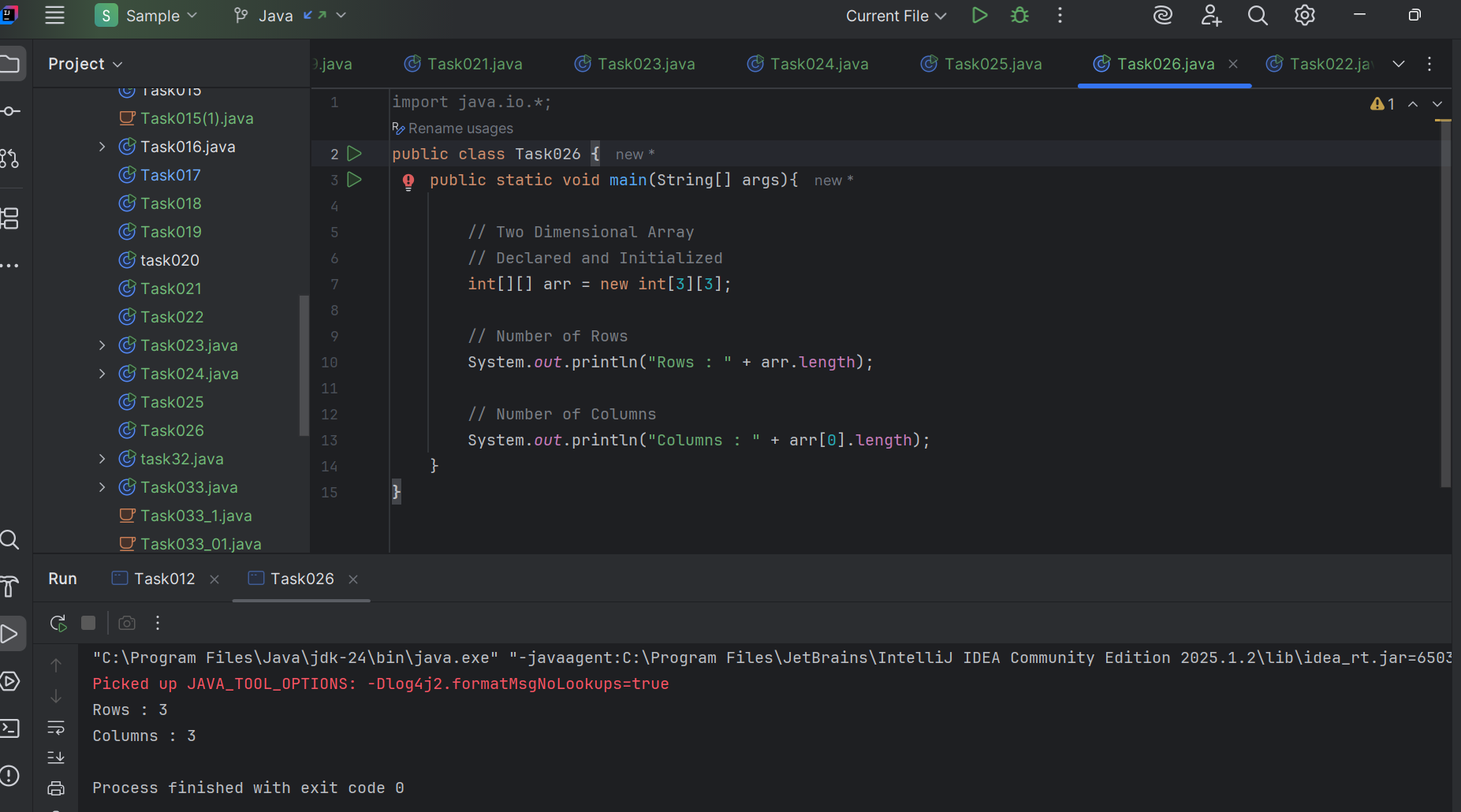
Task 24



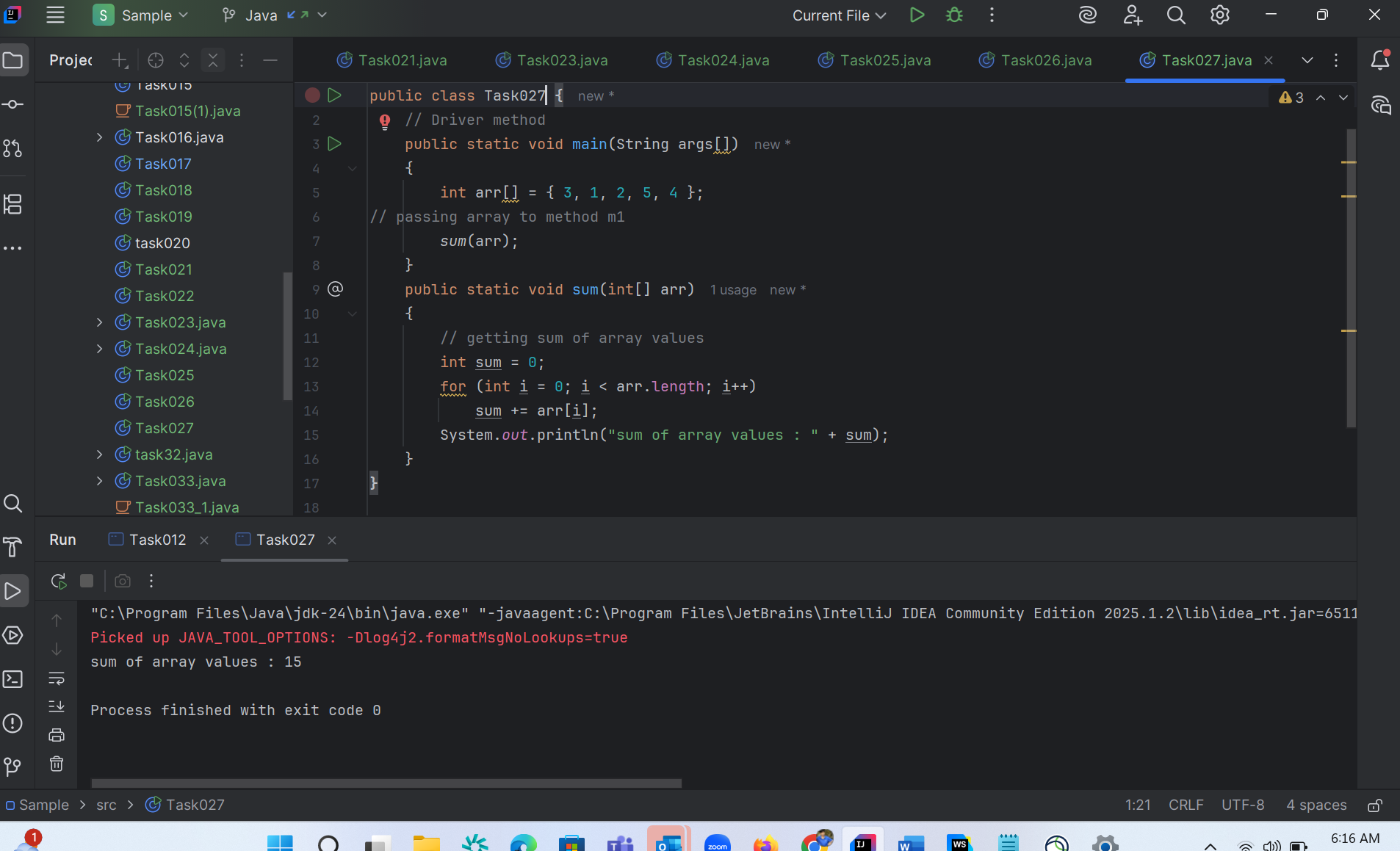
Task 25:



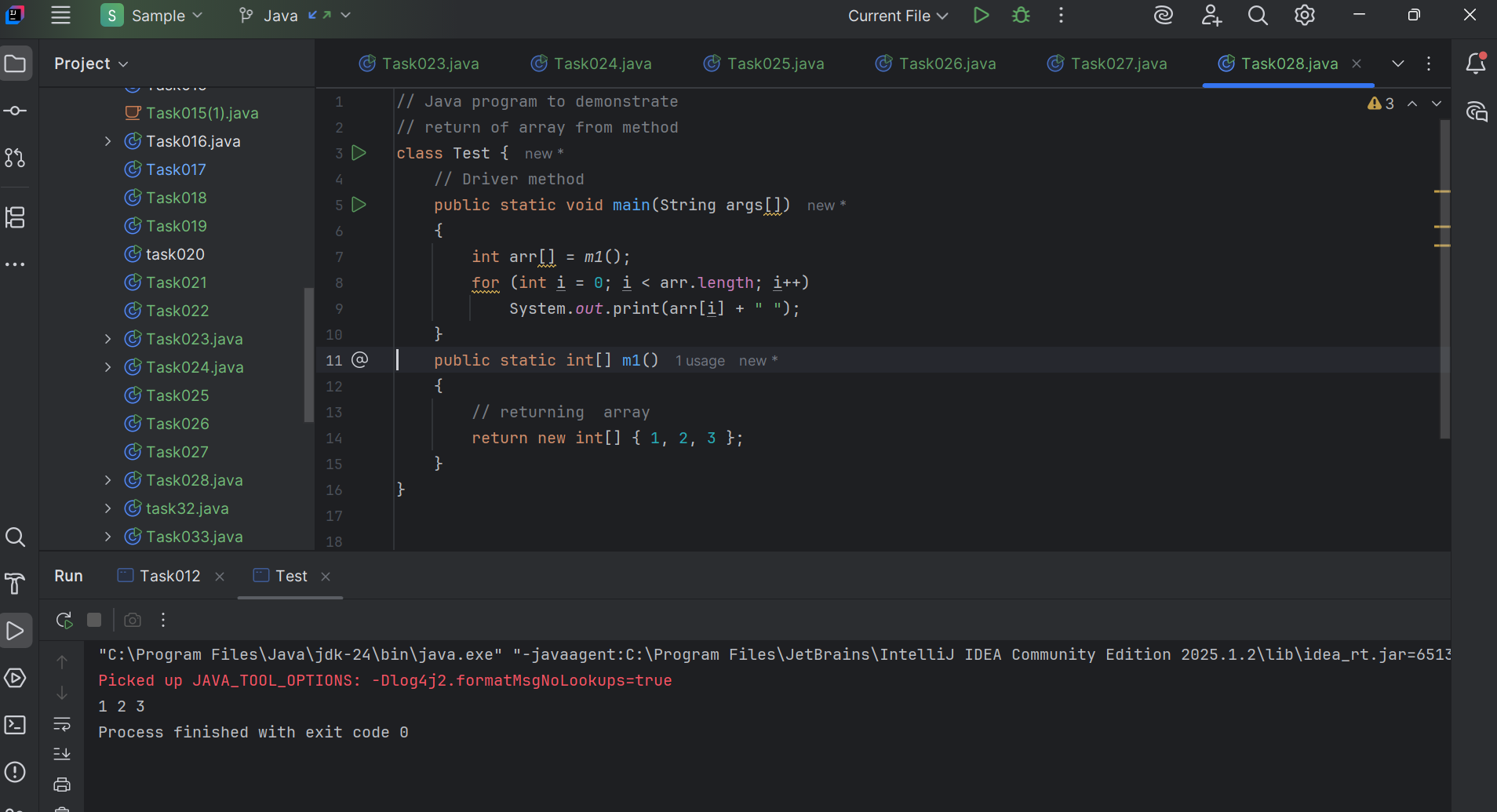
Task 26:



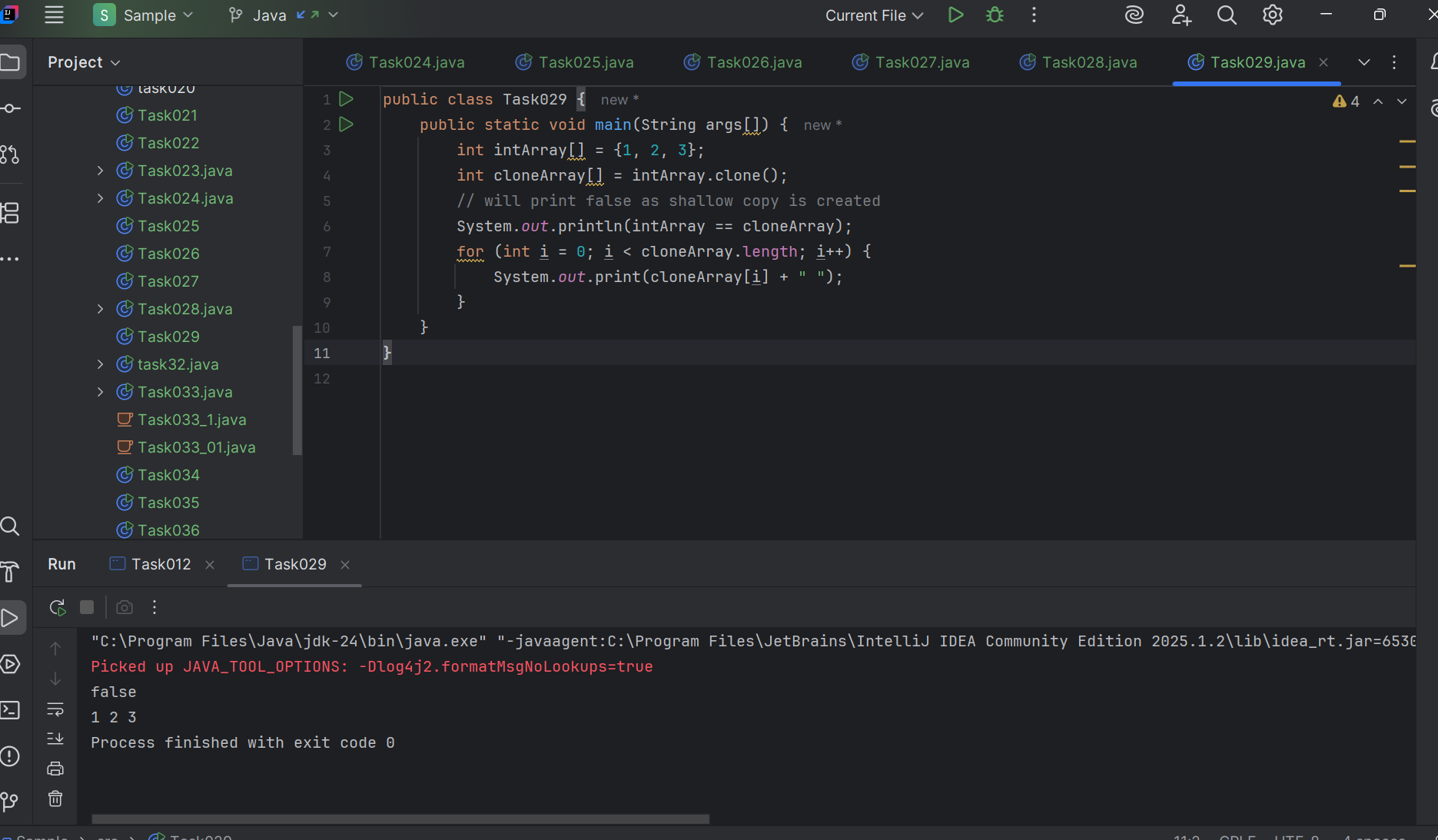
Task27:



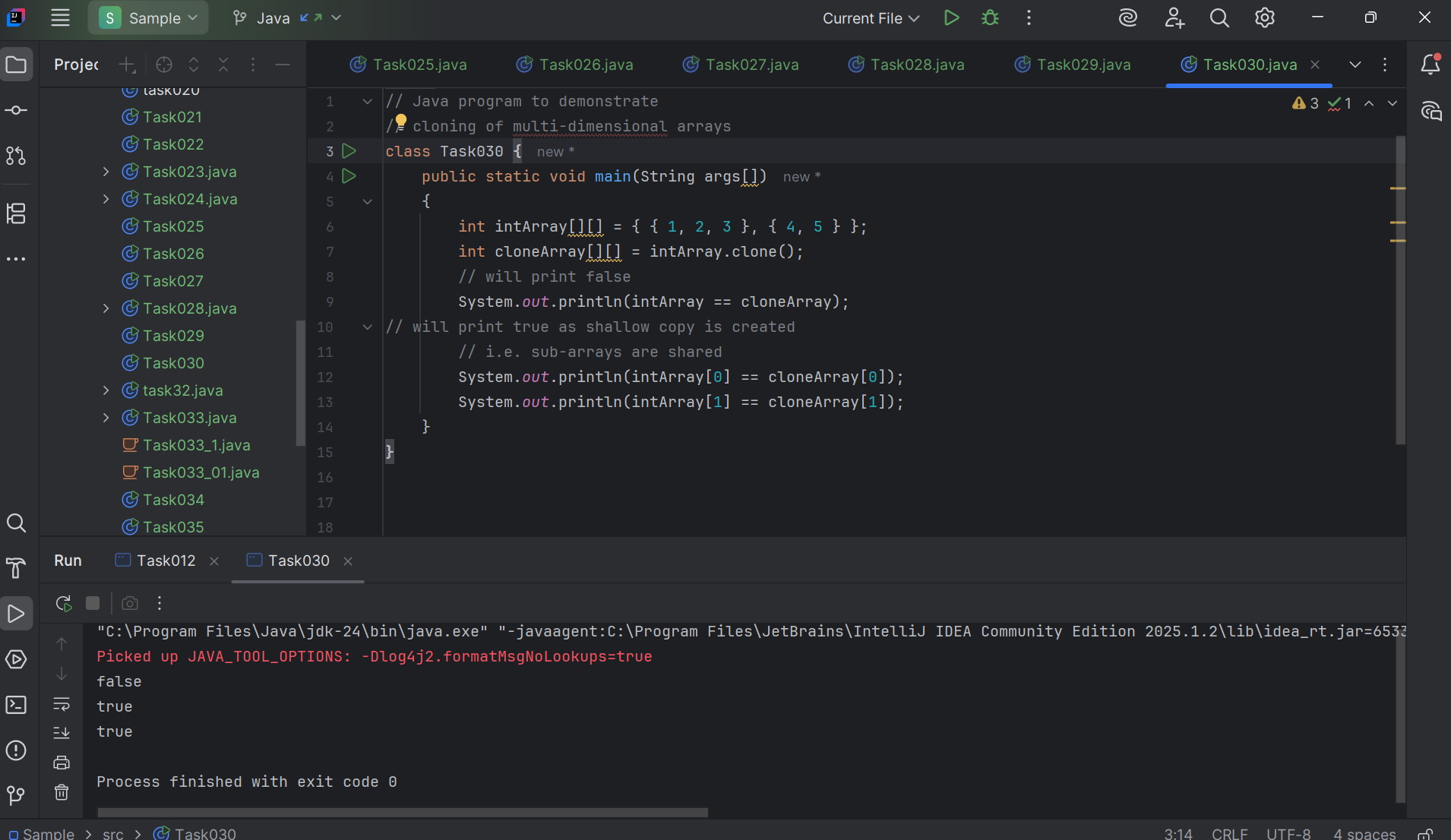
Task 28:



Task29:



Task 30:



Java OOPs

Task 31 :

Task 031

class Calculation {

   int z;

   public void addition(int x, int y) {

      z = x + y;

      System.out.println("The sum of the given numbers:"+z);

   }

   public void Subtraction(int x, int y) {

      z = x - y;

      System.out.println("The difference between the given numbers:"+z);

   }

}

public class My\_Calculation extends Calculation {

   public void multiplication(int x, int y) {

      z = x \* y;

      System.out.println("The product of the given numbers:"+z);

   }

   public static void main(String args[]) {

      int a = 20, b = 10;

      My\_Calculation demo = new My\_Calculation();

      demo.addition(a, b);

      demo.Subtraction(a, b);

      demo.multiplication(a, b);

   }

}

public class My\_Calculation2 extends Calculation {

   public void multiplication(int x, int y) {

      z = x \* y;

      System.out.println("The product of the given numbers:"+z);

   }

   public static void main(String args[]) {

      int a = 20, b = 10;

      My\_Calculation2 demo = new My\_Calculation2();

      demo.addition(a, b);

      demo.Subtraction(a, b);

      demo.multiplication(a, b);

   }

}

Inheritance provides - reusability

It avoids - duplication

Multi level inheritance

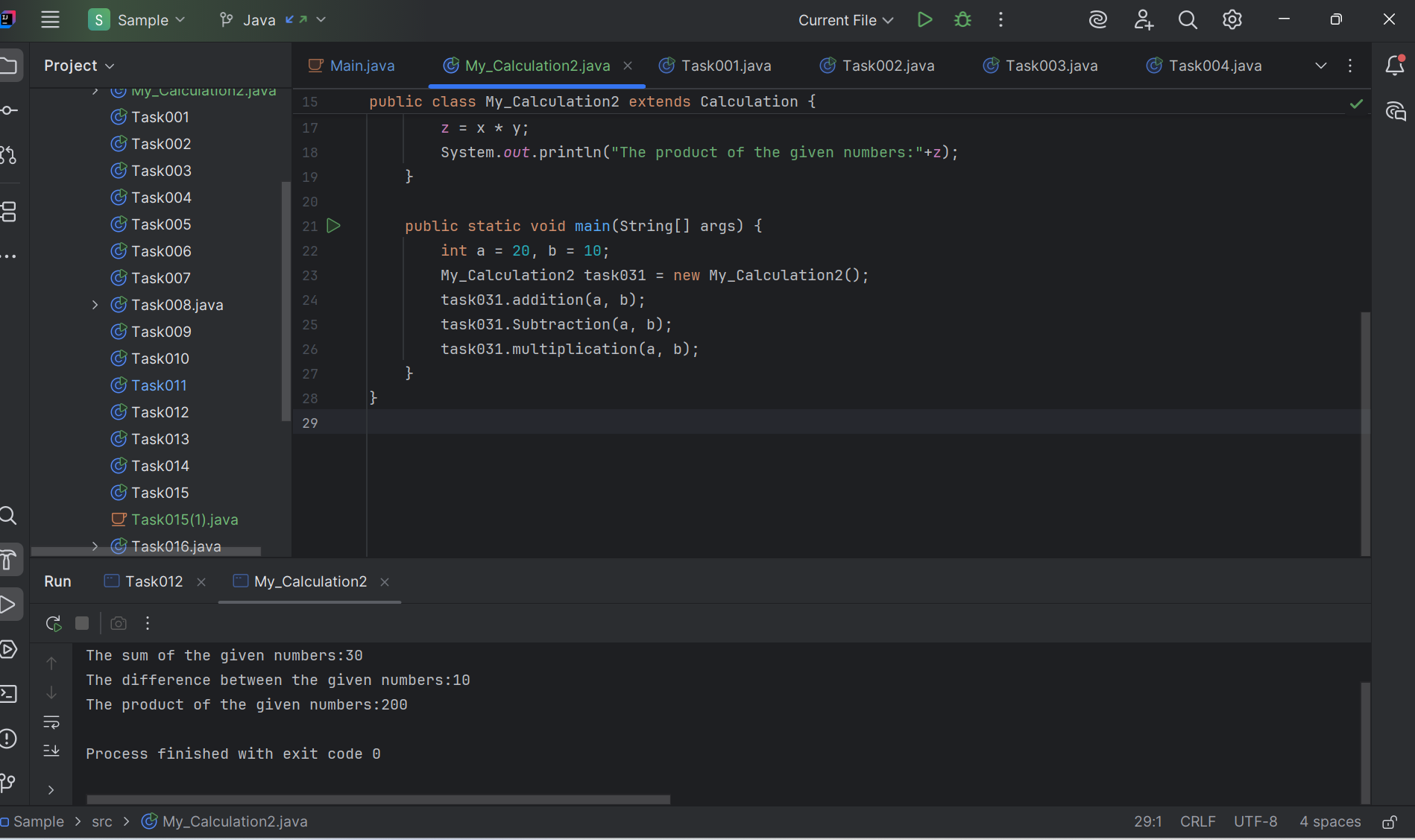
Clac < ========= My\_calculation < ======= calculation

Class calc extends My\_calculation{

}

Code :

class Calculation{  
 int z;  
  
 public void addition(int x, int y) {  
 z = x + y;  
 System.*out*.println("The sum of the given numbers:"+z);  
 }  
  
 public void Subtraction(int x, int y) {  
 z = x - y;  
 System.*out*.println("The difference between the given numbers:"+z);  
 }  
}  
  
public class My\_Calculation2 extends Calculation {  
 public void multiplication(int x, int y) {  
 z = x \* y;  
 System.*out*.println("The product of the given numbers:"+z);  
 }  
  
 public static void main(String[] args) {  
 int a = 20, b = 10;  
 My\_Calculation2 task031 = new My\_Calculation2();  
 task031.addition(a, b);  
 task031.Subtraction(a, b);  
 task031.multiplication(a, b);  
 }  
}



—--Task 032 ------------------------------------------------------------------------------

In the above code add a class clock — and try to extend calculation and clock in the my calculation class..   Is it possible ???? give reason.

class clock {

—--

—--

}

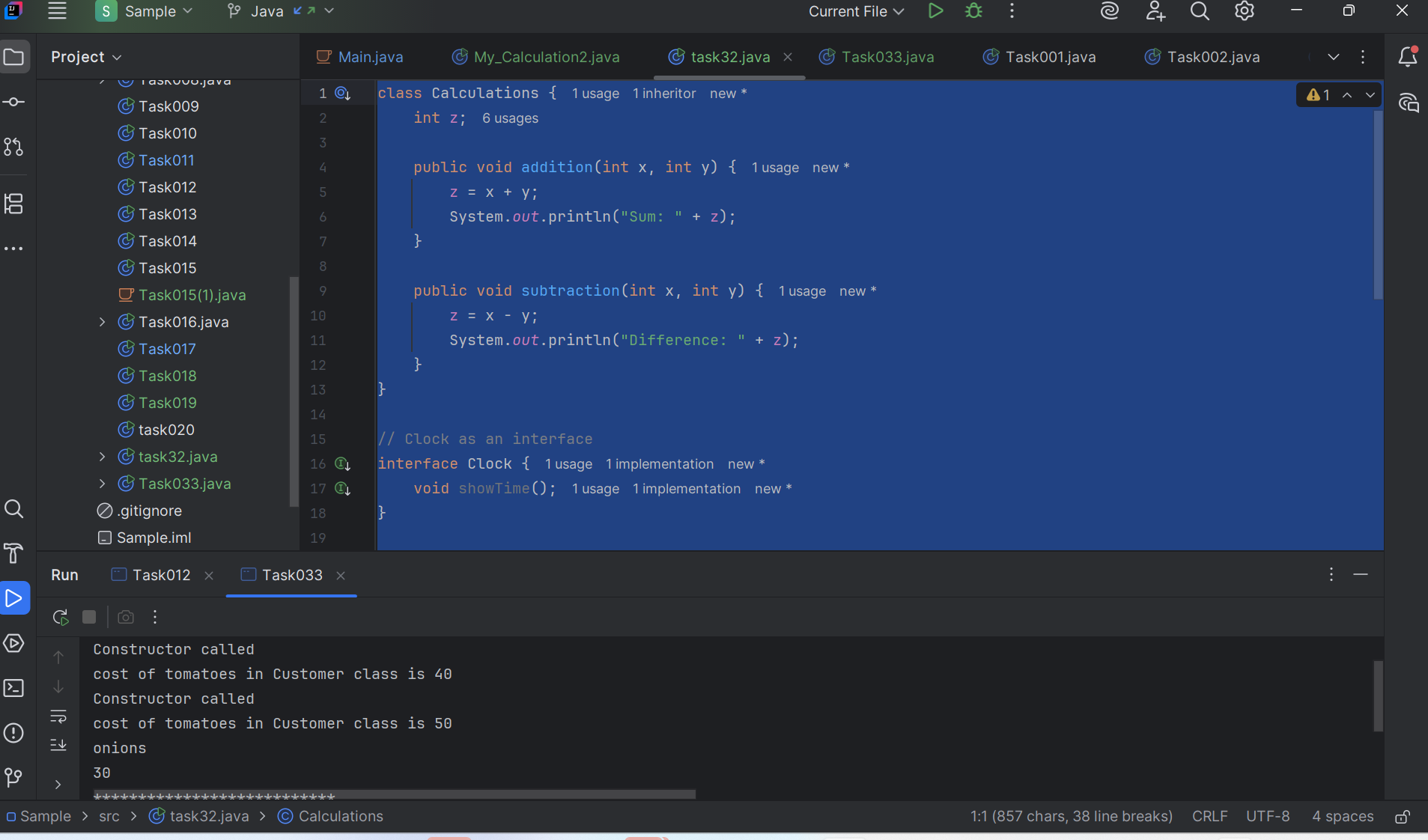
class my\_calculation extends calculation , clock{  // multiple inheritance

// —---------------------------------- ???????????????????????

}

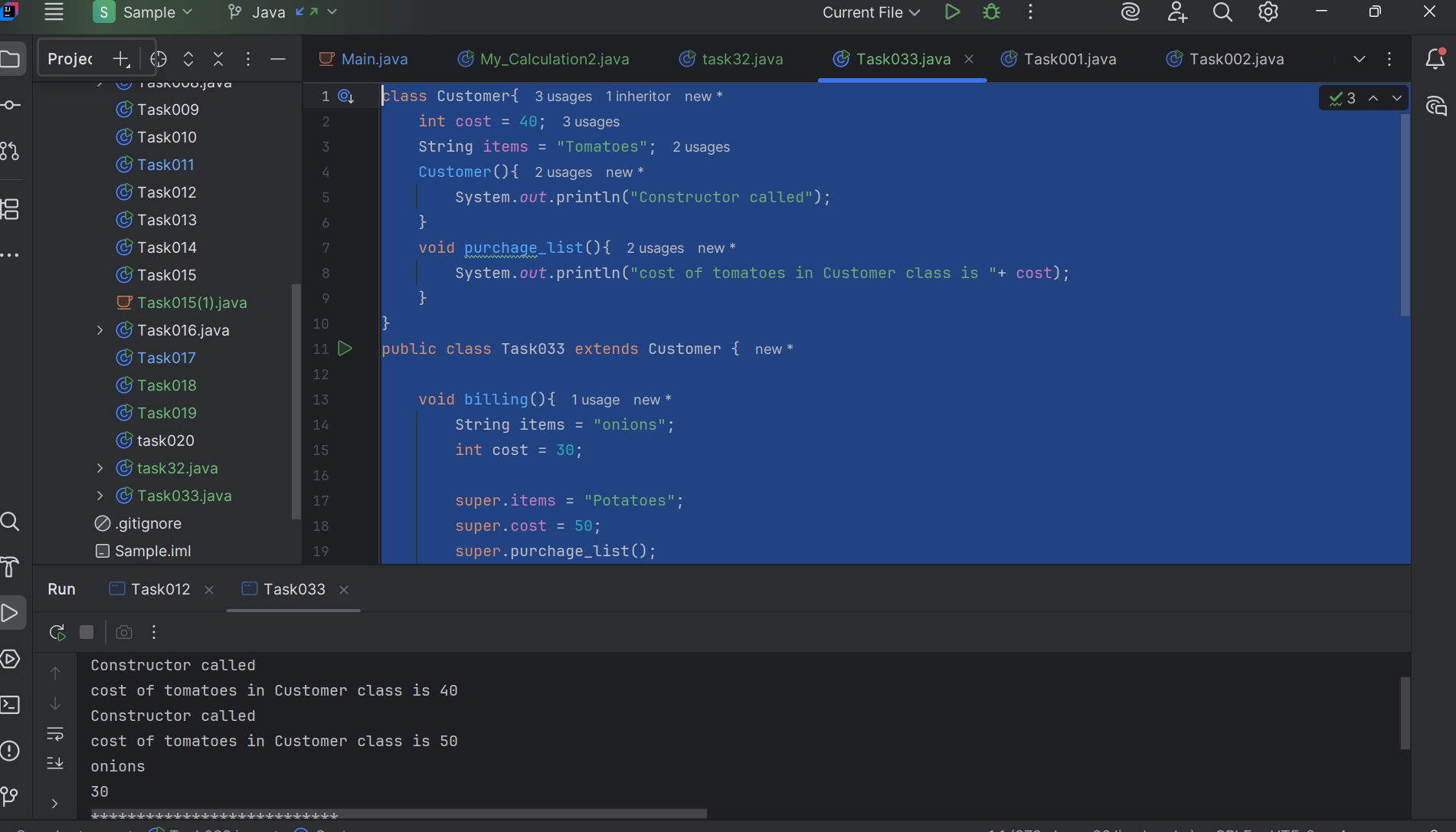
Code :

class Calculations {  
 int z;  
  
 public void addition(int x, int y) {  
 z = x + y;  
 System.*out*.println("Sum: " + z);  
 }  
  
 public void subtraction(int x, int y) {  
 z = x - y;  
 System.*out*.println("Difference: " + z);  
 }  
}  
  
// Clock as an interface  
interface Clock {  
 void showTime();  
}  
  
public class task32 extends Calculations implements Clock {  
  
 public void multiplication(int x, int y) {  
 z = x \* y;  
 System.*out*.println("Product: " + z);  
 }  
  
 // Implementing Clock method  
 public void showTime() {  
 System.*out*.println("Current time: 10:30 AM");  
 }  
  
 public static void main(String[] args) {  
 task32 obj = new task32();  
 obj.addition(10, 5);  
 obj.subtraction(10, 5); // ✅ corrected here  
 obj.multiplication(10, 5);  
 obj.showTime();  
 }  
}



Task 33:

Code :

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);  
 }  
}  
public class Task033 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();  
  
 }  
  
}  
 

Task 034

Void add(int x, int y){

Sout —> x and y values

}

Void add(int x, int y, int z){

Sout —-> x, y, z values

}

psvm(){

add(10,20,30);

add(50,100);

}

=======================================================================

// no.of parameters

public class Task034{

    void add(int x, int y){

        System.out.println(x+ " &&& "+ y);

    }

    void add(int x, int y, int z ){

        System.out.println(x + " $$$ "+ y+ " $$$"+ z);

    }

    public static void main(String[] args){

        Task034 tobj = new Task034();

        add(10,20);

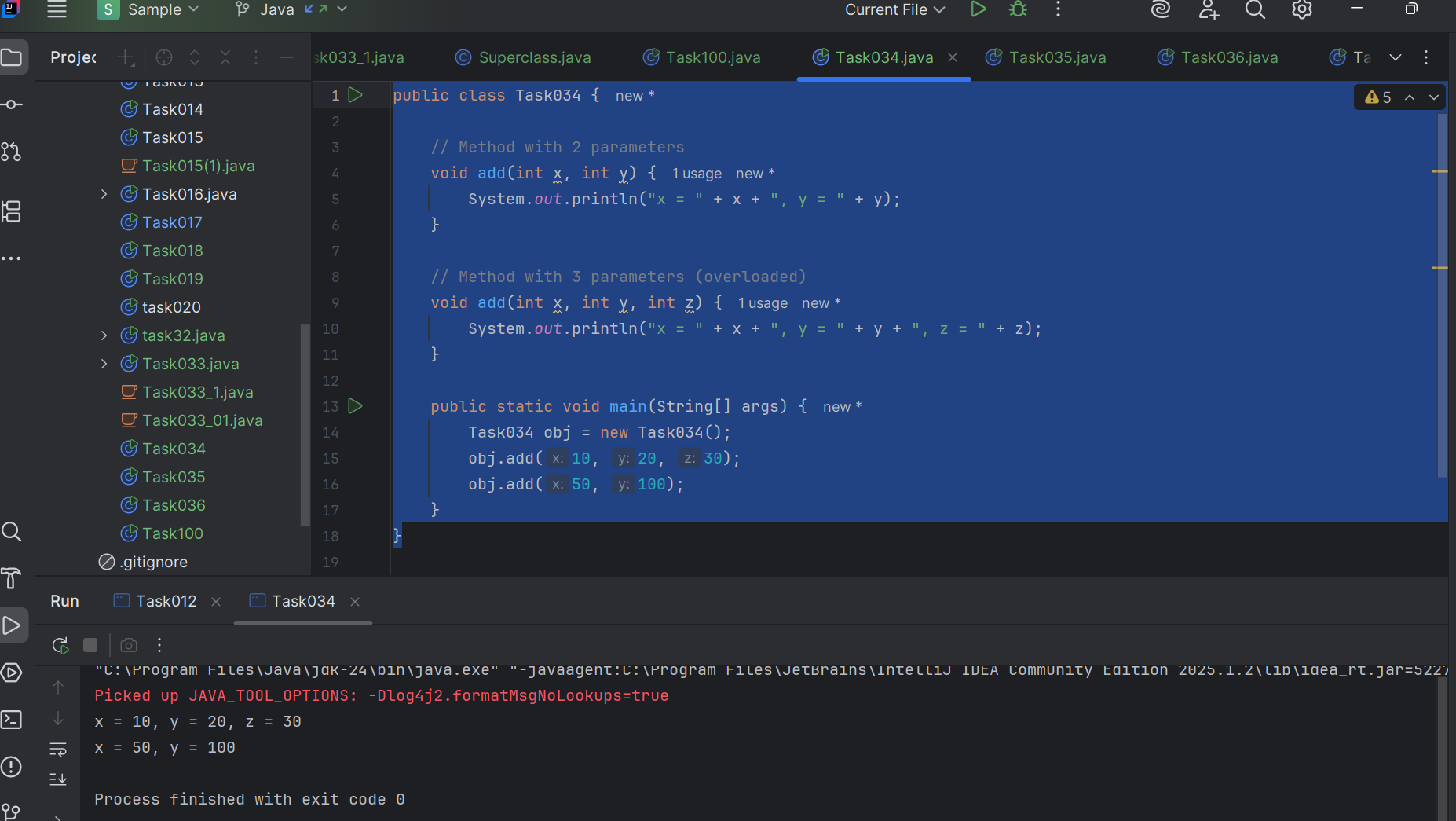
        add(100,200,300);

    }

}

Code :

public class Task034 {  
  
 // Method with 2 parameters  
 void add(int x, int y) {  
 System.*out*.println("x = " + x + ", y = " + y);  
 }  
  
 // Method with 3 parameters (overloaded)  
 void add(int x, int y, int z) {  
 System.*out*.println("x = " + x + ", y = " + y + ", z = " + z);  
 }  
  
 public static void main(String[] args) {  
 Task034 obj = new Task034();  
 obj.add(10, 20, 30);  
 obj.add(50, 100);  
 }  
}



===================================================================================

Type of parameters

Task 035

Void add(char x, char y){

Sout —-> x, y values

}

Void add(int x, int y) {

Sout —> x, y values

}

psvm(){

add(‘d’, ‘a’);

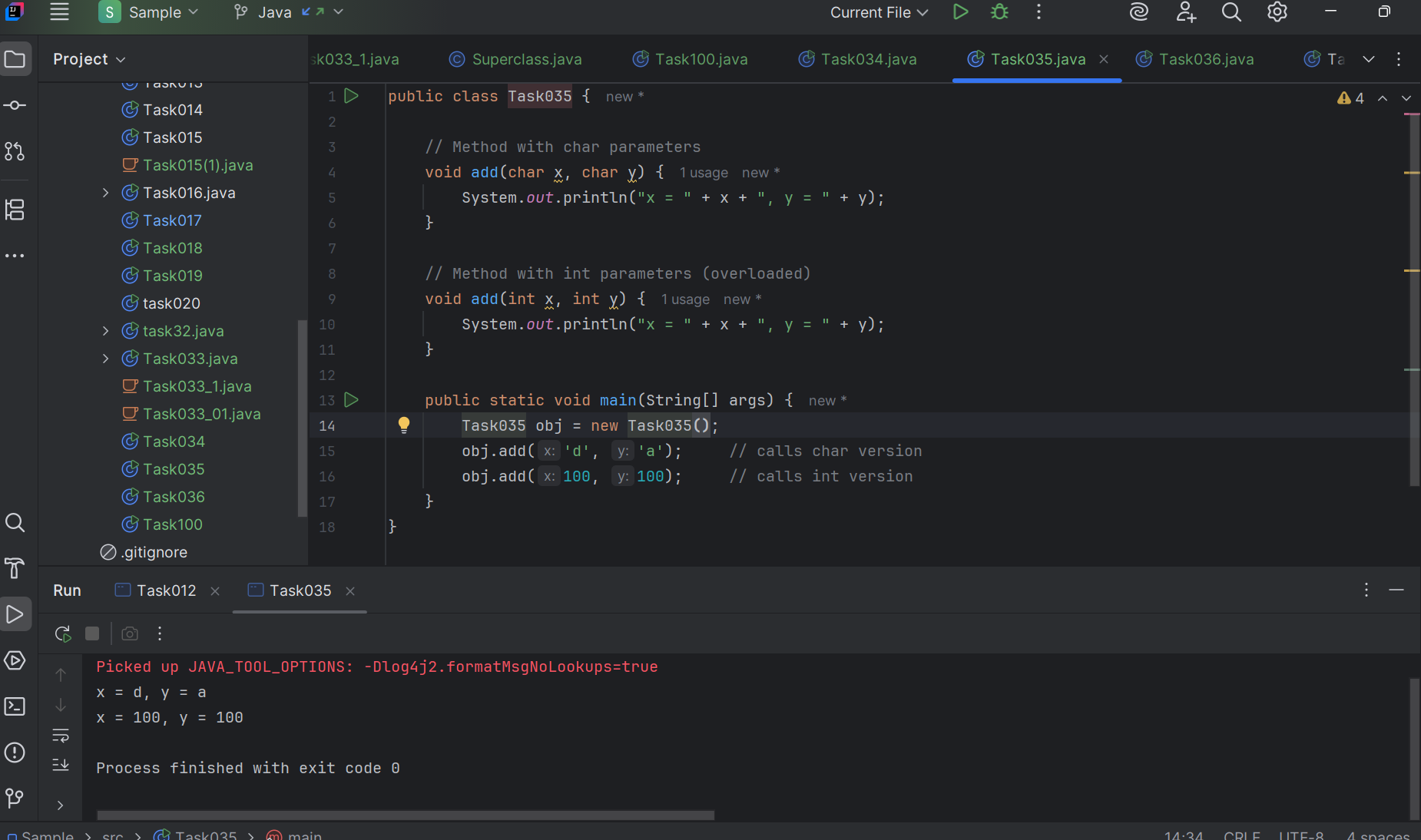
add(100, 100);

}

Sequence of Parameters

CODE :

public class Task035 {  
  
 // Method with char parameters  
 void add(char x, char y) {  
 System.*out*.println("x = " + x + ", y = " + y);  
 }  
  
 // Method with int parameters (overloaded)  
 void add(int x, int y) {  
 System.*out*.println("x = " + x + ", y = " + y);  
 }  
  
 public static void main(String[] args) {  
 Task035 obj = new Task035();  
 obj.add('d', 'a'); // calls char version  
 obj.add(100, 100); // calls int version  
 }  
}



Task 036

Void add(int x, float y){

Sout → x, y values

}

Void add(float x, int y){

Sout  → x, y

}

psvm(){

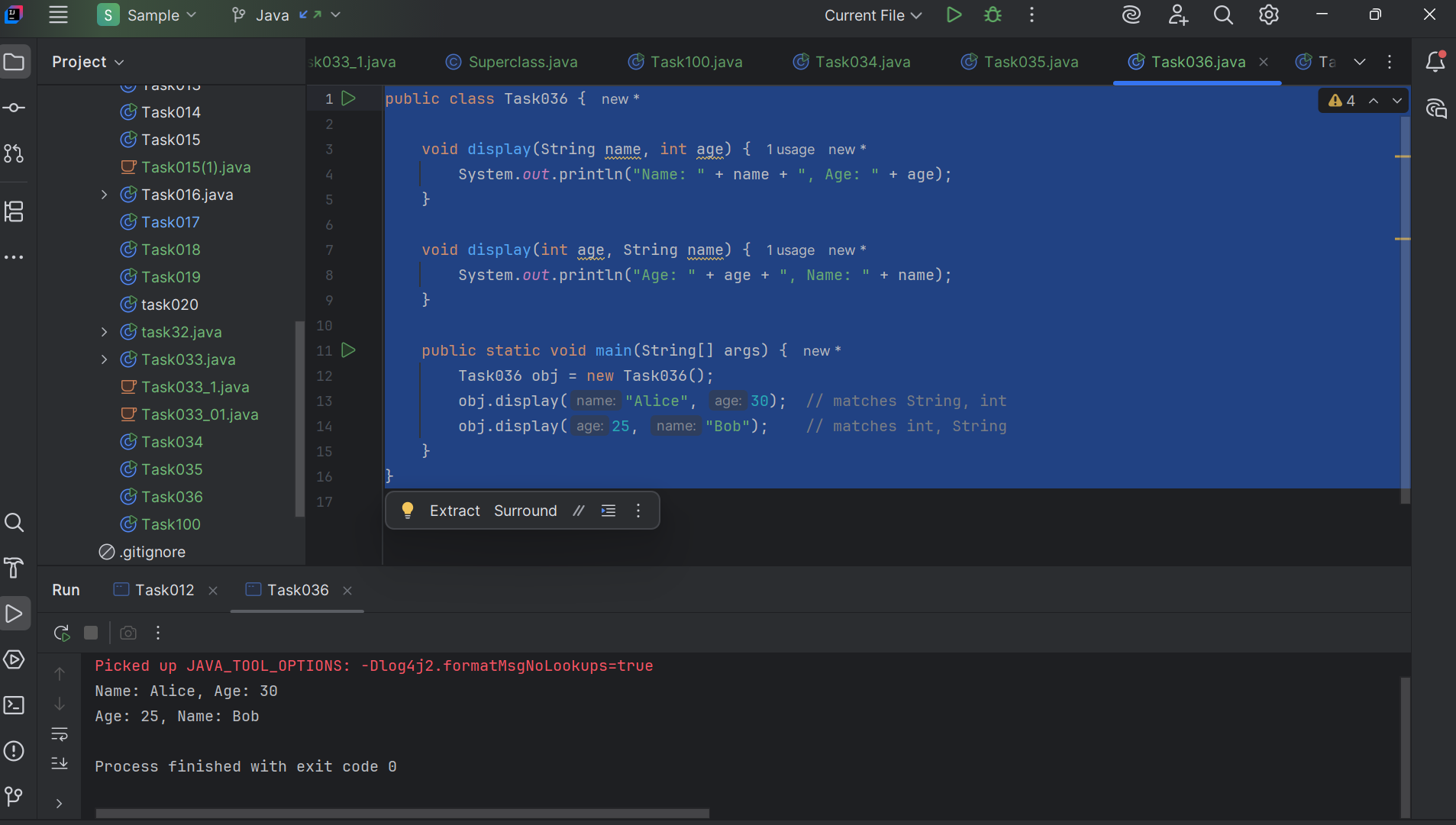
add(10.50f, 60);

add(100, 80.80f)

}

Code:

public class Task036 {  
  
 void display(String name, int age) {  
 System.*out*.println("Name: " + name + ", Age: " + age);  
 }  
  
 void display(int age, String name) {  
 System.*out*.println("Age: " + age + ", Name: " + name);  
 }  
  
 public static void main(String[] args) {  
 Task036 obj = new Task036();  
 obj.display("Alice", 30); // matches String, int  
 obj.display(25, "Bob"); // matches int, String  
 }  
}



========================================================================

Encapsulation

========================================================================

Data hiding – secured data

Access modifiers 👍

private

Protected  – inheritance

Public  – anyone can access

Task 037:

Class Employee{

Private int pwd;

Protected int Salary;

Public int empid:

employee(){ // constructors are methods having same name as class name  (we have in c++)

}

~employee(){// destructors used in c++ but not in java

}

}

Class Hr extends Employee {

super.pwd = 1254; //===============>  ??????

super.Salary = 50000; //==================>  ?

Super.empid = 10001; // ======================>?

}

Class Driver{

psvm(){

Access all variables…????

pwd = 1254; //===============>  ??????

Salary = 50000; //==================>  ?

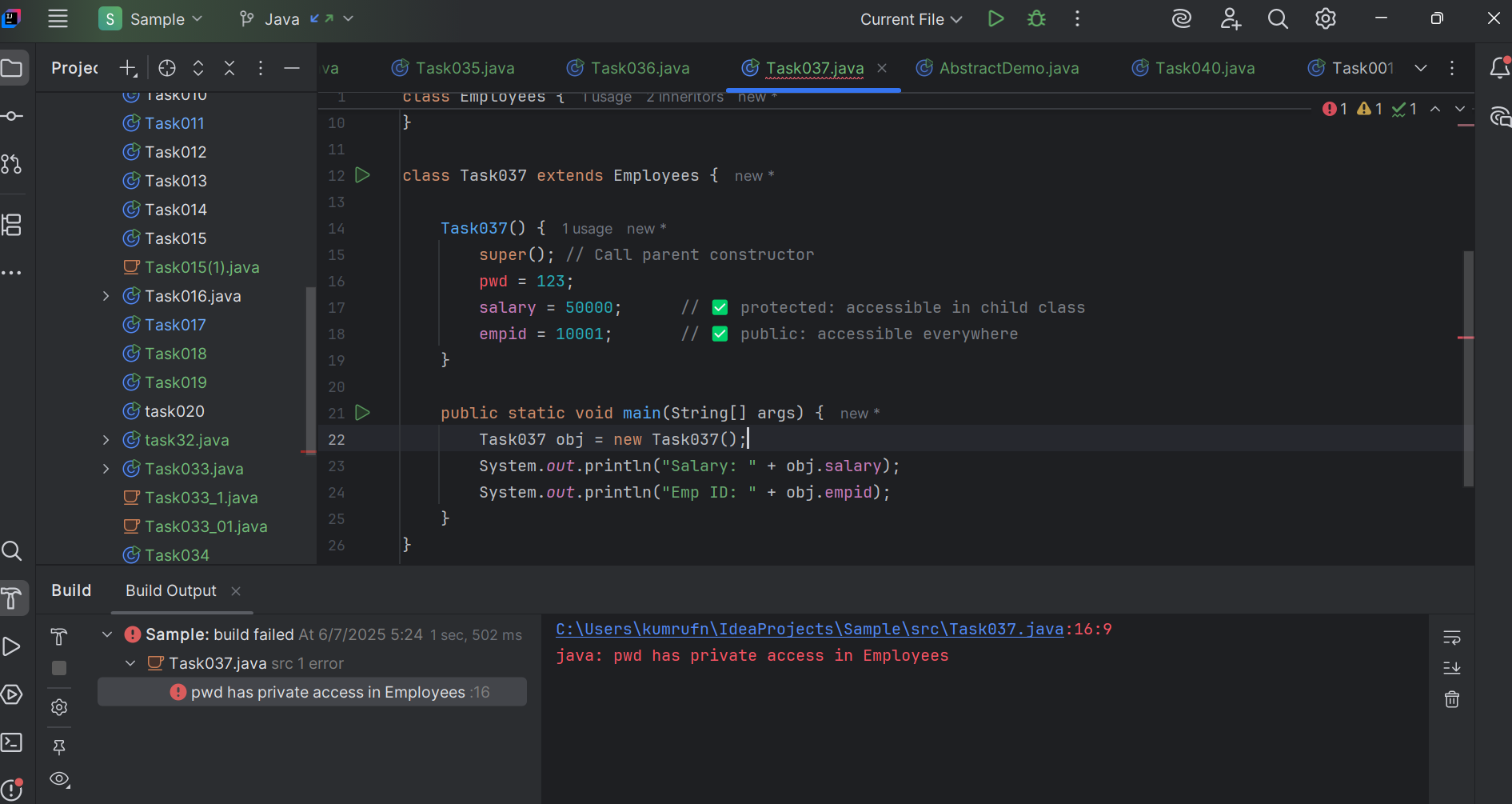
empid = 10001; // ======================>?

}

}

Code:

class Employees {  
 private int pwd;  
 protected int salary;  
 public int empid;  
  
  
 Employees() {  
 System.*out*.println("Employee constructor called");  
 }  
}  
  
class Task037 extends Employees {  
  
 Task037() {  
 super(); // Call parent constructor  
 pwd = 123;  
 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);  
 }  
}



Task 038 & Task 039

Rewrite the above code to give the output without errors.

/\* File name : AbstractDemo.java \*/

Public class AbstractDemo {

   public static void main(String [] args) {

      /\* Following is not allowed and would raise error \*/

      Employee e = new Employee("George W.", "Houston, TX", 43);

      System.out.println("\n Call mailCheck using Employee reference--");

      e.mailCheck();

   }

}

abstract class Employee {

   private String name;

   private String address;

   private int number;

   public Employee(String name, String address, int number) {

      System.out.println("Constructing an Employee");

      this.name = name;

      this.address = address;

      this.number = number;

   }

   public double computePay() {

     System.out.println("Inside Employee computePay");

     return 0.0;

   }

   public void mailCheck() {

      System.out.println("Mailing a check to " + this.name + " " + this.address);

   }

   public String toString() {

      return name + " " + address + " " + number;

   }

   public String getName() {

      return name;

   }

   public String getAddress() {

      return address;

   }

   public void setAddress(String newAddress) {

      address = newAddress;

   }

   public int getNumber() {

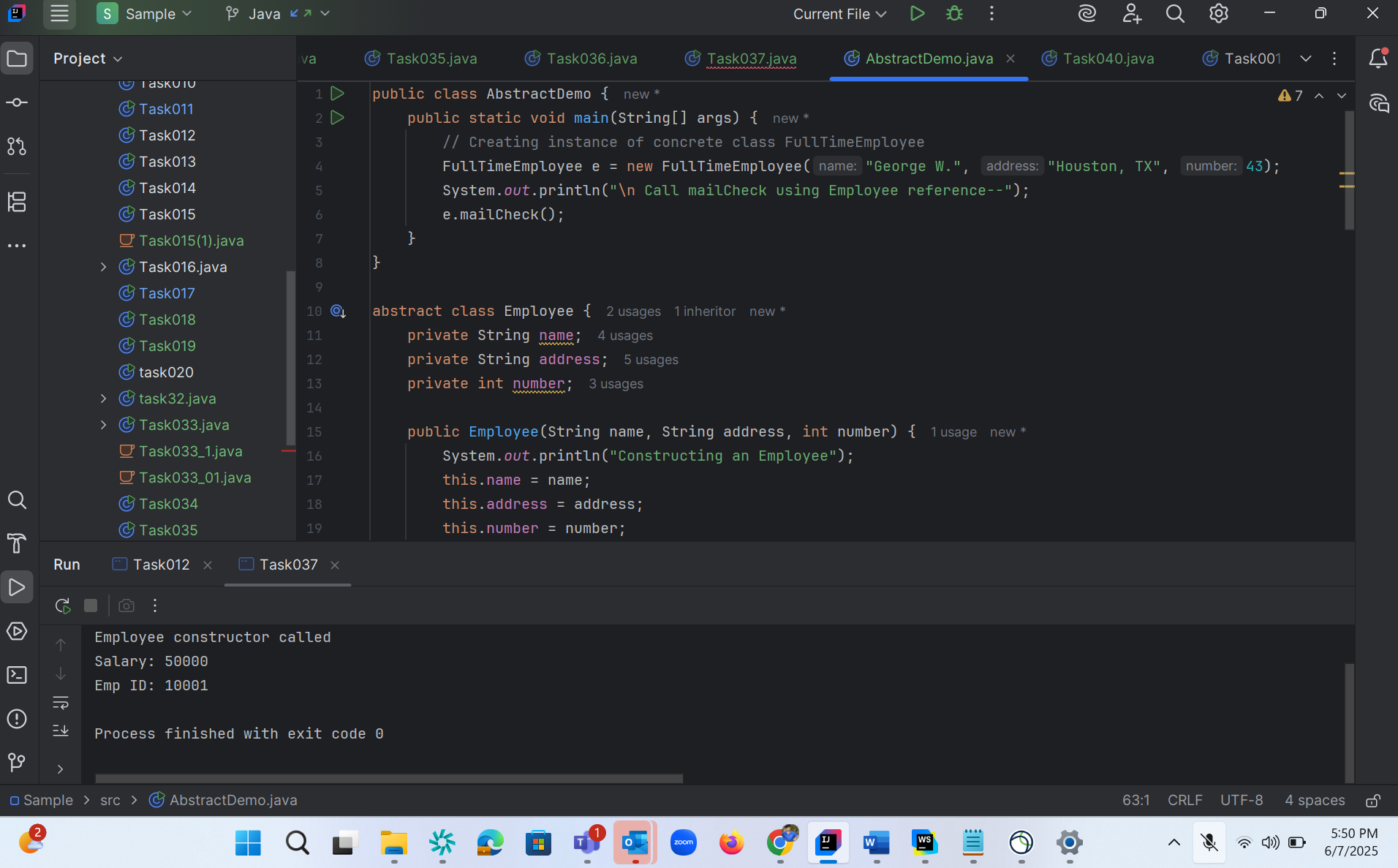
      return number;

   }

}

Code:

public class AbstractDemo {  
 public static void main(String[] args) {  
 // Creating instance of concrete class FullTimeEmployee  
 FullTimeEmployee e = new FullTimeEmployee("George W.", "Houston, TX", 43);  
 System.*out*.println("\n Call mailCheck using Employee reference--");  
 e.mailCheck();  
 }  
}  
  
abstract class Employee {  
 private String name;  
 private String address;  
 private int number;  
  
 public Employee(String name, String address, int number) {  
 System.*out*.println("Constructing an Employee");  
 this.name = name;  
 this.address = address;  
 this.number = number;  
 }  
  
 public double computePay() {  
 System.*out*.println("Inside Employee computePay");  
 return 0.0;  
 }  
  
 public void mailCheck() {  
 System.*out*.println("Mailing a check to " + this.name + " " + this.address);  
 }  
  
 public String toString() {  
 return name + " " + address + " " + number;  
 }  
  
 public String getName() {  
 return name;  
 }  
  
 public String getAddress() {  
 return address;  
 }  
  
 public void setAddress(String newAddress) {  
 address = newAddress;  
 }  
  
 public int getNumber() {  
 return number;  
 }  
}  
  
// Concrete class that extends Employee  
class FullTimeEmployee extends Employee {  
 public FullTimeEmployee(String name, String address, int number) {  
 super(name, address, number);  
 }  
  
 @Override  
 public double computePay() {  
 return 1000.0; // Just an example value  
 }  
}



Task 040

// 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.");

    }

}

// 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();

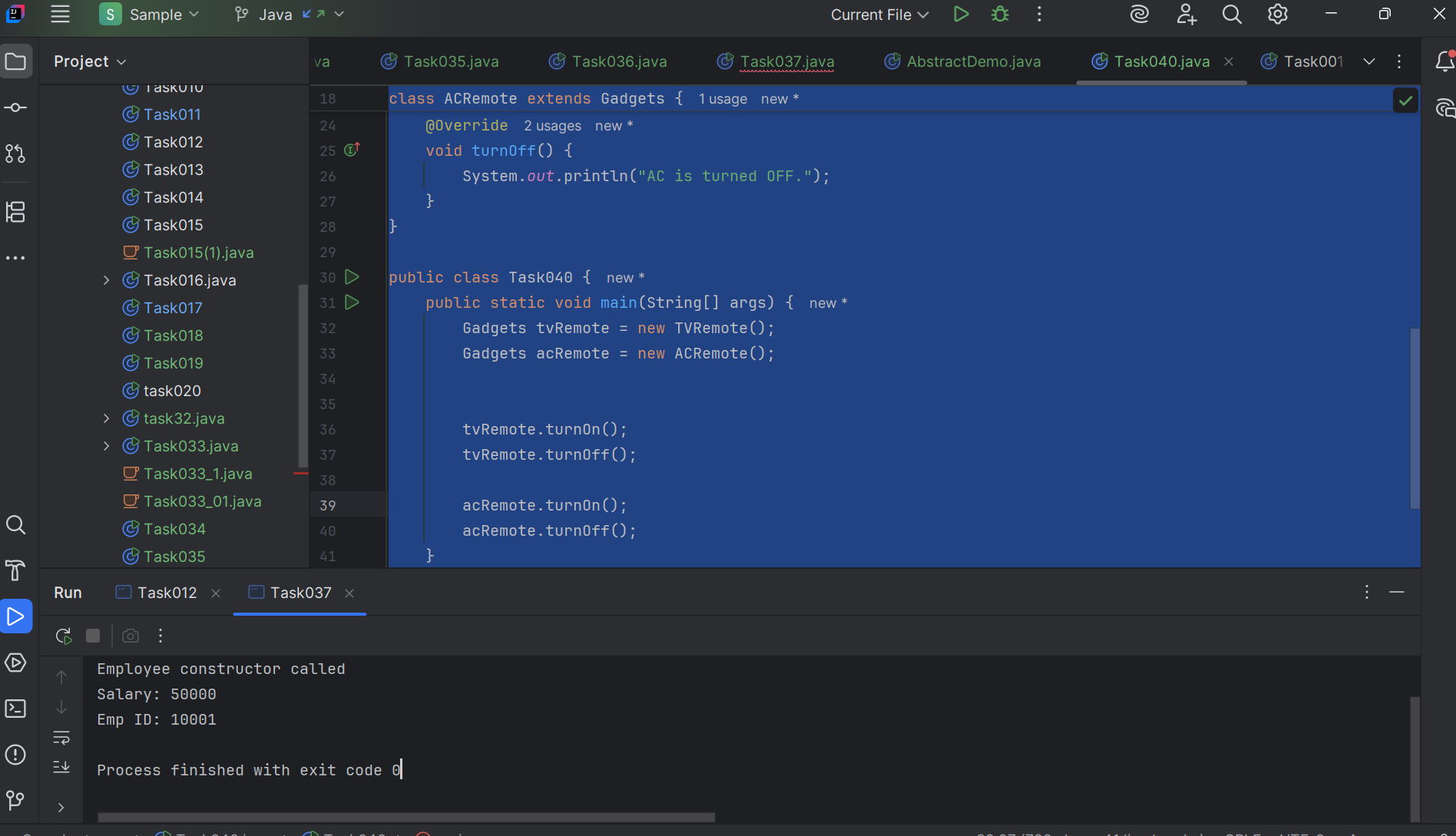
        remote.turnOff();

    }

}

Code :

abstract class Gadgets {  
 abstract void turnOn();  
 abstract void turnOff();  
}  
  
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.");  
 }  
}  
  
public class Task040 {  
 public static void main(String[] args) {  
 Gadgets tvRemote = new TVRemote();  
 Gadgets acRemote = new ACRemote();  
  
  
 tvRemote.turnOn();  
 tvRemote.turnOff();  
  
 acRemote.turnOn();  
 acRemote.turnOff();  
 }  
}



Task 041

import java.io.\*;

// Interface Declared

//Driver Code Ends

interface testInterface {

    // public, static and final

    final int tax = 10;

    // public and abstract

    void display();

}

// Class implementing interface

class TestClass implements testInterface {

    // Implementing the capabilities of

    // Interface

    public void display(){

      System.out.println("Myclass");

    }

}

class Myclass

//Driver Code Starts

{

    public static void main(String[] args)

    {

        TestClass t = new TestClass();

        t.display();

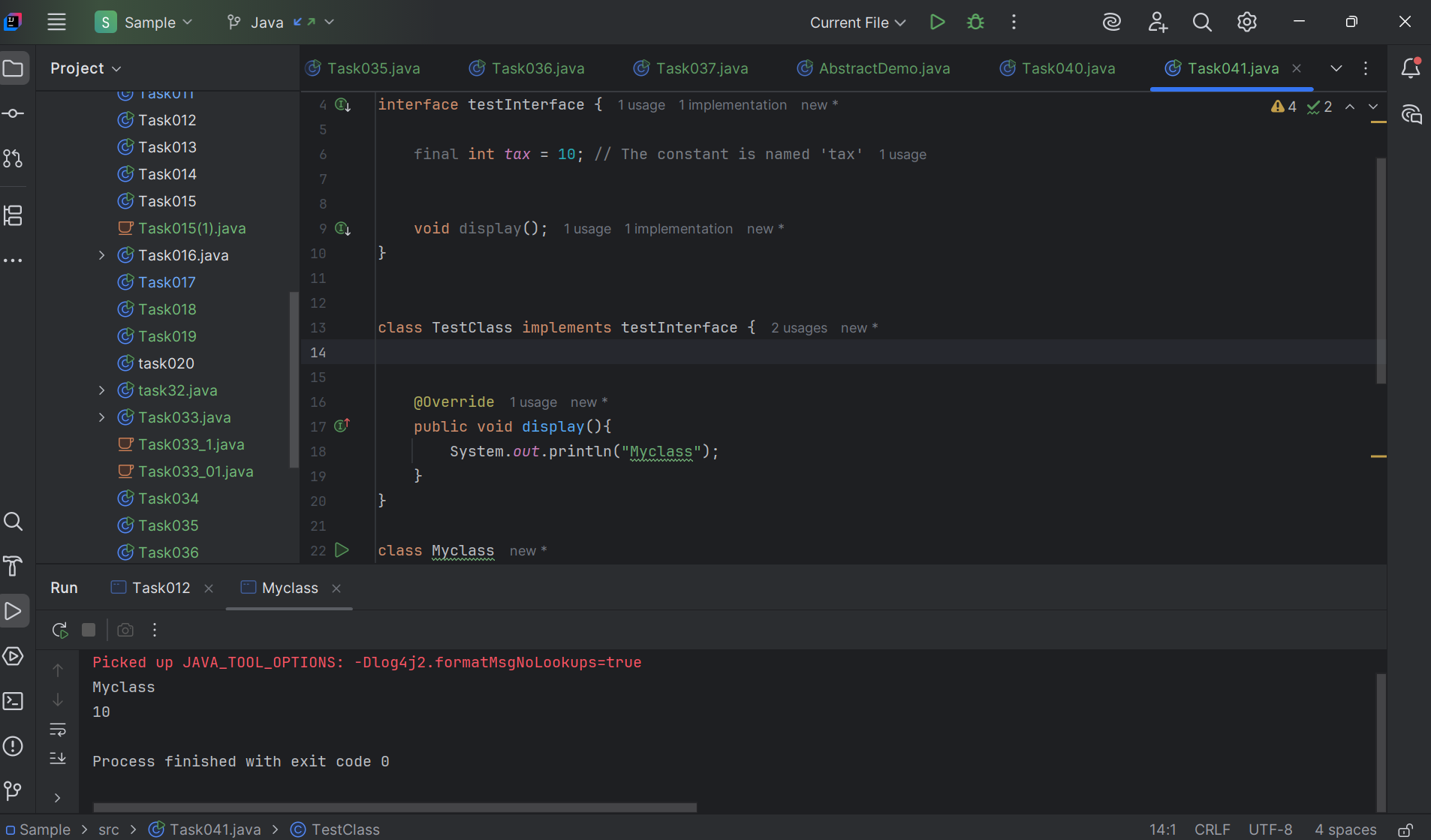
        System.out.println(t.tax);

    }

}

Code :

import java.io.\*;  
  
  
interface testInterface {  
  
 final int *tax* = 10; // The constant is named 'tax'  
  
  
 void display();  
}  
  
  
class TestClass implements testInterface {  
  
  
 @Override  
 public void display(){  
 System.*out*.println("Myclass");  
 }  
}  
  
class Myclass  
{  
 public static void main(String[] args)  
 {  
 TestClass t = new TestClass();  
 t.display();  
  
 System.*out*.println(t.*tax*); // ✅ This will work.  
  
 }  
}



Task 042:

Difference between constant and final..

In Java, the concept of a "constant" is implemented using the final keyword. While final is the tool, the term "constant" refers to the idea of an unchangeable value. Here's a breakdown:

Final Keyword

* **Purpose:**The final keyword in Java is a non-access modifier that indicates that a variable, method, or class cannot be modified after its initial definition.
* **Variables:**When applied to a variable, final makes it a constant. Its value must be assigned at the time of declaration or within the constructor for instance variables. Once assigned, the value cannot be changed.
* **Methods:**A final method cannot be overridden by subclasses, ensuring its behavior remains consistent.
* **Classes:**A final class cannot be subclassed, preventing inheritance.

Constants

* **Concept:**A constant is a variable whose value is fixed and cannot be changed after initialization. Constants are used to represent values that should not be modified during the program's execution.
* **Implementation:**In Java, constants are created using the final keyword. It's common practice to declare constants as public static final, making them accessible from anywhere and ensuring they have a single copy throughout the program.
* **Compile-Time vs. Run-Time:**
  + final variables can be initialized at compile-time or run-time.
  + When final variables are initialized with literal values or constant expressions at compile-time, they behave as true constants.
  + When final variables are initialized at run-time, they become run-time constants, meaning their value is fixed after their first assignment but not known until the program runs.

Key Differences

* **Terminology:**"Constant" is a conceptual idea, while final is the Java keyword to realize that idea.
* **Flexibility:**final variables can be initialized at run-time, allowing for values that are not known at compile time.
* **Scope:**final can be applied to variables, methods, and classes, while "constant" is primarily associated with variables.

In summary, final is the mechanism to create constants in Java. By declaring a variable as final, you ensure its value remains unchanged, promoting code clarity and preventing accidental modifications.