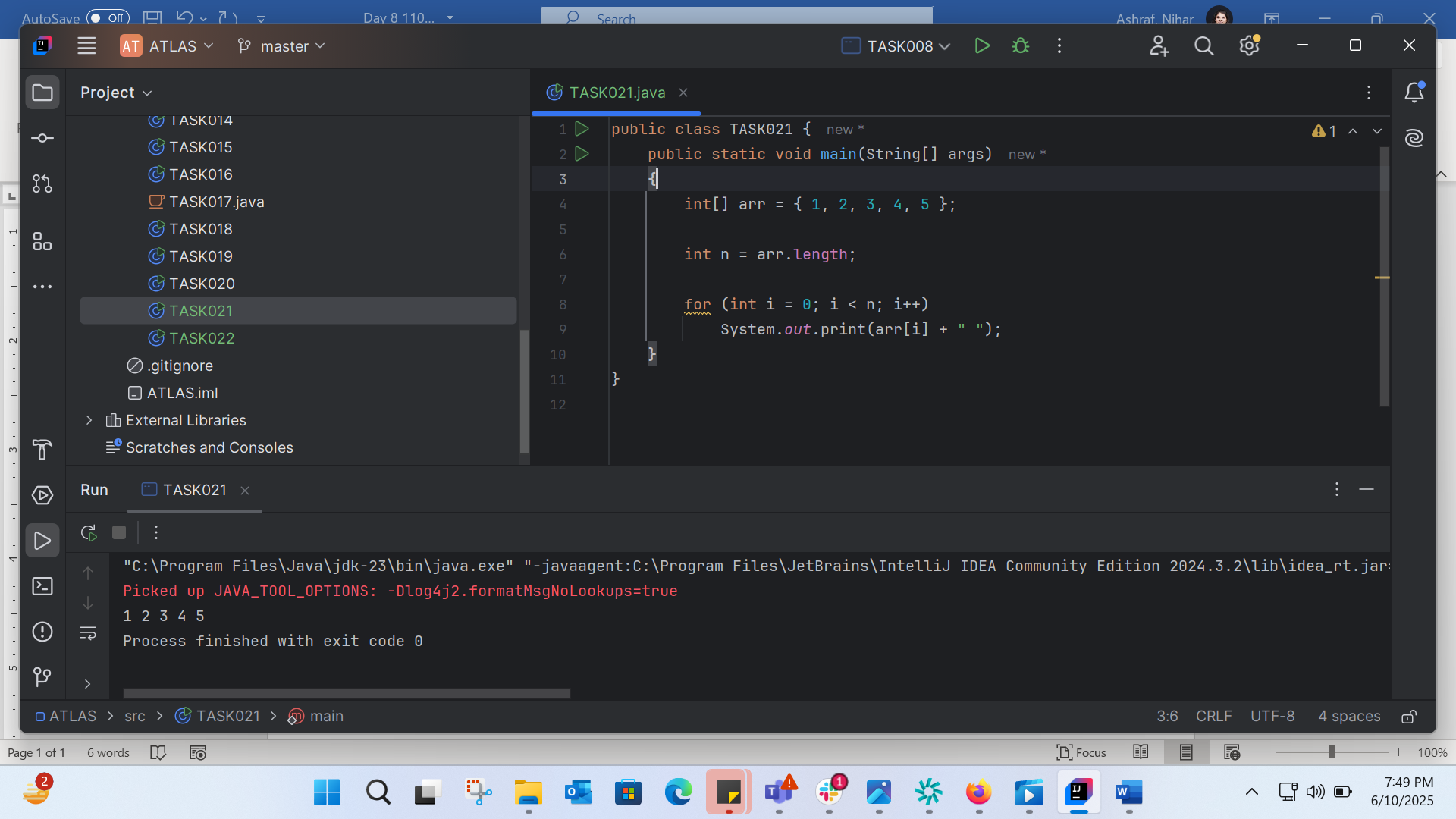
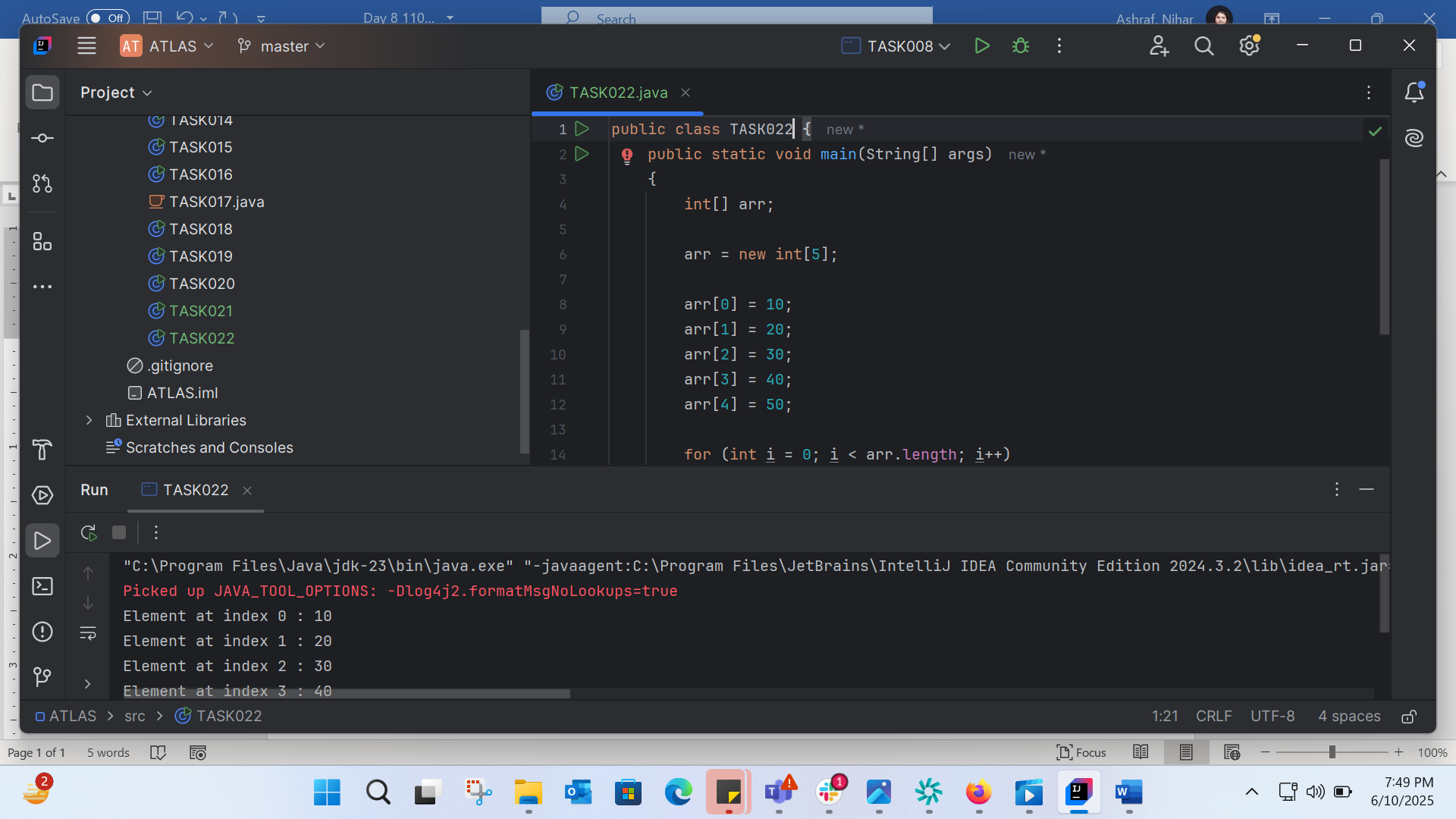
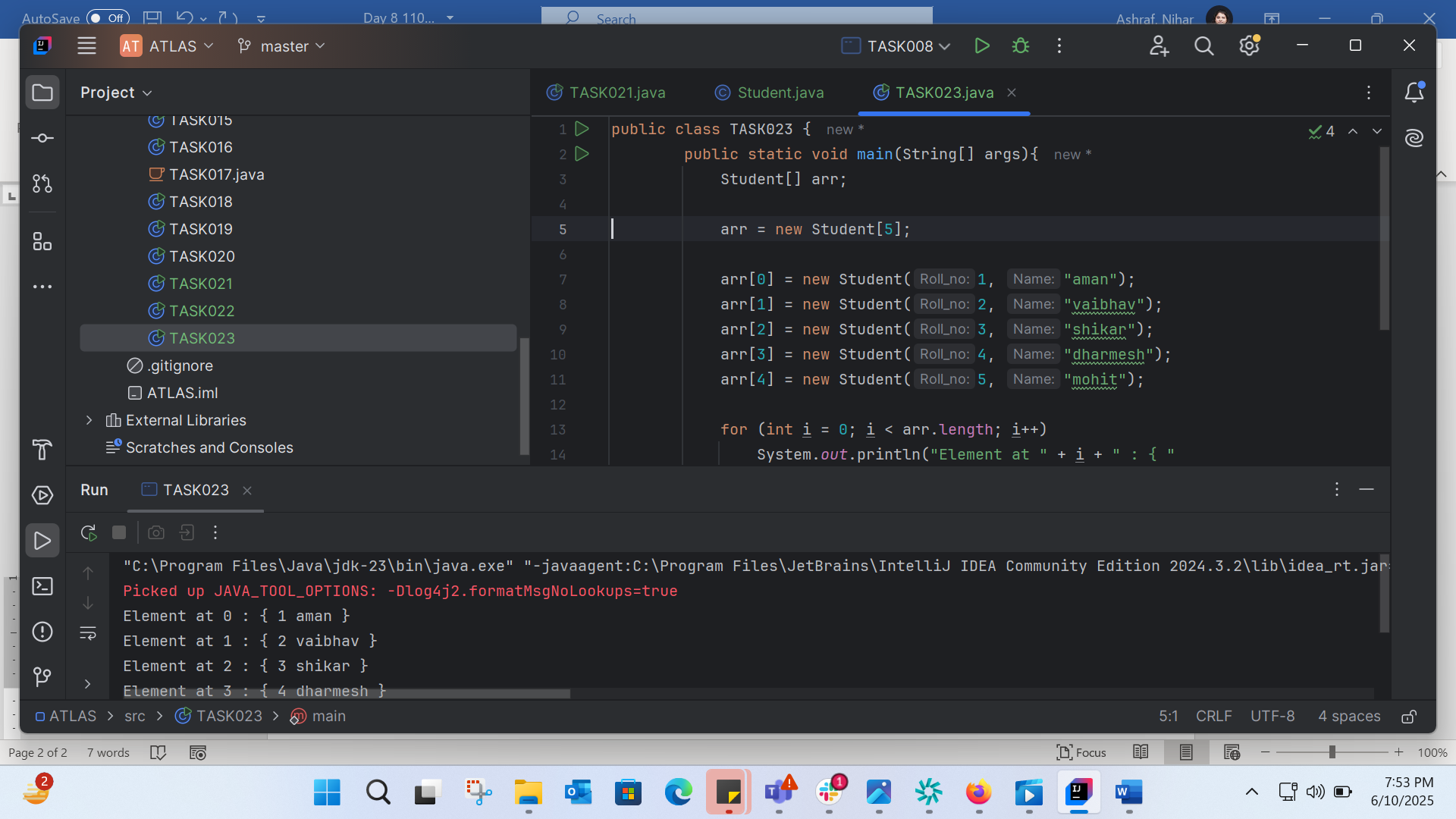
DAY 8

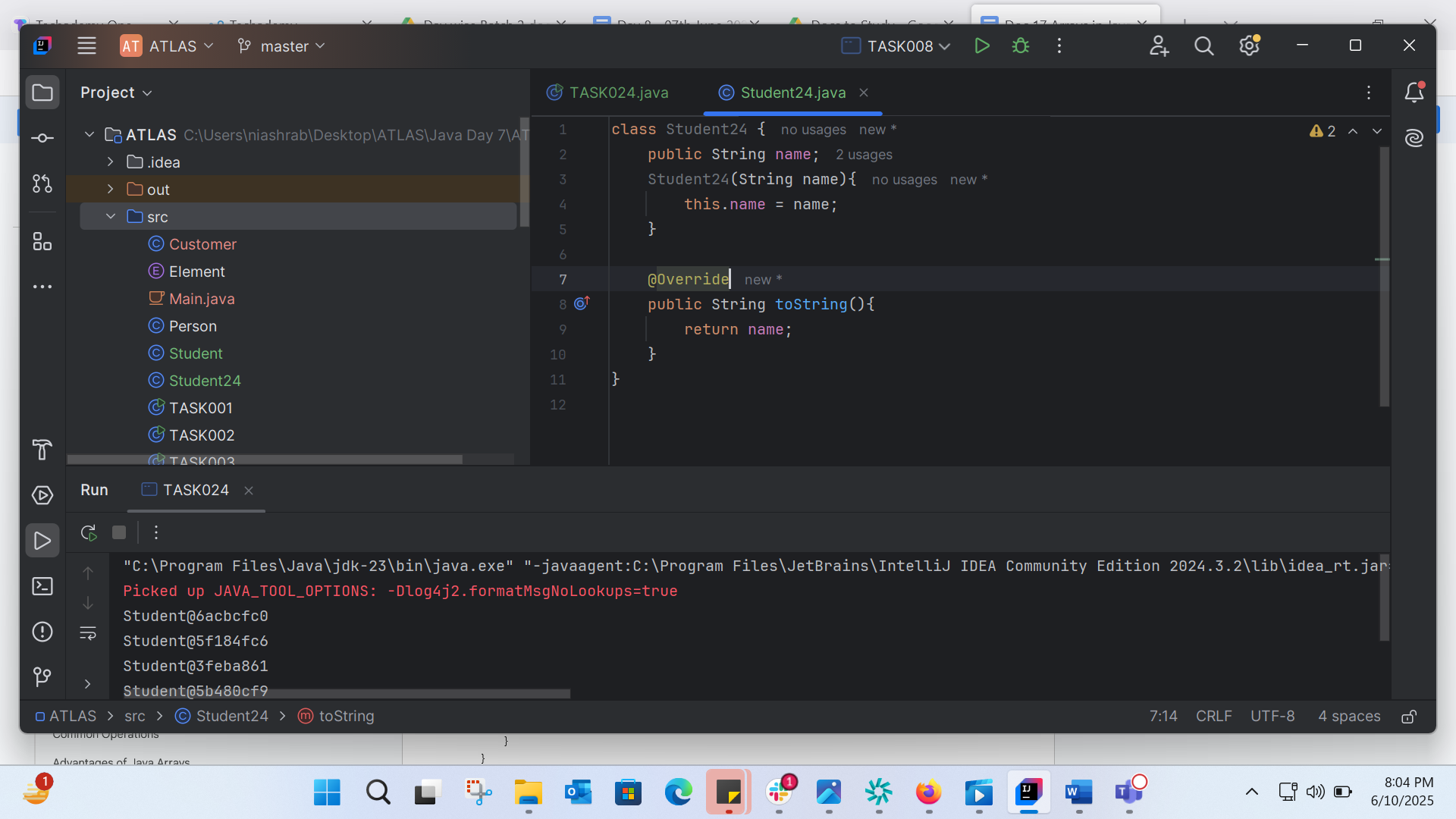
Date: 7/6/2025

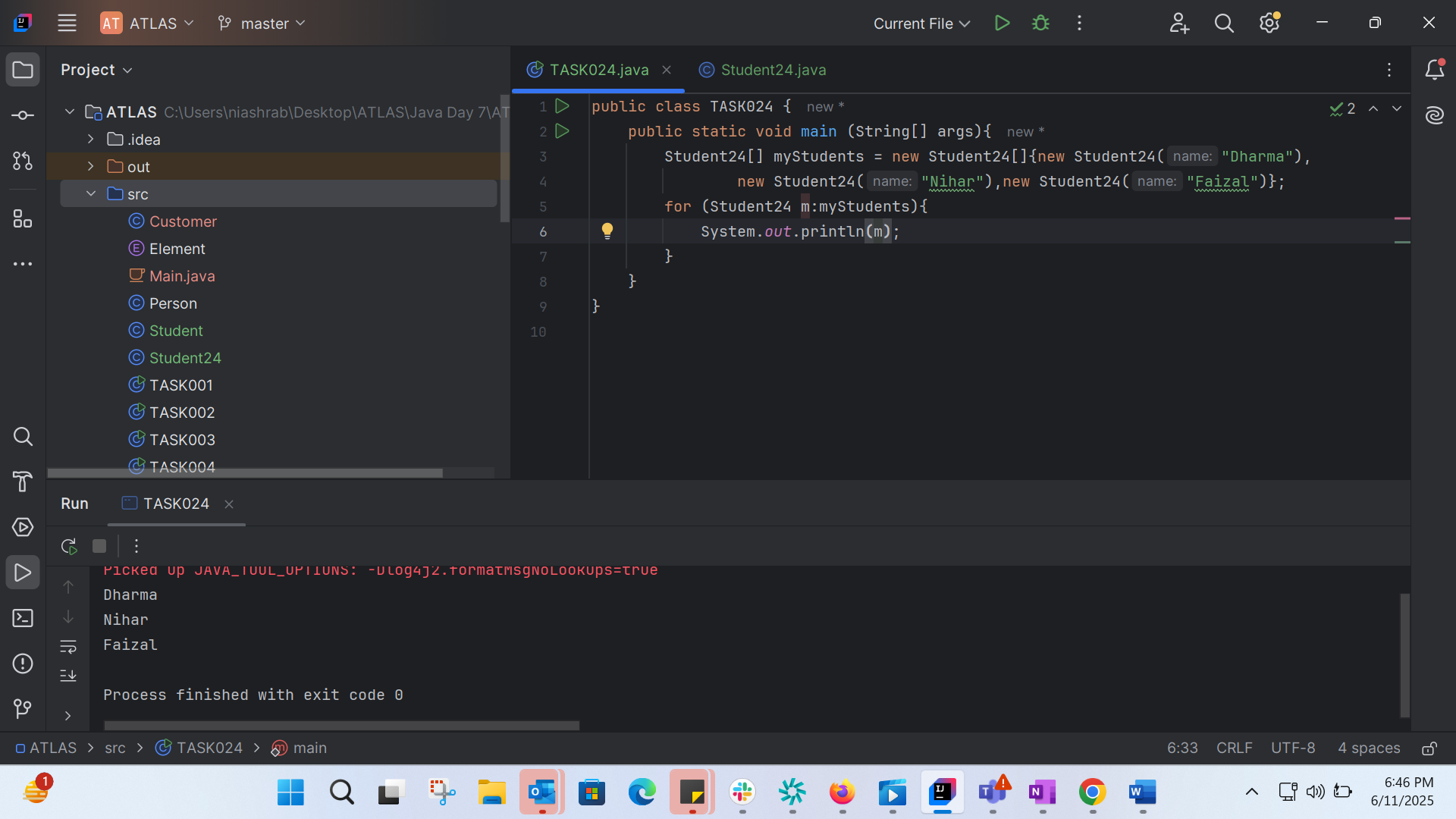
Task021: 

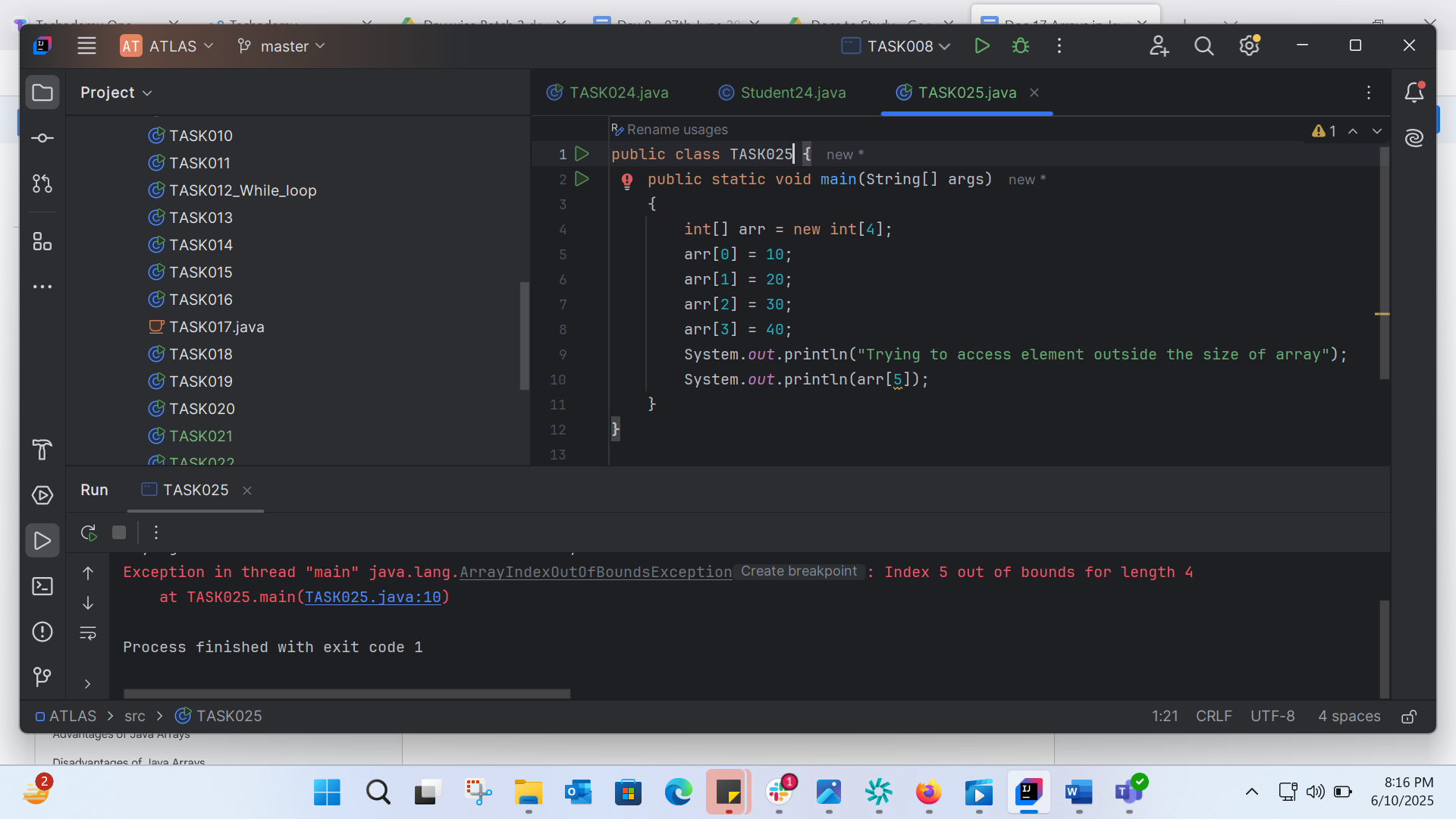
TASK022: 

TASK023: 

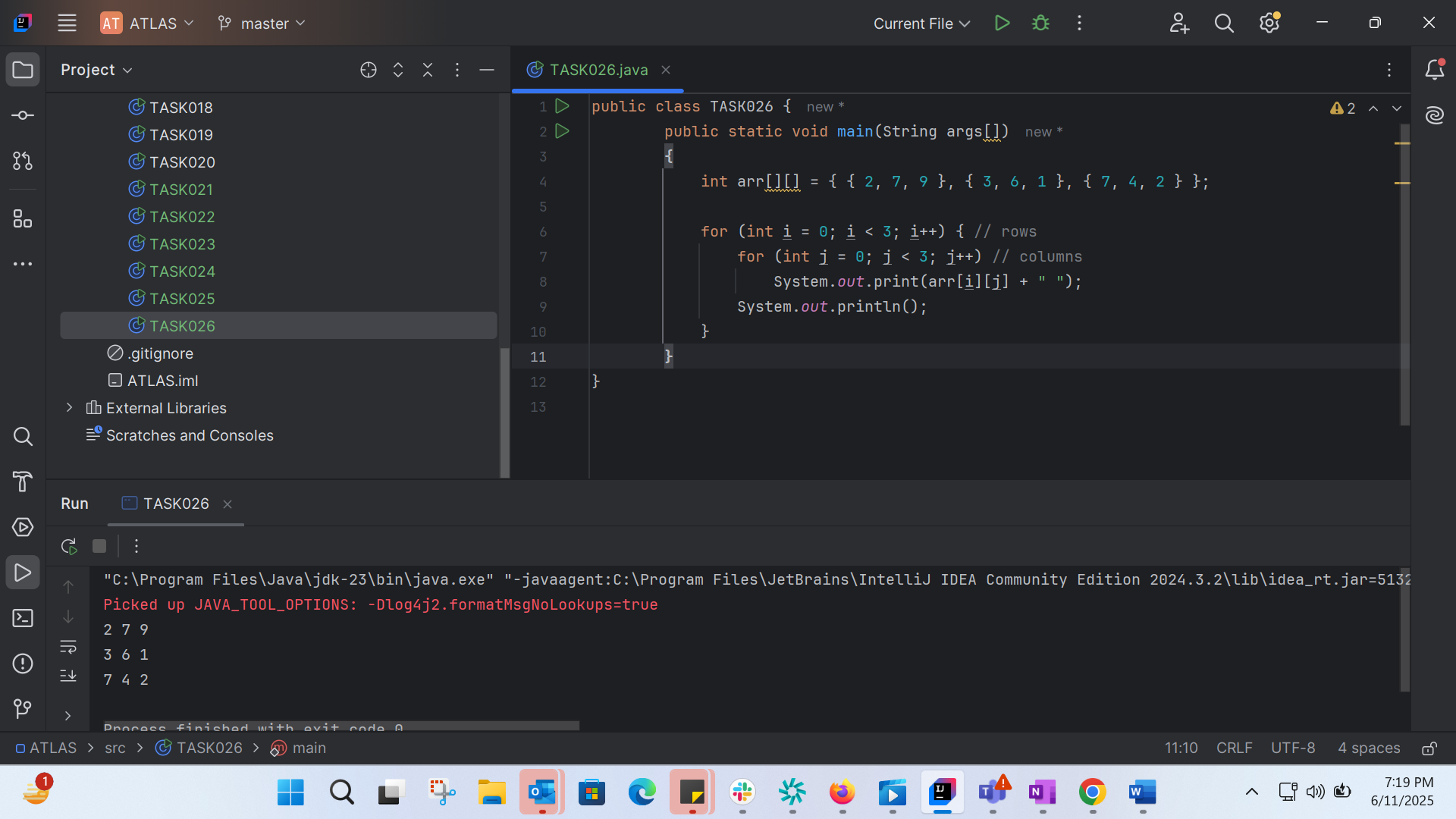
TASK024:

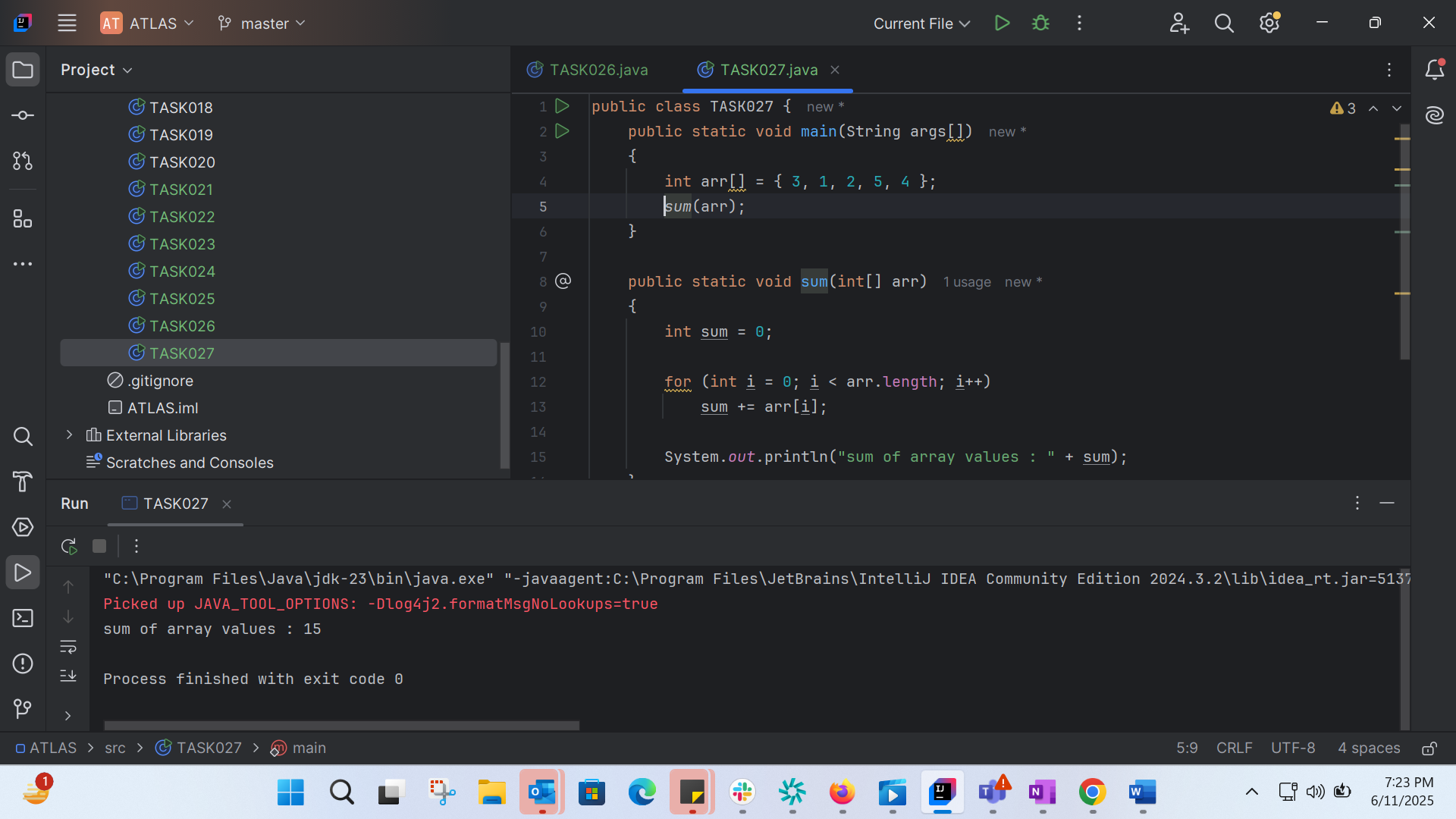
Without Override: 

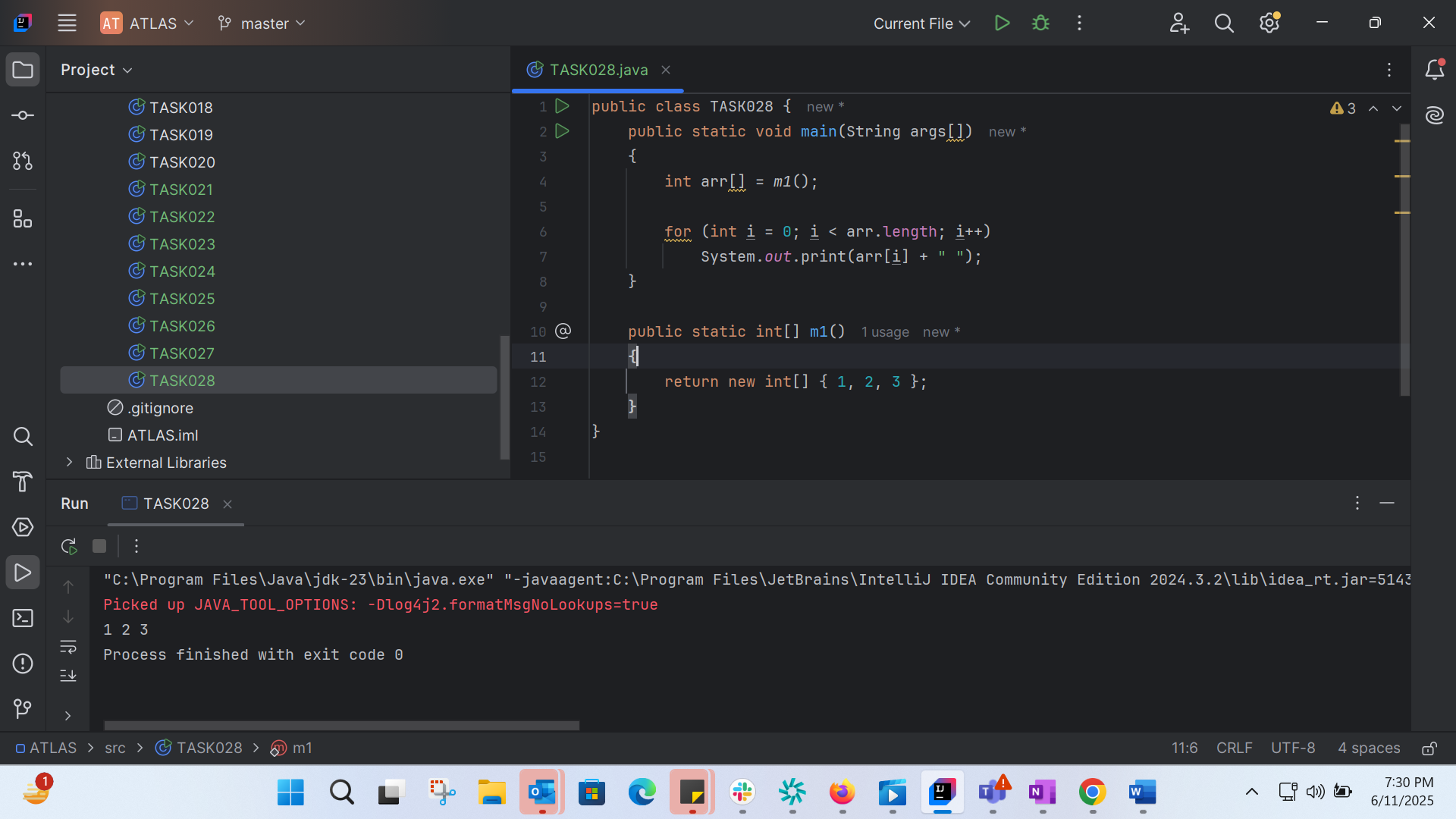
With override: 

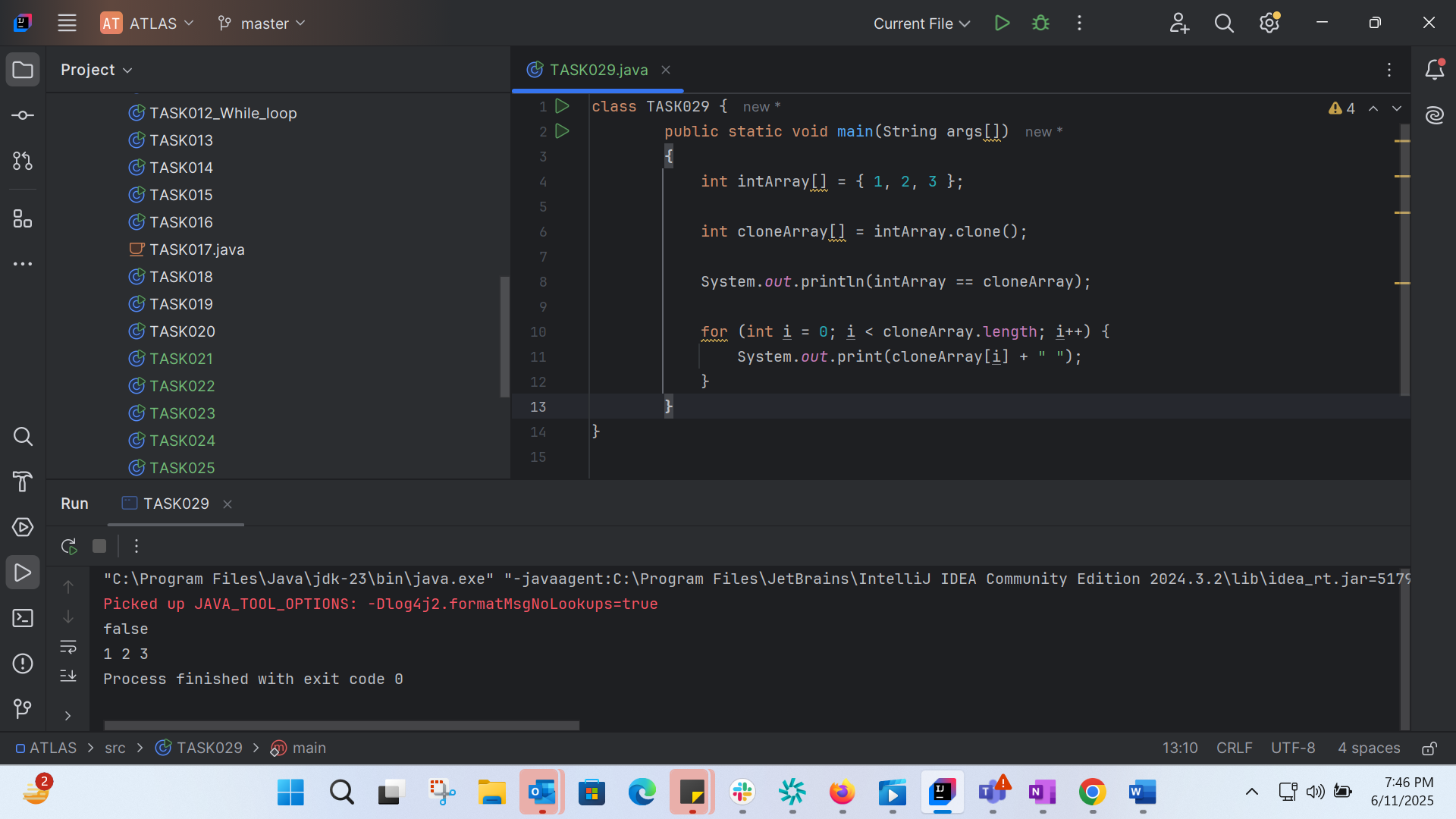
TASK025: 

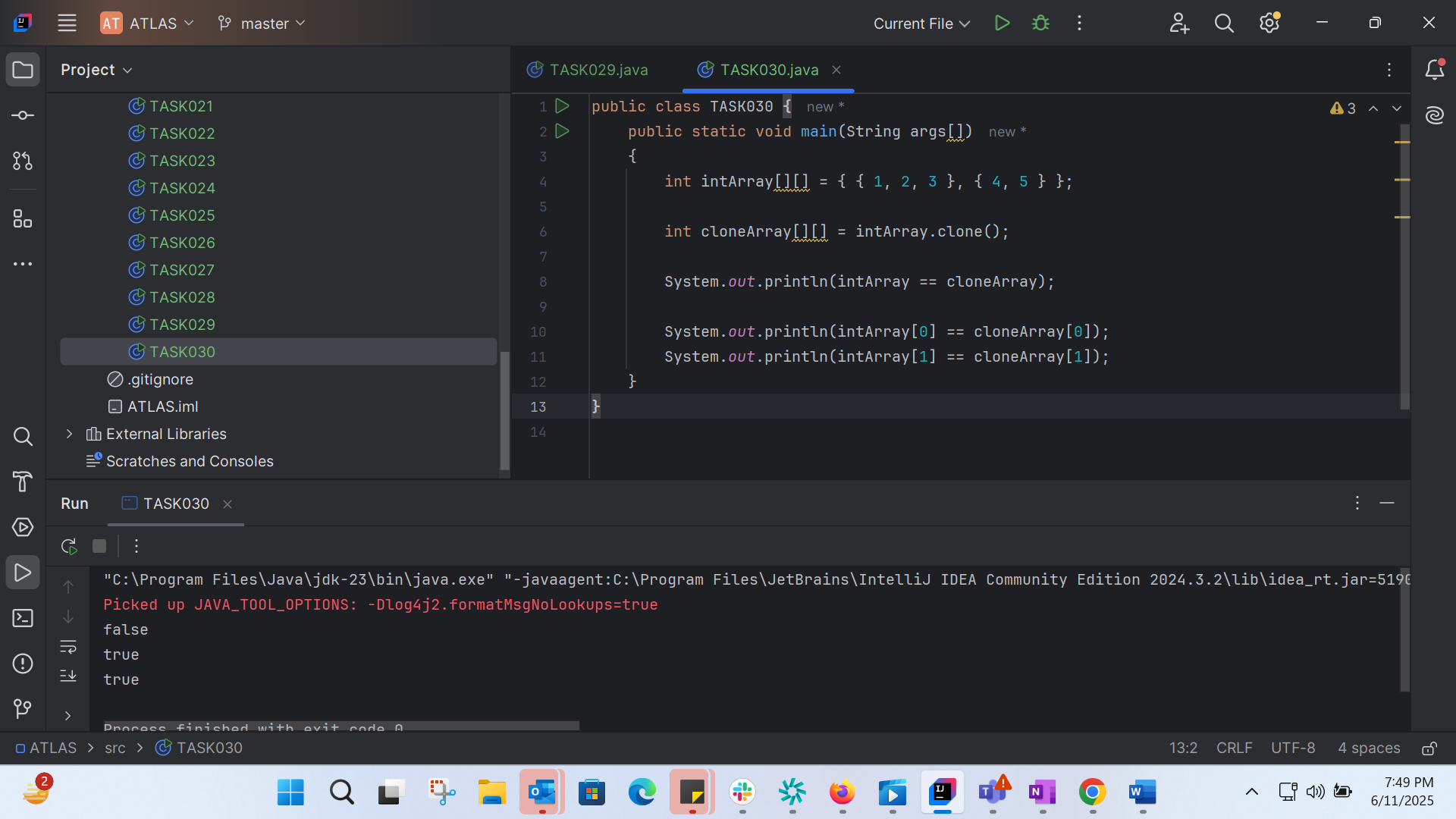
Cant bcz array size is 4.

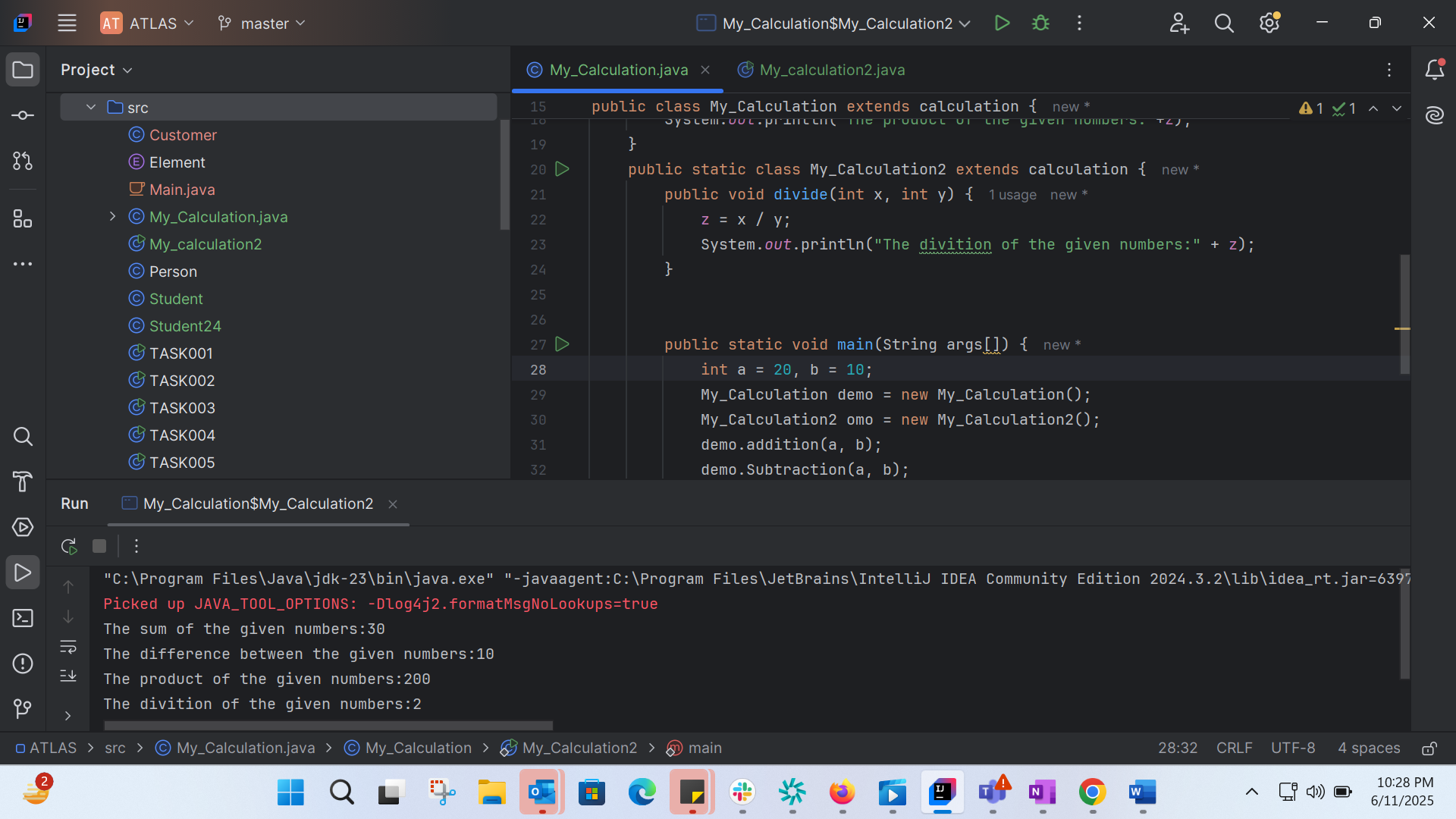
TASK026: 

TASK27: 

TASK28: 

TASK29: 

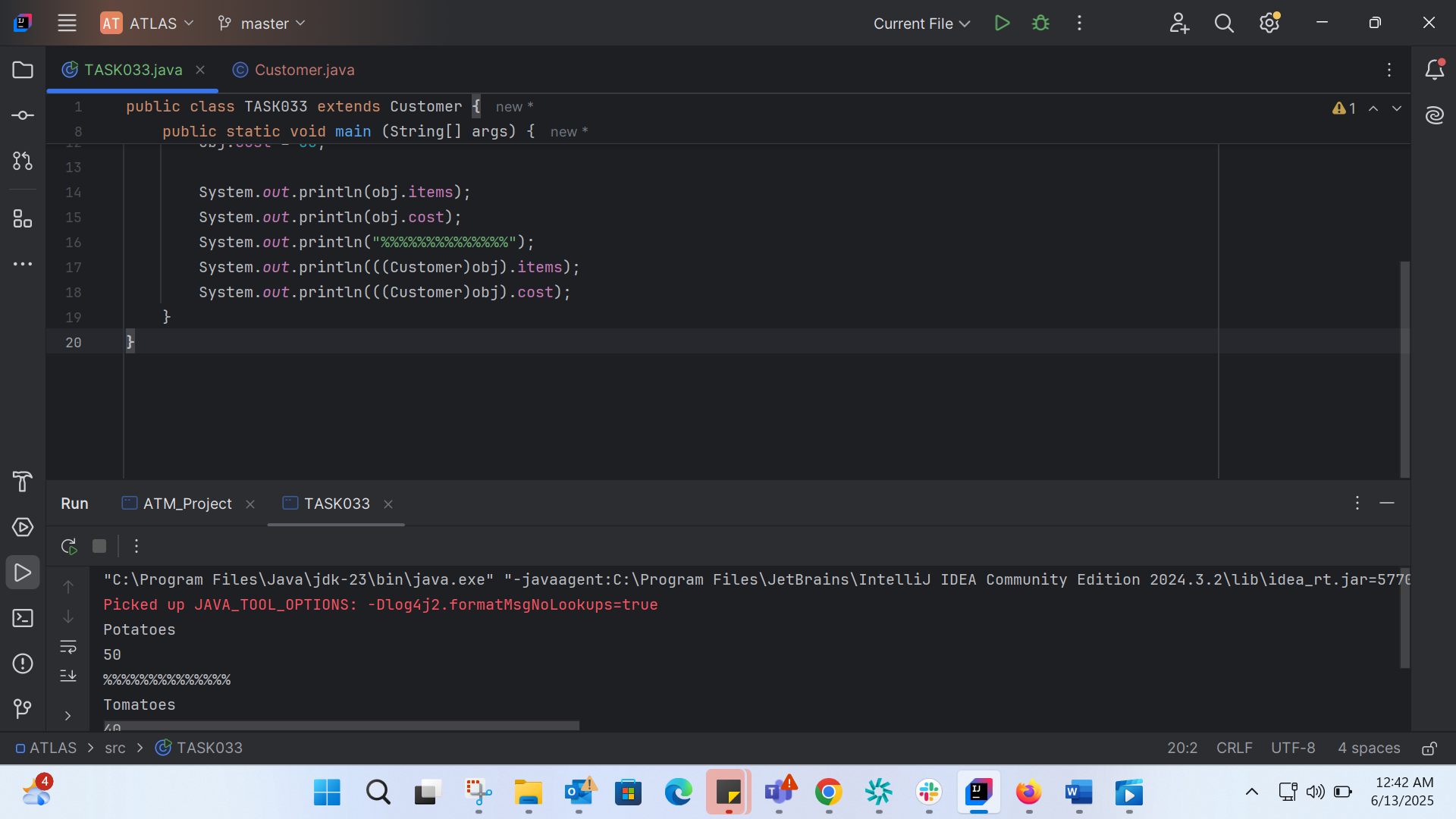
Task30: 

TASK31: 

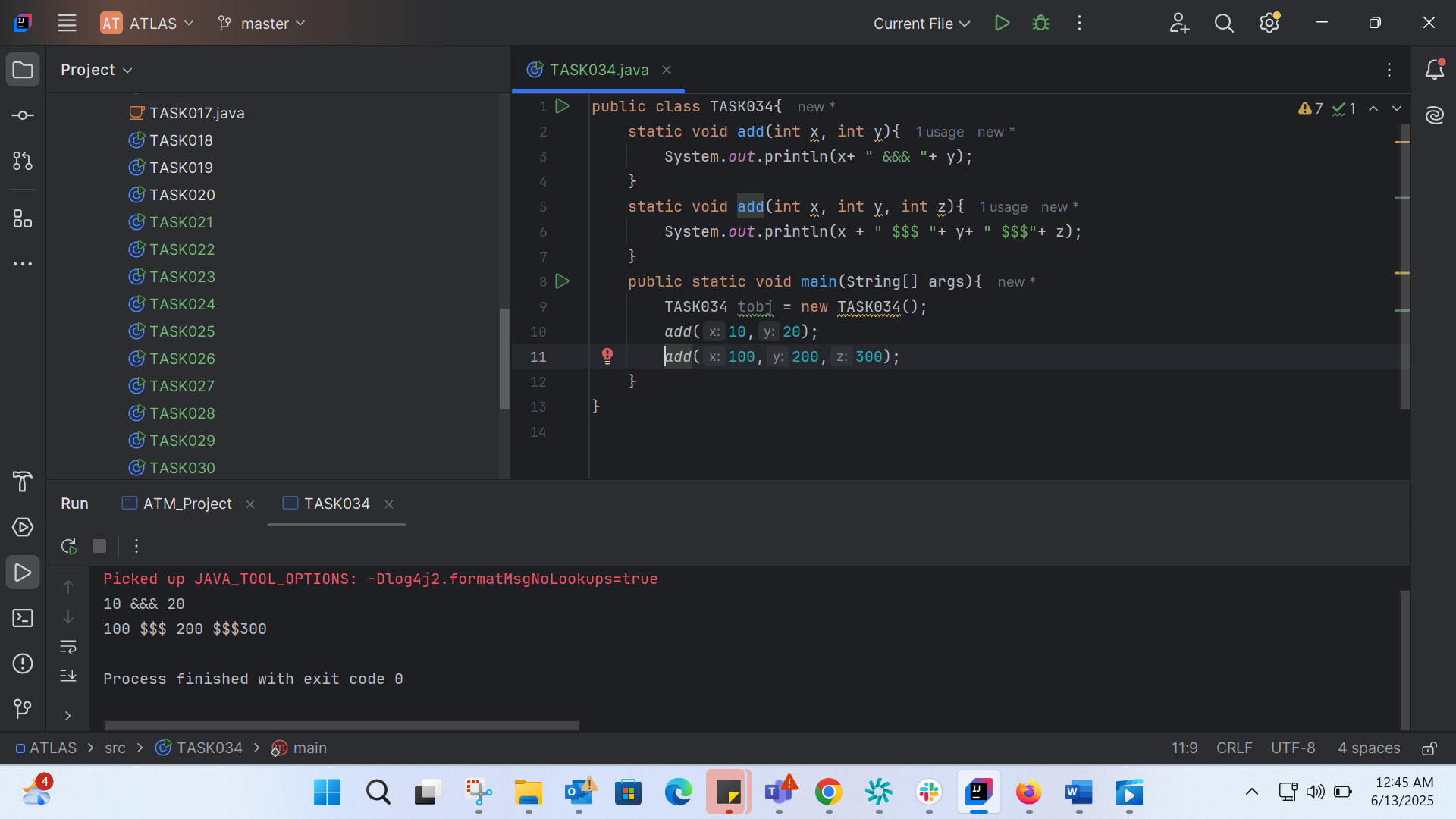
TASK 32:

Multiple inheretance is not supported in Java

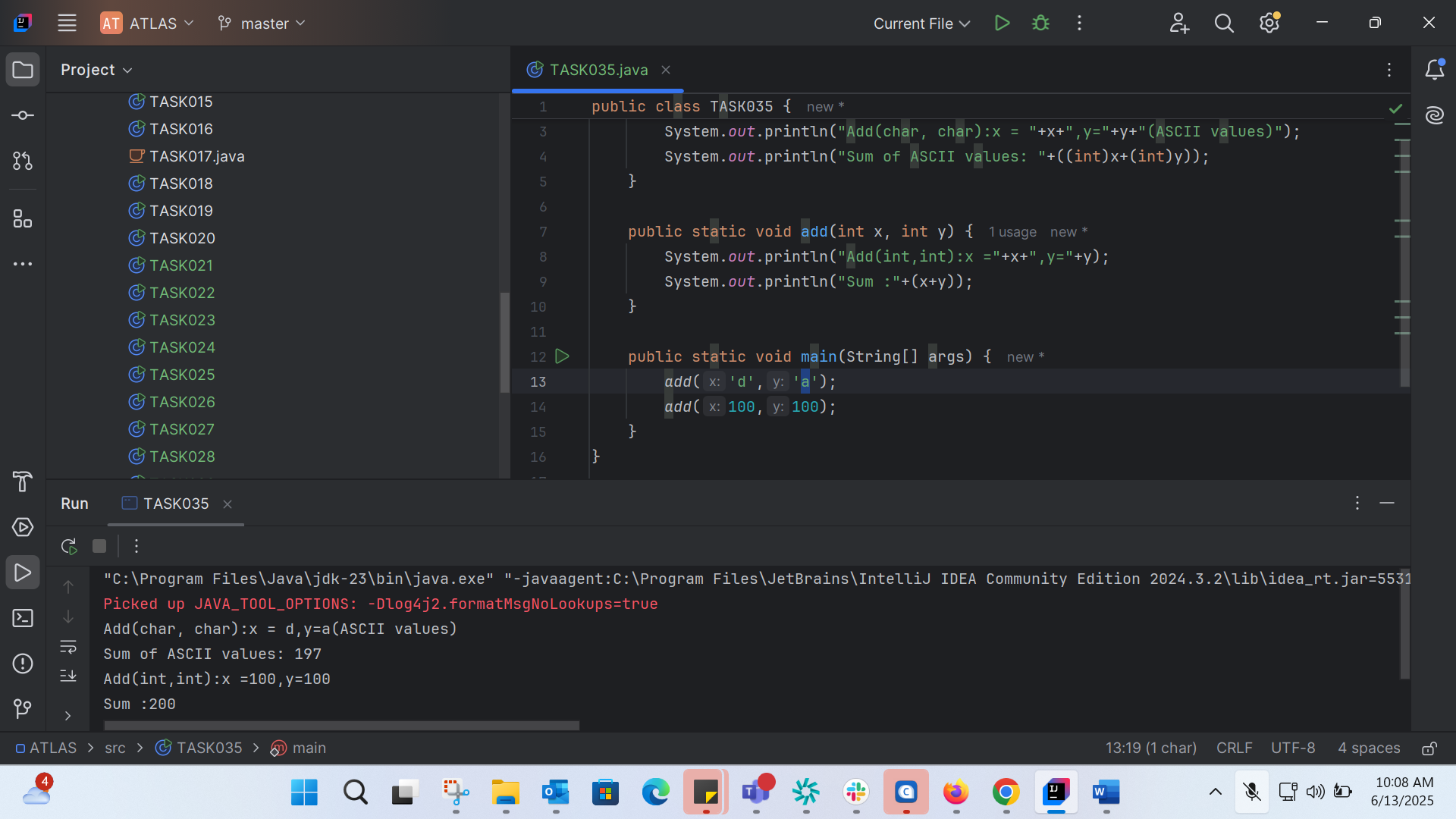
TASK33:



TASK 34:

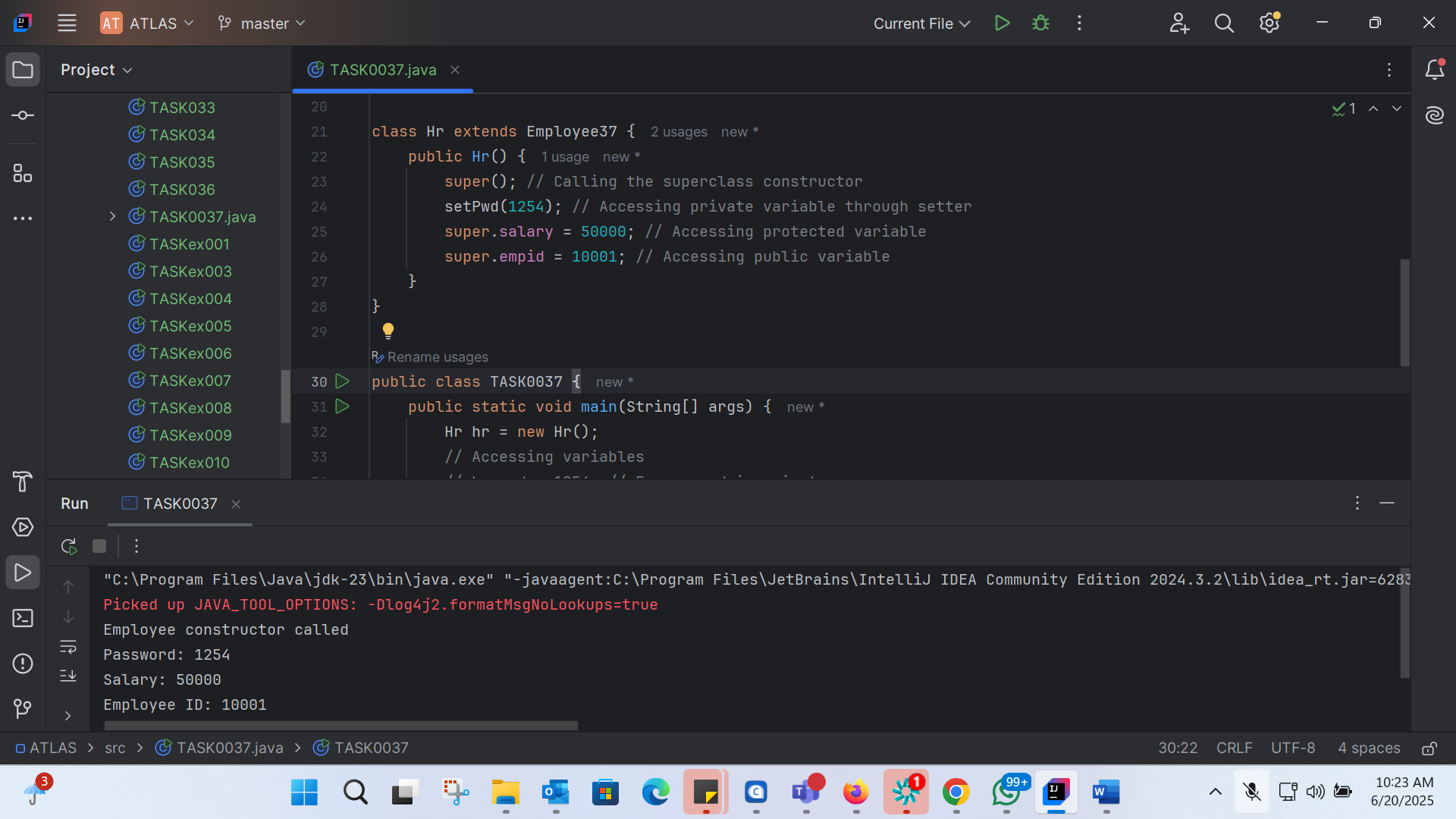


TASK035:



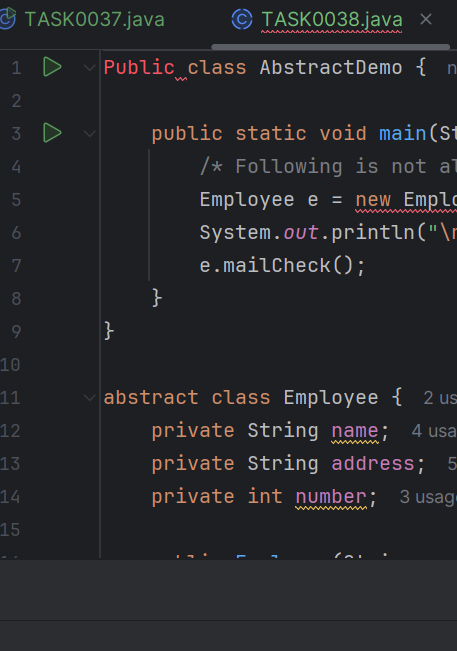
TASK036: 

TASK37:

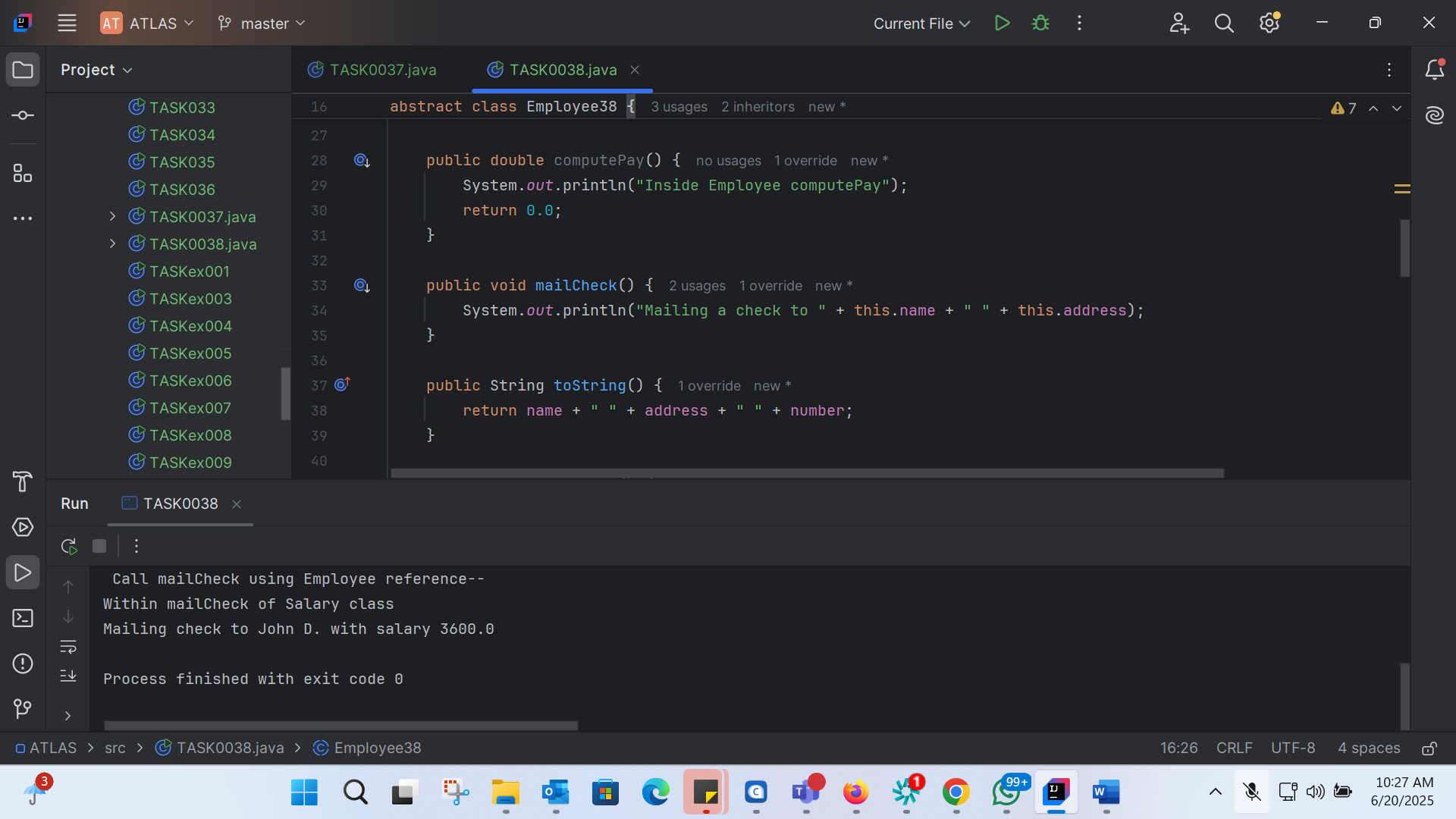


class Employee37 {  
 private int pwd;  
 protected int salary;  
 public int empid;  
  
 // Constructor  
 public Employee37() {  
 System.*out*.println("Employee constructor called");  
 }  
  
 // Getters and setters for private variables  
 public int getPwd() {  
 return pwd;  
 }  
  
 public void setPwd(int pwd) {  
 this.pwd = pwd;  
 }  
}  
  
class Hr extends Employee37 {  
 public Hr() {  
 super(); // Calling the superclass constructor  
 setPwd(1254); // Accessing private variable through setter  
 super.salary = 50000; // Accessing protected variable  
 super.empid = 10001; // Accessing public variable  
 }  
}  
  
public class TASK0037 {  
 public static void main(String[] args) {  
 Hr hr = new Hr();  
 // Accessing variables  
 // hr.pwd = 1254; // Error: pwd is private  
 hr.setPwd(1254); // Accessing private variable through setter  
 hr.salary = 50000; // Accessing protected variable  
 hr.empid = 10001; // Accessing public variable  
  
 System.*out*.println("Password: " + hr.getPwd());  
 System.*out*.println("Salary: " + hr.salary);  
 System.*out*.println("Employee ID: " + hr.empid);  
 }  
}

TASK38:

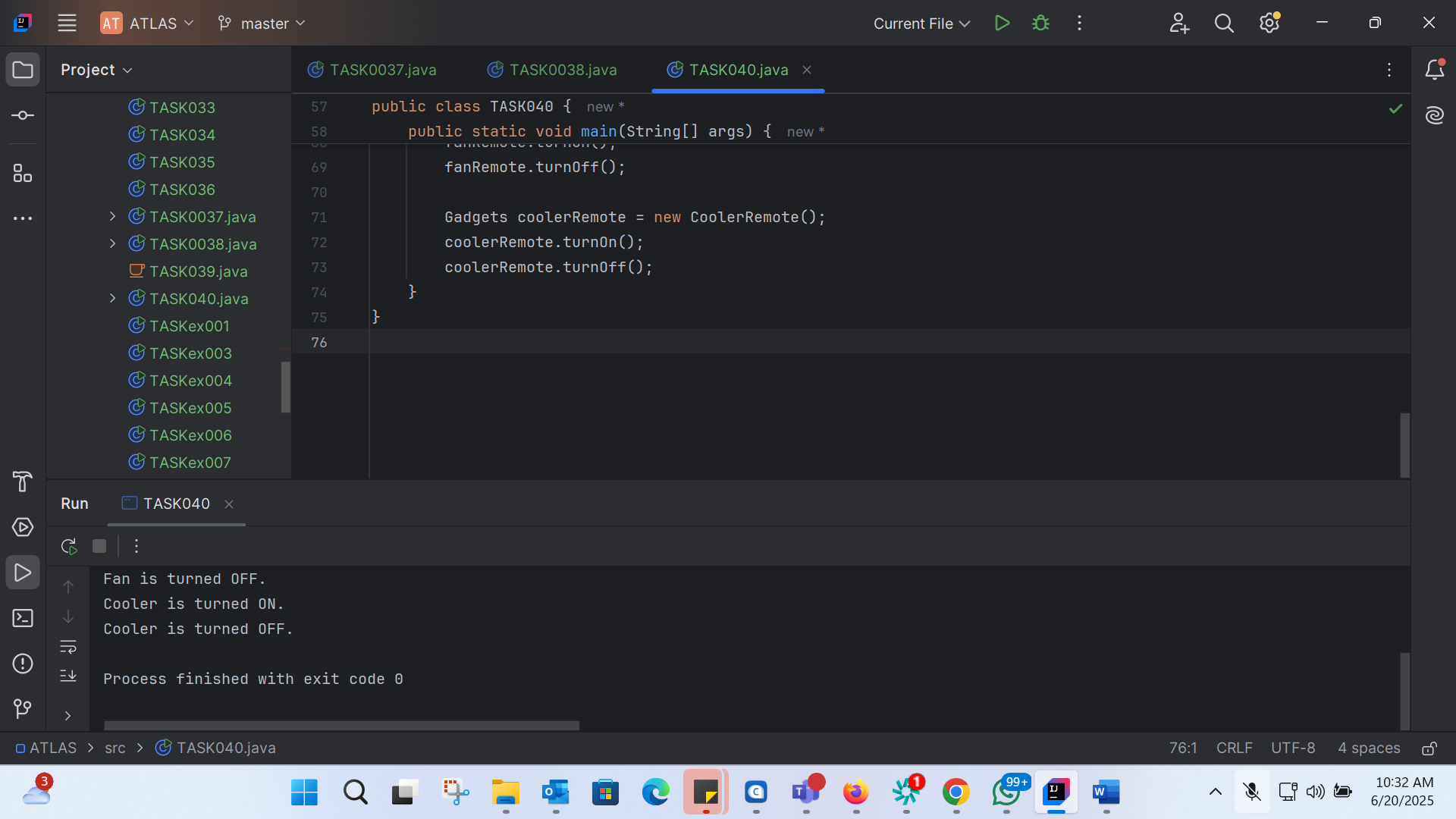


TASK 39:



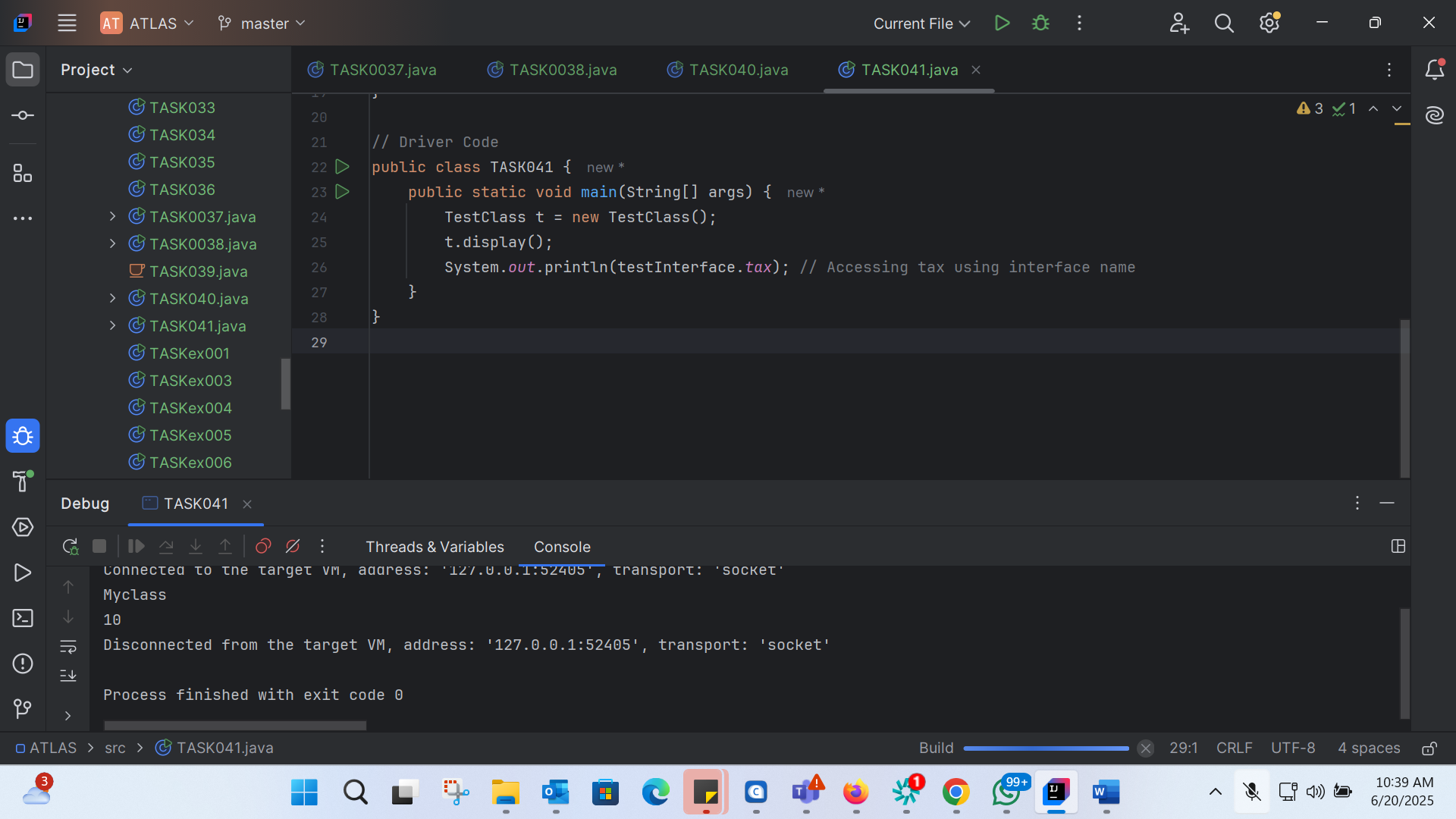
public class TASK0038 {  
 public static void main(String[] args) {  
 // Create an instance of Salary (concrete subclass)  
 Salary s = new Salary("George W.", "Houston, TX", 43, 3600.00);  
 Employee38 e = new Salary("John D.", "Houston, TX", 43, 3600.00); // Using Employee reference  
  
 System.*out*.println("\n Call mailCheck using Salary reference--");  
 s.mailCheck();  
  
 System.*out*.println("\n Call mailCheck using Employee reference--");  
 e.mailCheck();  
 }  
}  
  
// Abstract class Employee  
abstract class Employee38 {  
 private String name;  
 private String address;  
 private int number;  
  
 public Employee38(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 subclass Salary  
class Salary extends Employee38 {  
 private double salary; // Annual salary  
  
 public Salary(String name, String address, int number, double salary) {  
 super(name, address, number);  
 setSalary(salary);  
 }  
  
 public void mailCheck() {  
 System.*out*.println("Within mailCheck of Salary class ");  
 System.*out*.println("Mailing check to " + getName() + " with salary " + salary);  
 }  
  
 public double getSalary() {  
 return salary;  
 }  
  
 public void setSalary(double newSalary) {  
 if (newSalary >= 0.0) {  
 salary = newSalary;  
 }  
 }  
  
 public double computePay() {  
 System.*out*.println("Computing salary pay for " + getName());  
 return salary / 52;  
 }  
}

TASK40:



// Working of Abstraction in Java  
abstract class Gadgets {  
 public abstract void turnOn();  
 public abstract void turnOff();  
}  
  
// Concrete class implementing the abstract methods  
class TVRemote extends Gadgets {  
 @Override  
 public void turnOn() {  
 System.*out*.println("TV is turned ON.");  
 }  
  
 @Override  
 public void turnOff() {  
 System.*out*.println("TV is turned OFF.");  
 }  
}  
  
class ACRemote extends Gadgets {  
 @Override  
 public void turnOn() {  
 System.*out*.println("AC is turned ON.");  
 }  
  
 @Override  
 public void turnOff() {  
 System.*out*.println("AC is turned OFF.");  
 }  
}  
  
class FanRemote extends Gadgets {  
 @Override  
 public void turnOn() {  
 System.*out*.println("Fan is turned ON.");  
 }  
  
 @Override  
 public void turnOff() {  
 System.*out*.println("Fan is turned OFF.");  
 }  
}  
  
class CoolerRemote extends Gadgets {  
 @Override  
 public void turnOn() {  
 System.*out*.println("Cooler is turned ON.");  
 }  
  
 @Override  
 public void turnOff() {  
 System.*out*.println("Cooler is turned OFF.");  
 }  
}  
  
// Main class to demonstrate abstraction  
public class TASK040 {  
 public static void main(String[] args) {  
 Gadgets tvRemote = new TVRemote();  
 tvRemote.turnOn();  
 tvRemote.turnOff();  
  
 Gadgets acRemote = new ACRemote();  
 acRemote.turnOn();  
 acRemote.turnOff();  
  
 Gadgets fanRemote = new FanRemote();  
 fanRemote.turnOn();  
 fanRemote.turnOff();  
  
 Gadgets coolerRemote = new CoolerRemote();  
 coolerRemote.turnOn();  
 coolerRemote.turnOff();  
 }  
}

TASK041:

  
import java.io.\*;  
  
// Interface Declared  
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  
 @Override  
 public void display() {  
 System.*out*.println("Myclass");  
 }  
}  
  
// Driver Code  
public class TASK041 {  
 public static void main(String[] args) {  
 TestClass t = new TestClass();  
 t.display();  
 System.*out*.println(testInterface.*tax*); // Accessing tax using interface name  
 }  
}

TASK042:

Constants:

- Constants are values that do not change during the execution of a program.

- In Java, constants are typically declared using the final keyword along with static.

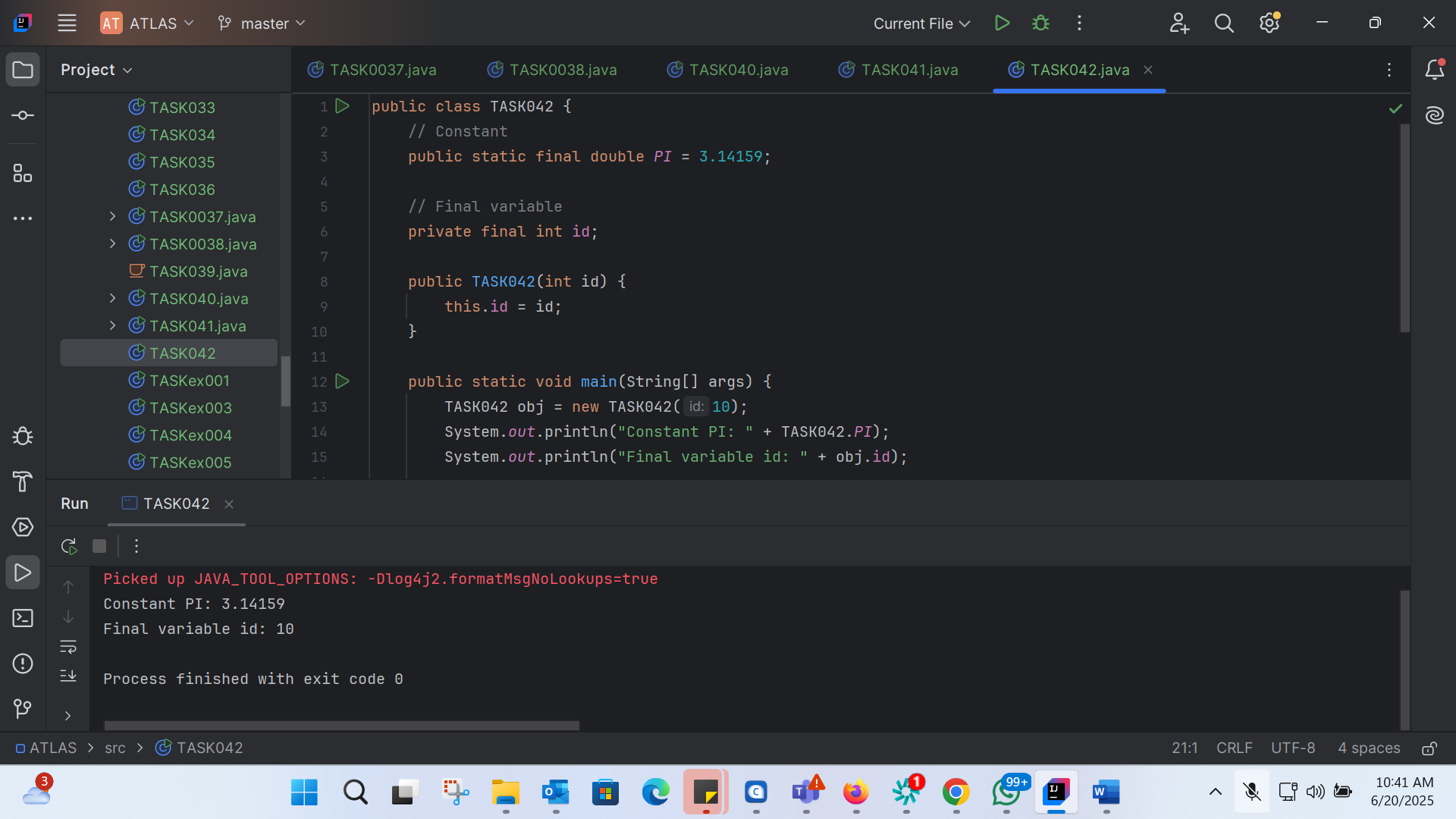
- Constants are usually declared at the class level and are shared by all instances of the class.

Final Variables:

- Final variables are variables that can be assigned a value only once.

- Once a final variable is assigned a value, it cannot be reassigned.

- Final variables can be declared at the class level, instance level, or local level.



public class TASK042 {  
 // Constant  
 public static final double *PI* = 3.14159;  
  
 // Final variable  
 private final int id;  
  
 public TASK042(int id) {  
 this.id = id;  
 }  
  
 public static void main(String[] args) {  
 TASK042 obj = new TASK042(10);  
 System.*out*.println("Constant PI: " + TASK042.*PI*);  
 System.*out*.println("Final variable id: " + obj.id);  
  
 // Trying to reassign a value to a final variable will result in a compile-time error  
 // obj.id = 20; // Uncommenting this line will cause a compile-time error  
 }  
}