**Student Assessment Submission and Declaration**

When submitting evidence for assessment, each student must sign a declaration confirming that the work is their own.

|  |  |  |  |
| --- | --- | --- | --- |
| Student name: Rashed Hassan Qahah. | | Assessor name: | |
| Issue date (1St Submission): | Submission date (1St Submission): | | Submitted on: |
| In case of resubmission | | | |
| Issue date (2nd Submission): | Submission date (2nd Submission): | | Submitted on: |
|  | | | |
| Assignment number and title: 1: OOP Concepts | | | |

**Task 1:**

Briefly describe the following Object-oriented concepts. Provide diagrams and code snippets (**you must write code**) to supplement your explanations.

1. Static variables and methods.

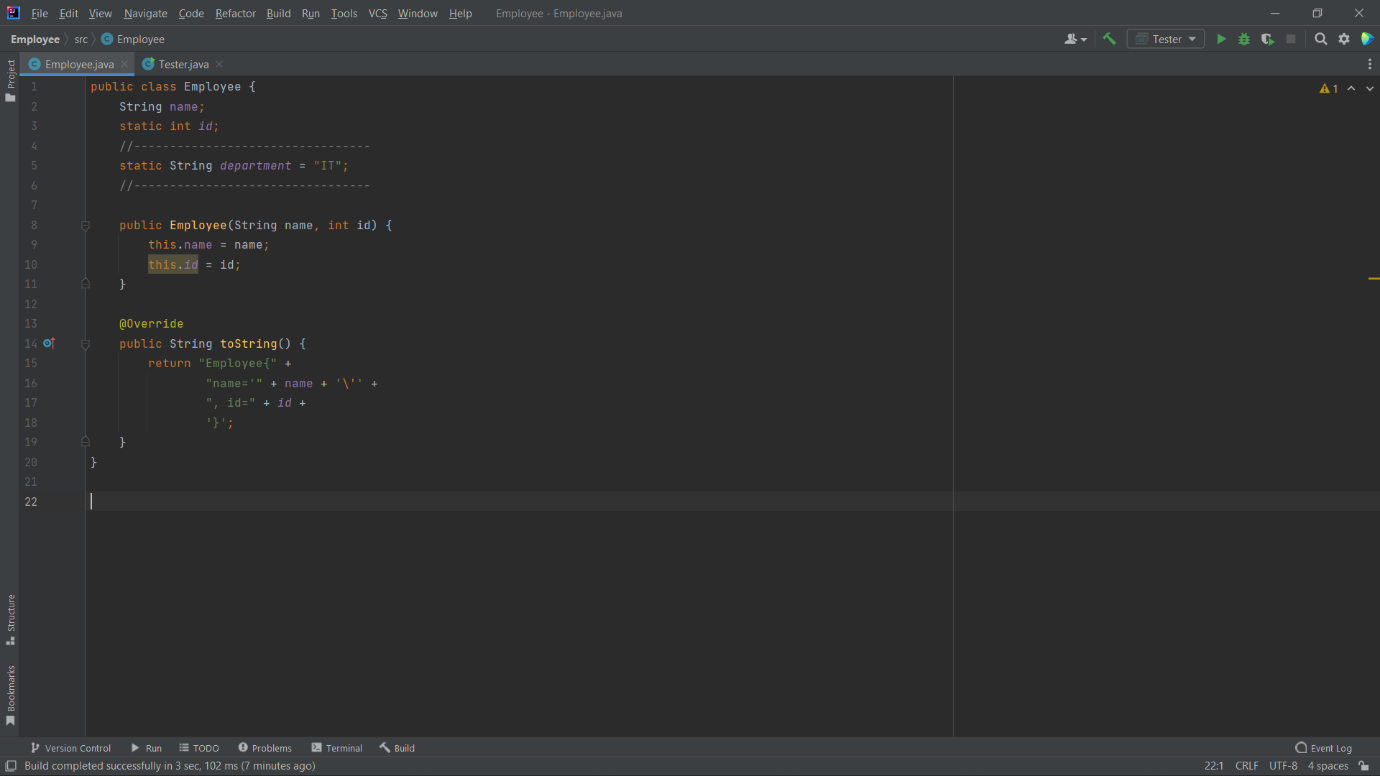
Answer:

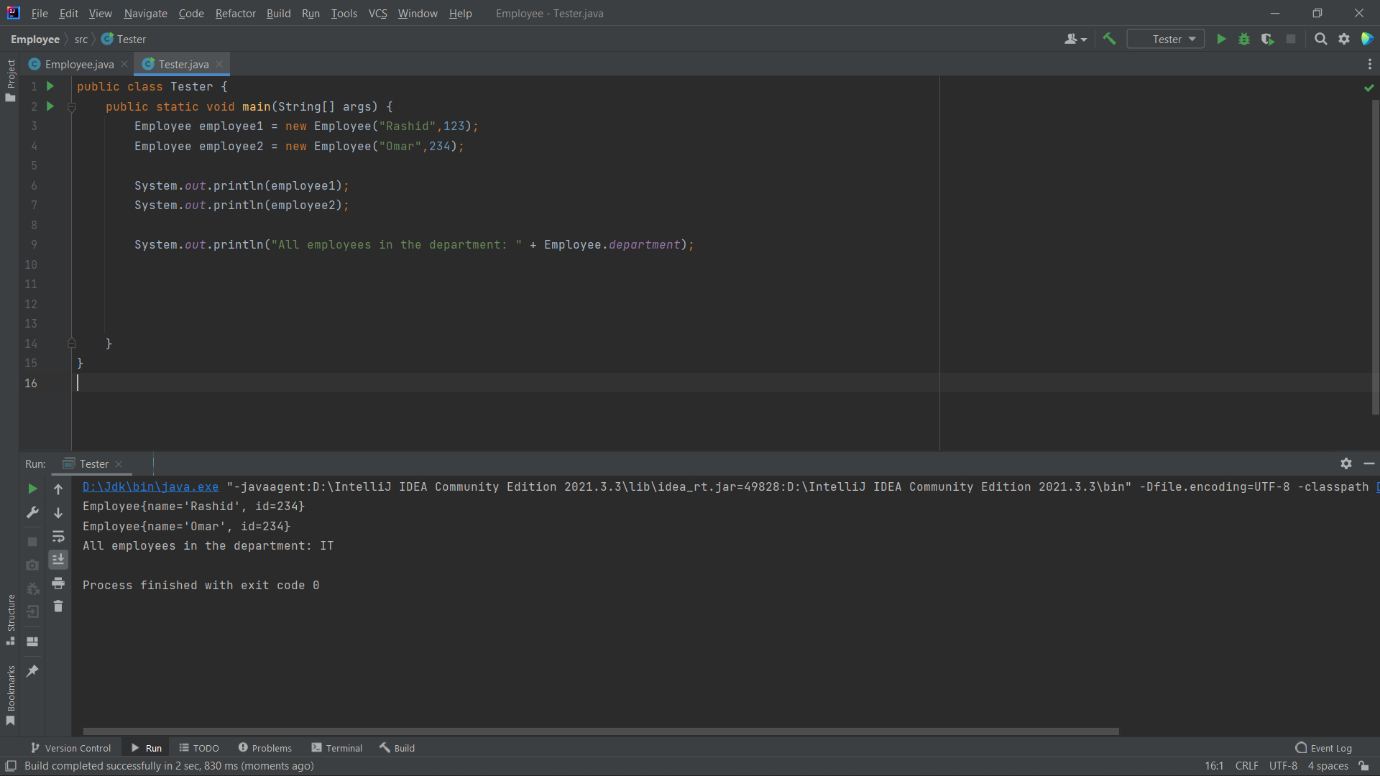
Static variable

If we declare a variable as static here, this variable becomes the static variable.

We benefit from converting the variable to a static variable, so the property that is the variable will become common to all objects, and this means that it is not private or exclusive to a specific object or instance.

A static variable is reserved in memory only once, which makes the program memory efficient, and it is specific to the class itself so that it can be called by the class name.

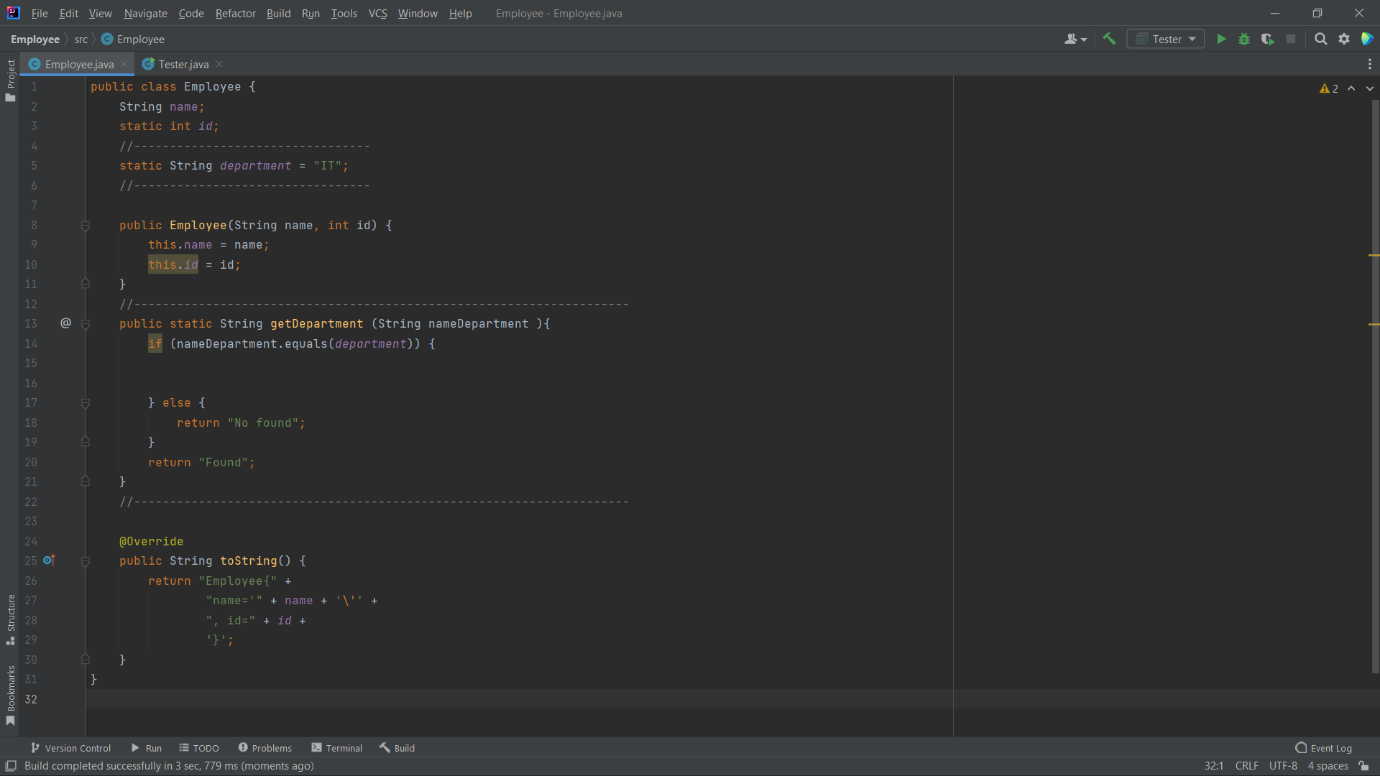


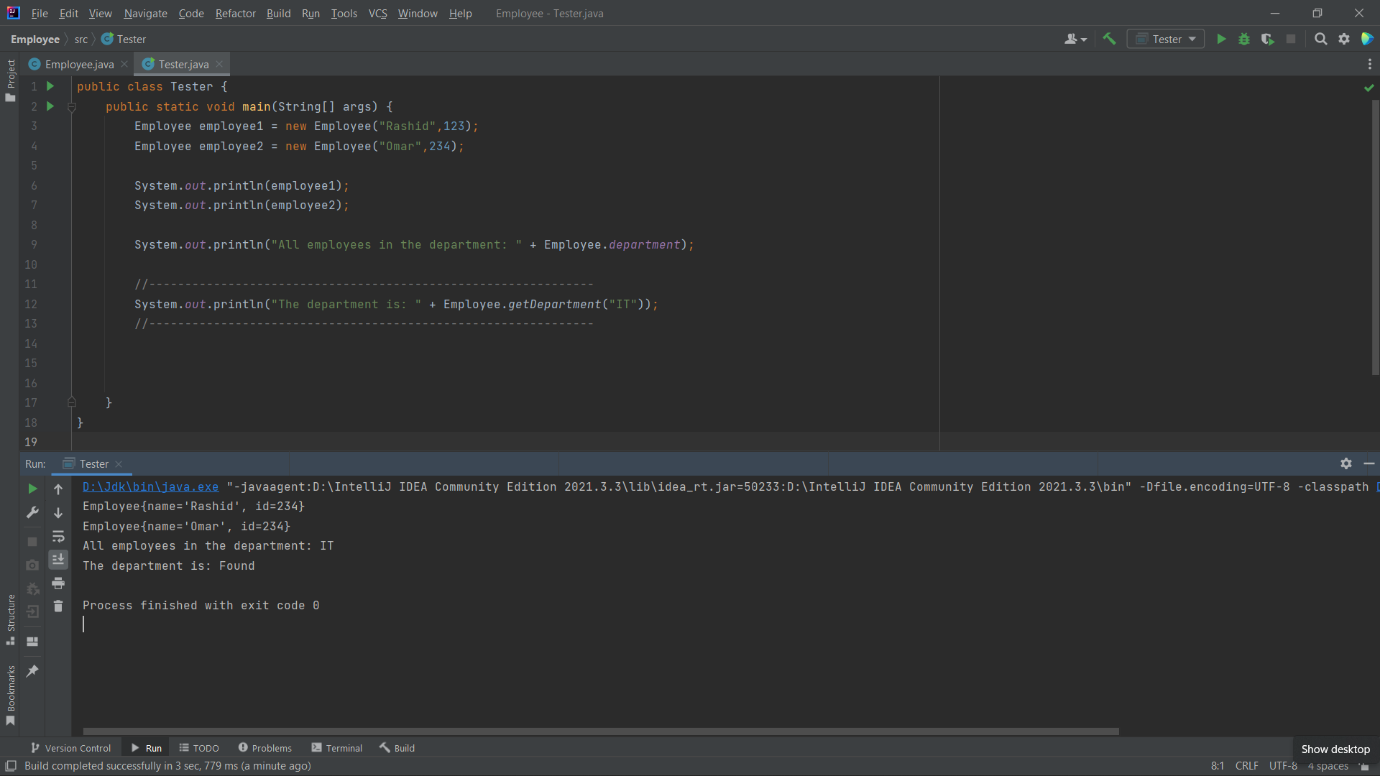


Static methods

We can use static by methods, take advantage of static methods being called without object argument.

Static methods are called by class name.





ii) Default constructor and parametrise constructors.

Answer:

- Rules for creating constructor:

- Constructor must be the same as the class name.

- Constructor must not be an explicit return type.

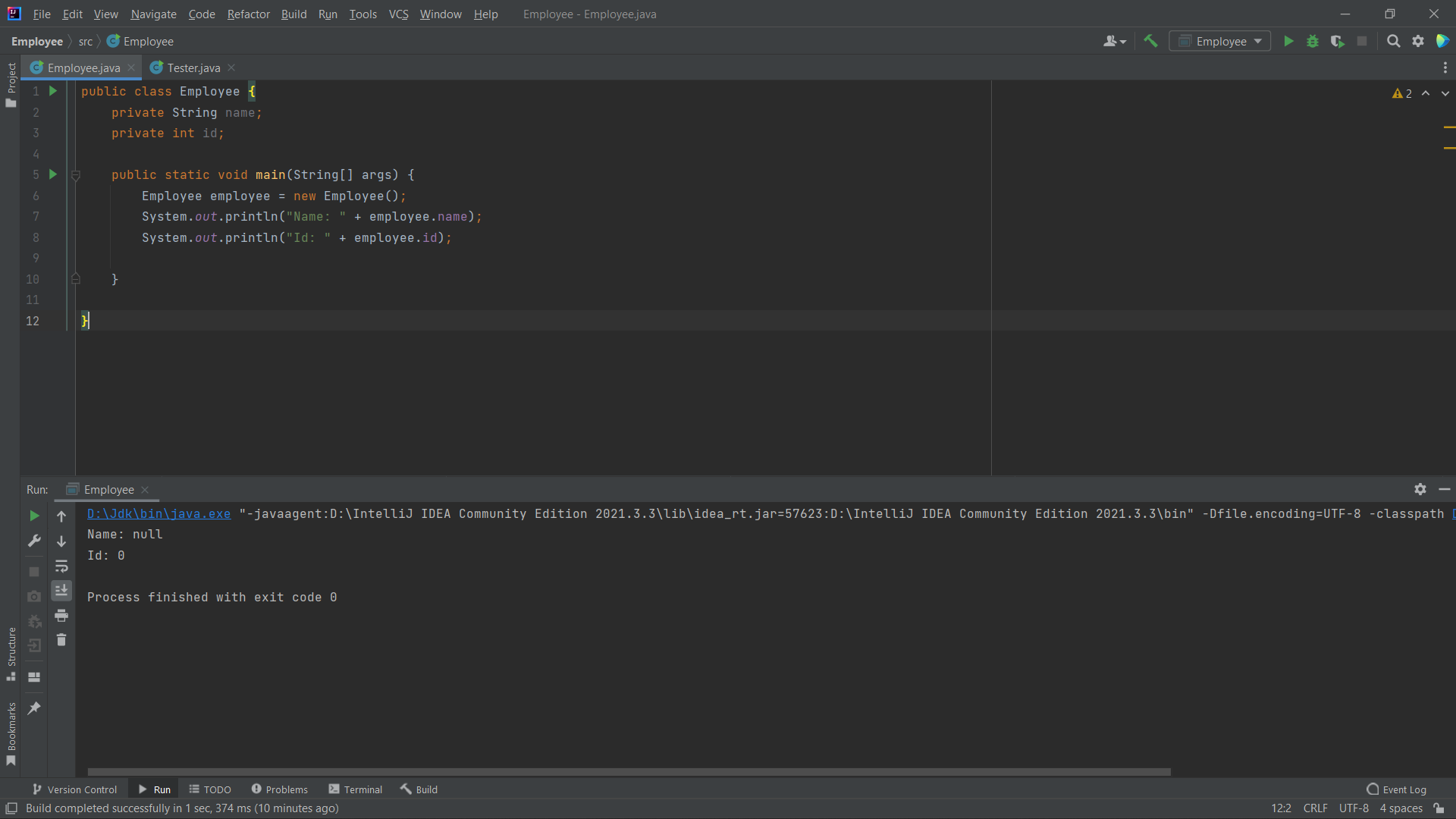
- Constructor cannot be final, static, abstract, and synchronized.

If we do not create any Constructor, the Java compiler creates a no-arg when executing the program. This is called the default constructor.

public class Employee {  
 private String name;  
 private int id;  
  
 public static void main(String[] args) {  
 Employee employee = new Employee();  
 System.*out*.println("Name: " + employee.name);  
 System.*out*.println("Id: " + employee.id);  
  
 }  
  
}

Here we didn't create any constructors, after that we can put default constructor or if we don't, java will create it automatically.

public class Employee {  
 private String name;  
 private int id;  
   
 public Employee(){  
  
 }  
  
 public static void main(String[] args) {  
 Employee employee = new Employee();  
 System.*out*.println("Name: " + employee.name);  
 System.*out*.println("Id: " + employee.id);  
  
 }  
  
}

Output:

|  |  |
| --- | --- |
| Type | Default Value |
| boolean | false |
| byte | 0 |
| short | 0 |
| int | 0 |
| long | 0L |
| char | \u0000 |
| float | 0.0f |
| double | 0.0d |
| String | null |
| object | Reference null |

Here the default constructor initializes any variables so that if they are not initialized with default values.

The Constructor can also be one or more Parameterized so that it can be more than one type of data type or of the same type and this Constructor is known as parameterized constructor.

public class Employee {  
 String name;  
  
 public Employee(String name) {  
 this.name = name;  
 }  
  
 public static void main(String[] args) {  
 Employee employee = new Employee("Rashid");  
 System.*out*.println("Name: " + employee.name);  
  
  
 }  
  
}

Here we created a constructor named employee 🡪 public Employee()

After that, the constructor takes a single parameter.

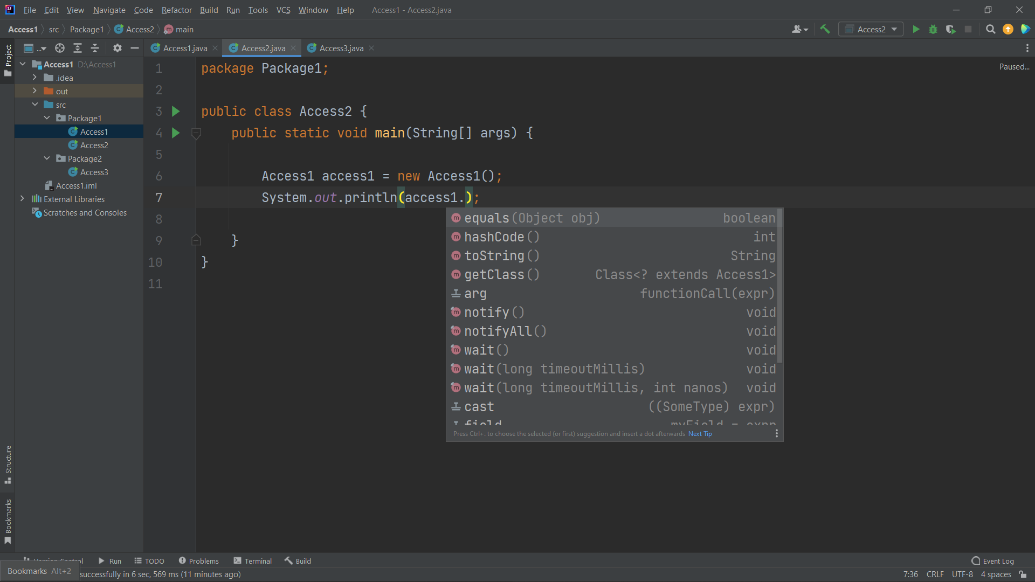
Employee employee = new Employee("Rashid");

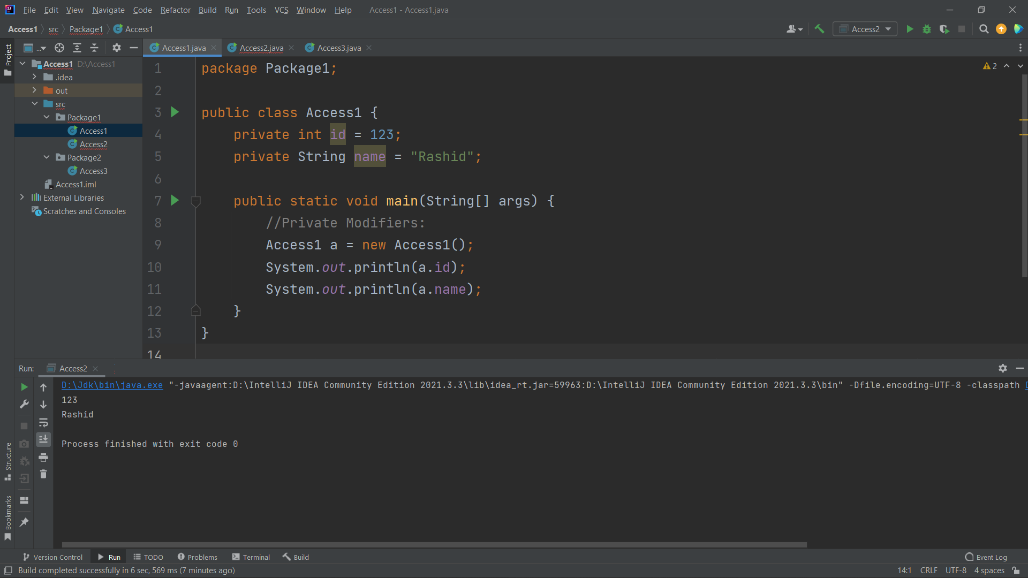
Here, we will be passing the single value to the constructor.

iii) Access modifiers and encapsulation.

Answer:

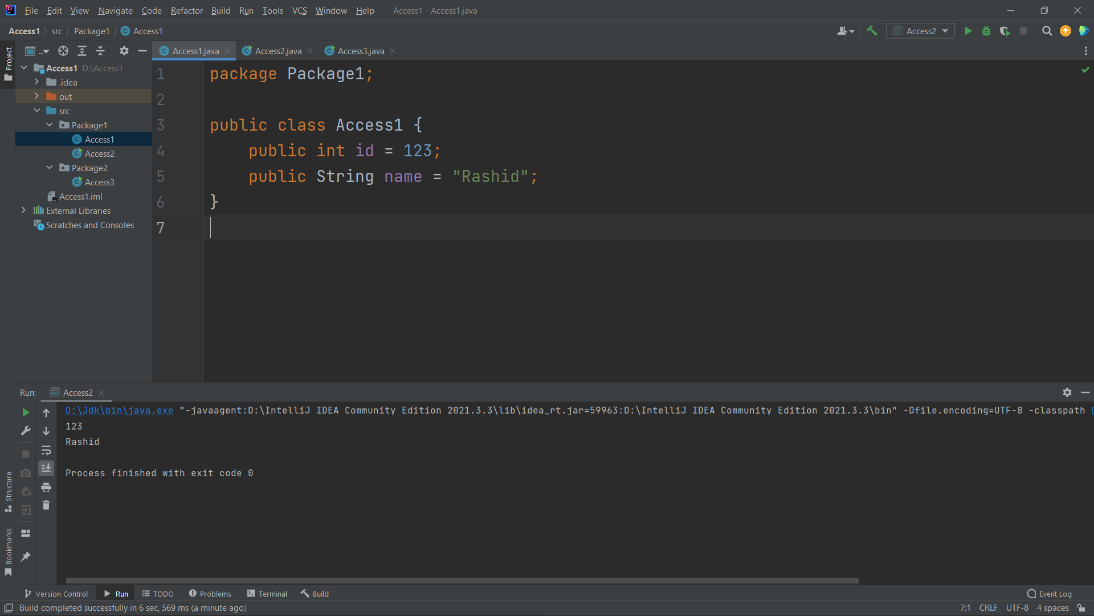
Private: The level of access to the private modifier will only be within the class itself and cannot be accessed from outside the class, this means that you can see it in the same class, but if it is outside the class, you will not be allowed to see it.

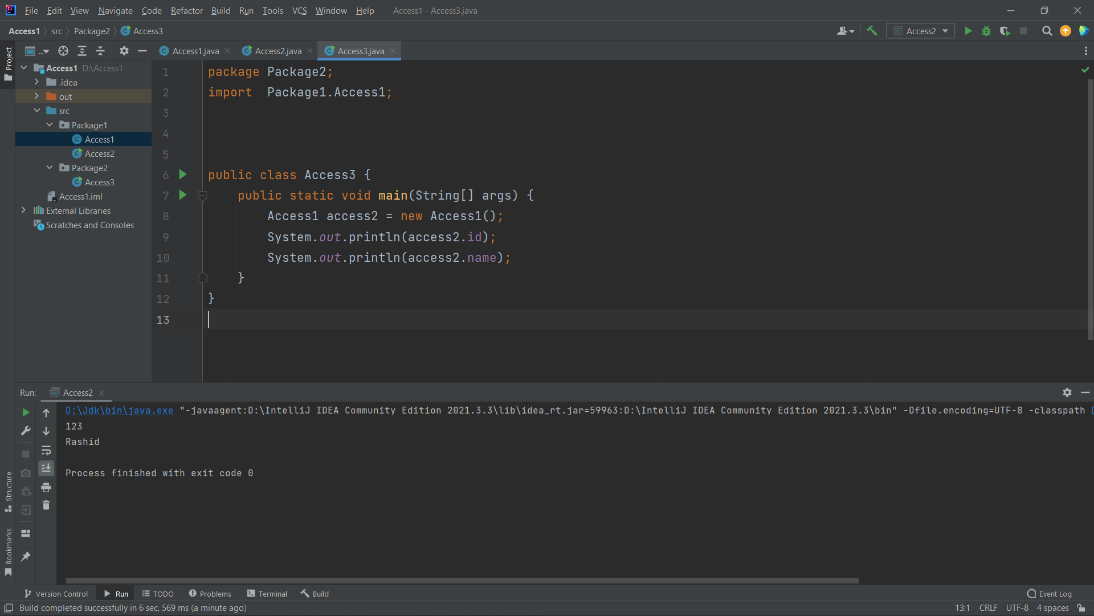
Example:



Public: If we set the global modifier access level to be public, this means that we can access it from inside the class, from outside the class, from inside the package, and from outside the package.

Example:



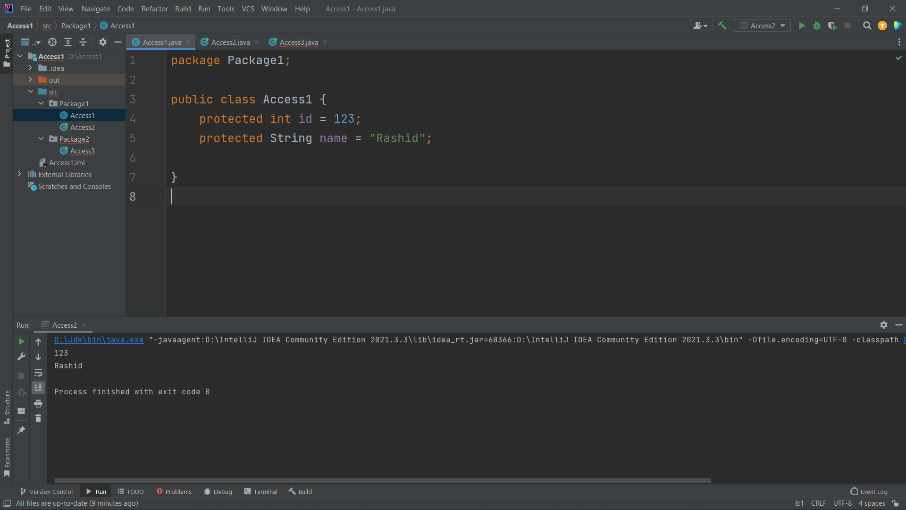


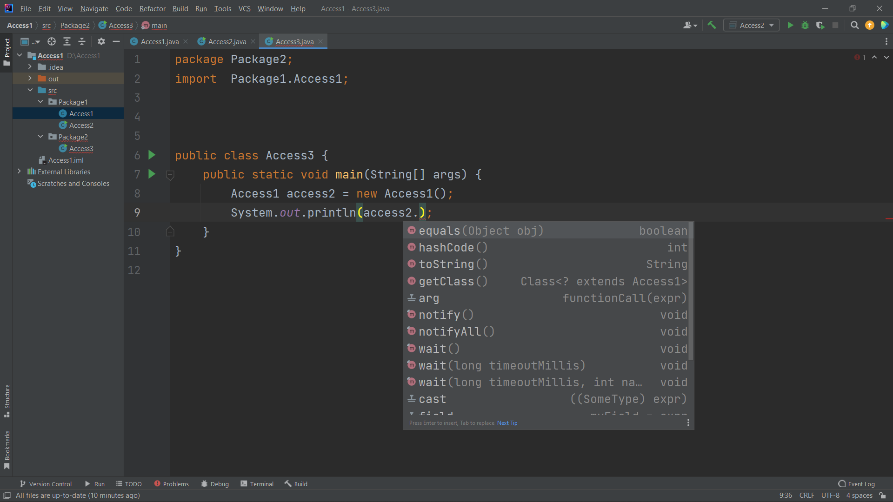
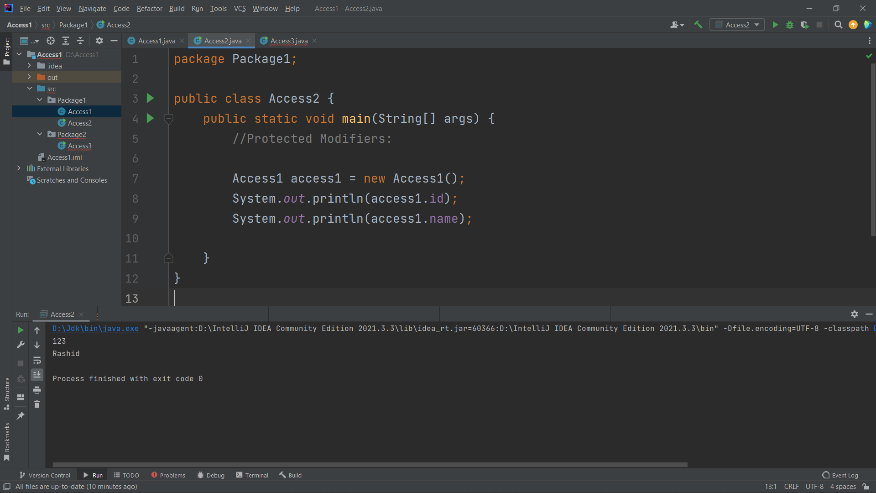
Protected: Here is the protected access level

Rate inside and outside the package package through sup class. If we don't sup class, not accessible from out of the package.

This means that I can see the attributes or do an access operation from the super class in addition to the sup class. In this case, the Access Modifiers are protected, meaning that any methods or attributes are protected. Direct access from the super class will work for them, in addition to the sup class.

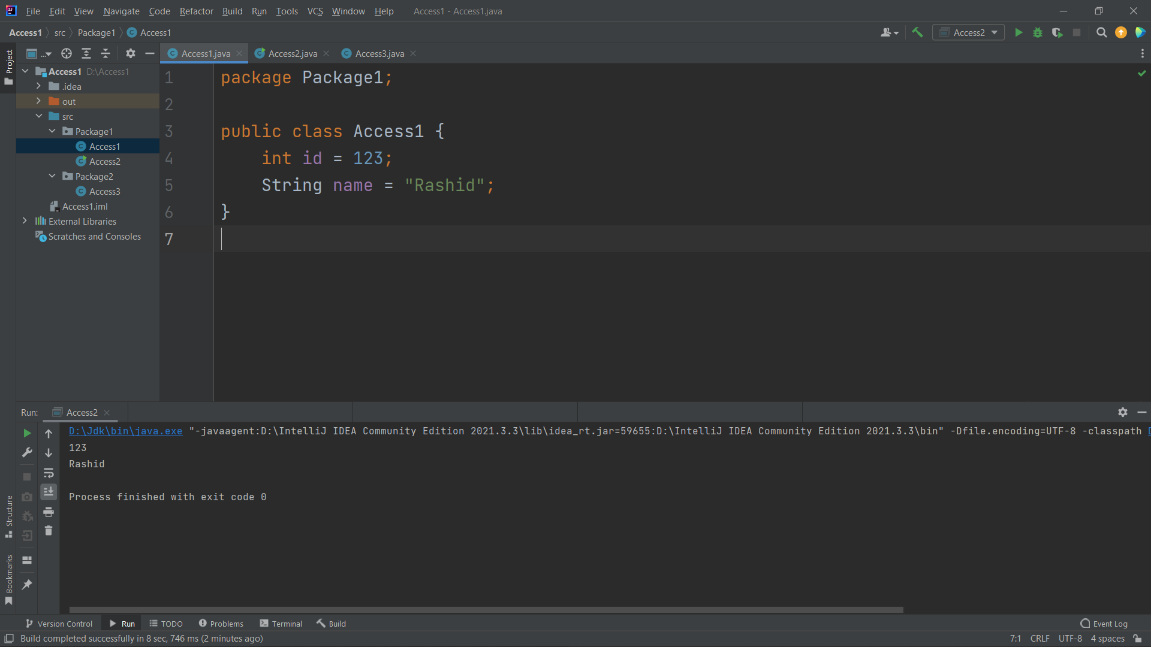
In the protected, any class can be accessed if it is inside the package, but if the class is outside the package, it cannot be accessed.

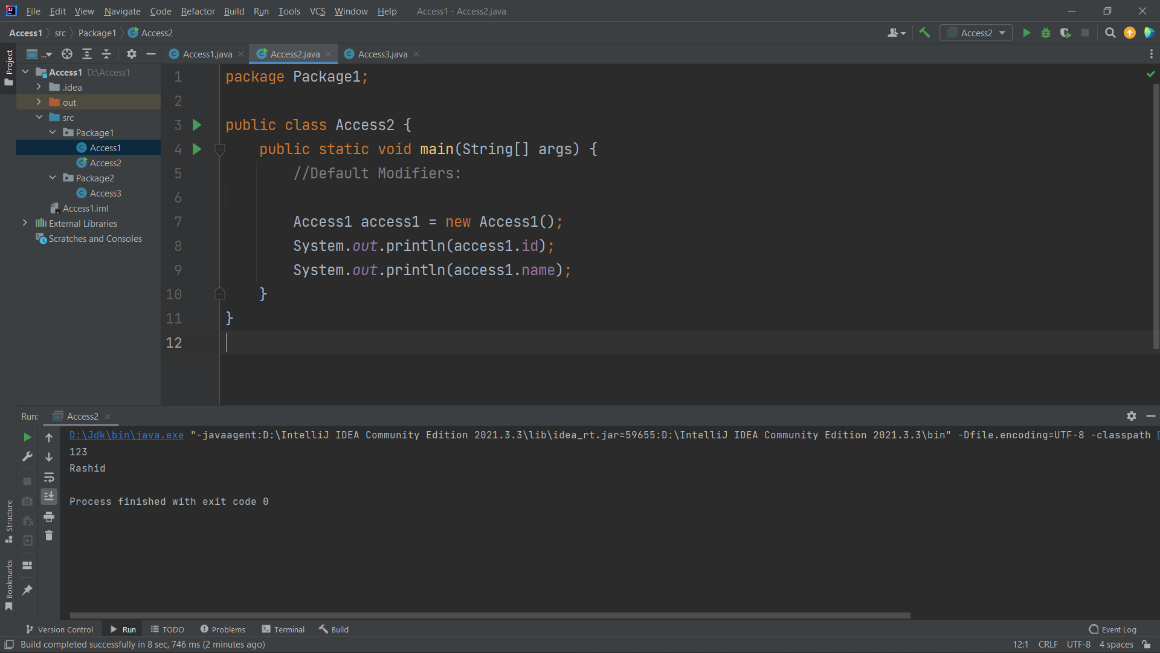
Example:



Default: If we do not use any access modifiers, it will be treated by java as a default. The default modifier can be accessed from within the package only and cannot be accessed from outside the package, but provides easier access than the private modifier, but it is more restrictive than it is a protected modifier or a public modifier.

Example:





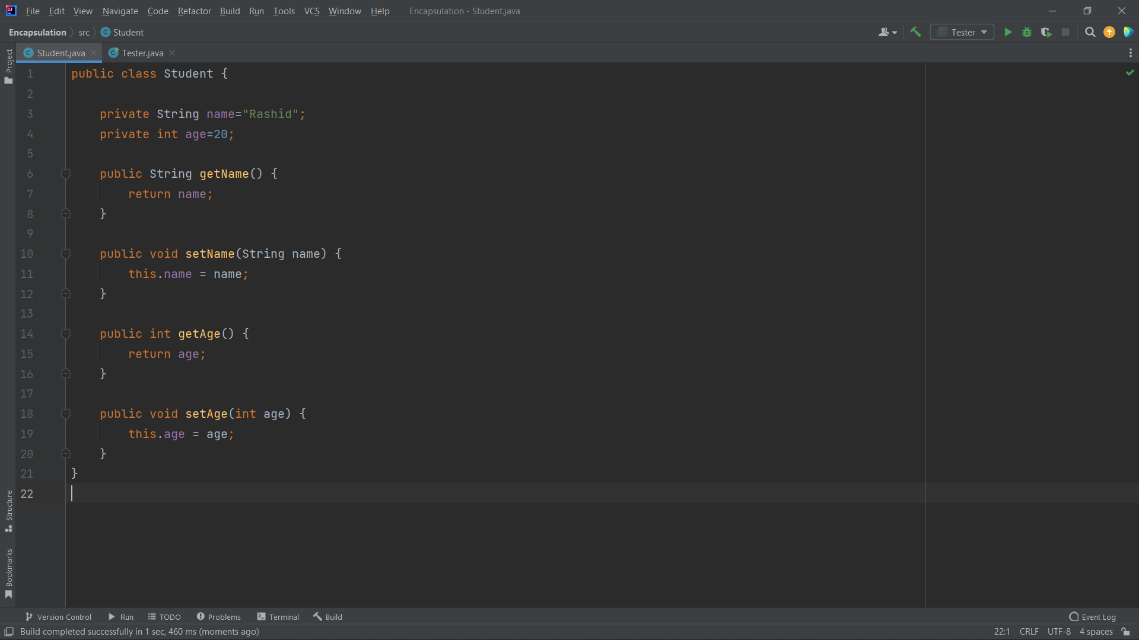
Encapsulation:

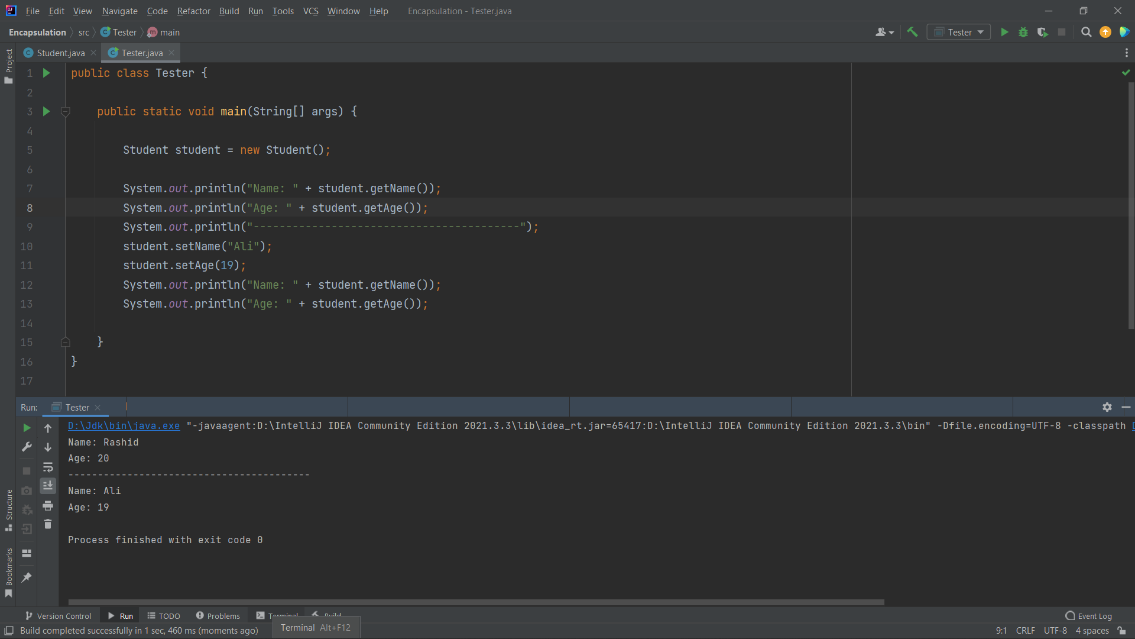
Encapsulation in Java refers to a code-wrapping mechanism that binds the code and the data it processes together into a single unit, for example a medicine capsule has mixed types of modules inside. We can use getter and setter methods to specify the data and how to get it.

Encapsulation kind of sounds like in a little capsule so if you think of like a pill everything is inside a pill.

But during capsulation in java all the setting of variables of a class is inside a method.

Example:





iv) Abstraction.

Answer:

It is the process of hiding the details or internal implementation of a method or feature and showing only basic information or functionality to the user.

We can achieve or access an abstraction using abstract classes or interfaces.

When we want to create an abstract class here, we cannot create any object of this class.

So, what do we benefit from an abstract class?

We benefit from the abstract class through inheritance, when creating an abstract class from a class, another class must inherit from it, because if it is not inherited from it, then we do not benefit from the abstract class.

- Abstract Class: Partial Abstraction (0 to 100%)

- Interface: Full Abstraction (100%)

When we want to add an abstract class, an abstract class must be declared with the keyword that is reserved by default in Java, and it cannot be instantiated.

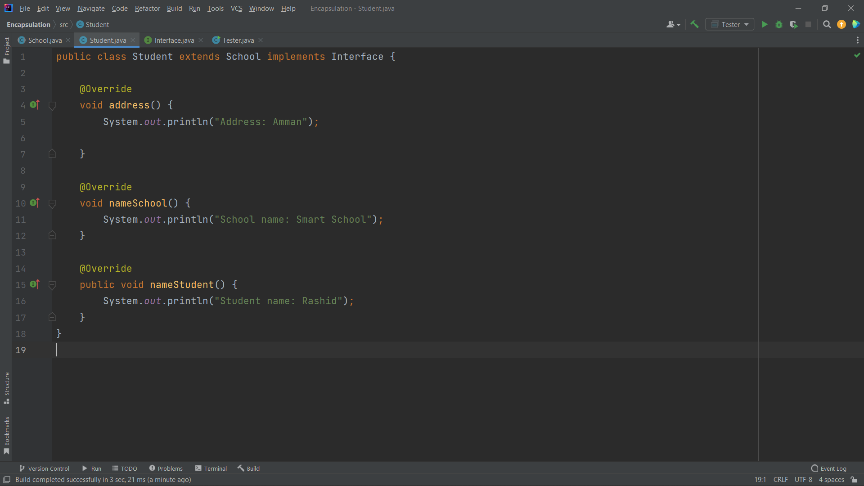
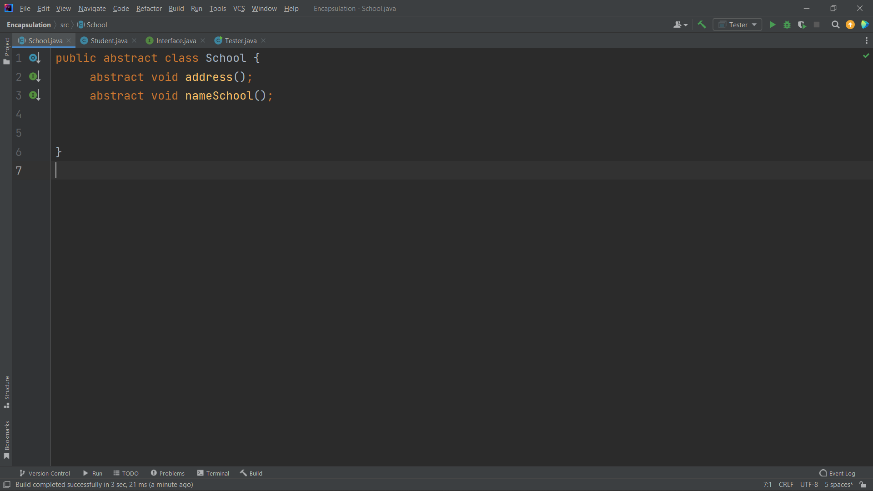
Iinterfaces:

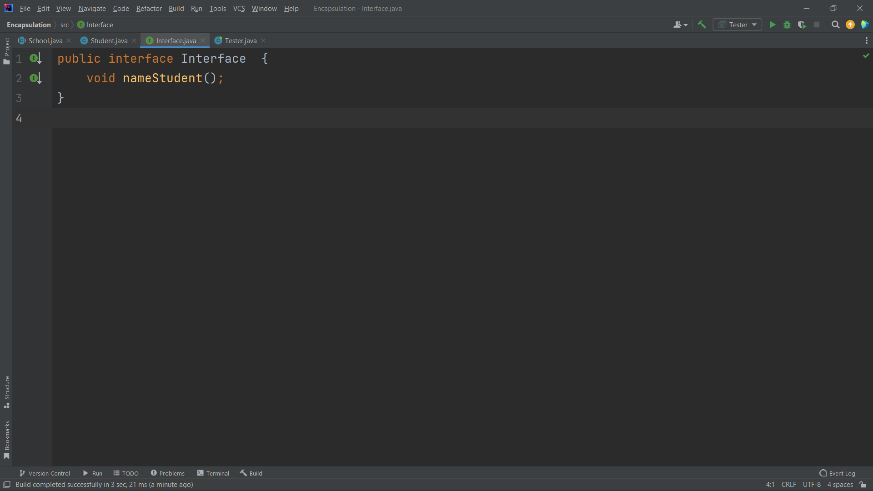
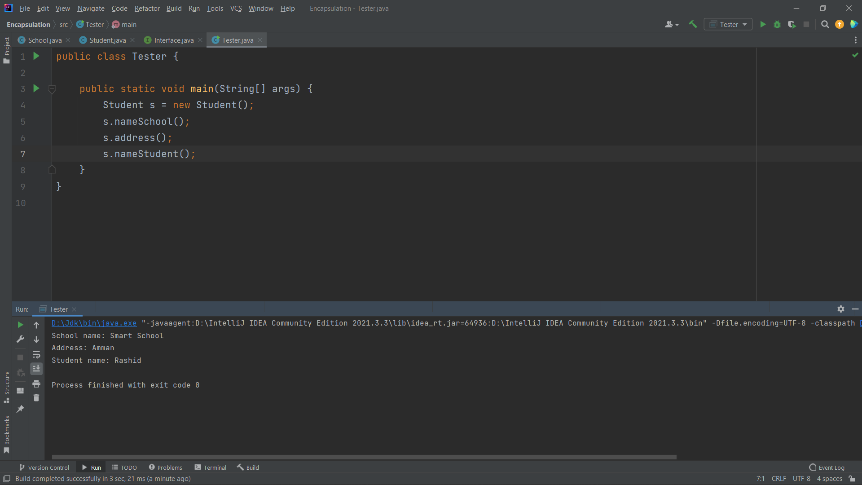
The interface here is like an abstract class in that it contains all the abstract methods.

It also cannot be created for it, and we must write all the methods in the interface elsewhere.

So, what is the big difference between an abstract class and an interface? The difference is that an interface is all the methods in which an abstract class exists only.

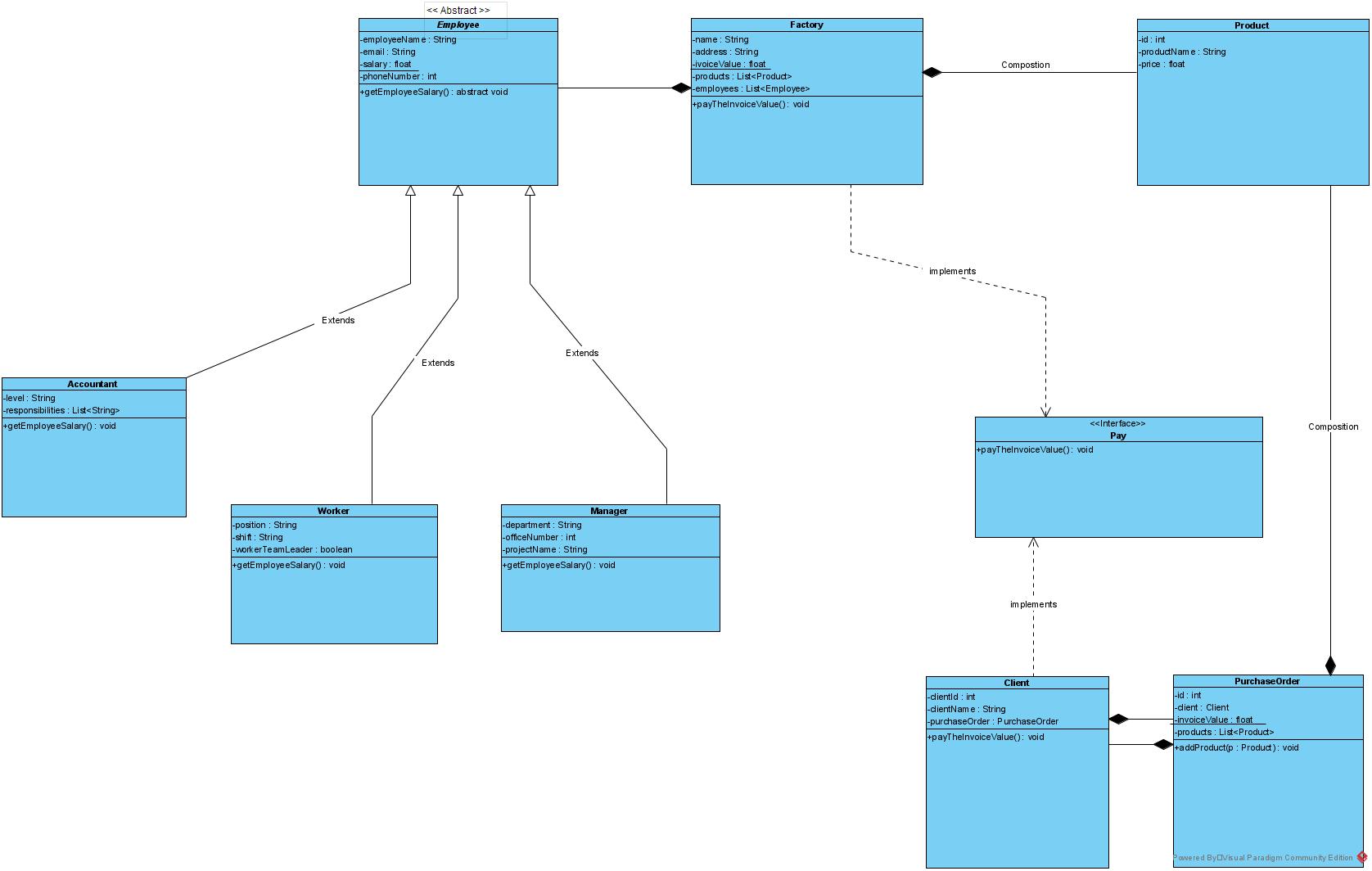
Example:





**Task 2**

1. Create a detailed UML class diagram for the factory system using a suitable UML tool. The classes should include attributes and methods.



1. Explain one inheritance and one composition relationships existed in your UML.

**Answer:**

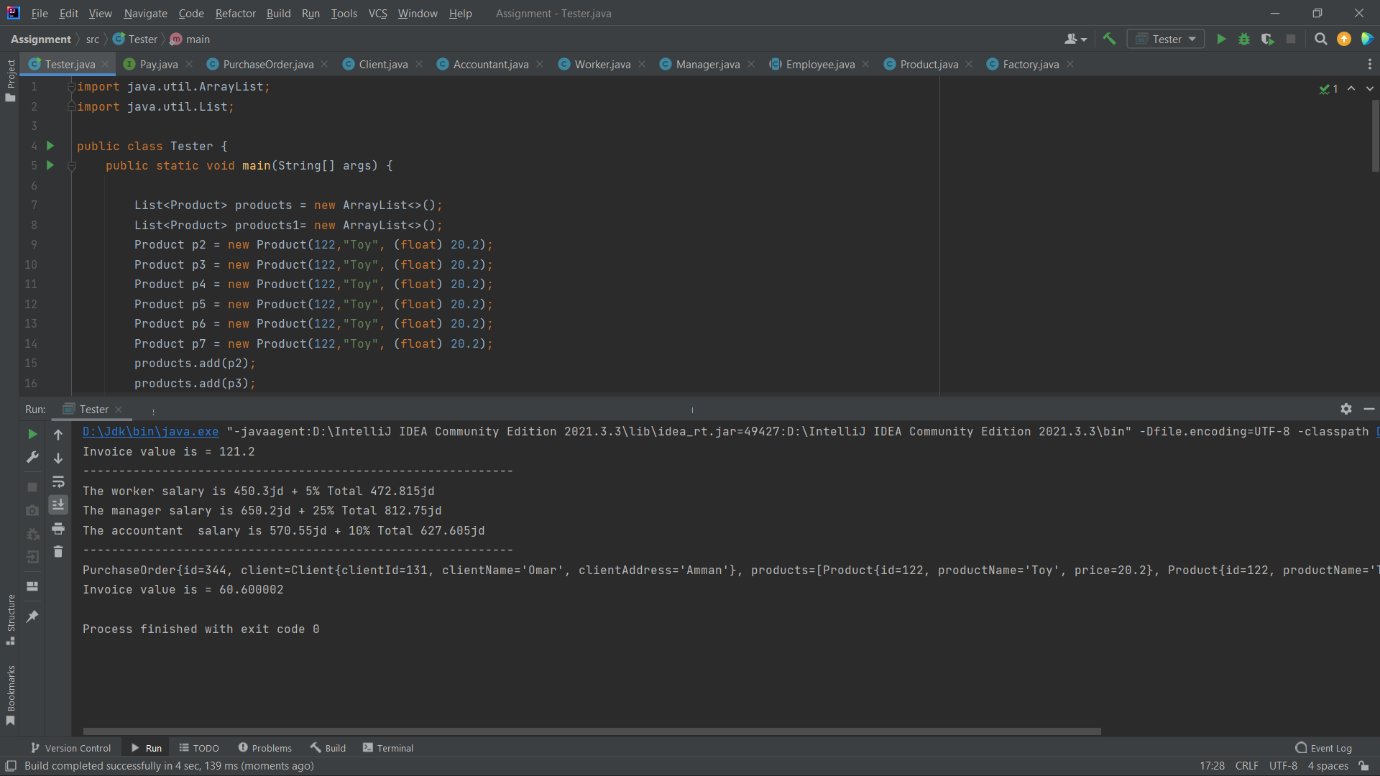
In composition, both classes are dependent on each other, so for example is it possible to be in a library that has no books, of course not so in composition both classes must be dependent on each other.

First, in the factory class we have a composition of type Array List for the product which has each product consisting of id, name and price and we know that products are not the same thing so we made a composition of type Array List and also we have a composition for employees of type Array List because we know that employees can be more than Person So we created composition of type Array List.

The employees have three types of employees, including the worker, the accountant, and the manager, all of whom share a name, email, salary, and phone number

Therefore, we understand that this is to do three classes and each class has a type of employee, so we inherit the three classes from the Super Class because they share the name, email, salary and phone number by adding extends next to the name of the class and also in the Employees class we have to add a way to calculate the salary for each A type of employee, so we created an abstract for the employee class, which is a superclass, which forces the classes that inherited from the superclass, which is employees, by adding the getEmployeeSalary method to calculate each employee type with its own salary.

**Task 3**

1. Implement the system using a suitable object-oriented language. Provide screen shots (in addition to source code) as evidence of program execution.

**Task 4**

1. The factory system has a HR system and they want to integrate it with your new developed system. The main problem is that they do not have any documentation. You need to extract the UML class diagram from the already existed code to able to do the task. Investigate how we can create a tool to extract the UML automatically from the given code in details.

Answer:

First, we have to track our code, so if we find the word extends next to the class name, this means that this class has inherited from the superclass.

Composition is accomplished through the use of an instance variable that references other objects. Composition occurs when one object contains another object and the contained object cannot exist without the existence of the other object.

Thus, we can know from the code a way or representation of the composition:

private PurchaseOrder purchaseOrder;

Abstract which is written or known by tracking code like this:

public abstract class Employee

Here we know when writing an abstract when creating the class that it is a keyword about the non-access rate, used for classes or methods

, An abstract class is a restricted class that cannot be used to create objects (to access them, they must be inherited from another class).

Thus, when tracking the code, we know how to represent it on the UML, or where the abstract is located, and with any class or any method.

encapsulation and here is how we verify by tracking the code that there is encapsulation to represent it on the UML and so when declaring class variables as private and also encapsulation refers to merging data (variables) and code (methods) into one unit in encapsulation the class variables are hidden from other classes and cannot be accessed to it only through the separation methods in which it is located.

**Task 5:**

1. There are three categories of design patterns which are creational, behavioural and structural. Given the list below that has design patterns examples. Sort them into the three categories and explain your choices.
   1. Singleton Pattern

**Answer:**

Creational design patterns focus and concern on the method of creating objects, which use these design patterns when a decision must be taken when creating an object from a class, and this indicates from its name and it is concerned with how to create an object, and we know when we create an object, we put the name of a super class or class and then the name of an object and constructor.

Like this:

Class object = new Class();

But here there are certain methods, for example we want to create a class, then we create one object for this class only, and we cannot then create another object from the same class, and this applies to the most widely used design pattern which is the singleton pattern.

So, anything related to creating objects is related to creational design patterns.

b. Facade Pattern.

**Answer:**

The facade pattern hides the complexities and provides an interface for the client through which the client can access the system, and also uses the Facade design pattern to help the client's applications efficiently and interact easily with the system, and this type of design pattern falls within structural design.

We have a set of classes and interfaces; how can they be arranged? Inheritance is not always effective or working and has problems. The idea of ​​structural design is how can I link classes or objects with each other in a correct and effective way.

Structural design patterns are those that are based on simplifying and facilitating the structures of objects by defining the relationships between them. structural design also explains how to combine objects and classes into larger structures.

c. Observer Pattern.

**Answer:**

It is a one-many relationships. You have one object that a group of people depends on. If the state of this dominant object changes, all the people associated with this object know that its state has changed. This object must when its state changes, it is required to alert all people, and this is the idea of ​​the observer pattern

Behavioural patterns are interested in developing solutions or providing them with regard to interaction or communication between objects, as well as how to rely on each other with others, and the relationship of behavioural patterns with the observer that when an object changes, which prompts other objects to take action or make some decisions in such cases, the observer pattern is It is the most suitable and it falls under behavioural patterns.

d. Decorator Pattern.

**Answer:**

The idea of the decorator pattern is that it inherits the properties of a specific class and then adds other procedures to it without changing anything in the basic class. So, it is a structural pattern that allows us to add additional functions to an object dynamically and is also used to change the functions of the object during operation, and other examples of the same class will not be affected, and since it is a pattern structural if falls under structural design patterns.

e. Factory Pattern.

**Answer:**

The Factory pattern is one of the most used patterns in Java and also this type of pattern comes under the creational pattern as this pattern provides one of the best ways to create an object.

It is from the categories of creational pattern that deal with the creation object.

Factory design pattern is a pattern that is used for a lot in software design and it adds a method that enables us to create an object without specifying the class we want to use meaning that it allows the factory method in the class to postpone the creation of an instance of one or more concrete subclasses and because these design patterns speak When creating an instance of an object, it falls under the category of creational design pattern.

f. Strategy Pattern.

**Answer:**

A strategy is a behavioural design pattern that transforms behaviours into objects and makes them interchangeable within the original context. It is used when we have multiple algorithms for a specific purpose and the client takes the actual implementation to use it in operation. In other words, you can define a set of algorithms and put each of them into A separate class and makes its objects interchangeable instead of executing a single algorithm and receives instructions during runtime directly regarding which of the set of algorithms should be used, so the behaviour of the class or its algorithm can be changed during runtime, and this type of design pattern comes under the design pattern behaviourist.

**Plagiarism**

Plagiarism is a particular form of cheating. Plagiarism must be avoided at all costs and students who break the rules, however innocently, may be penalised. It is your responsibility to ensure that you understand correct referencing practices. As a university level student, you are expected to use appropriate references throughout and keep carefully detailed notes of all your sources of materials for material you have used in your work, including any material downloaded from the Internet. Please consult the relevant unit lecturer or your course tutor if you need any further advice.

**Student Declaration**

|  |
| --- |
| **Student declaration**  I certify that the assignment submission is entirely my own work and I fully understand the consequences of plagiarism. I understand that making a false declaration is a form of malpractice.  Student signature: Rashed Hassan. Date: |