FUNDAMENTALS OF OBJECT ORIENTED PROGRAMMING

Lecture59

Lecture 59: Case Study - Employee Payroll System Java

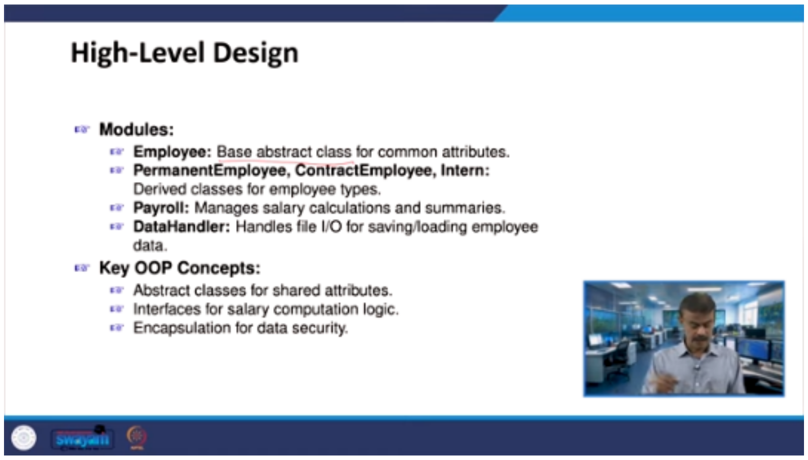
Welcome to lecture number 59. case studies in object-oriented programming. So, in this case study, what we can do in Java, I will talk about the employee payroll system, right. The main objective is to build a system to manage employee payrolls using OP in Java. So, the key features are employee hierarchy.

So, the employee may be permanent, may be contract, those who are doing internship, So, you call it as employee hierarchy. Then we have dynamic salary calculation based on employee type. So, because the salary will be differing from different people and there may be bonus for different hierarchy. So, in that case we are going to have the dynamic salary calculation.

So, for secure information handling we are going to use data encapsulation. So, we know that right the getter function and setter function suppose you are using from outside the class right there are private member data which cannot be accessed. So, we use data encapsulation. So, then for saving and loading employee data we use serialization. So, these are all the main key features.

And the use cases are we manage payrolls in organization of different sizes, right. So, we take a let us say for example, any university or any company, right. So, those organization where you will have different sizes, right. So, this we have to manage. And we have to simplify the salary calculations and the storage, storage of all the information about the employee.

So now we are going to have these modules in the high level design. We are going to have the modules like employee. So employee is a base abstract class, right, for common attributes. And in the derived class, you will have permanent employee, contract employee and intern. OK, so this will be in the derived class employee types.



High-Level Design

- **Modules:**
  - **Employee:** Base abstract class for common attributes.
  - **PermanentEmployee, ContractEmployee, Intern:** Derived classes for employee types.
  - **Payroll:** Manages salary calculations and summaries.
  - **DataHandler:** Handles file I/O for saving/loading employee data.
- **Key OOP Concepts:**
  - Abstract classes for shared attributes.
  - Interfaces for salary computation logic.
  - Encapsulation for data security.

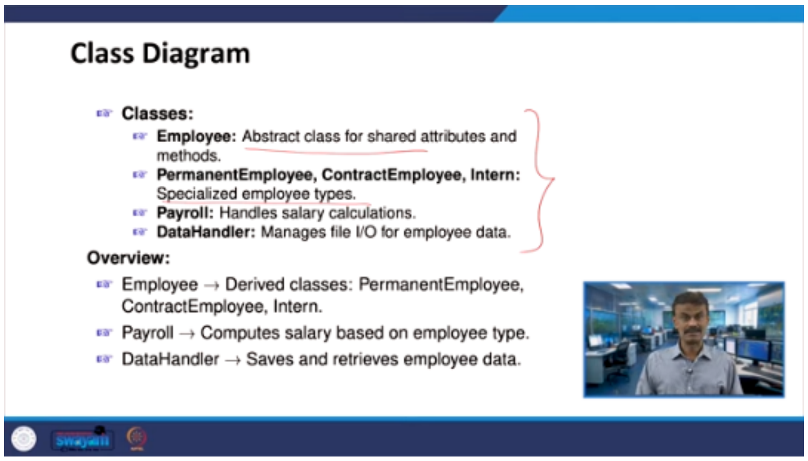
And the payroll, which is managing salary calculations and summaries, right? So, in the payroll, we will have salary calculations and summary. And the data handler, which handles input output file for saving or loading employee data. So, these are all the important modules that we are going to see. And what are the OOP concepts that we are going to use?

Abstract classes for shared attributes, interfaces, right? So, first time. In the case study, we are using interfaces. All right, it is 100 percent abstract—we can see over interfaces. So, the salary computation logic we use interface, and for the data security encapsulation. So, these concepts we use, all right, to solve the problem, right?

So, we are writing the code based on abstract classes, interfaces, and encapsulation. So, these are all the important classes. Right. So, what are all the classes? We have an Employee class.

The Employee class you have will be called the abstract class. Right. And Permanent Employee, Contract Employee, and Intern will be the derived classes. You call them specialized employee types. Payroll will be another class.

So, that is handling salary calculation. Data handler that is managing that input output file for employee data. So, as I said, employee is abstract class and permanent employee, contract employee, interns will be derived classes. And payroll, computing the salary based on the employee type. And data handler will save and retrieve the employee data.



Class Diagram

- **Classes:**
  - **Employee:** Abstract class for shared attributes and methods.
  - **PermanentEmployee, ContractEmployee, Intern:** Specialized employee types.
  - **Payroll:** Handles salary calculations.
  - **DataHandler:** Manages file I/O for employee data.

Overview:

- Employee → Derived classes: PermanentEmployee, ContractEmployee, Intern.
- Payroll → Computes salary based on employee type.
- DataHandler → Saves and retrieves employee data.

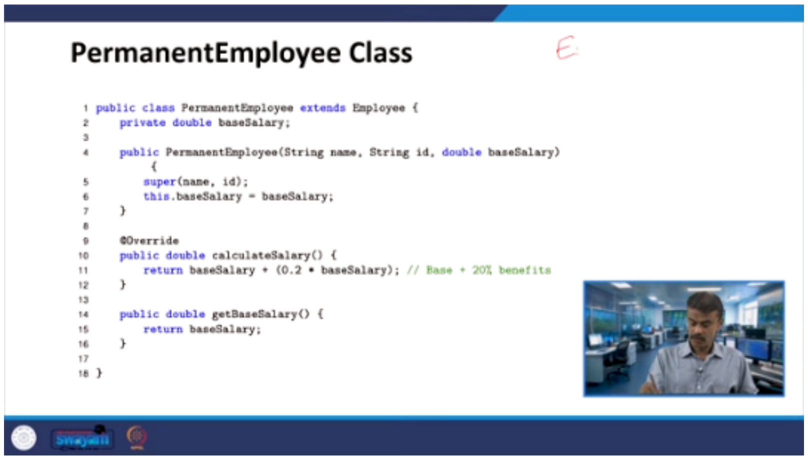
So, let us go to the employee abstract class. All right. So, here you have Java input output serializable. This is your importing. We have an abstract class called employee, right?

So this employee implements serializable, right? Inbuilt class, right? So this implements serializable. And you have a private member data name and a private member data ID whose data types are strings. So now you have constructor, right?

So under constructor, you have name and ID. This example we had seen several times. And name is assigned to this dot name and id is assigned to this dot id, right. So, you have a method called get name. Get name is returning name and get id is returning id, ok.

And here you have the abstract method called calculate salary, right. So, in the inner returns, further inner returns, this will be defined. So, this is your employee class. And now you have a permanent employee. So, which is the extending employee.

So, here you have employee, right. So, which is in fact, here the implement serializable, the serializable is the interface, right. Serializable is nothing but interface. This is interface. I will write it, okay.



PermanentEmployee Class

```java
public class PermanentEmployee extends Employee {
    private double baseSalary;

    public PermanentEmployee(String name, String id, double baseSalary)
        {
        super(name, id);
        this.baseSalary = baseSalary;
    }

    @Override
    public double calculateSalary() {
        return baseSalary + (0.2 * baseSalary); // Base + 20% benefits
    }

    public double getBaseSalary() {
        return baseSalary;
    }

}
```

And here, employee is the abstract class, and now the permanent employee is inheriting employee. So, I put PE. PE stands for permanent employee, OK. So, permanent employee is the class, and here you have base salary, right. So, base salary is a private member data whose data type is double, and here you have the constructor permanent employee.

So, here you have name, ID, and base salary, OK, right? The super method. Or super keyword, so you have name, comma, ID. So that means it will invoke the superclass. So what is your superclass? Employee is your superclass, the base class or superclass. So this dot name will be assigned name, and ID will be assigned to this dot ID. So we know the meaning of that. So now, all right, those two, name and ID were taken care of, and base salary will be assigned to this dot base salary, right. So, we have calculate salary in the superclass, right? That is, in fact, an abstract method, and here we are defining it in permanent employee, right. So, base salary plus 0.2, that means a 20 percent benefit: 0.2 into base salary. Fine.

And you have a get base salary. The get base salary will return base salary. That is the usual method in the permanent employee class. And then you have contract employee. So, contract employee is also extending employee.

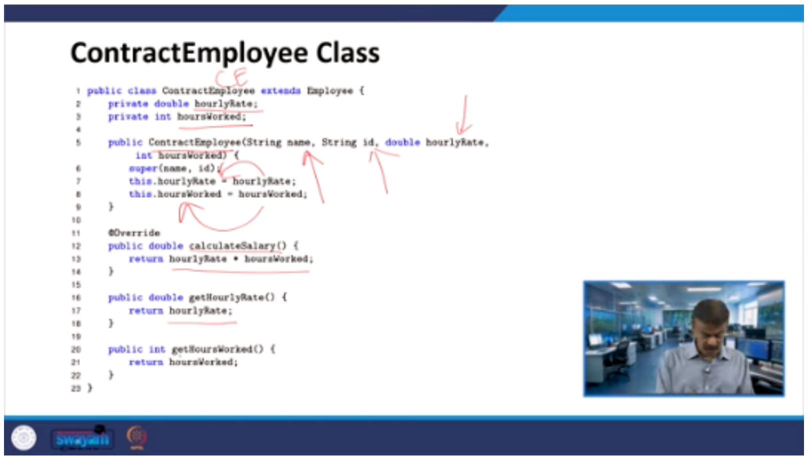
So we call it a CE, right? So here, you have CE. CE is also an extended employee. CE is also, CE stands for contract employee, right? So here you have two private member data: hourly rate.

Whose data type is double. And hours worked, whose data type is integer. And here, the contract employee is the constructor. So the constructor has name string, id string, and hourly rate, which is double. So four parameters.

Hours worked is another parameter. So here we call it an argument. This is a constructor. Right. And then the superclass constructor will be called in line number 6.

So, that means this dot name equals name, this dot id equals id; that is the meaning. And this dot hourly rate equals hourly rate; that means this will be stored here, and hours worked will be stored in this dot hours worked. Again, the calculate salary will be overridden, right, from the superclass, and hourly rate will be multiplied by hours worked, right. And get hourly rate. We will return hourly rate and get hours worked.

We will return hours worked, right? So, this is the contract employee class. And the last one, the intern, right? The intern class is extending employee. So, intern is also extending employee.



ContractEmployee Class

```java
1  public class ContractEmployee extends Employee {
2      private double hourlyRate;
3      private int hoursWorked;
4
5      public ContractEmployee(String name, String id, double hourlyRate,
           int hoursWorked) {
6          super(name, id);
7          this.hourlyRate = hourlyRate;
8          this.hoursWorked = hoursWorked;
9      }
10
11     @Override
12     public double calculateSalary() {
13         return hourlyRate * hoursWorked;
14     }
15
16     public double getHourlyRate() {
17         return hourlyRate;
18     }
19
20     public int getHoursWorked() {
21         return hoursWorked;
22     }
23 }
```

So, these three classes are inheriting or inheriting employee. That is the meaning, right? So, here you have private member data stipend under double. And here you have name, id, stipend, right? Three-argument constructor.

And here you have name and id, super. So, that means this dot name equal to name and this dot id equal to id. And line number 6, you have stipend is assigned to this dot stipend, okay? And calculate salary will be overridden, return stipend, right? It is a fixed stipend.

So, that is what we call it as a hierarchy. right, all the three are getting different salaries, that is why dynamic calculation. And then getStipend will return Stipend. So, this is the employee class and then we will have payroll class. So, the payroll class you have employees, right, which is array list, I am importing array list, right, whose class is array list, that means array list of employee and here you have

The constructor payroll, right? Constructor payroll, this object employees, all right? Dynamically allocated memory with the constructor array list, right? Here you have the method, all right? So, the method, we have an argument employee under class employee, small e object and capital E is the class, all right?

That argument and then you are adding employee. ArrayList you can add that means employee is invoking which is invoking add function by putting employee that means you put suppose there are employees E1, E2, E3, etc, EN right it is like this. So, you may have several whatever list you use it is fine. So, now here you have generate payroll method and the employee is assigned to EMP. So, this you call it as EMP.

So, the base class, I mean this is the base address, the Java, right, identity hash map of both employees and EMP are same, right. So, now you are printing, it is a range for loop, right. So, for all the cases it will print, right, name, get name, get ID, right, calculate salary, right. So, this will do. So, accordingly, right, so this will do, right.

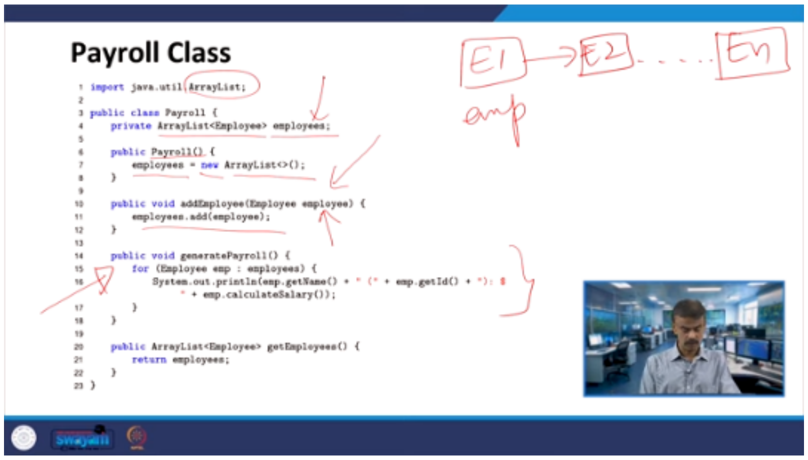
So, here you have maybe one example we can see get name, when EMP is invoking get name, All right. So EMP is under employees. Employees under ArrayList. All right.

So, get name. All right. So here you have get name, which will return name, and get ID will return ID. And then you will have calculate salary. All right.

So, accordingly. All right. EMP. So, EMP is under employee. All right.

So, EMP is under employee. All right. So, calculate salary. So, it will calculate the abstract method over here. And this will do accordingly, right, whether it is an intern, a contract employee, or a permanent employee.

So accordingly, this will calculate the salary, okay. So this is your printing: System.out.println. You are printing this. And then here, it will return. So here you are having getEmployees, right? ArrayList. So that is a getEmployees method.



Payroll Class

```java
import java.util.ArrayList;

public class Payroll {
    private ArrayList<Employee> employees;

    public Payroll() {
        employees = new ArrayList<>();
    }

    public void addEmployee(Employee employee) {
        employees.add(employee);
    }

    public void generatePayroll() {
        for (Employee emp : employees) {
            System.out.println(emp.getName() + " (" + emp.getId() + "): $
              " + emp.calculateSalary());
        }
    }

    public ArrayList<Employee> getEmployees() {
        return employees;
    }
}
```

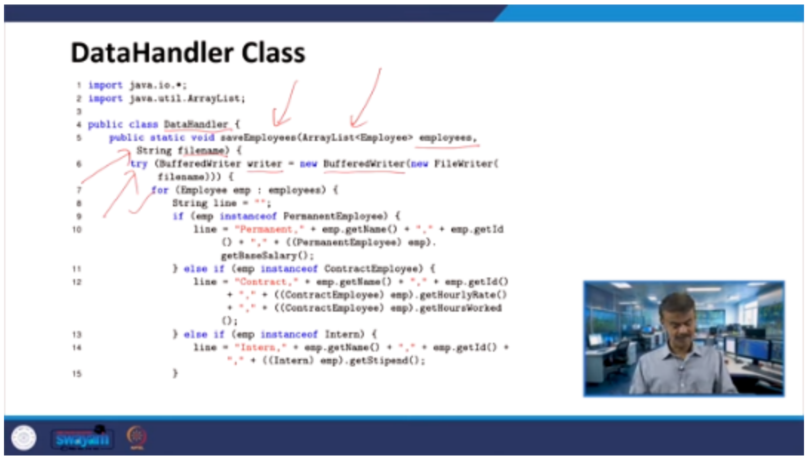
Whose return type is ArrayList. So this will return employees. Right. So this will return employees. So employees is of ArrayList.

OK. So this is your Payroll class. Right. So this is your Payroll class. So the next one is the DataHandler class.

So here we are importing all the utilities and ArrayList. The star indicates whatever utility you want it will be taken care all right input output with whatever utility and ArrayList utility. So, now let us have the class data handler all right. So, here you have a method called save employees. So, under save employees we have two arguments employees, employees is under ArrayList and file name under string ok file name under string.

So now we use try-throw-catch, right? So you are creating writer under buffered writer, right? Under buffered writer. Dynamically allocating a memory with buffered writer is the constructor, right? So under constructor, you have file writer by passing file name, right?

So here you are passing the file name. So now it's a range for loop, right? Now employees will be copied into EMP. So, that means both identity ashmaps are same. First you have line put an I or no character.



DataHandler Class

```
1  import java.io.*;
2  import java.util.ArrayList;
3
4  public class DataHandler {
5      public static void saveEmployees(ArrayList<Employee> employees,
           String filename) {
6          try (BufferedWriter writer = new BufferedWriter(new FileWriter(
               filename))) {
7              for (Employee emp : employees) {
8                  String line = "";
9                  if (emp instanceof PermanentEmployee) {
10                     line = "Permanent," + emp.getName() + "," + emp.getId
                           () + "," + ((PermanentEmployee) emp).
                           getBaseSalary();
11                 } else if (emp instanceof ContractEmployee) {
12                     line = "Contract," + emp.getName() + "," + emp.getId()
                           + "," + ((ContractEmployee) emp).getHourlyRate()
                           + "," + ((ContractEmployee) emp).getHoursWorked
                           ();
13                 } else if (emp instanceof Intern) {
14                     line = "Intern," + emp.getName() + "," + emp.getId() +
                           "," + ((Intern) emp).getStipend();
15                 }
```

If employee emp instance of permanent employee that means if employee is equal to permanent employee that equivalent statement. So, then permanent is assigned to line right permanent is assigned to line plus you get the name employee is invoking get name then you have comma. It is all printing and then EMP is invoking get ID, right? Again one comma. So, then you are typecasting.

So, this permanent employee of your EMP, okay? So, EMP will be now whatever, right? The role of permanent employee. So, that can take over EMP. It is like a typecasting.

You put int of A, right? Like this, right? So, this is now that EMP, right? It is like a permanent employee which is invoking get base salary, right. So, get base salary, right.

So, it is equivalent to permanent employee. I will just write PE, PE dot get base salary. So, whatever be the basic salary will that particular method will be invoked, right. So, that particular method will be invoked, right. So, this is what up to here we have done, right.

Else if this employee is contract employee. So, the line will be contract comma you are getting the name right employee get id and the contract employee dot get hourly rate that means it is a kind of typecasting all right and employee dot get hours worked all right. So, all these will be called and then you can see the line will contain contract let us say some name x. comma is our id and hourly rate and get hours worked so all will be one line so line is nothing but a string right so that we defined in line number eight else if employee is a intern all right in that case line will be intern emp dot get name you have comma and emp dot get id and intern is invoking get stryphon that means EMP is invoking get stryphon equal like this all right.

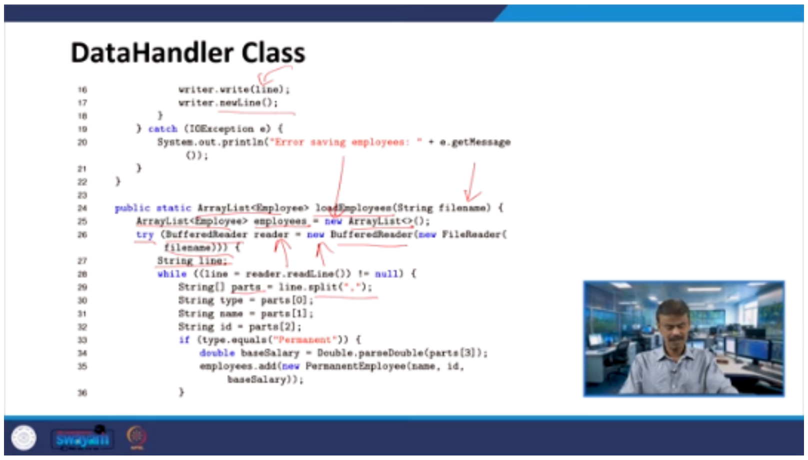
So, this is what up to line number 15. Now the writer object which is invoking the write function by passing line. So, now you have line. So, either one of the employees will be there right is invoking line all right and then writer will invoke new line all right. So, in case any exception that means assume that I am not able to open the file.

All right file writer you are passing the file name file name does not exist or file name we are not able to open right. So, in that case it will throw the exception right and it will be caught here the error saving employees. So, whatever be the message that we are passing. So, that will be caught ok. So, this is the error handling exception and the next method you have load employees.

So, the load employees the return type is ArrayList and you have file name you are passing the file name string. You have employees object under ArrayList right. So, dynamically allocating a memory with constructor ArrayList ok. So, now again you are trying right. So, reader try reader the object buffer reader the class the right hand side you are allocating a memory with the constructor and file reader will do the file name exactly like a last case.

So, line is string. So, now while line equal to reader dot read line as far as right. So, you are able to read the line right which is not equal to null right not at the end of the line or next line. So, now parts right string of parts right this is array all right. So, parts is an array whose data type is string.

So the right-hand side line is invoking split. So the line is invoking split. So whenever you find a comma. So this will split. So suppose I have the line.



DataHandler Class

```java
                writer.write(line);
                writer.newLine();
            }
        } catch (IOException e) {
            System.out.println("Error saving employees: " + e.getMessage
                ());
        }
    }

    public static ArrayList<Employee> loadEmployees(String filename) {
        ArrayList<Employee> employees = new ArrayList<>();
        try (BufferedReader reader = new BufferedReader(new FileReader(
            filename))) {
            String line;
            while ((line = reader.readLine()) != null) {
                String[] parts = line.split(",");
                String type = parts[0];
                String name = parts[1];
                String id = parts[2];
                if (type.equals("Permanent")) {
                    double baseSalary = Double.parseDouble(parts[3]);
                    employees.add(new PermanentEmployee(name, id,
                        baseSalary));
                }
```

A, B, C. I split into A, B, C. And then you are putting parts of 0. First one A. You put type. The comma will be removed. Parts 1, B, name, and C, ID. So you put it in an array.

Suppose I have A, B, C. I am just giving an example. It can be anything. A, B, C can be anything. In the line we had seen. Get ID, salary, etc.

So now part 0 is A, 0th index. First index is 1, parts of 1. And parts of 2 is C. That is the meaning. If type is invoking equals, let us say permanent. You may get permanent or contract or intern.

So suppose that is equal to permanent right so that means the first part right I mean to say parts of 0 you are assigned to type right if the object type under string or you call it as a variable type under string which invokes equals permanent right so that means if it is equal to permanent so then double is invoking parse double of parts of 3. So, parts of 3, what do you have? So, you will have the salary. Parts of 3 will be salary.

So, that salary will be assigned to base salary. So, parts of 3, so whatever be, let us say It is a string. So that string is converted into double. Pass double.

Right. So now base salary will be double. That is the meaning. Right. And you are adding the employee.

Right. So here you have permanent employee. The constructor permanent employee will be called. You are passing name, ID and base salary. So, that is what we are doing over here.

So, these we are doing for data persistence right. So, yes if the type dot equals let us say this is not permanent right suppose it is contract all right again we are doing. So, what we do the parts 3 right that will be converted into hourly rate right parts of 3 it is a string convert into double parts double will convert to double. That means the double when it is invoking this and hourly rate is now double. And here parse int, the integer parse int.

That means parse 4 is a string. The string will be converted into integer. That is hours worked. All right. So now employees is invoking add.

All right. So you are adding. You keep on adding the employees. All right. So, contract employee name id hourly rate and hours worked.

So, this particular constructor will be called and we had seen this right. Else if the type dot equals is intern right in that case parse 3. So, which is parse double. So, the stipend will be double. So, this will be assigned.

So, this is a string convert into double. So, double will be assigned to stipend right and then the constructor will be called. All right. That means employees is invoking at intern constructor will be caught. You will you are passing name, ID and style.

Right. So in case any problem, that means when I am right doing this line number 26. Right. I am not able to read the file or the file does file name does not exist. So then the exception will be caught here.

So error loading employees and whatever the message that message will be printed. All right. otherwise it will return employees so now go to the main program so main program right so this is a driver class and payroll is a object right so payroll is an object under payroll class all right so here you have the constructor payroll that is a default constructor so now i am adding the employees right add employee so here allies P 101 5000 is a permanent employee, right. So, name, ID and salary, permanent employee.

I will just go, we will just recall because we have seen so many, right. So, employee, permanent employee, name, ID, and base salary. So, name and ID, it will be, right, assigned from the superclass, right. That means this.name. Allies this dot id p 101, right? So these will be copied, right? So this we are passing, so permanent employee and next one is contract. You can put e1, right? So let us say e1 is allies, then we are adding payroll, right? We'll invoke add employee. So next one is a contract employee, bob c202.

Right. Twenty and hundred. Right. Contract employee. We'll just see.

All right. Name, ID, hourly rate, and hours worked. Right. So here, the salary calculation is different. Right.

Hourly rate dot hours worked. All right. So Bob C20 to twenty hundred. So four parameters we are passing. And the next one is add employee.

Payroll is invoking add employee. So here it is an intern. Charlie intern I 303 and 1500 ok. So, it is hourly 100 and here it is fixed 1500 right. So, in the case of intern in the case of intern we have name ID and stipend name ID and stipend.

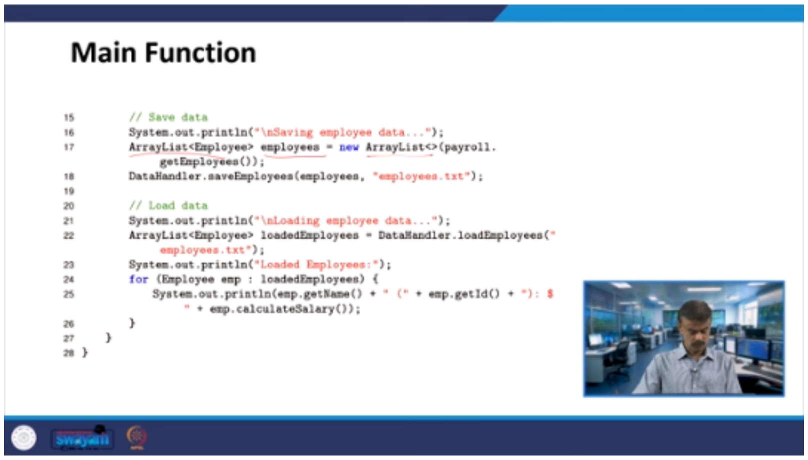
So, here you go the name is Charlie ID is 303 and stipend is 1500. right. So, now system dot out dot printer then this I am printing payroll summary and payroll is invoking generate payroll right. So, payroll is invoking generate payroll. So, here you go payroll class which is invoking generate payroll right under payroll right.

So, here you have the object payroll under payroll right. So, we see in the payroll class right. So, you will have this particular method right. So, we see the payroll class under payroll class we have generate payroll right so whatever the employees three employees we have used right so accordingly right so accordingly you have get name get id and the calculate salary right so here you have three different employees one is a permanent employee that means the permanent employee

Constructor will be called, another contract employee that particular constructor will be called intern, right. So, when you are calculating a salary that particular method. So, accordingly, right. So, you have three subclasses, right. Employee is a superclass, you have permanent subclass, contract subclass and for the intern you have the subclass, right.

So, this will call and then generate the payroll, right. This is not very difficult. and saving the employee data. So, you have employees object under ArrayList right. So, this is a constructor ArrayList constructor pausing payroll and get employees right.

So, here we have payroll and get employees. So, these are all the employees that you have been added and the payroll is invoking get employees means. So, here you have in the payroll class right. So, get employee So that will return employees.



Main Function

```
15        // Save data
16        System.out.println("\nSaving employee data...");
17        ArrayList<Employee> employees = new ArrayList<>(payroll.
              getEmployees());
18        DataHandler.saveEmployees(employees, "employees.txt");
19
20        // Load data
21        System.out.println("\nLoading employee data...");
22        ArrayList<Employee> loadedEmployees = DataHandler.loadEmployees("
              employees.txt");
23        System.out.println("Loaded Employees:");
24        for (Employee emp : loadedEmployees) {
25            System.out.println(emp.getName() + " (" + emp.getId() + "): $
                  " + emp.calculateSalary());
26        }
27    }
28 }
```

All right. So this will return employees. So there are three employees. The names will be returned. All right.

So here the payroll dot employees. This you are passing in fact. All right. Payroll dot get employees. So this you are passing.

So, all the employees list have been passed right and here data handler this particular object which is invoking save employees right. So, you are pausing employees and then a file name right. So, you are pausing employees this is an object and here you have the constructor. So, this is invoking gate employees. So, whatever be the employees let us say E1, E2, E3.

And then save employees. So all the employees will be saved in the text file, a text file called employees.txt. So line number 21, your system.out.println. So this will be printed. Loading employee data will be printed.

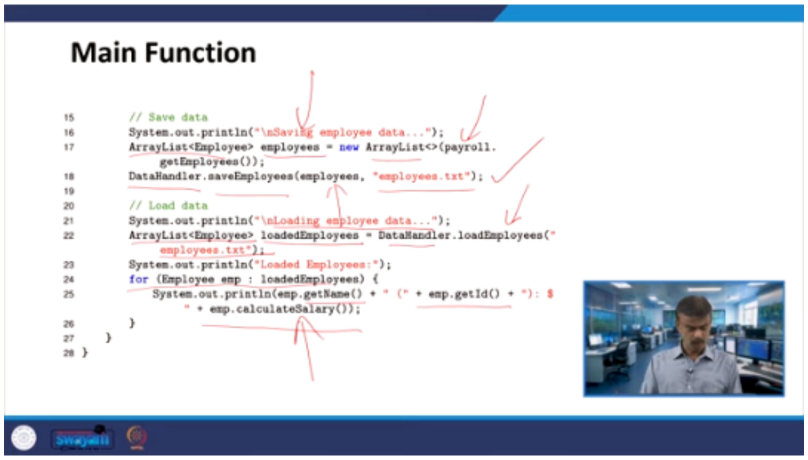
And here you have loaded employees, which is the object. under the class array list of employee right and in the right hand side correct so right hand side you have the data handler is invoking load employees all right and then you are pausing employees dot txt right you are pausing employees dot txt so that you have created in line number 18 all right so then you are printing loaded employees and this is the range for loop so the loaded employees and emp right so they are pointing to the same ashmap And you are printing get name. Right. And then here you have get ID.

Right. So, get the ID and calculate the salary. Right. So, according to the employees. So, this particular constructor, anyway, you have.

Right. So, based on this constructor, and since this is the overridden function or overridden method. So, the calculate salary will be computed accordingly. Right. Which constructor are you going to?

So, the permanent employee constructor you go to. So, there you will have the calculate salary. So, that will be calculated, and similarly for the contract employee and similarly for the intern, right. So, this will be calculated, right. So, this is about your driver class, payroll system demo.

In fact, it is a main method, right. So, now we have seen, right. So, you have seen so many classes and so many methods and this is the main method and when you go through right line by line and in fact we will also work it out when you run the code right so you have to get the output like this all right so all the calculation all right so what about the salary all right the get salary will be called accordingly you can check So whatever you are passing, that particular get salary method will be called overridden method.



Main Function

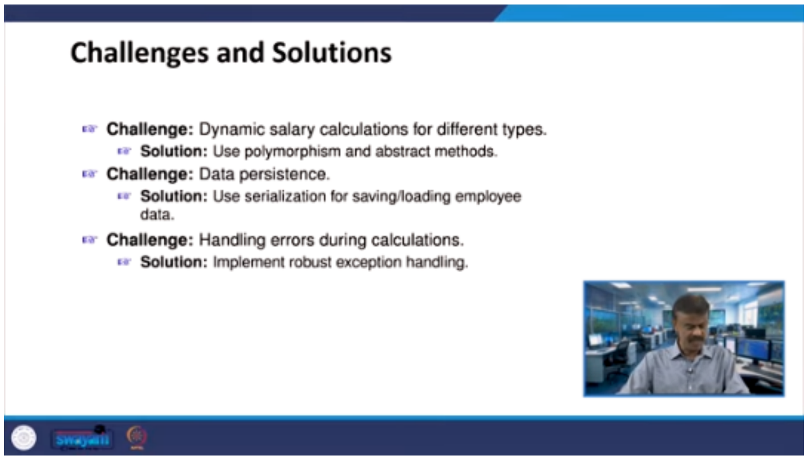
```
        // Save data
        System.out.println("\nSaving employee data...");
        ArrayList<Employee> employees = new ArrayList<>(payroll.
            getEmployees());
        DataHandler.saveEmployees(employees, "employees.txt");

        // Load data
        System.out.println("\nLoading employee data...");
        ArrayList<Employee> loadedEmployees = DataHandler.loadEmployees("
            employees.txt");
        System.out.println("Loaded Employees:");
        for (Employee emp : loadedEmployees) {
            System.out.println(emp.getName() + " (" + emp.getId() + "): $
                " + emp.calculateSalary());
        }
    }
}
```

So accordingly, the salary will be printed. Fine. So this is what we will get. Now what we can do, we will run the code in the Java compiler. I will go to Java.

So the same code we have written. as payroll system demo right. So, when we run the code we will get the output like this ok. These are the output we are getting ok. So, now we will see the text file right what it is being written employees dot txt right.

So, this is what employees.txt is. This is what we are getting, OK. So, the allies. ID, all right, all the information we have. So, this is what is being written in the text file. So, we have seen the complete case study.

In fact, we have seen two case studies, right, using Java. So, now, in this problem, what are all the challenges you are facing, all right? So, the challenge is dynamic salary calculation, right. So, we have seen three different types of employees: permanent, contract, and intern, right. So, how to do the dynamic salary calculation?



Challenges and Solutions

- **Challenge:** Dynamic salary calculations for different types.
  - **Solution:** Use polymorphism and abstract methods.
- **Challenge:** Data persistence.
  - **Solution:** Use serialization for saving/loading employee data.
- **Challenge:** Handling errors during calculations.
  - **Solution:** Implement robust exception handling.

Right, so for this solution, what we had given was we use polymorphism, right? The overriding and then abstract methods. And the second challenge is data persistence. So, for the data persistence, right, we use serialization for saving or loading employee data. So, this I mentioned, right? I talked about the data persistence, and we have used so many times the try-throw-catch. So, this will handle the exception. So, the solution we had given, we have implemented this with robust exception handling. So, in this case study, we have developed a payroll system using object-oriented principles. So, what you can do, what my advice is, you try to extend this code.

So, put more employee category. And also, right so when you are overriding right so how to change the or how to calculate the salary for different people and also right so whatever be the problems that we face the challenges that we face what you can do you can further extend right so you can further work it on a data persistence and also and still you can elegantly handle the errors with the exceptions so what my advice is try to extend right or you can see the small companies what are all the other things you left out in this particular problem. So, you can add it right.

So, we have used abstract classes and polymorphism for salary calculations and here you had seen serialization concept for loading and saving the employee data and the we have used the error handling exceptions the try through catch we have used for if the file is invalid or file itself is a input wrong or the file does not exist. So, in this case, so we have used the robust error handling exception, right. And also we had a modular design, right. So, that enables scalability for future enhancements, right.

So, these are all the concepts. So, in fact, we have implemented this and we have seen the problem of payroll system, right. So, this we have developed using The concepts that you have studied. In object oriented programming.

So with this, I am concluding this particular lecture. Thank you.