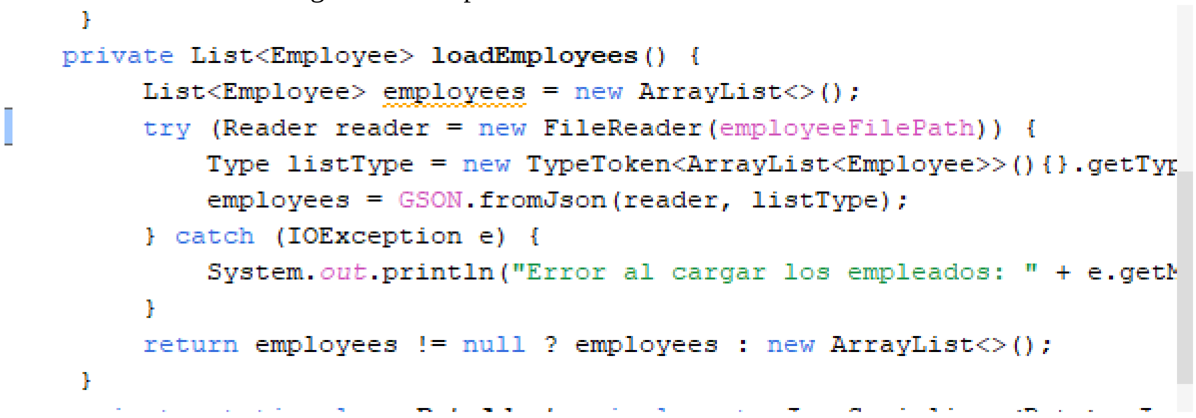
HW21-Find pitfalls in the Project

Long Methods:

Location: loadEmployees() method in EmployeeManager

Problem: Method too long Solution: Split into smaller methods:

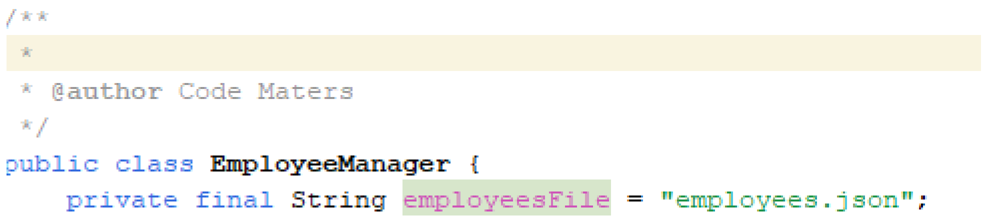


```
    }
   private List<Employee> loadEmployees() {
        List<Employee> employees = new ArrayList<>();
        try (Reader reader = new FileReader(employeeFilePath)) {
            Type listType = new TypeToken<ArrayList<Employee>>(){}.getTyp
            employees = GSON.fromJson(reader, listType);
        } catch (IOException e) {
            System.out.println("Error al cargar los empleados: " + e.getM
        }
        return employees != null ? employees : new ArrayList<>();
    }
```

Type embedded in names:

Problem:

In some places like EmployeeFile, it might be better to change it to EmployeeFilePath to avoid having the type (file) in the name.

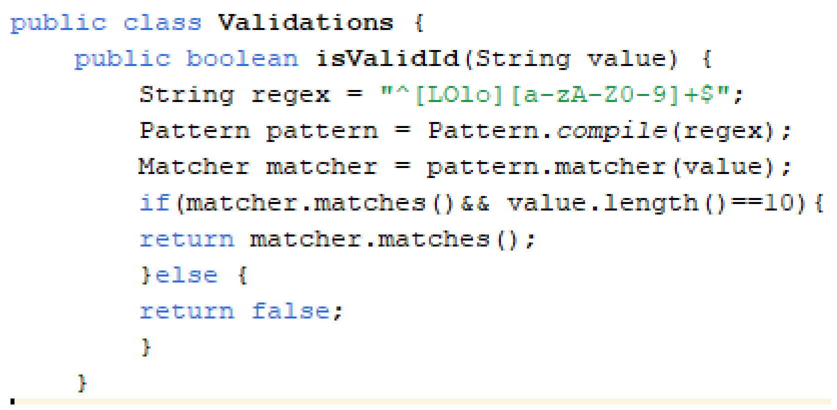


```
/**
 *
 * @author Code Maters
 */
public class EmployeeManager {
    private final String employeesFile = "employees.json";
```

Oddball Solution:

Problem:

In the Validations class, the isValidId method returns matcher.matches() twice within an if that adds no actual value. This logic is unnecessarily confusing.



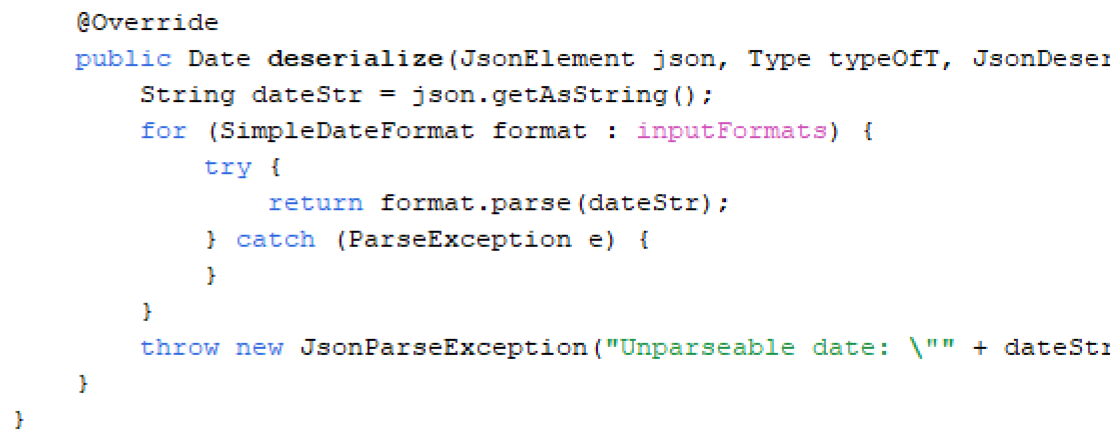
```
public class Validations {
    public boolean isValidId(String value) {
        String regex = "^[LOlo][a-zA-Z0-9]+$";
        Pattern pattern = Pattern.compile(regex);
        Matcher matcher = pattern.matcher(value);
        if(matcher.matches()&& value.length()==10){
        return matcher.matches();
        }else {
        return false;
        }
    }
```

Conditional Complexity:

Location: Deserialize method in DateAdapter

Problem: Complex conditional structure

Solution: Use a stream to simplify:



```
    @Override
    public Date deserialize(JsonElement json, Type typeOfT, JsonDeser
        String dateStr = json.getAsString();
        for (SimpleDateFormat format : inputFormats) {
            try {
                return format.parse(dateStr);
            } catch (ParseException e) {
            }
        }
        throw new JsonParseException("Unparseable date: \"" + dateStr
    }
}
```

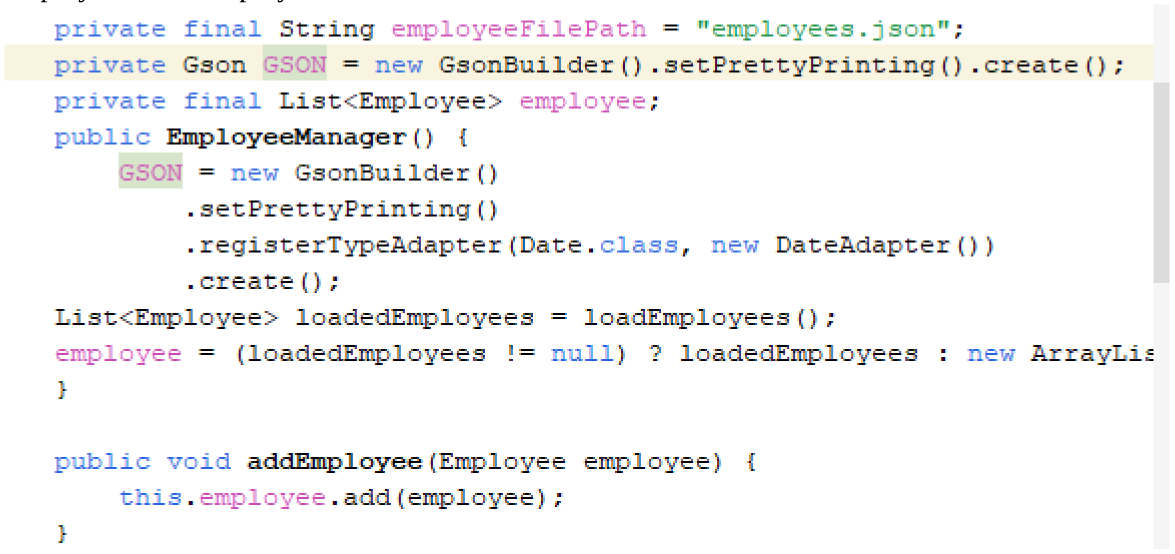
Non-descriptive variable names:

Explanation:

Variables like GSON, employees, aIessPercentage, and aOvertimeHourIncrease are not descriptive enough.

Possible solution:

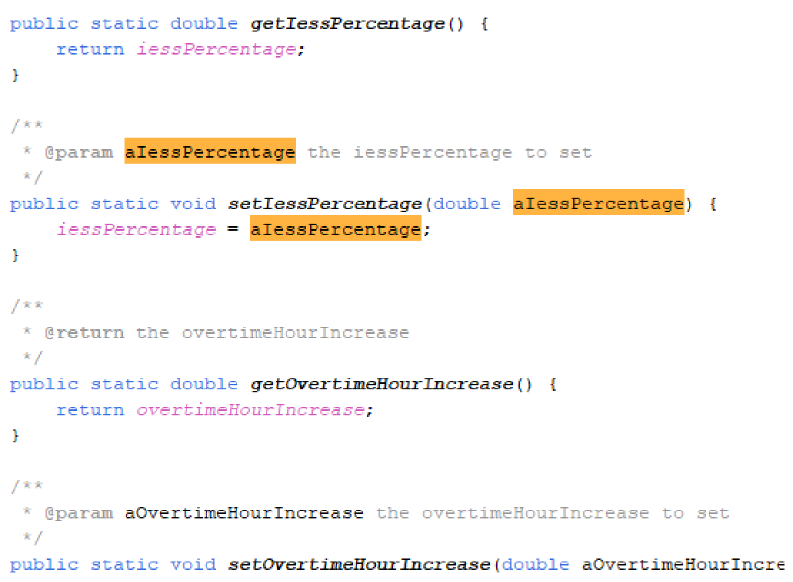
They should have clearer names that indicate their exact purpose, such as gsonInstance or employeeList or employees.



```
private final String employeeFilePath = "employees.json";
private Gson GSON = new GsonBuilder().setPrettyPrinting().create();
private final List<Employee> employee;
public EmployeeManager() {
    GSON = new GsonBuilder()
        .setPrettyPrinting()
        .registerTypeAdapter(Date.class, new DateAdapter())
        .create();
List<Employee> loadedEmployees = loadEmployees();
employee = (loadedEmployees != null) ? loadedEmployees : new ArrayLis
}

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

Second;



```
public static double getIessPercentage() {
    return iessPercentage;
}

/**
 * @param aIessPercentage the iessPercentage to set
 */
public static void setIessPercentage(double aIessPercentage) {
    iessPercentage = aIessPercentage;
}

/**
 * @return the overtimeHourIncrease
 */
public static double getOvertimeHourIncrease() {
    return overtimeHourIncrease;
}

/**
 * @param aOvertimeHourIncrease the overtimeHourIncrease to set
 */
public static void setOvertimeHourIncrease(double aOvertimeHourIncre
```

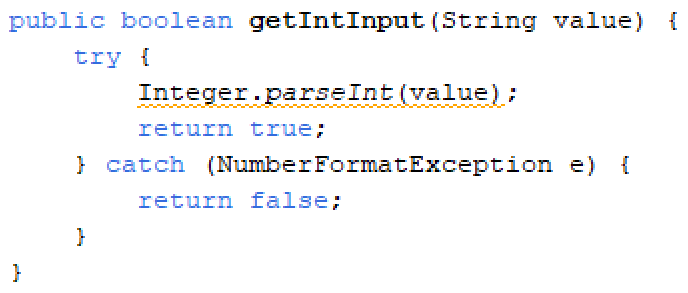
Dead Code:

Explanation:

Use useless code that is not used in the same code

Possible solution:

Remove getOIntInput function as it has no functionality



```
public boolean getIntInput(String value) {
    try {
        Integer.parseInt(value);
        return true;
    } catch (NumberFormatException e) {
        return false;
    }
}
```