ESCUELA SUPERIOR POLITÉCNICA DEL LITORAL

Diseño de software

Taller Procedimientos Almacenados

INTEGRANTES:

Justin Soto

Geovanny Jimenez

Danna Urdánigo

PAO II 2025- 2026

Técnicas de refactorización

Simplifying Method Calls:

• Rename method

En la clase **Medico** el nombre **realizaConsul** está cortado y no es el ideal. realizarConsulta comunica claramente la intención.

Error:

```
public void realizaConsul(Paciente paciente, HistorialMedico medico, SistemaAtencionMedico sistemaAtencionMedico) {
    String tratamiento;
    System.out.println("Realizando consulta a " + paciente.getNombre() +
            " (Solo se usa el objeto persona en este metodo...)");
    tratamiento = "Conservador...";
    prescribirTratamiento(paciente, tratamiento);
}
```

Solución:

```
public void realizaConsulta(Paciente paciente) {
    String tratamiento;
    System.out.println("Realizando consulta a " + paciente.getNombre() +
            " (Solo se usa el objeto persona en este metodo...)");
    tratamiento = "Conservador...";
    prescribirTratamiento(paciente, tratamiento);
}
```

• Remove parameter

En la clase **Medico** el método recibe **HistorialMedico** y **SistemaAtencionMedico** pero no los usa. El código se ensucia con datos innecesarios.

Error:

```
public void realizaConsul(Paciente paciente, HistorialMedico medico, SistemaAtencionMedico sistemaAtencionMedico) {
    String tratamiento;
    System.out.println("Realizando consulta a " + paciente.getNombre() +
            " (Solo se usa el objeto persona en este metodo...)");
    tratamiento = "Conservador...";
    prescribirTratamiento(paciente, tratamiento);
}
```

Solución:

```
public void realizaConsulta(Paciente paciente) {
    String tratamiento;
    System.out.println("Realizando consulta a " + paciente.getNombre() +
            " (Solo se usa el objeto persona en este metodo...)");
    tratamiento = "Conservador...";
    prescribirTratamiento(paciente, tratamiento);
}
```

• Preserve whole object

• Parameteriza method

En la clase SistemaAtencionMedico los métodos **obtenerPaciente** y **obtenerMedico** son idénticos en lógica ya que recorren una lista y comparan el nombre. Se está repitiendo código. La técnica consiste en crear un solo método que reciba la lista como parámetro.

Error:

```java
public Paciente obtenerPaciente(String nombrePaciente) {
    for(Paciente paciente : pacientes){
        if (paciente.getNombre().equals(nombrePaciente))
            return paciente;
    }
    return null;
}
```

```java
public Medico obtenerMedico(String nombreMedico) {
    for(Medico medico : medicos){
        if (medico.getNombre().equals(nombreMedico))
            return medico;
    }
    return null;
}
```

Solución:

```java
public Paciente obtenerPaciente(String nombrePaciente) {
    return (Paciente) obtenerPersona(this.pacientes, nombrePaciente);
}

public ServicioMedico obtenerServicioMedico(String nombreServicio) {
    for(ServicioMedico servicioMedico : serviciosMedicos){
        if (servicioMedico.getNombre().equals(nombreServicio))
            return servicioMedico;
    }
    return null;
}

public Medico obtenerMedico(String nombreMedico) {
    return (Medico) obtenerPersona(this.medicos, nombreMedico);
}

public Persona obtenerPersona(List<? extends Persona> listaPersonas, String nombre) {
    for(Persona p : listaPersonas){
        if (p.getNombre().equals(nombre))
            return p;
    }
    return null;
}
```

• Hide method

En la clase **Medico** el método **prescribirTratamiento** es público, pero el comentario dice explícitamente que nadie más debe usarlo. Debe ser privado.

Error:

```
public void prescribirTratamiento(Paciente paciente, String tratamiento) {
    // Implementación para prescribir un tratamiento al paciente.
    System.out.println("Prescribiendo tratamineto... " +
            "(Este metodo solo debe ser utilizado en esta clase, nadie mas lo puede utilizar)");
}
```

Solución:

```
private void prescribirTratamiento(Paciente paciente, String tratamiento) {
    // Implementación para prescribir un tratamiento al paciente.
    System.out.println("Prescribiendo tratamineto... " +
            "(Este metodo solo debe ser utilizado en esta clase, nadie mas lo puede utilizar)");
}
```

• Self encapsulate field

En la clase **ServicioMedico** en el **constructor** se está asignando this.costo = costo directamente. El comentario en el código nos dice que la validación (que el costo no sea negativo) está en el setter. Debemos usar el setter dentro del constructor para aprovechar esa validación.

Error:

```
public ServicioMedico(String nombre, String descripcion, double costo, int duracion) {
    this.nombre = nombre;
    this.descripcion = descripcion;
    // Ojo que las dos asignaciones de abajo deben de tener logica de validacion en el setter, esos valores no pueden ser negativos
    this.costo = costo;
    this.duracion = duracion;
}
```

Solución:

```
public ServicioMedico(String nombre, String descripcion, double costo, int duracion) {
    this.nombre = nombre;
    this.descripcion = descripcion;
// CORRECCIÓN: Usamos los setters en lugar de asignación directa
// Esto asegura que la validación (if costo < 0) se ejecute al crear el objeto
    setCosto(costo);
    setDuracion(duracion);
    }

public void setCosto(double costo) {
    if(costo < 0){
        System.out.println(x: "El costo no puede ser menor a 0");
        return;
    }
    this.costo = costo;
    }
```

• Replace data value with object

En la clase **Consulta** tenemos los atributos día, mes, año y hora. Teniendo en cuenta que existe la clase LocalDateTime la cual agrupa los datos y facilita operaciones de fecha.

Error:

```
public class Consulta {
    public int dia;
    public int mes;
    public int año;
    private String hora;
    private Paciente paciente;
    private Medico medico;
    private ServicioMedico servicioMedico;
    private boolean realizada;
    private String diagnostico;
    private String tratamiento;
    private List<String> examenesMedicos;

    public Consulta(int dia, int mes, int año, String hora, Paciente paciente, Medico medico, ServicioMedico servicioMedico, String diagnostico, String tratamiento, List<String> examenesMedicos)
        this.dia = dia;
        this.mes = mes;
        this.año = año;
        this.hora = hora;
        this.servicioMedico = servicioMedico;
        this.paciente = paciente;
        this.medico = medico;
        this.realizada = false;
    }
```

Solución:

```
public class Consulta {
    // reemplazo de los atributo dia, mes, año y hora
    private LocalDateTime fechaHora;
    private Paciente paciente;
    private Medico medico;
    private ServicioMedico servicioMedico;
    private boolean realizada;
    private String diagnostico;
    private String tratamiento;
    private List<String> examenesMedicos;


    // Constructor actualizado
    public Consulta(LocalDateTime fechaHora, Paciente paciente, Medico medico, ServicioMedico servicioMedico, String diagnostico, String tratamiento, List<String> examenesMedicos) {
        this.fechaHora = fechaHora; // Asignación limpia
        this.paciente = paciente;
        this.medico = medico;
        this.servicioMedico = servicioMedico;
        this.realizada = false;

    }

    // Getter y Setter del nuevo objeto
    public LocalDateTime getFechaHora() {
        return fechaHora;
    }

    public void setFechaHora(LocalDateTime fechaHora) {
        this.fechaHora = fechaHora;
    }
```

• Replace magic number with symbolic constant

En la clase **SistemaAtencionMedico** en el método **calcularValorFinalConsulta** el número 0.25 aparece en el cálculo. Si el descuento cambia mañana por ejemplo, es difícil de encontrar y actualizar. Una constante con nombre explica el significado.

Error:

```
public double calcularValorFinalConsulta(double costoConsulta, int edadPaciente){
    double valorARestar = 0;
    if(edadPaciente>=65){
        valorARestar = costoConsulta*0.25; //0.25 es el descuento para adultos mayores
    }
    return costoConsulta-valorARestar;
}
```

Solución:

```
private static final double PORCENTAJE_DESCUENTO_ADULTO_MAYOR = 0.25;
```

```java
public double calcularValorFinalConsulta(double costoConsulta, int edadPaciente){
    double valorARestar = 0;
    if(edadPaciente >= 65){
        valorARestar = costoConsulta * PORCENTAJE_DESCUENTO_ADULTO_MAYOR;
    }
    return costoConsulta - valorARestar;
    }
```

• Encapsulate field

En la clase **Paciente** el atributo **historialMedico** es public. Esto rompe el encapsulamiento porque cualquiera puede modificarlo directamente. Debe ser privado con acceso controlado (getters/setters).

Error:

```java
public class Paciente extends Persona {
    public HistorialMedico historialMedico;
```

Solución:

```java
public class Paciente extends Persona {
    private HistorialMedico historialMedico;
    public Paciente(String nombre, int edad, String genero, String direccion, String telefono, String correoElectronico) {
        super(nombre, edad, genero, direccion, telefono, correoElectronico);
        this.historialMedico = new HistorialMedico();
    }

    public HistorialMedico getHistorialMedico() {
        return historialMedico;
    }

    public void setHistorialMedico(HistorialMedico historialMedico) {
        this.historialMedico = historialMedico;
    }
```

• Encapsulate collection

En la clase **SistemaAtencionMedico** cuando se añade una consulta a la lista a la fuerza (getConsultas().add(...)). Al devolver la lista cruda (List<Consulta>), se rompe el encapsulamiento porque cualquiera puede borrar o limpiar el historial desde fuera. Se debe proteger devolviendo una copia de solo lectura y obligando a usar un método agregar controlado.

Error:

```
public void agendarConsulta(Paciente paciente, Consulta consulta){
    double costoConsulta = consulta.getServicioMedico().getCosto();
    int edadPaciente = paciente.getEdad();
    costoConsulta = calcularValorFinalConsulta(costoConsulta,edadPaciente);
    System.out.println("Se han cobrado "+ costoConsulta+ " dolares de su tarjeta de credito");
    paciente.historialMedico.getConsultas().add(consulta); //Hacer esto es incorrecto
}
```

Solución: