



Instituto Tecnológico de Costa Rica

Escuela de Ingeniería en Computación

Programación Orientada a Objetos

Grupo 02

Proyecto: Parques Callejeros

Juan Pablo Cambronero Alpízar

Ximena Molina

Contenido:

Definición del proyecto..... 3

Temas Investigados..... 4

- Detallar las funciones de depuración ofrecidas por el IDE que está usando.
Poner ejemplos de cada funcionalidad.
- Uso de JavaDoc.
- Envío de Correos.
- Uso de archivos.
- Emitir información por medio de archivos tipo PDF.

Solución 12

Conclusiones del trabajo 15

Bibliografía 19

Lista de revisión del proyecto..... 20

Definición del proyecto:

Esta aplicación permite administrar y usar espacios de parqueo que están habilitados sobre las calles.

La aplicación tendrá tres tipos de usuarios, cada uno con sus propias funciones:

- Los administradores de los parqueos
- Los usuarios
- Los inspectores

Los usuarios de estos parqueos podrán reservar espacios de parqueo, mientras que los administradores e inspectores se encargan del control de los mismos. Para mantener la persistencia de la información la aplicación la guarda en archivos.

Temas Investigados:

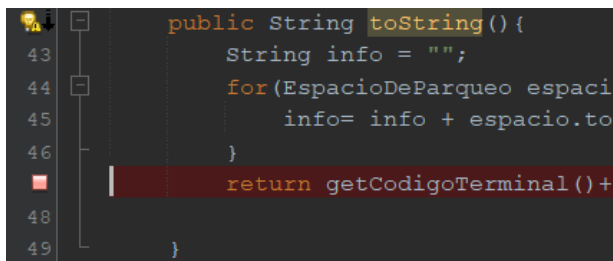
- Detallar las funciones de depuración ofrecidas por el IDE que está usando.

Para este proyecto se utilizó Apache NetBeans como entorno de desarrollo.

NetBeans 22 ofrece una amplia gama de herramientas de depuración que facilitan la identificación y corrección de errores en el código. A continuación, se profundizará en algunas de ellas.

Puntos de Interrupción (Breakpoints):

Los breakpoints permiten pausar la ejecución del programa en una línea específica para examinar el estado del programa. Esto es útil para verificar si el flujo de ejecución es el esperado y para inspeccionar variables en ese punto.



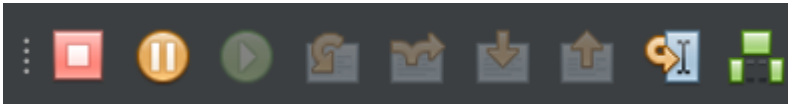
Para colocar un breakpoint solo basta con irse al margen izquierdo del código y hacer click en la línea que se desea el breakpoint. Se representa con un cuadrado rojo.

Depuración Paso a Paso:

La depuración paso a paso permite ejecutar el programa línea por línea, lo que es útil para analizar cómo se comporta el código en detalle.

- **Step Over:** Ejecuta la siguiente línea sin entrar en funciones llamadas en esa línea.
- **Step Into:** Entra en la función que se llama en la línea actual para depurarla.
- **Step Out:** Sale de la función actual y vuelve al contexto donde se llamó.

Una vez activado el debugger, en la barra superior tendrás las opciones mencionadas en el siguiente formato.



Inspección de Variables:

NetBeans te permite ver el valor de las variables en cualquier momento durante la depuración. Esto te ayuda a entender si las variables están tomando los valores esperados.

Esto se puede hacer cuando el programa se detiene en un breakpoint y se pasa el cursor sobre las variables en el código o revisarlas en la pestaña variables en la ventana de depuración.

Evaluación de Expresiones (Watches):

Se pueden agregar expresiones personalizadas que se evalúan en tiempo real durante la depuración. Esto es útil cuando quieres monitorear el valor de una expresión compleja o un objeto a lo largo del tiempo.

Durante la depuración, hacer clic derecho en una variable y seleccionar "New Watch" para agregar una expresión personalizada que se evaluará mientras el programa está detenido.

Monitoreo de Hilos (Threads):

NetBeans permite ver y controlar los diferentes hilos en ejecución dentro del programa. Esto es particularmente útil cuando depuras aplicaciones multihilo.

Acceder a la pestaña "Threads" en la ventana de depuración para ver todos los hilos activos. Puedes pausar y reanudar hilos específicos para entender cómo interactúan entre sí.

Condiciones en Breakpoints:

Haz clic derecho en un breakpoint y selecciona "Breakpoint Properties".

Establecer una condición (por ejemplo, `x == 10`) para que el programa solo se detenga cuando se cumpla esa condición.

Depuración Remota:

Si tu aplicación se ejecuta en un servidor o en otro entorno, NetBeans ofrece la posibilidad de conectar su depurador de manera remota para examinar el código.

Configurar una sesión de "Debug Remoto" desde el menú de depuración y especificar los parámetros de conexión remota para tu servidor o máquina objetivo.

Depuración de Interfaz Gráfica (GUI Debugging):

NetBeans ofrece herramientas para depurar aplicaciones con interfaces gráficas (Swing, JavaFX) y ver cómo se comportan los elementos visuales en tiempo de ejecución.

Usar la herramienta "Swing Debugger" para inspeccionar los componentes gráficos y sus propiedades mientras tu aplicación está en ejecución.

-Uso de JavaDoc:

Javadoc es una herramienta que forma parte del JDK (Java Development Kit) y permite generar documentación API directamente a partir de comentarios en el código fuente Java. Esta documentación se presenta en formato HTML, lo que facilita su consulta para desarrolladores y usuarios de una API o biblioteca.

La principal ventaja de Javadoc es que permite mantener la documentación y el código en un mismo lugar, reduciendo el riesgo de inconsistencias entre el comportamiento real del código y lo que la documentación describe. Los comentarios Javadoc utilizan una sintaxis especial para proporcionar descripciones detalladas de clases, métodos, atributos y otros elementos del **código**.

Ventajas de usar Javadoc:

1. Facilita la documentación: Permite que la documentación esté integrada con el código y sea accesible tanto para desarrolladores como para usuarios de la API.
2. Estándar en Java: Javadoc se ha convertido en un estándar para la creación de documentación en proyectos Java.
3. Generación automática: A partir de los comentarios en el código, Javadoc genera una documentación clara, con hipervínculos que permiten navegar entre clases y métodos.

-Envío de Correos:

El envío de correos electrónicos es una funcionalidad común en muchas aplicaciones modernas, como notificaciones automáticas, confirmaciones de registro o restablecimiento de contraseñas. En Java, esta funcionalidad se puede implementar utilizando la API de JavaMail, que proporciona un conjunto de clases y métodos para enviar y recibir correos electrónicos a través de protocolos estándar como SMTP, IMAP y POP3.

JavaMail simplifica el proceso de envío de correos electrónicos al proporcionar una abstracción de los detalles técnicos del protocolo de envío de correo. La API maneja el protocolo subyacente y permite a los desarrolladores enfocarse en el contenido del mensaje, sus destinatarios y otras configuraciones básicas.

Componentes de la API JavaMail

1. **Session:** Es la clase que representa una sesión de correo, que se utiliza para configurar las propiedades necesarias para la conexión (como el servidor SMTP, el puerto, la autenticación, etc.).

2. **Message:** Esta clase representa el mensaje de correo electrónico que será enviado. Se utiliza para definir el destinatario, el asunto, el cuerpo del mensaje, y otras configuraciones del correo.
3. **Transport:** Es la clase encargada de enviar el correo a través del servidor SMTP configurado.
4. **Authenticator:** Se utiliza para autenticar al usuario si el servidor SMTP requiere autenticación (como suele ser el caso con la mayoría de los servidores modernos).

Funcionalidades avanzadas

Además del envío básico de correos electrónicos, la API de JavaMail permite agregar características avanzadas como:

- Envío de archivos adjuntos.
- Formato de mensajes en HTML.
- Gestión de correos entrantes (recepción de correos electrónicos).
- Soporte para varios protocolos de correo como IMAP y POP3.

El primer punto, envío de archivos adjuntos fue utilizado durante el desarrollo del proyecto para enviar un archivo pdf de los respectivos reportes o notificaciones al usuario.









-Uso de Archivos:

El manejo de archivos es una funcionalidad esencial en la mayoría de los sistemas y aplicaciones. En Java, la manipulación de archivos se realiza a través de varias clases y métodos que permiten crear, leer, escribir, y modificar archivos de texto o binarios. Las clases más utilizadas para trabajar con archivos en Java incluyen File, FileWriter, FileReader, BufferedWriter, BufferedReader, y las clases del paquete java.io.file para operaciones más avanzadas.

En este proyecto, el manejo de archivos fue crucial para la persistencia de datos de distintas entidades del sistema de parqueo. A lo largo del desarrollo, se utilizaron archivos de texto, como los que se muestran en la imagen, para llevar el registro de información clave, como los detalles de los administradores, clientes, inspectores y el parqueo en sí. Adicionalmente, se crearon archivos adicionales para registrar los historiales de parqueo y generar reportes sobre las multas y los pagos realizados.

Principales operaciones realizadas sobre archivos en este proyecto:

1. Creación de Archivos: Se crearon archivos para cada rol en el sistema (Administrador, Cliente, Inspector, Parqueo) que almacenan los datos pertinentes a cada uno. Esto permite que la información persista entre ejecuciones del programa.
2. Escritura en Archivos: Cada vez que un cliente parquea o desparca su vehículo, o cuando un inspector registra una multa, los cambios se reflejan en los archivos correspondientes. Por ejemplo, el archivo Parqueo registra los detalles de los espacios de parqueo disponibles y ocupados.
3. Lectura de Archivos: Para restaurar el estado del sistema al iniciar la aplicación, los datos se leen de los archivos mencionados, cargando la información de los administradores, clientes, inspectores y parqueos almacenados.
4. Historiales y Reportes: Además de los archivos mostrados en la imagen, se utilizaron archivos adicionales para llevar el control de los historiales de parqueo y reportes. Estos archivos permiten la generación de reportes de multas, historial de pagos y reportes detallados del estado de cada cliente o parqueo, facilitando el acceso a información histórica para auditorías o revisiones por parte de los inspectores y administradores.

 src	13/10/2024 19:43	Carpeta de archivos	
 target	19/10/2024 17:39	Carpeta de archivos	
 Administrador	19/10/2024 8:20	Documento de tex...	1 KB
 Cliente	19/10/2024 8:20	Documento de tex...	1 KB
 HistorialMulta	19/10/2024 8:20	Documento de tex...	1 KB
 HistorialParqueo	19/10/2024 15:28	Documento de tex...	2 KB
 Inspector	18/10/2024 11:06	Documento de tex...	1 KB
 Parqueo	19/10/2024 15:28	Documento de tex...	1 KB

-Emitir información por medio de archivos PDF:

En el proyecto, la funcionalidad de generar y emitir información en formato PDF se implementó utilizando la librería **iText7**, una de las bibliotecas más populares para la creación y manipulación de archivos PDF en Java. A continuación, describo los principales aspectos relacionados con esta implementación:

Generación de PDF en el Proyecto

En el contexto del proyecto, el objetivo principal de utilizar archivos PDF fue para generar **reportes de cliente, administrador e inspector**, incluyendo información relacionada con los espacios de parqueo utilizados, así como el historial de multas etc. Para ello, se implementó una función que genera el reporte y lo guarda como un archivo PDF.

Implementación en el Proyecto:

La funcionalidad de generación de PDF se implementó en los métodos de las clases Reporte, estos métodos generan un documento PDF con la información respectiva.

Pasos:

- **Crear el archivo PDF:** Se creó un archivo PDF utilizando las clases PdfWriter y PdfDocument de iText7, donde PdfWriter se encarga de escribir en el archivo y PdfDocument de gestionar el documento en sí.

- **Agregar contenido al PDF:** Una vez creado el archivo PDF, se añadió el contenido del reporte utilizando la clase Document y Paragraph de iText7. Estas clases permiten estructurar el contenido en bloques de texto.
- **Cerrar el documento:** Finalmente, una vez que se ha añadido todo el contenido deseado, se cierra el documento para finalizar la escritura y guardarlo en el sistema de archivos.

Solución:

El diagrama de clases se encuentra en la carpeta del proyecto.

-Organización del Proyecto:

Paquetes:

El proyecto está organizado bajo el paquete principal `com.mycompany.proyectoparqueos`. Dentro de este paquete, se encuentran varias clases divididas en diferentes categorías según sus responsabilidades.

Clases de Usuarios:

- **Administrador.java, Cliente.java, Inspector.java:** Estas clases representan los diferentes tipos de usuarios que interactúan con el sistema. Cada clase contiene atributos y métodos específicos para gestionar las funciones de cada tipo de usuario.
- **AñadirCarroCliente.java, ConsultarAdmin.java, ConsultarCliente.java, ConsultarInspector.java:** Clases que permiten gestionar las consultas relacionadas con los datos de administradores, clientes e inspectores, así como agregar carros a los clientes.

Manejo del Parqueo:

- **EspacioDeParqueo.java:** Esta clase gestiona la información de los espacios disponibles en el parqueo, incluyendo si están ocupados y por qué carro.
- **HistorialMultas.java, HistorialParqueo.java:** Estas clases gestionan el registro de las multas y el historial de parqueo, lo que facilita la creación de reportes y análisis históricos.
- **Parqueo.java:** Representa la lógica principal del sistema de parqueo, gestionando los espacios, horarios y las tarifas.

Menús de Interfaz:

- Clases como **MenuAdministrador.java**, **MenuCliente.java**, **MenuInspector.java**, **MenuParquear.java**, y **MenuRevisarParqueo.java** están orientadas a las interfaces gráficas (GUIs) para que cada tipo de usuario (cliente, administrador o inspector) pueda interactuar con el sistema según sus permisos. Cada menú tiene funciones específicas y permite realizar operaciones como parquear, revisar multas, pagar, configurar parqueo, etc.

Validaciones y Autenticación:

- **ValidacionesExceptions.java**: Clase dedicada a manejar las excepciones personalizadas del sistema, facilitando la validación de entradas de usuario.
- **VerificarPinAdmin.java**, **VerificarPinCliente.java**, **VerificarPinInspector.java**: Estas clases garantizan que solo los usuarios autenticados puedan acceder a su respectiva funcionalidad.

Control de Acceso:

- **Autenticación por PIN**: Las clases **VerificarPinAdmin**, **VerificarPinCliente**, y **VerificarPinInspector** validan que los usuarios ingresen un PIN correcto antes de acceder a sus respectivas funciones.
- **Roles de Usuario**: Las interfaces gráficas se diseñan para mostrar solo las opciones que son relevantes para el tipo de usuario autenticado. Por ejemplo, el cliente puede acceder a opciones de parqueo y pago, mientras que el inspector puede revisar el estado del parqueo y asignar multas.

Acceso a Funciones por Tipo de Usuario:

- **Cliente**: Tiene acceso a la visualización y manipulación de su perfil y los carros asociados, así como al historial de parqueo. Puede parquear, pagar multas, y agregar tiempo a su espacio.
- **Administrador**: Puede configurar el parqueo (como tarifas y disponibilidad de espacio) y consultar reportes del sistema.

- **Inspector:** Puede revisar los espacios de parqueo ocupados, asignar multas y generar reportes sobre el estado de los espacios y los carros parqueados.

Clases de Tipo Reporte:

ReporteCliente:

La clase ReporteCliente genera informes detallados sobre el uso del parqueo por parte de los clientes. Proporciona un historial de espacios utilizados, tiempos de parqueo y costos asociados. También incluye información sobre multas registradas, todo presentado en formato claro, ya sea en consola o en PDF.

ReporteAdmin:

La clase ReporteAdmin ofrece una visión general del parqueo para los administradores. Incluye estadísticas como la ocupación de los espacios, ingresos generados y multas acumuladas. Este reporte es clave para la toma de decisiones estratégicas sobre la eficiencia y el rendimiento del parqueo.

ReporteInspector:

La clase ReporteInspector proporciona información relevante para los inspectores. Muestra detalles de vehículos actualmente estacionados, infracciones recientes y espacios ocupados sin pago. Facilita la supervisión y cumplimiento de las normativas del parqueo.

Conclusiones del trabajo:

-Problemas encontrados y soluciones a los mismos:

Control de Acceso Basado en Roles:

Problema: Inicialmente, no había una diferenciación clara en las interfaces ni en las funcionalidades entre los tipos de usuarios (administrador, cliente e inspector). Esto ocasionaba confusión y riesgos en el acceso a funciones no autorizadas.

Solución: Se implementó un sistema de autenticación mediante PIN, donde cada usuario accede a sus funciones específicas (administrador, cliente o inspector) basándose en el tipo de usuario. Adicionalmente, las interfaces gráficas (GUI) se personalizaron para que cada rol tuviera acceso solo a las funcionalidades correspondientes.

Manejo de Parqueos y Espacios Ocupados:

Problema: Al agregar la lógica para que los carros no puedan parquearse en espacios ocupados, surgieron dificultades con la actualización en tiempo real de los espacios disponibles y los carros no aparcados.

Solución: Se ajustaron los métodos que manejan los espacios de parqueo y los carros para garantizar que al parquear un carro, los espacios ocupados se reflejen de inmediato en la interfaz. Se utilizó un ArrayList para rastrear los carros no aparcados y asegurar que los espacios ocupados no aparezcan disponibles en los menús.

Validación de Entradas:

Problema: Durante la entrada de datos en los formularios (como tiempo de parqueo y precios), surgieron errores de validación, especialmente cuando se ingresaban caracteres no numéricos.

Solución: Se crearon excepciones personalizadas (ValidacionesExceptions) que permiten manejar errores de validación de manera más eficiente y brindar retroalimentación clara al usuario mediante cuadros de diálogo (JOptionPane).

Integración de Archivos para el Manejo de Historial:

Problema: La persistencia de datos en archivos, como el historial de parqueo y multas, presentaba dificultades debido a la estructura de los archivos y la escritura en tiempo real.

Solución: Se diseñó una estructura de archivos bien definida para almacenar los datos de cada tipo de usuario (Administrador, Cliente, Inspector) y sus interacciones (historial de parqueo y multas). Esto permitió un manejo más fluido de la información, facilitando la generación de reportes y consultas históricas.

Problema con GitHub:

Problema: Durante el desarrollo del proyecto, enfrentamos un problema importante relacionado con la integración de código mediante GitHub. En particular, tuvimos un conflicto entre los push y pull realizados por mí y mi compañera en la rama main, lo que llevó a la corrupción de ciertos archivos críticos. Uno de los problemas más serios fue que, debido a este conflicto, el código relacionado con los elementos de Swing, generado automáticamente por NetBeans, se modificó de forma inesperada, y dado que NetBeans no permite la manipulación directa de algunas partes del código de interfaz gráfica, esto causó que la rama main quedara inutilizable.

Solución: Nuestra solución fue crear una nueva rama denominada prueba_final, donde trasladamos el trabajo sin los errores de corrupción. A partir de ese momento, trabajamos exclusivamente en esta nueva rama, lo que nos permitió avanzar sin más interrupciones. Optamos por no intentar reparar main, ya que el problema con los archivos dañados y el código de Swing era complejo y preferimos evitar más riesgos. Esta decisión nos permitió concentrarnos en finalizar el proyecto sin problemas adicionales relacionados con la gestión del repositorio.

Conclusión General

El proyecto se llevó a cabo con éxito, logrando implementar un sistema de parqueo que permite gestionar de manera eficiente los espacios, tiempos de parqueo y las

funciones específicas de cada tipo de usuario. Se vencieron los desafíos técnicos presentados casi en su totalidad, lo que dio lugar a un sistema robusto y funcional que cumple con los requerimientos iniciales.

-Aprendizajes Obtenidos:

Durante el desarrollo del proyecto, se adquirieron varios conocimientos y habilidades que resultaron clave para superar los retos técnicos y lograr una implementación exitosa. A continuación, se detallan algunos de los aprendizajes más relevantes:

Manejo de Archivos en Java:

A lo largo del proyecto, fue necesario trabajar con archivos para almacenar información de manera persistente, como los datos de parqueo, multas y usuarios. Esto permitió reforzar el conocimiento en el uso de clases como File, FileWriter y FileReader en Java, así como la importancia de controlar excepciones al manejar entradas y salidas de archivos.

Control de Acceso Basado en Roles:

Implementar un sistema que diferencie las funcionalidades según el tipo de usuario (Administrador, Cliente e Inspector) permitió aprender sobre la importancia de estructurar correctamente el acceso a funcionalidades críticas de un sistema. Se practicó el uso de sistemas de autenticación simples, como el manejo de PINs, y su integración con interfaces gráficas.

Diseño de Interfaces Gráficas con Swing:

Utilizar la biblioteca Swing para desarrollar la interfaz gráfica (GUI) fue un desafío importante. A lo largo del proyecto, se aprendió cómo estructurar mejor las ventanas, botones y cuadros de diálogo para ofrecer una experiencia de usuario intuitiva y clara. Además, se practicó cómo manejar eventos en los componentes gráficos para actualizar la información en tiempo real, como en el caso del manejo de los carros parqueados.

Validación de Entradas de Usuario:

Aprender a validar las entradas del usuario fue fundamental para evitar errores en la aplicación. El uso de excepciones personalizadas (`ValidacionesExceptions`) facilitó la captura de errores como la entrada de datos no numéricos o valores fuera de los rangos permitidos, y brindó retroalimentación clara al usuario mediante ventanas emergentes.

Gestión de Lógica de Negocio:

Uno de los aprendizajes más significativos fue la importancia de gestionar correctamente la lógica de negocio dentro de las clases del proyecto. La separación adecuada de responsabilidades entre clases como `Cliente`, `Inspector`, `Carro` y `Parqueo` permitió mantener un código más organizado, facilitando el desarrollo del sistema. El manejo de eventos como la liberación automática de espacios y la generación de multas también fueron aspectos clave en esta gestión.

Trabajo con Colecciones de Datos:

La implementación de listas (`ArrayList`) y su correcta manipulación para almacenar y consultar objetos como `Carro`, `EspacioDeParqueo` y `Multa` fue un área de aprendizaje importante. Aprender a gestionar estas colecciones y a optimizarlas para la búsqueda y actualización de datos permitió mejorar el rendimiento de la aplicación.

Conclusión

Este proyecto proporcionó una experiencia valiosa en el desarrollo de una aplicación compleja en Java, integrando tanto la lógica de negocio como la interacción con el usuario a través de una interfaz gráfica. Los aprendizajes obtenidos, desde el manejo de archivos hasta la depuración y validación, refuerzan las bases necesarias para desarrollar sistemas más robustos y eficientes en el futuro.

Bibliografía:

Oracle. (n.d.). *Java Platform, Standard Edition Documentation*. Oracle.

<https://docs.oracle.com/javase/8/docs/>

Java Community. (n.d.). *The Swing Tutorial*. Oracle.

<https://docs.oracle.com/javase/tutorial/uiswing/>

LISTA DE REVISIÓN DEL PROYECTO

Concepto	Puntos originales	Avance 100%/0	Puntos obtenidos	Análisis de resultados
CRUD administradores	3	100		
CRUD usuarios	4			
CRUD inspectores	3			
Enviar correo	3			
Cambio de PIN	6	100		
Configuración del parqueo	4	100		
Enviar correo	1			
Reportes de administrador:		85		No se realizaron las estadísticas debido a un gran atraso de tiempo con el GitHub
Ingresos de dinero por estacionamiento	3			
Ingresos de dinero por multas	3			
Lista de espacios de parqueos:				
Todos los espacios	1			
Espacios ocupados	1			
Espacios vacíos	1			
Historial de espacios usados	3			
Historial de multas	3			
Estadística detallada de uso	4			
Estadística resumida de uso	4			
Parquear (cobro)	9			
Envío de correo	1			
Agregar tiempo:		100		
Actualizar tiempo comprado	3			
Enviar correo	1			
Desaparcar cuando se tiene solo un vehículo parqueado	2	100		
Desaparcar cuando se tienen varios vehículos parqueados	3			
Enviar correo	1			
Reportes de usuario:		100		
Buscar parqueos disponibles	1			
Historial de espacios usados	2			
Multas del usuario	2			
Revisar parqueo por inspector	4	100		
Generar datos de la multa	4			
Emitir el PDF	1			
Enviar correo	1			
Reportes de inspector:		100		
Lista de espacios de parqueos	1			
Historial de multas	2			
Validación de datos y procesos	10	100		
Documentación del proyecto	5	100		
TOTAL	100	98, 6		
Partes desarrolladas adicionalmente				