ENTREGA PRÁCTICAS

Cuenta de gastos

Interfaces Persona Computador

IPC – DSIC

UPV

Curso 2023-2024

Índice

1. Descripción y escenarios de uso

Se desea desarrollar una aplicación de escritorio que permita a los usuarios registrados llevar un control y seguimiento de sus gastos. La aplicación se adaptará al tamaño del dispositivo donde se ejecute.

Para poder utilizar la aplicación el usuario deberá de estar registrado, una vez registrado y logueado el usuario podrá acceder a las diferentes funcionalidades de la aplicación. El usuario debería de poder hacer un logout sin que la aplicación termine.

El usuario podrá añadir en cualquier momento apuntes de gastos a su histórico, así como visualizar el histórico de todos sus apuntes. De cada apunte se guardará información como la fecha del gasto, el producto, las unidades y el precio. De manera opcional también se guardará una imagen del tique o factura para poder ser utilizada para reclamar la garantía del producto.

Seguidamente se detallan los escenarios de uso que se han obtenido tras el análisis de requisitos del sistema. Estos se deben utilizarse para diseñar e implementar adecuadamente la aplicación requerida.

1.1. Registro de usuario

Alberto ha visto que en la asignatura IPC han desarrollado una aplicación para llevar el seguimiento de las cuentas personales, como tiene que ahorrar y no sabe cómo gasta su dinero le parece una buena idea utilizar esta aplicación. Tras descargarla, Alberto accede a la opción de registro. Desde la opción se abre un formulario donde Alberto puede introducir sus datos de contacto:

- Nombre
- Nickname (usado para acceder a la aplicación y para ser mostrado a los demás usuarios. No puede contener espacios. No puede repetirse en la aplicación)
- Password (cualquier combinación de letras y números con más de 6 caracteres)
- Correo electrónico
- Imagen de perfil (opcional)

Tras introducir todos sus datos y elegir una ardilla como imagen de perfil, el sistema comprueba que todos los datos son correctos y no se ha introducido ningún dato con un formato no permitido. El sistema guarda también la fecha en la que Alberto se ha registrado. Como todos los datos son correctos, informa a Alberto de que ha sido añadido correctamente como usuario y que debe autenticarse para poder utilizar la aplicación.

1.2. Autenticarse

Pilar acaba de recordar que ayer lleno el depósito de gasolina y decide apuntar el gasto antes de que se le olvide. Nada más abrir la aplicación, accede a la opción de autenticarse, donde aparece un formulario donde introduce su nickname y contraseña. Tras acceder, el sistema comprueba si el usuario está registrado en el sistema.

Pilar se ha confundido al introducir la contraseña, por lo que el sistema no lo encuentra y le informa que no está registrado en el sistema, permitiéndole introducir los datos de

nuevo. Pilar vuelve a introducir sus datos, esta vez de forma correcta. El sistema la autentifica, permitiéndole el acceso al resto de funcionalidades.

1.3. Añadir un gasto o apunte a la cuenta

Alberto, quiere apuntar en su cuenta de gastos lo que le ha costado la comida de hoy. Tras autenticarse, Alberto accede a la opción de añadir gasto. Mientras está añadiendo el gasto se da cuenta que le resultaría interesante saber lo que gasta en restaurantes así que decide crear una nueva categoría de gasto. Acede a la opción de crear categoría y añade la categoría de "Restaurante". Alberto continua con su tarea, añade el coste, la fecha del gasto, un título o nombre, una pequeña descripción y selecciona como categoría "Restaurante".

Alberto se acuerda que ayer también compro unos auriculares BlueTooth así que añade este otro gasto, tras introducir todos los datos añade también una captura de la factura para poder tenerla disponible en caso de requerir la garantía del producto.

1.4. Visualizar cuenta de gastos

Llega final de mes y Pilar no entiende como su cuenta bancaria esta tan mal así que quiere ver los gastos que ha realizado, tras autenticarse Pilar utiliza las opciones disponibles para visualizar el gasto mensual, de manera total y también es sus diferentes categorías.

Como Pilar ya lleva tiempo utilizando la aplicación también quiere ver cuánto ha gastado este mes respecto al resto de meses del año e incluso respecto al mismo mes del año anterior

Este escenario queda abierto a que podáis diseñarlo como consideréis más adecuado, podéis utilizar filtros de cualquier tipo y graficas. Javafx incorpora gráficas que podéis utilizas pues lo único que requieren es de listas observables de datos.

1.5. Imprimir estado de cuenta de gastos

Alberto quiere mostrar a su compañera las razones por las que este año no ha ahorrado nada, después de autenticarse y navegar por su cuenta genera un reporte en formato PDF con la evolución de sus gastos anuales.

1.6. Actualizar datos del usuario

Pilar quiere actualizar los datos de su perfil, después de autenticarse accede a la opción que le permite actualizar los datos de su perfil. Después de hacer los cambios que considera en cualquiera de los campos excepto en su nickName que no se puede modificar, indica al sistema que quiere guardar los cambios introducidos. Tras comprobar que los nuevos valores cumplen con los requisitos necesarios el sistema guarda la información y se despide de Pilar.

1.7. Actualizar cargo

Pilar acaba de encontrar el tique de la comida del lunes pasado que ya ha introducido como gasto, así pues, quiere añadir la imagen escaneada del tique. Tras loguearse accede

a la opción que le permite modificar un gasto, tras encontrar el gasto procede a subir la imagen escaneada y cierra con ello la aplicación.

1.8. Eliminar un cargo

Alberto acaba de recibir un email de la empresa en el que le confirman que le van a abonar la gasolina de su último viaje, como el gasto ya lo tiene anotado en su cuenta decide acceder y eliminarlo. Tras autenticarse y buscar el apunte pasa a eliminarlo.

1.9. Eliminar una categoría

Alberto definió varias categorías que pasado un tiempo no ha utilizado, así pues, ha decidido eliminarlas para que no aparezcan en la visualización de estado de sus gastos. Tras autenticarse busca las categorías y las elimina.

2. Modelo de datos

La persistencia de los datos se realizará a través de una base de datos SQLite. El proceso es transparente para el programador. Para ello proporcionamos el fichero acountIPC.jar (librería) que contiene las clases del modelo que se deben de utilizar para realizar la entrega. Para que la librería funcione de manera adecuada es necesario incluir también en el proyecto la librería sqlite-jdbc-3.41.2.1.jar. Ambos ficheros estarán disponibles en poliformat.

Te recomendamos que crees la carpeta "lib" dentro de la carpeta donde se encuentra tu proyecto de Netbeans para la entrega y después copia estos dos ficheros en dicha carpeta, Figura 1.

Los proyectos creados por Netbeans para aplicaciones Java tienen una carpeta `Libraries` donde añadir las librerías externas que son necesarias en el proyecto. Par añadir una librería basta con situarse sobre esa carpeta `Libraries`, y desde el menú contextual (botón derecho del ratón), seleccionar la opción `Add JAR/Folder`, tal y como se muestra en la Figura 2.

Tras seleccionar la opción, aparecerá un diálogo donde se deben seleccionar los dos ficheros jar que hemos indicado y que deberían estar almacenado en la carpeta lib dentro del proyecto. Tras aceptar, las librerías se cargan, mostrando todas las clases disponibles (Figura 3).

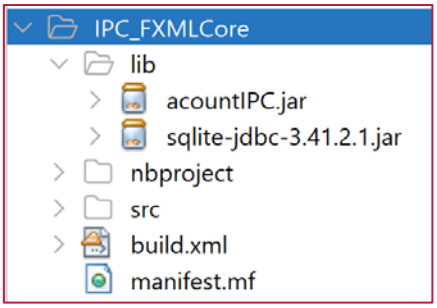

Figura 1. Carpeta lib

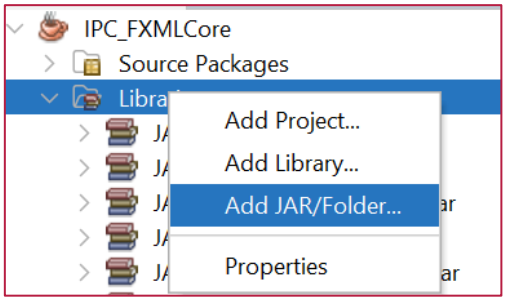

Figura 1 Seleccionar la opción de incluir la librería

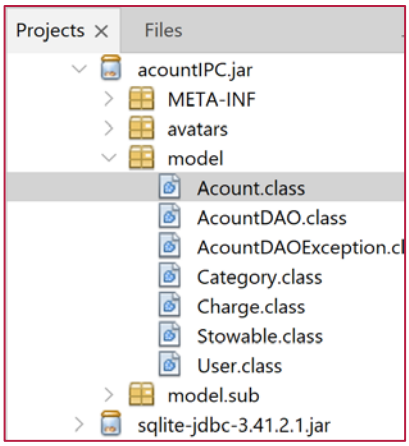

Figura 3 Clases en acountIPC.jar

2.1. Clases del modelo

Como hemos visto en clase, los objetos primarios del problema son: usuario, categoría y gasto (**User**, **Category**, **Charge**). Para facilitar el desarrollo de la práctica hemos creado un conjunto de clases que representan el modelo, dentro del cual se encuentran estas tres clases. El modelo se ha estructurado de tal manera que cuando se crean objetos de estas clases o cuando se modifican sus campos, estos se guardan en una base de datos SQLite, así, cada vez que arranquemos la aplicación dispondremos de los objetos que teníamos al cerrarla.

Esto implica algunas restricciones, NO podemos crear objetos de estas clases mediante sus constructores pues no los hemos definido públicos. Para crear y recuperar los objetos hemos creado la clase **Acount** en la que se han definido los métodos necesarios para poder registrar objetos y para poder recuperarlos al arrancar la aplicación. Esta clase implementa el patrón Singleton, este patrón de programación obliga a que en toda la aplicación solo se pueda crear un objeto de la clase Acount y siempre estará disponible desde cualquier parte del código. A efectos prácticos reducirá la complejidad del código pues evitará pasar objetos entre las diferentes clases controladoras. El constructor de esta clase también es privado y hay que utilizar el método estático **getInstance()** para obtener el objeto.

Para facilitar el desarrollo de la práctica se ha guarda en la clase Acount un objeto del tipo User que sirve para almacenar al usuario logueado en la aplicación.

User

La clase User almacena toda la información sobre un usuario de la aplicación, los atributos de la clase son:

- String **name**: nombre del usuario.
- String **surname**: apellido del usuario.
- String **email**: cadena que contiene el email del usuario.
- String **nickName**: credencial que usará el usuario para conectarse en el sistema.
- String **password**: contraseña que usará el usuario para conectarse en el sistema.
- LocalDate **registerDate**: fecha de alta del usuario en la aplicación.

- Image **image**: avatar con la imagen del miembro. Si en el constructor no se utiliza (null), se añade automáticamente un avatar por defecto.

Se pueden modificar todos los campos salvo **nickName**, para ello se dispone de los correspondientes *setter*.

El constructor de User no es público, para crear un usuario nuevo hay que invocar al método de la clase Acount, registerUser(). Este método se encargar de crear el nuevo usuario y de guardarlo. El método retorna un boolean, true si se ha creado.

Esta clase tiene métodos estáticos para facilitar la validación de los campos nickeName, email y password.

Category

Los atributos de la clase son:

- String **name**: nombre de la categoría.
- String **description**: descripción.

El constructor de Category no es público, para crear una categoría hay que invocar al método de la clase Acount, registerCategory(). La categoría depende de cada usuario, para crear una categoría es necesario que en la clase Acount exista un usuario logueado al que se le asignará la categoría.

Charge

Los atributos de la clase son:

- Int **Id**: es un campo automático que sirve para identificar al objeto en la base de datos, se genera en el momento de registrar el gasto
- String **name**: un título del gasto.
- String **description**: cadena que contiene la descripción.
- Category **category**: categoria del gasto
- Double **cost**: coste del gasto.
- Int **units:** unidades
- LocalDate **date**: fecha de realización del gasto.
- Image **scanImage**: escaneo de la factura (opcional).

Se pueden modificar todos los campos salvo el **id**, para ello se dispone de los correspondientes *setter*.

El constructor de Charge no es público, para crear un objeto nuevo hay que invocar al método de la clase Acount, registerCharge().

Acount

Esta clase mantiene una variable del tipo User que facilita acceder al usuario loguedo desde cualquier clase controladora de la aplicación. Los métodos definidos son:

---

public static Acount getInstance()

Retorna el único objeto de la clase Acount que se va a crear en toda la aplicación. En el momento que se crea el objeto de la clase Acount se busca en el directorio donde se

ejecuta el proyecto el fichero data.db, en el caso de no encontrarlo se genera uno nuevo con la estructura de la base de datos correspondiente. Este fichero se puede editar con herramientas de bases de datos como DB Browser que podéis descargar en https://sqlitebrowser.org/

---

public boolean registerUser(String name, String surname, String email,

String login, String password, Image image, LocalDate date)

Se utiliza para registrar un nuevo usuario, una vez registrado el usuario NO queda logueado en acount. Retorna true si se ha registrado el usuario

public boolean logInUserByCredentials(String login, String password)

Cuando se invoca este método y si se encuentra un usuario con las credenciales introducidas, este usuario queda guardado dentro de la clase Acount como usuario logueado y las consultas sobre categorías y gastos se realizan sobre este usuario. Si no existe usuario con estas credenciales retorna false

public boolean logOutUser()

Se elimina el usuario logueado en Acount, las consultas sobre categorías y gastos retornarán null. Si no hay usuario logueado retorna false.

---

public boolean registerCategory(String name, String description)

Se registra una categoría para el usuario logueado. Si se ha registrado retorna true.

public boolean removeCategory(Category category)

se elimina una categoria del usuario logeado y todos los gastos de esta categoría del usuario logueado. Retorna true si todo ha ido bien.

public List<Category> getUserCategories()

Retorna todas las categorías del usuario logueado, en otro caso retorna null

---

public boolean registerCharge(String name, String description, double cost, int units, Image scanImage, LocalDate date, Category category)

El método registra un cargo sobre el usuario logueado, al cargo creado se le añade el id. Retorna true si se ha resgistrado

public boolean removeCharge(Charge charge)

Elimina un cargo a partir de su id. Retorna true si se ha eliminado

public List<Charge> getUserCharges()

Retorna la lista de todos los cargos del usuario logueado. Si no hay usuario logueado retorna null

2.2. Uso de la librería desde el proyecto

Para acceder a los métodos de la librería, es necesario instanciar primero un objeto de la clase *Acount*, utilizando para ello el método estático *getInstance*(). Después, puede obtenerse la información registrada o modificarla a través del API proporcionada.

```
boolean isOK = Acount.getInstance().logInUserByCredentials("usuarioIPC", "123456");
```

3. Ayudas a la programación

3.1. Carga de imágenes desde disco duro

Existen diversas formas de cargar una imagen desde el disco duro y mostrarla en una *ImageView*:

1. La imagen está en un subdirectorio del directorio src del proyecto, por ejemplo, llamado images:

```
String url = File.separator+"images"+File.separator+"woman.PNG";
Image avatar = new Image(new FileInputStream(url));
myImageView.imageProperty().setValue(avatar);
```

2. La imagen está en cualquier parte del disco duro y tenemos el *path* completo de la misma:

```
String url = "c:"+File.separator+"images"+File.separator+"woman.PNG";
Image avatar = new Image(new FileInputStream(url));
myImageView.imageProperty().setValue(avatar);
```

3.2. Manejo de fechas y del tiempo

En la aplicación se deben gestionar atributos de tipo *LocalDateTime, LocalDate* y de tipo *DateTime*. A continuación, os explicamos algunos métodos de utilidad.

Obtención de la semana del año a la que pertenece una fecha

El siguiente código obtiene la semana a la que pertenece el día de hoy, así como el número del día de la semana (1 para lunes, 2 para martes, etc.)

```
WeekFields weekFields = WeekFields.of(Locale.getDefault());
int currentWeek = LocalDate.now().get(weekFields.weekOfWeekBasedYear());
int numDayNow=LocalDate.now().get(weekFields.dayOfWeek());
```

Creación de una fecha o un campo de tiempo

Consulta el API de *LocalDate* y *LocalTime* para conocer todos los métodos que proporciona para crear un objeto de sus clases, pero algunos que te pueden resultar útiles son:

```
LocalDate sanJose = LocalDate.of(2020, 3,19);
LocalTime mascleta = LocalTime.of(14,0);
```

Actualizando una fecha

Es posible incrementar o decrementar días, meses, años, minutos, horas, etc. a los campos de tipo *LocalTime*, *LocalDate*, o *LocalDateTime* usando su propia API. Por ejemplo, el siguiente código incrementa en siete días la fecha actual, guardando el resultado en una nueva variable. También incrementa en un mes la fecha actual, salvando la información en otra variable. Finalmente, incrementa el tiempo actual en 90 minutos, guardando el resultado en otra variable de tipo *LocalTime*.

```
LocalDateTime nextWeekDay= LocalDateTime.now().plusDays(7);
LocalDateTime nextMontDay = LocalDateTime.now().plusMonths(1);
LocalTime endedTime = LocalTime.now().plusMinutes(90);
```

3.3. Configurar DatePicker

Los componentes DatePicker permiten seleccionar al usuario una fecha, pudiendo acceder al valor seleccionado a través de su propiedad valueProperty(). Es posible configurar la forma en la que se visualiza por defecto cada día del calendario, siendo posible deshabilitar algunos días, cambiar el color de fondo, etc. La forma en la que se configura esta visualización es similar a la que empleamos para configurar una ListView o una TableView. La diferencia es que en este caso debemos extender la clase DateCell, y usar el método setDayCellFactory. Por ejemplo, el siguiente código configura un DatePicker para que se inhabiliten los días anteriores al 1 de Marzo de 2020 (ver Figura 3).

```
dpBookingDay.setDayCellFactory((DatePicker picker) -> {
            return new DateCell() {
                @Override
                public void updateItem(LocalDate date, boolean empty) {
                    super.updateItem(date, empty);
                    LocalDate today = LocalDate.now();
                    setDisable(empty || date.compareTo(today) < 0 );
                }
            };
        });
```

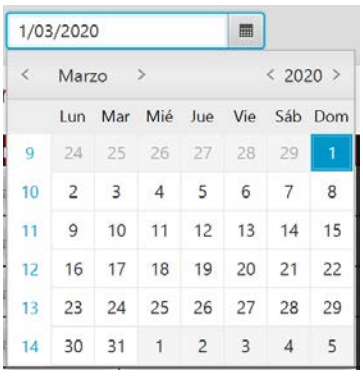

Figura 2. DataPicker configurado para inhabilitar días en el pasado

4. Instrucciones de Entrega

El proyecto debe implementar los escenarios de usos planteados en el punto Caso de Estudio. Debe diseñarse utilizando todo los principios, heurísticas y contenido visto en el apartado de teoría de la asignatura.

Aunque para la entrega no se requiere de la documentación generada durante el proceso de desarrollo de la entrega, sí que se pide que se sigua el proceso de desarrollo centrado en el usuario visto en la asignatura con el análisis de tareas, diseño conceptual, y diseño físico con prototipos.

Con respecto a la entrega:

- Exporta el proyecto Netbeans a un fichero zip (opción `Fichero>Exportar Proyecto> A Zip`). Una alternativa es guardar toda la carpeta del proyecto en un fichero zip. Para reducir espacio puedes eliminar de este zip la carpeta dist
- Un único miembro del grupo sube el fichero zip a la tarea correspondiente, incluyendo en el campo de comentarios los nombres de los miembros del grupo.
- La fecha de entrega para todos los grupos es el **26 de mayo de 2024**

5. Evaluación

- Aquellos proyectos que no compilen o que no se muestren la pantalla principal al arrancar se calificarán con un cero.
- Se deberán incluir los diálogos de confirmación, errores, etc. que se considere necesario.
- Para evaluar el diseño de la interfaz de la aplicación se tendrán en consideración **las directrices estudiadas en clase de teoría**.
- Debe ser posible redimensionar la pantalla principal, cuyos controles se ajustarán adecuadamente para usar el espacio disponible (usa los contenedores vistos en clase: *VBox*, *HBox*, *BorderPane*, *GridPane*, etc.).