

**BEDU**

**Desarrollo Móvil**

**Kotlin Fundamentals**

**19:00 – 22:00**

**Equipo 22**

**Proyecto Guía**

**ChronoMaster 2021**

**Presentan:**

**Manuel Díaz Capistran**

**José Bernal Fonseca**

**Ángel Omar Gómez Castillo**

**Braulio David Hernández Palagot**

Contenido

[Introducción 2](#_Toc75723410)

[Definición del proyecto 3](#_Toc75723411)

[Funcionalidades 3](#_Toc75723412)

[Clases 3](#_Toc75723413)

[Código de clases 4](#_Toc75723414)

[Usuario 4](#_Toc75723415)

[Tarea 4](#_Toc75723416)

[Código de funciones 5](#_Toc75723417)

[main() 5](#_Toc75723418)

[bienvenida() 6](#_Toc75723419)

[visualizarPrograma() 8](#_Toc75723420)

[revisarConflictos() 8](#_Toc75723421)

[resumenUsuario() 10](#_Toc75723422)

[agregarTarea() 10](#_Toc75723423)

[editarTarea() 11](#_Toc75723424)

[eliminarTarea() 14](#_Toc75723425)

[mostrarTareas() 14](#_Toc75723426)

[calcularLapso() 15](#_Toc75723427)

# Introducción

Durante el primer módulo de la segunda etapa del curso de Desarrollo Móvil se realizó un proyecto guía con la finalidad de comprender mejor y aplicar los conocimientos adquiridos durante las sesiones de dicho módulo.

El equipo 22 decidió que su proyecto guía se basaría en una agenda y se realizaría completamente en consola. En este documento se plasmará toda la información necesaria para comprender cómo se realizó el proyecto.

# Definición del proyecto

El proyecto guía se realizará en base a la idea de crear una agenda totalmente funcional, dejando la ventana abierta para añadir mayor funcionalidad en el módulo siguiente. El nombre comercial provisional de nuestra aplicación será: ***ChronoMaster 2021***.

## Funcionalidades

Las funcionalidades planificadas para la aplicación ***ChronoMaster 2021*** son las siguientes:

* Loggin para controlar el acceso a la agenda.
* Almacenaje de tareas. Incluyendo las opciones de agregar, eliminar o editar tareas.
* Visualización del listado de tareas almacenadas.
* Revisión de conflictos de horario entre las tareas almacenadas.

## Clases

La aplicación ***ChronoMaster 2021*** utilizará dos tipos de datos abstractos, los cuales son los siguientes:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| |  | | --- | | Usuario | | + nombre: String  + contrasenna: String  + tareasTotales: String  + tareasRealizadas: String | | loggin(nombre: String, contrasenna: String) | | |  | | --- | | Tarea | | + titulo: String  + fechaInicio: LocalTime  + fechaFinalizacion: LocalTime  + objetivo: String  + descripcion: String  + lapso: String  + estado: Boolean  + dependenciaInterna: String  + dependenciaExterna: String  + frecuencia: String  + prioridad: int | |

# Código de clases

## Usuario

/\*  
AUTOR: BRAULIO DAVID HERNANDEZ PALAGOT.  
DESCRIPCION: Se creó una clase llamada Usuario que servirá para almacenar los datos del usuario y la funcionalidad de  
loggin, tanto para los datos de prueba que se estarán utilizando durante este proyecto guía, así como los datos de más  
usuarios, pensando en la escalabilidad del proyecto.  
 \*/  
class Usuario(var nombre: String, var contrasenna: String, var tareasTotales: Int = 0, var tareasRealizadas: Int = 0) {  
 //DESCRIPCION: init que mostrará los datos de inicio de sesión del usuario.  
 init {  
 *println*(  
 "Datos de inicio de sesión:\n" +  
 "\tUsuario: ${this.nombre}.\n" +  
 "\tContraseña: ${this.contrasenna}."  
 )  
 }  
 //DESCRIPCION: Función para validar el nombre de usuario y contraseña.  
 fun loggin(nombre: String, contrasenna: String): Boolean {  
 return nombre.equals(this.nombre) && contrasenna.equals(this.contrasenna)  
 }  
}

## Tarea

/\*  
AUTOR: ANGEL OMAR GOMEZ CASTILLO.  
DESCRIPCION: Se creó una data class llamada Tarea que se utilizará para almacenar los datos de cada tarea que el usuario  
ingrese en la agenda.  
 \*/  
data class Tarea(  
 var titulo: String,  
 var fechaInicio: LocalDateTime,  
 var fechaFinalizacion: LocalDateTime,  
 var objetivo: String = "",  
 var descripcion: String = "",  
 var lapso: String = "",  
 var estado: Boolean = false,  
 var dependenciaInterna: String = "",  
 var dependenciaExterna: String = "",  
 var frecuencia: String = "",  
 var prioridad: Int = 0  
)

# Código de funciones

## main()

/\*  
DESCRIPCION: Variables globales que serán utilizdas en varias partes del proyecto, por lo que se declaran en esta  
sección para poder accederlas desde cualquier función del proyecto.  
 tareas: Lista mutable que almacenará las tareas guardadas en la agenda.  
 usuario: Objeto de tipo Usuario inicializado con los datos de inicio de sesión que se utilizarán durante este  
 proyecto guía.  
 \*/  
val *tareas* = *mutableListOf*<Tarea>()  
val *usuario*: Usuario = Usuario("Usuario de prueba", "contraseña\_de\_prueba")  
  
/\*  
AUTOR: EQUIPO 22.  
DESCRIPCION: Función main del proyecto, en la que se encuentra la función bienvenida() que da lugar al funcionamiento  
completo del proyecto.  
 \*/  
fun main() {  
 *bienvenida*()  
}

## bienvenida()

/\*  
AUTOR: EQUIPO 22.  
DESCRIPCION: En esta función se inicia con la pantalla de loggin del programa, donde se piden los datos de nombre de  
usuario y contraseña y se comparan con los datos inicializados en el objeto usuario.  
Si el usuario es validado correctamente se da acceso al menú principal, el cual variará dependiendo de si la lista de  
tareas está vacía o tiene algún elemento. En caso de estar vacía el menú sólo mostrará las opciones de acceder a la  
sección de agregar tarea o de cerrar la agenda; si la lista contiene algún elemento se mostrarán todas las opciones a  
que contiene esta agenda.  
 \*/  
fun bienvenida() {  
 //DESCRIPCION: Variables para leer los datos del usuario y para controlar la opción del menú que se ha seleccionado.  
 var op = 0  
 var inputNombre = ""  
 var inputContrasenna = ""  
  
 /\*  
 DESCRIPCION: El proceso de inicio de sesión se rodea por un ciclo do-while para asegurar que se repita hasta que  
 el usuario sea validad correctamente.  
 \*/  
 do {  
 *print*(  
 "---------------------------------------------\n" +  
 "Iniciar sesión:\n"  
 )  
 *print*("\tIngrese el nombre de usuario: ")  
 inputNombre = *readLine*().*toString*()  
 *print*("\tIngrese la contraseña: ")  
 inputContrasenna = *readLine*().*toString*()  
 } while (!*usuario*.loggin(inputNombre, inputContrasenna)) //DESCRIPCION: Condición para el ciclo do-while.  
  
 //DESCRIPCION: Una vez validado el usuario se le da la bienvenida a la agenda.  
 *println*("\nBienvenido a tu agenda, ${*usuario*.nombre}.")  
  
 /\*  
 DESCRIPCION: El menú principal se rodea de un ciclo do-while para asegurar que ésta sólo se cierre si se selecciona  
 la opción correspondiente al cierre de la agenda.  
 \*/  
 do {  
 *print*(  
 "---------------------------------------------\n" +  
 "Menú principal:\n"  
 )  
 //DESCRIPCION: Condición para revisar si la lista de tareas se encuentra vacía o no.  
 if (*tareas*.isEmpty()) {  
 *println*(  
 "\t1) Agregar tarea.\n" +  
 "\t2) Cerrar Agenda."  
 )  
 } else {  
 *println*(  
 "\t1) Visualizar programa.\n" +  
 "\t2) Revisar conflictos.\n" +  
 "\t3) Ver resumen del usuario.\n" +  
 "\t4) Agregar tarea.\n" +  
 "\t5) Editar tarea.\n" +  
 "\t6) Eliminar tarea.\n" +  
 "\t7) Cerrar Agenda."  
 )  
 }  
  
 //DESCRIPCION: Lectura de la opción seleccionada por el usuario.  
 *print*("Seleccione una opcion: ")  
 op = *readLine*()!!.*toInt*()  
  
 /\*  
 DESCRIPCION: Condición para revisar qué menú se utilizará en el condicional when que controla el menú  
 principal.  
 \*/  
 if (*tareas*.isEmpty()) {  
 when (op) {  
 1 -> *agregarTarea*()  
 2 -> {  
 op = 7  
 *print*("Hasta la próxima, ${*usuario*.nombre}.")  
 }  
 }  
 } else {  
 when (op) {  
 1 -> *visualizarPrograma*()  
 2 -> *revisarConflictos*()  
 3 -> *resumenUsuario*()  
 4 -> *agregarTarea*()  
 5 -> *editarTarea*()  
 6 -> *eliminarTarea*()  
 7 -> *print*("Hasta la próxima, ${*usuario*.nombre}.")  
 else -> {  
 *println*("Opción no valida, inténtelo de nuevo...")  
 }  
 }  
 }  
 } while (op != 7) //DESCRIPCION: Condición para el ciclo do-while.  
}

## visualizarPrograma()

/\*  
AUTOR: JOSE BERNAL FONSECA.  
DESCRIPCION: En esta función si la lista de tareas se encuantra vacía nos mostrará un mensaje que nos lo indicará, de lo  
contrario se ejecutará la función mostrarTareas().  
 \*/  
fun visualizarPrograma() {  
 *println*(  
 "---------------------------------------------\n" +  
 "Programa del día:\n"  
 )  
 if (*tareas*.isEmpty()) {  
 *println*("No hay ninguna tarea.")  
 } else {  
 *mostrarTareas*()  
 }  
}

## revisarConflictos()

/\*  
AUTOR: BRAULIO DAVID HERNANDEZ PALAGOT.  
DESCRIPCION: En esta función se revisará si los horarios de alguna tarea entran en conflicto con los de otra.  
 \*/  
fun revisarConflictos() {  
 //DESCRIPCION: Set mutable que almacenará los mensajes de conflicto.  
 val setConflictos = *mutableSetOf*<String>()  
 *println*(  
 "---------------------------------------------\n" +  
 "Revisión de conflictos:\n"  
 )  
  
 /\*  
 DESCRIPCION: Se utiliza un primer ciclo for para asegurarnos de revisar cada tarea. Se optó por un for que utiliza  
 la función WithIndex() de la lista.  
 \*/  
 for ((i, horarioComparando) in *tareas*.*withIndex*()) {  
 /\*  
 DESCRIPCION: Se utiliza un segundo ciclo for para asegurarnos de comparar la tarea del ciclo anterios con cada  
 tarea de la lista. Se optó por un for que utiliza la función WithIndex() de la lista.  
 \*/  
 for ((j, horarioComparador) in *tareas*.*withIndex*()) {  
 /\*  
 DESCRIPCION: Se utiliza un condicional para evitar que una tarea se compare consigo misma, pues esto  
 generaría un falso conflicto.  
 \*/  
 if (i != j) {  
 /\*  
 DESCRIPCION: Se comparan ahora los horarios de inicio y fin de cada una de las tareas, si el horario de  
 la tarea que se estácomparando contiene el horario de la otra se agrega el mensaje de conflicto al set  
 de conflictos declarado anteriormente.  
 \*/  
 if (horarioComparando.fechaInicio.toString().*contains*(horarioComparador.fechaInicio.toString()) ||  
 horarioComparando.fechaFinalizacion.toString()  
 .*contains*(horarioComparador.fechaFinalizacion.toString())  
 ) {  
 setConflictos.add(  
 "La tarea ${*tareas*[i]}, con horario: ${horarioComparando.fechaInicio} - ${horarioComparando.fechaFinalizacion}\n" +  
 "entra en conflicto con la tarea ${*tareas*[j]}, con horario: ${horarioComparador.fechaInicio} - ${horarioComparador.fechaFinalizacion}.\n" +  
 "Es necesario editar alguna de las dos."  
 )  
 }  
 }  
 }  
 }  
  
 /\*  
 DESCRIPCION: Si el set de conflictos no se encuentra vacío nos enlistará cada uno de los conflictos encontrados, de  
 lo contrario nos mostrará un mensaje avisándonos que no se encontró ningún conflicto.  
 \*/  
 if (!setConflictos.isEmpty())  
 for ((i, elemento) in setConflictos.*withIndex*())  
 *println*(" ${i + 1}) $elemento\n")  
 else  
 *println*("No se encontraron conflictos")  
}

## resumenUsuario()

/\*  
AUTOR: BRAULIO DAVID HERNANDEZ PALAGOT.  
DESCRIPCION: En esta función se mostrarán los datos del usuario, con el objetivo de que pueda revisar cuántas tareas  
tiene aún pendientes.  
 \*/  
fun resumenUsuario() {  
 *println*(  
 "---------------------------------------------\n" +  
 "Resumen del usuario:\n"  
 )  
 *println*(  
 "\tNombre de usuario: ${*usuario*.nombre}.\n" +  
 "\tTareas totales: ${*usuario*.tareasTotales}.\n" +  
 "\tTareas finalizadas: ${*usuario*.tareasRealizadas}.\n" +  
 "\n\tTareas pendientes: ${*usuario*.tareasTotales - *usuario*.tareasRealizadas}"  
 )  
}

## agregarTarea()

/\*  
AUTOR: BRAULIO DAVID HERNANDEZ PALAGOT.  
DESCRIPCION: En esta función se registrarán los datos de una tarea para agregarla a la lista de tareas pendientes.  
 \*/  
fun agregarTarea() {  
 //DESCRIPCION: Las tareas totales del usuario se establecen en cero para contarlas al terminar de agregar la tarea.  
 *usuario*.tareasTotales = 0  
 //DESCRIPCION: Petición de datos de la tarea al usuario.  
 *println*(  
 "---------------------------------------------\n" +  
 "Agregar tarea:\n"  
 )  
 *print*("Ingrese el título de la tarea: ");  
 val title = *readLine*().*toString*()  
 *print*("\tIngrese el mes de inicio [MM]: ");  
 val mesInicio = *readLine*()!!.*toInt*()  
 *print*("\tIngrese el dia de inicio [DD]: ");  
 val diaInicio = *readLine*()!!.*toInt*()  
 *print*("\tIngrese la hora de inicio [HH]: ");  
 val horarioInicio = *readLine*()!!.*toInt*()  
 *print*("\tIngrese el mes de finalización [MM]: ");  
 val mesFin = *readLine*()!!.*toInt*()  
 *print*("\tIngrese el dia de finalización [DD]: ");  
 val diaFin = *readLine*()!!.*toInt*()  
 *print*("\tIngrese la hora de finalización [HH]: ");  
 val horarioFin = *readLine*()!!.*toInt*()  
 *print*("\tObjetivo: ")  
 val objetivo = *readLine*().*toString*()  
 *print*("\tDescripcion: ")  
 val descripcion = *readLine*().*toString*()  
 *print*("\tDependencia interna: ")  
 val dependenciaInterna = *readLine*().*toString*()  
 *print*("\tDependencia externa: ")  
 val dependenciaExterna = *readLine*().*toString*()  
 *print*("\tFrecuencia: ")  
 val frecuencia = *readLine*().*toString*()  
 *print*("\tPrioridad [1(Poco urgente) - 10(Muy urgente)]: ")  
 val prioridad = *readLine*()!!.*toInt*()  
  
 /\*  
 DESCRIPCION: Los datos registrados antes son guardados en un objeto de tipo Tarea. El lapso se calcula de forma  
 automática para que el usuario no necesite ingresar información redundante; además el estado de la tarea se  
 establece automáticamente como Pendiente con un booleano false.  
 \*/  
 val newTarea = Tarea(  
 title,  
 LocalDateTime.of(2021, mesInicio, diaInicio, horarioInicio, 0, 0),  
 LocalDateTime.of(2021, mesFin, diaFin, horarioFin, 0, 0),  
 objetivo, descripcion,  
 *calcularLapso*(  
 LocalDateTime.of(2021, mesInicio, diaInicio, horarioInicio, 0, 0),  
 LocalDateTime.of(2021, mesFin, diaFin, horarioFin, 0, 0)  
 ),  
 false, dependenciaInterna,  
 dependenciaExterna, frecuencia, prioridad  
 )  
 //DESCRIPCION: La tarea creada anteriormente se agrega al final de la lista de tareas.  
 *tareas*.add(newTarea)  
 //DESCRIPCION: Las tareas totales del usuario se establecen como el tamaño de la lista de tareas.  
 *usuario*.tareasTotales = *tareas*.size  
}

## editarTarea()

/\*  
AUTOR: ANGEL OMAR GOMEZ CASTILLO.  
DESCRIPCION: En esta función se registrarán los datos de una tarea ya existente para editarla en la lista de tareas  
pendientes.  
 \*/  
fun editarTarea() {  
 //DESCRIPCION: Las tareas totales del usuario se establecen en cero para contarlas al terminar de agregar la tarea.  
 *usuario*.tareasTotales = 0  
 //DESCRIPCION: Las tareas realizadas del usuario se establecen en cero para contarlas al terminar de editar la tarea.  
 *usuario*.tareasRealizadas  
 *println*(  
 "---------------------------------------------\n" +  
 "Editar tarea:\n"  
 )  
 //DESCRIPCION: Si la lista de tareas está vacía se nos avisa de ello. De lo contrario se solicitan los datos.  
 if (*tareas*.isEmpty()) {  
 *println*("No hay ninguna tarea.")  
 } else {  
 //DESCRIPCION: Se muestran las tareas que existen para que el usuario seleccione la que desea editar.  
 *mostrarTareas*()  
 //DESCRIPCION: Se le solicita al usuario el número de tarea a editar.  
 *print*("Numero de tarea a editar: ")  
 val tareaIndice = *readLine*()!!.*toInt*()  
 *print*("\n\tEscribe el nuevo título para la tarea: ")  
 val tareaEdit = *readLine*().*toString*()  
 *print*("\tIngrese el mes de inicio [MM]: ");  
 val mesInicio = *readLine*()!!.*toInt*()  
 *print*("\tIngrese el dia de inicio [DD]: ");  
 val diaInicio = *readLine*()!!.*toInt*()  
 *print*("\tIngrese la hora de inicio [HH]: ");  
 val horarioInicio = *readLine*()!!.*toInt*()  
 *print*("\tIngrese el mes de finalización [MM]: ");  
 val mesFin = *readLine*()!!.*toInt*()  
 *print*("\tIngrese el dia de finalización [DD]: ");  
 val diaFin = *readLine*()!!.*toInt*()  
 *print*("\tIngrese la hora de finalización [HH]: ");  
 val horarioFin = *readLine*()!!.*toInt*()  
 *print*("Objetivo: ")  
 val objetivo = *readLine*().*toString*()  
 *print*("Descripcion: ")  
 val descripcion = *readLine*().*toString*()  
 //DESCRIPCION: En esta ocasión sí se le solicita el estado de la tarea: Pendiente(false) o Finalizada(true)  
 *print*("Estado de la tarea [Pendiente/Finalizada]: ")  
 val estadoLectura = *readLine*().*toString*()  
 val estadoBoolean = estadoLectura.*equals*("Finalizada", true)  
 *print*("dependenciaInterna: ")  
 val dependenciaInterna = *readLine*().*toString*()  
 *print*("dependenciaExterna: ")  
 val dependenciaExterna = *readLine*().*toString*()  
 *print*("frecuencia: ")  
 val frecuencia = *readLine*().*toString*()  
 *print*("\tPrioridad [1(Poco urgente) - 10(Muy urgente)]: ")  
 val prioridad = *readLine*()!!.*toInt*()  
  
 /\*  
 DESCRIPCION: Los datos registrados antes son guardados en un objeto de tipo Tarea. El lapso se calcula de forma  
 automática para que el usuario no necesite ingresar información redundante; además el estado de la tarea se  
 establece según lo que el usuario haya introducido.  
 \*/  
 val editTarea = Tarea(  
 tareaEdit,  
 LocalDateTime.of(2021, mesInicio, diaInicio, horarioInicio, 0, 0),  
 LocalDateTime.of(2021, mesFin, diaFin, horarioFin, 0, 0),  
 objetivo, descripcion,  
 *calcularLapso*(  
 LocalDateTime.of(2021, mesInicio, diaInicio, horarioInicio, 0, 0),  
 LocalDateTime.of(2021, mesFin, diaFin, horarioFin, 0, 0)  
 ),  
 estadoBoolean, dependenciaInterna,  
 dependenciaExterna, frecuencia, prioridad  
 )  
 //DESCRIPCION: La tarea creada anteriormente se guarda en el índice de la tarea que se está ediando.  
 *tareas*.set(tareaIndice.minus(1), editTarea)  
 *println*("Tarea editada satisfactoriamente.")  
 //DESCRIPCION: Mediante un for y un if se contabilizan las tareas cuyo estado sea Finalizada(true)  
 for (tarea in *tareas*) {  
 if (tarea.estado)  
 *usuario*.tareasRealizadas++  
 }  
 }  
}

## eliminarTarea()

/\*  
AUTOR: JOSE BERNAL FONSECA.  
DESCRIPCION: En esta función se solicita el indice de una tarea que se desee eliminar de la lista de tareas.  
 \*/  
fun eliminarTarea() {  
 //DESCRIPCION: Las tareas totales del usuario se establecen en cero para contarlas al terminar de agregar la tarea.  
 *usuario*.tareasTotales = 0  
 //DESCRIPCION: Las tareas realizadas del usuario se establecen en cero para contarlas al terminar de editar la tarea.  
 *usuario*.tareasRealizadas  
 *println*(  
 "---------------------------------------------\n" +  
 "Eliminar tarea:\n"  
 )  
 if (*tareas*.isEmpty()) {  
 *println*("No hay ninguna tarea")  
 } else {  
 *mostrarTareas*()  
 *print*("Escriba el número de la tarea que desea borrar: ")  
 val delTarea = *readLine*()?.*toInt*()  
 *tareas*.removeAt(delTarea!!.minus(1))  
 *mostrarTareas*()  
 }  
  
 //DESCRIPCION: Las tareas totales del usuario se establecen como el tamaño de la lista de tareas.  
 *usuario*.tareasTotales = *tareas*.size  
 //DESCRIPCION: Mediante un for y un if se contabilizan las tareas cuyo estado sea Finalizada(true)  
 for (tarea in *tareas*) {  
 if (tarea.estado)  
 *usuario*.tareasRealizadas++  
 }  
}

## mostrarTareas()

/\*  
AUTOR: JOSE BERNAL FONSECA.  
DESCRIPCION: En esta función se utiliza un for con la función WithIndex() de la lista de tareas para generar un listado  
con el índice y el título de las tareas que contiene la lista.  
 \*/  
fun mostrarTareas() {  
 for ((i, tarea) in *tareas*.*withIndex*()) *println*("\t${i + 1}) ${tarea.titulo}: ${tarea.lapso}.")  
}

## calcularLapso()

/\*  
AUTOR: BRAULIO DAVID HERNANDEZ PALAGOT.  
DESCRIPCION: En esta función se genera un String que indica el lapso de tiempo que abarcará la tarea. Para ello necesia  
recibir dos argumentos de tipo LocalDateTime: el que indica el inicio y el que indica el fin de la tarea.  
 \*/  
fun calcularLapso(inicio: LocalDateTime, fin: LocalDateTime): String {  
 return ("Desde $inicio hasta $fin")  
}