

**BEDU**

**Desarrollo Móvil**

**Kotlin Fundamentals**

**19:00 – 22:00**

**Proyecto Guía**

**ChronoMaster 2021**

**Presentan:**

**Manuel Díaz Capistran**

**José Bernal Fonseca**

**Ángel Omar Gómez Castillo**

**Braulio David Hernández Palagot**

Contenido

[Introducción 2](#_Toc75384847)

[Definición del proyecto 3](#_Toc75384848)

[Funcionalidades 3](#_Toc75384849)

[Clases 3](#_Toc75384850)

[Código de clases 4](#_Toc75384851)

[Usuario 4](#_Toc75384852)

[Tarea 4](#_Toc75384853)

[Código de funciones 5](#_Toc75384854)

[main() 5](#_Toc75384855)

[bienvenida() 6](#_Toc75384856)

[visualizarPrograma() 7](#_Toc75384857)

[revisarConflictos() 7](#_Toc75384858)

[resumenUsuario() 8](#_Toc75384859)

[agregarTarea() 8](#_Toc75384860)

[editarTarea() 8](#_Toc75384861)

[eliminarTarea() 9](#_Toc75384862)

[mostrarTareas() 9](#_Toc75384863)

# 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 | | editarTarea() | |

# Código de clases

## Usuario

class Usuario(var nombre: String, var contrasenna: String, var tareasTotales: Int = 0, var tareasRealizadas: Int = 0) {  
 init {  
 *println*(  
 "Datos de inicio de sesión:\n" +  
 "\tUsuario: ${this.nombre}.\n" +  
 "\tContraseña: ${this.contrasenna}."  
 )  
 }  
  
 fun loggin(nombre: String, contrasenna: String): Boolean {  
 return nombre.equals(this.nombre) && contrasenna.equals(this.contrasenna)  
 }  
}

## Tarea

class Tarea(var titulo: String, var fechaInicio: LocalTime, var fechaFinalizacion: LocalTime, 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) {  
 fun editarTarea() {  
  
 }  
}

# Código de funciones

## main()

import java.time.LocalTime  
  
val *tareas* = *mutableListOf*<String>()  
val *horariosTareas* = *mutableListOf*<String>() //Lista mutable para guardar los horarios de cada tarea  
  
fun main() {  
 *bienvenida*()  
}

## bienvenida()

fun bienvenida() {  
 var op = 0  
 var inputNombre = ""  
 var inputContrasenna = ""  
 val usuario = Usuario("Usuario de prueba", "contraseña\_de\_prueba")  
  
 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))  
  
  
 *println*("\nBienvenido a tu agenda, ${usuario.nombre}.")  
  
 do {  
 *print*(  
 "---------------------------------------------\n" +  
 "Menú principal:\n"  
 )  
 *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."  
 )  
 *print*("Seleccione una opcion: ")  
 op = *readLine*()!!.*toInt*()  
  
 when (op) {  
 1 -> *visualizarPrograma*()  
 2 -> *revisarConflictos*()  
 3 -> *resumenUsuario*()  
 4 -> *AgregarTarea*()  
 5 -> *editarTarea*()  
 6 -> *eliminarTarea*()  
 7 -> *print*("Hasta la próxima, ${usuario.nombre}.")  
 else -> {  
 *print*("Opción no valida.")  
 }  
 }  
 } while (op != 7)  
}

## visualizarPrograma()

fun visualizarPrograma() {  
 *println*(  
 "---------------------------------------------\n" +  
 "Programa del día:\n"  
 )  
 if (*tareas*.isEmpty()) {  
 *println*("No hay ninguna tarea.")  
 } else {  
 *mostrarTareas*()  
 }  
}

## revisarConflictos()

fun revisarConflictos() {  
 val setConflictos = *mutableSetOf*<String>()  
 *println*(  
 "---------------------------------------------\n" +  
 "Revisión de conflictos:\n"  
 )  
 for ((i, horarioComparando) in *horariosTareas*.*withIndex*()) {  
 for ((j, horarioComparador) in *horariosTareas*.*withIndex*()) {  
 if (i != j) {  
 if (horarioComparando.*contains*(horarioComparador.*substring*(0, 5)) ||  
 horarioComparando.*contains*(horarioComparador.*substring*(8))  
 )  
 setConflictos.add(  
 "La tarea ${*tareas*[i]}, con horario: $horarioComparando\n" +  
 "entra en conflicto con la tarea ${*tareas*[j]}, con horario: $horarioComparador.\n" +  
 "Es necesario editar alguna de las dos."  
 )  
 }  
 }  
 }  
 if (!setConflictos.isEmpty())  
 for ((i, elemento) in setConflictos.*withIndex*())  
 *println*(" ${i + 1}) $elemento\n")  
 else  
 *println*("No se encontraron conflictos")  
}

## resumenUsuario()

fun resumenUsuario() {  
 *println*(  
 "---------------------------------------------\n" +  
 "Resumen del usuario:\n"  
 )  
}

## agregarTarea()

fun AgregarTarea() {  
 *println*(  
 "---------------------------------------------\n" +  
 "Agregar tarea:\n"  
 )  
 *print*("Ingrese el título de la tarea: ");  
 val title = *readLine*().*toString*()  
 *tareas*.add(title)  
 *print*("\tIngrese la hora de inicio [HH:mm]: ");  
 var horario: String = *readLine*().*toString*()  
 val horarioInicio: LocalTime = LocalTime.of(horario.*substring*(0,2).*toInt*(), horario.*substring*(3).*toInt*())  
 *print*("\tIngrese la hora de finalización [HH:mm]: "); horario = *readLine*().*toString*()  
 val horarioFin: LocalTime = LocalTime.of(horario.*substring*(0,2).*toInt*(), horario.*substring*(3).*toInt*())  
  
 horario = "$horarioInicio - $horarioFin"  
 *horariosTareas*.add(horario)  
}

## editarTarea()

fun editarTarea() {  
 var horario = ""  
 *println*(  
 "---------------------------------------------\n" +  
 "Editar tarea:\n"  
 )  
 if (*tareas*.isEmpty()) {  
 *println*("No hay ninguna tarea.")  
 } else {  
 *mostrarTareas*()  
 *print*("Numero de tarea a editar: ")  
 val tareaIndice = *readLine*()  
 *print*("Escribe el nuevo título para la tarea: ")  
 val tareaEdit = *readLine*()  
 *print*("\tIngrese la hora de inicio [HH:mm]: "); horario += *readLine*().*toString*()  
 horario += " - "  
 *print*("\tIngrese la hora de finalización [HH:mm]: "); horario += *readLine*().*toString*()  
  
 *tareas*.set(tareaIndice?.*toInt*()!!.minus(1), tareaEdit.*toString*())  
 *horariosTareas*.set(tareaIndice?.*toInt*()!!.minus(1), horario)  
 *println*("Tarea editada satisfactoriamente.")  
 }  
}

## eliminarTarea()

fun eliminarTarea() {  
 *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*()  
 }  
}

## mostrarTareas()

fun mostrarTareas() {  
 for ((i, tarea) in *tareas*.*withIndex*()) *println*("\t${i + 1}.- $tarea")  
}