|  |  |  |
| --- | --- | --- |
| **CARRERA:**  Ingeniería en Sistemas e informática | **GUÍA**  No. 01 | **TIEMPO ESTIMADO:**  1h y 20 min. |
| **ASIGNATURA:**  Programación Móvil | **FECHA DE ELABORACION:** 25-06-20  **SEMESTRE**: Mayo 2020 – Agosto 2020 | |
| **TÍTULO:**  Calculadora Científica | **DOCENTE: Ing. Fernando Solís** | |

**INTEGRANTES:**

* **Dennys Iza**
* **Capt. Angos Maria**

**OBJETIVO**

Creación de una Calculadora científica como aplicación móvil

**INSTRUCCIONES**

1. Utilice como material principal, aquel indicado en clase por el docente.
2. Utilice información consultada en Internet y conocimiento adquirido en clase.

**ACTIVIDADES**

1. **Ubicación de recursos**
2. Formar grupos de máximo 2 personas por computador
3. Instalar la herramienta Android Studio
4. **Planteamiento del problema**

Desarrollo de una Calculadora Científica para un dispositivo móvil, utilizando todo el material entregado en clase, los conocimientos previamente obtenidos

1. **Entregable (s)**

**COMPUTACION MOVIL**

**Introducción**

En muchas obras de ciencias ficción es frecuente encontrarse con personajes que utilizan ciertos aparatos que con sólo oprimir un botón los transportan por los aires a gran velocidad, especies de cinturones a propulsión. Otros que con el uso de control remoto son capaces de controlar lo que sucede en la casa. Abren y cierran la puerta del garaje, mandan la comida al horno y hasta poseen un perro mecánico que trae las pantuflas cundo llegan del trabajo y todo con el uso de la tecnología, que a cada autor de ciencias ficción se le ocurre de una manera particular.

Los que tienen en común estas obras que se leen en libros, suplementos y revistas o se ven televisor o el cine es el uso de una tecnología desconocida por el espectador, pero que sin embargo su imaginación no la cree imposible de alcanzar en el futuro.

Quien se imaginaba en los años sesenta que muchas situaciones presentes en los comics "Los Supersónicos" hoy las pudiéramos vivir. Quien duda hoy que la clonación masiva de seres humanos que presenta Adolf Huxley en "Un Mundo Perfecto" puede suceder a mediano plazo. Y a quien se le ocurriría tratar de loco hoy a Isaac Azimov, como lo hicieron cuando planteó la existencia de cajeros automáticos en una de sus novelas de los años 50"

La cuestión es que muchas cosas que se imaginaban imposibles hace poco tiempo, hoy son realidades o se intuye que muy pronto se podrán alcanzar con el uso de la tecnología. En la actualidad no resulta extraño hallar personas en el metro haciendo llamadas locales o de larga distancia haciendo uso de la telefonía celular, o adelantando algún trabajo con el uso de un laptop.

Estos dos ejemplos, comunes para todos, es sólo una muestra de lo que se llama Computación Móvil y que en este trabajo buscamos aclarar. Esta investigación expone en forma generar la que se entiende por Computación Móvil, analiza algunos usos conocidos y explica su tendencia en el mundo y especialmente Venezuela y América Latina.

**2. Computación Móvil**

Se puede definir la Computación Móvil como la serie de artefactos y equipos portátiles, hardware, que hacen uso de la computación para lograr su funcionamiento, así, se tiene a las computadoras portátiles, los teléfonos celulares, los cuadernos de notas computarizados, las calculadoras de bolsillo, etc.

Aunque en la actualidad no son muchas las personas que traen su computadora bajo el brazo, tampoco son raras, y es que la computación móvil ha dejado de ser un instrumento de presunción, para convertirse en un medio de trabajo. Pero eso no es todo: empieza transformarse en EL medio de trabajo. Ya es parte del uniforme del profesionales, del vendedor y hasta de los repartidores de refrescos que con él controlan el inventario de su camión.

**3. Algunas Aplicaciones de la Computación Móvil**

A pesar de que las novelas de ciencia ficción lo anticipaban, no deja de sorprender la diversidad de aplicaciones que tiene hoy en día —y puede tener en el futuro— esta tecnología. Hay varios campos en los que no sólo es de utilidad, sino de competencia esencial para las empresas. Aquí algunos ejemplos:

Manejo de pacientes. La computación móvil permite al médico o institución mantener contacto con un paciente cuyo estado requiere continua vigilancia. Y esto no se limita al envío o recepción de mensajes (hablados o escritos), sino que incluye también el monitoreo constante de signos vitales críticos que pueden anticipar una emergencia.

Ventas directas. Posiblemente ésta sea la aplicación más evidente: poder consultar inventarios, precios y realizar pedidos en forma inmediata resulta de particular interés para cualquier empresa que se dedique a la comercialización de productos.

Servicio a clientes. La asesoría, servicio técnico y consultoría es una área en donde la computación móvil es vital. La consulta a bancos de información, bases de datos inteligentes, acopio de información actualizada y consulta de especialistas, es sólo una pequeña muestra de todo lo que puede impactar esta tecnología, sin mencionar la posibilidad de contacto permanente con el cliente.

Personal móvil en oficinas. No es raro encontrar a personal que, pese a encontrarse siempre en el mismo edificio, se mudan de lugar con frecuencia para, por ejemplo, dar soporte técnico al personal o revisar proyectos. La computación móvil no sólo les permite ser localizados con facilidad, sino que también le auxilia en la consulta de datos que por lo regular estarían en su oficina.

Profesionales viajeros. Los modernos guerreros del camino son usuarios potenciales que pueden aprovechar esta herramienta: contadores con los registros de una empresa bajo el brazo, gerentes regionales que integran metas empresariales, dirigentes corporativos que requieren información actualizada y, en fin, una gran cantidad de personas que necesitan viajar para realizar su trabajo.

Manejo de sucursales. En un mundo en el que las empresas han dejado de tener una sola oficina para expandirse, las sucursales aparecen casi sin desearlo, con una gran cantidad de datos que deben consolidarse.

Grupos de trabajo. La globalización y expansión de empresas hace que sea cada vez más común atacar proyectos con el personal adecuado, el cual no siempre trabaja bajo un mismo techo y, en ocasiones, ni siquiera en la misma ciudad o país.

**4. Tendencias de la Computación Móvil**

En fin, nunca se terminaría de enumerar el enorme número de áreas en que esta forma de computación puede cobrar una importancia vital. Pero no se crea tan a futuro, sólo en Canadá se reportaron durante 1993 ventas por $1,000 millones de dólares y —según datos de Datacomm Research—, las ventas en Estados Unidos llegarán a $3,700 millones de dólares en el año 2002 para la industria de computación móvil, en la que se incluye equipo, programas e infraestructura de comunicaciones (inalámbrica y servicios de redes).

Seguramente México experimentará una expansión proporcional de dicho mercado, nicho que ya empieza a ser atacado por algunas empresas. Sin embargo, en general representa una oportunidad de negocios en un mercado muy poco competido hasta ahora.

Aunque mucha gente se imagina que esta tecnología consiste sólo en comprar una notebook y llevarla a todas partes con los datos necesarios, el concepto va mucho más allá: se está hablando de enlazar estos equipos de forma constante o por demanda, para tener acceso bidireccional a información empresarial y de consulta

Una de las realidades de trabajar fuera de la oficina es que el flujo de información mensajes de correo, documentos, llamadas telefónicas no cesan una vez que el trabajador deja el escritorio. La computación móvil ha cambiado la naturaleza del trabajo para profesionales móviles.

Aunque en Venezuela ha sido lenta la asimilación de esta tecnología al máximo o al día del comportamiento mundial del fenómeno, la tecnología de los computadores portátiles marcha en vías de proporcionar el acceso remoto a información corporativa, para permitir a los usuarios "llevar su escritorio" con ellos cuando viajan, transformando los aeropuertos, hoteles y automóviles en espacios de trabajo con información actualizada al último minuto.

**5. Últimos Avances de la Computación Móvil**

Los servicios de replicación bi-direccional de Notes crean una plataforma ideal para el trabajo móvil. Los usuarios Notes pueden asegurar que el trabajo que efectúan fuera de su oficina no salga de la sincronización con respecto al trabajo que están efectuando sus colegas. Los usuarios Notes pueden replicar aplicaciones mientras están lejos del lugar de trabajo, con el fin de sincronizar su base de datos portátil con el servidor para obtener información al minuto.

Notes permite a los usuarios obtener acceso a esa información estando fuera de la oficina, es decir, "extraerla" del servidor para su computador de cliente móvil. Con el fin de extraer esta información, los usuarios móviles Notes simplemente inician una comunicación con un servidor Notes, utilizando una línea telefónica. La tecnología de la replicación proporciona a los usuarios móviles la habilidad de extraer información para agilizar el flujo de comunicación a través de los grupos de trabajo cuando están fuera, proveer información al día a clientes y socios de negocios, asegurarse de que esté disponible la información apropiada necesaria para completar un contrato u orden de compra y enviar mensajes desde cualquier parte.

Por el contrario, los mensajes y eventos importantes raramente siguen una programación que corresponda a las conexiones manuales del trabajador para un servidor. Los trabajadores viajeros están todavía en desventaja si no son capaces de estar informados fácilmente y sin esfuerzo de las noticias urgentes. Es decir, los usuarios necesitan información relevante que les ha sido proporcionada automáticamente por el servidor. Los servicios de buscapersonas (pagers) Notes trabajan para mantener a los usuarios móviles informados sobre mensajes y eventos importantes.

**6. La Computación Móvil y las Comunicaciones, Asunto de Competencia**

El mercado de las redes inalámbricas

Resultaría ilógico pensar en la computación móvil con un largo cable de conexión, así que la comunicación inalámbrica juega un papel preponderante en este concepto, para ello muchas empresas han realizado esfuerzos por proporcionar los recursos necesarios para hacerlo posible. El proveedor más grande y conocido en este sentido es Skytel, el cual proporciona en Estados Unidos no sólo servicios de mensajería, sino que además cuenta con la opción de recibir mensajes de cualquier correo electrónico (MCI, Compuserve, etcétera.) que cumpla con el estándar X.400. Sin embargo, no está solo, Motorola entró ya de lleno a la competencia y, mediante el sistema EMBARC (Electronic Mail Broadcasting to a Roaming Computer), ofrece servicios similares. Si en el área anterior son competidores, en otras son socios. Ese es el caso del sistema Newstream que Skytel comercializa con el nombre de Skystream. Este sistema ofrece a los suscriptores la recepción de información proveniente de servicios como el diario USA Today o la agencia Reuter. El sistema consta de un receptor portátil que se conecta al puerto serial (interfaz RS232C) que algunos fabricantes de equipo de cómputo ofrecen como opción. Entre ellos están: HewlettPackard (95LX y 100LX), Casio (PDA Z7000) y Apple (PowerBook). Pero si la comunicación en un sentido está competida, en la de dos vías hay una verdadera lucha encarnizada entre los sistemas de ARDIS y RAM Mobile Data, los cuales ofrecen servicios de redes de el área metropolitana inalámbricas (MAN) mediante receptores transmisores que se conectan a una notebook o palmtop compatible con un sistema o con ambos, aunque también existen equipos dedicados como el InfoTAC de Motorola.

Ambos operan sobre la banda de 900 MHz y transmiten paquetes de 240 a 500 bytes, la diferencia está en las velocidades de conexión. Mientras ARDIS lo hace a 19.2 Kbps, RAM lo realiza a 8 Kbps. Estos sistemas resultan apropiados para los representantes de ventas, servicio en sitio y consultoría básica, dado que se enfocan a la transferencia de pequeños paquetes de información. Sin embargo, la solución para las limitaciones de los sistemas descritos es el desarrollado por las grandes corporaciones de comunicaciones (GTE, Mobilnet/Contel Cellular, AT&T, McCaw Cellular, Nynex Mobile Communications y Pactel Cellular, entre otros) y bautizado como CDPD (Celullar Digital Packet Data), el cual aprovecha los tiempos muertos de la telefonía celular, en los que no se está transmitiendo voz.

La velocidad de transferencia es de 19.2 Kbps, pero utiliza la infraestructura existente y permite el uso de fax/modems convencionales en lugar de equipos especiales (sin capacidad de fax, por ejemplo), lo que asegura el abaratamiento del sistema para el usuario final. Actualmente, algunos fabricantes de equipo ofrecen la opción de CDPD en sus productos, como IBM en su Thinkpad. Pero no todo son redes metropolitanas en las comunicaciones inalámbricas, algunas aplicaciones no van más allá del área de un edificio o construcción cercana. Simplemente el ahorrarse el cableado es un buen motivo para moverse a esta tecnología. Existen dos formas de resolver este problema: por medio de comunicación electromagnética (ondas de radio) o de luz.

El mercado de las redes inalámbricas no es nada despreciable, según estimaciones este sector alcanzará ventas mundiales hasta por $500 millones de dólares en el año 2000, lo cual de nuevo representa oportunidades de negocio.

**7. Hardware**

Pero no todo son comunicaciones, existe también el equipo en sí. En este sentido se encuentra uno de los problemas más sensibles de la computación móvil, ya que existen compromisos entre la capacidad de los equipos, su tamaño y peso, la facilidad de uso y la alimentación continua de energía.

El equipo debe ser pequeño y liviano para transportarse, pero en el momento de usarlo debe ser suficientemente grande para poder operarlo con facilidad, contradictorio ¿No? Y qué decir de la capacidad. El usuario exige grandes almacenamientos masivos, memoria volátil, capacidad de comunicación, pantallas a color y velocidad de procesamiento, todo eso consume grandes cantidades de energía, lo cual incide sobre el tamaño de las baterías y el tiempo de uso. Aunque existen grandes adelantos en este sentido, sigue siendo uno de los puntos más críticos de solución para los fabricantes de equipo.

La existencia en el mercado de asistentes digitales personales (PDA), cargadores solares, tarjetas PCMCIA con prácticamente todas las aplicaciones posibles, teclados extensibles en notebooks, fax/modems y hasta sistemas en multimedia son una prueba de esta preocupación. Basta abrir una revista especializada para darse cuenta de la gran cantidad de accesorios que surgen día a día para solventar este problema. Y es que no existe la solución total, cada usuario tiene necesidades particulares que requieren equipos específicos, un PDA será una mala inversión para un escritor y una notebook pésima opción para un almacenista que realiza inventarios.

**8. Seguridad**

Este es otro punto delicado de la computación móvil, porque el concepto de seguridad es muy amplio y depende del punto de vista. Mientras para el empresario significa el control del acceso remoto y la confidencialidad de su información, el usuario lo centra en la actualización de los datos en su terminal (parece raro el enfoque, pero si se piensa en un vendedor que requiere los precios y existencias actualizadas), al administrador de sistemas le preocupa la integridad de información en su servidor —cosa de la que debe encargarse las aplicaciones apoyado en las comunicaciones— y para todos ellos se encuentra latente la posibilidad de robo del equipo terminal, lo cual no es precisamente una situación rara o casual, la pérdida de notebook o PDA es uno de los problemas más comunes a los que se enfrenta cualquier usuario.

Para mostrar qué tantas variantes tiene este concepto, Kantek Inc. ofrece un filtro por $199 dólares que distorsiona las imágenes y el texto cuando se mira la pantalla en forma diagonal, de manera que sólo el usuario que tiene de frente el equipo pueda ver la información desplegada, con lo que puede trabajar en lugares públicos, como aeropuertos o el metro, sin temor a que un intruso pueda ver su trabajo.

Si, adicionalmente, no se cuenta con los recursos de comunicaciones, entonces habrá que tener especial cuidado en la sincronización de datos, es decir, que la información sea la misma al menos la mayor parte del tiempo. Esta es una tarea trivial si sólo una persona accede a la información, dado que sólo deberá revisarse la actualización más cercana de los archivos entre la computadora fija y la móvil, pero cuando varias personas actualizan información al mismo tiempo, el asunto se complica y deberán establecerse reglas estrictas en el juego, responsabilizando a cada usuario de un juego de datos y reflejándolo en el desarrollo de las aplicaciones.

**9. Software**

Los conceptos anteriores se identifican con facilidad, pero los programas son un elemento de primordial importancia en la computación móvil y, posiblemente, la semilla que permita su crecimiento o muerte. En este sentido, se ponen en juego muchos conceptos que deberán trabajar en armonía para proporcionar al usuario un ambiente propicio en el que pueda aprovechar toda la infraestructura de la que se ha hablado hasta el momento.

Así, se requiere especial atención en:

Las aplicaciones. Éstas deberán ser prácticamente diseñadas exprofeso para aplicarse bajo esta tecnología. Hay que recordar que en mucho depende de las comunicaciones y éstas no son tan rápidas como para emular la transferencia local de información.

Así que deberán estar basadas en arquitecturas cliente/servidor para que el tráfico sea lo estrictamente indispensable para actualizar información y el equipo terminal o remoto deberá encargarse de hacerlo presentable al usuario en su pantalla o impresión local.

El sistema operativo. Este punto es complemento del anterior, aunque no es estrictamente indispensable que el sistema operativo otorgue facilidades como multiproceso y multitareas, la utilización de un ambiente estándar le permite a los fabricantes de programas y aplicaciones ofrecer productos que cumplan con las expectativas de los usuarios.

El ambiente estándar. Es parte de lo mismo, el ambiente en que las aplicaciones trabajan debe ser lo más estándar posible, de manera que las aplicaciones que el usuario utiliza puedan comunicarse entre sí para complementar información, tanto con independencia de la diversidad de fuentes como de si el proveedor de aplicaciones es el mismo o son diferentes fabricantes.

Aquí es donde los ambientes gráficos toman importancia, ya que se necesita una interfaz coherente, estándar y que facilite el desarrollo de sistemas clienteservidor. Y no se habla sólo del ambiente de trabajo, sino de sistemas operativos con facilidades adicionales para la computación móvil, es decir, que no sólo otorgue facilidades de multitarea y multiproceso, sino también soporte dispositivos especiales, ahorro de energía, seguridad y comunicaciones que requiere la computación móvil, facilitando la vida tanto del usuario como del personal que lo desarrolla.

Algo no debe pasarse por alto: el ambiente debe ser transportable a cualquier equipo que esté dentro de este concepto. Y esto no significa sólo laptops y notebooks, sino que también abarca PDA, estaciones de consulta, equipos portátiles de mensajes y todo aquel producto susceptible de introducirse en este concepto en el futuro. En él se incluyen cajas permanentes, terminales de tarjetas de crédito, cajas registradoras, centros de mensaje, periódicos permanentes, entre otros.

**ANDROID STUDIO**

Android Studio es el entorno de desarrollo integrado oficial para la plataforma Android. Fue anunciado el 16 de mayo de 2013 en la conferencia Google I/O, y reemplazó a Eclipse como el IDE oficial para el desarrollo de aplicaciones para Android. La primera versión estable fue publicada en diciembre de 2014.

Está basado en el software IntelliJ IDEA de JetBrains y ha sido publicado de forma gratuita a través de la Licencia Apache 2.0. Está disponible para las plataformas Microsoft Windows, macOS y GNU/Linux. Ha sido diseñado específicamente para el desarrollo de Android.

Estuvo en etapa de vista previa de acceso temprano a partir de la versión 0.1, en mayo de 2013, y luego entró en etapa beta a partir de la versión 0.8, lanzada en junio de 2014. La primera compilación estable, la versión 1.0, fue lanzada en diciembre de 2014.2​

Desde el 7 de mayo de 2019, Kotlin es el lenguaje preferido de Google para el desarrollo de aplicaciones de Android.3​ Aún así, Android Studio admite otros lenguajes de programación, como Java y C ++.



**Características**

Se espera que se desarrollen nuevas funciones con cada versión de Android Studio. Las siguientes características se proporcionan en la versión estable actual

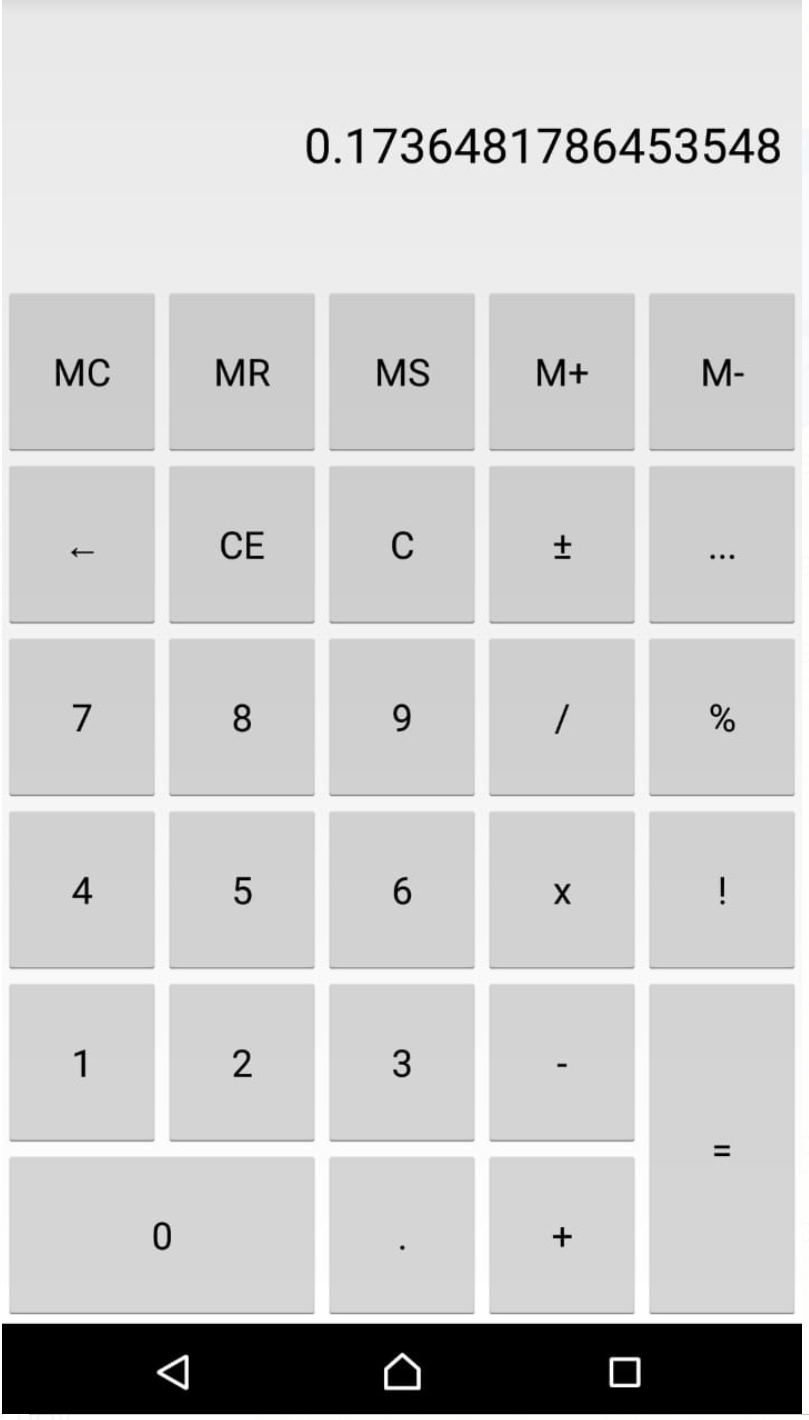
* Soporte para construcción basada en Gradle.
* Refactorización específica de Android y arreglos rápidos.
* Herramientas Lint para detectar problemas de rendimiento, usabilidad, compatibilidad de versiones y otros problemas.
* Integración de ProGuard y funciones de firma de aplicaciones.
* Plantillas para crear diseños comunes de Android y otros componentes.
* Un editor de diseño enriquecido que permite a los usuarios arrastrar y soltar componentes de la interfaz de usuario.7​
* Soporte para programar aplicaciones para Android Wear.
* Soporte integrado para Google Cloud Platform, que permite la integración con Firebase Cloud Messaging (antes 'Google Cloud Messaging') y Google App Engine.8​
* Un dispositivo virtual de Android que se utiliza para ejecutar y probar aplicaciones.
* Renderizado en tiempo real.
* Consola de desarrollador: consejos de optimización, ayuda para la traducción, estadísticas de uso.

Android Studio admite los mismos lenguajes de programación de IntelliJ (y CLion), como Java, C++ y más con extensiones, como Go;9​ y Android Studio 3.0 o posterior es compatible con Kotlin10​ y "todas las características de lenguaje Java 7 y un subconjunto de características de lenguaje Java 8 que varían según la versión de la plataforma".11​ Los proyectos externos soportan algunas características de Java 9.12​ Si bien IntelliJ en el que se basa Android Studio admite todas las versiones de Java lanzadas, y Java 12, no está claro en qué nivel Android Studio admite versiones de Java hasta Java 12 (la documentación menciona el soporte parcial de Java 8). Al menos algunas características nuevas del lenguaje hasta Java 12 se pueden usar en Android

**CALCULADORA CIENTIFICA**

Una Calculadora Científica. es un tipo de la calculadora electrónica, por lo general, aunque no siempre de mano, diseñado para calcular los problemas de la ciencia, la ingeniería y las matemáticas. Ellos han reemplazado casi por completo las reglas de cálculo en casi todas las aplicaciones tradicionales, y son ampliamente utilizados en la educación y la actividad profesional. En ciertos contextos, como la educación superior, calculadoras científicas han sido sustituidos por las calculadoras gráficas, que ofrecen un superconjunto de la funcionalidad calculadora científica, junto con la capacidad de datos de entrada de gráficos y escribir y almacenar programas para el dispositivo. También existe cierta superposición con el mercado de las calculadoras financieras.

**CALCULADORA CIENTIFICA EN ANDROID**



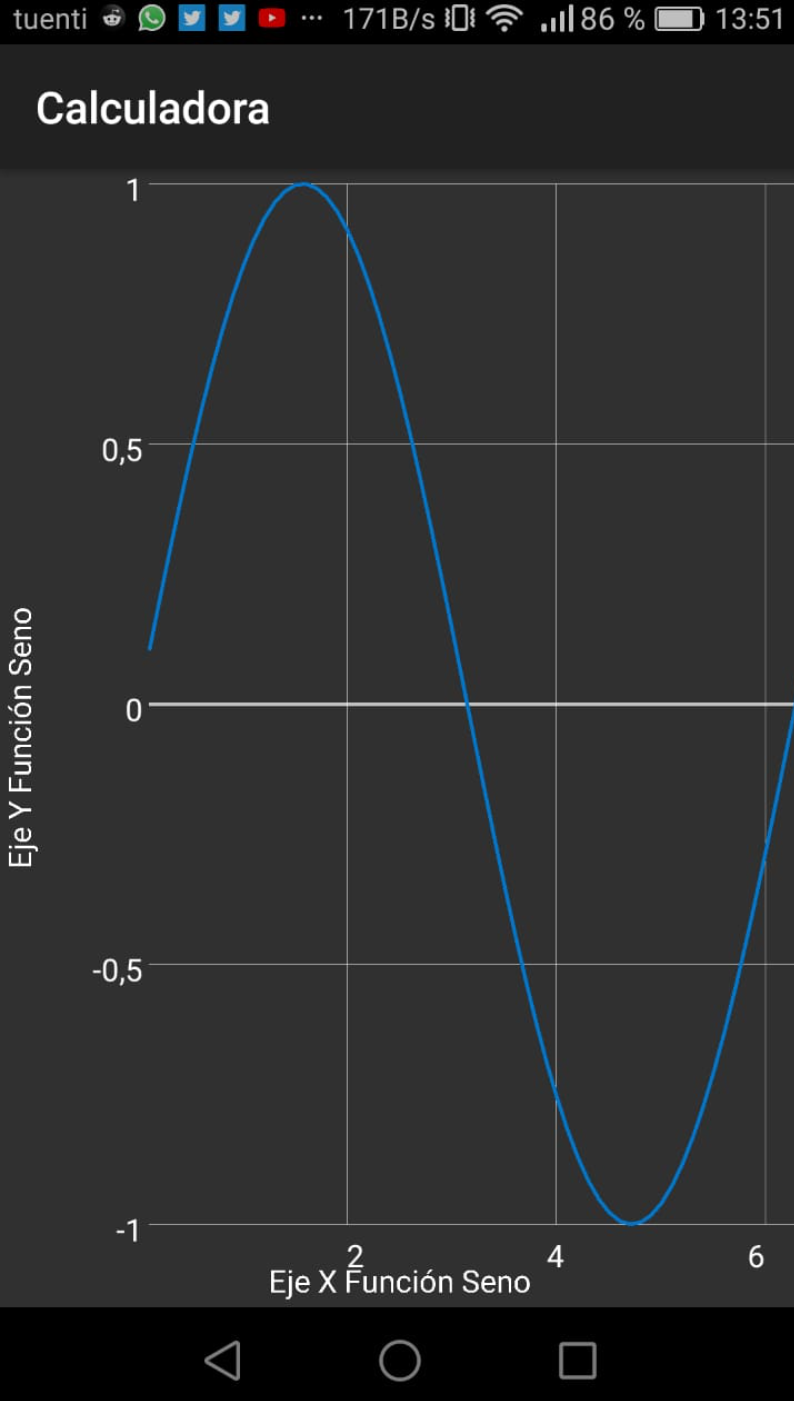
Dentro de las funciones que hemos desarrollado son la básicas:

* Sumar
* Restar
* Dividir
* Multiplicar

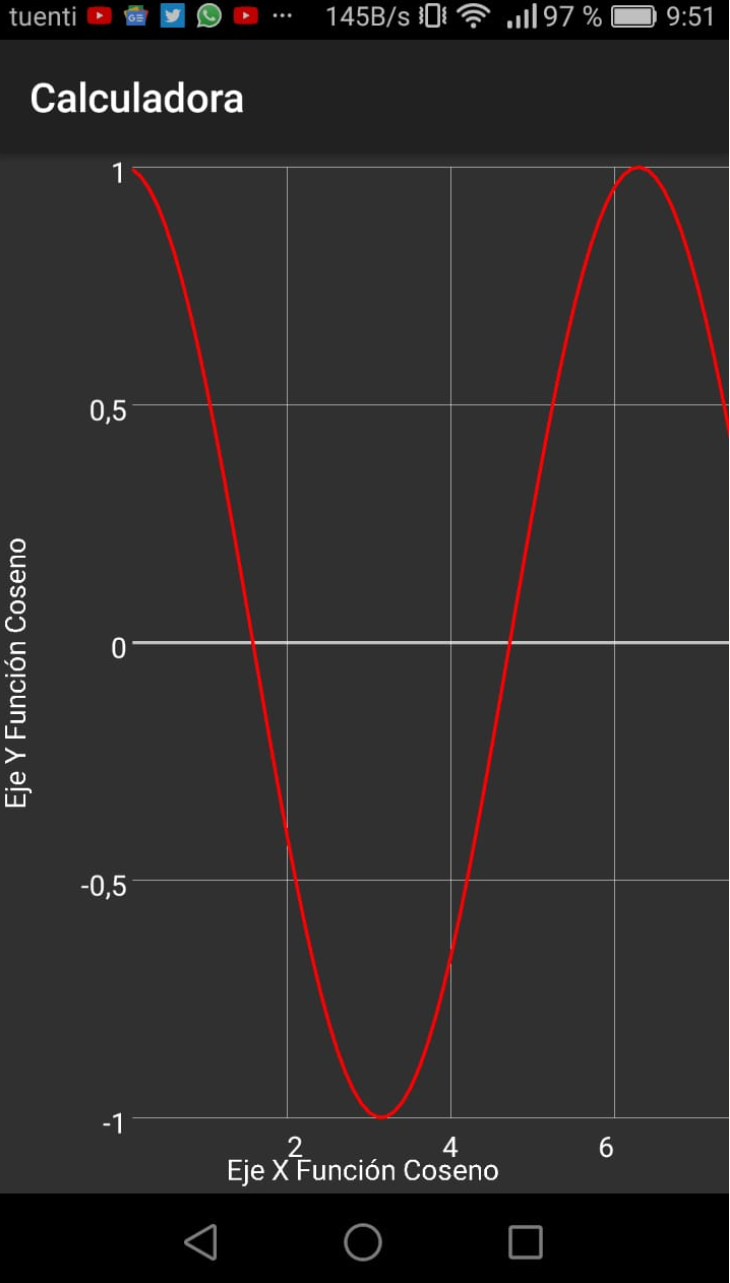
Funciones Científicas

* M+
* M-
* Mr
* Ms
* Mc
* Funciones logarítmicas
* Seno
* Coseno
* Tangente

Graficación de Seno y coseno



Graficacion del Coseno:



Código:

Activity Main

public class ActivityMain extends Activity

{

private CalculatorLCD LCD; // Variable de tipo CalculatorLCD que incluye tres TextView y métodos específicos para tratar el texto y los números o información almacenados en él.

private BigDecimal memoryM = new BigDecimal(0.0F); // Para almacenar la memoria de los botones MC, MR, MS, M+ y M-

private BigDecimal memoryLCD = new BigDecimal(0.0F); // Para almacenar el primer numero de la operacion (el segundo esta en pantalla)

private Character lastOperator = ' '; // Ultima operacion ( + - \* / % ) realizada.

private String lastKeyPressed = " "; // El ultimo boton pulsado (no incluye todos, ver codigo)

private Integer numberCharacterLCD = 16;

private Character lastConvert = 'd';

SettingsCalc setting; // Clase de configuraciones que facilita ciertas propiedades, como la de vibracion, desde un fichero.

Button graficos;

Button graficos2;

/\*\*

\* En el momento de crearse el activity se carga en pantalla el layaout, después las referencias y configuraciones.

\*/

@Override

public void onCreate(Bundle savedInstanceState)

{

// Se carga el layout del activity y se elimina la barra del titulo.

super.onCreate(savedInstanceState);

this.requestWindowFeature(Window.FEATURE\_NO\_TITLE);

this.setContentView(R.layout.activity\_main);

graficos=(Button) findViewById(R.id.main\_btn\_G);

graficos.setOnClickListener(new View.OnClickListener() {

@Override

public void onClick(View v) {

startActivity(new Intent(ActivityMain.this,GraphActivity.class));

}

});

graficos2=(Button) findViewById(R.id.main\_btn\_GC);

graficos2.setOnClickListener(new View.OnClickListener() {

@Override

public void onClick(View v) {

startActivity(new Intent(ActivityMain.this,ParametersGraphics.class));

}

});

// Se obtiene una referencia a la pantalla de la calculadora de tipo CalculatorLCD:

LCD = (CalculatorLCD)findViewById(R.id.main\_CalculatorLCD);

// Se carga la Clase de configuraciones y se obtienen un HashMap que contiene la mayoria de

// propiedades de la clase ActivityMain.

setting = new SettingsCalc(this);

HashMap< String, Object > hm = SettingsCalc.getData();

// Una de las configuraciones que permite la clase de configuración es establecer si

// se quiere la barra de notificaciones o no:

if( !SettingsCalc.isShowNotificationBar() )

this.getWindow().setFlags(WindowManager.LayoutParams.FLAG\_FULLSCREEN, WindowManager.LayoutParams.FLAG\_FULLSCREEN);

// También, si el usuario ha indicado que quiere guardar la ultima operación realizada

// (aun incluso si apaga el móvil), se cargan todas las propiedades de la clase

// (las del HashMap).

if( SettingsCalc.isRememberLastResult() && hm.size() > 0 )

{

LCD.setMemory((Boolean)hm.get("LCDgetMemory"));

LCD.addHistory((String)hm.get("LCDgetHistory"));

LCD.setOperation((String)hm.get("LCDgetOperationString"));

memoryLCD = (BigDecimal)hm.get("memoryLCD");

memoryM = (BigDecimal)hm.get("memoryM");

lastOperator = (Character)hm.get("lastOperator");

lastKeyPressed = (String)hm.get("lastKeyPressed");

lastConvert = (Character)hm.get("lastConvert");

}

}

/\*\*

\* Para modificar desde java los Controles del XML antes se deben de haber creado en pantalla.

\* Desde el método onWindowsFocusChanged se puede hacer:

\* Expande los botones de la calculadora para que ocupen toda la pantalla. Ante la imposibilidad

\* de ajustar el tamaño de los botones desde el XML se ha tenido que recurrir a realizar ese

\* ajuste en tiempo de ejecución desde un método llamado desde el onCreate. Dicho método calcula

\* el tamaño de la pantalla y la posición en la que se encuentra, de este modo, y sabiendo

\* de antemano el numero de filas y columnas, se reparte la pantalla entre todos los botones,

\* en el caso del botón 0 e = el tamaño es el doble por lo que se ha tenido que tener en cuenta.

\* Para recuperar todos los botones en vez de recuperarlos uno a uno de forma manual

\* se ha hecho recuperando el layaut y después todos sus hijos.

\*/

@Override

public void onWindowFocusChanged(boolean hasFocus)

{

super.onWindowFocusChanged(hasFocus);

if( hasFocus )

{

// Una de las configuraciones que permite la clase de configuración es

// establecer si se quiere la barra de notificaciones o no:

if( !SettingsCalc.isShowNotificationBar() )

this.getWindow().setFlags(WindowManager.LayoutParams.FLAG\_FULLSCREEN, WindowManager.LayoutParams.FLAG\_FULLSCREEN);

// Se recupera el tamaño del activity principal y las filas y columnas que tiene el GridLayout:

GridLayout layout = (GridLayout)findViewById(R.id.main\_gridlayout);

int height = layout.getHeight();

int width = layout.getWidth();

int row = layout.getRowCount();

int column = layout.getColumnCount();

// Se calculan las medidas para cada Control de la pantalla:

int heightMemory = (int)( height - ( height / 1.10 ) );

int heightHistory = (int)( height - ( height / 1.10 ) );

int heightOperation = (int)( height - ( height / 1.15 ) );

int heightBotones = (int)( height - heightOperation - heightHistory );

int heightOneButton = (int)( heightBotones / ( row - 1 ) );

int widthOneButton = (int)( width / column );

// Ahora se recupera los TextView y EditText que forman la pantalla y se les cambia el tamaño:

TextView LCDMemory = (TextView)findViewById(R.id.tvc\_label\_memory);

TextView LCDHistory = (TextView)findViewById(R.id.tvc\_label\_history);

EditText LCDOperation = (EditText)findViewById(R.id.tvc\_label\_operation);

LCDMemory.setHeight(heightMemory);

LCDHistory.setHeight(heightHistory);

LCDOperation.setHeight(heightOperation);

//Cambia el tamaño de la letra:

LCDMemory.setTextSize(15);

LCDHistory.setTextSize(15);

LCDOperation.setTextSize(30);

// Por ultimo se recorren todos los hijos o controles contenidos en el layout del

// activity para ajustar el tamaño de los botones:

for( int count = 0 ; count < layout.getChildCount() ; count++ )

{

View v = layout.getChildAt(count);

// Si el elemento actual, definido como de tipo View, es una instancia de un tipo

// Button se procederá a modificar sus dimensiones:

if( v instanceof Button )

{

Button vB = (Button)v;

if( vB.getText().equals("0") )

{

vB.setWidth(widthOneButton \* 2);

vB.setHeight(heightOneButton);

}

else if( vB.getText().equals("=") )

{

vB.setWidth(widthOneButton);

vB.setHeight(heightOneButton \* 2);

}

else

{

vB.setWidth(widthOneButton);

vB.setHeight(heightOneButton);

}

}

}

}

}

/\*\*

\* El metodo onRestoreInstanceState es llamado de forma automática cuando se restaura la

\* aplicación. Al restaurarse se carga la información previamente guardada y se copia a las

\* propiedades de la clase/pantalla.

\*/

@Override

protected void onRestoreInstanceState(Bundle savedInstanceState)

{

super.onRestoreInstanceState(savedInstanceState);

memoryM = new BigDecimal(savedInstanceState.getDouble("memoryM"));

memoryLCD = new BigDecimal(savedInstanceState.getDouble("memoryLCD1"));

LCD.setOperation(new BigDecimal(savedInstanceState.getDouble("memoryLCD2")));

lastOperator = savedInstanceState.getChar("lastOperator");

lastKeyPressed = savedInstanceState.getString("lastKeyPressed");

LCD.addHistory(savedInstanceState.getString("History"));

lastConvert = savedInstanceState.getChar("lastConvert");

if( memoryM.compareTo(new BigDecimal(0.0F)) != 0 )

LCD.setMemory(true);

}

/\*\*

\* El método onSaveInstanceState se ejecuta de forma automática cuando la aplicación pasa a

\* estar en segundo plano. En ese momento se guardan todas las propiedades de la clase,

\* en un Bundle recibido por parámetro, antes de ser destruido.

\*/

@Override

public void onSaveInstanceState(Bundle outState)

{

super.onSaveInstanceState(outState);

outState.putDouble("memoryM", memoryM.doubleValue());

outState.putDouble("memoryLCD1", memoryLCD.doubleValue());

outState.putDouble("memoryLCD2", LCD.getOperationBigDecimal().doubleValue());

outState.putChar("lastOperator", lastOperator);

outState.putString("lastKeyPressed", lastKeyPressed);

outState.putString("History", LCD.getHistory());

outState.putChar("lastConvert", lastConvert);

}

/\*\*

\* Desde el método que destruye la activity se han de guardar las configuraciones.

\* Pero esto solo

\* se ha de realizar cuando la actividad vaya a ser destruida completamente

\* (y no vaya a ser creada a continuación).

\*/

@Override

protected void onDestroy()

{

super.onDestroy();

if( isFinishing() )

{

HashMap< String, Object > hm = new HashMap< String, Object >();

hm.put("LCDgetMemory", LCD.getMemory());

hm.put("LCDgetHistory", LCD.getHistory());

hm.put("LCDgetOperationString", LCD.getOperationString());

hm.put("memoryLCD", memoryLCD);

hm.put("memoryM", memoryM);

hm.put("lastOperator", lastOperator);

hm.put("lastKeyPressed", lastKeyPressed);

hm.put("lastConvert", lastConvert);

SettingsCalc.setData(hm);

setting.save();

}

}

/\*\*

\* Se crea el menú de opciones que se desplega al pulsar la tecla menu del dispositivo.

\*/

@Override

public boolean onCreateOptionsMenu(Menu menu)

{

this.getMenuInflater().inflate(R.menu.activity\_main, menu);

return true;

}

/\*\*

\* Según la opción elegida del menu se hace lo que corresponda:

\*/

@Override

public boolean onOptionsItemSelected(MenuItem item)

{

ClipboardManager clipboard = (ClipboardManager)getSystemService(Context.CLIPBOARD\_SERVICE);

ClipData clip;

switch( item.getItemId() )

{

// Si se pulsa copiar, se copia el numero en pantalla al portapapeles.

case R.id.main\_menu\_copy:

clip = ClipData.newPlainText("", LCD.getOperationString());

clipboard.setPrimaryClip(clip);

Toast.makeText(this, getResources().getString(R.string.main\_menu\_message\_copied), Toast.LENGTH\_SHORT).show();

break;

// Si se pulsa pegar, se comprueba que el contenido del portapapeles

// sea un numero y se pega a la pantalla.

case R.id.main\_menu\_paste:

clip = clipboard.getPrimaryClip();

if( clip != null )

{

ClipData.Item item2 = clip.getItemAt(0);

try

{

Double temp = Double.parseDouble(item2.getText().toString());

LCD.setOperation(new BigDecimal(temp));

}

catch( Exception e )

{

Toast.makeText(this, getResources().getString(R.string.main\_menu\_message\_no\_copied), Toast.LENGTH\_SHORT).show();

}

}

break;

//Abre el activiti de configuraciones de la app.

case R.id.main\_menu\_settings:

startActivity(new Intent(this, ActivityMenu.class));

break;

// Abre el activiti con información de la aplicación

case R.id.main\_menu\_about:

startActivity(new Intent(this, ActivityAbout.class));

break;

}

return true;

}

/\*\*

\* Evento que se realiza al pulsar las teclas numéricas y la coma.

\*

\* @param view

\*/

public void eventNumericButton(View view)

{

final String textBTN = ( (Button)findViewById(view.getId()) ).getText().toString();

String textLCD = LCD.getOperationString();

( (Vibrator)getSystemService(VIBRATOR\_SERVICE) ).vibrate(SettingsCalc.getVibrationTime());

// Si la ultima tecla pulsada fue un Operador: Se pasa el numero en pantalla a

// la memoria, y se borra la pantalla.

if( "+-x/%".indexOf(lastKeyPressed) != -1 )

{

memoryLCD = LCD.getOperationBigDecimal();

LCD.clearOperation();

textLCD = "";

}

// Si la ultima tecla fue el igual: Se borra la memoria y la pantalla.

else if( lastKeyPressed.equals("=") )

{

memoryLCD = new BigDecimal(0.0F);

LCD.clearOperation();

textLCD = "";

}

// Se registra la ultima tecla pulsada, en este caso numerica o la coma.

lastKeyPressed = textBTN;

// Si no se ha superado el limite maximo de caracteres en pantalla, se escribe...

if( textLCD.length() < numberCharacterLCD )

{

// el texto del boton numerico, pero solo si en la pantalla hay un float0 y el

// boton pulsado no es la coma.

if( textLCD.equals("0") && !textBTN.equals(".") )

LCD.setOperation(textBTN);

// el propio texto de la pantalla y el del boton pulsado, pero solo si la pantalla

// no esta vacia, no hay ninguna coma en la pantalla y el boton pulsado es una coma.

else if( !textLCD.isEmpty() && textLCD.indexOf(".") == -1 && textBTN.equals(".") )

LCD.setOperation(textLCD + textBTN);

// el propio texto de la pantalla y el del boton pulsado, pero solo si el

// boton pulsado no es una coma.

else if( !textBTN.equals(".") )

LCD.setOperation(textLCD + textBTN);

}

}

/\*\*

\* Evento que se realiza al pulsar alguna de las teclas relacionada con la memoria o pantalla.

\*

\* @param view

\*/

public void eventMemoryButton(View view)

{

final String textBTN = ( (Button)findViewById(view.getId()) ).getText().toString();

String textLCD = LCD.getOperationString();

( (Vibrator)getSystemService(VIBRATOR\_SERVICE) ).vibrate(SettingsCalc.getVibrationTime());

// Se borra la memoria y la M en pantalla:

if( textBTN.equals("MC") )

{

memoryM = new BigDecimal(0.0F);

LCD.setMemory(false);

}

// Se copia la memoria a la pantalla:

else if( textBTN.equals("MR") )

LCD.setOperation(memoryM);

// Se guarda la pantalla en la memoria y se añade una M a la pantalla:

else if( textBTN.equals("MS") )

{

memoryM = LCD.getOperationBigDecimal();

LCD.setMemory(true);

}

// Se suma a la memoria lo que haya en pantalla:

else if( textBTN.equals("M+") )

memoryM = memoryM.add(LCD.getOperationBigDecimal());

// Se resta a la memoria lo que haya en pantalla:

else if( textBTN.equals("M-") )

memoryM = memoryM.subtract(LCD.getOperationBigDecimal());

// Se elimina el ultimo caracter introducido por el usuario en la pantalla:

else if( textBTN.equals("←") )

{

// Solo si la pantalla tiene texto y la ultima tecla pulsada es numerica, la coma

// o la flcha de borrar:

if( !textLCD.isEmpty() && "0123456789.←".indexOf(lastKeyPressed) != -1 )

{

// Se borra el ultimo caracter.

String cadTemp = textLCD.substring(0, textLCD.length() - 1);

// Si la pantalla no esta vacia y lo que hay no es solo el signo negativo,

// se escribe en pantalla el texto. En el caso contrario, se escribe un float0.

if( !cadTemp.equals("") && !cadTemp.equals("-") )

LCD.setOperation(cadTemp);

else

LCD.setOperation(new BigDecimal(0.0F));

// En el evento para los botones de memoria solo interesa guarda la ultima

// tecla pulsada en el caso de ser la de borrar un digito, el resto no.

lastKeyPressed = textBTN;

}

}

// Se borra la pantalla:

else if( textBTN.equals("CE") )

LCD.setOperation(new BigDecimal(0.0F));

// Se borra la pantalla, el historial y la memoriaLCD:

else if( textBTN.equals("C") )

{

LCD.setOperation(new BigDecimal(0.0F));

LCD.clearHistory();

memoryLCD = new BigDecimal(0.0F);

}

}

/\*\*

\* Evento que se realiza al pulsar alguna de las teclas de operador.

\*

\* @param view

\*/

public void eventOperatorButton(View view)

{

final String textBTN = ( (Button)findViewById(view.getId()) ).getText().toString();

( (Vibrator)getSystemService(VIBRATOR\_SERVICE) ).vibrate(SettingsCalc.getVibrationTime());

BigDecimal numLCD = LCD.getOperationBigDecimal();

// Se cambia el signo del numero en pantalla:

if( textBTN.equals("±") )

LCD.setOperation(numLCD.multiply(new BigDecimal(-1.0F)));

// Se realiza la raiz cuadrada del numero en pantalla:

else if( textBTN.equals("√") )

{

if( LCD.getOperationBigDecimal().compareTo(new BigDecimal(0.0F)) == 1 )

{

LCD.setOperation(new BigDecimal(raices(numLCD.doubleValue())));

LCD.addHistory("sqrt(" + CalculatorLCD.removeDecimalEmpty(numLCD.doubleValue()) + ")");

}

}

// Determina el factorial de un número:

else if( textBTN.equals("!") )

LCD.setOperation(new BigDecimal(Factorial(numLCD.doubleValue())));

// Se realiza la operacion + - x / %

else if( "+-x/%".indexOf(textBTN) != -1 )

{

// Se añade al historial el ultimo numero (solo una vez indistintamente de cuantas

// veces pulsemos a las teclas operador)

if( "0123456789.←=".indexOf(lastKeyPressed) != -1 )

LCD.addHistory(numLCD);

// Se realiza la operacion que ademas sera mostrada en pantalla solo si la memoria

// no estaba vacia, es decir, si se ha podido realizar una operacion:

if( memoryLCD.compareTo(new BigDecimal(0.0F)) != 0

&& "+-x/%".indexOf(lastKeyPressed) == -1 )

LCD.setOperation(LCD.makeOperation(memoryLCD, lastOperator, numLCD));

else if( lastKeyPressed.equals("=") )

memoryLCD = numLCD;

// Se guarda el operador para cuando pulsemos la tecla =.

lastOperator = textBTN.charAt(0);

// Y por ultimo se añade el operador al historial:

LCD.addHistory(lastOperator + "");

}

// Realiza la operacion con el numero guardado en memoria y el de la pantalla segun la

// operacion elegida:

else if( textBTN.equals("=") )

{

// Si la memoria y la pantalla no estan vacias y el ultimo operador es uno de los

// indicados. Se escribe en la pantalla el resultado de la operacion y se borra

// la memoria y el historial.

if( memoryLCD.compareTo(new BigDecimal(0.0F)) != 0 &&

numLCD.compareTo(new BigDecimal(0.0F)) != 0 &&

"+-x/%".indexOf(lastOperator) != -1 )

{

LCD.setOperation(LCD.makeOperation(memoryLCD, lastOperator, numLCD));

memoryLCD = new BigDecimal(0.0F);

LCD.clearHistory();

}

}

else if( textBTN.equals("x²") )

{

if( LCD.getOperationBigDecimal().compareTo(new BigDecimal(0.0F)) == 1 )

{

LCD.setOperation(LCD.makeOperation(numLCD, 'x', numLCD));

LCD.addHistory(CalculatorLCD.removeDecimalEmpty(numLCD.doubleValue()) + "²");

}

}

lastKeyPressed = textBTN;

}

/\*\*

\* Cuando se pulsa el boton de menu en la calculadora avanzada, se ejecutan una serie de

\* eventos hasta que finalmente se crea el menu con las opciones.

\* Primero el boton hace saltar el evento eventMemuButton(), este hace saltar a

\* onCreateContextMenu() y este a su vez a onContextItemSelected().

\*

\* @param view

\*/

public void eventMemuButton(View view)

{

( (Vibrator)getSystemService(VIBRATOR\_SERVICE) ).vibrate(SettingsCalc.getVibrationTime());

registerForContextMenu(view); // Se agrega el boton que ha hecho saltar el evento a

// la lista de botones que pueden interactuar con el ContextMenu.

openContextMenu(view); // Despues se abre un ContextMenu y se pasa como parametro

// el boton que hizo saltar el evento.

}

/\*\*

\* El metodo onCreateContextMenu() se encarga de crear los diferentes menus. Segun el boton que

\* hayamos pulsado deberemos de asignarle su menu correspondiente haciendo una comparacion

\* de IDs entre el que se recibe como parametro y el que corresponda al boton que queramos.

\* Asi podremos indicar que boton habre cada menu ya que aqui se han declarar

\* todos los ContextMenu.

\*/

@Override

public void onCreateContextMenu(ContextMenu menu, View view, ContextMenuInfo menuInfo)

{

super.onCreateContextMenu(menu, view, menuInfo);

// Si el id del view que se recibe por parametro es el mismo que el id del boton menu,

// entonces se infla el menu de opciones avanzadas.

if( view.getId() == R.id.main\_btn\_menu )

{

MenuInflater inflater = getMenuInflater();

inflater.inflate(R.menu.mathematical\_menu, menu);

}

}

/\*\*

\* Por ultimo el método onContextItemSelected() aplica la logica que queramos a cada item del

\* ContextMenu. Para ello se compara el titulo del item y el del boton y si coinciden

\* se ejecuta lo que corresponda.

\*/

@Override

public boolean onContextItemSelected(MenuItem item)

{

( (Vibrator)getSystemService(VIBRATOR\_SERVICE) ).vibrate(SettingsCalc.getVibrationTime());

int value = 0;

switch (lastConvert){

case 'b':

value = Integer.parseInt(Integer.toString(LCD.getOperationBigDecimal().intValue()), 2);

break;

case 'h':

value = Integer.parseInt(LCD.getOperationString(), 16);

break;

case 'o':

value = Integer.parseInt(Integer.toString(LCD.getOperationBigDecimal().intValue()), 8);

break;

default:

value = LCD.getOperationBigDecimal().intValue();

break;

}

switch (item.getItemId()){

case R.id.mathematical\_menu\_convert\_binary:

lastConvert = 'b';

LCD.clearHistory();

LCD.setOperation(Integer.toBinaryString(value));

LCD.addHistory("Dec: " + value + ", Hex: "+Integer.toHexString(value).toUpperCase() + ", Oct:"+Integer.toOctalString(value));

break;

case R.id.mathematical\_menu\_convert\_decimal:

lastConvert = 'd';

LCD.clearHistory();

LCD.setOperation(Integer.toString(value));

LCD.addHistory("Bin: " + Integer.toBinaryString(value) + ", Hex: "+Integer.toHexString(value).toUpperCase() + ", Oct:"+Integer.toOctalString(value));

break;

case R.id.mathematical\_menu\_convert\_hexadecimal:

lastConvert = 'h';

LCD.clearHistory();

LCD.setOperation(Integer.toHexString(value).toUpperCase());

LCD.addHistory("Bin: " + Integer.toBinaryString(value) + ", Dec: "+Integer.toHexString(value).toUpperCase() + ", Oct:"+Integer.toOctalString(value));

break;

case R.id.mathematical\_menu\_convert\_octal:

lastConvert = 'o';

LCD.clearHistory();

LCD.setOperation(Integer.toOctalString(value));

LCD.addHistory("Bin: " + Integer.toBinaryString(value) + ", Dec: "+ value + ", Hex:"+Integer.toHexString(value).toUpperCase());

break;

case R.id.mathematical\_menu\_sin:

LCD.setOperation(new BigDecimal(serieTaylorSeno(LCD.getOperationBigDecimal().doubleValue()

\* 2.0 \* Math.PI / 360.0)));

break;

case R.id.mathematical\_menu\_cos:

LCD.setOperation(new BigDecimal(serieTaylorCoseno(LCD.getOperationBigDecimal().doubleValue()

\* 2.0 \* Math.PI / 360.0)));

break;

case R.id.mathematical\_menu\_tan:

LCD.setOperation(new BigDecimal(Math.tan(LCD.getOperationBigDecimal().floatValue() \*

2.0 \* Math.PI / 360.0)));

break;

case R.id.mathematical\_menu\_raiz:

if( LCD.getOperationBigDecimal().compareTo(new BigDecimal(0.0F)) == 1 )

{

LCD.setOperation(new BigDecimal(raices(LCD.getOperationBigDecimal().doubleValue())));

LCD.addHistory("sqrt(" + CalculatorLCD.removeDecimalEmpty(LCD.getOperationBigDecimal().doubleValue()) + ")");

}

break;

case R.id.mathematical\_menu\_pi:

LCD.setOperation(new BigDecimal(Math.PI));

break;

default:

return false;

}

return true;

}

/\*\*

\* Método que calcula el factorial de un número entero con recursividad.

\*

\* @param entrada

\*/

public static double Factorial(double entrada){

if(entrada<0){

return 0;

}

else{

if(entrada==0){

return 1;

}

else

{

return entrada\*Factorial(entrada-1);

}

}

}

/\*\*

\* Método que calcula el coseno de un número usando series de Taylor.

\* @param x

\*/

static double serieTaylorCoseno(double x) {

double sumando, sumatoria = 0, precision = 0.0001d;

// limite superior, iteracion de la sumatoria

int n = 0;

do {

sumando = Math.pow(-1, n) / Factorial(2\*n) \* Math.pow(x, 2 \* n);

sumatoria = sumatoria + sumando;

n = n + 1;

} while (Math.abs(sumando) > precision);

return sumatoria;

}

/\*\*

\* Método que calcula el seno de un número usando series de Taylor.

\* @param x

\*/

static double serieTaylorSeno(double x) {

double sumando, sumatoria = 0, precision = 0.0001d;

// limite superior, iteracion de la sumatoria

int n = 0;

do {

sumando = Math.pow(-1, n) / Factorial((2\*n)+1) \* Math.pow(x, (2 \* n)+1);

sumatoria = sumatoria + sumando;

n = n + 1;

} while (Math.abs(sumando) > precision);

return sumatoria;

}

/\*\*

\* Método que calcula la raíz de un número en base a aproximaciones.

\*

\* @param entrada

\* @return double x

\*/

public double raices(double entrada)

{

double x=1.0;

if(entrada>=0){

for(int k = 1; k < 10; k++){

x = (x + entrada/x) / 2;

}

return x;

}

else

{

x=0;

return x;

}

}

}

Graphactivity

package ec.edu.espe.programacion.calculator\_iza\_angos;

import androidx.appcompat.app.AppCompatActivity;

import android.os.Bundle;

import com.jjoe64.graphview.GraphView;

import com.jjoe64.graphview.series.DataPoint;

import com.jjoe64.graphview.series.LineGraphSeries;

public class GraphActivity extends AppCompatActivity {

private LineGraphSeries<DataPoint> series1;

@Override

protected void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.activity\_graph);

double x,y;

x=0;

GraphView graph=(GraphView) findViewById(R.id.graph);

series1= new LineGraphSeries<>();

int NumDataPoints=250;

for(int i=0;i<NumDataPoints;i++){

x=x+0.1;

y=Math.sin(x);

series1.appendData(new DataPoint(x,y),true,10000);

}

// activate horizontal zooming and scrolling

graph.getViewport().setScalable(true);

// activate horizontal scrolling

graph.getViewport().setScrollable(true);

// activate horizontal and vertical zooming and scrolling

graph.getViewport().setScalableY(true);

// activate vertical scrolling

graph.getViewport().setScrollableY(true);

graph.getGridLabelRenderer().setHorizontalAxisTitle("Eje X Función Seno");

graph.getGridLabelRenderer().setVerticalAxisTitle("Eje Y Función Seno");

graph.addSeries(series1);

}

}

Parameters

package ec.edu.espe.programacion.calculator\_iza\_angos;

import androidx.appcompat.app.AppCompatActivity;

import android.graphics.Color;

import android.os.Bundle;

import com.jjoe64.graphview.GraphView;

import com.jjoe64.graphview.series.DataPoint;

import com.jjoe64.graphview.series.LineGraphSeries;

public class ParametersGraphics extends AppCompatActivity {

private LineGraphSeries<DataPoint> series1;

@Override

protected void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.activity\_parameters\_graphics);

double x,y;

x=0;

GraphView graph=(GraphView) findViewById(R.id.graph2);

series1= new LineGraphSeries<>();

int NumDataPoints=250;

for(int i=0;i<NumDataPoints;i++){

x=x+0.1;

y=Math.cos(x);

series1.appendData(new DataPoint(x,y),true,10000);

}

// activate horizontal zooming and scrolling

graph.getViewport().setScalable(true);

// activate horizontal scrolling

graph.getViewport().setScrollable(true);

// activate horizontal and vertical zooming and scrolling

graph.getViewport().setScalableY(true);

// activate vertical scrolling

graph.getViewport().setScrollableY(true);

graph.getGridLabelRenderer().setHorizontalAxisTitle("Eje X Función Coseno");

graph.getGridLabelRenderer().setVerticalAxisTitle("Eje Y Función Coseno");

series1.setColor(Color.RED);

graph.addSeries(series1);

}

}

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

DOCENTE RESPONSABLE COORDINADOR DE ÁREA

Ing. Fernando Solis. MsC. Ing. Silvia Arévalo.