Introducción

JME (antes J2ME, se ha perdido el 2) es una plataforma muy extendida que permite instalar aplicaciones java en dispositivos móviles. La universalidad de estas aplicaciones nos permite despreocuparnos del modelo de nuestro terminal o de su escasa memoria RAM.

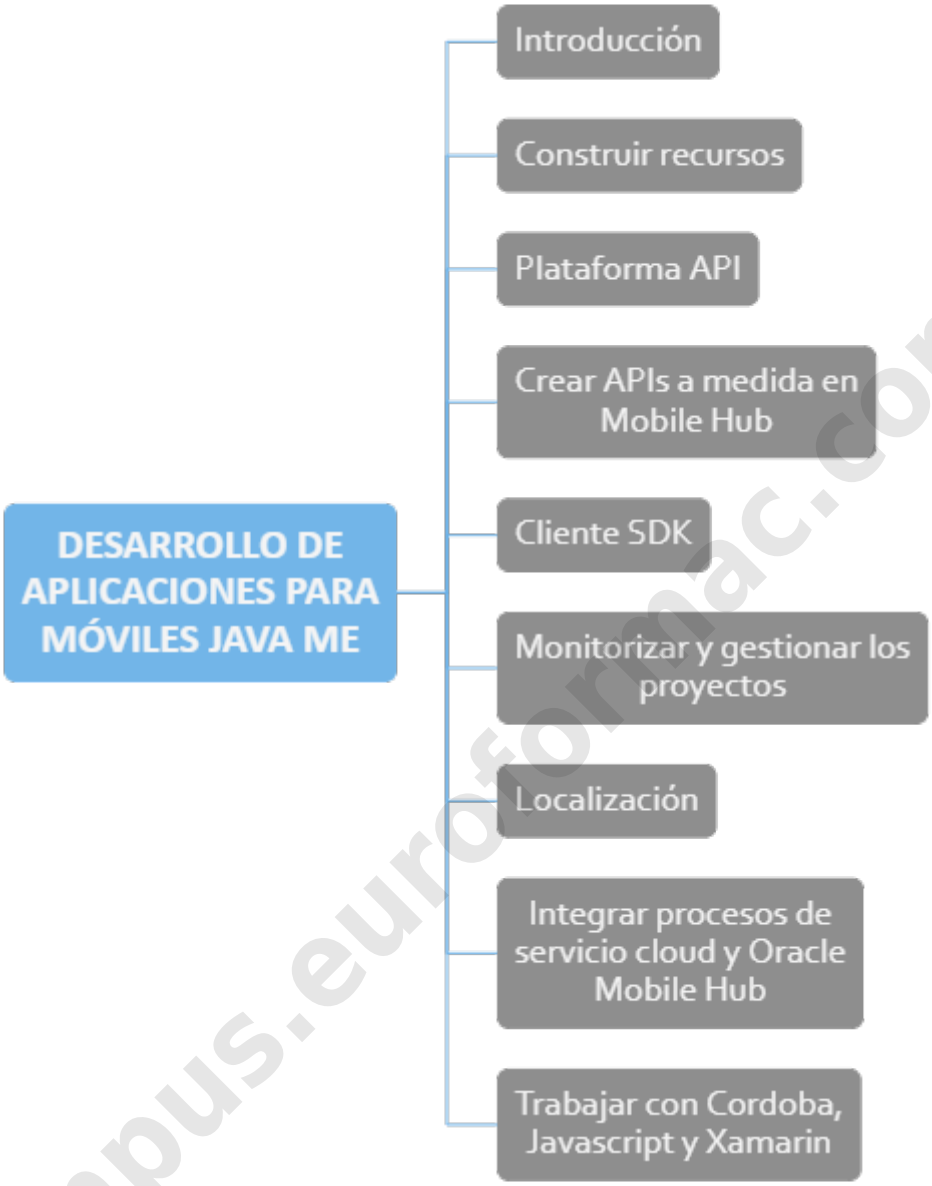
Objetivos

Identificar las interfaces de usuario básicas con Java ME.

Saber las capacidades que aporta la API.

Comprender el paquete de ciclo de vida de las aplicaciones.

Mapa Conceptual



Introducción

Java ME está compuesto por dos conjuntos de bibliotecas, que se conocen como configuración de dispositivo limitada conectada (CLDC) y configuración de dispositivo conectado (CDC). El CLDC está diseñado para dispositivos con limitaciones significativas caracterizadas por baja potencia de procesamiento, espacio de almacenamiento, RAM y capacidades gráficas. Los dispositivos que son más adecuados para el CLDC pueden tener una velocidad de reloj de CPU de tan solo 16 MHz, un tamaño de ROM tan pequeño como 180 KB, RAM tan pequeño como 192 KB y cero gráficos. Los dispositivos CDC pueden ser más potentes. Ejemplos de tales dispositivos incluyen teléfonos inteligentes, computadoras de bolsillo y PDA.

Las aplicaciones Java ME suelen asociarse con pequeñas aplicaciones llamadas MIDlets, que son solo un grupo de aplicaciones escritas con Java ME. Sin embargo, los MIDlets son en realidad aplicaciones escritas utilizando el perfil del dispositivo de información móvil, que se encuentra en la parte superior del CLDC.

Un aspirante a desarrollador de Java ME normalmente necesitaría el kit de desarrollo de software (SDK) de Java ME. Contiene todas las herramientas necesarias para el desarrollo de aplicaciones móviles Java, incluidas la API, el depurador, el compilador y el emulador. Para simplificar el proceso de desarrollo, los desarrolladores pueden usar el SDK junto con entornos de desarrollo integrados (IDE) como Netbeans y Eclipse. Estos IDE permiten a los desarrolladores aprovechar las GUI, que permiten los procedimientos de arrastrar y soltar y apuntar y hacer clic, para diseñar el diseño de la GUI de la aplicación Java ME. Junto con el SDK, los IDE permiten a los usuarios ver cómo aparecería una aplicación en un dispositivo, a través de emuladores.

Construir recursos

Java me es un lenguaje orientado a objetos (OO), por lo que, antes de empezara ver qué elementos componen los programas Java, conviene tener claros algunos conceptos de la programación orientada a objetos (POO).

Concepto de clase y objeto

El elemento fundamental a la hora de hablar de programación orientada a objetos es el concepto de objeto en sí, así como el concepto abstracto de clase. Un objeto es un conjunto de variables junto con los métodos relacionados con éstas. Contiene la información (las variables) y la forma de manipular la información (los métodos). Una clase es el prototipo que define las variables y métodos que va a emplear un determinado tipo de objeto, es la definición abstracta de lo que luego supone un objeto en memoria.
Poniendo un símil fuera del mundo de la informática, la clase podría ser el concepto de coche, donde nos vienen a la memoria los parámetros que definen un coche (dimensiones, cilindrada, maletero, etc), y las operaciones que podemos hacer con un coche (acelerar, frenar, adelantar, estacionar). La idea abstracta de coche que tenemos es lo que equivaldría a la clase, y la representación concreta de coches concretos (por ejemplo, Peugeot 307, Renault Megane, Volkswagen Polo...) serían los objetos de tipo coche

Concepto de campo, método y constructor

Toda clase u objeto se compone internamente de constructores, campos y/o métodos. Veamos qué representa cada uno de estos conceptos: un campo es un elemento que contiene información relativa a la clase, y un método es un elemento que permite manipular la información de los campos. Por otra parte, un constructor es un elemento que permite reservar memoria para almacenar los campos y métodos de la clase, a la hora de crear un objeto de la misma.

Concepto de herencia y polimorfismo

Con la herencia podemos definir una clase a partir de otra que ya exista, de forma que la nueva clase tendrá todas las variables y métodos de la clase a partir de la que se crea, más las variables y métodos nuevos que necesite. A la clase base a partir de la cual se crea la nueva clase se le llama

superclase.

Por ejemplo, podríamos tener una clase genérica Animal, y heredamos de ella para formar clases más específicas, como Pato, Elefante, o León. Estas clases tendrían todo lo de la clase padre Animal, y además cada una podría tener sus propios elementos adicionales. Una característica derivada de la herencia es que, por ejemplo, si tenemos un método dibuja(Animal a), que se encarga de hacer un dibujo del animal que se le pasa como parámetro, podremos pasarle a este método como parámetro tanto un Animal como un Pato, Elefante, o cualquier otro subtipo directo o indirecto de Animal. Esto se conoce como polimorfismo

Modificadores de acceso

Tanto las clases como sus elementos (constructores, campos y métodos) pueden verse modificados por lo que se suelen llamar modificadores de acceso, que indican hasta dónde es accesible el elemento que modifican. Tenemos tres tipos de modificadores:

• **privado**: el elemento es accesible únicamente dentro de la clase en la que se encuentra.

• **protegido**: el elemento es accesible desde la clase en la que se encuentra, y además desde las subclases que hereden de dicha clase.

• **público**: el elemento es accesible desde cualquier clase

Plataforma API

La plataforma JavaME nos ofrece una serie de APIs con las que desarrollar las aplicaciones en lenguaje Java.

Una vez tengamos la aplicación podremos descargarla en cualquier dispositivo con soporte para JavaME y ejecutarla en él.

JavaME soporta una gran variedad de dispositivos, no únicamente MIDs. Actualmente define APIs para dar soporte a los dispositivos conectados en general, tanto aquellos con una gran capacidad como a tipos más limitados de estos dispositivos.

JavaME es el nombre corto de Java Micro Edition y también es ampliamente conocido con su anterior nombre comercial, J2ME que significa Java 2 Micro Edition. A partir de ahora nos referiremos a la plataforma por ambos nombres indistintamente

Hemos visto que existen dispositivos de tipos muy distintos, cada uno de ellos con sus propias necesidades, y muchos con grandes limitaciones de capacidad. Si obtenemos el máximo común denominador de todos ellos nos quedamos prácticamente con nada, por lo que es imposible definir una única API en J2ME que nos sirva para todos.

Por ello en J2ME existirán diferentes APIs cada una de ellas diseñada para una familia de dispositivos distinta. Estas APIs se encuentras arquitecturadas en dos capas: configuraciones y perfiles.

Configuraciones

Las configuraciones son las capas de la API de bajo nivel, que residen sobre la máquina virtual y que están altamente vinculadas con ella, ofrecen las características básicas de todo un gran conjunto de dispositivos. En esta API ofrecen lo que sería el máximo denominador común de todos ellos, la API de programación básica en lenguaje Java.

Encontramos distintas configuraciones para adaptarse a la capacidad de los dispositivos. La configuración CDC (Connected Devices Configuration) contiene la API común para todos los dispositivos conectados, soportada por la máquina virtual Java. Sin embargo, para algunos

dispositivos con grandes limitaciones de capacidad esta máquina virtual Java puede resultar demasiado compleja. Hemos de pensar en dispositivos que pueden tener 128 KB de memoria. Es evidente que la máquina virtual de Java (JVM) pensada para ordenadores con varias megas de RAM instaladas no podrá funcionar en estos dispositivos.

Por lo tanto aparece una segunda configuración llamada CLDL (Connected Limited Devices Configuration) pensada para estos dispositivos con grandes limitaciones. En ella se ofrece una API muy reducida, en la que tenemos un menor número de funcionalidades, adaptándose a las posibilidades de estos dispositivos. Esta configuración está soportada por una máquina virtual mucho más reducida, la KVM (Kilobyte Virtual Machine), que necesitará muchos menos recursos por lo que podrá instalarse en dispositivos muy limitados

Perfiles

Como ya hemos dicho, las configuraciones nos ofrecen sólo la parte básica de la API para programar en los dispositivos, aquella parte que será común para todos ellos. El problema es que esta parte común será muy reducida, y no nos permitirá acceder a todas las características de cada tipo de dispositivo específico. Por lo tanto, deberemos extender la API de programación para cada familia concreta de dispositivos, de forma que podamos acceder a las características propias de cada familia.

Esta extensión de las configuraciones es lo que se denomina perfiles. Los perfiles son una capa por encima de las configuraciones que extienden la API definida en la configuración subyacente añadiendo las operaciones adecuadas para programar para una determinada familia de dispositivos.

Por ejemplo, tenemos un perfil MIDP (Mobile Information Devices Profile) para programar los dispositivos móviles de información. Este perfil MIDP reside sobre CLDC, ya que estos son dispositivos bastante limitados a la mayoría de las ocasiones.

Crear APIs a medida en Mobile Hub

Con Oracle Mobile Hub, puede desarrollar aplicaciones móviles cliente-servidor y desplegar varias API diseñadas para admitir tareas como la gestión de usuarios, el acceso a la base de datos, los servicios de ubicación, el análisis, etc. Estas API realizan integraciones con otras soluciones de Oracle Cloud, servicios de REST y aplicaciones de cliente móvil posibles.

Un backend móvil dentro de Oracle Mobile Hub es un grupo de API seguras y otros recursos utilizados para desarrollar aplicaciones móviles cliente-servidor.

Puede utilizar Oracle Mobile Hub para agrupar y gestionar distintas API que definan un backend móvil.

En Oracle Mobile Hub, puede tener varios backends, cada uno con diferentes aplicaciones móviles de cliente. Sin embargo, también puede tener varias API compartidas entre distintos backends. La siguiente imagen muestra una arquitectura de backend móvil típica en Oracle Mobile Hub:

Cuando una aplicación móvil de cliente accede a una API a través de Oracle Mobile Hub, siempre está en el contexto de un backend móvil. La aplicación se puede autenticar mediante credenciales definidas en Oracle Mobile Hub (OAuth o autenticación HTTP básica) específicas del backend móvil o a través de un almacén de identidades (o un proveedor de conexión social) mediados por el

backend móvil. Si la API llamada incluye llamadas a otras API del backend, la identidad y las credenciales del emisor de llamada original se propagan por la cadena de llamadas.

El trabajo en backend móvil le ayuda a visualizar los recursos disponibles para las aplicaciones de destino y a trabajar juntos. Además, puede utilizar el contexto de seguridad de backend móvil para probar las llamadas a las API, incluso en las primeras etapas del desarrollo.

En Oracle Mobile Hub, puede crear backends móviles para asociar API y recopilaciones de almacenamiento, e integrarlas con las aplicaciones móviles del cliente. Para crear un backend móvil en Oracle Mobile Hub:

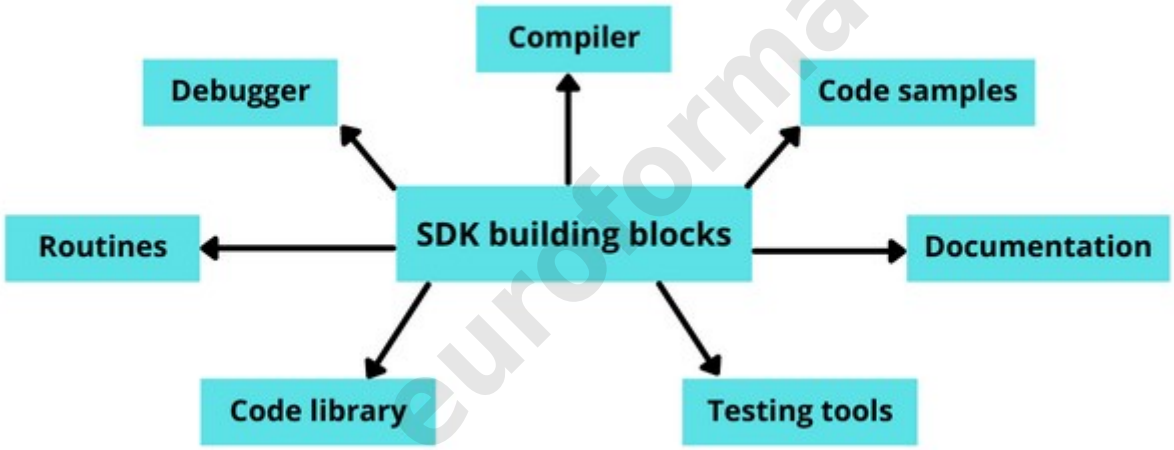
- Inicie sesión en Oracle Mobile Hub.
- Haga clic en el icono de menú y, a continuación, seleccione Desarrollo y Backends.
- Haga clic en Nuevo Backend.
- Introduzca un nombre y una descripción para el backend móvil.
- Haga clic en Crear.
- Se crea el backend móvil y se abre su página de configuración.
- En la página Valores, active la opción Consumidor de OAuth.
- Haga clic en la casilla de control Activar Single Sign-On.

Después de crear su backend, debe aparecer en la sección Aplicaciones de Oracle Mobile Hub en backends móviles.

Cliente SDK

SDK es un acrónimo de "Software Development Kit", también conocido como "devkit". SDK se define mediante un conjunto de herramientas de creación de software que son específicas de las plataformas. Estas herramientas incluyen depuradores, compiladores, bibliotecas de código (es decir, marco) o rutinas y subrutinas dirigidas a un sistema operativo.

Los componentes básicos de un SDK típico son los siguientes:



Depurador: Un depurador permite a los desarrolladores identificar y corregir errores en el código del programa.
Compilador: Un compilador es un programa que procesa lenguaje de programación declaraciones y las traduce a un lenguaje o "código" comprensible por máquina utilizado por el procesador.
Ejemplos de código divulgar tareas de programación o escenarios que dan una imagen más clara de una aplicación o página web.
Rutinas y subrutinas: Una rutina o subrutina es un método, función, procedimiento, subprograma o código que se puede llamar y ejecutar en cualquier parte del código del programa completo. Por ejemplo, la opción de guardar archivo se ejecuta mediante una rutina.
Biblioteca de códigos: Una biblioteca de código permite a los desarrolladores utilizar recursos existentes (por ejemplo, secuencias de código) en lugar de reinventarlos.
Herramientas analíticas y de prueba: Estas herramientas evalúan el rendimiento de la aplicación en un entorno de prueba y producción.
Documentación: Los desarrolladores consultan las instrucciones documentadas (según sea necesario) durante el proceso de desarrollo.

Monitorizar y gestionar los proyectos

Las herramientas para el monitoreo de Java de Applications Manager están diseñadas para monitorear proactivamente las aplicaciones de Java y las tecnologías que utilizan, y optimizar su rendimiento.

El monitoreo de Java monitorea las métricas importantes y envía alarmas de precaución cuando observa un patrón anómalo en el rendimiento. Usted puede reducir el MTTR y las intervenciones manuales, ya que nuestro sistema de gestión de fallos inteligente rastrea e identifica la causa raíz de los errores y ayuda con medidas correctivas. Además del monitoreo de Java en tiempo real y el análisis del rendimiento histórico, el monitor de Java de Applications Manager también permite a los usuarios prevenir picos en el uso de recursos con informes de proyección que emplean técnicas de machine learning para predecir el uso. La solución para el monitoreo de Java de Applications Manager también ofrece informes estadísticos para ayudarle a obtener visibilidad de cómo varios parámetros de líneas de comando y algoritmos de recolección de basura impactan el rendimiento de las aplicaciones Java.

La solución para el monitoreo de aplicaciones Java en Applications Manager le permite captar y realizar el monitoreo del rendimiento de aplicaciones Java sobre todas las métricas clave de rendimiento de sus servidores de aplicaciones basados en Java como Apache Tomcat, JBoss, IBM Websphere y Oracle WebLogic en tiempo real. Vea métricas detalladas directamente en su dashboard de monitoreo de Java, que da representaciones gráficas y numéricas de los KPI. Los parámetros importantes en los servidores de aplicaciones Java que necesitan una vigilancia constante son los relacionados con bases de datos, concurrencia y memoria; monitorear estas métricas le ayuda a optimizar el rendimiento de las aplicaciones Java

Localización

Con las API de ubicación de los Servicios de Google Play, tu app puede solicitar la ubicación más reciente del dispositivo del usuario. En la mayoría de los casos, te interesará la ubicación actual del usuario, que suele equivaler a la ubicación más reciente del dispositivo.

Usa específicamente el proveedor de ubicación combinada para obtener la ubicación más reciente del dispositivo. Este proveedor es una de las API de ubicación de los Servicios de Google Play. Administra la tecnología de ubicación subyacente y es una API simple que te permite especificar requisitos (como precisión alta o poca energía) a nivel general. También optimiza el uso de la batería del dispositivo.

Cómo crear un cliente de servicios de ubicación

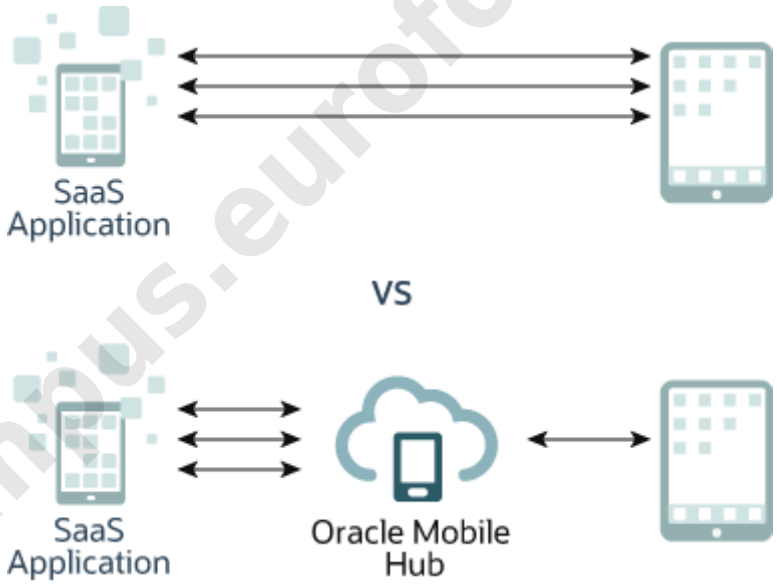
En el método onCreate() de tu actividad, crea una instancia del cliente de proveedor de ubicación combinada, como se muestra en el siguiente fragmento de código:

private FusedLocationProviderClient fusedLocationClient; // .. @Override protected void onCreate(Bundle savedInstanceState) { // ... fusedLocationClient = LocationServices.getFusedLocationProviderClient(this); }

Integrar procesos de servicio cloud y Oracle Mobile Hub

La creación de una aplicación móvil consta de varias etapas discretas. Es importante comprender las consideraciones y los requisitos de cada etapa antes de seleccionar sus herramientas y empezar a crear su aplicación.

Para comenzar, suele ser una buena idea crear una capa API ficticia. Esto es importante porque las API del proveedor de SaaS suelen ser complejas y exhaustivas, y es posible que no desee exponer esas API completas a una aplicación cliente, ni siquiera en el nivel de código. Además, se debe tener en cuenta el rendimiento de la aplicación, los límites de llamada de la API, la seguridad de los datos personalizados y otras consideraciones. Esta plantilla consume las API de servicio y expone una API limpia y precisa al cliente.



El siguiente paso es crear la capa de la interfaz de usuario. Desea desarrollar una interfaz de usuario de forma rápida y eficaz y mantener la flexibilidad en cuanto a la representación de la interfaz de usuario y la flexibilidad para cambiar pantallas con un mínimo esfuerzo.

El último paso para crear esta aplicación es integrar el motor de flujo de trabajo con la interfaz de usuario.

El uso de servicios de Oracle Cloud como Oracle Mobile Hub y Oracle Integration Service le ayuda a

realizar cada uno de estos pasos de una forma eficaz y ágil. Las API de Oracle Mobile Hub se crean con JavaScript en ejecución en Node.js. Esto permite emitir una sola llamada a Oracle Mobile Hub y, a continuación, dejarla en Oracle Mobile Hub para emitir todas las llamadas necesarias al servicio backend de forma asíncrona. Los resultados se agregan y se devuelven como una única carga útil al cliente. Esto reduce significativamente el tráfico de red entre Oracle Mobile Hub y el cliente, lo que mejora el rendimiento. Opcionalmente, puede utilizar Oracle Integration Service para proporcionar Single Sign-On (SSO) entre los servicios.

Trabajar con Cordoba, Javascript y Xamarin

Cordova

Cordova es otra alternativa de código abierto a Xamarin. Esta plataforma utiliza CSS, JavaScript y HTML para desarrollar aplicaciones móviles con funcionalidades móviles nativas. Cordova es la más adecuada para desarrollar aplicaciones móviles que pueden usar funciones híbridas como etiquetado de ubicación geográfica, cámara, sistema de archivos, etc.

La interfaz de usuario de Cordova actúa como WebView, ocupa toda la pantalla y se ejecuta en el contenedor nativo. Debido a esto, los contenedores nativos siguen cambiando según el sistema operativo móvil utilizado. La presencia de una biblioteca elaborada hace que sea aún más conveniente para las aplicaciones móviles producir la mejor experiencia de usuario nativa. Las aplicaciones desarrolladas con el marco Cordova prometen una experiencia de usuario más rica en diferentes plataformas móviles.

Xamarin

Xamarin es una herramienta de software de código abierto para desarrollar aplicaciones de Android y iOS. Esta herramienta proviene de la casa de Microsoft y usa .NET y C# para manejar los procedimientos de desarrollo de la aplicación. Mientras que .NET actúa como una plataforma base para desarrollar aplicaciones móviles, Xamarin agrega más destreza al escenario al crear un tipo de aplicación específico. El marco de Xamarin usa un conjunto adicional de bibliotecas y herramientas para extender la plataforma .NET para desarrollar tipos particulares de aplicaciones para Android y iOS.

Si bien Xamarin es un excelente marco de código abierto para extender la funcionalidad .Net, últimamente existen varias alternativas disponibles. Algunas de estas alternativas han producido resultados aún mejores en comparación con Xamarin. Hoy, discutiremos algunas de las mejores alternativas a Xamarin.

Recuerda

- Connected Limited Device Configuration (CLDC), esta configuración está diseñada para los dispositivos con recursos limitados de memoria, almacenamiento, procesamiento y capacidades graficas, como pueden ser teléfonos móviles antiguos, PDAs, etc
- La interfaz de usuario es la parte del programa que permite al usuario interaccionar con el.

Autoevaluación

1. Java...

- es un lenguaje de programación muy completo, el cual es utilizado para la creación de aplicaciones de todo tipo, desde aplicaciones para dispositivos móviles como aplicaciones empresariales de tipo web, aplicaciones de escritorio y aplicaciones de automatización de procesos.
- no es un lenguaje de programación muy completo, el cual es utilizado para la creación de aplicaciones de todo tipo, desde aplicaciones para dispositivos móviles como aplicaciones empresariales de tipo web, aplicaciones de escritorio y aplicaciones de automatización de procesos.
- es un lenguaje de programación muy completo, el cual no es utilizado para la creación de aplicaciones de todo tipo, desde aplicaciones para dispositivos móviles como aplicaciones empresariales de tipo web, aplicaciones de escritorio y aplicaciones de automatización de procesos.

2. J2SE...

- no es una colección de APIs del lenguaje de programación Java útiles para muchos programas de la Plataforma Java
- es una colección de APIs del lenguaje de programación Java inútiles para muchos programas de la Plataforma Java
- es una colección de APIs del lenguaje de programación Java útiles para muchos programas de la Plataforma Java

3.Un JDK|SDK...

- no ofrece las herramientas para compilar y ejecutar programas en Java
- ofrece las herramientas para compilar pero no para ejecutar programas en Java.
- ofrece las herramientas para compilar y ejecutar programas en Java

4. Indica si la siguiente afirmación es verdadera o falsa *:* "Enterprise Java Beans (EJB) es una plataforma para construir aplicaciones de negocio portables, reutilizables y escalables usando el lenguaje de programación Java".

- Verdadero.
- Falso.

5. Indica si la siguiente afirmación es verdadera o falsa *:* "Los componentes de software son unidades binarias de producción, adquisición y despliegue independientes, que al interactuar, forman un sistema en funcionamiento"

- Verdadero.
- Falso.