# *RESUMEN AATT*

# TEMA 1. Introducción a Java

Para compilar: Javac Prueba.java %%%Aparece Prueba.cl

ass si no hay errores

Para depurar: jdb

Para ejecutar: java Prueba

Concatenación: Si se pone un + entre dos números, se hará suma, pero si alguno de los operandos es string se concatenarán.

Comparación de strings: equals dará siempre true si son la misma cadena, pero == solo dará true en caso de haberse declarado igual (string x= “ab”) pero no si se pone new String . == mira la referencia en memoria, equals mira los valores.

**Clases:** Describen una serie de métodos y atributos. Cuando se instancia una clase se consigue un objeto, a través del operador new (este objeto puede dar unos ciertos valores a unos parámetros, realizar alguna actividad..). Para acceder a un miembro (atributo o método) se hace objeto.x

**Sobrecarga**: dos métodos con el mismo nombre y distintos parámetros. Se distinguen al usarlas por los parámetros que le introduces. Si dos métodos comparten nombre y parámetros, dará error de compilación.

No existen métodos locales (métodos dentro de métodos).

El **constructor** es el método ejecutado automáticamente al instanciar una clase y obtener un objeto. Tiene el mismo nombre de la clase. Puede sobrecargarse. Pueden haber todos los que se quieran. No tiene que existir necesariamente. Puede tener modificadores de acceso.

Hay que usar **this** para que no haya ambigüedad en nombres de parámetros.

En el **package** se agrupan clases. En cada package no puede repetirse el nombre de clases.

**Modificadores de no acceso:**

- **Static:** Existe independientemente de las instancias (vale para variables, clases o métodos).
- **Final:** Indica que no habrá sobreescritura. Vale para variables, clases y métodos.
- **Abstract:** Crea métodos o clases sin implementar. No hay variables abstractas.

**Modificadores de acceso:**

| | Desde la misma clase | Desde otra clase del mismo paquete | Desde una subclase de otro paquete | Desde otra clase de otro paquete |
|---|---|---|---|---|
| public | X | X | X | X |
| protected | X | X | X | |
| (Nada: privada fuera de paquete) | X | X | | |
| private | X | | | |

Si se intenta acceder desde donde no se puede a algo, dará un error de compilación.

**Herencia:** La hija hereda atributos (los heredables) de una única clase padre (superclase). Los objetos de la hija también pertenecen a la padre. Puede usar el constructor de la clase padre. Se pueden sobreescribir métodos (@Override). Si se accede algo del padre se usa super, y si se accede algo de la hija this. Public class x **extends** y. Solo se puede extender una clase.

**Clases abstractas:** Se puede extender pero no instanciarse. Si una clase tiene métodos abstractos deberá ser abstracta.

**Clases anónimas:** Clase sin nombre resultante de extender cierta clase padre y en la que no aparecen nuevos miembros, sino que se sobreescriben algunos. Es de un solo uso, puede usarse en el contexto del poliformismo, no puede referenciarse tras su creación.

**Interfaces:** Permite una herencia múltiple , implementándose por una clase. Esta debe implementar todos los métodos abstractos. Una clase puede implementar varias interfaces. Una interfaz puede extender varias interfaces.

**Polimorfismo:** Conversión de clase de hijo a padre. Los métodos y atributos exclusivos del hijo quedan escondidos. De forma normal el proceso no es invertible.

Class Y extends X (Y es la hija, X es la padre). X xy=new Y(); bien . Al revés mal.

**Clases anidadas:**

**Excepciones:** try catch finally, throws

**Multihilo:** Al ejecutar el start(), se hará un run del hilo.

Compilación y ejecución:

Javac Principal.java. Si le pones -d lo reubica en el directorio en el que estes.

Java Principal

Jar: permite empaquetar ficheros rpara la ejecución en entorno java.

# *TEMA 1.2: XML y JSON*

**JSON:** Funciona con pares nombre valor. Se compone de arrays (corchetes) y objetos (llaves). Hay JsonReader, JsonWriter, JsonArray, JsonObject, JsonString…

Los objetos tienen propiedades del formato “nombre”:valor (si el valor es string ira con comillas también).

Los arrays tienen valores separados por comas y pueden ser de distinto tipo.

Para leer un Json:

## Ejemplo. Lectura JSON desde String

```
System.out.println("String raiz:\n" + raizStr + "\n");

JsonReader lector = Json.createReader(new StringReader(raizStr));

JsonObject raiz = lector.readObject();

JsonObject reguladorObject = raiz.getJsonObject("regulador");

JsonArray condensadores = reguladorObject.getJsonArray("condensadores");

for (JsonValue condensador : condensadores) {
```

```
for (JsonValue condensador : condensadores) {
    JsonObject capacitor = (JsonObject) condensador;
    String ID = capacitor.getString("ID");
    String tipo = capacitor.getString("tipo");
    String fabricante = capacitor.getString("fabricante");
    JsonObject valor = capacitor.getJsonObject("valor");
    double capacidad = valor.getJsonNumber("magnitud").doubleValue();
    String unidades_capacidad = valor.getString("unidades");
```

```
"condensadores" : [
        {
          "ID":"C1",
          "tipo":"ceramico",
          "fabricante":"Murata",
          "valor":{"magnitud":0.47, "unidades":"uF"},
          "voltaje":{"magnitud":50, "unidades":"V"},
          "encapsulado":"0805",
          "disipador":false
        }
```

**XML:** Se puede codificar cada carácter con entre 1 a 4 octetos. Hay un elemento raíz que incluye varios elementos. Cada elemento tiene unos atributos que lo definen y unos elementos que dan sus características o uso.

Hay un espacio de nombres xmlns. Es una forma de identificar la pertenencia de ciertos atributos a cierta categoría para evitar ambigüedades.

# *Tema 2.1: Intro Android*

Hay distintos estados para las actividades:

- Resumed: En ejecución en primer plano.
- Paused: Parcialmente visible y no se puede interactuar.
- Stopped: Esta oculta.
- Destroyed: Finalizada.
- Created o starded

# *TEMA 2.2: Vistas y Layouts*

Los views son todo aquellos que podemos ver e interactuar con ello. El layout es la disposición geométrica de los objetos de tipo view.

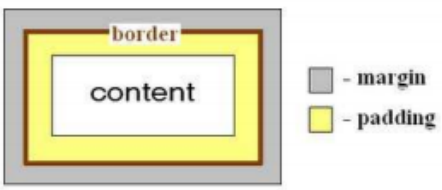
El objeto view tiene un id, unas dimensiones, una posición y un relleno/márgenes.

En cuanto al tamaño, se le puede dar con unidades, o se le puede dar en base a la necesidad (wrap-content), al padre (match_parent) o al constraint (match_constraint), este último importante.

Dip=px*(160/dpi) o px=dip*(dpi/160)

**Layouts**

- **Layout_Weight:** Peso relativo con respecto a los otros elementos contenidos en el layout.
- **Layout_height:** Altura
- **Layout_width:** Anchura
- **Layout_gravity:** Justificación para que lo tenga en cuenta el padre. (afecta  a un elemento view).
- **Gravity:** Justifica el contenido del layout. (afecta a todo el layout)
- **Orientation:** Vertical o horizontal
- **Padding:** Relleno entre el borde y el contenido
- **Margin:** Relleno entre el borde y otro borde.




Si das wrap_content a todo y 1 de peso son equivalentes. Si le das 0dp y peso igual son exactamente iguales.

LinearLayout sirve para agrupar unos cuantos objetos de la clase view y que puedan referenciarse entre sí.

Aparte se puede poner relleno y márgenes (padding y margin).

TableLayout sirve para colocar las cosas de forma cuadriculada (y el span te sirve para decir cuantas cuadriculas ocupa determinado texto o botón, como si fuera el weight pero en el rango del máximo espacio ocupado).

Frame Layout va haciendo capas.

Relative Layout sirve para colocar las cosas en comparación con los otros elementos. La clave son los constraints. No hay orientation.

**Recursos:**

En los recursos tenemos drawables (fotos), listas de strings (se usan para listas o spinners). Se accede desde @[paquete:]drawable(o string)/fichero

# *Tema 2.3: Controles de la interfaz*

setOnClickListener (objeto) es lo que le da funcionalidad a clickar un botón. Debe implementar View.onClickListener. Se implementa en el onCreate, después de seleccionar la interfaz que quieres en el setContentView. Ej: findViewById(R.id.boton).setOnClickListener(this).

Después, se debe implementar un método onClick en el que se dará la funcionalidad a la acción de clicar el botón.

Los bundles sirven para guardar variables de una actividad a otra. En la clase **onSaveInstanceState**, el método savedInstanceState.putX() guarda las variables, y también llama a la clase superior para preservar la jerarquía: super.onSaveInstanceState(savedInstanceState). Esto se ejecuta después de onStop

```
static final String PUNTOS = "PUNTOS";
static final String NIVEL = "NIVEL";
// ...


@Override
public void onSaveInstanceState(Bundle savedInstanceState) {
    // Save the user's current game state
    savedInstanceState.putInt(PUNTOS, puntuacion);
    savedInstanceState.putInt(NIVEL, nivel_juego);

    // Always call the superclass so it can save the view hierarchy state
    super.onSaveInstanceState(savedInstanceState);
}
```

En la actividad posterior, en la clase **onRestoreInstanceState**, emplearemos super.onRestoreInstanceState para restaurar lo anterior, y definiremos la variable como savedInstanceState.getX(Parámetros).

```
public void onRestoreInstanceState(Bundle savedInstanceState) {
    // Always call the superclass so it can restore the view hierarchy
    super.onRestoreInstanceState(savedInstanceState);

    // Restore state members from saved instance
    puntos = savedInstanceState.getInt(PUNTOS);
    nivel_juego = savedInstanceState.getInt(NIVEL);
}
```

Para lanzar una actividad, tras la acción deseada (pulsar un botón, por ejemplo, sería en el onClick) se ejecuta

```
Intent intento = new Intent(MainActivity.this,actividad_configuracion.class);
startActivity(intento);
```

Hay que declarar las actividades en el AndroidManifest.

También se usa startActivityForResult(intento, Codigo_Peticion) y se declara un método onActivityResult() para hacer algo si se obtiene el resultado esperado.

Se pueden hacer listas (ListView), tablas (GridView) o spinners con arrays de string (res/strings.xml).

En los ListViews, el adaptador sirve para que se carguen los elementos de la lista.

Se puede comprobar el estado de un objeto heredado de View mediante isChecked().

El botón UP o HOME implica el retorno a cierta actividad indicada en el manifest, destruyéndola y creándola de nuevo.

El botón BACK sirve para retornar a la actividad invocante tal y como se encontraba en el momento de la invocación.

# *Tema 2.4: Tareas asíncronas*

Para actividades muy largas se pueden usar varias opciones.

Por ejemplo, runOnUiThreade(Runnable tarea) o un handler en el hilo alternativo.

Handler: Las tareas encoladas por el Handler se ejecutarán en el hilo donde se declara (habitualmente el UI), y los métodos del handler pueden ejecutarse en cualquier hilo.

Asynctask: Tienes que implementarla en la tarea asíncrona(extends AsyncTask) y tiene tres entradas (parámetros, progreso y resultado). En ella debes implementar un método *Resultado doInBackground(Parámetros...parámetros)*

Y a su vez se pueden hacer los métodos onPostExecute, onPreExecute, onProgressUpdate y onCancelled.

Para arrancarla, se emplea tarea.execute(parámetros) y tarea.executeOnExecutor(AsyncTask.Thread_pool_executor,parámetros).

# *Tema 5: Geolocalización y sensores*

Para obtener información de ciertos aspectos se deben incluir peticiones de permisos en el AndroidManifest.xml (por ejemplo, para acceder a la posición del terminal). Fino incluye grueso, y el de internet no suele ser necesario explicitarlo. Hay permisos normales y peligrosos (en estos hay que pedir permiso al usuario). El permiso se comprueba con ContextCompat.checkSelfPermission

El objeto LocationManager se crea invocando el getSystemService(Location_Service) y será el manager de la localización.

Hay varios proveedores, por ejemlo, LocationManager.GPS_PROVIDER("gps"). Se pueden pedir los proveedores (gps, network, passive)

El método LocationManager.requestLocationUpdates(proveedor, minTimee, minDist, locListener) pide actualizaciones de posición.

Se puede pausar el GPS, o reanudarlo, para consumir menos recursos.

Hay una forma de preguntar si el GPS está encendido, se hace en el onresume con el método isProviderEnabled.

Para conseguir una clave API de Google Maps necesitas una cuenta de Google, la huella digital SHA-1 y el nombre del paquete.

Para usar la API de Google maps no hace falta GPS ni aceleromentro, solo hace falta una clave de API.

Para usar geolocalización solo hace falta el permiso, no hace falta la clave API.

# *Tema 2.7: Actividades e intentos*

# *Práctica 1*

Usamos el formato NTP Timestamp format, que usa 32 bits sin signo para representar la cantidad de segundos desde el 1 de enero de 1900.

El protocolo NTP solo emplea paquetes UDP. En la práctica usamos paquetes UDP de 48 octetos para enviar la solicitud al servidor y para recibir la respuesta.

Los únicos campos que modificamos del paquete UDP son la versión (VN=4) y el modo (Mode=3) lo que da un valor en el octeto 0 de 0010 0011=0x23=35.

Para enviar el paquete, creamos un datagram packet con el puerto, la longitud, la dirección IP destino y el paquete a enviar, y a su vez, creamos un datagram packet que funcionará como Buffer para recibir el envío, necesitando solo el paquete a recibir y su longitud.

Para hacer uso de los datagram packets, crearemos un socket que envie la solicitud y reciba la respuesta con el socket.send y socket.receive.

La parte que nos interesa de la respuesta son los octetos 40-43 (parte entera, la parte fraccionaria se obvia).

**deArray4BytesaLong:** Metemos en aux el byte correspondiente de lo recibido en UDP (desde el 40 al 43), y con ello, le hacemos una AND con FF para que se quite la extensión de signo. Una vez hecho ello, se debe desplazar bits para coger la posición del long que le toca (primero 24, luego 16, luego 8 y luego no se desplaza). Tras ello, se hace una OR con tiempo para que se vayan acumulando ahí y el auxiliar quede igual para su siguiente uso.

**deUTCaHMS:** Para conseguir el tiempo correspondiente, bastará con hacer tiempo%86400/3600 para obtener las horas, tiempo%3600/60 para obtener los minutos y tiempo%60 para los segundos. Ello quedará almacenado en un vector hms de bytes de 3 posiciones que será lo que mostraremos

# *Práctica 2*

Se empieza pidiendo el certificado con:

```
System.setProperty("javax.net.ssl.trustStore","certs/minombre");
```

Hay que importar los paquetes javax.json, okhttp3 y java.io.

Hay que conseguir la API_Key desde el aemet, y después en acceso para desarrolladores obtener la URL adecuada con el municipio elegido.

Hay que hacer dos peticiones HTTP client para conseguir los dos Json. Una vez tenemos el json se crea un lector para obtener la raíz, que en este caso es un array, y de este iremos sacando los distintos objetos a partir de x.getJsonY(posición).

# *Práctica 3*

Se pedirá el permiso de internet.

En el onCreate solo se crea la interfaz y un botón que al ser clickado lanzará la tarea asíncrona inicial.

Para ello, implementamos el AsyncTask con Strings ya que tanto el parámetro (un URL) como el resultado (un Json) son strings. En ella, debemos sobreescribir el método doInBackground, el cual implementará el método API_REST que es el que nos permite obtener el Json ya dado.

Como será necesario hacer dos extracciones de Json, en el método onPostExecute comprobaremos si es el primer acceso, y en caso de que sea el primero, obtendremos la URL de los datos del Json, y generaremos una nueva tarea asíncrona (con un n distinto); una vez hecho esto, sería el segundo acceso, y ya obtendremos todos los datos del Json, que asociaremos a TextViews en la pantalla.

La función API_REST dada hará una conexión HTTP en la que a través de Buffers y Readers obtendrá el contenido que hay en la URL.