**PACKAGE: AdvancedMapViewer**

**AdvancedMapViewer**

A map application which uses the features from the mapsforge library. The map can be centered to the current GPS coordinate. A simple file browser for selecting the map file is also included. Some preferences can be adjusted via the EditPreferences activity and screenshots of the map may be taken in different image formats.

**import** java.io.File;

**import** java.io.FileFilter;

**import** java.io.IOException;

**import** java.text.DateFormat;

**import** java.util.Date;

**import** org.mapsforge.android.maps.ArrayCircleOverlay;

**import** org.mapsforge.android.maps.GeoPoint;

**import** org.mapsforge.android.maps.MapActivity;

**import** org.mapsforge.android.maps.MapController;

**import** org.mapsforge.android.maps.MapDatabase;

**import** org.mapsforge.android.maps.MapView;

**import** org.mapsforge.android.maps.MapViewMode;

**import** org.mapsforge.android.maps.OverlayCircle;

**import** org.mapsforge.android.maps.MapView.TextField;

**import** android.app.AlertDialog;

**import** android.app.Dialog;

**import** android.content.Context;

**import** android.content.DialogInterface;

**import** android.content.Intent;

**import** android.content.SharedPreferences;

**import** android.graphics.Color;

**import** android.graphics.Paint;

**import** android.graphics.Rect;

**import** android.graphics.Bitmap.CompressFormat;

**import** android.graphics.drawable.AnimationDrawable;

**import** android.location.Location;

**import** android.location.LocationListener;

**import** android.location.LocationManager;

**import** android.location.LocationProvider;

**import** android.os.Bundle;

**import** android.os.Environment;

**import** android.os.PowerManager;

**import** android.os.PowerManager.WakeLock;

**import** android.preference.PreferenceManager;

**import** android.view.LayoutInflater;

**import** android.view.Menu;

**import** android.view.MenuItem;

**import** android.view.MotionEvent;

**import** android.view.View;

**import** android.view.WindowManager;

**import** android.view.View.OnClickListener;

**import** android.widget.EditText;

**import** android.widget.ImageView;

**import** android.widget.SeekBar;

**import** android.widget.TextView;

**import** android.widget.Toast;

**CacheSizePreference**

Preferences class for adjusting the cache size.

**import** org.mapsforge.android.maps.MapView;

**import** android.content.Context;

**import** android.util.AttributeSet;

**EditPreferences**

Activity to edit the application preferences.

**import** android.os.Bundle;

**import** android.preference.PreferenceActivity;

**import** android.preference.PreferenceManager;

**import** android.view.WindowManager;

**FilePicker**

A FilePicker displays the contents of directories. The user can navigate within the file system and select a single file whose path is then returned to the calling activity. The ordering of directory contents can be specified via {@link #setFileComparator(Comparator)}. By default subfolders and files are grouped and each group is ordered alphabetically.

<p>A {@link FileFilter} can be activated via {@link #setFileDisplayFilter(FileFilter)} to restrict the displayed files and folders. By default all files and folders are visible.<p>

Another <code>FileFilter</code> can be applied via {@link #setFileSelectFilter(FileFilter)} to check if a selected file is valid before its path is returned. By default all files are considered as valid and can be selected by the user.

**import** java.io.File;

**import** java.io.FileFilter;

**import** java.util.Arrays;

**import** java.util.Comparator;

**import** android.app.Activity;

**import** android.app.AlertDialog;

**import** android.app.Dialog;

**import** android.content.Intent;

**import** android.content.SharedPreferences;

**import** android.content.SharedPreferences.Editor;

**import** android.os.Bundle;

**import** android.preference.PreferenceManager;

**import** android.view.View;

**import** android.view.WindowManager;

**import** android.widget.AdapterView;

**import** android.widget.GridView;

**FilePickerIconAdapter**

An adapter for the FilePicker GridView.

**import** java.io.File;

**import** android.content.Context;

**import** android.view.Gravity;

**import** android.view.View;

**import** android.view.ViewGroup;

**import** android.widget.BaseAdapter;

**import** android.widget.TextView;

**InfoView**

Simple activity to display the info web page from the assets folder.

**import** android.app.Activity;

**import** android.os.Bundle;

**import** android.preference.PreferenceManager;

**import** android.view.WindowManager;

**import** android.webkit.WebView;

**MoveSpeedPreference**

Preferences class for adjusting the move speed.

**import** android.content.Context;

**import** android.util.AttributeSet;

**SeekBarPreference**

This abstract class provides all code for a seek bar preference. Deriving classes only need to set the current and maximum value of the seek bar. An optional text message above the seek bar is also supported as well as an optional current value message below the seek bar.

**import** android.content.Context;

**import** android.content.DialogInterface;

**import** android.content.SharedPreferences;

**import** android.content.SharedPreferences.Editor;

**import** android.preference.DialogPreference;

**import** android.preference.PreferenceManager;

**import** android.util.AttributeSet;

**import** android.view.Gravity;

**import** android.view.View;

**import** android.widget.LinearLayout;

**import** android.widget.SeekBar;

**import** android.widget.TextView;

**import** android.widget.SeekBar.OnSeekBarChangeListener;

**PACKAGE: MapForges**

**ArrayCircleOverlay**

ArrayCircleOverlay is a thread-safe implementation of the {@link CircleOverlay} class using an {@link ArrayList} as internal data structure. Default paints for all {@link OverlayCircle OverlayCircles} without individual paints can be defined via the constructor.

<p>The ArrayCircleOverlay handles tap events on CircleOverlays by displaying their title in an {@link AlertDialog}. To change this behavior, override the {@link #onTap(int)} method.<p>

**import** java.util.ArrayList;

**import** java.util.Collection;

**import** org.mapsforge.android.maps.MapView.TextField;

**import** android.app.AlertDialog;

**import** android.app.AlertDialog.Builder;

**import** android.content.Context;

**import** android.graphics.Paint;

**import** android.graphics.Canvas;

**import** android.graphics.Path;

**ArrayItemizedOverlay**

ArrayItemizedOverlay is a thread-safe implementation of the {@link ItemizedOverlay} class using an {@link ArrayList} as internal data structure. A default marker for all {@link OverlayItem OverlayItems} without an individual marker can be defined via the.

<p>The ArrayItemizedOverlay handles tap events on OverlayItems by displaying their title and description in an {@link AlertDialog}. To change this behavior, override the {@link #onTap(int)} method. <p>

**import** java.util.ArrayList;

**import** java.util.Collection;

**import** org.mapsforge.android.maps.MapView.TextField;

**ArrayWay**

ArrayWayOverlay is a thread-safe implementation of the {@link WayOverlay} class using an {@link ArrayList} as internal data structure. Default paints for all {@link OverlayWay OverlayWays} without individual paints can be defined via the constructor.

**import** android.app.AlertDialog;

**import** android.app.AlertDialog.Builder;

**import** android.content.Context;

**import** android.graphics.drawable.Drawable;

**CanvasRenderer**

A map renderer which uses a Canvas for drawing.

**import** java.util.ArrayList;

**import** java.util.Collection;

**import** android.graphics.Paint;

**import** java.util.ArrayList;

**import** android.graphics.Bitmap;

**import** android.graphics.Canvas;

**import** android.graphics.Color;

**import** android.graphics.Matrix;

**import** android.graphics.Paint;

**import** android.graphics.Path;

**import** android.graphics.Typeface;

**CircleContainer**

**CircleOverlay**

CircleOverlay is an abstract base class to display {@link OverlayCircle OverlayCircles}. The class defines some methods to access the backing data structure of deriving subclasses. Besides organizing the redrawing process it handles tap events from the user to check if an OverlayCircle has been touched and {@link #onTap(int)} must be executed.

<p>The overlay may be used to indicate positions which have a known accuracy, such as GPS fixes. The radius of the circles is specified in meters and will be automatically converted to pixels at each redraw. <p>

**import** java.util.ArrayList;

**import** android.graphics.Canvas;

**import** android.graphics.Paint;

**import** android.graphics.Path;

**import** android.graphics.Point;

**CoastlineAlgorithm**

The CoastlineAlgorithm generates closed polygons from disjoint coastline segments. The algorithm is based on the close-areas.pl script, written by Frederik Ramm for the Osmarender program. This implementation is optimized for high performance and memory reusing.

**import** java.util.ArrayList;

**import** java.util.Collections;

**import** java.util.Comparator;

**import** java.util.HashSet;

**import** java.util.TreeMap;

**CoastlineWay**

A CoastlineWay is a special way to outline a sea or an island.

**DatabaseMapGenerator**

A MapGenerator that reads map data from a database and renders them.

**import** java.util.ArrayList;

**import** android.graphics.Bitmap;

**import** android.graphics.Color;

**import** android.graphics.DashPathEffect;

**import** android.graphics.Paint;

**import** android.graphics.Typeface;

**import** android.graphics.Paint.Align;

**DependencyCache**

This class process the methods for the Dependency Cache. It's connected with the LabelPlacement class. The main goal is, to remove double labels and symbols that are already rendered, from the actual tile. Labels and symbols that, would be rendered on an already drawn Tile, will be deleted too.

**import** java.util.ArrayList;

**import** java.util.Hashtable;

**import** java.util.LinkedList;

**import** android.graphics.Bitmap;

**import** android.graphics.Paint;

**import** android.graphics.Rect;

**Deserializer**

This class converts byte arrays to numbers.

**GeoPoint**

A GeoPoint represents an immutable pair of latitude and longitude coordinates. Both values are internally stored as integer numbers.

**ImmutablePoint**

An ImmutablePoint represents an fixed pair of float coordinates.

**IndexCacheEntryKey**

An immutable container class which is the key for the index cache.

**ItemizedOverlay**

ItemizedOverlay is an abstract base class to display {@link OverlayItem OverlayItems}. The class defines some methods to access the backing data structure of deriving subclasses. Besides organizing the redrawing process it handles tap events from the user to check if an OverlayItem has been touched and {@link #onTap(int)} must be executed.

**import** java.util.ArrayList;

**import** android.graphics.Canvas;

**import** android.graphics.Point;

**import** android.graphics.Rect;

**import** android.graphics.drawable.Drawable;

**LabelPlacement**

This class place the labels form POIs, area labels and normal labels. The main target is avoiding collisions of these different labels.

**import** java.util.ArrayList;

**import** java.util.Comparator;

**import** java.util.LinkedList;

**import** java.util.PriorityQueue;

**LayersIds**

List of all layers that are used for correct rendering.

**LineClipping**

Fast implementation of the Cohen-Sutherland line clipping algorithm.

**Logger**

Class used for logging text to the console.

**import** android.util.Log;

**MapActivity**

MapActivity is the abstract base class which must be extended in order to use a {@link MapView}. There are no abstract methods in this implementation that subclasses need to override. In addition, no API key or registration is required.

<p>A subclass may create a MapView either via one of the MapView constructors or by inflating an XML layout file. It is possible to use more than one MapView at the same time as each of them works independently from the others.<p>

When the MapActivity is shut down, the current center position, zoom level and map file of the MapView are saved in a preferences file and restored automatically during the setup process of a MapView.

**import** java.util.ArrayList;

**import** android.app.Activity;

**import** android.content.SharedPreferences;

**import** android.content.SharedPreferences.Editor;

**MapController**

A MapController is used to programmatically modify the position and zoom level of a map. Each MapController is assigned to a single MapView instance. To retrieve a MapController for a given MapView, call the {@link MapView#getController()} method.

**import** android.view.KeyEvent;

**import** android.view.View;

**MapDatabase**

A database class for reading binary map files. Byte order is big-endian.

**import** java.io.File;

**import** java.io.IOException;

**import** java.io.RandomAccessFile;

**import** java.io.UnsupportedEncodingException;

**import** java.util.HashMap;

**import** android.graphics.Rect;

**MapDatabaseIndexCache**

A cache for database index blocks with a fixed size and LRU policy.

**import** java.io.IOException;

**import** java.io.RandomAccessFile;

**import** java.util.LinkedHashMap;

**import** java.util.Map;

**MapFileParameters**

Holds all parameters of a map file.

**import** android.graphics.Rect;

**MapGenerator**

A MapGenerator provides map images. This abstract base class handles all thread specific actions and provides the queue for jobs, which need to be processed and scheduled.

**import** android.graphics.Bitmap;

**MapGeneratorJob**

A container class that holds all immutable rendering parameters for a single map image together with a mutable priority field, which indicates the importance of this task.

**import** java.io.IOException;

**import** java.io.ObjectInputStream;

**import** java.io.Serializable;

**MapMover**

A MapMover moves the map horizontally and vertically at a constant speed. It runs in a separate thread to avoid blocking the UI thread.

**import** android.os.SystemClock;

**MapnikTileDownload**

A MapGenerator that downloads tiles from the Mapnik server at OpenStreetMap.

**MapPatterns**

This class holds all patterns that can be rendered on the map. All bitmaps are created when the MapPatterns constructor is called and are recycled when the recycle() method is called.

**import** android.graphics.Bitmap;

**import** android.graphics.BitmapFactory;

**import** android.graphics.BitmapShader;

**import** android.graphics.Shader;

**import** android.graphics.Shader.TileMode;

**MapSymbols**

This class holds all symbols that can be rendered on the map. All bitmaps are created when the MapSymbols constructor is called and are recycled when the recycle() method is called.

**import** android.graphics.Bitmap;

**import** android.graphics.BitmapFactory;

**MapView**

A MapView shows a map on the display of the device. It handles all user input and touch gestures to move and zoom the map. This MapView also comes with an integrated scale bar, which can be activated via the {@link #setScaleBar(boolean)} method. The built-in zoom controls can be enabled with the {@link #setBuiltInZoomControls(boolean)} method. The {@link #getController()} method returns a <code>MapController</code> to programmatically modify the position and zoom level of the map.

<p>This implementation supports offline map rendering as well as downloading map images (tiles) over an Internet connection. All possible operation modes are listed in the {@link MapViewMode} enumeration. The operation mode of a MapView can be set in the constructor and changed at runtime with the {@link #setMapViewMode(MapViewMode)} method. Some MapView parameters like the maximum possible zoom level or the default starting point depend on the selected operation mode.<p>

In offline rendering mode a special database file is required which contains the map data. Such map files can be stored in any readable folder. The current map file for a MapView is set by calling the {@link #setMapFile(String)} method. To retrieve a <code>MapDatabase</code> that returns some metadata about the map file, use the {@link #getMapDatabase()} method.

<p>Map tiles are automatically cached in a separate directory on the memory card. The size of this cache may be adjusted via the {@link #setMemoryCardCacheSize(int)} method. The {@link MapView#setMemoryCardCachePersistence(boolean)} method sets the cache persistence.<p>

{@link Overlay Overlays} can be used to display geographical data such as points and ways. To draw an overlay on top of the map, add it to the list returned by {@link #getOverlays()}. Overlays may be added or removed from the list at any time.

<p>All text fields from the {@link TextField} enumeration can be overridden at runtime via the

{@link #setText(TextField, String)} method. The default texts are in English.<p>

**import** java.io.File;

**import** java.io.FileOutputStream;

**import** java.io.IOException;

**import** java.nio.ByteBuffer;

**import** java.util.ArrayList;

**import** java.util.Collection;

**import** java.util.Collections;

**import** java.util.List;

**import** android.content.Context;

**import** android.graphics.Bitmap;

**import** android.graphics.Canvas;

**import** android.graphics.Color;

**import** android.graphics.Matrix;

**import** android.graphics.Paint;

**import** android.graphics.Typeface;

**import** android.graphics.Bitmap.CompressFormat;

**import** android.os.Build;

**import** android.os.Environment;

**import** android.os.Handler;

**import** android.os.Message;

**import** android.os.SystemClock;

**import** android.util.AttributeSet;

**import** android.view.KeyEvent;

**import** android.view.MotionEvent;

**import** android.view.ScaleGestureDetector;

**import** android.view.View;

**import** android.view.ViewConfiguration;

**import** android.view.ViewGroup;

**import** android.widget.ZoomControls;

**MapViewMode**

The MapViewMode enumeration lists all possible {@link MapView} operating modes. To check if a MapViewMode requires an Internet connection, use the {@link #requiresInternetConnection()} method.

**MercatorProjection**

A performance optimized implementation of the spherical Mercator projection.

**import** android.graphics.Point;

**OpenCycleMapTileDownload**

A MapGenerator that downloads tiles from the OpenCycleMap server.

**OsmarendererTileDownload**

A MapGenerator that downloads tiles from the TilesAtHome server at OpenStreetMap.

**Overlay**

Overlay is the abstract base class for all types of overlays. It handles the lifecycle of the overlay thread and implements those parts of the redrawing process which all overlays have in common.

<p>To add an overlay to a <code>MapView</code>, create a subclass of this class and add an instance to the list returned by {@linkMapView#getOverlays()}. When an overlay gets removed from the list, the corresponding thread is automatically interrupted and all its resources are freed. Re-adding a previously removed overlay to the list will therefore cause an {@link IllegalThreadStateException}.<p>

**import** android.graphics.Bitmap;

**import** android.graphics.Canvas;

**import** android.graphics.Color;

**import** android.graphics.Matrix;

**import** android.graphics.Point;

**OverlayCircle**

OverlayCircle holds all parameters of a single circle on a {@link CircleOverlay}. All rendering parameters like color, stroke width, pattern and transparency can be configured via two {@link Paint} objects. Each circle is drawn twice - once with each paint object – to allow for different outlines and fillings.

**import** android.graphics.Paint;

**import** android.graphics.Point;

**OverlayItem**

OverlayItem holds all parameters of a single element on an {@link ItemizedOverlay}, such as position, marker, title and textual description.

**import** android.graphics.Point;

**import** android.graphics.drawable.Drawable;

**OverlayWay**

OverlayWay holds all parameters of a single way on a {@link WayOverlay}. All rendering parameters like color, stroke width, pattern and transparency can be configured via two {@link Paint} objects. Each way is drawn twice - once with each paint object - to allow for different outlines and fillings.

**import** android.graphics.Paint;

**import** android.graphics.Point;

**PointTextContainer**

**import** android.graphics.Paint;

**import** android.graphics.Rect;

**Projection**

A Projection translates between the pixel coordinate system on the screen and geographical points on the earth. To retrieve the currently used Projection for a given MapView, call the {@link MapView#getProjection()} method.

**import** android.graphics.Point;

**ShapeContainer**

**ShapePaintContainer**

**import** android.graphics.Paint;

**ShapeType**

**SutherlandHodgmanClipping**

Implementation of the Sutherland-Hodgman clipping algorithm.

**SymbolContainer**

**import** android.graphics.Bitmap;

**TagIDsNodes**

**import** java.util.HashMap;

**TagIDsWays**

**import** java.util.HashMap;

**Tile**

A tile represents a rectangular part of the world map. All tiles can be identified by their X and Y number together with their zoom level. The actual area that a tile covers on a map depends on the underlying map projection.

**import** java.io.IOException;

**import** java.io.ObjectInputStream;

**import** java.io.Serializable;

**import** android.graphics.Rect;

**TileDownloadMapGenerator**

A MapGenerator that downloads map tiles from a server. To build an implementation for a certain tile server, extend this class and implement the abstract methods.

**import** java.io.IOException;

**import** java.io.InputStream;

**import** java.net.URL;

**import** java.net.UnknownHostException;

**import** android.graphics.Bitmap;

**import** android.graphics.BitmapFactory;

**TileMemoryCardCache**

A thread-safe cache for image files with a fixed size and LRU policy.

**import** java.io.File;

**import** java.io.FileInputStream;

**import** java.io.FileNotFoundException;

**import** java.io.FileOutputStream;

**import** java.io.FilenameFilter;

**import** java.io.IOException;

**import** java.io.ObjectInputStream;

**import** java.io.ObjectOutputStream;

**import** java.nio.ByteBuffer;

**import** java.util.LinkedHashMap;

**import** java.util.Map;

**import** android.graphics.Bitmap;

**TileRAMCache**

A thread-safe cache for bitmap images with a fixed size and LRU policy.

**import** java.nio.ByteBuffer;

**import** java.util.LinkedHashMap;

**import** java.util.LinkedList;

**import** java.util.Map;

**import** android.graphics.Bitmap;

**WayContainer**

**WayOverlay**

WayOverlay is an abstract base class to display {@link OverlayWay OverlayWays}. The class defines some methods to access the backing data structure of deriving subclasses.

<p>The overlay may be used to show additional ways such as calculated routes. Closed polygons, for example buildings or areas, are also supported. A way node sequence is considered as a closed polygon if the first and the last way node are equal.

**import** android.graphics.Canvas;

**import** android.graphics.Paint;

**import** android.graphics.Path;

**import** android.graphics.Point;

**WayTextContainer**

**import** android.graphics.Paint;

**ZoomAnimator**

**import** android.os.SystemClock;

**Bibliothek**

**JAVA**

**import** java.io.File;

**import** java.io.FileFilter;

**import** java.io.IOException;

**import** java.io.RandomAccessFile;

**import** java.io.UnsupportedEncodingException;

**import** java.io.FileOutputStream;

**import** java.io.ObjectInputStream;

**import** java.io.Serializable;

**import** java.io.InputStream;

**import** java.io.FileNotFoundException;

**import** java.io.FilenameFilter;

**import** java.io.ObjectOutputStream;

**import** java.lang.Comparable;

**import** java.net.URL;

**import** java.net.UnknownHostException;

**import** java.nio.ByteBuffer;

**import** java.text.DateFormat;

**import** java.util.Collection;

**import** java.util.Collections;

**import** java.util.List;

**import** java.util.Date;

**import** java.util.Arrays;

**import** java.util.Comparator;

**import** java.util.ArrayList;

**import** java.util.HashSet;

**import** java.util.TreeMap;

**import** java.util.Hashtable;

**import** java.util.HashMap;

**import** java.util.PriorityQueue;

**import** java.util.LinkedList;

**import** java.util.LinkedHashMap;

**import** java.util.Map;

**ANDROID-APP**

**import** android.app.AlertDialog;

* java.awt.Dialog

**import** android.app.Dialog;

* java.awt.Dialog

**import** android.app.Activity;

* java.

**import** android.app.AlertDialog.Builder;

* java.

**ANDROID-CONTENT**

**import** android.content.Context;

* java.

**import** android.content.DialogInterface;

* java.

**import** android.content.Intent;

* java.

**import** android.content.SharedPreferences;

* java.

**import** android.content.SharedPreferences.Editor;

**ANDROID-GRAPHICS**

**import** android.graphics.Color; //OK

The Color class defines methods for creating and converting color ints. Colors are represented as packed ints, made up of 4 bytes: alpha, red, green, blue. The values are unpremultiplied, meaning any transparency is stored solely in the alpha component, and not in the color components. The components are stored as follows (alpha << 24) | (red << 16) | (green << 8) | blue. Each component ranges between 0..255 with 0 meaning no contribution for that component, and 255 meaning 100% contribution. Thus opaque-black would be 0xFF000000 (100% opaque but no contributions from red, green, or blue), and opaque-white would be 0xFFFFFFFF

* java.awt.Color

**import** android.graphics.Paint; //??

The Paint class holds the style and color information about how to draw geometries, text and bitmaps.

* java.awt.Color

**import** android.graphics.Rect;

Rect holds four integer coordinates for a rectangle. The rectangle is represented by the coordinates of its 4 edges (left, top, right bottom). These fields can be accessed directly. Use width() and height() to retrieve the rectangle's width and height. Note: most methods do not check to see that the coordinates are sorted correctly (i.e. left <= right and top <= bottom).

* java.awt.Rectangle

**import** android.graphics.Bitmap.CompressFormat; //OK

Specifies the known formats a bitmap can be compressed into

**import** android.graphics.BitmapFactory;

Creates Bitmap objects from various sources, including files, streams, and byte-arrays.

* java.awt.

**import** android.graphics.BitmapShader; //OK

Shader used to draw a bitmap as a texture. The bitmap can be repeated or mirrored by setting the tiling mode.

* OK

**import** android.graphics.Shader;

Shader is the based class for objects that return horizontal spans of colors during drawing. A subclass of Shader is installed in a Paint calling paint.setShader(shader). After that any object (other than a bitmap) that is drawn with that paint will get its color(s) from the shader.

* java.awt.

**import** android.graphics.Shader.TileMode; //OK

Enum

* OK

**import** android.graphics.drawable.Drawable;

Provides classes to manage a variety of visual elements that are intended for display only, such as bitmaps and gradients. These elements are often used by widgets as background images or simply as indicators (for example, a volume level indicator). You can create most of these drawables using XML, as described in [Drawable Resources](http://developer.android.com/guide/topics/resources/drawable-resource.html)

* java.awt.

**import** android.graphics.drawable.AnimationDrawable;

An object used to create frame-by-frame animations, defined by a series of Drawable objects, which can be used as a View object's background.

The simplest way to create a frame-by-frame animation is to define the animation in an XML file, placed in the res/drawable/ folder, and set it as the background to a View object. Then, call [run()](http://developer.android.com/reference/android/graphics/drawable/AnimationDrawable.html#run()) to start the animation.

An AnimationDrawable defined in XML consists of a single <animation-list> element, and a series of nested <item> tags. Each item defines a frame of the animation. See the example below.

spin\_animation.xml file in res/drawable/ folder:

* java.awt.

**import** android.graphics.Point; //OK

Point holds two integer coordinates

* java.awt.Point

**import** android.graphics.Bitmap;

-

* java.awt.Image

**import** android.graphics.Canvas;

The Canvas class holds the "draw" calls. To draw something, you need 4 basic components: A Bitmap to hold the pixels, a Canvas to host the draw calls (writing into the bitmap), a drawing primitive (e.g. Rect, Path, text, Bitmap), and a paint (to describe the colors and styles for the drawing).

* java.awt.Canvas

**import** android.graphics.Matrix; //OK

The Matrix class holds a 3x3 matrix for transforming coordinates. Matrix does not have a constructor, so it must be explicitly initialized using either reset() - to construct an identity matrix, or one of the set..() functions (e.g. setTranslate, setRotate, etc.)

* java.awt.

**import** android.graphics.Typeface;

The Typeface class specifies the typeface and intrinsic style of a font. This is used in the paint, along with optionally Paint settings like textSize, textSkewX, textScaleX to specify how text appears when drawn (and measured).

* java.awt.

**import** android.graphics.Path;

The Path class encapsulates compound (multiple contour) geometric paths consisting of straight line segments, quadratic curves, and cubic curves. It can be drawn with canvas.drawPath(path, paint), either filled or stroked (based on the paint's Style), or it can be used for clipping or to draw text on a path.

* java.awt.GeneralPath

**import** android.graphics.DashPathEffect;

-

* java.awt.

**import** android.graphics.Paint.Align;

Align specifies how drawText aligns its text relative to the [x,y] coordinates. The default is LEFT

* java.awt.

**ANDROID-LOCATION**

**~~import~~** ~~android.location.Location;~~

* java.

**~~import~~** ~~android.location.LocationListener;~~

* java.

**~~import~~** ~~android.location.LocationManager;~~

* java.

**~~import~~** ~~android.location.LocationProvider;~~

* java.

**ANDROID-OS**

**import** android.os.Bundle;

* java.

**import** android.os.Environment;

* java.

**import** android.os.PowerManager;

* java.

**import** android.os.PowerManager.WakeLock;

* java.

**import** android.os.Build;

* java.

**import** android.os.Handler;

* java.

**import** android.os.Message;

* java.

**import** android.os.SystemClock;

* java.

**ANDROID-PREFERENCES**

**import** android.preference.PreferenceManager;

* java.

**import** android.preference.PreferenceActivity;

* java.

**import** android.preference.DialogPreference;

* java.

**ANDROID-UTIL**

**import** android.util.Log;

* java.util.logging.Logger

**import** android.util.AttributeSet;

* java.util.

**import** android.util.SparseArray; //SparseBooleanArray;

* java.util.HashMap / Map / SortMap

**ANDROID-VIEW**

**import** android.view.LayoutInflater;

* java.

**import** android.view.Menu;

* java.awt.Menu

**import** android.view.MenuItem;

* java.awt.MenuItem

**import** android.view.MotionEvent;

* java.

**import** android.view.View;

* java.

**import** android.view.WindowManager;

* java.

**import** android.view.View.OnClickListener;

* java.

**import** android.view.Gravity;

* java.

**import** android.view.ViewGroup;

* java.

**import** android.view.KeyEvent;

* java.

**import** android.view.ScaleGestureDetector;

* java.

**import** android.view.ViewConfiguration;

* java.

**ANDROID-WIDGET**

**import** android.widget.EditText;

* java.

**import** android.widget.ImageView;

* java.

**import** android.widget.SeekBar;

* java.

**import** android.widget.TextView;

* java.

**import** android.widget.AdapterView;

* java.

**import** android.widget.GridView;

* java.

**import** android.widget.BaseAdapter;

* java.

**import** android.widget.LinearLayout;

* java.

**import** android.widget.SeekBar.OnSeekBarChangeListener;

* java.

**import** android.widget.Toast;

* java.

**import** android.widget.ZoomControls;

* java.