Android Search Framework

Sang Shin
Michèle Garoche
www.javapassion.com
"Learn with Passion!"

Disclaimer

- Portions of this presentation are modifications based on work created and shared by the Android Open Source Project
  - http://code.google.com/policies.html
- They are used according to terms described in the Creative Commons 2.5 Attribution License
  - http://creativecommons.org/licenses/by/2.5/

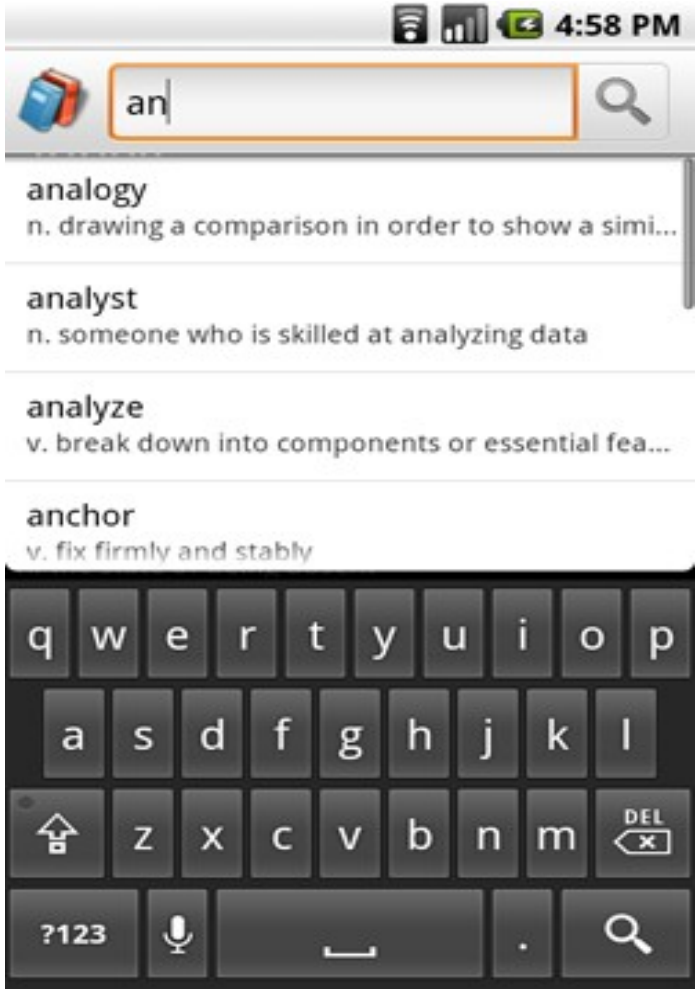
Topics

- Android Search Framework
- Steps of implementing search
- How to write searchable Activity - how to perform search
- Invoking Search Dialog
- Passing Search Context Data
- Adding Voice Search
- Adding recent query suggestions
- Quick Search Box

Android Search Framework

Android Search Framework Provides

- A search dialog
  - Each app does not have to build its own search dialog box
  - It is customizable
- An interaction layer that communicates with your application.
  - The search framework will pass the query text to your application so that your application can begin a search

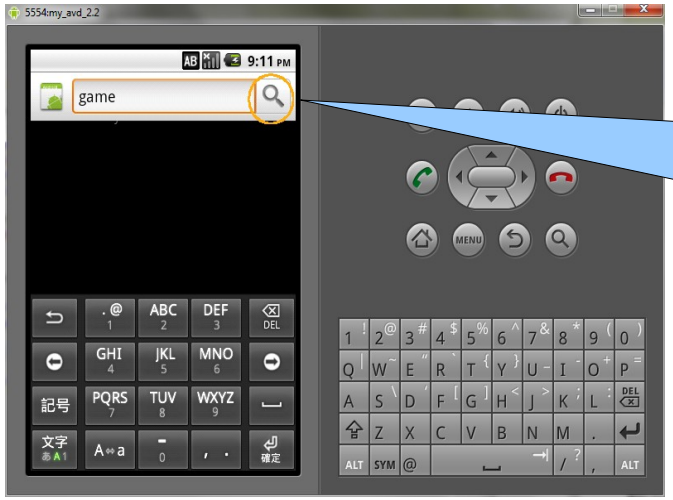


Customization Features

- Customize some of the search dialog characteristics
- Enable voice search
- Provide search suggestions based on recent user queries
- Provide search suggestions that match actual results in your application data
- Offer your application's search suggestions in the system-wide Quick Search Box

What is Search Manager?

- The SearchManager (of the Android Search Framework) manages the search dialog
  - > It manages the life of the Android search dialog
- When the user requests a search (by clicking search icon), it will use a specially-formed Intent to pass the search query to the Activity that you've declared to handle searches (this activity is called “searchable Activity”)



When search icon is clicked, the Search Manager sends the query to the “searchable activity” (onCreate() method of the searchable activity gets invoked.)

Steps of Implementing Search

Steps of Implementing Search

1. Create “searchable configuration” XML file
2. Create and declare “searchable Activity” class
3. Choose activities that function as “searchable contexts” and declare them in the *AndroidManifest.xml* file

1. Create a Searchable Configuration

- The searchable configuration is an XML file that defines several settings for the Android search dialog in your application.
- This file is typically named as *searchable.xml* and must be saved in the *res/xml/* directory

Example: Searchable Configuration

```
<!-- The attributes in this XML file provide configuration information -->
<!-- for the Search Manager. -->

<searchable xmlns:android="http://schemas.android.com/apk/res/android"
    android:label="@string/search_label"
    android:hint="@string/search_hint"

    android:voiceSearchMode="showVoiceSearchButton|launchRecognizer"
    android:voiceLanguageModel="free_form"
    android:voicePromptText="@string/search_invoke"

    android:searchSuggestAuthority="com.example.android.apis.SuggestionProvider"
    android:searchSuggestSelection=" ? "
/>
```

2. Create/Declare Searchable Activity

- Searchable activity receives the search query from the framework then performs search and displays the search results.
  - When the user executes a search from the search dialog, the Search Manager will send the search query to the your searchable Activity with an Intent set with *ACTION_SEARCH* action
  - Your searchable Activity will then search your data and present the results.
- You have to declare the searchable Activity in AndroidManifest.xml with *ACTION_SEARCH* Intent as a filtered action

Declaring a Searchable Activity in the AndroidManifest.xml

```
<application ... >
    <activity android:name=".MySearchableActivity" >
        <intent-filter>
            <action android:name="android.intent.action.SEARCH" />
        </intent-filter>
        ...

    </activity>
    ...
</application>
```

3. Declare Searchable Contexts

- The search dialog is not, by default, available from every Activity of your application.
- Rather, the search dialog is presented to users only when they invoke search from a searchable context of your application.
- A searchable context is any Activity for which you have declared searchable meta-data in the manifest file.

Declaring Only Searchable Activity as a sole Searchable Context

```
<!-- In the example below, the searchable Activity itself is a searchable context because it includes meta-data that defines the searchable configuration. Any other Activity in this application is not a searchable context, by default, and thus, does not reveal the search dialog.-->

<application ... >
    <activity android:name=".MySearchableActivity" >
        <intent-filter>
            <action android:name="android.intent.action.SEARCH" />
        </intent-filter>
        <meta-data android:name="android.app.searchable"
                android:resource="@xml/searchable"/>
    </activity>

    <activity android:name=".NotSearchableContextActivity" >
    </activity>
    ...
</application>
```

Declaring All Activities As Searchable Contexts

<!-- If you want all of your activities to provide the search dialog, add another <meta-data> element inside the <application> element. Use this element to declare the existing searchable Activity as the default searchable Activity.-->

```
<application ... >
   <activity android:name=".MySearchableActivity" >
      <intent-filter>
         <action android:name="android.intent.action.SEARCH" />
      </intent-filter>
      <meta-data android:name="android.app.searchable"
                 android:resource="@xml/searchable"/>
   </activity>
   <activity android:name=".AnotherActivity" ... >
   </activity>
   <!-- Declare the default searchable Activity (activity that performs the search) for the whole app -->
   <!-- All other activities in the application, such as AnotherActivity, are now
        considered a searchable context and can invoke the search dialog.
        When a search is executed, MySearchableActivity is launched to
        handle the search query. -->
   <meta-data android:name="android.app.default_searchable"
              android:value=".MySearchableActivity" />
   ...
</application>
```

Demo:
Run “search_helloworld1” & “search_helloworld2”

Exercise_1 of “6129_android_search.zip”

How to Write Searchable Activity: Performing Search

Performing a search

- Performing the actual search (in your searchable activity) involves three steps:
  - Step 1: receiving the query (from the Search Manager of the Android framework)
  - Step 2: searching your data (perform search)
  - Step 3: presenting the results.

Step 1: Receiving the query

- When a user executes a search from the search dialog, the Search Manager sends the *ACTION_SEARCH* Intent to your searchable Activity.
- This Intent carries the search query in the *Extra* field of the Intent
  - In the form of Key-Value pair in which the Key is *SearchManager.QUERY* and the Value is the search query

Step 1: Receiving query

```
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.search);

    Intent intent = getIntent();

    // Extract the query string from Extra.
    // The QUERY string is always included with the ACTION_SEARCH Intent.
    // In this example, the query is retrieved and passed to a local
    // performSearchAndShowResults() method where the actual search
    // operation is done.
    if (Intent.ACTION_SEARCH.equals(intent.getAction())) {
      String query = intent.getStringExtra(SearchManager.QUERY);
      myPerformSearchAndShowResults(query);
    }
}
```

Step 2: Searching your data

- The process of searching your data is unique to your application
- Best practice guidelines
  - If your data is stored in a SQLite database on the device, performing a full-text search (using FTS3, rather than a LIKE query) can provide a more robust search across text data and can produce results many, many times faster.
  - If your data is stored online, then the perceived search performance may be inhibited by the user's data connection. You may want to display a spinning progress wheel until your search returns.

Step 2 Example: Search your data (Query)

```
private void myPerformSearchAndShowResults(String query) {

    // Perform the search query from a content provider
    Cursor cursor = managedQuery(
                        DictionaryProvider.CONTENT_URI,
                        null,                       // Which columns to retrieve
                        null,                       // Which rows to return (all rows)
                        new String[] {query}, // Selection arguments
                        null);                      // Order

    // Display the result
```

Step 3: Presenting search results

- Regardless of where your data lives and how you search it, we recommend that you return search results to your searchable Activity with an Adapter.
  - This way, you can easily present all the search results in a *ListView*.
  - If your data comes from a SQLite database query, then you can easily apply your results to a *ListView* using a *CursorAdapter*.
  - If your data comes in some other type of format, then you can create an extension of the *BaseAdapter*.

Invoking Search Dialog

3 Ways Search Dialog Gets Displayed

- Search Manager (of the platform) displays the search dialog when a user clicks search key
- Your activity displays the search dialog by calling *onSearchRequested()*
- Enable "type-to-search" functionality, which reveals the search dialog when the user starts typing on the keyboard and the keystrokes are inserted into the search dialog.

SEARCH key vs. Search Button UI

- Many Android devices provide a dedicated SEARCH key, which reveals the search dialog when the user presses it from a searchable context of your application.
- However, you should not assume that a SEARCH key is available on the user's device and should always provide a search button in your UI that invokes search.

How to invoke search dialog yourself?

- To invoke search dialog from your Activity (instead of search dialog being invoked by Search Manager), call *onSearchRequested()*
  - For instance, you should provide a menu item in your Options Menu or a button in your UI to invoke search with this method.
- You can also enable "type-to-search" functionality, which reveals the search dialog when the user starts typing on the keyboard and the keystrokes are inserted into the search dialog.
  - You can enable type-to-search in your Activity by calling *setDefaultKeyMode(DEFAULT_KEYS_SEARCH_LOCAL)* during your Activity's *onCreate()* method.

Overriding onSearchRequested()

- Your searchable Activity can also provide overridden onSearchRequested() method
  - To provide custom search behavior
- Examples
  - To provide search context data
  - To prefill query string
  - To force global search, e.g. in response to a dedicated search key, or to block search entirely (by simply returning false)
- Unless overidden, calling this function is the same as calling *startSearch(null, false, null, false)*, which launches search for the current activity as specified in its manifest.

Search Dialog and Activity Lifecycle

- The search dialog is a Dialog that floats at the top of the screen.
- It does not cause any change in the Activity stack, so when the search dialog appears, no lifecycle methods for the currently open Activity (such as onPause()) are called.
  - Your Activity just loses input focus as it is given to the search dialog.

Passing Search Context Data

When to Pass Search Context Data?

- To refine your search criteria from the current Activity instead of depending only on the user's search query, you can provide additional data in the Intent that the Search Manager sends to your searchable Activity.
- Useful when your search criteria varies from one searchable context to another

How Do you Pass Search Context Data

- Override *onSearchRequested()* method for the Activity in which search can be invoked
- Pass whatever data is necessary to refine your search in the APP_DATA Bundle, which is included in the ACTION_SEARCH Intent.

```
// Returning "true" indicates that you have successfully
// handled this callback event.
public boolean onSearchRequested() {
    Bundle appData = new Bundle();
    appData.putBoolean(MySearchableActivity.JARGON, true);
    appData.putString("demo_key", queryAppDataString);
    startSearch(null, false, appData, false);
    return true;
}
```

How Do you Receive Search Context Data

- Then in your searchable Activity, you can extract the data placed inside appdata from the APP_DATA Bundle to refine the search.

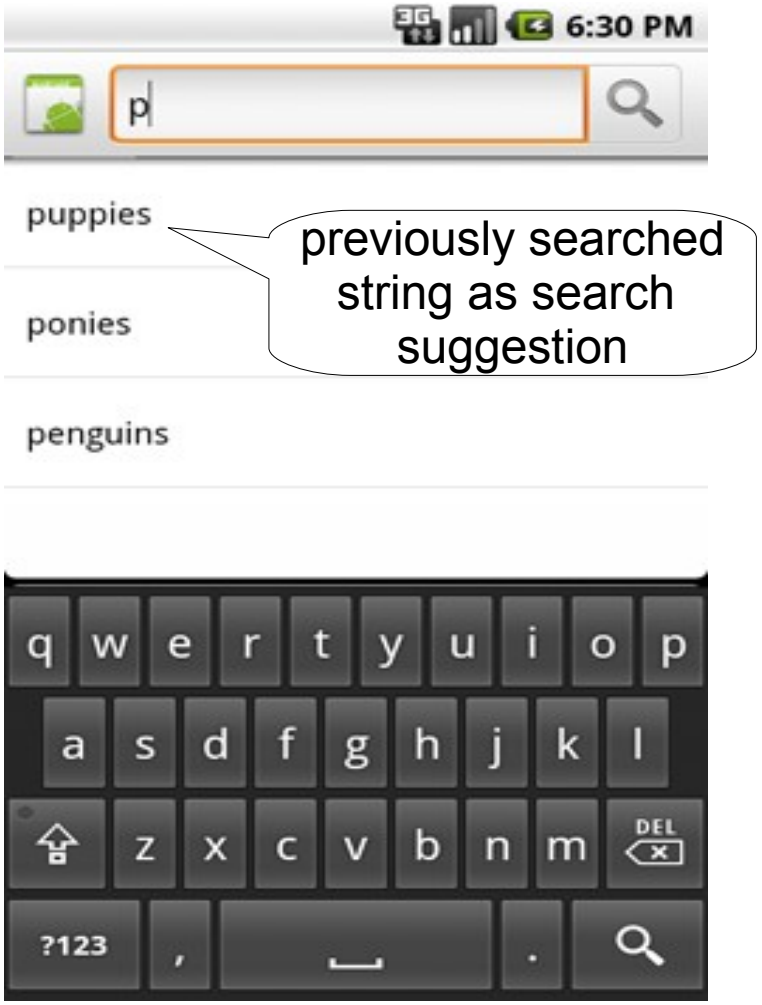
```
Bundle appData =
        getIntent().getBundleExtra(SearchManager.APP_DATA);

if (appData != null) {
    boolean jargon =
        appData.getBoolean(MySearchableActivity.JARGON);
    String demo_key =
        appData.getString("demo_key");
}
```

Adding Recent Query Suggestions

Adding Recent Queries

- When using the Android search dialog, you can provide search suggestions based on recent search queries.
- For example, if a user previously searched for "puppies," then that query appears as a suggestion once he or she begins typing the same query.



Steps To Take

- Step #1 - Create a content provider that extends *SearchRecentSuggestionsProvider* and declare it in your application manifest.
- Step #2 - Add information on the content provider (Suggestion provider) to the searchable configuration
- Step #3 - Add code of saving queries to your content provider each time a search is executed.
- Step #4 - Add code of clearing saved queries (optional)

Search Suggestion Mechanism

- When the Search Manager identifies that your Activity is a searchable activity and provides search suggestions, the following procedure takes place as soon as the user types into the search dialog:
  - Search Manager takes the search query text (whatever has been typed so far) and performs a query to the content provider that contains your suggestions.
  - Your content provider returns a Cursor that points to all suggestions that match the search query text.
  - Search Manager displays the list of suggestions provided by the Cursor.

Adding Recent Query Suggestions:
Step #1:Create and Declare Content Provider

SearchRecentSuggestionsProvider

- The content provider that you need for recent query suggestions must be an implementation of *SearchRecentSuggestionsProvider*.
- This class does practically everything for you. All you have to do is write a class constructor that executes one line of code.

SearchRecentSuggestionsProvider

```
public class MySuggestionProvider
    extends SearchRecentSuggestionsProvider {

    public final static String AUTHORITY
            = "com.example.MySuggestionProvider";

    // DATABASE_MODE_QUERIES mode bit configures the database
    // to record recent queries.
    public final static int MODE
            = DATABASE_MODE_QUERIES;

    // The constructor has a very important responsibility: When it calls
    // setupSuggestions(String, int), it configures this provider to match
    // the requirements of your searchable activity.
    public MySuggestionProvider() {
        setupSuggestions(AUTHORITY, MODE);
    }
}
```

Declare Content Provider

- Declare the content provider in your application manifest with the same authority string used in your SearchRecentSuggestionsProvider class (and in the searchable configuration).

```
<application>
   <provider android:name=".MySuggestionProvider"
             android:authorities="com.example.MySuggestionProvider" />
   ...
</application>
```

Adding Recent Query Suggestions:
Step #2: Add Info on Suggestion Provider to Searchable Configuration

Modify Searchable Configuration

- To configure your search dialog to use your suggestions provider, you need to add the *android:searchSuggestAuthority* and *android:searchSuggestSelection* attributes to the *<searchable>* element in your searchable configuration file.

```
<!-- The value for android:searchSuggestSelection must be a single question mark, preceded by a
space (" ?"), which is simply a placeholder for the SQLite selection argument (which is automatically
replaced by the query text entered by the user).-->
<?xml version="1.0" encoding="utf-8"?>
<searchable xmlns:android="http://schemas.android.com/apk/res/android"
    android:label="@string/app_label"
    android:hint="@string/search_hint"
    android:searchSuggestAuthority="com.example.MySuggestionProvider"
    android:searchSuggestSelection="word MATCH  ?" >
</searchable>
```

Adding Recent Query Suggestions:
Step #3: Add Code of Saving Queries

Saving Queries to Collection of Recent Queries

- To populate your collection of recent queries, add each query received by your searchable Activity to your *SearchRecentSuggestionsProvider*.
- To do this, create an instance of *SearchRecentSuggestions* and call *saveRecentQuery()* each time your searchable Activity receives a query.

Saving Queries to Collection of Recent Queries

```
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);

    Intent Intent  = getIntent();

    if (Intent.ACTION_SEARCH.equals(Intent .getAction())) {
        String query = Intent .getStringExtra(SearchManager.QUERY);

        // Save queries
        SearchRecentSuggestions suggestions = new SearchRecentSuggestions(this,
                MySuggestionProvider.AUTHORITY, MySuggestionProvider.MODE);
        suggestions.saveRecentQuery(query, null);
    }
}
```

Adding Recent Query Suggestions:
Step #4: Add Code of Clearing the Suggestion Data

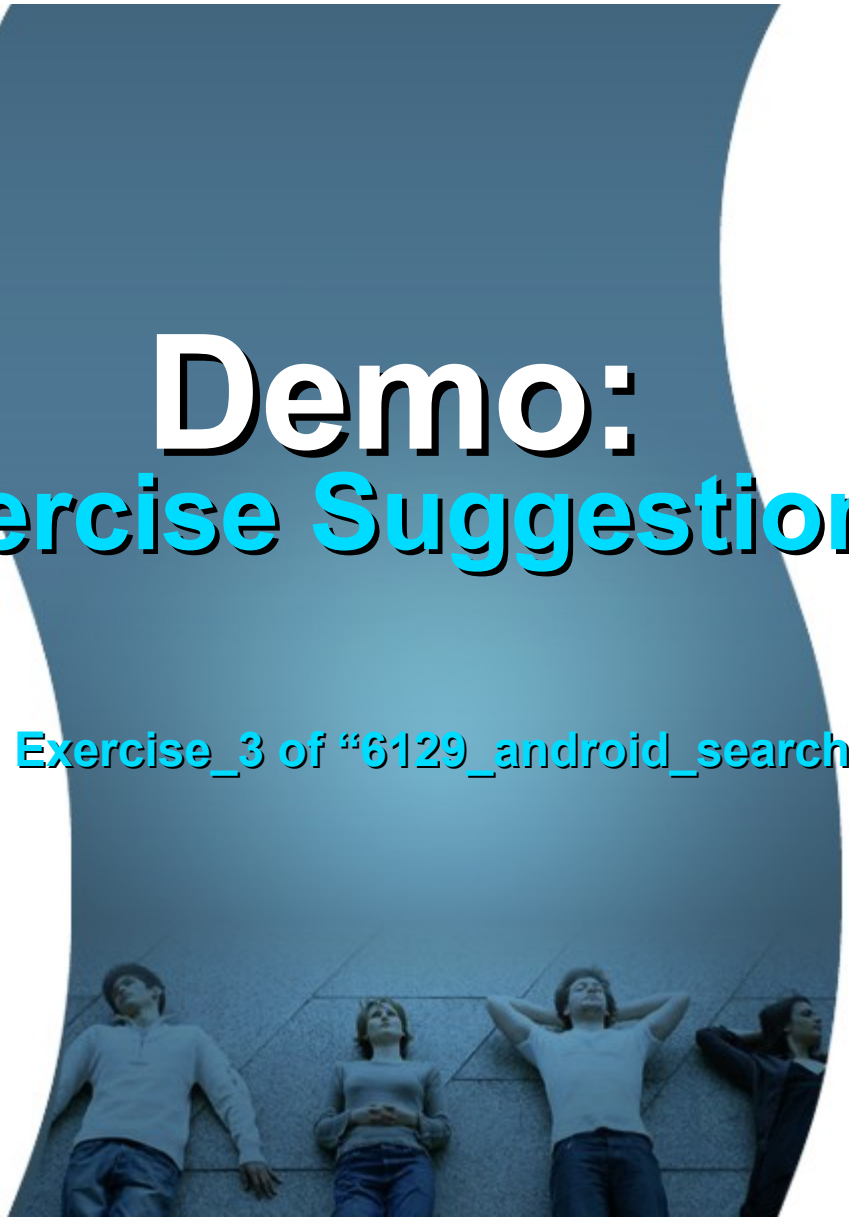
Clearing Suggested Data

- To protect the user's privacy, you should always provide a way for the user to clear the recent query suggestions.
- To clear the query history, call *clearHistory()*.
- Execute this from your choice of a "Clear Search History" menu item, preference item, or button.
  - You should also provide a confirmation dialog to verify that the user wants to delete their search history.

```
SearchRecentSuggestions suggestions = new SearchRecentSuggestions(this,
        HelloSuggestionProvider.AUTHORITY, HelloSuggestionProvider.MODE);
suggestions.clearHistory();
```

Demo:
Exercise Suggestions

Exercise_3 of “6129_android_search.zip”

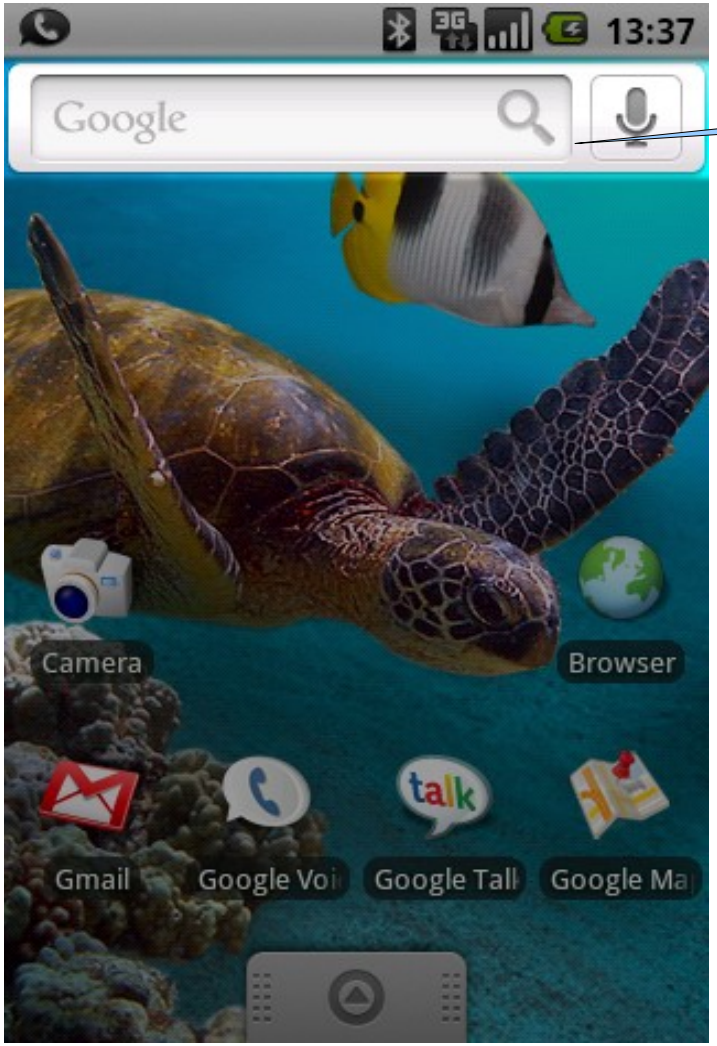


Quick Search Box

What is Quick Search Box?

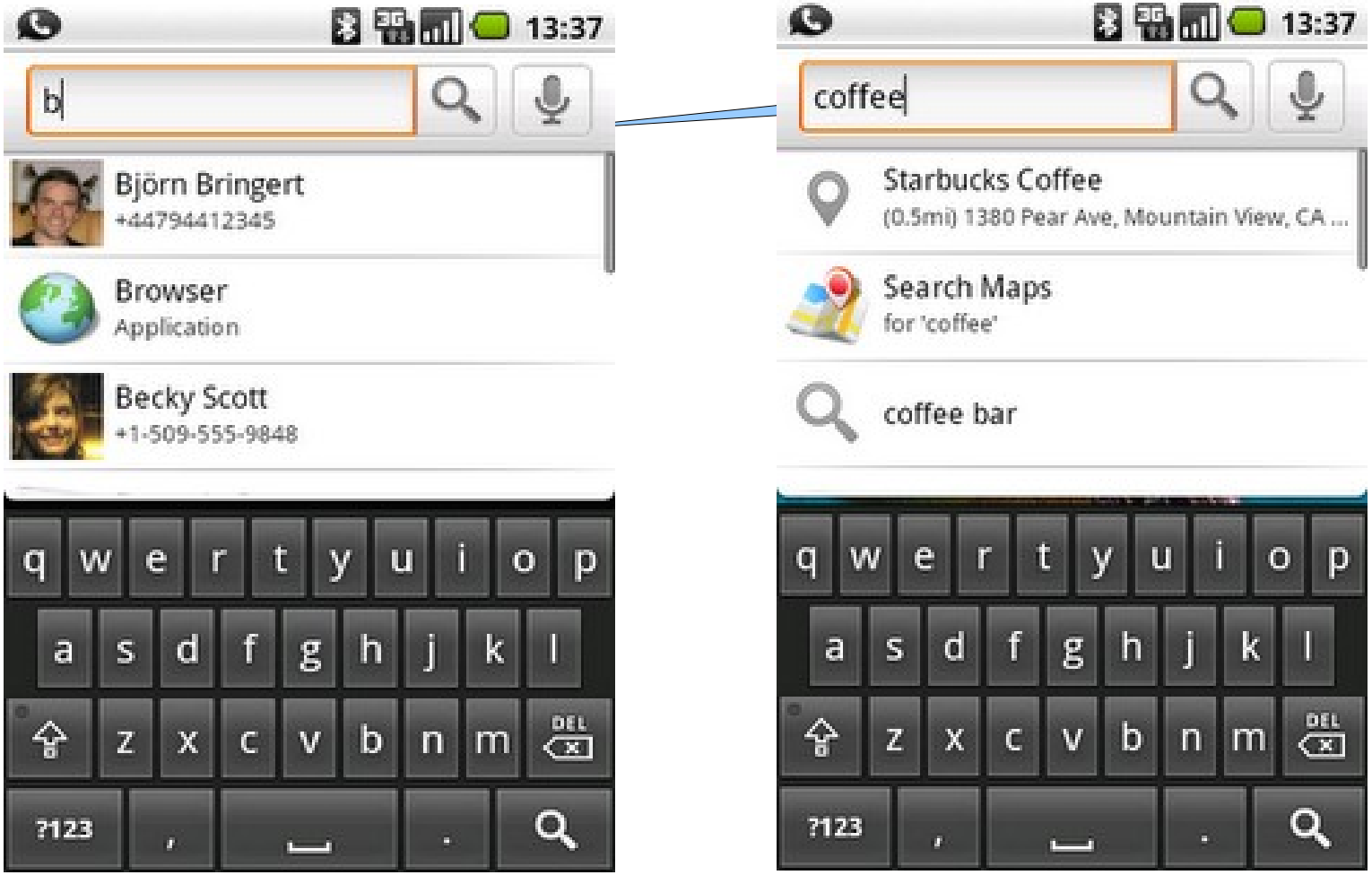
- System-wide search framework
- Available right from the home screen
- Makes it possible for users to quickly and easily find what they're looking for, both on their devices and on the web.
  - > It suggests content on your device as you type, like apps, contacts, browser history, and music.
  - > It also offers results from the web search suggestions, local business listings, and other info from Google, such as stock quotes, weather, and flight status.

Quick Search Box



Quick Search Box

Quick Search Box



Including App Search Suggestions in Quick Search Box (QSB)

- Your apps can provide search suggestions that will surface to users in QSB alongside other search results and suggestions.
- This makes it possible for users to access your application's content from outside your application—for example, from the home screen.

Steps to Take

- Set *android:includeInGlobalSearch* to "*true*" in your application's *searchable.xml*
- Enable particular suggestion sources from the system settings for search

Including Your App's Recent Suggestions in Quick Search Box

```
<!-- By setting android:includeInGlobalSearch to "true" in your application's searchable.xml, you allow Quick Search Box to pick up your search suggestion content provider and include its suggestions along with the rest (if the user enables your suggestions from the system search settings). -->

<?xml version="1.0" encoding="utf-8"?>
<searchable xmlns:android="http://schemas.android.com/apk/res/android"
    android:label="@string/search_label"
    android:hint="@string/search_hint"
    android:voiceSearchMode="showVoiceSearchButton|launchRecognizer"
    android:includeInGlobalSearch="true"
    android:searchSettingsDescription="@string/settings_description">
</searchable>
```

Configure “Searchable Items” in Setting

Demo:
Configure for QSB

Exercise_4 of “6129_android_search.zip”

Adding Voice Search

Adding Voice Search

- You can add voice search functionality to your search dialog by adding the *android:voiceSearchMode* attribute to your searchable configuration.
- This adds a voice search button in the search dialog that launches a voice prompt.
- When the user has finished speaking, the transcribed search query is sent to your searchable Activity.

Adding Voice Search

```
<!-- The value showVoiceSearchButton is required to enable voice search, while the second value,
launchRecognizer, specifies that the voice search button should launch a recognizer that returns the
transcribed text to the searchable Activity. -->

<?xml version="1.0" encoding="utf-8"?>
<searchable xmlns:android="http://schemas.android.com/apk/res/android"
    android:label="@string/search_label"
    android:hint="@string/search_hint"
    android:voiceSearchMode="showVoiceSearchButton|launchRecognizer" >
</searchable>
```

Thank you!

Check JavaPassion.com Codecamps!
http://www.javapassion.com/codecamps
"Learn with Passion!"