CS 213 Spring 2017
Lecture 22: April 13

Movies App

Search Functionality

Implementing Search Using the Android Search Framework

Android Search Framework

See Develop -> API Guides -> User Interface -> Search

There are two alternatives to providing a search function UI:

- A search dialog at the top of the screen
- OR, a search widget that is embedded in the app bar

The search widget is recommended, so we'll go with that.

Add Search Action to Menu Resource

- Change `res/menu/add_menu.xml` to `res/menu/add_search_menu.xml`:

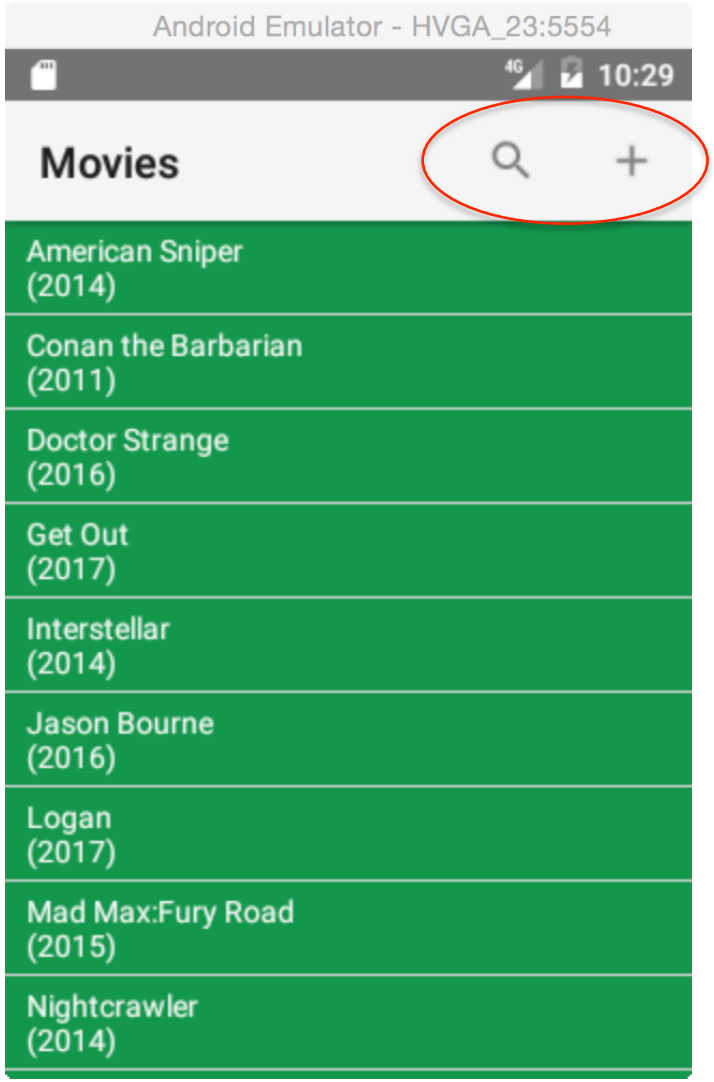
```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android"
      xmlns:appcompat="http://schemas.android.com/apk/res-auto">
    <item android:id="@+id/action_search"
        android:icon="@drawable/ic_action_search_black"
        android:title="@string/search”
        appcompat:showAsAction="always”/>
    <item android:id="@+id/action_add"
        android:icon="@drawable/ic_action_plus_black"
        android:title="@string/add”
        appcompat:showAsAction="always”/>

</menu>
```

Using black now

- Update `onCreateOptionsMenu` to inflate `R.menu.add_search_menu`:

Add Search Function to Menu Resource for App Bar

Movies loaded from file



Implementing the Search Function

1. Implement the search logic

The first thing to do is to add functionality to the app that will enable searching on movies.

The user can search for all movies that start with a prefix string.

Since the `Movies` class maintains the movie list, it would be appropriate to code the search functionality in that class: a method that would return a range of indices in the list of movies whose names start with the query prefix

Implementing the Search Function

1.

```
public int[] search(String query) {
   int lo=0, hi=songs.size()-1;
   int[] extent;
   String movie= query.toLowerCase();
   while (lo <= hi) {
      int mid=(lo+hi)/2;
      if (movies.get(mid).name.toLowerCase().startsWith(movie)) {
         // need to scan left and right of mid for all matches
         return getExtent(mid, movie);
      }
      // mid does not start with the given name, go left or right
      int c = query.compareToIgnoreCase(movies.get(mid).name);
      if (c < 0) {
         hi = mid-1;
      } else {
         lo = mid+1;
      }
   }

   return null;
}
```

Implementing the Search Function

1.

```
private int[] getExtent(int mid, String movie) {
   int[] extent = new int[2];
   extent[0] = mid;
   // scan left
   while (extent[0] > 0) {
      if (movies.get(extent[0]-1).name.toLowerCase().startsWith(
                                                          movie)) {
         extent[0]--;
      } else { break; }
   }
   // scan right
   extent[1] = mid;
   while (extent[1] < movies.size()-1) {
      if (movies.get(extent[1]+1).name.toLowerCase().startsWith(
            movie)) {
         extent[1]++;
      } else { break; }
   }
   return extent;
}
```

Implementing the Search Function

2. Create a searchable configuration

To plug the search function into the Android search framework, we have to make what's called a searchable configuration, which is basically an xml file that should be placed in a directory called `xml` under `res`. (This is not a standard directory that's created with the project) Here's a sample file, called `searchable.xml`:

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

In `strings.xml`:

```
<string name="search_hint">Search Movie</string>
```

Implementing the Search Function

3. Creating/declaring a searchable activity

Next, one activity in the app must be declared to be searchable, which will have code to process the search query.
In our app, we will make `Movies` the searchable activity – this is done by defining an intent-filter tag for `Movies` in the manifest:

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

Implementing the Search Function

4. Using a Search Widget as Action View in Action Bar

The search icon in the Action Bar is used to show up the "search widget", with the following modification to the `add_search_menu.xml` file:

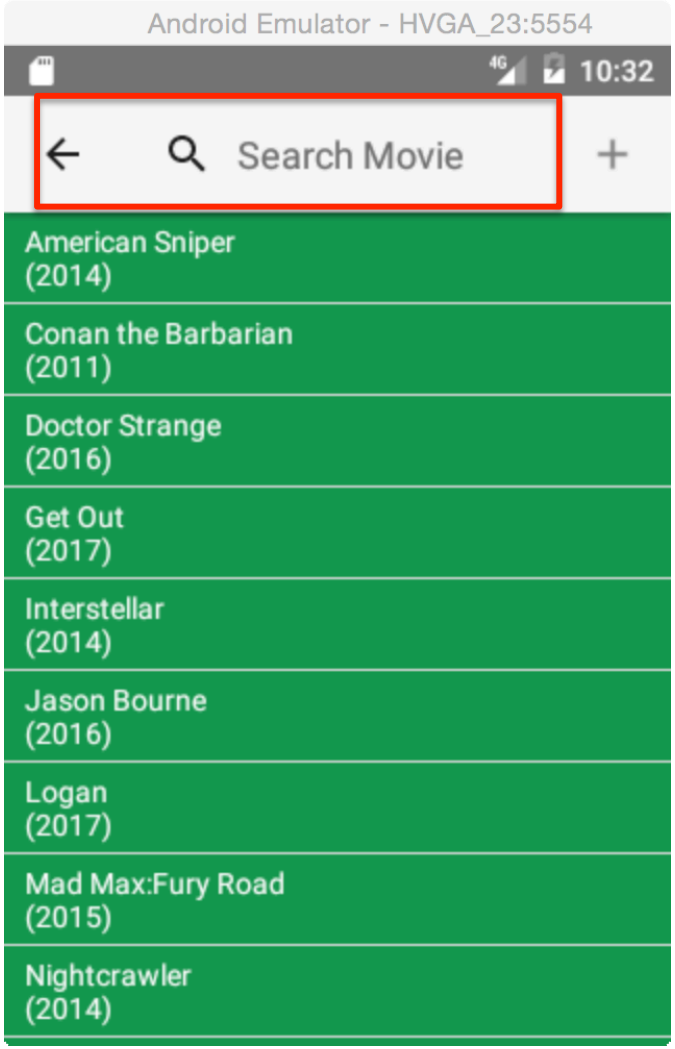
```
<item android:id="@+id/action_search"
    android:icon="@drawable/ic_action_search_black"
    android:title="@string/action_search"
    appcompat:showAsAction="collapseActionView|always"
    appcompat:actionViewClass="android.support.v7.widget.SearchView" />
```

See
Develop->Training->Best Practices for User Interface->Adding the App Bar->
Action View and Action Providers

http://developer.android.com/training/appbar/action-views.html

Search Widget as Action View in App Bar

The widget is an instance of `SearchView`



Implementing the Search Function

5. Associating the Searchable Configuration (XML) with the Search View

   This is done by overriding the `onCreateOptionsMenu` method in `Movies`:

```
public boolean onCreateOptionsMenu(Menu menu) {
    getMenuInflater().inflate(R.menu.add_search_menu,menu);

    // Get the SearchView and set the searchable configuration
    SearchManager searchManager =
       (SearchManager) getSystemService(Context.SEARCH_SERVICE);
    SearchView searchView =
       (SearchView) menu.findItem(R.id.action_search).getActionView();
    // Assumes current activity is the searchable activity
    searchView.setSearchableInfo(searchManager
              .getSearchableInfo(getComponentName()));
    // Do not iconify the widget; expand it by default
    searchView.setIconifiedByDefault(false);

    return super.onCreateOptionsMenu(menu);
}
```

Implementing the Search Function

6. Running `Movies` in `singleTop` mode

When search is activated, a new instance of `Movies` would be created and launched (since is declared in the manifest as the activity to call when a search is done.

But it's a waste to have a new instance of an activity for every search request. Setting the activity to run in `singleTop` mode makes sure the same instance of `Movies` that is on the top of the activity stack is used for the `search` as well:

```
<activity android:name=".Movies"
          android:launchMode="singleTop">
```

Implementing the Search Function

7. Coding `Movies` to work with regular and search intents

Since `Movies` will also act as the target of a search request, there needs to be a way to distinguish between two intents.

First, override the `onNewIntent` method:

```
protected void onNewIntent(Intent intent) {
    setIntent(intent);
    handleIntent(intent);
}
```

Then, implement the `handleIntent` method:

```
private void handleIntent(Intent intent) {
    if (Intent.ACTION_SEARCH.equals(intent.getAction())) {
      String query =
           intent.getStringExtra(SearchManager.QUERY);
      showSearchResults(query);
    } else {
      showMovieList();
    }
}
```

Implementing the Search Function

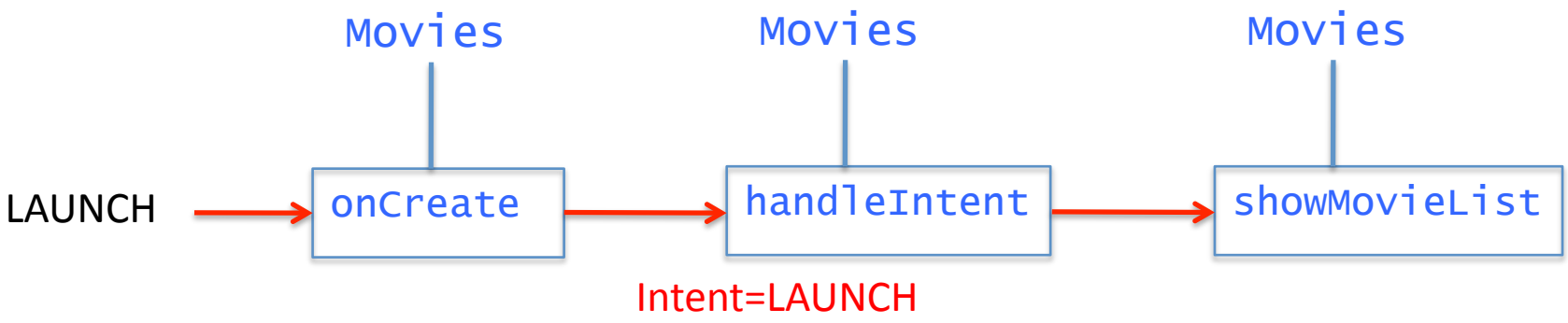
8. Refactoring the original intent (show song list) code

From `onCreate`, move the block of code that sets list adapter and list item selection listener, into the `showMovieList` method:

```
private void showMovieList() {
    listView.setAdapter(
       new ArrayAdapter<Movie>(this, R.layout.movie, movies));

    listView.setOnItemClickListener(
         new AdapterView.OnItemClickListener() {
             @Override
             public void onItemClick(AdapterView<?> parent,
                        View view, int position, long id) {
                        showMovie(position);
             }
         }
     );
}
```

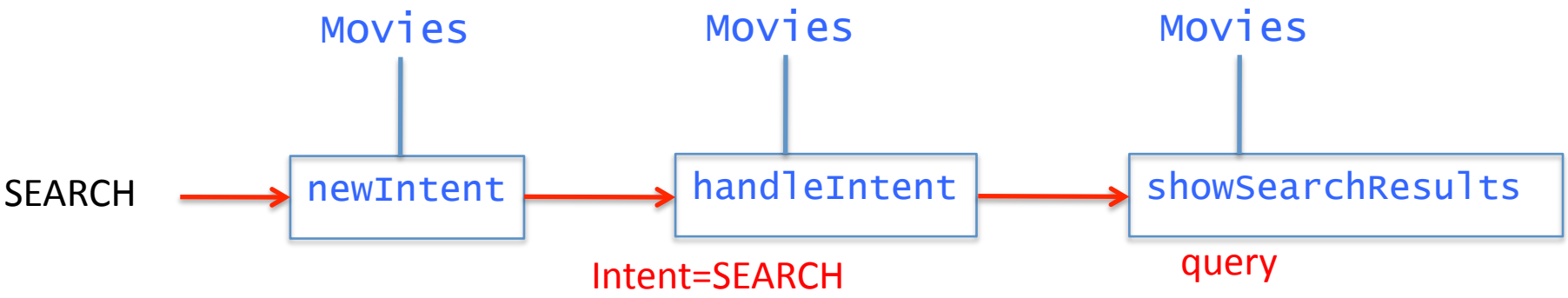
Call `handleIntent(getIntent())` in place of the moved code block

Control Flow: App is Launched



Control Flow: Search is made

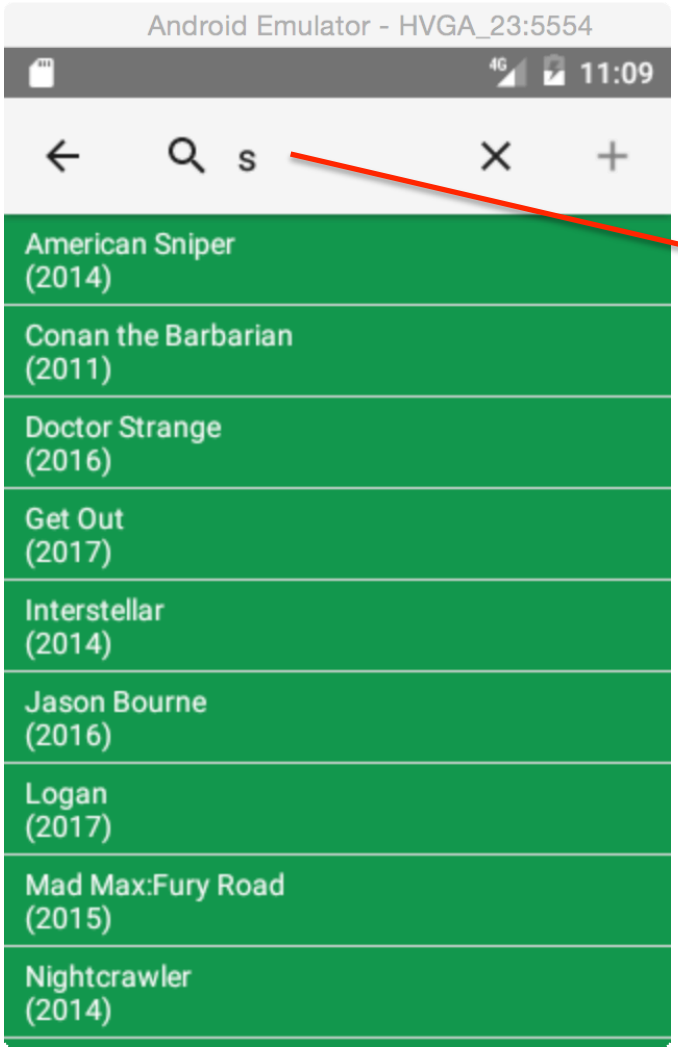
On the same instance of `Movies`, a new intent (SEARCH) is issued, which invokes a call to the `newIntent` method



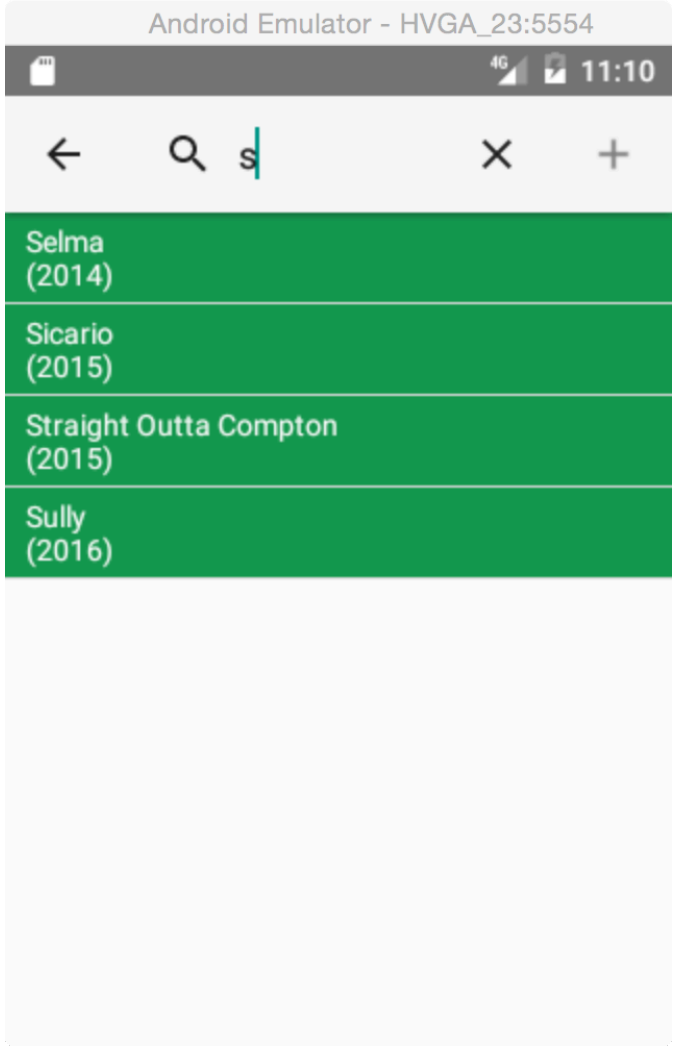
The Android framework ships the search query with the intent, which can be retrieved:

```
String query =
            intent.getStringExtra(SearchManager.QUERY);
```

Making a search



Search logic uses search query as prefix



Implementing the Search Function

9. Implementing `showSearchResults`

```
private void showSearchResults(String query) {
    int[] extent = myList.search(query);
    if (extent == null || extent.length == 0) {  // no matches
        String msg = getString(R.string.search_empty, new Object[] {query});
        Toast.makeText(this,msg,Toast.LENGTH_SHORT).show();
        return;
    }

    makeResultList(extent);

    searchListStartPos = extent[0];
    listView.setOnItemClickListener(
            new AdapterView.OnItemClickListener() {
                public void onItemClick(AdapterView<?> parent,
                                        View view, int position,
                                        long id) {
                    int lastPickedIndex =
                            Movies.this.searchListStartPos+position;
                    showMovie(lastPickedIndex);
                }
            });
}
```

No match found for \"%s\"

Define instance field

listener for search results list

Implementing the Search Function

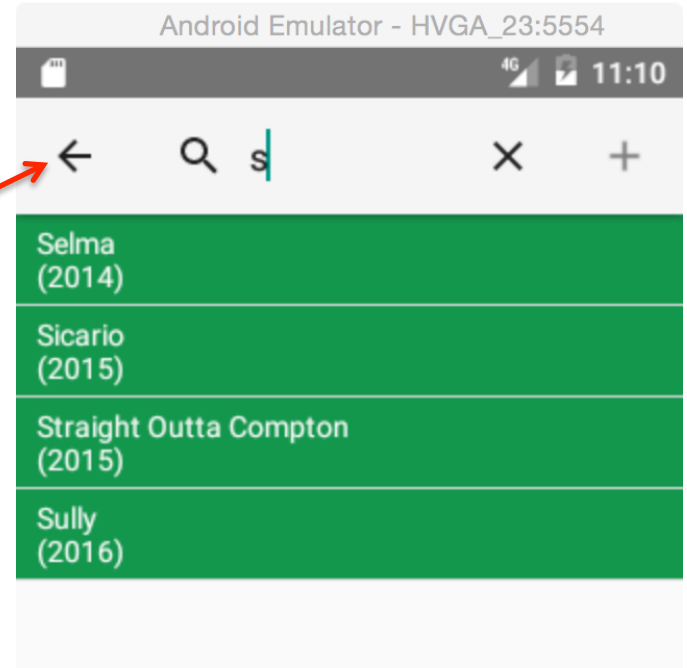
9. Implementing `showSearchResults` - `makeResultList`

```
private void makeResultList(int[] extent) {
    int count = extent[1] - extent[0] + 1;
    Movie[] matches = new Movie[count];

    for (int i=0; i < matches.length; i++) {
        matches[i] = movies.get(extent[0]+i);
    }
    listView.setAdapter(
            new ArrayAdapter<Movie>(this, R.layout.movie, matches));
}
```

Implementing the Search Function

10. Making search widget's up nav go back to main list

Without explicit handling, the up nav will recyle back into `Movies` with filtered list of search results, not the original list



Need to watch for collapse of the action view (associated with the search menu item in action bar), and reset the list to the original

Implementing the Search Function

10. Take action when action view is collapsed

```
public boolean onCreateOptionsMenu(Menu menu) {
    getMenuInflater().inflate(R.menu.add_search_menu,menu);

    // Get the SearchView and set the searchable configuration
    SearchManager searchManager =
       (SearchManager) getSystemService(Context.SEARCH_SERVICE);
    ...

    return super.onCreateOptionsMenu(menu);
}
```

We're going to get a hold of the search action menu item (corresponding to the action icon), and handle the collapse event (which happens when the up nav arrow is tapped on the search widget) to reset the list view to the original master list

Implementing the Search Function

10. Handle action collapse event callback

```
public boolean onCreateOptionsMenu(Menu menu) {
    getMenuInflater().inflate(R.menu.add_search_menu,menu);

    // Get the SearchView and set the searchable configuration
    ...

    MenuItem searchItem = menu.findItem(R.id.action_search);
    MenuItemCompat.setOnActionExpandListener(searchItem,
                new MenuItemCompat.OnActionExpandListener() {
        @Override
        public boolean onMenuItemActionCollapse(MenuItem item) {
            // Do something when collapsed
            listView.setAdapter(new ArrayAdapter<Movie>(Movies.this, movie, movies));
            searchListStartPos=0;
            return true;  // Return true to collapse action view
        }

        @Override
        public boolean onMenuItemActionExpand(MenuItem item) {
            // Do something when expanded
            return true;  // Return true to expand action view
        }
    });

    return super.onCreateOptionsMenu(menu);
}
```