Click to Open Details

This lesson will teach you how to add click listener to the list item.

WE'LL COVER THE FOLLOWING

Details screen changes

The very first thing we need to do is to update the details screen. Instead of loading blog details again, we can just pass the data from the `MainActivity` to the `BlogDetailsActivity` via the intent bundle.

Let's remove the `loadData` method and instead, call the `showData` method. To get `Blog` from the parameter we need to:

- (1) acquire reference to the Intent object via `getIntent` method *(object which was used to launch this Activity)*
- (2) retrieve Bundle object from the *Intent* via `getExtras` method *(object which contains some data carried by Intent)*
- (3) retrieve the `Blog` object from the *Bundle* via `getParcelable` method passing the *String* key as a parameter

```
public class BlogDetailsActivity extends AppCompatActivity {

    private static final String EXTRAS_BLOG = "EXTRAS_BLOG";

    ...

    @Override
    protected void onCreate(@Nullable Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activtiy_blog_details);
```

```
        ...

        showData(getIntent() // 1
                    .getExtras() // 2
                    .getParcelable(EXTRAS_BLOG)); // 3
    }

    private void showData(Blog blog) {
        progressBar.setVisibility(View.GONE);
        textTitle.setText(blog.getTitle());
        textDate.setText(blog.getDate());
        textAuthor.setText(blog.getAuthor().getName());
        textRating.setText(String.valueOf(blog.getRating()));
        textViews.setText(String.format("(%d views)", blog.getViews()));
        textDescription.setText(Html.fromHtml(blog.getDescription()));
        ratingBar.setRating(blog.getRating());
        ratingBar.setVisibility(View.VISIBLE);

        Glide.with(this)
                .load(blog.getImageURL())
                .transition(DrawableTransitionOptions.withCrossFade())
                .into(imageMain);

        Glide.with(this)
                .load(blog.getAuthor().getAvatarURL())
                .transform(new CircleCrop())
                .transition(DrawableTransitionOptions.withCrossFade())
                .into(imageAvatar);
    }
}
```

When we launch `BlogDetailsActivity`, we expect the *Intent* to carry a `Blog` object, but it's pretty easy to forget to do this.

To make things a bit safer, let's add a static method with the `Blog` parameter to start the `BlogDetailsActivity`. After creating the `Intent` object we can use the `putExtra` method to put the data object into this `Intent` via *String* key.

```
public class BlogDetailsActivity extends AppCompatActivity {

    private static final String EXTRAS_BLOG = "EXTRAS_BLOG";
    ...
    public static void startBlogDetailsActivity(Activity activity, Blog blog) {
        Intent intent = new Intent(activity, BlogDetailsActivity.class);
        intent.putExtra(EXTRAS_BLOG, blog);
        activity.startActivity(intent);
    }
}
```

Parcelable

Previously, we used `getParcelable` and `putExtra` to put and get our `Blog` data object, but in *Android* we can't just pass objects from one *Activity* to another, because *Android* doesn't know how to do this.

Fortunately, there is a way to teach *Android* how to serialize an object into *Bundle* and deserialize it back. This is done via implementing the Parcelable interface. *Android Studio* can generate the implementation for you, so usually you don't need to manually write this code.

Let's briefly go through how *Android* system uses *Parcelable* object:

- (1) The `writeToParcel` method to serialize an object; inside this method we have `Parcel` object which we can use to store data via various `write*` methods.
- (2) The constructor with `Parcel` parameter is used to deserialize an object. We can use the various `read*` methods to do so, make sure that the order of `write` and `read` methods is the same.
- (3) The objects that implement the `Parcelable` interface must also have a type of non-null static field, called `CREATOR`, that implements the `Parcelable.Creator` interface.

```
public class Author implements Parcelable {

    private String name;
    private String avatar;

    protected Author(Parcel in) {  // 1
        name = in.readString();
        avatar = in.readString();
    }

    @Override
    public void writeToParcel(Parcel dest, int flags) {  // 2
        dest.writeString(name);
        dest.writeString(avatar);
    }

    @Override
    public int describeContents() {
        return 0;
    }

    public static final Creator<Author> CREATOR = new Creator<Author>() {  // 3
        @Override
        public Author createFromParcel(Parcel in) {
            return new Author(in);
        }

        @Override
        public Author[] newArray(int size) {
            return new Author[size];
        }
```

```
    };


    ...
}
```

Handling item click

It's time to add the item click listener to the recycler view adapter.

Let's create an `OnItemClickListener` interface (1) which the client has to pass to the `MainAdapter` constructor (2). We can use the reference to this listener and pass it to the `MainViewHolder` (3).

```
public class MainAdapter extends ListAdapter<Blog, MainAdapter.MainViewHolder> {

    public interface OnItemClickListener { // 1
        void onItemClicked(Blog blog);
    }

    private OnItemClickListener clickListener;

    public MainAdapter(OnItemClickListener clickListener) { // 2
        super(DIFF_CALLBACK);
        this.clickListener = clickListener;
    }

    @NonNull
    @Override
    public MainViewHolder onCreateViewHolder(@NonNull ViewGroup parent,
                                             int viewType) {
        LayoutInflater inflater = LayoutInflater.from(parent.getContext());
        View view = inflater.inflate(R.layout.item_main, parent, false);
        return new MainViewHolder(view, clickListener); // 3
    }

    ...
}
```

Now, in the `MainViewHolder`, we can set click listener to the root item view and trigger our custom `OnItemClickListener`.

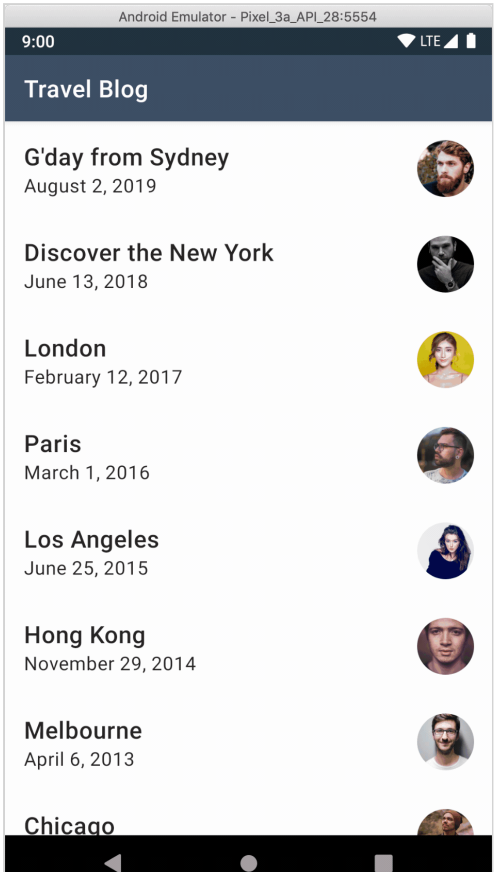
```
static class MainViewHolder extends RecyclerView.ViewHolder {

    private TextView textTitle;
    private TextView textDate;
    private ImageView imageAvatar;
    private Blog blog;

    MainViewHolder(@NonNull View itemView, OnItemClickListener listener) {
        super(itemView);
        itemView.setOnClickListener(v -> listener.onItemClicked(blog));
        textTitle = itemView.findViewById(R.id.textTitle);
```

```
        textDate = itemView.findViewById(R.id.textDate);
        imageAvatar = itemView.findViewById(R.id.imageAvatar);
    }

    ...
}
```

Finally, we can modify adapter creation in the `MainActivity` and react to `OnItemClickListener` by launching `BlogDetailsActivity`.

```
public class MainActivity extends AppCompatActivity {

    private MainAdapter adapter;
    ...

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        adapter = new MainAdapter(blog ->
                BlogDetailsActivity.startBlogDetailsActivity(this, blog));

        ...
    }
    ...
}
```

When we launch the application and click on the blog item, it should open the corresponding details screen.



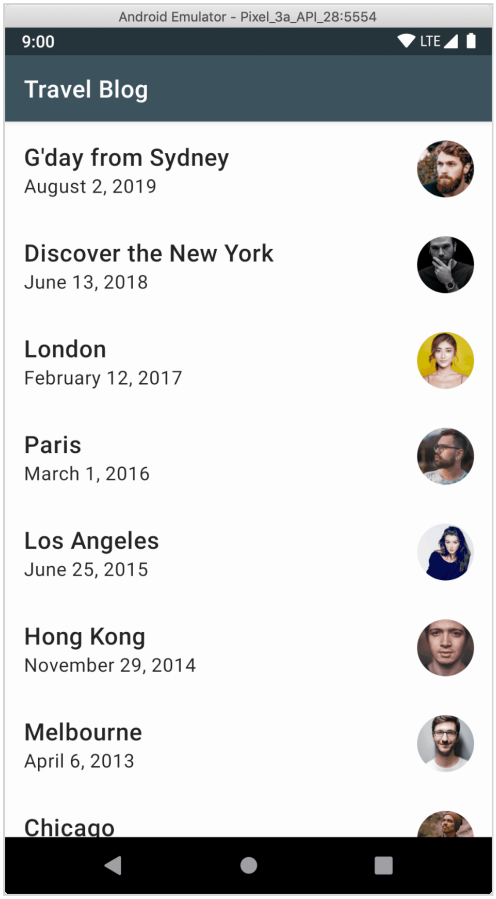
The only problem is that when we click on the list item it doesn't highlight which item was clicked. We can fix this by setting a background to the `item_main.xml` root layout.

Let's use `?android:attr/selectableItemBackground` as an attribute value, which is a reference to the Android background drawable for bordered standalone items that need focus/pressed states.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
        xmlns:tools="http://schemas.android.com/tools"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:background="?android:attr/selectableItemBackground"
        android:orientation="horizontal"
        android:padding="16dp">

    <LinearLayout
            android:layout_width="0dp"
            android:layout_height="wrap_content"
            android:layout_weight="1.0"
            android:orientation="vertical">

        ...

</LinearLayout>
```

item_main.xml

Now, when we click on the blog item it should highlight the clicked item; this provides a better user experience.



Hit the *run* button to try it yourself.

```
package com.travelblog.adapter;

import android.view.*;
import android.widget.*;

import androidx.annotation.*;
import androidx.recyclerview.widget.ListAdapter;
import androidx.recyclerview.widget.*;

import com.bumptech.glide.*;
import com.bumptech.glide.load.resource.bitmap.*;
import com.bumptech.glide.load.resource.drawable.*;
import com.travelblog.R;
import com.travelblog.http.*;

public class MainAdapter extends ListAdapter<Blog, MainAdapter.MainViewHolder> {

    public interface OnItemClickListener {
        void onItemClicked(Blog blog);
    }

    private OnItemClickListener clickListener;

    public MainAdapter(OnItemClickListener clickListener) {
        super(DIFF_CALLBACK);
        this.clickListener = clickListener;
    }

    @NonNull
```

```
    @Override
    public MainViewHolder onCreateViewHolder(@NonNull ViewGroup parent, int viewType) {
        LayoutInflater inflater = LayoutInflater.from(parent.getContext());
        View view = inflater.inflate(R.layout.item_main, parent, false);
        return new MainViewHolder(view, clickListener);
    }

    @Override
    public void onBindViewHolder(MainViewHolder holder, int position) {
        holder.bindTo(getItem(position));
    }


    static class MainViewHolder extends RecyclerView.ViewHolder {

        private TextView textTitle;
        private TextView textDate;
        private ImageView imageAvatar;
        private Blog blog;

        MainViewHolder(@NonNull View itemView, OnItemClickListener listener) {
            super(itemView);
            itemView.setOnClickListener(v -> listener.onItemClicked(blog));
            textTitle = itemView.findViewById(R.id.textTitle);
            textDate = itemView.findViewById(R.id.textDate);
            imageAvatar = itemView.findViewById(R.id.imageAvatar);
        }

        void bindTo(Blog blog) {
            this.blog = blog;
            textTitle.setText(blog.getTitle());
            textDate.setText(blog.getDate());

            Glide.with(itemView)
                    .load(blog.getAuthor().getAvatarURL())
                    .transform(new CircleCrop())
                    .transition(DrawableTransitionOptions.withCrossFade())
                    .into(imageAvatar);
        }

    }

    private static final DiffUtil.ItemCallback<Blog> DIFF_CALLBACK =
            new DiffUtil.ItemCallback<Blog>() {
                @Override
                public boolean areItemsTheSame(@NonNull Blog oldData,
                                               @NonNull Blog newData) {
                    return oldData.getId().equals(newData.getId());
                }

                @Override
                public boolean areContentsTheSame(@NonNull Blog oldData,
                                                  @NonNull Blog newData) {
                    return oldData.equals(newData);
                }
            };
}
```

In the next chapter, we will learn how to implement search and sort functionality for the list screen.

functionality for the list screen.