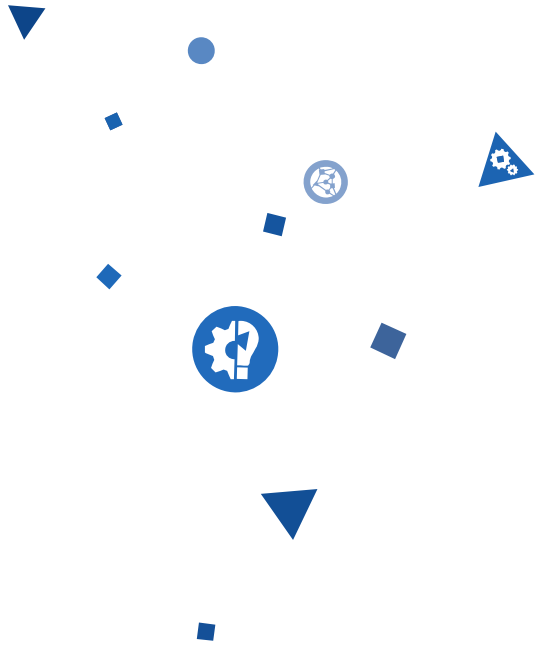


ANDROID BEST TOOLS

Tools which will save your time & increase productivity

TRAINING ROADMAP: STRUCTURE

- GSON
- ButterKnife
- Dagger 2
- Glide
- Retrofit
- Stetho
- AutoValue
- RxJava

GSON: JSON

```
Gson gson = new Gson();
gson.toJson(123);
gson.toJson("hello");
gson.toJson(Long.valueOf(10));

Integer integer = gson.fromJson("1", int.class);
String string = gson.fromJson("\"world\"", String.class);
Boolean bool = gson.fromJson("true", Boolean.class);

String string = gson.toJson(new int[] { 10, 100 }); // [10,100]
int[] array = gson.fromJson("[10,100]", int[].class);
```

GSON: JSON

```
Entity entity = new Entity(100, "name");
entity.random = 1234;

String json = gson.toJson(entity);
Entity read = gson.fromJson(json, Entity.class);
System.out.println(read.random);

Map<String, Integer> map = new LinkedHashMap<>();
map.put("USD", 123);
map.put("EUR", 321);

String json = gson.toJson(map);

Type type = new TypeToken<Map<String,
Integer>>(){}.getType();
Map<String, Integer> read = gson.fromJson(json, type);
```

```
public static class Entity {
   volatile int id;
   String name;
   transient long random;

   public Entity(int id, String name) {
      this.id = id;
      this.name = name;
   }
}
```

GSON: JSON

```
public class CustomConverter implements JsonSerializer<Custom>, JsonDeserializer<Custom>  {
   public JsonElement serialize(Custom src, Type type,
                        JsonSerializationContext context) {
      JsonObject object = new JsonObject();
      object.addProperty("date", src.date.getTime());
      object.addProperty("integer", src.integer.toString());
      return object;
   }

   public Custom deserialize(JsonElement json, Type type,
                        JsonDeserializationContext context) throws JsonParseException {
      JsonObject object = json.getAsJsonObject();
      Date date = new Date(object.get("date").getAsLong());
      BigInteger integer = new BigInteger(object.get("integer").getAsString());
      return new Custom(date, integer);
   }
}
```

GSON: JSON

```
public class CustomConverter implements JsonSerializer<Custom>, JsonDeserializer<Custom>  {
   public JsonElement serialize(Custom src, Type type,
                        JsonSerializationContext context) {
      JsonObject object = new JsonObject();
      object.addProperty("date", src.date.getTime());
      object.addProperty("integer", src.integer.toString());
      return object;
   }

   public Custom deserialize(JsonElement json, Type type,
                        JsonDeserializationContext context) throws JsonParseException {
      JsonObject object = json.getAsJsonObject();
      Date date = new Date(object.get("date").getAsLong());
      BigInteger integer = new BigInteger(object.get("integer").getAsString());
      return new Custom(date, integer);
   }
}
```

```
GsonBuilder builder = new GsonBuilder();
builder.registerTypeAdapter(Custom.class, new CustomConverter());
Gson gson = builder.create();
```

BUTTERKNIFE

```
class ExampleActivity extends Activity {
  @BindView(R.id.title) TextView title;
  @BindView(R.id.subtitle) TextView subtitle;
  @BindView(R.id.footer) TextView footer;

  @Override public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.simple_activity);
    ButterKnife.bind(this);
    // TODO Use fields...
  }
}
```

```
public void bind(ExampleActivity activity) {
  activity.subtitle = (android.widget.TextView) activity.findViewById(2130968578);
  activity.footer = (android.widget.TextView) activity.findViewById(2130968579);
  activity.title = (android.widget.TextView) activity.findViewById(2130968577);
}
```

BUTTERKNIFE

```
class ExampleActivity extends Activity {
    @BindString(R.string.title) String title;
    @BindDrawable(R.drawable.graphic) Drawable graphic;
    @BindColor(R.color.red) int red;
    @BindDimen(R.dimen.spacer) Float spacer;
    // ...
}
```

BUTTERKNIFE

```
public class FancyFragment extends Fragment {
   @BindView(R.id.button1) Button button1;
   @BindView(R.id.button2) Button button2;

   @Override public View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle savedInstanceState) {
      View view = inflater.inflate(R.layout.fancy_fragment, container, false);
      ButterKnife.bind(this, view);
      // TODO Use fields...
      return view;
   }
}
```

BUTTERKNIFE: LISTVIEW

```
@BindViews({ R.id.first_name, R.id.middle_name, R.id.last_name })
List<EditText> nameViews;

ButterKnife.apply(nameViews, DISABLE);
ButterKnife.apply(nameViews, ENABLED, false);
```

Action and Setter interfaces allow specifying simple behavior.

```
static final ButterKnife.Action<View> DISABLE = new ButterKnife.Action<View>() {
  @Override public void apply(View view, int index) {
    view.setEnabled(false);
  }
};
static final ButterKnife.Setter<View, Boolean> ENABLED = new ButterKnife.Setter<View, Boolean>() {
  @Override public void set(View view, Boolean value, int index) {
    view.setEnabled(value);
  }
};
```

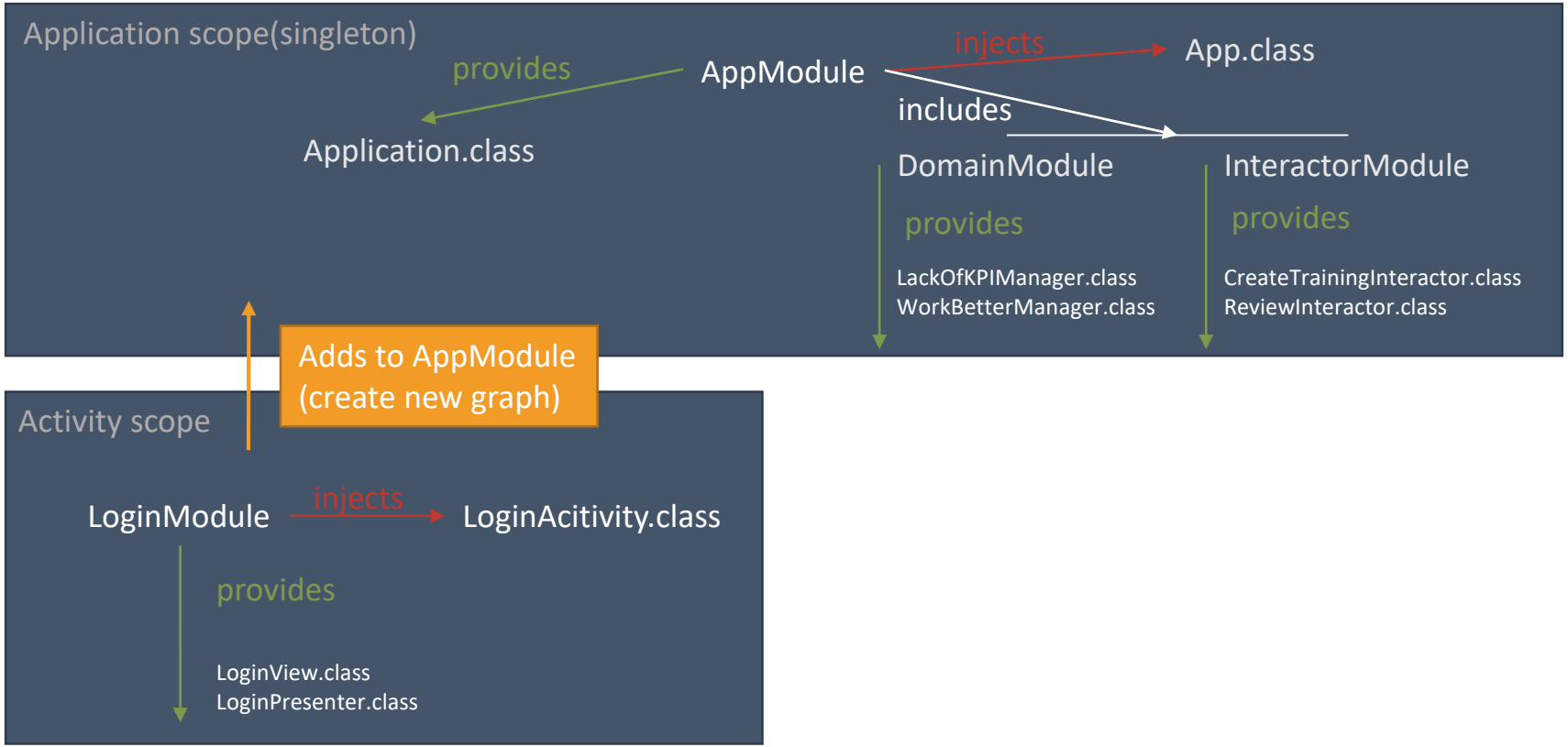
BUTTERKNIFE: BINDING

```
@OnClick(R.id.submit)
public void sayHi(Button button) {
   button.setText("Hello!");
}

@OnClick({ R.id.door1, R.id.door2, R.id.door3 })
public void pickDoor(DoorView door) {
   if (door.hasPrizeBehind()) {
      Toast.makeText(this, "You win!", LENGTH_SHORT).show();
   } else {
      Toast.makeText(this, "Try again", LENGTH_SHORT).show();
   }
}
```

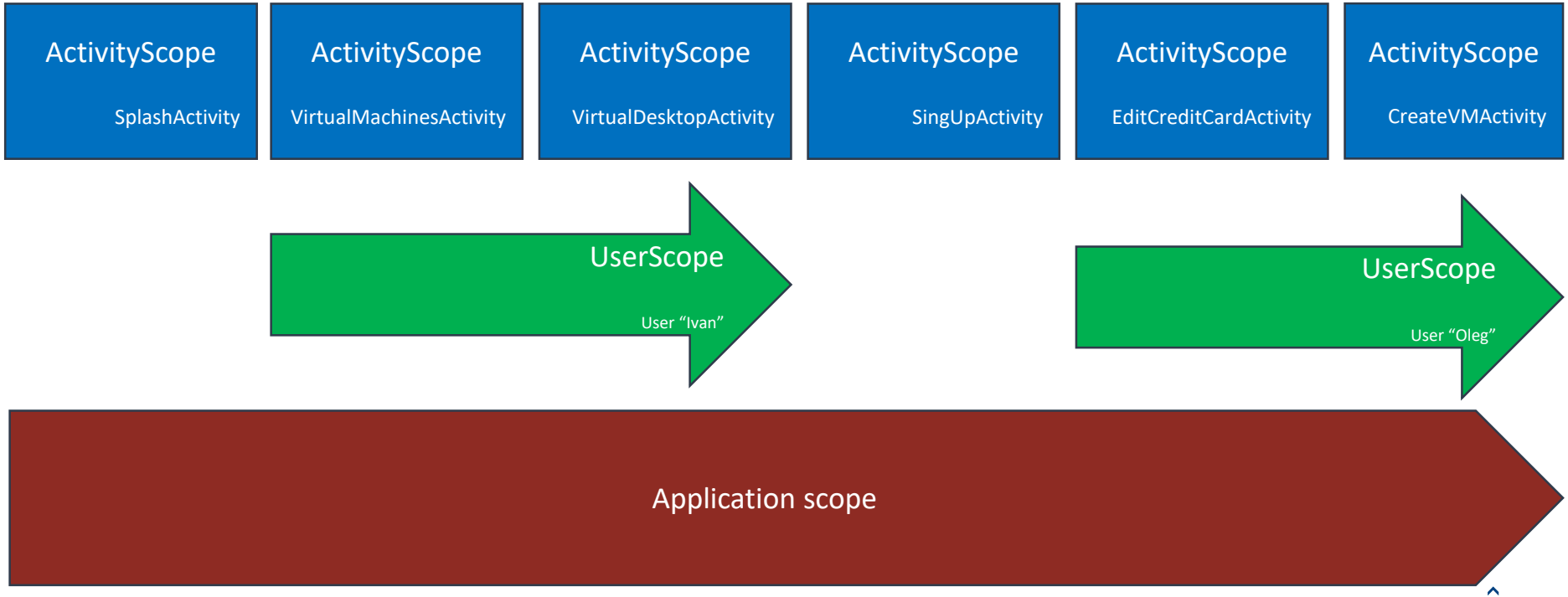
BUTTERKNIFE: BINDING

```
public class FancyFragment extends Fragment {
  @BindView(R.id.button1) Button button1;
  @BindView(R.id.button2) Button button2;
  private Unbinder unbinder;

  @Override public View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle savedInstanceState) {
    View view = inflater.inflate(R.layout.fancy_fragment, container, false);
    unbinder = ButterKnife.bind(this, view);
    // TODO Use fields...
    return view;
  }

  @Override public void onDestroyView() {
    super.onDestroyView();
    unbinder.unbind();
  }
}
```

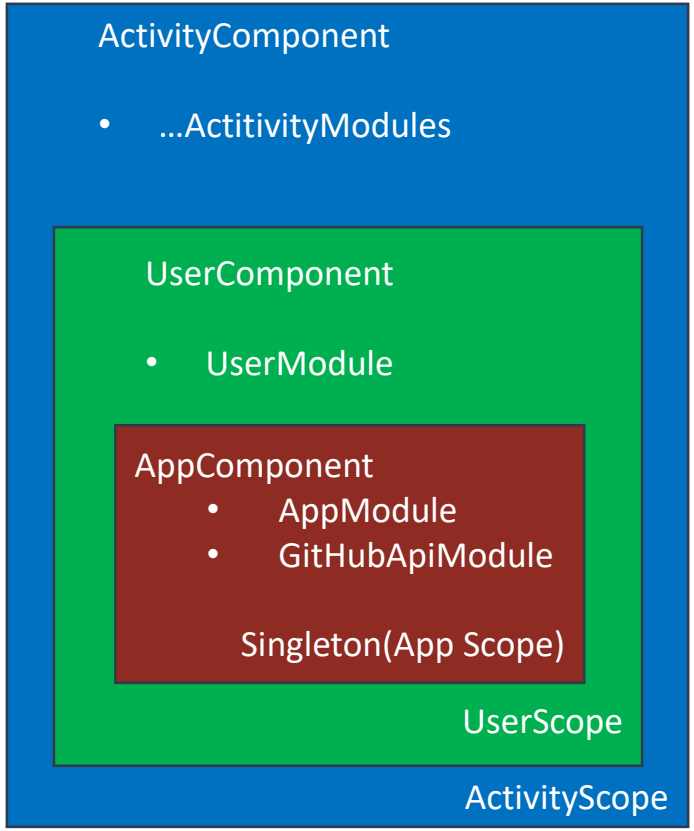
DAGGER


Application scope(singleton)
provides
AppModule
injects
App.class
includes
Application.class
DomainModule
provides
LackOfKPIManager.class
WorkBetterManager.class
InteractorModule
provides
CreateTrainingInteractor.class
ReviewInteractor.class
Adds to AppModule
(create new graph)
Activity scope
LoginModule
injects
LoginAcitivity.class
provides
LoginView.class
LoginPresenter.class

DAGGER: SCOPES LIFECYCLE



DAGGER: SCOPES NESTING



GLIDE: IMAGE LIBRARY

```
@Override public void onCreate(Bundle savedInstanceState) {
   ImageView imageView = (ImageView) findViewById(R.id.my_image_view);
   GlideApp.with(this).load("http://goo.gl/gEgYUd").into(imageView);
}

@Override public View getView(int position, View recycled, ViewGroup container) {
   final ImageView myImageView;
   if (recycled == null) {
      myImageView = (ImageView) inflater.inflate(R.layout.my_image_view, container, false);
   } else {
      myImageView = (ImageView) recycled;
   }
   String url = myUrls.get(position);
   GlideApp
         .with(myFragment)
         .load(url)
         .centerCrop()
         .placeholder(R.drawable.loading_spinner)
         .into(myImageView);

   return myImageView;
}
```

RETROFIT

```
public interface GitHubService {
   @GET("users/{user}/repos")
   Call<List<Repo>> listRepos(@Path("user") String user);
}
```

RETROFIT

```
public interface GitHubService {
   @GET("users/{user}/repos")
   Call<List<Repo>> listRepos(@Path("user") String user);
}


Retrofit retrofit = new Retrofit.Builder()
      .baseUrl("https://api.github.com/")
      .build();

GitHubService service = retrofit.create(GitHubService.class);
```

RETROFIT

```
public interface GitHubService {
   @GET("users/{user}/repos")
   Call<List<Repo>> listRepos(@Path("user") String user);
}


Retrofit retrofit = new Retrofit.Builder()
      .baseUrl("https://api.github.com/")
      .build();

GitHubService service = retrofit.create(GitHubService.class);


Call<List<Repo>> repos = service.listRepos("octocat");
```

STETHO: DEBUG BRIDGE

- Stetho is a sophisticated debug bridge for Android applications.
- Developers have access to the Chrome Developer Tools - part of the Chrome desktop browser.
- Also can choose to enable the optional dumpapp tool - command-line interface to application internals.

STETHO

- Network Inspection is possible with the full spectrum of Chrome Developer Tools features, including image preview, JSON response helpers, and even exporting traces to the HAR format.
- Database Inspection - SQLite databases can be visualized and interactively explored with full read/write capabilities.
- View Hierarchy - View hierarchy support for ICS (API 15) and up! Lots of goodies such as instances virtually placed in the hierarchy, view highlighting, and the ability to tap on a view to jump to its position in the hierarchy.

STETHO

- Dumpapp - Dumpapp extends beyond the DevTools UI features shown above to provide a much more extensible, command-line interface to application components. A default set of plugins is provided, but the real power of dumpapp is the ability to easily create your own
- Javascript Console - Javascript Console allows for execution of javascript code that can interact with the application or even the Android SDK.

STETHO: DEBUG BRIDGE

- Chrome DevTools
- Network Inspection
- View Hierarchy
- Database Inspection
- Javascript Console
- dumpapp

STETHO: SETUP

```
public class MyApplication extends Application {
   public void onCreate() {
      super.onCreate();
      Stetho.initializeWithDefaults(this);
   }
}
```

```
OkHttpClient client = new OkHttpClient();
client.networkInterceptors().add(new StethoInterceptor());
```

```
new OkHttpClient.Builder()
      .addNetworkInterceptor(new StethoInterceptor())
      .build();
```

STETHO: CUSTOM PLUGINS

```
Stetho.initialize(Stetho.newInitializerBuilder(context)
        .enableDumpapp(new DumperPluginsProvider() {
            @Override
            public Iterable<DumperPlugin> get() {
                return new Stetho.DefaultDumperPluginsBuilder(context)
                    .provide(new MyDumperPlugin())
                    .finish();
            }
        })
        .enableWebKitInspector(Stetho.defaultInspectorModulesProvider(context))
        .build());
```

AUTOVALUE

```
@AutoValue
abstract class Animal {
  static Animal create(String name, int numberOfLegs) {
    // See "How do I...?" below for nested classes.
    return new AutoValue_Animal(name, numberOfLegs);
  }

  abstract String name();
  abstract int numberOfLegs();
}
```

AUTOVALUE

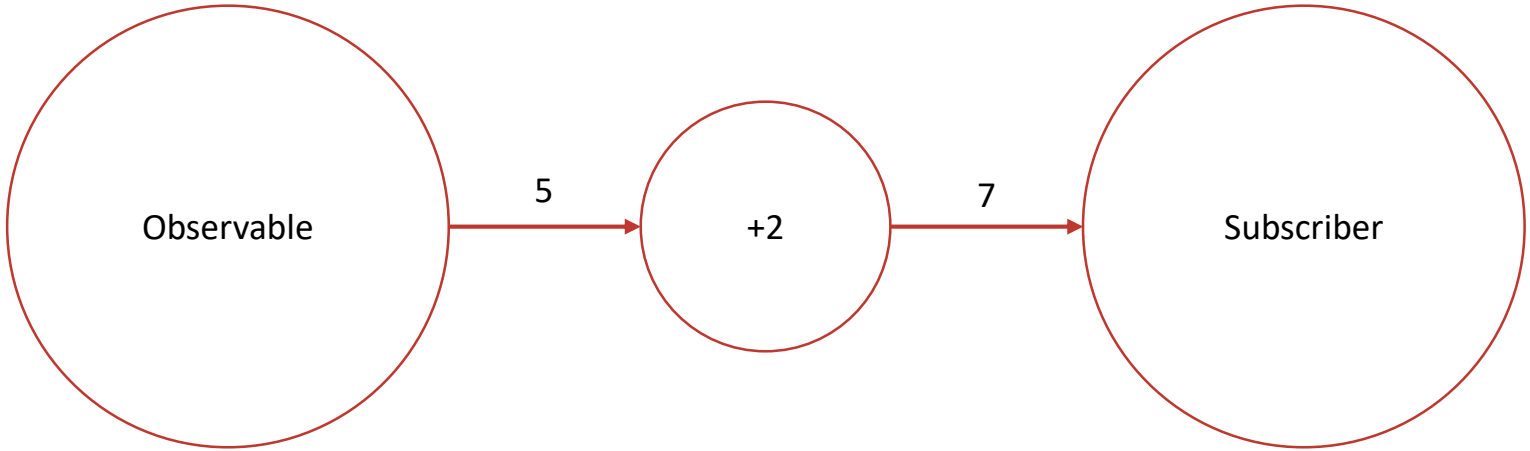
```
public void testAnimal() {
    Animal dog = Animal.create("dog", 4);
    assertEquals("dog", dog.name());
    assertEquals(4, dog.numberOfLegs());

    // You probably don't need to write assertions like these; just illustrating.
    assertTrue(Animal.create("dog", 4).equals(dog));
    assertFalse(Animal.create("cat", 4).equals(dog));
    assertFalse(Animal.create("dog", 2).equals(dog));

    assertEquals("Animal{name=dog, numberOfLegs=4}", dog.toString());
}
```

RXJAVA BASICS: BE READY TO

- Switch from **imperative** to **functional** from **sync** to **async** from **pull** to **push**
- Composable data flow
- Push concept
- Combination of:
  - observer pattern
  - iterator pattern
  - functional programming

RXJAVA: REACTIVE IS..

- Of, relating to, or marked by reaction or reactance.
- Readily responsive to a stimulus.



AND LET’S NOT FORGET ABOUT

- Android Jetpack