Firebase Realtime Database

Landon Cox

April 6, 2017

Databases so far

- SQLite (store quiz progress locally)
  - User starts app
  - Check database to see where user was
- Say you want info about your friends' quizzes
  - Need to store info in a shared database
  - Can't be on your device
  - Need data to be stored on server
  - Want to be notified when data changes

Relational databases

- Data is organized into tables
    - Tables have named, typed columns
    - Data is stored as rows in a table
    - Can place constraints on columns (e.g., uniqueness)
    - Structure + constraints define the schema
- Read/write the data base with SQL
    - Structured Query Language (SQL)
    - SQL is declarative
    - It describes what result you want, not how to compute it
- Example databases: mysql, postgresql, sqlite

SQLite

- SQLite is the primary database for Android apps

- Classes for managing your app's SQLite database
    - Contract class w/ inner BaseColumns class
    - DbHelper class that extends SQLiteOpenHelper
    - Cursor for iterating through answers to queries

https://developer.android.com/training/basics/data-storage/databases.html

Define the contract/schema

- Contract class
    - Place to put all constants related to your database

- BaseColumns inner class
    - Table names
    - Column names

- One BaseColumns class for each table in the db

_id	quiz_title	num_correct	num_wrong	last_question	finished_quiz	timestamp
0	Duke Basketball	0	0	0	0	1488460945

Firebase features

- Authentication
  - Integrate with identity providers or email
  - Google, Twitter, Facebook, others
- Storage
  - Remote storage for the user
  - Can store large files
- Messaging
  - Send/receive notifications
  - Requires app server

https://github.com/firebase/FirebaseUI-Android/

Firebase features

- Authentication
  - Integrate with identity providers or email
  - Google, Twitter, Facebook, others
- Storage
  - Remote storage for the user
  - Can store large files
- Messaging
  - Send/receive notifications
  - Requires app server

https://github.com/firebase/FirebaseUI-Android/

Tables vs JSON trees

- SQL databases are stored as tables
  - Use SQL language to access data

_id	quiz_title	num_correct	num_wrong	last_question	finished_quiz	timestamp
0	Duke Basketball	0	0	0	0	1488460945

- Firebase databases are stored as a tree
  - Access via *keys* (Strings) that map to *values* (Objects)
  - Objects are stored in JSON format
  - We've seen JSON before ...

JSON

- Javascript Object Notation
  - Similar to XML, but more restricted
  - Solved the need to exchange client/server data on web
  - Designed to ease marshalling/unmarshalling

- Example javascript (marshalling)

```
var myObj = { "name":"John", "age":31, "city":"New York" };
var myJSON = JSON.stringify(myObj);
window.location = "demo_json.php?x=" + myJSON;
```

https://www.w3schools.com/js/js_json_intro.asp

JSON

- Javascript Object Notation
  - Similar to XML, but more restricted
  - Solved the need to exchange client/server data on web
  - Designed to ease marshalling/unmarshalling

- Example javascript (marshalling)

Fields: string name + value

```
var myObj = { "name":"John", "age":31, "city":"New York" };
var myJSON = JSON.stringify(myObj);
window.location = "demo_json.php?x=" + myJSON;
```

https://www.w3schools.com/js/js_json_intro.asp

JSON

- Javascript Object Notation
    - Similar to XML, but more restricted
    - Solved the need to exchange client/server data on web
    - Designed to ease marshalling/unmarshalling

- Example javascript (marshalling)

Convert js object to string via stringify

```
var myObj = { "name":"John", "age":31, "city":"New York" };
var myJSON = JSON.stringify(myObj);
window.location = "demo_json.php?x=" + myJSON;
```

https://www.w3schools.com/js/js_json_intro.asp

JSON

- Javascript Object Notation
  - Similar to XML, but more restricted
  - Solved the need to exchange client/server data on web
  - Designed to ease marshalling/unmarshalling

- Example javascript (marshalling)

Once a string, can print

```
var myObj = { "name":"John", "age":31,   ity":"New York" };
var myJSON = JSON.stringify(myObj);
window.location = "demo_json.php?x=" + myJSON;
```

https://www.w3schools.com/js/js_json_intro.asp

JSON

- Javascript Object Notation
  - Similar to XML, but more restricted
  - Solved the need to exchange client/server data on web
  - Designed to ease marshalling/unmarshalling

- Example javascript (marshalling)

Once a string, can also save to database!

```
var myObj = { "name":"John", "age":31,   ity":"New York" };
var myJSON = JSON.stringify(myObj);
window.location = "demo_json.php?x=" + myJSON;
```

https://www.w3schools.com/js/js_json_intro.asp

JSON

- Javascript Object Notation
  - Similar to XML, but more restricted
  - Solved the need to exchange client/server data on web
  - Designed to ease marshalling/unmarshalling

- Example javascript (marshalling)

Once a string, can also save to database!

```
var myObj = { "name":"John", "age":31,    ity":"New York" };
var myJSON = JSON.stringify(myObj);
window.location = "demo_json.php?x=" + myJSON;
```

https://www.w3schools.com/js/js_json_intro.asp

JSON

- Javascript Object Notation
  - Similar to XML, but more restricted
  - Solved the need to exchange client/server data on web
  - Designed to ease marshalling/unmarshalling

- Example javascript (unmarshalling)

```
var myJSON = ’{ "name":"John", "age":31, "city":"New York" }’;
var myObj = JSON.parse(myJSON);
document.getElementById(“demo”).innerHTML = myObj.name;
```

https://www.w3schools.com/js/js_json_intro.asp

JSON

- Javascript Object Notation
  - Similar to XML, but more restricted
  - Solved the need to exchange client/server data on web
  - Designed to ease marshalling/unmarshalling

- Example javascript (unmarshallin

Note that this is a string

```
var myJSON = '{ "name":"John", "age":31, "city":"New York" }';
var myObj = JSON.parse(myJSON);
document.getElementById("demo").innerHTML = myObj.name;
```

https://www.w3schools.com/js/js_json_intro.asp

JSON

- Javascript Object Notation
  - Similar to XML, but more restricted
  - Solved the need to exchange client/server data on web
  - Designed to ease marshalling/unmarshalling

- Example javascript (unmarshallin

Convert string to object via parse

```
var myJSON = '{ "name":"John", "age":31, "city":"New York" }';
var myObj = JSON.parse(myJSON);
document.getElementById("demo").innerHTML = myObj.name;
```

https://www.w3schools.com/js/js_json_intro.asp

JSON

- Javascript Object Notation
  - Similar to XML, but more restricted
  - Solved the need to exchange client/server data on web
  - Designed to ease marshalling/unmarshalling

- Example javascript (unmarshallin

Can now access named object fields

```
var myJSON = '{ "name":"John", "age":31, "     ty":"New York" }';
var myObj = JSON.parse(myJSON);
document.getElementById("demo").innerHTML = myObj.name;
```

https://www.w3schools.com/js/js_json_intro.asp

JSON

- Javascript Object Notation
  - Similar to XML, but more restricted
  - Solved the need to exchange client/server data on web
  - Designed to ease marshalling/unmarshalling

- Example javascript (unmars

Awesome! But we're building apps in java, not javascript …

```
var myJSON = '{ "name":"John", "age":3     "city":"New York" }';
var myObj = JSON.parse(myJSON);
document.getElementById("demo").innerHTML = myObj.name;
```

https://www.w3schools.com/js/js_json_intro.asp

Firebase database

- Map String paths to objects
  - “/users/$uid/” for uid “alovelace” might map to

```
{
  "users": {
    "alovelace": {
      "name": "Ada Lovelace",
      "contacts": { "ghopper": true },
    },
    "ghopper": { ... },
    "eclarke": { ... }
  }
}
```

https://firebase.google.com/docs/database/android/structure-data

Firebase database

- Map String paths to objects
  - “/users/$uid/” for uid “alovelace” might map to

```
{
  "users": {
    "alovelace": {
      "name": "Ada Lovelace",
      "contacts": { "ghopper": true },
    },
    "ghopper": { ... },
    "eclarke": { ... }
  }
}
```

String key (“users”)

Firebase database

- Map String paths to objects
  - “/users/$uid/” for uid “alovelace” might map to

```
{
  "users": {
    "alovelace": {
      "name": "Ada Lovelace",
      "contacts": { "ghopper": true },
    },
    "ghopper": { ... },
    "eclarke": { ... }
  }
}
```

Keys can be defined by you, or by database via “push”

Firebase database

- Map String paths to objects
  - “/users/$uid/” for uid “alovelace” might map to

Colon defines mapping

```
{
  "users": {
    "alovelace": {
      "name": "Ada Lovelace",
      "contacts": { "ghopper": true },
    },
    "ghopper": { ... },
    "eclarke": { ... }
  }
}
```

Firebase database

- Map String paths to objects
  - “/users/$uid/” for uid “alovelace” might map to

```
{
  "users": {
    "alovelace": {
      "name": "Ada Lovelace",
      "contacts": { "ghopper": true },
    },
    "ghopper": { ... },
    "eclarke": { ... }
  }
}
```

Object value (“{ … }”)

Firebase database

- Map String paths to objects
  - “/users/$uid/” for uid “alovelace” might map to

```
{
  "users": {
    "alovelace": {
      "name": "Ada Lovelace",
      "contacts": { "ghopper": true },
    },
    "ghopper": { ... },
    "eclarke": { ... }
  }
}
```

What kind of objects can you define and store?

Firebase dat

- Map String paths to objects
  - “/users/$uid/” for uid “alovelac

Objects can be a:

- String
- Long
- Double
- Boolean
- Map<String, Object>
- List<Object>

```
{
  "users": {
    "alovelace": {
      "name": "Ada Lovelace",
      "contacts": { "ghopper": true },
    },
    "ghopper": { ... },
    "eclarke": { ... }
  }
}
```

Writing to Firebase

- Easy marshalling/unmarshalling is the point
    - Need to define Java objects for easy conversion
    - Two ways to do this …

```java
@IgnoreExtraProperties
public class User {

    public String username;
    public String email;

    public User() {
        // Default constructor
    }
    public User(String username, String email) {
        this.username = username;
        this.email = email;
    }
}
```

Writing to Firebase

- Easy marshalling/unmarshalling is the point
  - Need to define Java objects for easy conversion
  - Two ways to do this …

```java
@IgnoreExtraProperties
public class User {

    public String username;
    public String email;

    public User() {
        // Default constructor
    }
    public User(String username, String email) {
        this.username = username;
        this.email = email;
    }
}
```

Default constructor w/ no parameters

Writing to Firebase

- Easy marshalling/unmarshalling is the point
  - Need to define Java objects for easy conversion
  - Two ways to do this …

```java
@IgnoreExtraProperties
public class User {

    public String username;
    public String email;

    public User() {
        // Default constructor
    }
    public User(String username, String email) {
        this.username = username;
        this.email = email;
    }
}
```

Public fields with names matching JSON keys

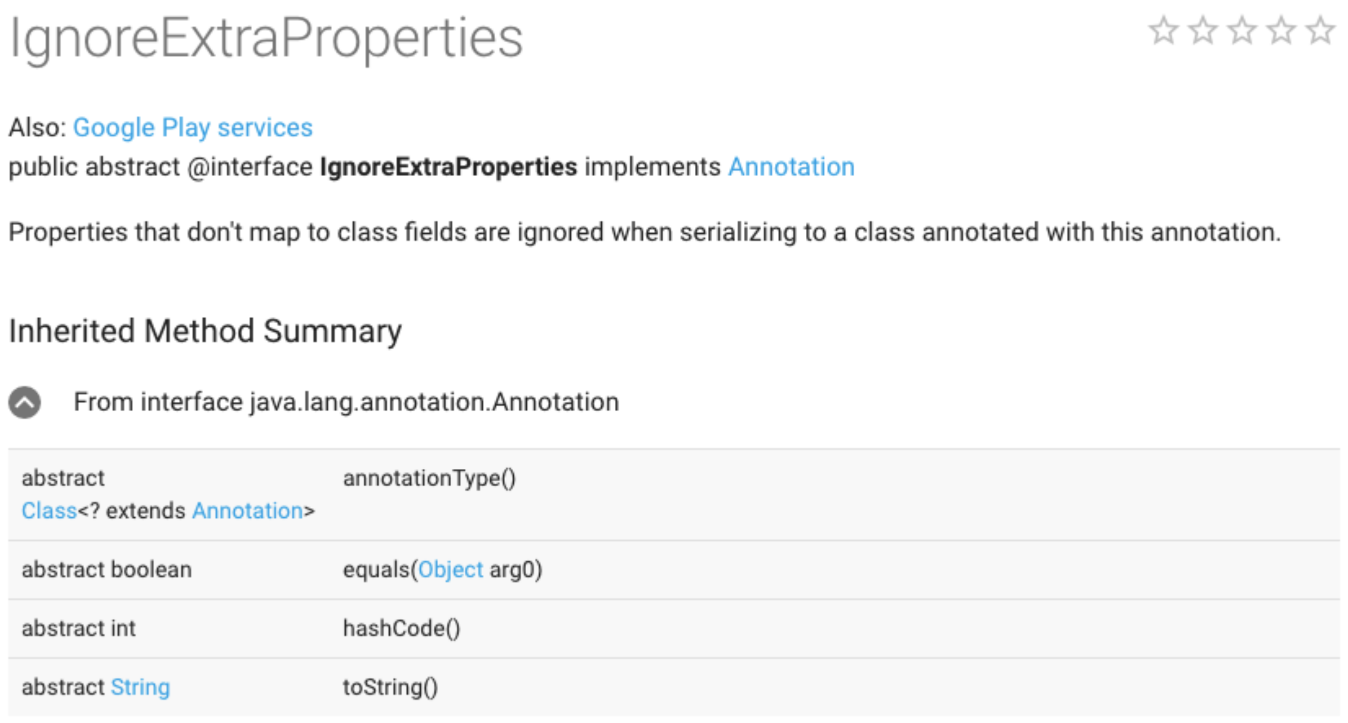
Writing to Firebase

- Easy marshalling/unmarshalling is the point
  - Need to define Java objects for easy conversion
  - Two ways to do this …

```
@IgnoreExtraProperties
public class User {

    public String username;
    public String email;

    public User() {
        // Default constructor
    }
    public User(String username, String email) {
        this.username = username;
        this.email = email;
    }
}
```

One last bit of magic …
`@IgnoreExtraProperties?`

Firebase documentation



IgnoreExtraProperties

Also: Google Play services

public abstract @interface **IgnoreExtraProperties** implements Annotation

Properties that don't map to class fields are ignored when serializing to a class annotated with this annotation.

Inherited Method Summary

From interface java.lang.annotation.Annotation

abstract Class<? extends Annotation>	annotationType()
abstract boolean	equals(Object arg0)
abstract int	hashCode()
abstract String	toString()

Writing to Firebase

- Easy marshalling/unmarshalling is the point
    - Need to define Java objects for easy conversion
    - Two ways to do this …

```
@IgnoreExtraProperties
public class User {

    public String username;
    public String email;

    public User() {
        // Default constructor
    }
    public User(String username, String email) {
        this.username = username;
        this.email = email;
    }
}
```

The other way to do this is with getter/setter methods.

Writing to Firebase

- Easy marshalling/unmarshalling is the point
  - Need to define Java objects for easy conversion
  - Two ways to do this …

```
@IgnoreExtraProperties
public class User {

    private String mUsername;
    private String mEmail;

    public User() {
        // Default constructor
    }
    public String getUsername() { return mUsername;}
    public void setUsername(String username) { mUsername = username; }
    public String getEmail() { return mEmail;}
    public void setEmail(String email) { mEmail = email; }
}
```

Fields are private

Writing to Firebase

- Easy marshalling/unmarshalling is the point
  - Need to define Java objects for easy conversion
  - Two ways to do this …

```
@IgnoreExtraProperties
public class User {

    private String mUsername;
    private String mEmail;

    public User() {
        // Default constructor
    }
    public String getUsername() { return mUsername;}
    public void setUsername(String username) { mUsername = username; }
    public String getEmail() { return mEmail;}
    public void setEmail(String email) { mEmail = email; }
}
```

Methods for accessing fields are public w/ specific names

Writing to Firebase

- Easy marshalling/unmarshalling is the point
  - Need to define Java objects for easy conversion
  - Two ways to do this …

```
@IgnoreExtraProperties
public class User {

    private String mUsername;
    private String mEmail;

    public User() {
        // Default constructor
    }
    public String getUsername() { return mUsername;}
    public void setUsername(String username) { mUsername = username; }
    public String getEmail() { return mEmail;}
    public void setEmail(String email) { mEmail = email; }
}
```

getX/setX where
X corresponds to JSON key

Writing to Firebase

- Easy marshalling/unmarshalling is the point
  - Need to define Java objects for easy conversion
  - Two ways to do this …

```java
@IgnoreExtraProperties
public class User {

    public String username;
    public String email;

    public User() {
        // Default constructor
    }
}
```

Going back to public fields …

Writing to Firebase

- Easy marshalling/unmarshalling is the point
  - Need to define Java objects for easy conversion
  - Two ways to do this …

```
@IgnoreExtraProperties
public class User {

    public String username;
    public String email;

    public User() {
        // Default constructor
    }
}
```

Regardless of which approach you choose, Androd will handle converting your object to and from JSON

Writing to Firebase

- Easy marshalling/unmarshalling is the point
  - Need to define Java objects for easy conversion
  - Two ways to do this …

```
@IgnoreExtraProperties
public class User {

    public String username;
    public String email;

    public User() {
        // Default constructor
    }
}
```

```
{
      ″username": ″lpcox",
      ″email": “lpcox@cs.duke.edu”
}
```

Writing to Firebase

- Easy marshalling/unmarshalling is the point
  - Need to define Java objects for easy conversion
  - Two ways to do this …

```
@IgnoreExtraProperties
public class User {

    public String username;
    public String email;

    public User() {
        // Default construct
    }
}
```

➡

```
{
    ″username": ″lpcox",
    ″email": “lpcox@cs.duke.edu”
}
```

Note that field names have to match exactly.

Writing to Firebase

- Easy marshalling/unmarshalling is the point
  - Need to define Java objects for easy conversion
  - Two ways to do this …

```java
@IgnoreExtraProperties
public class User {

    public String username;
    public String email;

    public User() {
        // Default constructor
    }
}
```

```
{
    ”username": ”lpcox",
    ”email": “lpcox@cs.duke.edu”
}
```

Note that field names have to match exactly.

Putting it all together

```java
private DatabaseReference mDatabase;
private FirebaseAuth mAuth;

// ...
mDatabase = FirebaseDatabase.getInstance().getReference();
mAuth = FirebaseAuth.getInstance();

String uid = mAuth.getCurrentUser().getUid();
String name = mAuth.getCurrentUser().getDisplayName();
String email = mAuth.getCurrentUser().getEmail();

writeNewUser(uid, name, email);

// ...

private void writeNewUser(String userId, String name, String email) {
    User user = new User(name, email);

    mDatabase.child("users").child(userId).setValue(user);
}
```

Putting it all together

Note the interplay between the authentication framework and our user database

```
private DatabaseReference mDatabase;
private FirebaseAuth mAuth;

// ...
mDatabase = FirebaseDatabase.getInstance().getReference();
mAuth = FirebaseAuth.getInstance();

String uid = mAuth.getCurrentUser().getUid();
String name = mAuth.getCurrentUser().getDisplayName();
String email = mAuth.getCurrentUser().getEmail();

writeNewUser(uid, name, email);

// ...

private void writeNewUser(String userId, String name, String email) {
    User user = new User(name, email);

    mDatabase.child("users").child(userId).setValue(user);
}
```

Putting it all together

Why store same data in authentication and database?

```
private DatabaseReference mDatabase;
private FirebaseAuth mAuth;

// ...
mDatabase = FirebaseDatabase.getInstance().getReference();
mAuth = FirebaseAuth.getInstance();

String uid = mAuth.getCurrentUser().getUid();
String name = mAuth.getCurrentUser().getDisplayName();
String email = mAuth.getCurrentUser().getEmail();

writeNewUser(uid, name, email);

// ...

private void writeNewUser(String userId, String name, String email) {
    User user = new User(name, email);

    mDatabase.child("users").child(userId).setValue(user);
}
```

Putting it all together

```java
private DatabaseReference mDatabase;
private FirebaseAuth mAuth;

// ...
mDatabase = FirebaseDatabase.getInstance().getReference();
mAuth = FirebaseAuth.getInstance();

String uid = mAuth.getCurrentUser().getUid();
String name = mAuth.getCurrentUser().getDisplayName();
String email = mAuth.getCurrentUser().getEmail();

writeNewUser(uid, name, email);

// ...

private void writeNewUser(String userId, String name, String email) {
    User user = new User(name, email);

    mDatabase.child("users").child(userId).setValue(user);
}
```

Note this is the same User class defined earlier

Putting it all together

```java
private DatabaseReference mDatabase;
private FirebaseAuth mAuth;

// ...
mDatabase = FirebaseDatabase.getInstance().getReference();
mAuth = FirebaseAuth.getInstance();

String uid = mAuth.getCurrentUser().getUid();
String name = mAuth.getCurrentUser().getDisplayName();
String email = mAuth.getCurrentUser().getEmail();

writeNewUser(uid, name, email);

// ...

private void writeNewUser(String userId, String name, String email) {
    User user = new User(name, email);

    mDatabase.child("users").child(userId).setValue(user);
}
```

To access the database we walk the tree with child()

Putting it all together

```
private DatabaseReference mDatabase;
private FirebaseAuth mAuth;

// ...
mDatabase = FirebaseDatabase.getInstance().getReference();
mAuth = FirebaseAuth.getInstance();

String uid = mAuth.getCurrentUser().getUid();
String name = mAuth.getCurrentUser().getDisplayName();
String email = mAuth.getCurrentUser().getEmail();

writeNewUser(uid, name, email);

// ...

private void writeNewUser(String userId, String name, String email) {
    User user = new User(name, email);

    mDatabase.child("users").child(userId).setValue(user);
}
```

Top key is “users”

Putting it all together

```java
private DatabaseReference mDatabase;
private FirebaseAuth mAuth;

// ...
mDatabase = FirebaseDatabase.getInstance().getReference();
mAuth = FirebaseAuth.getInstance();

String uid = mAuth.getCurrentUser().getUid();
String name = mAuth.getCurrentUser().getDisplayName();
String email = mAuth.getCurrentUser().getEmail();

writeNewUser(uid, name, email);

// ...

private void writeNewUser(String userId, Strin    me, String email) {
    User user = new User(name, email);

    mDatabase.child("users").child(userId).setValue(user);
}
```

Next key is the user id

Putting it all together

```
private DatabaseReference mDatabase;
private FirebaseAuth mAuth;

// ...
mDatabase = FirebaseDatabase.getInstance().getReference();
mAuth = FirebaseAuth.getInstance();

String uid = mAuth.getCurrentUser().getUid();
String name = mAuth.getCurrentUser().getDisplayName();
String email = mAuth.getCurrentUser().getEmail();

writeNewUser(uid, name, email);

// ...

private void writeNewUser(String userId, Strin    me, String email) {
    User user = new User(name, email);

    mDatabase.child("users").child(userId).setValue(user);
}
```

What does each all to child return?

Putting it all together

```java
private DatabaseReference mDatabase;
private FirebaseAuth mAuth;

// ...
mDatabase = FirebaseDatabase.getInstance().getReference();
mAuth = FirebaseAuth.getInstance();

String uid = mAuth.getCurrentUser().getUid();
String name = mAuth.getCurrentUser().getDisplayName();
String email = mAuth.getCurrentUser().getEmail();

writeNewUser(uid, name, email);

// ...

private void writeNewUser(String userId, String na      ring email) {
    User user = new User(name, email);

    mDatabase.child("users").child(userId).setValue(user);
}
```

Then we map the user id to an instance of the User class

Putting it all together

```
private DatabaseReference mDatabase;
private FirebaseAuth mAuth;

// ...
mDatabase = FirebaseDatabase.getInstance().getReference();
mAuth = FirebaseAuth.getInstance();

String uid = mAuth.getCurrentUser().getUid();
String name = mAuth.getCurrentUser().getDispla
String email = mAuth.getCurrentUser().getEmai

writeNewUser(uid, name, email);

// ...

private void writeNewUser(String userId, String nam       ring email) {
    User user = new User(name, email);

    mDatabase.child("users").child(userId).setValue(user);
}
```

Firebase will convert this Object into a string and then a JSON object

Putting it all together

```
private DatabaseReference mDatabase;
private FirebaseAuth mAuth;

// ...
mDatabase = FirebaseDatabase.getInstance().getReference();
mAuth = FirebaseAuth.getInstance();

String uid = mAuth.getCurrentUser().getUid();
String name = mAuth.getCurrentUser().getDispla
String email = mAuth.getCurrentUser().getEmai

writeNewUser(uid, name, email);

// ...

private void writeNewUser(String userId, String nan      ring email) {
    User user = new User(name, email);

    mDatabase.child("users").child(userId).setValue(user);
}
```

This code will overwrite whatever was previously mapped to by the user id

Putting it all together

```
private DatabaseReference mDatabase;
private FirebaseAuth mAuth;

// ...
mDatabase = FirebaseDatabase.getInstance().getReference();
mAuth = FirebaseAuth.getInstance();

String uid = mAuth.getCurrentUser().getUid();
String name = mAuth.getCurrentUser().getDispla
String email = mAuth.getCurrentUser().getEmai

writeNewUser(uid, name, email);

// ...

private void writeNewUser(String userId, String nan      ring email) {
    User user = new User(name, email);

    mDatabase.child("users").child(userId).setValue(user);
}
```

Might want to update a field within the object

Putting it all together

```
private DatabaseReference mDatabase;
private FirebaseAuth mAuth;

// ...
mDatabase = FirebaseDatabase.getInstance().getReference();
mAuth = FirebaseAuth.getInstance();

String uid = mAuth.getCurrentUser().getUid();
String name = mAuth.getCurrentUser().getDispla
String email = mAuth.getCurrentUser().getEmai

writeNewUser(uid, name, email);

// ...

private void writeNewUser(String userId, String nan      ring email) {
    User user = new User(name, email);

    mDatabase.child("users").child(userId).child("username”).setValue(name);
}
```

Might want to update a field within the object

Putting it all together

```
private DatabaseReference mDatabase;
private FirebaseAuth mAuth;

// ...
mDatabase = FirebaseDatabase.getInstance().getReference();
mAuth = FirebaseAuth.getInstance();

String uid = mAuth.getCurrentUser().getUid();
String name = mAuth.getCurrentUser().getDispla
String email = mAuth.getCurrentUser().getEmai

writeNewUser(uid, name, email);

// ...

private void writeNewUser(String userId, String        , String email) {
    User user = new User(name, email);

    mDatabase.child("users").child(userId).child("username”).setValue(name);
}
```

Maybe you don’t want to name your objects, like for messages. Use push().

Putting it all together

```java
private DatabaseReference mDatabase;
private FirebaseAuth mAuth;

// ...
mDatabase = FirebaseDatabase.getInstance().getReference();
mAuth = FirebaseAuth.getInstance();

String uid = mAuth.getCurrentUser().getUid();
String name = mAuth.getCurrentUser().getDispla
String email = mAuth.getCurrentUser().getEmai

writeNewUser(uid, name, email);

// ...

private void writeNewUser(String userId, String      , String email) {
    User user = new User(name, email);

    mDatabase.child("users").push(user);
}
```

Maybe you don’t want to name your objects, like for messages. Use push().

Putting it all together

```
private DatabaseReference mDatabase;
private FirebaseAuth mAuth;

// ...
mDatabase = FirebaseDatabase.getInstance().getReference();
mAuth = FirebaseAuth.getInstance();

String uid = mAuth.getCurrentUser().getUid();
String name = mAuth.getCurrentUser().getDispla
String email = mAuth.getCurrentUser().getEmai

writeNewUser(uid, name, email);

// ...

private void writeNewUser(String userId, Stri      me, String email) {
    User user = new User(name, email);

    mDatabase.child("users").push().setValue(user);
}
```

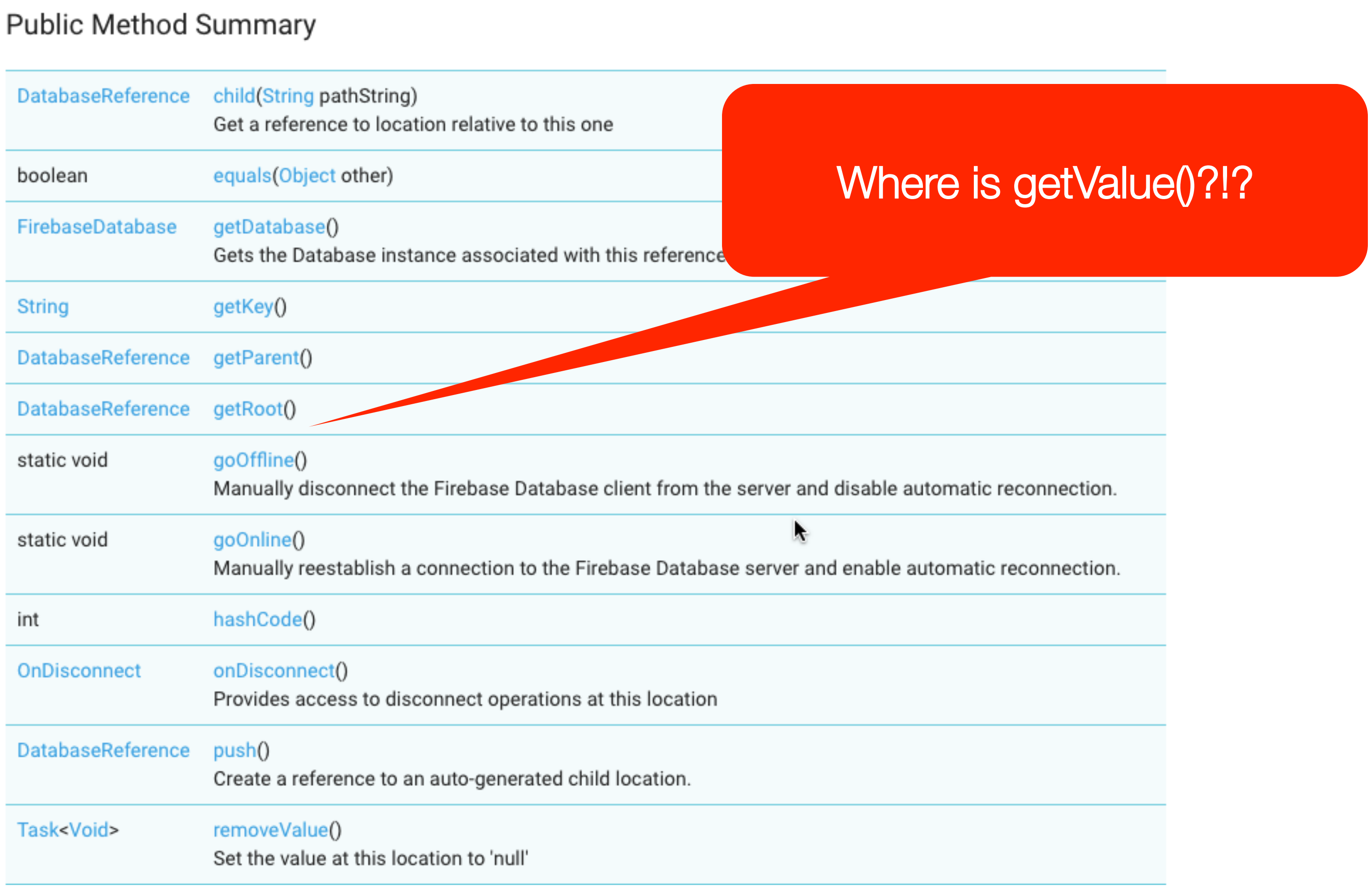
push() generates a random path name

Putting it all together

```
private DatabaseReference mDatabase;
private FirebaseAuth mAuth;

// ...
mDatabase = FirebaseDatabase.getInstance().getReference();
mAuth = FirebaseAuth.getInstance();

String uid = mAuth.getCurrentUser().getUid();
String name = mAuth.getCurrentUser().getDispla
String email = mAuth.getCurrentUser().getEmai

writeNewUser(uid, name, email);

// ...

private void writeNewUser(String userId, String      e, String email) {
    User user = new User(name, email);

    mDatabase.child("users").push().setValue(user);
}
```

Hmm, there's setValue, maybe we can use getValue to read?

DatabaseReference API



Public Method Summary

DatabaseReference	child(String pathString) Get a reference to location relative to this one
boolean	equals(Object other)
FirebaseDatabase	getDatabase() Gets the Database instance associated with this reference
String	getKey()
DatabaseReference	getParent()
DatabaseReference	getRoot()
static void	goOffline() Manually disconnect the Firebase Database client from the server and disable automatic reconnection.
static void	goOnline() Manually reestablish a connection to the Firebase Database server and enable automatic reconnection.
int	hashCode()
OnDisconnect	onDisconnect() Provides access to disconnect operations at this location
DatabaseReference	push() Create a reference to an auto-generated child location.
Task<Void>	removeValue() Set the value at this location to 'null'

Where is getValue()?!?

What about reading?

```
DatabaseReference mPostReference = FirebaseDatabase.getInstance()
                                                            .getReference()
                                                            .child(”posts");
// …
ValueEventListener postListener = new ValueEventListener() {
    @Override
    public void onDataChange(DataSnapshot dataSnapshot) {
        // Get Post object and use the values to update the UI
        Post post = dataSnapshot.getValue(Post.class);
        // ...
    }

    @Override
    public void onCancelled(DatabaseError databaseError) {
        // Getting Post failed, log a message
        Log.w(TAG, "loadPost:onCancelled", databaseError.toException());
        // ...
    }
};
mPostReference.addValueEventListener(postListener);
```

What about reading?

```
DatabaseReference mPostReference = FirebaseDatabase.getInstance()
                                                    .getReference()
                                                    .child(”posts");
// …
ValueEventListener postListener = new ValueEventListener() {
    @Override
    public void onDataChange(DataSnapshot dataSnapshot) {
        // Get Post object and use the values to update the UI
        Post post = dataSnapshot.getValue(Post.class);
        // ...
    }

    @Override
    public void onCancelled(DatabaseError da
        // Getting Post failed, log a message
        Log.w(TAG, "loadPost:onCancelled", dat     Error.toException());
        // ...
    }
};
mPostReference.addValueEventListener(postListener);
```

Register for a callback when subtree changes.

What about reading?

```
DatabaseReference mPostReference = FirebaseDatabase.getInstance()
                                                   .getReference()
                                                   .child(”posts");
// …
ValueEventListener postListener = new ValueEventListener() {
    @Override
    public void onDataChange(DataSnapshot dataSnapshot) {
        // Get Post object and use the val es to update the UI
        Post post = dataSnapshot.getValue(Po  .class);
        // ...
    }

    @Override
    public void onCancelled(DatabaseError d
        // Getting Post failed, log a messa
        Log.w(TAG, "loadPost:onCancelled",
        // ...
    }
};
mPostReference.addValueEventListener(postListener);
```

When subtree changes you get a snapshot of the database

What about reading?

```
DatabaseReference mPostReference = FirebaseDatabase.getInstance()
                                                    .getReference()
                                                    .child(”posts");
// …
ValueEventListener postListener = new ValueEventListener() {
    @Override
    public void onDataChange(DataSnapshot dataSnapshot) {
        // Get Post object and use the values to update the UI
        Post post = dataSnapshot.getValue(Post.class);
        // ...
    }

    @Override
    public void onCancelled(DatabaseError d
        // Getting Post failed, log a messa
        Log.w(TAG, "loadPost:onCancelled",
        // ...
    }
};
mPostReference.addValueEventListener(postListener);
```

And then … getValue()!!

What about reading?

```
DatabaseReference mPostReference = FirebaseDatabase.getInstance()
                                                   .getReference()
                                                   .child(”posts");
// …
ValueEventListener postListener = new ValueEventListener() {
    @Override
    public void onDataChange(DataSnapshot dataSnapshot) {
        // Get Post object and use the values to update the UI
        Post post = dataSnapshot.getValue(Post.class);
        // ...
    }

    @Override
    public void onCancelled(DatabaseError d
        // Getting Post failed, log a messa
        Log.w(TAG, "loadPost:onCancelled",
        // ...
    }
};
mPostReference.addValueEventListener(postListener);
```

Why pass in a class reference?