```
<<enumeration>>
~UserField
UID
IMAGE_BMP
FIRST_NAME
LAST_NAME
BIRTH_DATE
SEX
WEIGHT_GOAL
CURRENT_WEIGHT
PREVIOUS_WEIGHTS
+ UserField(fieldName: String)
```

```
<<interface>>
TaskListener                                    T
Used as a callback for asynchronous calls to Firebase
Example of Strategy pattern
+ onSuccess(value: T)
+ onFailure(e: Exception?)
```

```
LineDataConfig
+ label: String
+ startDate: Date
+ endDate: Date
+ movingAverageTerms: Int? = 0
```

Note that "~" means internal to this module. (the class and it's subclasses cannot be seen by the UI module)

The below class hierarchy allows us to pass a Manager object to the UI layer from the data layer while keeping data processing and cloud interactions internal to the data layer module.

Due to the differences between Workouts and Users requirements, this is a general structure and doesn't include too many abstract functions or parameters. Instead it gives us a lenient tree-like structure that can be added to very easily due to this leniency. This is clear from us being able to have sub-receivers, sub-dataprocessors, etc. Allowing us to break up the work being done into modular pieces.

This also means that on the front-end, we could pass only a UserManagementReceiver to a component if we want to limit it to only user management, and not authentication.

These implement a proxy design pattern since the data is processed as a sort of pipeline from Cloud -> DP -> Provider and Receiver -> DP -> Cloud

This is a **builder pattern**. We use this here because of how many arguments the FitUser has, and due to the fact that some are required and some are not. This way we can abstract the creation logic and exceptions away from FitUser so it's primarily a data class.

Note that this is used to create a HashMap to provide to the cloud for user-creation and used when a FitUser is requested from the cloud and we're just provided a DocumentSnapshot instead of a FitUser object

```
<<interface>>
SwimlaneItem
This is an interface so the UI knows what can be displayed in a SwimlaneItemView which requires a title and image.  Subclassed by other classes not in this diagram
+ getTitle(): String
+ getDrawable(): Drawable?
```

```
FitUser
+ birthDate: Date
+ firstName: Date
+ lastName: Date
+ sex: UserSex
+ imageBmp: Bitmap?
+ currentWeight: Weight?
+ currentHeight: Height?
+ weightGoal: Weight?
+ FitUser(birthDate: Date, ...)
+ FitUser(fitUser: FitUser)
+ getFullName(): String
```

This includes functionality available to any user, ie. viewing another user's profile

I'm not going to get into Weights and Heights in this diagram, but we have our own extension of Android's Measure framework to manage timestamps of Measures and conversions in the edge case of a user changing their locale.
Units are stored server-side and converted if required by locale or user preference.

If you need more info, the code is very clear and under datamodel > data > measure in master

```
FitUserBuilder
Instead of making this a big UML class, note that the bellow attrs all have setters that return the FitUserBuilder for easy chaining
- birthDate: Date
- firstName: Date
- lastName: Date
- sex: UserSex
- imageBmp: Bitmap?
- currentWeight: Weight?
- currentHeight: Height?
- weightGoal: Weight?
+ hasRequiredInputs(): Boolean
+ build(): FitUser <<throws UserBuildException>>
+ convertFieldsToHashMap(): HashMap<UserField, Any>
```

```
AuthenticatedUser
+ fbUser: FirebaseUser
+ previousWeights: List<Weight>
+ previousHeights: List<Height>
+ addPreviousWeight(weight: Weight)
+ getPreferences(): UserPreferences
```

This includes functionality only available to the authenticated user

FirebaseUser includes auth data required to push / pull from Firebase, provided by Firebase Android Library

```
Manager
- tag: String
# Manager()
```

tag is used for logging

```
~CloudInterface
# db: Firestore
# auth: FirebaseAuth
```

```
~DataProcessor
# getCloudInterface(): CloudInterface
```

```
<<singleton>>
UserManager
Contains all user-related CloudInterfaces, DataProcessors, Receivers, and Providers.

These are broken up into subgroups (ie. Auth, Management, Query) with getters for each (getters are internal for DataProcessor, CloudInterface)

Example of a Singleton pattern for cases where caching is present
```

```
Receiver
# getDataProcessor(): DataProcessor
```

```
Provider
# getDataProcessor(): DataProcessor
```

Receiver and Providers are proxies to/from the DataProcessors and DataProcessors are proxies to/from the CloudInterfaces

```
UserQueryCloudInterface
+ getFirebaseUser(): FirebaseUser?
+ getUserById(uid: String, listener: TaskListener<FitUser?>)
```

```
UserManagementCloudInterface
+ createFirestoreUser(fields: HashMap<String, Any?>, listener: TaskListener<Unit?>)
+ updateUserEmail(email: String, listener: TaskListener<Unit?>)
+ updateFirestoreUser(fields: HashMap<String, Any?>, listener: TaskListener<Unit?>)
+ updateUserPassword(password: String, listener: TaskListener<Unit?>)
+ sendVerificationEmail(listener: TaskListener<Unit?>)
+ signUpUser(email: String, password: String, listener: TaskListener<Unit?>)
+ deleteUser(listener: TaskListener<Unit?>)
+ sendPasswordResetEmail(listener: TaskListener<Unit?>)
```

```
UserAuthCloudInterface
+ authenticateUser(email: String, password: String, listener: TaskListener<Unit?>)
+ reAuthenticateUser(email: String, password: String, listener: TaskListener<Unit?>)
```

```
UserQueryDataProcessor
+ getAuthenticatedUser(listener: TaskListener<AuthenticatedUser?>)
+ getFirebaseUser(): FirebaseUser?
+ getUserById(uid: String, listener: TaskListener<FitUser?>)
+ createUserFromDocument(doc: DocumentSnapshot): FitUser?
```

```
UserManagementDataProcessor
+ signUpUser(email: String, password: String, listener: TaskListener<Unit?>)
+ createFirestoreUser(builder: FitUserBuilder, listener: TaskListener<Unit?>)
+ updateFirestoreUser(fields: HashMap<UserField, Any?>, listener: TaskListener<Unit?>)
+ updateUserEmail(email: String, listener: TaskListener<Unit?>)
+ updateUserPassword(password: String, listener: TaskListener<Unit?>)
+ sendVerificationEmail(listener: TaskListener<Unit?>)
+ sendPasswordResetEmail(listener: TaskListener<Unit?>)
+ deleteUser(listener: TaskListener<Unit?>)
```

```
UserAuthDataProcessor
+ authenticateUser(email: String, password: String, listener: TaskListener<Unit?>)
+ reAuthenticateUser(email: String, password: String, listener: TaskListener<Unit?>)
+ logoutUser(listener: TaskListener<Unit?>)
```

```
UserManagementReceiver
+ signUpUser(email: String, password: String, listener: TaskListener<Unit?>)
+ createFirestoreUser(builder: FitUserBuilder, listener: TaskListener<Unit?>)
+ updateFirestoreUser(fields: HashMap<UserField, Any?>, listener: TaskListener<Unit?>)
+ updateUserEmail(email: String, listener: TaskListener<Unit?>)
+ updateUserPassword(password: String, listener: TaskListener<Unit?>)
+ sendVerificationEmail(listener: TaskListener<Unit?>)
+ sendPasswordResetEmail(listener: TaskListener<Unit?>)
+ deleteUser(listener: TaskListener<Unit?>)
+ createUserFromDocument(doc: DocumentSnapshot): FitUser?
```

```
UserAuthReceiver
+ authenticateUser(email: String, password: String, listener: TaskListener<Unit?>)
+ reAuthenticateUser(email: String, password: String, listener: TaskListener<Unit?>)
+ logoutUser(listener: TaskListener<Unit?>)
```

```
UserProvider
+ getAuthenticatedUser(listener: TaskListener<AuthenticatedUser?>)
+ getFirebaseUser(): FirebaseUser?
+ getUserById(uid: String, listener: TaskListener<FitUser?>)
+ getBodyWeightLineDataSet(userId: String, listener: TaskListener<List<Workout>>)
+ getRecentWorkouts(config: LineDataConfig, listener: TaskListener<LineDataSet>)
```