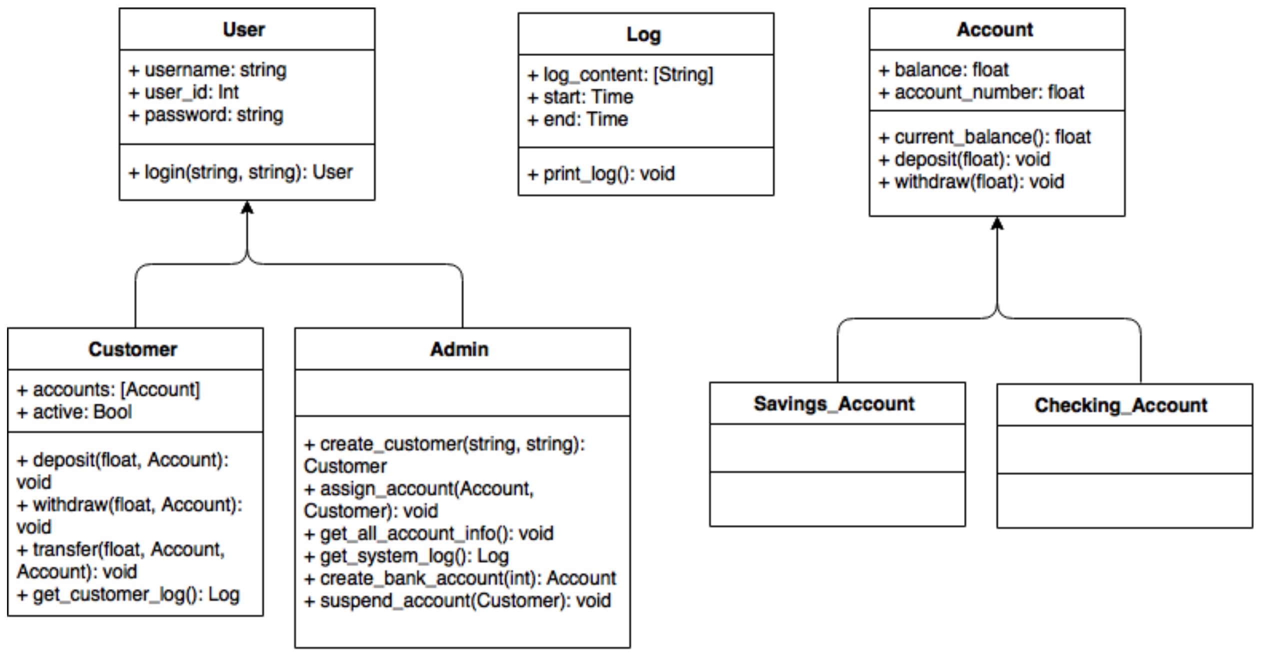
## Original Class Diagram


User
+ username: string
+ user_id: Int
+ password: string
+ login(string, string): User
Log
+ log_content: [String]
+ start: Time
+ end: Time
+ print_log(): void
Account
+ balance: float
+ account_number: float
+ current_balance(): float
+ deposit(float): void
+ withdraw(float): void
Customer
+ accounts: [Account]
+ active: Bool
+ deposit(float, Account): void
+ withdraw(float, Account): void
+ transfer(float, Account, Account): void
+ get_customer_log(): Log
Admin
+ create_customer(string, string): Customer
+ assign_account(Account, Customer): void
+ get_all_account_info(): void
+ get_system_log(): Log
+ create_bank_account(int): Account
+ suspend_account(Customer): void
Savings_Account
Checking_Account

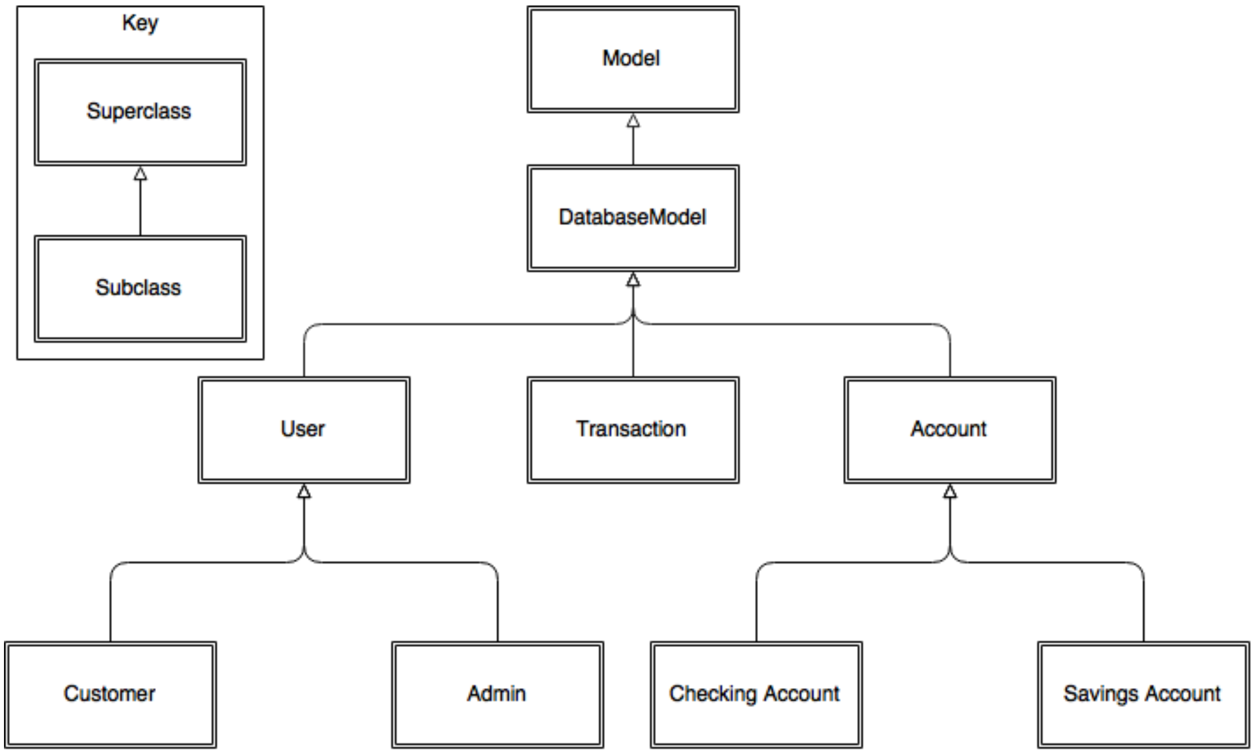
The User class is defined to represent the main functionality that all users share. Admin and Customer inherit from User and also add onto it (Customer with new functions and attributes, Admin with just new Functions).

Account is used to define the main attributes (account number, balance) and functionality (deposit, withdraw) that all accounts share. Savings_Account and Checking_Account inherit from it to get this functionality.

Log represents a log for a period of time, with start and end time illustrating the time period and the log_content containing all of the log files. This is intended to be used by Users to view different log types.

Assumes that passwords can be stored in plain text.

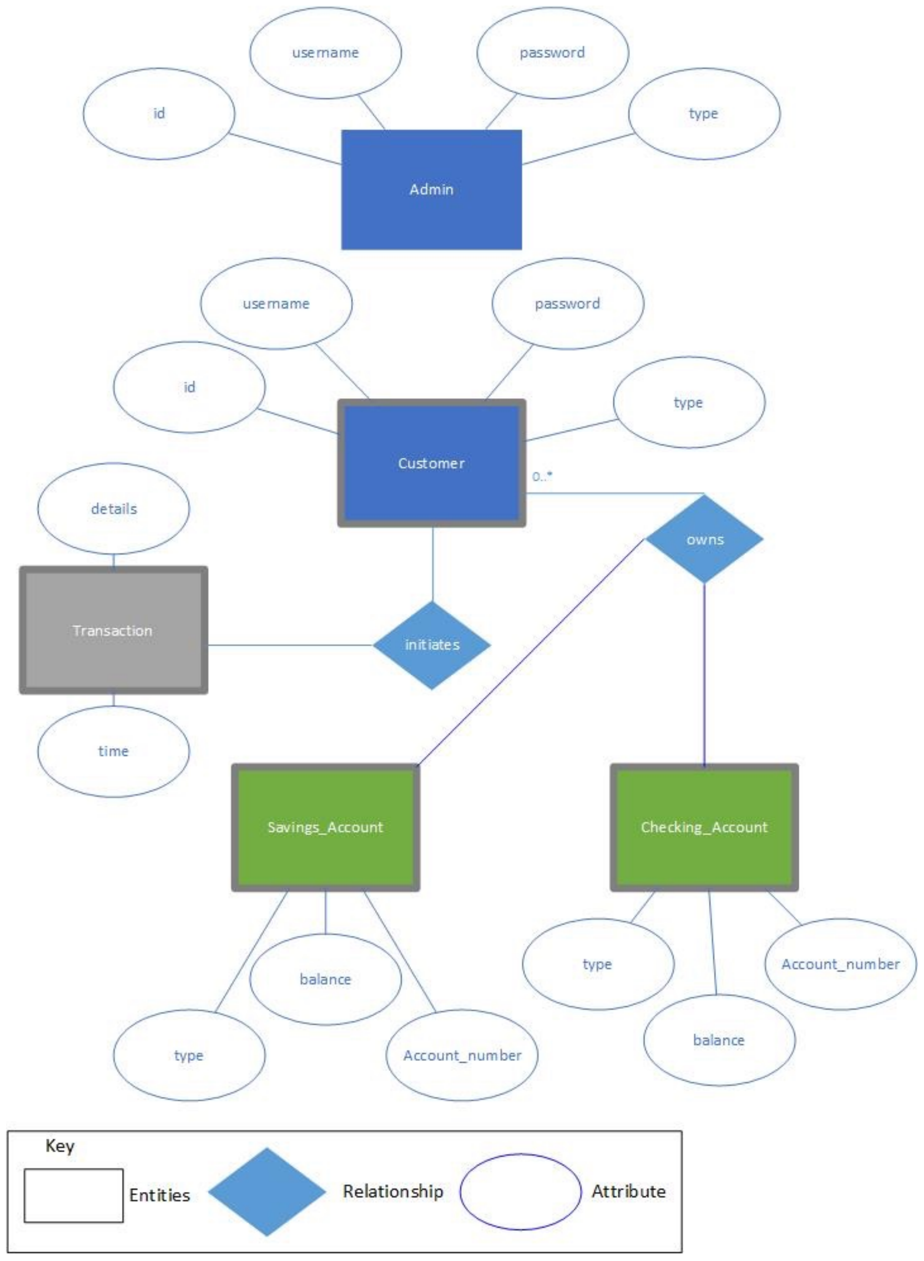
## Generalization View



The generalization diagram shows the general overview of the architecture of our system. The User object has Admin and Customer objects that inherit from it because they share core functionality but each have their own special functionality. The two account types, Savings_Account and Checking_Account, inherit from the Account because the Account object has all of the major Account functionality, they are just specific versions of it. Finally, the Account, User, and Transaction classes all inherit from DatabaseModel class. DatabaseModel inherits from the peewee Model to allow its subclasses to use that functionality. It also defines the way that the models should connect to the database.

Assumes that the generalization structure beyond Model (which comes from peewee) does not need to be illustrated.

## Entity Relationship View

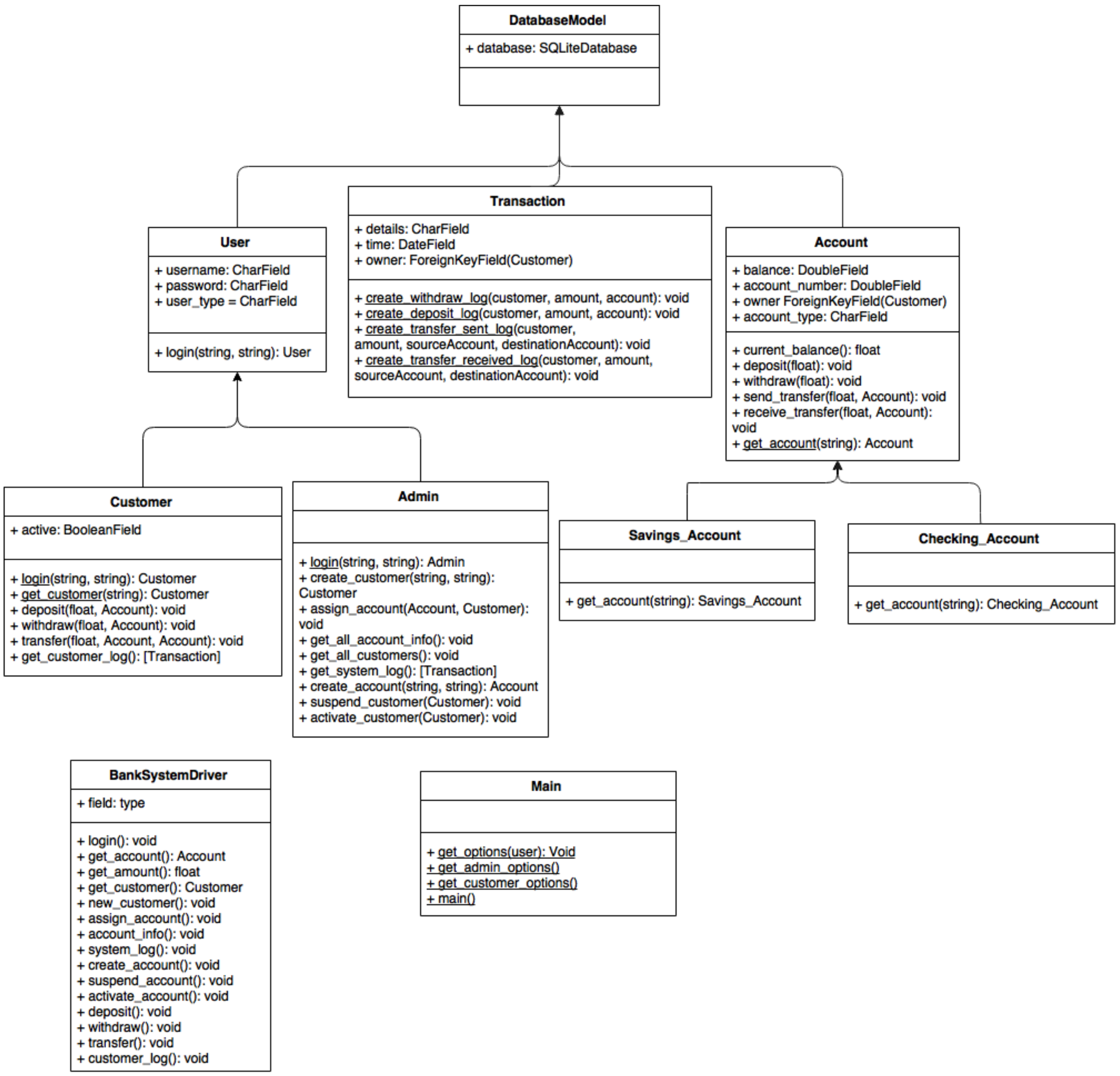


The two types of users currently in the system, Customer and Admin, are separate tables in the database because the python library we are using to access it maps tables to Python classes. This means that we can have both Customer and Admin as inheritors of User's attributes and add new ones without needing to relate to a User table. So we can keep the Customers and Admins completely separate in the database without requiring extra maintenance. When the user logs into the system, the system will have to check against the Admin table and the Customer Table to check the credentials.

Customers own two different types of accounts: a savings account and a checking account. Both of these tables the some attributes in common (balance, account number, type), but are unrelated tables for the same reason that Customer and Admin are not related in the database: we can keep them distinct while not requiring extra code by using peewee to interface from our Python classes to the database. Each account is related to a customer that owns it by a ForeignKey.

The final table is Transaction which stores information about what transactions have occurred in the system. The information about a transaction is stored in the details attribute and the time of the transaction is stored in the time attribute. Transactions are related to the Customer whose account was involved. Transfers that occur between users are broken into a pair of transactions, one for the sender and one for the receiver, so that the Account information stored in the details (a new balance) is not shown to the other customer. Relating the transactions to the owners rather than to the accounts also enables a user to keep track of their transactions, even if they no longer hold that account. This would be covered in the details data point.

## New Class Diagram


DatabaseModel
+ database: SQLiteDatabase
Transaction
+ details: CharField
+ time: DateField
+ owner: ForeignKeyField(Customer)
+ create_withdraw_log(customer, amount, account): void
+ create_deposit_log(customer, amount, account): void
+ create_transfer_sent_log(customer, amount, sourceAccount, destinationAccount): void
+ create_transfer_received_log(customer, amount, sourceAccount, destinationAccount): void
User
+ username: CharField
+ password: CharField
+ user_type = CharField
+ login(string, string): User
Account
+ balance: DoubleField
+ account_number: DoubleField
+ owner ForeignKeyField(Customer)
+ account_type: CharField
+ current_balance(): float
+ deposit(float): void
+ withdraw(float): void
+ send_transfer(float, Account): void
+ receive_transfer(float, Account): void
+ get_account(string): Account
Customer
+ active: BooleanField
+ login(string, string): Customer
+ get_customer(string): Customer
+ deposit(float, Account): void
+ withdraw(float, Account): void
+ transfer(float, Account, Account): void
+ get_customer_log(): [Transaction]
Admin
+ login(string, string): Admin
+ create_customer(string, string): Customer
+ assign_account(Account, Customer): void
+ get_all_account_info(): void
+ get_all_customers(): void
+ get_system_log(): [Transaction]
+ create_account(string, string): Account
+ suspend_customer(Customer): void
+ activate_customer(Customer): void
Savings_Account
+ get_account(string): Savings_Account
Checking_Account
+ get_account(string): Checking_Account
BankSystemDriver
+ field: type
+ login(): void
+ get_account(): Account
+ get_amount(): float
+ get_customer(): Customer
+ new_customer(): void
+ assign_account(): void
+ account_info(): void
+ system_log(): void
+ create_account(): void
+ suspend_account(): void
+ activate_account(): void
+ deposit(): void
+ withdraw(): void
+ transfer(): void
+ customer_log(): void
Main
+ get_options(user): Void
+ get_admin_options()
+ get_customer_options()
+ main()

The DatabaseModel class is inherited by all of the other Model classes. It inherits from peewee's Model class and defines the database used by the rest of the models.

The User class is defined to represent the main functionality that all users share. Admin and Customer inherit from User and also add onto it (Customer with new functions and attributes, Admin with just new Functions). Customer and Admin both override login to check for their specific login cases (Customer must check its table and must check for active, Admin must check its own database). All of these attributes are defined as peewee field objects so they can relate to database fields.

Account is used to define the main attributes (account number, balance) and functionality (deposit, withdraw) that all accounts share. Savings_Account and Checking_Account inherit from it to get this functionality. They also override the get_account function from Account to check their individual databases. All of these attributes are defined as peewee field objects so they can relate to database fields.

Transaction stores a list of all of the transactions performed in the system, as well as the owner of that transaction (whose account it relates to). It contains functions to create different types of transactions. All of these attributes are defined as peewee field objects so they can relate to database fields.

BankSystemDriver is the Controller that controls interaction between the view and the models. It stores the user that is logged in and enables the user to perform actions by interacting with the models through its functions.

Main is the main file for the python file. It acts as the view. It contains the menu options that users can input as well as how to translate those options into functionality.

Assumes that attributes created by peewee (object ids, related_names) are not shown in Class Diagram.