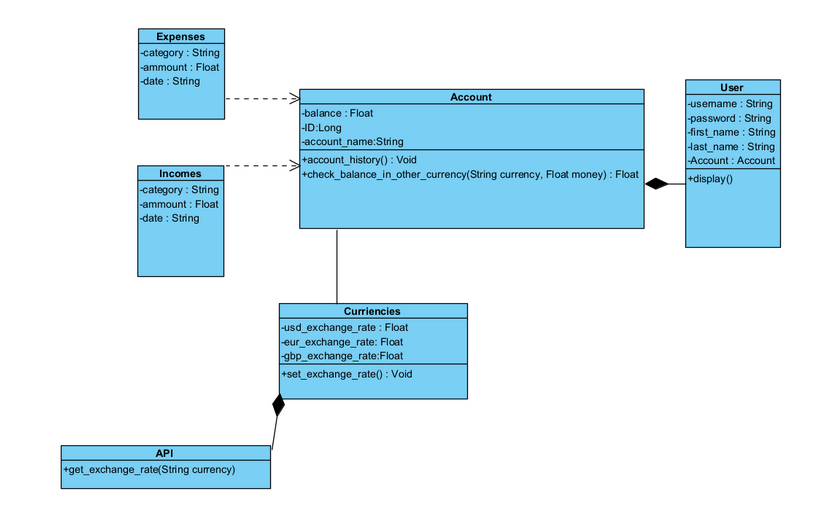
Diagram klas projektu



SOLID

1**.”Single responsibility”**

W naszym diagramie klas jest zachowana pierwsza zasada solid „Single responsibility”.

Każda z naszych klas jest odpowiedzialna za konkretną rzecz tj. klasy Expenses i Incomes są rozdzielone zamiast tworzyć jednej klasy dotyczącej wydatków i przychodów. Tym sposobem dane przechowywane w klasach są bardziej czytelne i zapewniany jest do nich łatwiejszy dostęp.

2**.”Open/Closed”**

W naszych klasach brakuje zastosowania polimorfizmu który umożliwiałby nam modyfikowanie metod wirtualnych w czasie wykonywania programu. Bez używania polimorfizmu, musielibyśmy już na etapie pisania programu, wiedzieć jak będzie się on zachowywał. To za sprawą wczesnego wiązania, które musi być dostarczone kompilatorowi w momencie kompilacji i linkowania.  
W przypadku użycia polimorfizmu dostajemy nieograniczone możliwości projektowania aplikacji, gdzie zachowanie programu może się ciągle zmieniać. Wtedy klasy moglibyśmy modyfikować bez ich edycji. Po wprowadzeniu klas abstrakcyjnych powinniśmy zadbać o zastosowanie się do zasad **„Interface segregation”** jak **i „Liskov substitution”**,które to są do siebie zbliżone. Należy uważać by nie złamać ich np. stosując dziedziczenia bez użycia mechanizmu polimorfizmu. Nasze klasy bazowe powinny być jak najbardziej rozbite (bez zawierania w nich bardzo dużej ilości metod).

3.Zasada **DRY** (Dont’t repeat yourself)

W naszym programach powinniśmy usunąć fragmenty kodu które powtarzają się tak jak np. różne definicje tej samej klasy Account. Dzięki temu gdy konieczne będzie zastosowanie jakiejś zmiany w kodzie będziemy mogli zastosować ją w jednym miejscu. Również zwiększona zostanie czytelność naszego kodu.

4.Zasada **Interface Segregation Principle**

W naszym programie nie mamy jednego dużego interfejsu odpowiedzialnego np. Za zmianę wydatków i przychodów, tylko 2 osobne interfejsy zajmujące się obydwoma operacjami osobno.

5.Zasada **Dependency Inversion Principle**

W naszym kodzie, zastosowanie zasady odwrócenia zależności polega na używaniu interfejsu API, co pozwala na elastyczne podmienianie różnych API bez konieczności zmiany kodu klasy Curriences. Ta zasada sprzyja również zastosowaniu wstrzykiwania zależności, co pozwala na jeszcze większą elastyczność w konfigurowaniu zależności między modułami.

GRASP

1.Information expert

W naszym kodzie dla przykładu klasa curriences przechowuje informacje na temat kursów walut, ponieważ to właśnie ta klasa zajmuje się konwertowaniem polskich złotych na inne waluty.

2.Creator

W naszym kodzie klasa User tworzy konta dla uzytkownika, ponieważ to walsnie ta klasa używa obiektów klasy Account

3.Controller

W naszym przypadku kontrolerem jest klasa User

4.Low coupling

Kod jest tak podzielony na klasy w taki sposób aby był łatwy do debugowania i łatwy do zrozumienia.

5.High cohesion

Wszyskie klasy stworzone są do pojedyńczych zadań i nie wykonują za dużo zadań.

6.