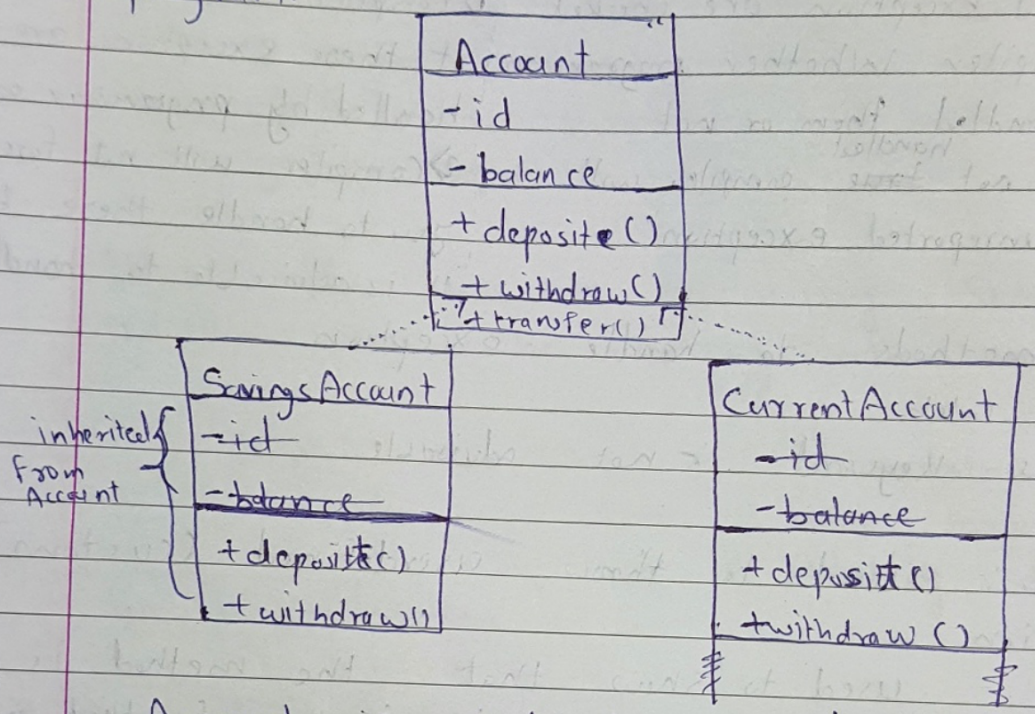
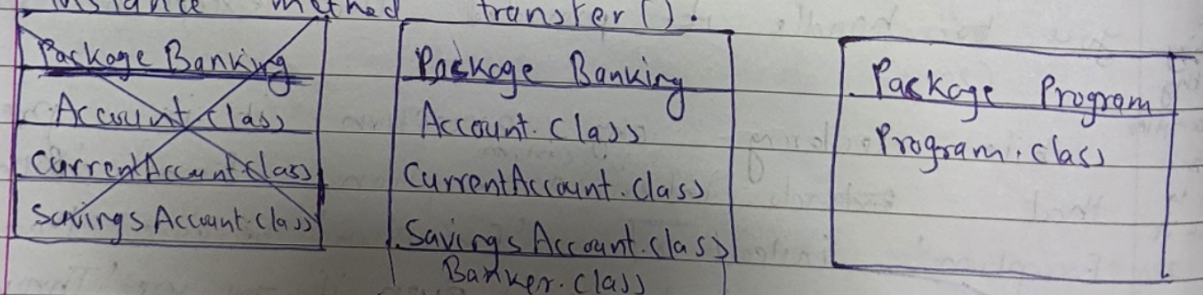
In try..catch & finally after execution of try block if exception is being caught control goes to finally block first to execute cleanup code (free resources) and the executes catch block.
∴ always write cleanp code in finally block.
Cleanup code can automatically execute by finillize() method in java by garbage collector

2) Try to maintain heirarchy while using catch block

3) Finally block can contain clean up code. So if catch block also have exception then still finally will get execute and terminates program.



Account is abstract class having abstract method deposite() and withdraw() and have instance method transfer().



1) Account class

fields:- we are making id default and balance protected balance is protected because if in future we want some other class to be inherited from Account and if that class is in

~~Same~~ ~~package~~ that class is not in Same package as in account that's why we are declaring ~~do~~ balance as protected. (think futuristic).

Methods :- (i) public double balance() {
return balance; // one linere method
}

(ii) public long id() {
return id; // one liner method
}

(iii) public final void transfer (double amount, Account that) throws Insufficent Funds Exception {

Concept :- ~~on~~final method :- when you know logic is not going to change ~~and~~ that's why we are not going to override it in future.

If we declare this method as final compiler will not do dynamic binding for it and ~~as~~ we know dynamic binding is 10 times ~~solwe~~ slower than static binding to make performance faster use final wherever needed.

One liners methods :- In java, compiler sees if methods ~~have~~ only one line then it replaces that one line with call. this is called inlining and this also make performance faster.

∴ if we ~~use~~ write public final double balance() {
return balance;
}

due to the final its also increase performance and as it is one liner compiler will use inlining which again increase performance (code optimizat^n)

~~Same~~ ~~package~~ that class is not in same package as in account that's why we are declaring ~~do~~ balance as protected. (think futuristic).

Methods :- (i) public double balance() {
return balance; // one liner method
}

(ii) public long id() {
return id; // one liner method
}

(iii) public final void transfer (double amount, Account that) throws InsufficentFundsException {

Concept :- ~~on~~ final method :- when you know logic is not going to change ~~and~~ that's why we are not going to override it in future.

If we declare this method as final compiler will not do dynamic binding for it and as we know dynamic binding is 10 times ~~solwe~~ slower than static binding to make performance faster use final wherever needed.

One liners methods :- In java, compiler sees if methods ~~have~~ only one line then it replaces that one line with call. this is called inlining and this also make performance faster.

∴ if we ~~use~~ write public final double balance() {
return balance;
}

due to the final its also increase performance and as it is one liner compiler will use inlining which again increase performance (code optimizat^n)

If your class is final then it is not allowed to inherit them.
and if you make class as final all methods will be automatically final cuz you will be not allowed to inherit class and override those method.

If field is final :- Right side of the field cannot contain another thing ~~once it is declared~~ later in program once it is declared to be final at the start of program. ~~fields~~ final fields are written in ~~Caps~~ capital letters.

* Fields can be
Static :- if you want to share field with all instances (object) (saves memory)
final :- ~~you~~ if you only need to make things unchangable
static final :- if you wantsto share field with all instances and also make it unchangable then use static final

If you want to intiallize static variable use static block
eg.
```
static {
    st = 1000;
}
```

Static initiallizer is executed when class is loaded in JVM (means after we run java program).

~~ass~~ as currentAccount and SavingsAccount classes will not be ~~ove~~ inherited that's why we make it Final.

* Dependancy Inversion Principle (DIP)

SavingsAccount() and currentAccount() are the classes which are not ~~to be~~ meant to be shown to the outer (other programmer) world because people might do changes or directly access it. which means object of SavingsAccount() and currentAccount() should not be created outside of that package. and that object should only be consumed by Account (abstract Type) when we said that object cannot be created directly thats why we implemented one more ~~method~~ class (Banker) in same Banking package to create object of currentAccount and savingAccount and from that class we are accessing objects of these both class in program.java.

There 23 other methods are there to make DIP satisfy, factory method is one of them.

Conclusion:- Saving and current account Kochuparam was dip

Creating Banking.java in same package is called Factory method

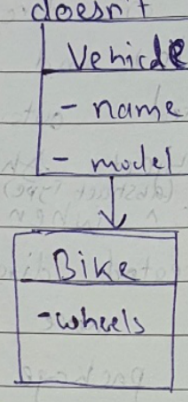
31/12/22

Now we need to add interest in savings account only currentaccount doesn't need interest so if we add interest in ~~current~~ Savings Account only but still we'll have to access it with Account's object ~~and if we add~~ that interest cuz we are hiding it from ~~pac~~ other's then if we put interest() in Account then that will be violation of SRP (Single Responsibility principle) cuz current Account doesn't need interest().

So now we'll create ~~whole~~ another class for interest() called package. but since Saving Account is ~~not~~ already inherit-ing Account it can't inherit profitable too cuz java doesn't support multiple inheritance.

∴ we well make profitable as interface not as class.

* Why java doesn't support multiple inheritance



C++

Bike b = new Bike()

name	0
model	4
wheel	8

b

b.wheel can be access by adding 4 (size of datatype) in 4 cuz pointers are arithmatic in C++. that's why pointers are not safe to use

java

Bike b = new Bike()

name	0
model	4
wheel	8

b

b.wheel cannot be access cuz in java 0, 4 & 8 are identities and they are non-arithmatic means we cannot add them.

that's why inheritance is not allowed <u>using classes</u>

Interface Seggregation :- fields are not common only methods are common.
always prefer interface over asstract class because it provides multiple inheritance.

with interfaces we can have multiple inheritance
bcuz interfaces doesn't have instance fields
~~Iterf~~ interface can only have Static Final fields

Abstract Class	Interface
1) cannot be instantiated but can define instance field	cannot define instance field. cannot be instantiated.
2) methods are bydefault not abstract can have instance method	Methods are bydefault abstract and instance methods has to be written with default modifier. (supported after java 8)
3) can contain public as well as non public members	can only contain public members.
4) can support defination of constructor called by its subclass	doesn't support defination of constructor.
5) can contain all types of fields	can only contain fields which are static final
6) class A extends abstract B	interface A ~~extends~~ implements interface B, C
7) Abstract class A extend class B	interface A, B implement class C
8) ~~only methods are common fields are not common~~	
8) Fields are common & methods are common	fields are not common only methods are common.

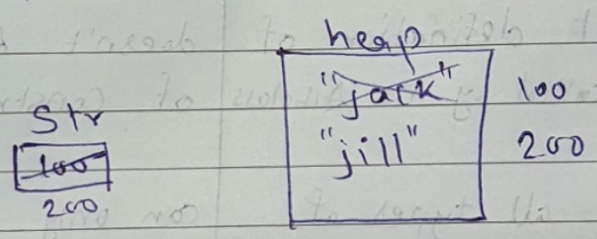
* Strings are immutable in java

```
Str = "jack"
Str.uppercase();
sop( str);
o/p:- jack
```

Here "jack" is stored in Str but after making it uppercase() Str will still have "jack" because you cannot change strings they are immutable, if you want o/p as "JACK" write this

```
str = Str.uppercase();
sop( str);
Str = "jack"
Str = "jill"
```

this is allowed cuz we are changing instance here. (reference to str) means



now Str will point to 200 which has "jill" as instance. but doing str.uppercase() is like we are telling compiler write "JACK" at position 100 which is not possible cuz strings are immutable you have to make another object Str1 to have strings in uppercase, or we can write str = Str.uppercase()

Garbage collector doesn't free your resources used by database, networks or files you have to explicitely mention there. This can be done in by ~~Finalize~~ finally block.

* Garbage Collection.

Java have memory allocation in two ways stack & heap

~~Stack is like handbag you need something you can directly take out~~

Heap is like cupboard all ~~objects~~ instance are stored in heap. GC checks if any instance doesn't have any object referring to it those object get removed out of heap by garbage collector in java it make new space for new objects. this is called "mark and sweep". just like your cupboard you throw out old objects and make space for new ones.

Stack is like handbag whatever you need from your cupboard you put it in handbag & leave ∴ In stack objects (references to all instance) get stored.

stack gets allocated and deallocated when method execution ends.

* Nested class

nested class is used when you are not gonna use ~~object of~~ that class outside of your outer class.

Nested class are always **static class**

* ~~Autoct~~ If we use Resources like DB, networking or File Garbage collector doesn't have access to close them only Kernel have that's why we have to mention code to close all resources explicitely.

we can use <u>try-with-resources</u> to close all your resources it keeps all your exception prone lines in single block.

try-with-resource is method defined in <u>autocloseable interface</u>

* Code which safely work with all data types is called generic programming