**Dining Philosophers Problem**

**Money Transaction System**

**1. Overall Description**

When a client performs a transfer, they should choose the account they want to send the selected amount of money they want to transfer. If another client wants to send money to any of those accounts, they must wait until the other accounts have completed the transaction in order for the transfer to occur in the correct order without being intercepted.

**1.1 Assumptions and Dependencies**

this system attempts to address issues like deadlock and starvation. Implementation By using JavaFX as our GUI and using OOP principles for more organized and clean code with good and clean graphics and animation.

**2. Pseudocode**

start

declare public class Monitor

define a set of attributes

define a public constructor and initialize attributes

void test

if state.i=hungry and state.left(i) != eating and state.right(i) != eating then

state.i =state.eating

signal(i)

end if

void pickUp(int i)

state(i)=state.hungry

test(i)

if state(i) != state.eating

eat(i)

end if

void eat(int i)

void putDown(int i)

think(i)

right = (i+n-1)%n //n=number of philosopher

left = (i+1)%n

test(right)

test(left)

void think(int i)

states philostate(int i)

return state(i)

end

start

declare public class Philosopher

define a set of attributes

define a public constructor and initialize attributes

void run

count =1

while count< timesToEat

monitor.pickUp(id)

print “philosopher (id value) eating (count value) “

thread.sleep(sleepTime)

end while

**3. Deadlock**

**3.1 Example**

The Deadlock issue appears when two customers start a transaction and they never get out from the critical section so that other customer can transfer to.

**3.2 Solution**

We solve this problem by using **condition.await()** and **signalAll** from the code

**4. Starvation**

**4.1 Example**

Starvation problem appears when a large number of customers transfer to the same account or having to wait too long for other customers to finish their transaction first so you can make your own transaction.

**4.2 Solution**

We can solve this problem by setting an time interval so that after exceeding that time it shut down the transfer operation and ask the customer to try again later. **By adding boolean true to the ReentrantLock**