Joel Veneracion
Jrv5247@psu.edu
936314482

Luxin Wang
Lxw5332@psu.edu
919540572

Robin Leckey
Rkl5161@psu.edu
956436389

Bank Script Summary:

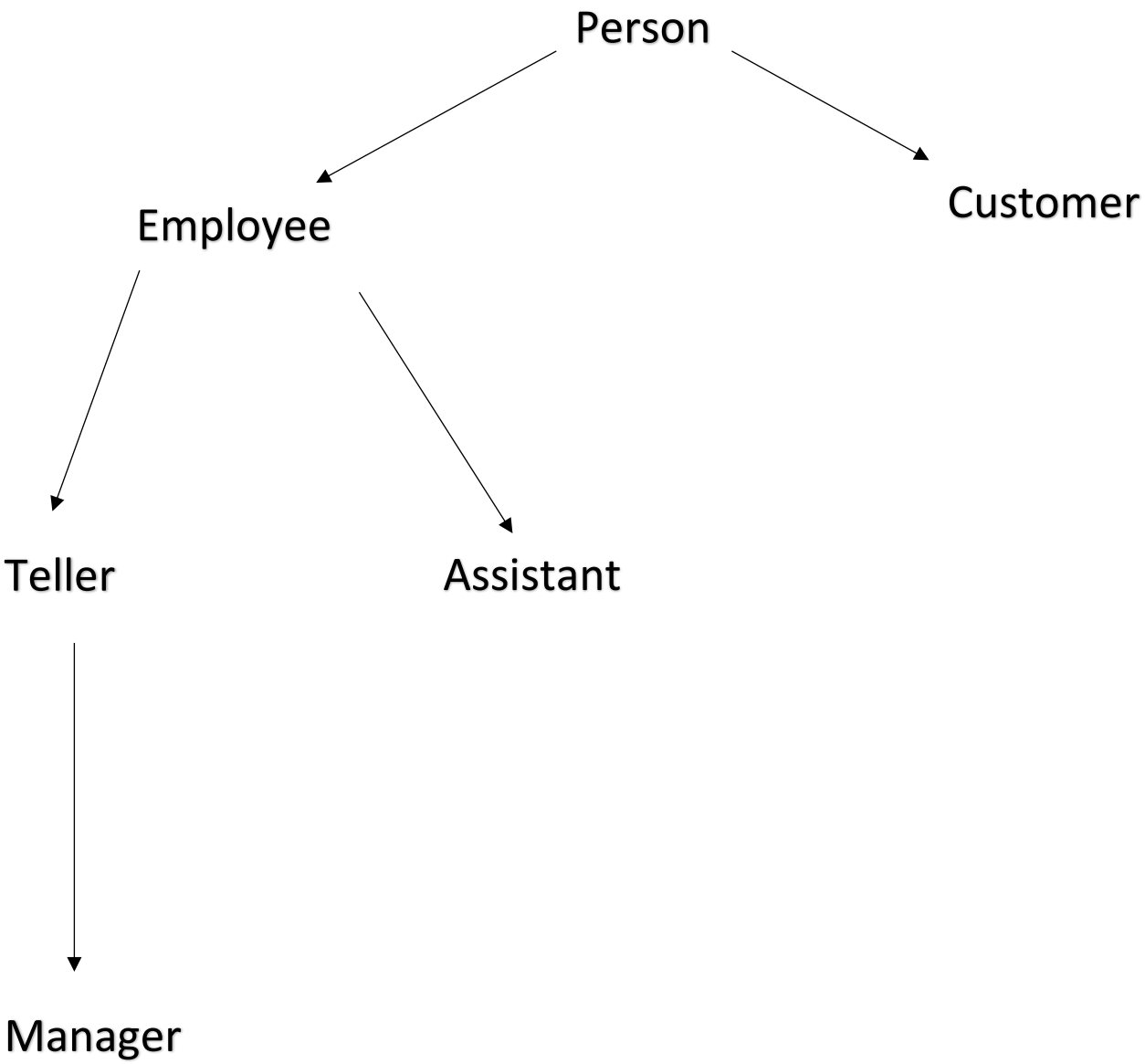
The bank.py script has 6 classes: Person, Customer, Employee, Assistant, Teller, and Manager. The Person class serves as a template for the other 5 classes. The Employee and Customer classes are the subclasses of the Person class, and the Assistant and Teller classes inherit from the Employee class. Finally, the Manager class inherits from the teller class.

The Person has the basic information of any given human being, including name, birthday, and calculated age. The Employee class is the most generic type of class for the employees in the bank. It only has access to the number of customers enrolled in the bank and can greet; it does not have permissions to withdraw or deposit for the customers.

The customer class contains the account information for a specific customer, including amount of balance and amount of remaining loans. If the customer is properly enrolled in the bank, the customer can withdraw, deposit, borrow loans, make loan payments, and check his or her bank statements (with options to get the raw string or get the formatted statement table). However, if a fake account object is created, no operations can be done with that account.

The teller can access the list of enrolled customers and withdraw and deposit for any customer who makes a request at the bank. The bank assistant does the same thing as the teller but does so with a fee that is taken from the customer's balance. In addition, for each customer that makes a withdrawal or deposit on his or her birthday, the assistant will remind that customer that it is his or her birthday. Finally, the manager has the same operations as the teller in addition to having special permissions to add accounts to and delete from the list of enrolled bank customers. Also, every class that inherits from Employee has a unique greeting statement.

Hierarchical Chart of Code



```python
from datetime import date
from time import ctime
class Person:
    """Parent class for Employee and Customer class"""
    def __init__(self, name, birthdate):

        """
        initializer for Person class
        It assign the name and birthdate to the self
        re-format the birthday input so that it can be test for is_birthday function later;
        If the input birthday format is in correct, an ValueError rise.
        ------------

        Parameters
        ----------------------
        name: str
        birthday: str (in the format MM/DD/YYYY)

        """

        self.__name = name
        self.__birthdate = birthdate
        try:
            month, day, year = self.__birthdate.split('/')
            month, day, year = int(month), int(day), int(year)
            if year < 1000:
                print("Wrong format of year")
                raise Exception('')
            elif year > date.today().year:
                print("Year is in the future.")
                raise Exception('')
        except Exception:
            raise ValueError("Correct format of date: MM/DD/YYYY")

    def get_name(self):
        """Function to return the name of an object"""

        return self.__name

    def is_birthday(self):
        """Function testing is the birthday input for a
        specific object match today's date"""

        current_date = date.today()
        month, day, year = self.__birthdate.split('/')
        return (int(month), int(day))==(current_date.month, current_date.day)

    def age(self):
        """
        Function calculate the age of an object
        This function accept the information of an object, and use the daytime function to get today's date
        then split month, day, and year; so that each number can be compared with the birthday of the object
        -------------------------
        return
        age: int
        """

        current_date = date.today()
        month, day, year = self.__birthdate.split('/')
        month, day, year = int(month), int(day), int(year)
        age = current_date.year - year
        if (month, day) > (current_date.month, current_date.day):
            age-=1
        return age
```

```
65
66
67  class Employee(Person):
68      """
69      Employee class is is a child class of its parent, the Person class.
70      Employee inherits attributes from the Person class
71
72      """
73
74
75      customers={}
76      def __init__(self, name, birthdate):
77          """Initializer for the Employee class
78
79          Summary lines:
80          Inherits the name and birthday variables from Person class.
81          ---------
82          Parameters:
83          name: str
84          birthdate: str
85          amount: float
86
87          """
88          Person.__init__(self, name, birthdate)
89
90      def num_of_customers():
91      """
92          Function to add money to one certain customer's bank account
93
94          Function that returns the number of customers
95          -----------------------------------
96          return:
97          Employee.customers : int
98
99
100         """
101         return len(Employee.customers)
102
103     def is_customer(name):
104         """function returns a boolean variable is the object a bank customers"""
105         return name in Employee.customers
106
107     def withdraw(self, name, amount):
108         """
109         Function that returns string, "Permission denied"
110         when Employee attempts to withdraw money
111         """
112
113         raise Exception("Permission denied")
114
115     def deposit(self, name, amount):
116         """
117         Function that returns string, "Permission denied"
118         when Employee attempts to deposit money
119         """
120         raise Exception("Permission denied")
121
122     def greetings(self):
123         """
124         This function prints the name of the Employee and greets the customer as a string
125         """
126         print("Hello, my name is {}".format(self.get_name()))
127
128
129 class Customer(Person):
130     """
131     The children class of the Person Class, inherit the name and birthday
```

```
132         variables from the Person Class
133         """
134 
135         def __init__(self, name, birthdate, balance):
136             """
137             Initializer for Customer class
138             inherit the name and birthday variables from Person class
139             create private variables called account, loan(initial value is set to 0),
140             account_changes(empty dictionary), and loan_changes(empty dictionary)
141             -----------
142 
143             Parameters:
144             name: str
145             birthday: str
146             account:float
147             balance:float
148             loans:float
149             account_changes:dict
150             loan_changes:dict
151 
152             """
153 
154             Person.__init__(self, name, birthdate)
155             self.__account=balance
156             self.__loans=0
157             self.__account_changes={}
158             self.__loan_changes={}
159 
160         def __check_bank_status(self):
161             """
162             Private function check the where a customer account is available
163             in the bank or not
164 
165             create a boolean variable called inbank, if the customer is deleted
166             by the manager, inbank will be assigned False
167             This function will be used later to prevent any further operation
168             by customers if his/her bank account is deleted
169 
170             """
171             inbank = Employee.is_customer(self.get_name())
172             if not inbank:
173                 raise Exception("{} is not in bank".format(self.get_name()))
174 
175         def bank_statement(self):
176             """
177             Function return a table of bank statement for a certain
178             customer object, including loans payment and account changes
179 
180             This function first check the bank status of the customer using
                __check_bank_status,
181             it will only execute if the __check_bank_status doesn't return an exception.
182             Tables titles are called "Account Changes" and "Loan Changes";
183             if no loans are lent or no changes are made, table content will be "None"
184             Once payment or loans are processed,
185             table content will show the amount of transaction and date with time.
186             **Note: 10/10 recommend to use print function for output---
187             PLEASE use print
188             ---------------------
189 
190             Parameters:
191 
192             statement_str:str
193             change: float
194             symbol: str
195             --------------------------
196             return:
197             statement_str: str (this will be a string format into a table)
```

```
198
199             """
200
201             self.__check_bank_status()
202             statement_str = "Account Changes\n"
203             if len(self.__account_changes)==0:
204                 statement_str += 'None\n'
205             for date in self.__account_changes:
206                 change = self.__account_changes[date]
207                 symbol = ''
208                 if (change >= 0):
209                     symbol = '+'
210                 else:
211                     symbol = '-'
212                 statement_str += "{}: {}${}\n".format(date, symbol, abs(change))
213             statement_str += "Loan Changes\n"
214             if len(self.__loan_changes)==0:
215                 statement_str += 'None\n'
216             for date in self.__loan_changes:
217                 change = self.__loan_changes[date]
218                 symbol = ''
219                 if (change >= 0):
220                     symbol = '+'
221                 else:
222                     symbol = '-'
223                 statement_str += "{}: {}${}\n".format(date, symbol, abs(change))
224             return statement_str
225
226         def print_statement(self):
227             """
228             Prints the bank statement onto the console
229             """
230             print(self.bank_statement())
231
232         def make_payment(self, payment):
233             """
234             Function transfer money from one specific customer's bank account to his/her
                loan payment
235             This function first check the bank status of the customer using
                __check_bank_status,
236             it will only execute if the __check_bank_status doesn't return an exception.
237             If the payment is great than the total loan amount,
238             the function will reject the payment and lets the user re-enter a plausible
                payment
239             If there is nothing left in the customer's bank account,
240             it is unable to make the payment, a message is printed.
241             When all conditioned met, the loan amount will be subtracted
242             by the payment amount, and the datetime will be assigned too.
243             --------------------------
244             Parameter:
245             payment: float
246             date: str
247             --------------------------
248             Return:
249
250             __loans: float (the remaining loan)
251
252             """
253
254             self.__check_bank_status()
255             if payment > self.__loans:
256                 print("Payment is greater than loan.  Readjusting payment.")
257                 payment = self.__loans
258             amount = self.withdraw(payment)
259             if amount <= 0:
260                 print("Unable to make loan payment")
261                 return
```

```
262             self.__loans = round(self.__loans - amount, 2)
263             date = ctime()
264             self.__loan_changes[date]=amount*-1
265             return self.__loans
266 
267         def borrow_loans(self, amount):
268             """
269             Function to let the customer borrow a loan and keep track of the loan amount
270 
271             This function first check the bank status of the customer using
                __check_bank_status,
272             it will only execute if the __check_bank_status doesn't return an exception.
273             When certain amount is entered for the loan, the loan will be added
274             by the amount and the current_date will be assigned for later tracking
275             ------------------------
276             Parameter:
277             amount: float
278             current_date:str
279             --------------------------------
280             Return:
281             __loans: float (the total amount of loan)
282             """
283             rounded_amount = round(amount, 2)
284             if rounded_amount <= 0:
285                 print("Invalid loan amount. Borrowing halted.")
286                 return self.__loans
287             self.__check_bank_status()
288             self.__loans = round(self.__loans + rounded_amount, 2)
289             current_date = ctime()
290             self.__loan_changes[current_date]= rounded_amount
291             self.deposit(rounded_amount)
292             return self.__loans
293 
294         def withdraw(self, amount):
295             """
296             Function to subtract money from one certain customer's bank account
297             This function first check the bank status of the customer using
                __check_bank_status,
298             it will only execute if the __check_bank_status doesn't return an exception.
299             Withdraws money from customer's account; Return the amount of remaining balance;
300             or message that says not enough to withdraw
301             ----------------------------------
302             Parameter:
303             amount: float
304             ----------------------------------
305             return:
306             amount: float (the amount of remain bank account)
307             """
308             self.__check_bank_status()
309             rounded_amount = round(amount, 2)
310             if (rounded_amount <= 0.00):
311                 print("Invalid withdrawal amount. Withdrawal halted.")
312                 return 0
313             elif (rounded_amount <= self.__account):
314                 self.__account = round(self.__account - rounded_amount, 2)
315                 self.__account_changes[ctime()] = rounded_amount * -1
316                 return rounded_amount
317             else:
318                 print("Withdrawal denied.  There is not enough money in the account")
319                 return 0
320 
321         def deposit(self, amount):
322             """
323             Function to add money to one certain customer's bank account
324             This function first check the bank status of the customer using
                __check_bank_status,
325             it will only execute if the __check_bank_status doesn't return an exception
```

```
326             deposit money to customer's account; Return the amount of remaining balance.
327             ----------------------------------
328             Parameter:
329             amount: float
330             ----------------------------------
331             return:
332             amount: float (the amount of remained bank account)
333             """
334             self.__check_bank_status()
335             rounded_amount = round(amount, 2)
336             if rounded_amount <= 0.00:
337                 print("Invalid deposit amount. Deposit halted.")
338                 return self.__account
339             self.__account = round(self.__account + rounded_amount, 2)
340             self.__account_changes[ctime()] = rounded_amount
341             return self.__account
342
343         def get_balance(self):
344             """Function to get total amount of one certain customer's bank balance
345             (This function first check the bank status of the customer using
                __check_bank_status,
346             it will only execute if the __check_bank_status doesn't return an exception)"""
347
348             self.__check_bank_status()
349             return self.__account
350
351         def get_loans(self):
352             """Function to get total amount of one certain customer's loan balance
353             (This function first check the bank status of the customer using
                __check_bank_status,
354             it will only execute if the __check_bank_status doesn't return an exception)"""
355
356             self.__check_bank_status()
357             return self.__loans
358
359     class Teller(Employee):
360         """
361         This class handles the regular bank teller's operations.
362         The bank teller has more access
363         than a generic Employee as he or she
364         is able to make withdrawals and deposits that
365         visiting customers request.  The bank teller can gain access
366         to the accounts but cannot create or delete them.
367         """
368         def __init__(self, name, birthdate):
369             """
370             Initializer for the Teller class
371
372             ---------------------------------
373             Parameters
374
375             name(str): the teller's full name
376             birthdate(str): the teller's birth date in MM/DD/YYYY format
377
378             """
379             Employee.__init__(self, name, birthdate)
380
381         def withdraw(self, name, amount):
382             """
383             Withdraws a given amount of money to a visiting customer from the customer's
                account
384
385             This method checks if the customer
386             has an account under his or her name before withdrawing.
387             If the customer has an account, the teller's withdraw method
388             calls the withdraw method of the customer account object
389             and returns the amount of money withdrawn.
```

```
        Otherwise, indicate that the customer is not in the bank,
        and return the amount as 0.

        ----------------------------------------------------
        Parameters

        name(str): the name string of the customer to withdraw money for
        amount(float/int): the amount of money to withdraw

        ----------------------------------------------------
        Return

        float: The amount of cash the customer receives from the withdrawal request.

        """
        if not Employee.is_customer(name):
            print("{} is not enrolled in the bank.".format(name))
            return 0.0
        return Employee.customers[name].withdraw(amount)

    def deposit(self, name, amount):
        """
        Deposits a given amount of money into an existing customer's bank account

        The method checks if the customer has an existing
        account under his name in the bank.  If so, the amount
        of money will be added to the customer's bank account
        and the new balance will be returned.  Otherwise,
        indicate that the customer does not have an account
        and return a value of 0 as the balance.

        ----------------------------------------------------
        Parameters

        name(str): the name string of the customer to deposit money for
        amount(int/float): the amount of money to deposit into the account

        ----------------------------------------------------
        Return

        float: The new amount of the customer's account balance.

        """
        if not Employee.is_customer(name):
            print("{} is not enrolled in the bank.".format(name))
            return 0.0
        customer_obj = Employee.customers[name]
        return customer_obj.deposit(amount)

    def greetings(self):
        """
        Greets and by introducing his/her name and lets customer
        know that he/she is the teller.
        """
        print("Hi, I'm {}. I will be your teller today".format(self.get_name()))

class Assistant(Employee):
    """
    Assistant is a child class of its parent, the Employee.
    Assistant inherits attributes from the Employee and Person class
    """

    def __init__(self, name, birthdate, interest_rate=0.02):
        """
        Function to add money to one certain customer's bank account
        ------------------------------------
        Parameter:
```

```
457             amount: float
458             ----------------------------------
459             """
460             Employee.__init__(self, name, birthdate)
461             self.__interest_rate = interest_rate
462 
463 
464         def remind_birthday(self, customer):
465             """
466             Prints the reminder string if the day the customer deposited/withdraw the money
                on
467             his/her birthday
468             """
469 
470             if (customer.is_birthday()):
471                 print("Oh, by the way, it's your birthday")
472 
473         def withdraw(self, name, amount):
474             """
475             This function will had an interest rate when
476             the customer takes money out of his/her bank account.
477             If the customer takes money out
478             of his/her account on their birthday,
479             then a message will greet them happy birthday.
480             ----------------------------------
481             Parameter:
482             Name: str
483             Amount: float
484             ----------------------------------
485             Return:
486             The total amount of money being subtracted from the balance: float
487 
488             """
489             if not Employee.is_customer(name):
490                 print("{} is not in the bank system.".format(name))
491                 return 0.0
492             customer=Employee.customers[name]
493             percent_interest = int(self.__interest_rate * 100)
494             print("Assistant fee: {}%".format(percent_interest))
495             self.remind_birthday(customer)
496             interest = amount * self.__interest_rate
497             net_withdrawal = customer.withdraw(amount*(1+self.__interest_rate)) - interest
498             if net_withdrawal <= 0.00:
499                 return 0
500             return round(net_withdrawal, 2)
501 
502         def deposit(self, name, amount):
503             """
504             This function will add money to his/her bank account.
505             ----------------------------------
506             Parameter:
507             Name: str
508             Amount: float
509             ----------------------------------
510             Return:
511             The total amount of money after adding to the balance: float
512 
513             """
514             if not Employee.is_customer(name):
515                 print("{} is not in the bank system.".format(name))
516                 return 0.0
517             customer_obj = Employee.customers[name]
518             fee = self.__interest_rate * amount
519             return customer_obj.deposit(amount - fee)
520 
521         def greetings(self):
522             """Function that returns the name of the Assistant and asks if the customer
```

```
523             needs assistance"""
524
525             print("Hi, I'm {}. How may I assist you?".format(self.get_name()))
526
527     class Manager(Teller):
528         """
529         This class handles the bank manager's operations.
530         The bank manager has the same exact operations
531         as a bank teller, but in addition, the manager has
532         special permissions to explicitly create, delete,
533         and give customers their bank accounts.
534         """
535         def __init__(self, name, birthdate):
536             """
537             Initializer for the Manager class
538
539             --------------------------------
540             Parameters
541
542             name(str): the manager's full name
543             birthdate(str): the manager's birth date in MM/DD/YYYY format
544
545             """
546             Teller.__init__(self, name, birthdate)
547
548         def delete_account(self, name):
549             """
550             Deletes an existing bank account from the bank system
551
552             When a customer's full name is given and
553             it matches one of the names in the dictionary
554             of bank accounts, the bank account under
555             that name will be deleted.  If there is no
556             bank account under that name,
557             the manager will be informed that the
558             bank account does not exist, and the delete
559             operation stops.
560
561             ---------------------------------------
562             Parameters
563
564             name(str): name of the customer
565                 whose account is to be deleted
566
567             """
568             if not Employee.is_customer(name):
569                 print("{} is not in the bank system.".format(name))
570             else:
571                 del Employee.customers[name]
572
573         def add_account(self, name, birthdate, balance=100):
574             """
575             Creates a new bank account for a new, eligible customer.
576
577             If the customer is at age or over 16 yrs. old
578             and does not have an existing account in the bank,
579             a new account with a given starting balance or
580             default balance of $100, either of which
581             the customer pays to open up the account.
582             If the customer's age is underage or is
583             already a customer in the bank, the system will
584             inform the manager of either case and
585             stop creating the new account,
586             as it will overwrite the existing account if created.
587
588             ---------------------------------------
589             Parameters
```

```
590
591             name(str): name of the customer whose account is to be opened
592             birthdate(str): the birth date of the customer in string formatted in MM/DD/YYYY
593             balance(optional int/float): the starting balance of the account; $100 if not
                provided
594
595             """
596             customer=Customer(name, birthdate, balance)
597             if (customer.age() < 16):
598                 print("{} is too young to have an account.".format(name))
599             elif Employee.is_customer(name):
600                 print("{} is already in the bank system.  Cannot add account".format(name))
601             else:
602                 Employee.customers[name]=customer
603
604         def get_account(self, name):
605             """Obtains the customer account under a given name"""
606             if not Employee.is_customer(name):
607                 print("{} is not a customer of this bank.".format(name))
608                 return None
609             return Employee.customers[name]
610
611         def greetings(self):
612             """
613             Greets the customer by introducing his or her
614             name and informs them that he or she is a manager.
615             """
616             print("Hi, I'm {}, the bank's manager.".format(self.get_name()))
```