INTPROG TB2: OBJECT ORIENTED PROGRAMMING

Practical Worksheet 2: Introduction to Inheritance

1. Introduction

This practical session will be split into two sections: one which focusses on using objects, and another to introduce inheritance. Work through this worksheet at your own pace, and be sure to ask questions to clarify any concepts you find challenging. Make sure that you fully test all of your classes and programs before moving on to the next question. Please feel free to experiment until you are confident you understand what is happening and why.

2. Using Objects

In the first part of this practical worksheet, we will be revising the contents of the second lecture from last week. To start with, we will be writing a main function which can test the Circle class file we created last week. Then, in the second example, we will be creating a cash machine file, which will use the BankAccount class to support its operations.

2.1. Example 1 – Circle.py

In the previous practical worksheet, you created a Circle class, which was then tested using the shell. In this example, rather than testing all the methods individually in the shell, we will create a main function within another file. This will allow us to easily test that the Circle.py class is functioning as anticipated. You will need to make sure that the Circle.py file is contained within the same directory as the file you will create in this practical.

Create a new file called UsingCircle.py. Start the file with the following import statement to ensure the Circle class is known to the interpreter.

```
from Circle import Circle
```

We will then create our main() function. Within the main() function, we can then create two objects using the Circle class. For the purposes of this example, we will call these circle1 and circle2.

```
def main():
      circle1 = Circle(4)
      circle2 = Circle(10)
      print("Circle 1")
      print()
      print("Object Definition: ",circle1)
      print()
      print("Radius: ",circle1.radius)
      print()
      print("Area: ",circle1.calculateArea())
      print("Circumference: ", circle1.calculateCircumference())
      print()
      print("Circle Information:\n",circle1.retrieveInformation())
      print()
      print()
      print("Circle 2")
      print()
```

```
      print("Object Definition: ",circle2)
      print()
      print("Radius: ",circle2.radius)
      print()
      print("Area: ",circle2.calculateArea())
      print("Circumference: ", circle2.calculateCircumference())
      print()
      print("Circle Information:\n",circle2.retrieveInformation())
```

Finally, we call the main function

```
main()
```

The completed file should be as follows:

```
from Circle import Circle

def main():
      circle1 = Circle(4)
      circle2 = Circle(10)

      print("Circle 1")
      print()
      print("Object Definition: ",circle1)
      print()
      print("Radius: ",circle1.radius)
      print()
      print("Area: ",circle1.calculateArea())
      print("Circumference: ", circle1.calculateCircumference())
      print()
      print("Circle Information:\n",circle1.retrieveInformation())
      print()

      print()
      print("Circle 2")
      print()
      print("Object Definition: ",circle2)
      print()
      print("Radius: ",circle2.radius)
      print()
      print("Area: ",circle2.calculateArea())
      print("Circumference: ", circle2.calculateCircumference())
      print()
      print("Circle Information:\n",circle2.retrieveInformation())

main()
```

Run the UsingCircle.py file as a script. This should run the test file, giving the following output:

```
Python 3.4.3 (v3.4.3:9b73f1c3e601, Feb 24 2015, 22:43:06) on Windows (
32 bits).
This is the IEP interpreter with integrated event loop for TK.
Type 'help' for help, type '?' for a list of *magic* commands.
Running script: "N:\MyDocuments\Teaching\INTPROG 2017-18\TB2\Wk2\Pract
ical\Example Code\UsingCircle.py"
Circle 1

Object Definition:  <Circle.Circle object at 0x02CA8D10>

Radius:  4

Area:  50.26548245743669
Circumference:  25.132741228718345

Circle Information:
 A circle of radius 4.0000
has an area of 50.2655
and circumference of 25.1327


Circle 2

Object Definition:  <Circle.Circle object at 0x02CA8CB0>

Radius:  10

Area:  314.1592653589793
Circumference:  62.83185307179586

Circle Information:
 A circle of radius 10.0000
has an area of 314.1593
and circumference of 62.8319
```

2.2. Example 2 – CashMachine.py

This example aim to demonstrate how classes can be used to support the operations of other functions or classes. We will be creating a function which will model a cash machine, using the BankAccount class from Worksheet 1.

Users of the cash machine will need to enter their bank account number, before they can carry out any changes to their account. The cash machine we are modelling will provide users the following functionality: deposit money, withdraw money, and get a balance.

Start by creating a new file called CashMachine.py.

Firstly, we need to import the BankAccount file. Again, make sure that the BankAccount.py file is stored in the same place as CashMachine.py.

```
from BankAccount import BankAccount
```

For this example, we will be creating a function called runCashMachine.

```
def runCashMachine():
```

Firstly, we need to create the bank accounts. For this example, we will use three. We also put it in a list to make it easier to manage them later on.

```
account1 = BankAccount("Alice Jones", "12345678", 1000)
account2 = BankAccount("Bob Smith", "23456789", 0)
account3 = BankAccount("Charlie Williams", "34567890", 50)

accountsList = [account1, account2, account3]
```

To keep the cash machine open, we need to use a while loop:

```
open = True

while open:
```

The first thing which needs to be done is to get the account number from the user, and check whether it is valid.

```
currentAccount = "" # variable to store the account number of current user
enteredAccountNo = input("Please enter your account number: ")
for account in accountsList: # loop through all the valid accounts
      if account.accountNumber == enteredAccountNo: # check whether account number exists
            # if account is valid, then assign the number to currentAccount and say hello
            currentAccount = account
            print("Hello", currentAccount.accountName)
            break # end the for loop as valid account has been found
```

We then need an 'if' statement which allows the user to carry out operation on their account when they have input a valid account. In the case of the user putting an invalid account number into the system, they need to get an error message:

```
if currentAccount != "":
      # carry out operations – the rest of the program will be entered here
else:
      print("Invalid account number, please try again")
```

Within the 'if' statement, we now need to code all the operations which needs to be available to the user. However, firstly they need to state what they want to do.

```
accountAction = input("\nWhat would you like to do? \n (W = withdraw, D = deposit, B =
balance, F = finished) \n")
print()
```

Then, we need to code all the operations. If the user enters D, they need to be able to deposit money. If W is entered, they want to withdraw money. When withdrawing money, we need to check whether withdrawal was possible. If it is not possible, an error message needs to be shown. Otherwise, the current balance should be shown. In the case where the user has an overdraft, the overdraft and amount available should be shown too. This will continue to loop until the user enters F to finish their interaction.

```
while accountAction.upper() != "F": # run whilst the user hasn't chosen to finish

      if accountAction.upper() == "D": # deposit money
            depositAmount = eval(input("Enter how much you would like to deposit: "))
            currentAccount.deposit(depositAmount)
            print("You have deposited", depositAmount, "in your account")
            print("You now have", currentAccount.amount, "in your account")

      elif accountAction.upper() == "W": # withdraw money
            withdrawAmount = eval(input("Enter how much you would like to withdraw: "))
            withdrawn = currentAccount.withdraw(withdrawAmount)
            if withdrawn: # money can be withdrawn
                  print("You have withdrawn", withdrawAmount)
                  print("You now have", currentAccount.amount, " in your account")
            else: # money can't be withdrawn
                  print("Sorry,", currentAccount.accountName, "you have insufficient funds
                  for the requested transaction")

      else: # print the account balance
            print("Your balance is:", currentAccount.amount)
            if currentAccount.accountOverdraft != 0:
                  print("Your overdraft is:", currentAccount.accountOverdraft)
                  print("You have", currentAccount.amount +
                  currentAccount.accountOverdraft, "available")

      accountAction = input("\nWhat would you like to do? \n (W = withdraw, D = deposit, B
      = balance, F = finished) \n")
      print()
```

Once the user has finished their operations, they are then asked whether the cash machine should stay open or close. If it stays open, they are then asked to enter their account number. When N is entered to close the cash machine, the program will finish.

```
closeCashMachine = input("Keep cash machine open? (Y or N)\n")
if closeCashMachine.upper() == "N":
      open = False
```

The whole file should look like this:

```
from BankAccount import BankAccount

def runCashMachine():

	account1 = BankAccount("Alice Jones", "12345678", 1000)
	account2 = BankAccount("Bob Smith", "23456789", 0)
	account3 = BankAccount("Charlie Williams", "34567890", 50)
	accountsList = [account1, account2, account3]
	open = True
	while open:
		currentAccount = ""
		enteredAccountNo = input("Please enter your account number: ")
		for account in accountsList:
			if account.accountNumber == enteredAccountNo:
				currentAccount = account
				print("Hello", currentAccount.accountName)
				break
		if currentAccount != "":
			accountAction = input("\nWhat would you like to do? \n (W = withdraw, D
			= deposit, B = balance, F = finished) \n")
			print()
			while accountAction.upper() != "F":
				if accountAction.upper() == "D":
					depositAmount = eval(input("Enter how much you would like
					to deposit: "))
					currentAccount.deposit(depositAmount)
					print("You have deposited", depositAmount, "in your
					account")
					print("You now have", currentAccount.amount, "in your
					account")
				elif accountAction.upper() == "W":
					withdrawAmount = eval(input("Enter how much you would like
					to withdraw: "))
					withdrawn = currentAccount.withdraw(withdrawAmount)
					if withdrawn:
						print("You have withdrawn", withdrawAmount)
						print("You now have", currentAccount.amount, " in
						your account")
					else:
						print("Sorry,", currentAccount.accountName, "you
						have insufficient funds for the requested
						transaction")
				else:
					print("Your balance is:", currentAccount.amount)
					if currentAccount.accountOverdraft != 0:
						print("Your overdraft is:",
						currentAccount.accountOverdraft)
						print("You have", currentAccount.amount +
						currentAccount.accountOverdraft, "available")
				accountAction = input("\nWhat would you like to do? \n (W =
				withdraw, D = deposit, B = balance, F = finished) \n")
				print()
			closeCashMachine = input("Keep cash machine open? (Y or N)\n")
			if closeCashMachine.upper() == "N":
				open = False
		else:
			print("Invalid account number, please try again")
```

Run the file as a script, and test all the possible options for input.

2.3. Practice Questions

1. Write test files (similar to the one created in example 1) for the following classes (created in worksheet 1). You only need to create and test one object, rather than two as shown in the example. Make sure you check all the instance variables have been initialised as anticipated and test all methods work correctly. If your methods include if statements, make sure you check each part is working correctly.
    a. Lecturer (worksheet 1, question 5)
    b. Student (worksheet 1, question 7)
    c. Borrower (worksheet 1, question 8)

2. Using a top-down approach, create a program which will use the Padlock class developed in the previous practical. The program will create a padlock with a random numerical combination, then give the user a fixed number of attempts to guess the code (this number is not given by the user, it should be coded into the program). Initially, the user should be asked the number of digits in the combination. This input is then used to generate a random number with the correct number of digits. For example, if the user enters 7, the combination will be between 1000000 and 9999999.
   Hint: You will need to test whether the padlock unlocks for each combination guess.

3. Write a Python file (UsingDice.py), which uses a throw of a dice to decide which shape to create, then print the information for that shape. You will need to use the Circle, Rectangle, and Dice classes from last week. You can either write your own Triangle class, or download the one from Moodle. You will need to use randint to create random numbers, between 1 and 10, for the required sizes (radius for circle, and sides for the triangle and rectangle).
   Hint: create the dice object using the following: dice = Dice(["Circle", "Circle", "Rectangle", "Rectangle", "Triangle", "Triangle"])

4. Create a program which will use a dice object to add random shapes (Rectangle, Hexagon, or Triangle) to a list. The class files for Hexagon and Triangle are available on Moodle. You will then need to print: the total area; the total perimeter; and the information for all the shapes in the list. The program will need to ask the user how many shapes they would like to create, and use a top-down approach.

5. Download the Book.py and Bookshop.py class files from Moodle. Using these 2 classes, create a file which will model a Popup Bookshop. You will need to use a top-down approach to ask the user how many days to model, and what books they require in their bookshop. The user should then be able to buy books (i.e. restock them), sell books, and close the bookshop (at the end of the simulated day).
   Hint: Go through both the provided class files and make sure you understand what it is doing before you start to answer this question.

3. Generalisation and Inheritance

Inheritance allows developers to implement generalisation. It provides a mechanism for developers to re-use similar code in other related classes. An example of this, as seen in the lectures, is the Polygon class. When the Rectangle, Triangle, and Hexagon class used the generalised Polygon, we were able to use the calculatePerimeter() method defined in the superclass, without needing any additional code.

In order to inherit from the superclass, we change the beginning of the class file to:

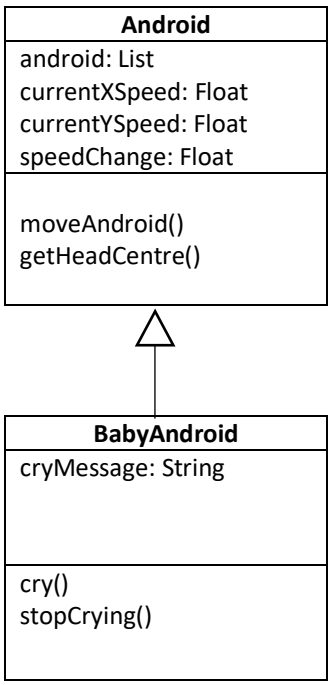
```
from Polygon import Polygon

class Rectangle(Polygon):
```

This tells the Python interpreter that the Rectangle class inherits all the instance variables and methods from the Polygon class.

3.1. Example 3 – Android Inheritance

The next example will use the Android class to develop a 'Baby' Android class. The Baby Android is able to do everything that a regular Android can do, but is also able to cry. The UML diagram looks as follows:



Before you can create the BabyAndroid class, download the Android class from Moodle. This has been adapted to take a scale factor, which allows us to draw a smaller (or larger) Android. You will also need to download the AndroidGame file, which will be used to control your Android classes. Make sure you go through both files to make sure you understand what they do. You will notice that there is nothing in the Android class which identifies itself as a superclass. The superclass is seen (and written) as a normal class – it is only the subclass which needs to be adapted so that Python knows what superclass to inherit from.

Create a new file called "BabyAndroid.py". Start the file with the following two import statements: one to import the superclass, and another to import the graphics file.

```
from android import Android
from graphics import *
```

We then need to start the class file, ensuring we tell the interpreter we are inheriting from the Android class:

```
class BabyAndroid(Android):
```

This will automatically mean that the BabyAndroid will be able to move and calculate where the centre of the head is.

The first thing to do is to create the constructor for the BabyAndroid:

```
def __init__(self, win, colour, headCentreX, headCentreY):
      # use the android constructor, but fix both the scale factor and the speed
      Android.__init__(self, win, colour, headCentreX, headCentreY, 0.8, 0.0001)
      # create the text field for when the BabyAndroid wants to cry
      self.cryMessage = Text(Point(0.5, 0.75), "")
      self.cryMessage.setSize(20)
      self.cryMessage.draw(win)
```

The `Android.__init__()` statement tells Python to use the constructor method within the Android class to initialise BabyAndroid.

Once this is completed, we will be able to add the additional methods which are required by the BabyAndroid class:

```
def cry(self):
      self.cryMessage.setText("Baby Android Says: Waaaaaaa!!!!")

def stopCrying(self):
      self.cryMessage.setText("")
```

The completed BabyAndroid will be as follows:

```
from android import Android
from graphics import *

class BabyAndroid(Android):

      def __init__(self, win, colour, headCentreX, headCentreY):
            Android.__init__(self, win, colour, headCentreX, headCentreY, 0.8, 0.0001)
            self.cryMessage = Text(Point(0.5, 0.75), "")
            self.cryMessage.setSize(20)
            self.cryMessage.draw(win)

      def cry(self):
            self.cryMessage.setText("Baby Android Says: Waaaaaaa!!!!")

      def stopCrying(self):
            self.cryMessage.setText("")
```

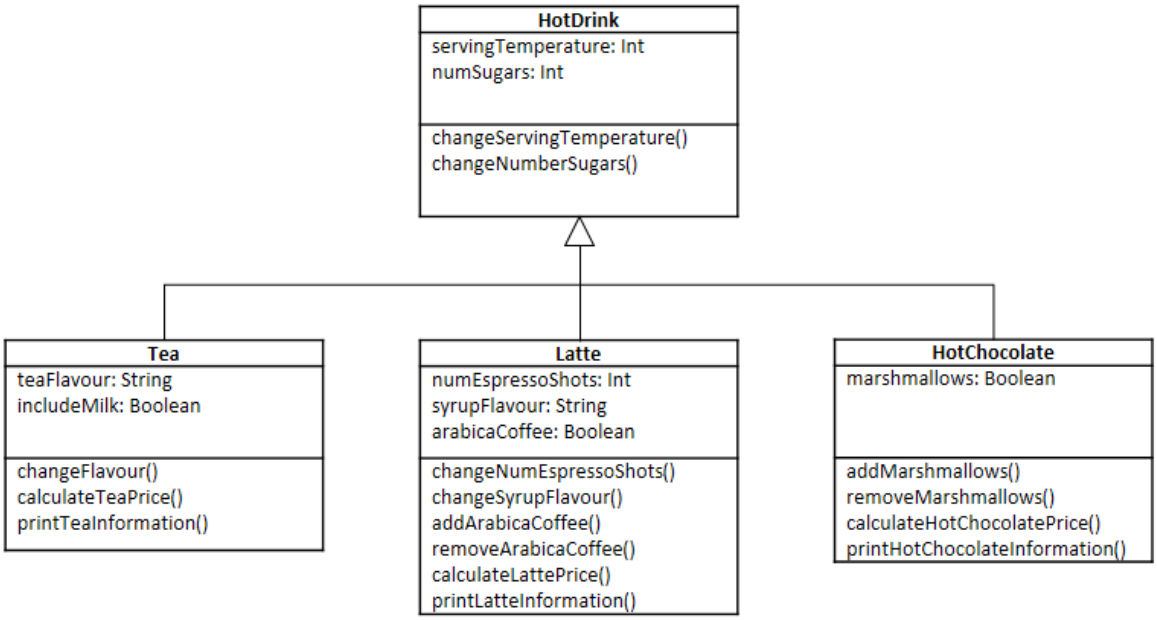
You should notice that we have not re-coded the `moveAndroid()` or `getHeadCentre()` methods. We are, however, able to use the methods, as they have been inherited from the superclass.

Now run the AndroidGame.py file. This should demonstrate using both the superclass and the subclass. Also, try modifying the AndroidGame file (not the classes), so that the Android tries to cry. You should see it produce an error. Why do you think it doesn't work?

3.2. Practice Questions

Create the following classes in Python using inheritance. Functionality for the classes can be modelled either through updating instance variables, using print statements, or both. In each class, you will need to make sure that the values being entered when changing an instance variable are sensible, and provide appropriate error messages. For each question, make sure you create a test file, to ensure that all the methods are working as anticipated.

1. Implement the following inheritance hierarchy:



You can assume the drinks have the following prices:

Tea	£1.20
Milk	5p
Latte (2 Espresso Shots)	£1.32
Extra Espresso Shots	25p
Syrup	20p
Arabica Coffee	45p
Hot Chocolate	£1.79
Marshmallows	40p
Sugar (per spoon)	5p

Modify the superclass and subclasses to make use of method overriding for the printInformation() methods (this will be covered in lecture 4).

2. You have been given the following two classes: Cat, and Dog. Design the generalised class, using the technique followed in lecture 3, and implement all three classes in Python.

Cat
name: String owner: String noiseMade: String furColour: String favouriteFood: String
changeName() changeOwner() scratchOwner() eatFavouriteFood() changeFavouriteFood() makeNoise() purr()

Dog
name: String owner: String noiseMade: String furColour: String favouriteChewToy: String
changeName() changeOwner() playWithFavouriteToy() changeFavouriteChewToy() makeNoise() wagTail()

3. You have been given three classes: Sports Car, 4x4, and Tractor. Design the generalised class, using the technique followed in lecture 3, and implement all four classes in Python.
Hint: Think about which parameters are required to be entered on the initialisation of the object, and which can be assumed (look at the Car example from last week's lecture).

Sports Car
numWheels: Int engineSize: Float turboEngine: Boolean manualTransmission: Boolean sportsMode: Boolean currentGear: Int doorsLocked: Boolean currentMileage: Int roofOpen: Boolean
drive() changeGear() startSportsMode() stopSportsMode() lockDoors() unlockDoors() openRoof() closeRoof()

4x4
numWheels: Int engineSize: Float manualTransmission: Boolean currentGear: Int doorsLocked: Boolean currentMileage: Int offRoadMode: Boolean
drive() changeGear() startOffRoad() stopOffRoad() lockDoors() unlockDoors()

Tractor
numWheels: Int engineSize: Float manualTransmission: Boolean currentGear: Int doorsLocked: Boolean currentMileage: Int trailerAttached:Boolean maximumWeight: Int
drive() changeGear() attachTrailer() detachTrailer() lockDoors() unlockDoors() changeMaximumWeight()

4. You have been given four classes: ISA Savings Account, Fixed Savings Account, Basic Current Account, and Gold Current Account. Design the inheritance hierarchy, and implement all seven classes in Python.
Hint: your hierarchy will have 3 levels in total. First, generalise the four classes provided into two generalised classes. Then generalise those two classes into one superclass.

ISA Savings Account
amountInAccount: Float name: String accountNumber: Int savingsRate: Float taxFreeAmount: Float branch: String
changeName() depositMoney() withdrawFromISA() changeRate() calculateInterestPayment() changeTaxFreeAmount() printISAInformation() changeBranch()

Basic Current Account
amountInAccount: Float name: String accountNumber: Int overdraftLimit: Float overdraftRate: Int branch: String
changeName() depositMoney() withdrawFromCurrentAccount() changeOverdraft() printBasicAccountInformation() changeBranch() changeOverdraftRate()

Fixed Savings Account
amountInAccount: Float name: String accountNumber: Int savingsRate: Float endDateOfFixedRate: Int branch: String
changeName() depositMoney() withdrawFromFixedSavings() changeRate() calculateInterestPayment() changeEndOfFixedRate() printFixedSavingsInformation() changeBranch()

Gold Current Account
amountInAccount: Float name: String accountNumber: Int overdraftLimit: Float accountManager: String branch: String rewardPercentage: Int accountCharge: Float
changeName() depositMoney() withdrawFromCurrentAccount() changeOverdraft() printGoldAccountInformation() changeBranch() changeManager() changeRewardPercentage() calculateReward() changeAccountCharge()

Note: the reward and interest is calculated based on a percentage of the amount which is in the account at the point that the method is run. The endOfFixedRate is the number of days left until the account owner can withdraw money.

Modify the superclasses and subclasses to make use of method overriding for the withdraw() and printInformation() methods (this is covered in lecture 4).