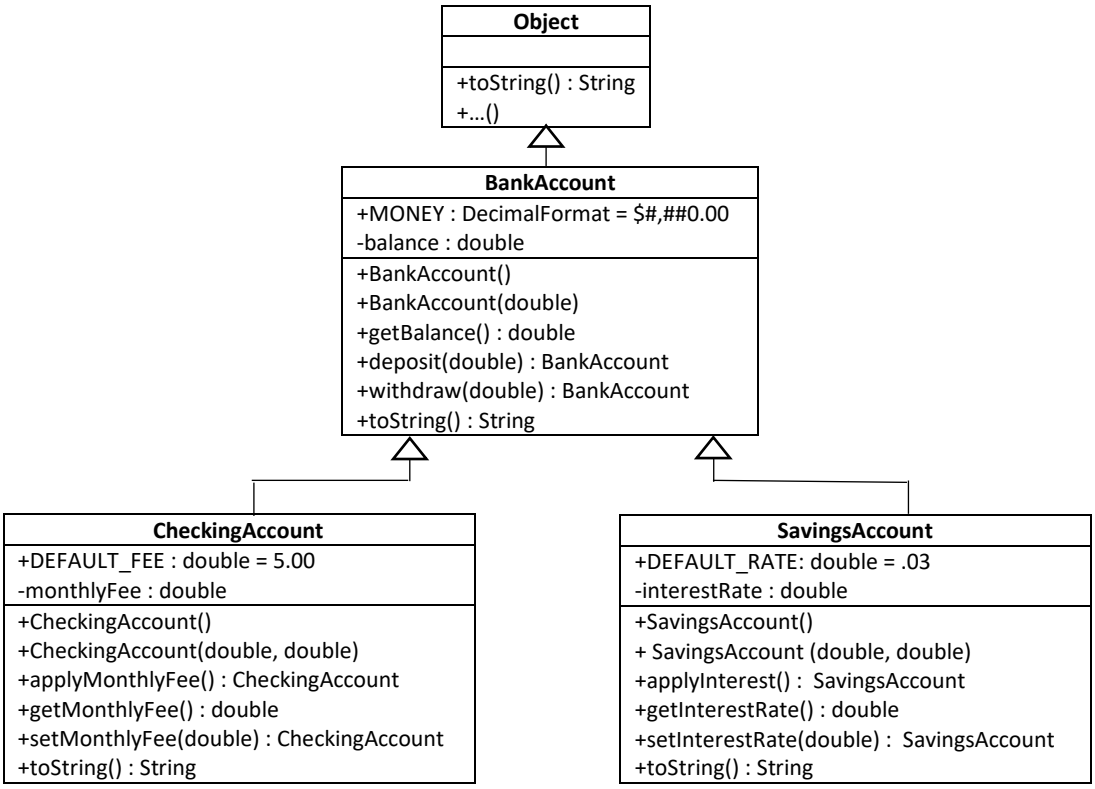
COMP47500 Part A: Programming Assignment

Attempt Parts 1 & 2 in sequence. Submit both Java projects finally. Aim to get each part working correctly before moving on to the next. The bulk of the marks will be awarded for how much you get working, so be sure to submit a working solution, commenting out sections of code if necessary. Marks are also awarded for clean design and coding, formatting and (light) commenting based on the practical exam marking criteria.

PART 1:

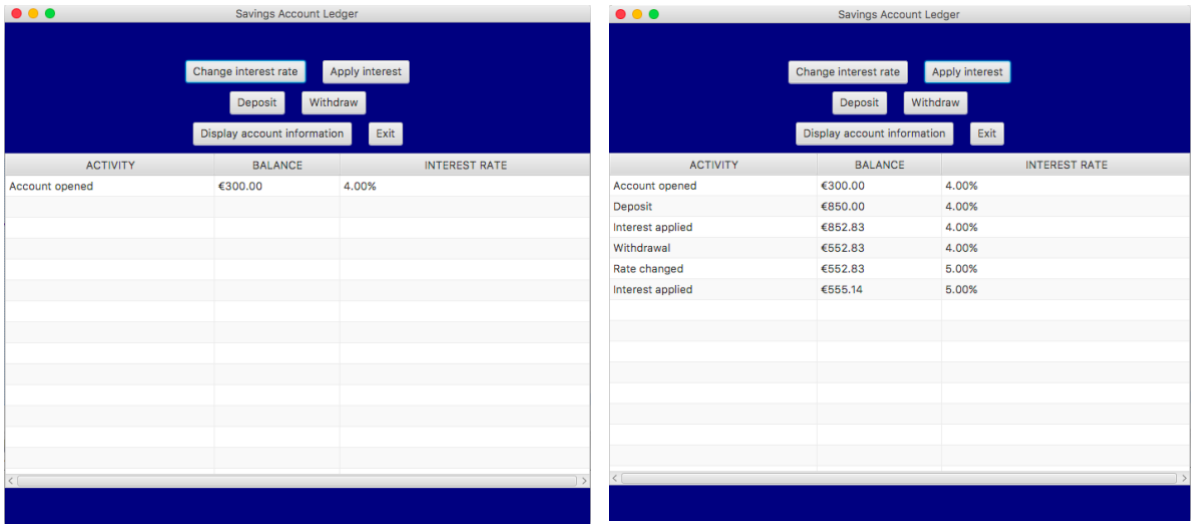
For this programming activity, you will create the *SavingsAccount* class, which inherits directly from the *BankAccount* class. The *SavingsAccount* class is similar to the *CheckingAccount* class in that both classes inherit from *BankAccount.* Figure below shows the resulting hierarchy. The *SavingsAccount* class inherits from the version of the *BankAccount* class in which the *balance* is declared to be *private.* The *SavingsAccount* subclass adds an annual *interestRate* instance variable, as well as supporting methods to access, change, and apply the interest rate to the account balance.



Copy the source files provided to a folder on your computer. Load the *SavingsAccount*.java source file and search for five asterisks in a row (*****). This will position you to the six locations in the file where you will add code to complete the *SavingsAccount* class. The *SavingsAccount.java* file is shown below.

When you have completed the six tasks, load, compile and run the *Teller* application (*Teller.java)*, which you will use to test your *SavingsAccount* class. When the Teller application begins, you will be prompted with a dialog box for a starting balance. If you press "Enter" or the "OK" button without entering a balance, the Teller application will use the default

constructor to instantiate a *SavingsAccount* object. If you enter a starting balance, the Teller application will prompt you for an interest rate and will instantiate a *SavingsAccount* object using the overloaded constructor. Once the *SavingsAccount* object has been instantiated, the Teller application will open the window shown below, which provides buttons for calling the *SavingsAccount* methods to test your code. Below the buttons is a ledger that displays the current state of the savings account. As you click on the various buttons, the ledger will display the operation performed and the values of the balance and the interest rate when that operation is complete.



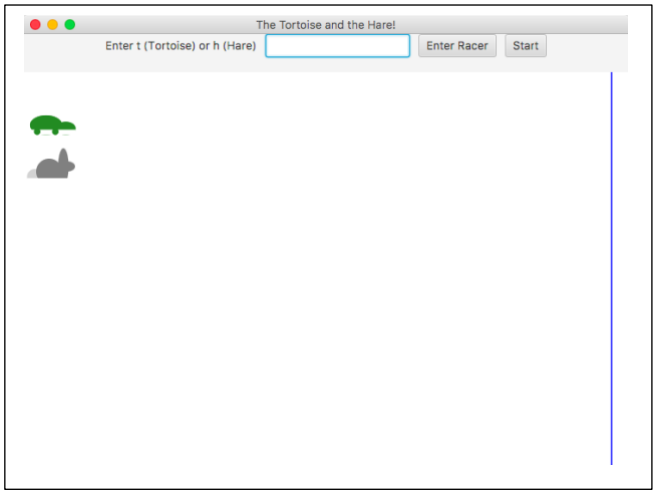
Part 1: Teller windows after several operations have been performed

The operations performed by each button are already coded for you and are the following:

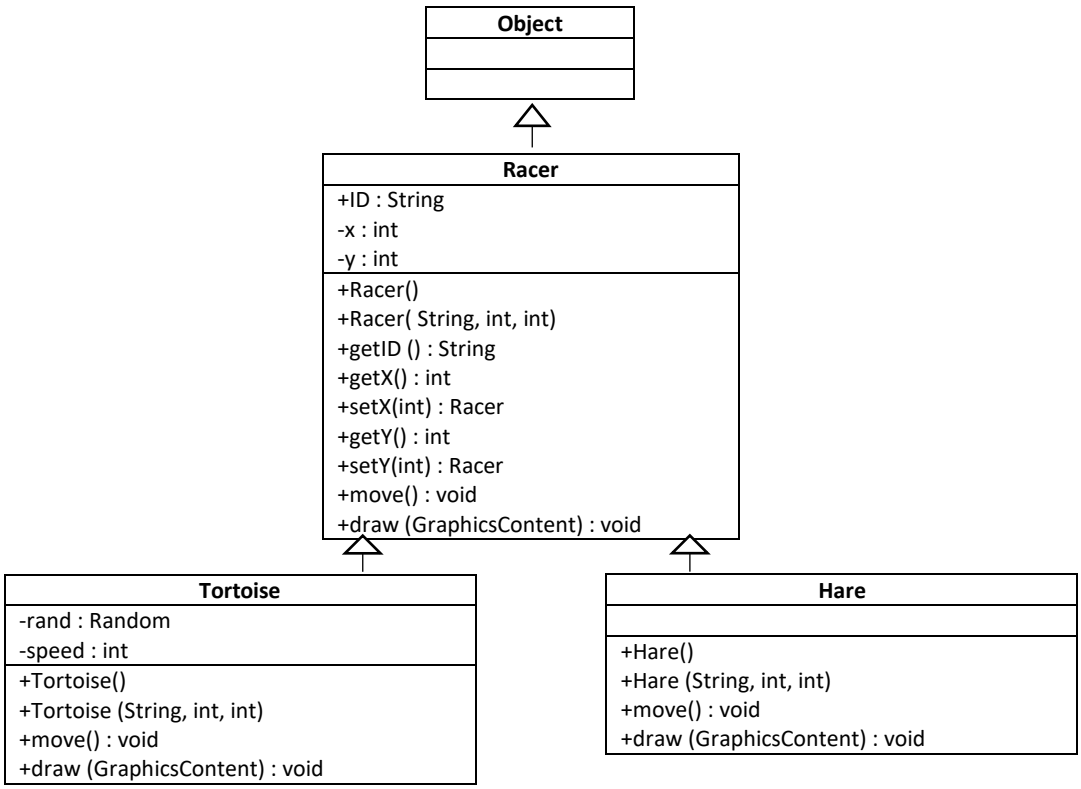
- *Change* interest rate – prompts for a new interest rate and calls your *setInterestRate* method;
- *Apply interest* – calls your *applyInterest* method;
- *Deposit* – prompts for the deposit amount and calls the *deposit* method inherited from *BankAccount;*
- *Withdraw* – prompts for the withdrawal amount and calls the *withdraw* method inherited from *BankAccount;*
- *Display account information* – calls your *toString* method and displays the result in a dialog box;
- *Exit – exits the program.*

PART 2:

For this programming activity, you will complete the implementation of the Tortoise and the Hare race. The Tortoise runs a slow and steady race, while the Hare runs in spurts with rests in between. Below you can see a sample run of the race. In this figure, we show only one tortoise and one hare; however, using polymorphism we can easily run the race with any number and combination of tortoises and hares. The class hierarchy for this programming activity is shown below.



Part 2: Sample run of the Tortoise and Hare Race



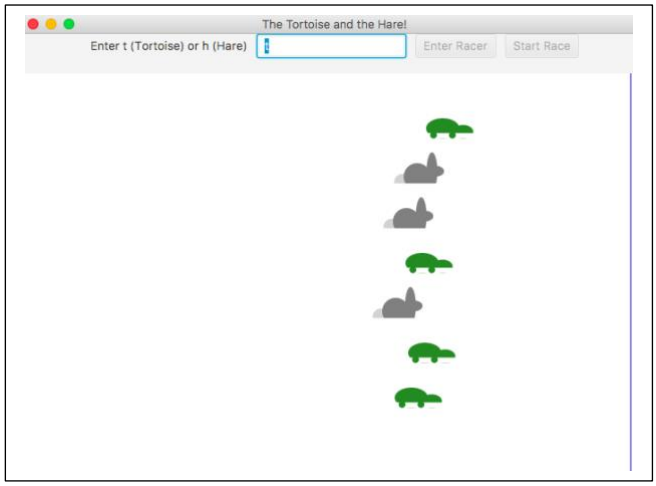
The code for the *Racer* class, which is the superclass of the *Tortoise* and *Hare* classes is provided. The Racer class has three instance variables: a *String ID*, which identifies the type

of racer; and *x* and *y* positions, both of which are *ints*. The class has the usual constructors, as well as accessor and mutator methods for the x and y positions and ID. These instance variables and methods are common to all racers, so we define them in *Racer* class. Individual racers, however, will differ in the way they move and in the way they are drawn. Thus, we declare the *Racer* class to be *abstract* and we define two *abstract* methods, *move* and *draw*. Classes that inherit from the Racer class will need to provide implementations of these two methods (or to be declared abstract as well).
The *Tortoise* and *Hare* classes inherit from the *Racer* class. Their only job is to pass constructor arguments to the *Racer* class and implement the *move* and *draw* methods. For this programming activity, we have provided the *Tortoise* and *Hare* classes with the *move* and *draw* methods already written. Your assignment is to add *Tortoise* and *Hare* objects to an *ArrayList* of *Racer* objects, as specified by the user. Then you will add code to run the race by stepping through the *ArrayList,* calling *move* and *draw* for each *Racer* object.

Copy the source files provided to a folder on your computer. Open *PolymorpshismController*.java file:

1. Write the code to determine which racers will run the race. Search for five asterisks in a row (*****). This will position you inside the *prepareToRace* method.
2. Next, write the code to display the racers at the starting position. Again, search for five asterisks in a row (*****). This will position you inside the *getReady* method, which is called whenever a new Racer is added. The method draws the finish line and displays the racers at the starting position. The code to display the finish line is already written. For this task, you will write code to loop through the *ArrayList* of *Racers*, calling the *draw* method for each racer.
3. Finally, write the code to run the race. Again, search for five asterisks in a row (*****). This will position you inside the *ranRace* method, which is called when the user presses the "Start Race" button. This task is similar to task 2 in that you will loop through the *ArrayList* of *Racers*. In this task however, you will call both the *move* and *draw* methods.

When you have finished writing the code, compile the source code and run the *PolymorpshismApplication* file. Try several runs of the race with different number of racers and with a different combination of *Tortoises* and *Hares*.



Part 2: Another run of the race with Tortoises and Hares