Learning Outcomes

- Create a sub-class of an existing class
- Identify cases of overriding methods and overloading methods
- Understand the purpose of the "super" keyword in inherited methods
- Observe and experiment with polymorphism and dynamic binding
- Use "instanceof" and casting to resolve invalid statements with polymorphism

Pre-Lab

- Create a new Java project called Lab2
- Download the files: BankAccount.java, CheckingAccount.java, TestBankAccounts.java, BankMachine.java, and AnswersLab2.txt
- Save these downloaded files into the Lab2 src folder

Exercise 1 – Creating a sub-class

1. Create a new SavingsAccount class in your project, which extends the class BankAccount.
2. Besides the instance variables and methods inherited from class BankAccount, this class must contain one instance variable of type double called interestRate. We might need to create several objects of the new class SavingsAccount, each one with its own balance and interest rate. Should instance variable interestRate be declared as static? Write your answer in AnswersLab2.txt.
3. This class must also contain the following methods:
    a. constructor with two parameters of type double. The method signature is thus: `public SavingsAccount(double initialAmount, double rate)`. This constructor must initialize the interestRate instance variable with the value of rate and it must also initialize the instance variable balance of class BankAccount with the value of initialAmount (if you do not know how to do this, please review the lecture notes on inheritance. If you still need additional help, please see class CheckingAccount to see how to do this).
    b. public getter method called `getInterestRate()` that returns the value of interestRate.
    c. public method `calculateInterest()`, which calculates one month's interest and adds this amount to the balance; the interest is computed by multiplying interestRate and the balance. How can you get the balance from BankAccount?

How do you add the interest to the balance of BankAccount? (If you are unsure about how to do this, please read class BankAccount.)

   d. public method `toString()`, which returns a string representation of the instance variables; this string must be of the form "SavingsAccount: balance $123, interest rate 0.12".

4. Add this test harness (copy and paste it) to your SavingsAccount class.
5. Run the program and write the output in AnswersLab2.txt.

Exercise 2 – Experimenting with inheritance

1. Open TestBankAccounts.java. This class has a main method that creates several bank account objects of the three types: BankAccount, CheckingAccount, and SavingsAccount.
2. In each step of this exercise, you will be adding statements to the main method and running it after each addition. Answer the questions at each step and write your answers in AnswersLab2.txt. If there is a compilation error, make sure you comment out the offending statement before advancing to the next step of this exercise.
3. Which method(s) of the class CheckingAccount demonstrate overriding? Which demonstrate overloading?
4. In the main method, add the statement `bacc0 = chacc1;` which makes the BankAccount variable bacc0 reference the CheckingAccount object referenced by chacc1. Is this legal? Why?
5. Add the statement `chacc1 = bacc1;`. Is this legal? Why not? If not, comment it out.
6. Add the statement `BankAccount bacc2 = new CheckingAccount(200.0);` then add the statement `chacc1 = bacc2;`. Fix this last statement using casting so that it compiles and runs.
7. Add statements to invoke method deductFees(), without parameters, on all variables bacc1, chacc1, and sacc1. On which type of objects was this legal? Why?
8. Try to fix the above compilation errors through casting: `((CheckingAccount)variable).deductFees()`. Which compilation error(s) could be fixed this way? Which one(s) could not be fixed this way? Why?
9. If there are compilation errors (cannot be more than 1), comment out the line causing the error. Then run the program. Did the program run? If the program crashed, why did it crash?
10. If the program crashed, comment out the statement that caused the program to crash. Add the statement `chacc1.deposit(100.0);` Your program should compile and run.
11. Now make the following change to CheckingAccount.java: in the code for the deposit method, change `super.deposit(amount);` to `deposit(amount);` Compile and run. What was the runtime error?

12. To understand why the program crashed, add a breakpoint in class TestBankAccounts at the line `chacc1.deposit(100.0);` (review Lab 2 if you do not remember how to do this).
13. Run the program using the debugger. The execution of the program will stop at the line where you added the breakpoint. Repeatedly press F5 to see how the execution of the program proceeds. How many times is the method `deposit()` invoked?
14. Terminate the program by clicking on the red square in the tool bar at the top of Eclipse's window.

Exercise 3 – Experimenting with polymorphism

1. Open BankMachine.java and read over the code in its `main()` method.
2. The program produces two compilation errors, one when invoking method `getTransactionCount()` and the other when invoking `getInterestRate()`. Explain why the compiler issues these two errors. Write your answer in AnswersLab2.txt.
3. In the statement `String accountInfo = newAcc.toString();` just by looking at the code can you tell whether the method `toString()` being invoked is from SavingsAccount or from CheckingAccount? Explain your answer and write it in AnswersLab2.txt.
4. Modify the code so that `newAcc.getTransactionCount()` is invoked only if newAcc references an object of the class CheckingAccount, and `newAcc.getInterestRate()` is invoked only if newAcc references an object of the class SavingsAccount.
5. Run the program and verify that it works correctly for both account types (run the program and type either 'c' or 's' so the program prints a message indicting that an account has been created and it also prints information about the account including either the interest rate or the number of transactions).

Submission

When you have completed the lab, navigate to the weekly module page on OWL and click the Lab link (where you found this document). Make sure you are in the page for the correct lab. Upload the files listed below and remember to hit Save and Submit. Check that your submission went through and look for an automatic OWL email to verify that it was submitted successfully.

LAB 2

Rules

- Please only submit the files specified below. Do not attach other files even if they were part of the lab.
- Do not ZIP or use any other form of compressed file for your files. Attach them individually.
- Submit the lab on time. Late submissions will receive a penalty.
- Forgetting to hit "Submit" is not a valid excuse for submitting late.
- Submitting the files in an incorrect submission page will receive a penalty.
- You may re-submit code if your previous submission was not complete or correct, however, re-submissions after the regular lab deadline will receive a penalty.

Files to submit

- SavingsAccount.java
- AnswersLab2.txt