CSCI 2302

**Abstract Classes & Interfaces Chapter**

**Abstract Classes Lab**

Contents

[Intro 1](#_Toc85439049)

[Notes 1](#_Toc85439050)

[Advantage of using the abstract methods 2](#_Toc85439051)

[Learning Goals 3](#_Toc85439052)

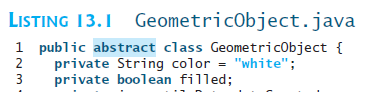
[Task 3](#_Toc85439053)

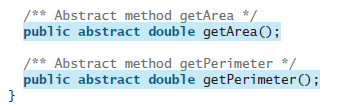
[Submit 4](#_Toc85439054)

Intro: Abstract classes allow another way to design classes and provides flexibility in the maintenance of classes.

Notes: An abstract class contains abstract methods, which are implemented in concrete subclasses. An abstract class is a superclass that contains common features of its subclasses. The abstract method cannot be defined in the superclass because each subclass has its own specific way that method is to be implemented/defined.

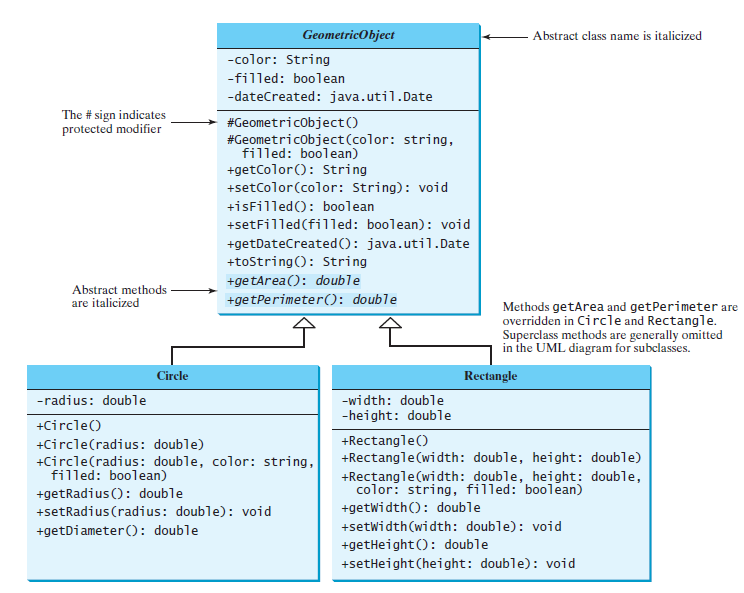
The purpose of an abstract class is to provide a common definition of a base class that multiple derived classes can share.

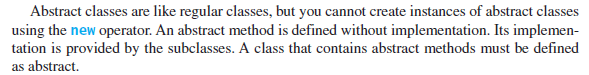
****



In the UML diagram, abstract classes and abstract methods use italics.

The abstract methods do not show in the subclass’UML that is defining the abstract method. That abtract method has to be defined in the concrete class.

****

****



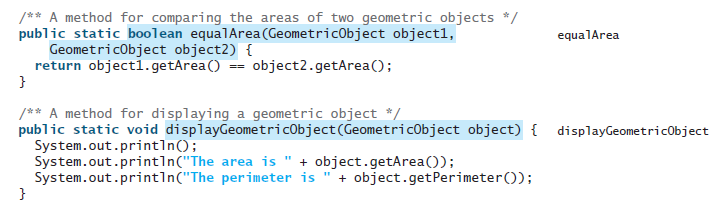
You cannot instantiate an abstract class object, but you can use polymorphism with an abstract class.

Advantage of using the abstract methods:

Abstract classes are a way of organizing a program. You can get the same thing done without using this way; it is a matter of program design. The advantage of using an abstract class is that you can group several related classes together as siblings. Grouping classes together is important in keeping a program organized and understandable.

The abstract class and inheritance collectively ensure that most of the code are written using abstract and higher-level classes, so that it can leverage Inheritance and Polymorphism to support future changes.

It also eliminates the need for testing and casting to get to methods/behaviors of the actual subtype. Since it is declared in the superclass and defined in the subclass, the JVM (during dynamic binding) knows which class to invoke the method.



Learning Goals: To integrate Abstract classes and abstract methods into our understanding about class/object design. To assess the advantages of using abstract classes.

Task: Complete the following steps.

1. Implement an abstract class based on the Account UML diagram.
2. Modify the SavingsAccount.java and the CheckingAccount.java to inherit from the Account class.
   1. In the CheckingAccount Modify the withdraw method by implementing an overdraft protection that charges $35.00 when there is NSF (non-sufficient funds) and allow the withdrawal; meaning, if there is not enough funds to cover the withdraw, the account will be charged $35.00.
3. Save the TestBank.java with your mySFAusername\_TestBank.java and modify the TestBank.java to do the following:
4. instantiate a SavingsAccount object with ID 2526, a balance of 20000 and an annual interest rate of 1.5%.
   1. Invoke the updateInterest method
   2. Invoke the withDraw method to withdraw 5000
   3. Invoke the deposit method to deposit 10000.
5. instantiate a CheckingAccount with ID 3565, a balance of 150, an annual interest rate of 1.05%.
   1. Invoke the updateInterest method
   2. Invoke the withDraw method to withdraw 300
   3. Invoke the depost method to deposit 200
6. Implement a method, printOutAccount, that accepts Account objects as the argument and prints the id, balance, and the interestRate.
   1. Invoke the printOutAccount with the SavingsAccount object instantiated in step 3a
   2. Invoke the printOutAccount with the CheckingAccount object instantiated in step 3b

**Input:** None, the data is hard-coded in the program.

Submit: the SavingsAccount.java, CheckingAccount.java, Account.java and mySFAUsername\_TestBank.java file(s) in the Dropbox in Brightspace by D2L.

|  |
| --- |
| ***Account*** |
| # id: int  # balance: double  # interestRate: double  # dateCreated: Date |
| #Account()  #Account(id: int, balance: double, interestRate: double)  +setId(id: int): void  +setBalance(balance: double): void  + setInterestRate(interestRate: double):void  + getId(): int  + getBalance(): double  + getInterestRate(): double  + getMonthlyInterestRate(): double  + getDateCreated(): Date  + toString(): String  *+ updateInterest(interestRate: double): double*  *+ withDraw(withDrawAmount: double): double*  *+ deposit(depositAmount: double): double* |

FYI: The getMonthlyInterest is the interestRate / 12 or interestRate / 1200 depending on how to format the APR.