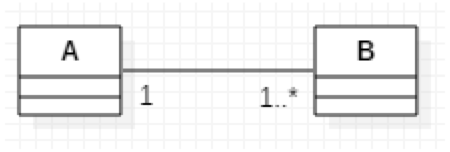
CS401 MPP Midterm Sample Solution

Name:____________________________ StudentId:_____________________________

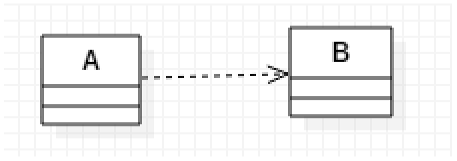
Part I	Part II.1	.2	.3	Part III	SCI

Part I: Short Answer (3 points each)

_F__ 1. (T/F) To implement the class diagram below in code, a list of type A objects must be placed inside the class B.



_F__2. (T/F) If the class diagram below has been implemented in code, the following must be true at runtime: When an instance of class A is created, it keeps a reference to class B.



3. Complete a MySingleton class:

```
public class MySingleton {
   //create an instance
   private static MySingleton instance = new MySingleton();

   //constructor
   private MySingleton() {}

   //implmement a factory method that return the instance
   public static MySingleton getInstance() {
      return instance;
   }



}
```

_C__4. What happens when the main method in the following code is executed?

```
public class Base extends Extension {
    public static void main(String[] args) {
        Extension s = new Base();
        s.print();
    }
    public void print() {
        System.out.println("From Base");
    }
}
```

```
public class Extension {
    void print() {
        System.out.println("From Extension");
    }
}
```

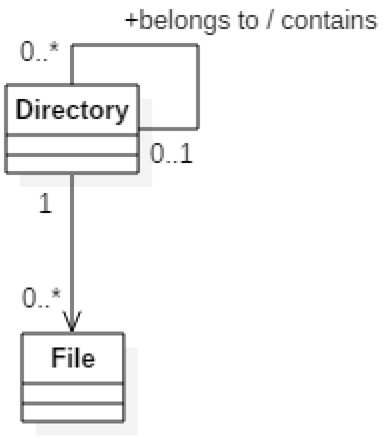
A. There is a compiler error.
B. There is a runtime error.
C. "From Base" is printed to the console.
D. "From Extension" is printed to the console.

Part II: Skill Questions

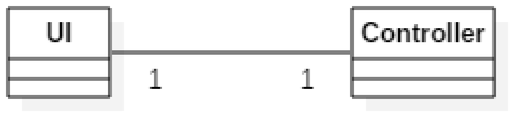
1. A file system for an operating system on a computer is set up so that files can be organized into directories that have a hierarchical structure. The file system satisfies the following requirements:
   - A directory may contain zero or more other directories and zero or more files
   - There is a top-level directory that is not a subdirectory of any other directory
   - Every file belongs to exactly one directory and every directory, other than the top-level directory, belongs to exactly one other directory.

   Draw a class diagram that models this kind of file system. Your diagram should consist of two classes together with the correct type of association(s) and multiplicities.

   //Your solution goes here



2. The diagram below shows that (for a particular application) there is a one-one bidirectional association between a UI class and a Controller class.



In the space provided below, write Java code that implements this diagram. Assume that UI and Controller are the only classes in a particular package. Your code must meet the following requirements:

   a. The UI class owns the relationship, so it should not be possible to create an instance of Controller independently of an already existing UI class
   b. The code must show relevant instance variables, constructor implementations, and methods, sufficient to implement this model. (Show only those instance variables that are implied by the diagram. Getters for instance variables should be provided.)
   c. All classes, properties, methods, and constructors must be given appropriate visibility qualifiers (private, protected, public, or package level).

```java
public class UI {
    private Controller controller;
    public UI() {
        controller = new Controller(this);
    }
}
public class Controller {
    private UI ui;
    //package level access
    Controller(UI ui) {
        if(ui == null)
            throw new NullPointerException(
                    "UI argument must not be null!");
        this.ui = ui;

    }
}
```

3. **You already practiced it. (it is practice 1.) see the solution from Sakai.** In a company, employees may have multiple bank accounts: zero or more savings accounts and zero or more checking accounts. Each checking account has an account id, a balance, and a monthly fee. Each savings account has an account id, a balance, and an interest rate associated with the particular type of savings account. It is possible to read the current balance in any of these accounts, but it

is also possible to determine the balance after interest or monthly fee is applied by calling the `computeUpdatedBalance` method on the account.

An administrator has access to all employee records and from time to time computes the total balance across all employee-owned accounts; for each account, the balance that is needed in this computation is the *updated* balance. This computation is performed in the static method

`computeUpdatedBalanceSum`

in the `Admin` class.

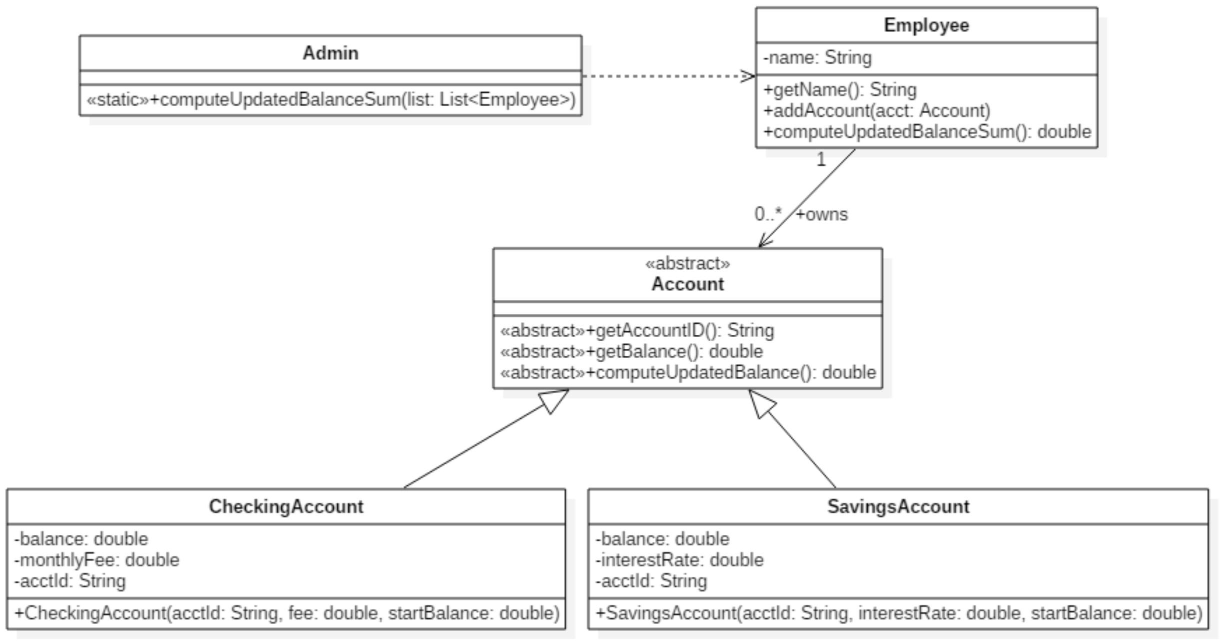
Below is a class diagram showing the classes involved and relationships between them. A sequence diagram for the operation `computeUpdatedBalanceSum` is also provided. Your task in this problem is to write Java code that implements the classes and relationships shown in the diagram. Shells for Admin and Employee have been provided for you (below).

The method `computeUpdatedBalance` in `CheckingAccount` does the following computation to obtain the return value:

`balance – monthlyFee.`

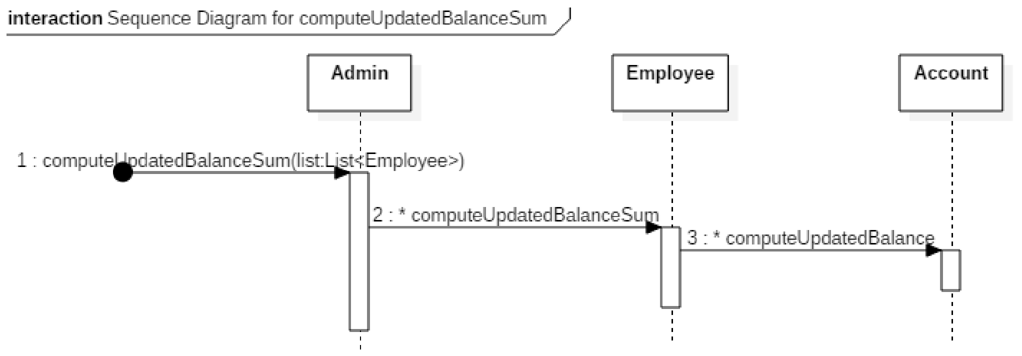
The method `computeUpdatedBalance` in `SavingsAccount` performs the following computation to obtain the return value:

`balance + (interestRate * balance).`



Admin
«static»+computeUpdatedBalanceSum(list: List<Employee>)
Employee
-name: String
+getName(): String
+addAccount(acct: Account)
+computeUpdatedBalanceSum(): double
1
0..*
+owns
«abstract»
Account
«abstract»+getAccountID(): String
«abstract»+getBalance(): double
«abstract»+computeUpdatedBalance(): double
CheckingAccount
-balance: double
-monthlyFee: double
-acctId: String
+CheckingAccount(acctId: String, fee: double, startBalance: double)
SavingsAccount
-balance: double
-interestRate: double
-acctId: String
+SavingsAccount(acctId: String, interestRate: double, startBalance: double)

}

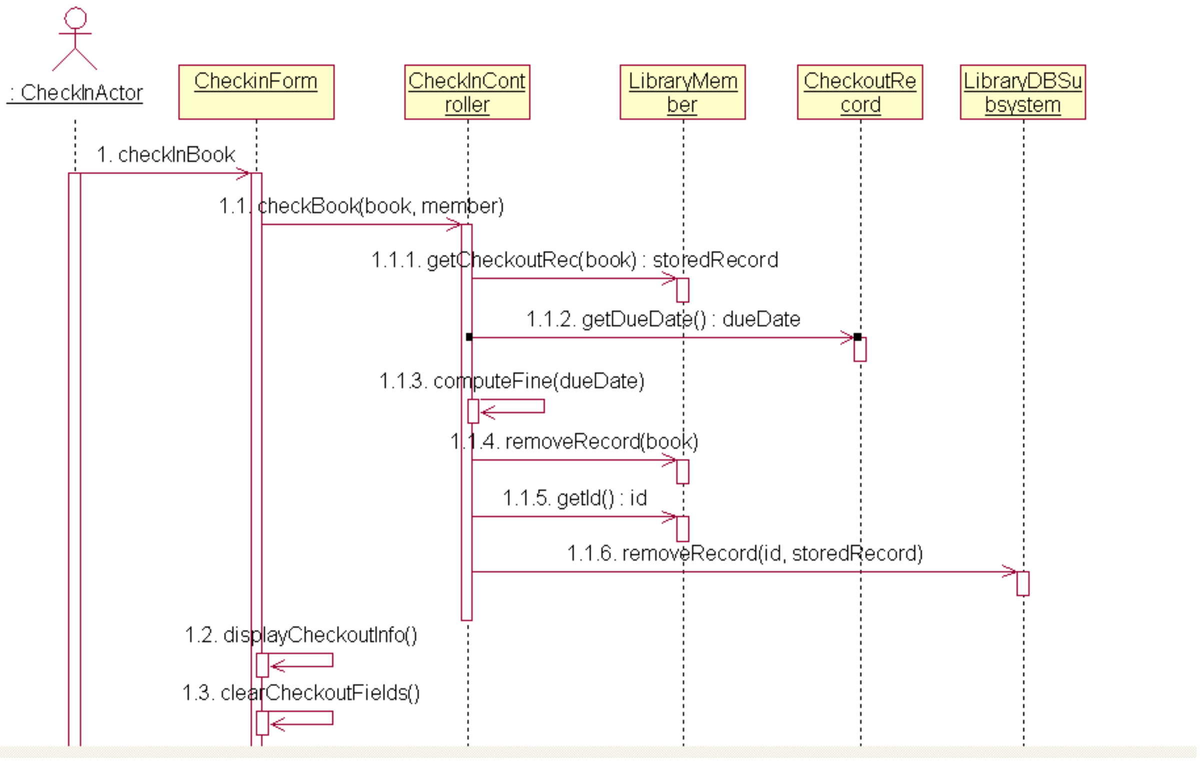

interaction Sequence Diagram for computeUpdatedBalanceSum
Admin
Employee
Account
1 : computeUpdatedBalanceSum(list:List<Employee>)
2 : * computeUpdatedBalanceSum
3 : * computeUpdatedBalance

//Your code for Problem 3 should begin here

```
public class Admin {
   public static double computeUpdatedBalanceSum(List<Employee> list) {









   }
}

public class Employee {
   public double computeUpdatedBalanceSum() {















   }
}
```

Part III. Draw the Sequence diagram in the space below when an actor invokes the checkinBook method on CheckinForm. (You may check the result from slide 9, Lecture 4.)

```
//FROM CLASS CheckinForm
public void checkinBook() {
   theCheckinController.checkBook(m_book, m_member);
   displayCheckoutInfo();
   clearCheckoutFields();
}

//FROM CLASS CheckinController
public void checkBook(Book book, LibraryMember member) {
   CheckoutRecord storedRecord = member.getCheckoutRec(book);
   Date dueDate = storedRecord.getDueDate();
   double fine = computeFine(dueDate);
   member.removeRecord(book);
   libraryDBSubsys.removeRecord(member.getID(), storedRecord);
}
```



: CheckInActor
CheckinForm
CheckInController
LibraryMember
CheckoutRecord
LibraryDBSubsystem
1. checkInBook
1.1. checkBook(book, member)
1.1.1. getCheckoutRec(book) : storedRecord
1.1.2. getDueDate() : dueDate
1.1.3. computeFine(dueDate)
1.1.4. removeRecord(book)
1.1.5. getId() : id
1.1.6. removeRecord(id, storedRecord)
1.2. displayCheckoutInfo()
1.3. clearCheckoutFields()

SCI. [3 pts] In an ancient text, one reads the following:

Know that by which all this is known.

1. What does this expression mean? What is it saying? Is it some kind of SCI point?
2. Does this expression illuminate any aspect of the software engineering discipline discussed in class? Explain.