Segundo examen parcial
Computacional II
Nombre alumno:

Resuelva los siguientes ejercicios. Por favor, tenga en cuenta todos los detalles de programación orientada a objetos (POO) discutidos durante el curso.

1. Defina una clase llamada "Fecha".

   a) Escriba un programa que pruebe su clase (se refiere a un main).

   b) Modifique la clase Fecha definida antes para incluir una función "diaSig()" que incremente una fecha en un día. Pruebe su función para asegurarse que incrementa de manera correcta los días en un nuevo mes y en un nuevo año.

      nota: para este ejercicio "una sola" solución es suficiente. Es decir, a) y b) pueden solucionarse juntos.

2. Clases y Herencia.

   Cree una jerarquía de herencia que podría usar un banco para representar las cuentas bancarias de los clientes. Todos los clientes en este banco pueden depositar (es decir, abonar) dinero en sus cuentas, y retirar (es decir, cargar) dinero de ellas. También existen tipos más específicos de cuentas. Por ejemplo, las cuentas de ahorro obtienen intereses sobre el dinero que contienen. Por otro lado, las cuentas de cheques cobran una cuota por transacción (es decir, abono o cargo).
   Cree una jerarquía de herencia que contenga la clase base “Cuenta", junto con las clases derivadas “CuentaAhorros" y “CuentaCheques" que hereden de la clase Cuenta. La clase base Cuenta debe incluir un miembro de datos de tipo double para representar el saldo de la cuenta. La clase debe proporcionar un constructor que reciba un saldo inicial y lo utilice para inicializar el miembro de datos. El constructor debe validar el saldo inicial, para asegurar que sea mayor o igual a 0.0 . De no ser así, el saldo debe establecerse en 0.0 y el constructor debe mostrar un mensaje de error, indicando que el saldo inicial es inválido. La clase debe proporcionar tres funciones miembro. La función miembro abonar debe sumar un monto al saldo actual. La función miembro cargar debe retirar dinero de la Cuenta y asegurar que el monto a cargar no exceda el saldo de la Cuenta . Si lo hace, el saldo debe permanecer sin cambio y la función debe imprimir el mensaje “El monto a cargar excedio el saldo de la cuenta." La función miembro getSaldo debe devolver el saldo actual.
   La clase derivada CuentaAhorros debe heredar la funcionalidad de una Cuenta, pero también debe incluir un miembro de datos de tipo double que indique la tasa de interés (porcentaje) asignada a la Cuenta. El constructor de CuentaAhorros debe recibir el saldo inicial, así como un valor inicial para la tasa de interés de CuentaAhorros. CuentaAhorros debe proporcionar una función miembro public llamada calcularInteres, que devuelva un valor double que indique el monto de interés obtenido por una cuenta. La función miembro calcularInteres debe determinar este monto, multiplicando la tasa de interés por el saldo de la cuenta. (Nota: CuentaAhorros debe heredar las funciones miembro abonar y cargar como

están, sin redefinirlas).
La clase derivada CuentaCheques debe heredar de la clase base Cuenta e incluir un miembro de datos adicional de tipo double, que represente la cuota que se cobra por transacción. El constructor de CuentaCheques debe recibir el saldo inicial, así como un parámetro que indique el monto de la cuota. La clase CuentaCheques debe redefinir las funciones miembro abonar y cargar de manera que resten la cuota del saldo de la cuenta, cada vez que se realice una de esas transacciones con éxito. Las versiones de CuentaCheques de estas funciones deben invocar la versión de la clase base Cuenta para realizar las actualizaciones en el saldo de una cuenta. La función cargar de CuentaCheques debe cobrar una cuota sólo si realmente se retiró dinero (es decir, que el monto a cargar no exceda el saldo de la cuenta). (Sugerencia: defina la función cargar de Cuenta de manera que devuelva un valor bool que indique si se retiró dinero. Después use el valor de retorno para determinar si se debe cobrar una cuota). Después de definir las clases en esta jerarquía, escriba un programa para crear objetos de cada clase y evaluar sus funciones miembro (se refiere a un “main"). Agregue interés al objeto CuentaAhorros , primero invocando a su función calcularInteres y después pasando el monto de interés devuelto a la función abonar del objeto.

3. Para jugar al ahorcado escriba un programa . El programa debe elegir una palabra (que se lee de un archivo de texto) y mostrar lo siguiente:

   adivine la palabra: xxxxx

   cada x representa una letra. El usuario tratara de adivinar las letras en la palabra. Debera mostrarce la respuesta apropiada (si la letra pertenece o no) despues de cada intento. Si la letra pertenece a la palabra buscada dele la opotunidad de adivinar si el desea. Despues de siete intentos incorrectos (ya sean palabra o letras) el usuario sera colgado.

   Gane o pierda dele la opcion al usuario de jugar de nuevo.

   Nota: Si desea hacer su programa mas grafico, después de cada intento incorrecto, muestre el diagrama con alguna parte del cuerpo incluida

Figure 1: Ahorcado.