CS 247 Spring 2014
Assignment 1 Question 1 --- Specifications

Q1 [40 marks] Polymorphism

You are to complete the implementation of a family of ADTs for cellphone accounts. A cellphone account keeps track of information needed to bill a customer for their cellphone usage. There are two types of cellphone plans that a customer might subscribe to, and the plan type affects what information the account needs to keep track of:

- **Cheap Plan:** has a monthly service fee of $30 and gives the customer 200 free minutes of calls each month. If the customer makes more than 200 minutes of calls in a month, the customer is charged $1 for each minute over and above the 200 free minutes.
- **Expensive Plan:** has a monthly service fee of $100 and gives the customer unlimited free minutes of call time.

Objective

You must complete the class declaration and provide the class implementation for an abstract base class `Account`. You must also declare and define derived classes `CheapAccount` and `ExpensiveAccount` that maintain billing information for Cheap cellphone service plans and Expensive cellphone service plans, respectively. ***Your solution cannot modify the provided public interface to class Account except to indicate virtual functions.*** You can add protected member functions, private data members and member functions, and friends.

Provided Code

We have provided an initial code file `AccountTestHarness.cpp` that includes the public interface for an abstract base class `Account`. An `Account` object maintains billing information for some customer's cell-phone plan. Billing information includes the balance owed by the customer, where a positive value indicates a credit and a negative value indicates the balance due. Billing information also includes any additional information needed to charge the customer appropriately at the end of the month (e.g., number of minutes of calls).

We have also provided a main program for your solution The main program is a **test harness** that you can use to test your ADT implementations by creating accounts of various types, performing operations on those accounts, and printing the contents of the accounts. The test harness is not robust. (It is throwaway code.) If you enter invalid commands, it might cause the program to terminate. It should go without saying (but I'll say it anyways) that your program will be tested only on valid input commands.

Initialization

When the program starts executing, it prints a welcome message and asks for the first command from the user.

Commands

There are 7 valid commands that the test harness will recognize:

E
C
P
b <acctNo>
c <acctNo> <duration>
p <acctNo> <amount>
B
CTL-d

a) E, C

Command E causes the test harness to create a new `ExpensiveAccount` object that has an automatically generated account number and a balance of 0. The command then prints the value of the object, using the object's `print()` method. The output of the `print()` method looks like this:

```
Expensive Account:
<sp><sp>Account Number = <acct no>
<sp><sp>Balance = $0<eoln>
```

Command C causes the test harness to create a new `CheapAccount` object that has an automatically generated account number and a balance of 0; the number of minutes used in the current billing period is initialized to 0. The command then prints the value of the object, using the object's `print()` method. The output of the `print()` method looks like this:

```
Cheap Account:
<sp><sp>Account Number = <acct no>
<sp><sp>Balance = $0
<sp><sp>Minutes = 0<eoln>
```

In the above outputs, <acct no> is a 4-digit account number (e.g., 0001).

b) P

The test harness prints all of the accounts, using the account objects' `print()` methods.

c) b <acctNo>

The test harness prints the balance of the specified account. The output looks like the following, where <bal> is the return value of the object's `balance()` method.

```
Value of balance data member is:  <bal>
```

d) c <acctNo> <duration>

The test harness updates the specified account object by invoking the object's `call()` method. Depending on the type of account, the account's information about minutes used may be updated.

e) p <acctNo> <amount>

The test harness updates the specified account object's balance by <amount> dollars, by invoking the object's `payment()` method.

f) B

The test harness invokes the `bill()` method for all accounts. The effect is that each account's balance is decremented by the appropriate monthly charge (depending on the type of account). An account's balance is further decremented by an appropriate amount if the number of minutes used since the last invocation of `bill()` exceeds the number of free minutes that the account's plan grants per month.

g) CTL-d

If you hold down the control key on your keyboard while pressing the D key, you will generate an EOF (end of file) character. When the test harness's input stream sees this, it terminates the program.

Sample Execution

Below is an example partial execution. User input is shown in **bold** font.

```
Test harness for family of phone-service accounts:

Command: E
ExpensiveAccount:
  Account Number = 0001
  Balance = $0

Command: C
CheapAccount:
  Account Number = 0002
  Balance = $0
  Minutes = 0

Command: c 2 400

Command: c 0002 600

Command: B

Command: P
ExpensiveAccount:
  Account Number = 0001
  Balance = -$100
CheapAccount:
  Account Number = 0002
  Balance = -$830
  Minutes = 0

Command: ^D
```