Exam 1
Febrary 21, 2003

Do not open this booklet until you are told to begin!

There are 20 points on this exam. Read each problem carefully, and avoid spending too much time on any one question! There is a 1-page appendix at the end of this exam that included code you have seen before. You may tear that page off your exam if you wish.

Problem 0 (1 point): Fill out all the following information. Once you are told to begin, also fill out your name and login at the top of each page.

Your name:	
Your cs61b login:	
Lab time or section #:	
Lab TA's name:	
Name of person to your left:	
Name of person to your right:	

Grading: Do not write below this line. The following grid will be used for grading.

Problem	Score	Possible
0		**1**
1		**5**
2		**5**
3		**4**
4		**5**
Total		**20**

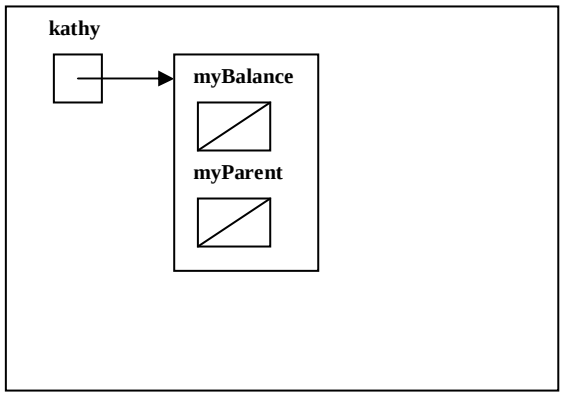
Do not open this booklet until you are told to begin!

Name: _____ Login: _____

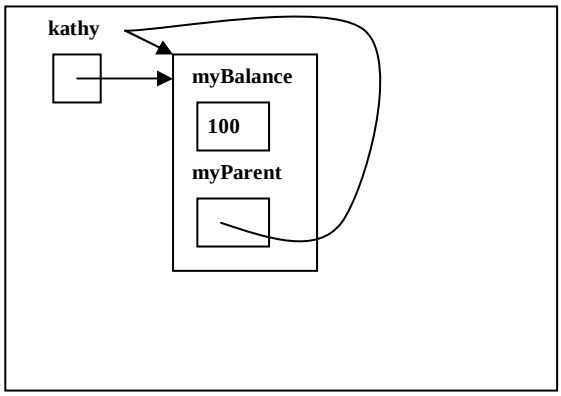
Problem 1: (5 points) This problem refers to the Account class from Lab2, which is attached to the end of this exam booklet as an appendix.

Parts A-D: Below are some possible box-and-pointer diagrams for Account objects. Which of the pictures are legal diagrams that could be produced by a program **outside** the Account class, e.g., a separate test class. Circle "yes" if such a program could produce the picture, "no" if it could not. **(For parts A-D only, +1 point for each correct answer, -2 for each incorrect one.)**

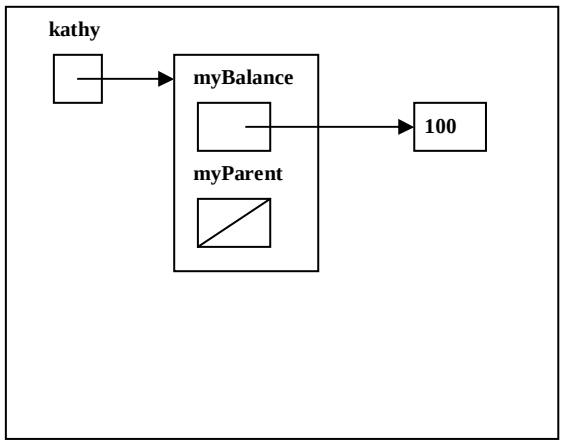
A) Yes No



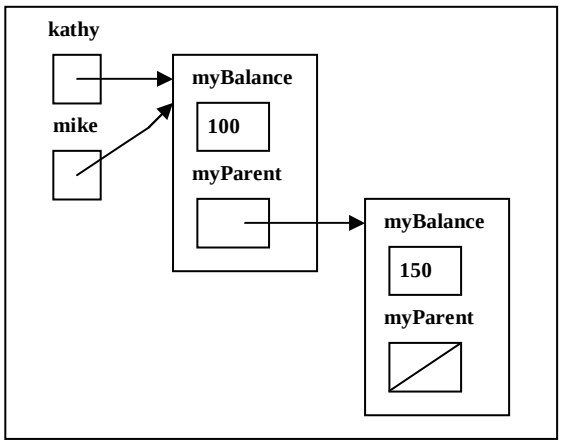
B) Yes No



C) Yes No



D) Yes No



Part B: Fill in the 1-line body of a method *clearAcct* with the following specification. As in parts A-D, this method is part of a test class that is **not** the Account class.

```
/** Modifies the given Account by clearing it of funds; afterwards, the balance() will be 0.
 *  @param a is the account to be cleared
 */
public static void clearAccount(Account a) {

  _____

}
```

Name: _____ Login: _____

Problem 2: (5 points) A *palindrome* is a word or phrase that is the same forward as backwards. That is, if you reverse the letters of the string, you get the same string. An example is the string, "able was I ere I saw elba." Write a non-recursive method *isPalindrome* that determines if the input String is a palindrome. (Your solution need not use every line.)

```
/** Determines if the input String is a palindrome.
 * @param s is the String to be checked; s must not be null.
 * @return true if s is a palindrome, false otherwise.
 */
public static Boolean isPalindrome (String s) {

    _____

    _____

    _____

    _____

    _____

    _____

}
```

Problem 3: (4 points) What are some interesting test strings on which *isPalindrome* should be tested? One is given in the table below. Fill in four additional test cases that test different kinds of inputs to the method. The "general category" column tells us why you chose this particular test. (Your tests need not cover all possible interesting cases, but each test should cover a different kind of input.)

Input	Expected Output	General category of strings tested
""	true	Empty strings

Name: _____ Login: _____

Problem 4 (5 points) Consider working with a *Riddle* class about which you have only the following specification of a constructor.

```
public class Riddle {

  /** Creates a new Riddle object.
   *  @param c is used in mysterious ways
   *  @exception IllegalArgumentException if c is not
   *  an acceptable input.
   */
  public Riddle (char c) {/* body of constructor unknown */}

  /* rest of Riddle class unknown */
}
```

Write a method that constructs a *Vector* of *Riddle* objects according to the specification below. The elements of the result may be in any order. (Your solution need not use every line.)

```
/** Creates a Vector of Riddle objects from the given String.
 *  @param s contains the characters used to construct
 *      each Riddle object
 *  @return a Vector of Riddle objects, one Riddle for each
 *      character in s that does not result in an
 *      IllegalArgumentException when passed to the
 *      Riddle constructor.
 */
public static Vector toRiddleVector (String s) {
  Vector result = new Vector();
  for (int i = 0; i < s.length(); i++) {
```

_____

_____

_____

_____

_____

_____

```
    }
    return result;
}
```

Name: _____ Login: _____

Account class from the Lab 2 solution. (Some methods simplified for brevity, but the simplifications do not affect your answers.) You may detach this page from your exam.

```
/** This class represents a bank account whose current balance is a
 *  non-negative amount in US dollars.
 */
public class Account {

    /** Construct an account with the given initial balance.
     */
    public Account (int balance) {
        myBalance = balance;
        myParent = null;
    }

    /** Construct an account with the given balance and parent account.
     * @param balance the initial balance; balance must be >= 0.
     * @param parent a parent account from which overdrafts will be taken.
     */
    public Account (int balance, Account parent) {
        myBalance = balance;
        myParent = parent;
    }

    /** Deposit into this account.
     * @param amount is the amount to be deposited; it must be non-negative.
     */
    public void deposit (int amount) {
        myBalance = myBalance + amount;
    }

    /** Subtract the amount from the account, if possible.  If it would leave
     *  a negative balance, print an error and leave the balance unchanged.
     *  @param amount is the amount to withdraw.
     */
    public Boolean withdraw (int amount) {
        if (myBalance < amount) {
            return false;
        } else {
            myBalance = myBalance – amount;
            return true;
        }
    }

    /** Get the balance.
     *  @return the number of dollars in the account.
     */
    public int balance ( ) {
        return myBalance;
    }

    private int myBalance;
    private Account myParent;
}
```