Name _____

CIS 180 – Object-Oriented Programming I
Practice Examination #1

1) (5 pts) Give the binary representation of the decimal number 11: `1011`
2) (5 pts) Give the binary representation of the decimal number 22: `10110`
3) (5 pts) Give the sum of the binary numbers 0101 and 0011 in binary: `1000`
4) (5 pts) What keyword in Java is used to define inheritance? `extends`
5) (5 pts) What operator or keyword is used in Java to create objects? `new`

6) (5 pts) Give a brief definition of the term *inheritance*:

   A mechanism for reusing code by defining a new class based on the attributes and operations of an existing class.

7) (5 pts) Give a brief explanation of what is meant by *type polymorphism*:

   The ability of variables and parameters to hold objects not only of the declared type, but also objects belonging to any subclass of the declared type.

8) (5 pts) Briefly describe the relationship between *classes* and *objects* in an object oriented program:

   A class is a template for creating objects of a given type. The class must be defined before any objects are created.

9) (5 pts) Briefly describe the relationship between *methods* and *messages* in an object oriented program:

   Methods define the way that objects respond to messages. A message can only be sent to an object if the object has a corresponding method.

10) Consider a java *Account* class for bank account objects:

- Each account object should have an account number and a balance.
- There should be a constructor with account number and balance parameters.
- There should be accessor methods *getAccountNumber*, *getBalance,* and *setBalance*.
- There should be a *deposit* method with an amount parameter that adjusts the balance according to the amount deposited.
- There should be a *withdraw* method for withdrawing money from the account.

The constructor method and the deposit and with withdraw methods should print "receipts" on the console. If a request is made to withdraw more money than is in the account, an error message should be printed and the account balance left unchanged.

The following example shows how the Account class could be used in an application:

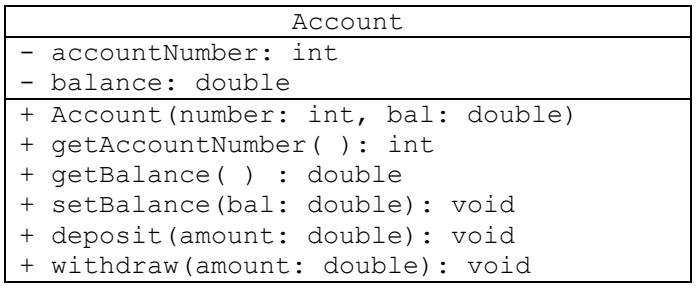
```
public class BankingApplication
{
     public static void main(String[] args)
     {
          Account a1 = new Account(101, 500);
          a1.deposit(50);
          a1.withdraw(100);
          a1.withdraw(800);
          a1.deposit(20);
     }
}
```

The example application shown above should produce the following output:

```
Opened account 101 with initial balance of $500.00

Deposited $50.00 in account 101
The new balance is: $550.00

Withdrew $100.00 from account 101
The new balance is: $450.00

Error: Insufficient funds for
withdrawal of $800.00 from account 101

Deposited $20.00 in account 101
The new balance is: $470.00
```

a. (10 pts) Draw a UML class diagram for the Account class showing its attributes and methods. Include all details (types, parameters, accessibility):

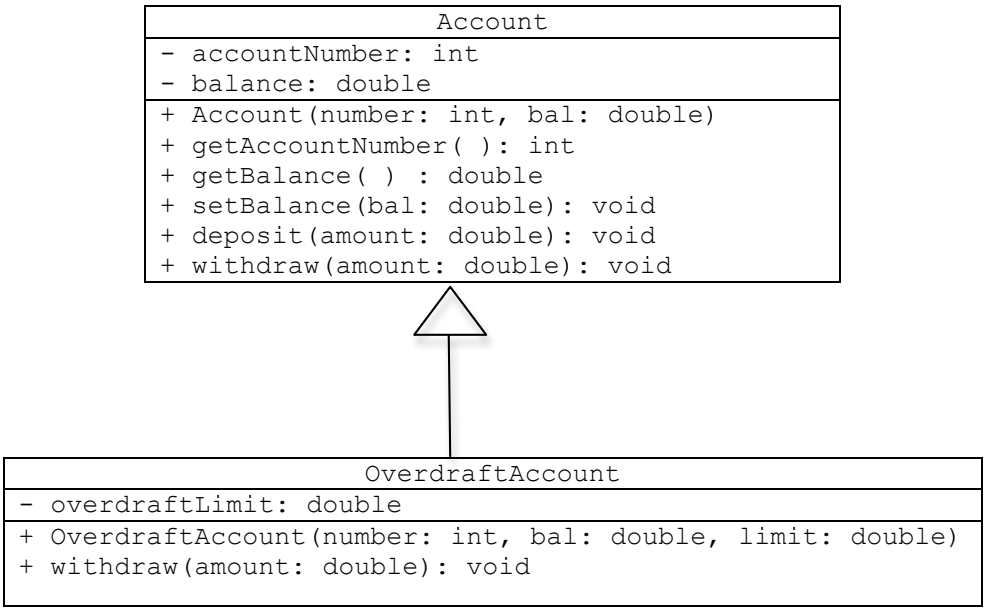
b. (25 pts) Write a Java implementation of the Account class:




```
public class Account {
  private int accountNumber;
  private double balance;

  public Account(int number, double bal) {
    accountNumber = number;
    balance = bal;
    System.out.print("Opened account " + number);
    System.out.println(" with initial balance of $" + balance);
  }

  public int getAccountNumber() {
    return accountNumber;
  }

  public double getBalance() {
    return balance;
  }

  public void setBalance(double bal) {
    balance = bal;
  }

  public void deposit(double amount) {
    balance = balance + amount;
    System.out.print("Deposited $" + amount);
    System.out.println(" in account " + accountNumber);
    System.out.println("The new balance is: " + balance);
  }

  public void withdraw(double amount) {
    if (amount <= balance) {
      balance = balance - amount;
      System.out.print("Withdrew $" + amount);
      System.out.println(" from account " + accountNumber);
      System.out.println("The new balance is: " + balance);
    }
    else {
      System.out.println("Error: Insufficient funds for ");
      System.out.print("withdrawal of " + amount);
      System.out.println(" from account " + accountNumber);
    }
  }
}
```

9) Consider an *OverdraftAccount* class. Overdraft accounts are similar to regular accounts except that they can be overdrawn (have a negative balance) up to a preset limit. The constructor for OverdraftAccounts should take the account number, initial balance, and overdraft limit as parameters.

   a. (5 pts) Add the OverdraftAccount class to the UML class diagram you drew in problem 10. Show the relationship between the two classes on the diagram.



b. (15 pts) Write a Java implementation of the OverdraftAccount class:

Note: The following "solution" is not quite correct, but is good enough for full credit on the exam. We will discuss the problems with this code and the correct solution in class.

```
public class OverdraftAccount extends Account {
  private double overdraftLimit;

  public OverdraftAccount(int number, double bal, double limit) {
    accountNumber = number;
    balance = bal;
    overdraftLimit = limit;
    System.out.print("Opened account " + number);
    System.out.println(" with initial balance of $" + balance);
  }


  public void withdraw(double amount) {
    if (amount <= balance + overdraftLimit) {
      balance = balance - amount;
      System.out.print("Withdrew $" + amount);
      System.out.println(" from account " + accountNumber);
      System.out.println("The new balance is: " + balance);
    }
    else {
      System.out.println("Error: Insufficient funds for ");
      System.out.print("withdrawal of " + amount);
      System.out.println(" from account " + accountNumber);
    }
  }
}
```