Letterkenny Institute of Technology

Course code: OOPR K6A03

BACHELOR OF SCIENCE IN COMPUTING
WITH
COMPUTER SECURITY & DIGITAL FORENSICS
COMPUTER GAMES DEVELOPMENT

BACHELOR OF SCIENCE (Honours)
IN APPLIED COMPUTING

Subject: Object Oriented Programming **Stage:** 2

Date: Autumn 2012 **Examiners:** Mr. D. Hegarty
Ms. O. McMahon

Time allowed: 2 hours

INSTRUCTIONS

Answer any **THREE** questions

NOTE: It may be useful to remove the appendices from the questions portion of the paper – i.e. you can have the questions side-by-side with the relevant code/diagrams, and don't need to go back and forth.

Question 1

(a) Give an account of arrays and ArrayLists in Java. Clearly distinguish between both. Use diagrams and examples in your answer where necessary. (7 marks)

(b) For the BankAccount class in Appendix A, add code to ensure that the balance never goes below zero.

(3 marks)

(c) Implement a Java class *Bank* that has an ArrayList reference to *BankAccount* objects as its only instance field.
See Appendix A for the BankAccount class.

Your class should have the following:-

(1) A default constructor that constructs an empty ArrayList

(2) A method to add an account to the Bank

(3) A method that returns the total money in the Bank

(10 marks)

(d) Write the main method of a tester class which

(1) creates a new Bank object

(2) Adds two accounts to the Bank with initial balances of 90.50 and 20.00

(3) Prints the total in the Bank using the method in part (3) of (c)

(5 marks)

Question 2

(a) "*All OOP languages, including Java, share 3 common defining traits*". Describe what these traits are, and explain the main features in Java that provide these traits. (12 marks)

(b) Show how you would declare a static instance field called `numberOfAccounts` in the `BankAccount` class.

How is this different from a non-static instance field?

(5 marks)

(c) Provide Java code to:-

- Add an extra integer instance field to hold the bank account number to the BankAccount class and a method to access it.
- Add a static instance field and use it to ensure that the first account number is 1001, the next is 1002 and so on, so that every account has a unique number.

(8 marks)

Question 3

Given the skeleton code for the CheckingAccount class in Appendix B, answer the following:

(a) "Shadowing of instance fields is a common mistake for programmers who are new to inheritance". Explain what this means. (4 marks)

With the above statement in mind, correct the mistake that is contained in the instance fields declared in the CheckingAccount class. (2 marks)

(b) Supply the code for the empty methods in the Checking Account class. (6 marks)

(c) Give an illustration of how the `super` keyword prevents infinite recursion. (4 marks)

(d) For the tester code given at the bottom of the page :-

- add code to print the total amount of money in harrysChecking at the end. (2 marks)
- calculate the total amount of money in harrysChecking at the end. (2 marks)

(e) We can assign a superclass reference to a subclass object. Briefly explain why this may be desirable. (5 marks)

```
public class AccountTester
{
   public static void main(String[] args)
   {
      CheckingAccount harrysChecking
            = new CheckingAccount(100);

      harrysChecking.withdraw(80);
      harrysChecking.deposit(50);
      harrysChecking.deposit(45);
      harrysChecking.deposit(16);
      harrysChecking.withdraw(50);
      // Simulate end of month
      harrysChecking.deductFees();

      // Print out how much is in the account
   }
}
```

Question 4

(a) Given the Sorter class in Appendix C write a small tester class that initially has an array of six unique values (either randomly generated or hard-coded), and invokes the bubbleSort method to sort the array.
(5 marks)

(b) Using a simple integer-based example, explain (using pseudo-code or otherwise) how the selection sort works.
(8 marks)

(c) Explain why a sorted group of elements is an important advantage to the efficiency of a search. Your answer should include a description of one search algorithm that uses this advantage.
(7 marks)

(d) For an array with 15 elements, give the worst-case scenarios (i.e. in terms of how many passes we must make through the array) for a sequential search of unsorted elements, and a binary-search of sorted elements. Assume that the search item is not present in the array.
(5 marks)

Question 5

With reference to the code in Appendix D (Animal classes), answer the following:

(a) Explain what run-time polymorphism is. Your answer may provide examples related to the classes in the appendix, or to some other classes of your own choice.

(10 marks)

(b) Describe briefly what a Java interface is. (4 Marks)

(c) List the changes required to Appendix D to incorporate a Talkative interface, which specifies the talk() method. Note, your solution should continue to have an abstract class.

(6 marks)

(d) Describe one potential advantage of using the Talkative interface solution over the non-interface solution.

(5 marks)

Appendix A – BankAccount class

```
/**
   A bank account has a balance that can be changed by
   deposits and withdrawals.
*/
public class BankAccount
{

   // declare instance variables
   private double balance;

   /**
      Constructs a bank account with a zero balance
   */
   public BankAccount()
   {
      balance = 0;
   }

   /**
      Constructs a bank account with a given balance
      @param initialBalance the initial balance
  */
   public BankAccount(double initialBalance)
   {
      balance = initialBalance;
   }

   /**
      Gets the current balance of the bank account.
      @return the current balance
   */
   public double getBalance()
   {
     return balance;
   }

   /**
      Deposits money into the bank account.
      @param amount the amount to deposit
   */
   public void deposit(double amount)
   {
      balance = balance + amount;
   }

//Continued...
```

```
   /**
      Withdraws money from the bank account.
      @param amount the amount to withdraw
  */
   public void withdraw(double amount)
   {

        balance = balance - amount;
   }
}
```

Appendix B – CheckingAccount class

```
/**
   A checking account that charges transaction fees.
*/
public class CheckingAccount extends BankAccount
{
   /**
      Constructs a checking account with a given balance.
      @param initialBalance the initial balance
   */
   public CheckingAccount(double initialBalance)
   {
   }

   /**
      Deposit into the account. This is a transaction.
      @param amount the amount to deposit
   */
   public void deposit(double amount)
   {
   }

   /**
      Withdraw from the account. This is a transaction.
      @param amount the amount to withdraw
   */
   public void withdraw(double amount)
   {
   }

   /**
      Deducts the accumulated fees and resets the
      transaction count.
   */
   public void deductFees()
   {
      if (transactionCount > FREE_TRANSACTIONS)
      {
         double fees = TRANSACTION_FEE *
               (transactionCount - FREE_TRANSACTIONS);
         super.withdraw(fees);
      }
      transactionCount = 0;
   }

   private int transactionCount;
   private double balance;


   private static final int FREE_TRANSACTIONS = 2;
   private static final double TRANSACTION_FEE = 1.5;
}
```

Appendix C – Sorter class

```
/* Sort Utility Class*/

public class Sorter
{
  /**  Uses Selection Sort to sort
  *      an integer array in ascending order
  *    @param the array to sort
  */
  public static void selectionSort( int [] array )
  {
   int max;  // index of maximum value in subarray

    for ( int i = 0; i < array.length; i++ )
    {
      // find index of largest value in subarray
      max = indexOfLargestElement( array, array.length - i );

      //Swap the elements at the index of the largest (max)
      // and the last index in our sub-array (see notes)
      swap(array, max, array.length - i - 1);
     }
  }

  /**  Finds index of largest element
  *    @param    size  the size of the subarray
  *    @ return  the index of the largest element in the subarray
  */
  private static int indexOfLargestElement( int [] array, int size )
  {
    int index = 0;
    for( int i = 1; i < size; i++ )
    {
       if ( array[i] > array[index] )
          index = i;
    }
    return index;
  }

  /**  Swaps 2 elements in a given array
  *    @param    array  the array on which we are to perform the swap
  *    @param    index1  the location of the 1st element
  *    @param    index2  the location of the 2nd element
  */
  private static void swap( int[] array, int index1, int index2)
  {
    int temp = array[index1];
    array[index1] = array[index2];
    array[index2] = temp;
  }



//Continued...
```

```
/**  Performs a Bubble Sort on an integer array,
  *      stopping when array is sorted
  *    @param array to sort
  */
  public static void bubbleSort( int [] array )
  {
     boolean arraySorted = false;

     for ( int i = 0; i < array.length - 1 && !arraySorted;
                                 i++ )
     {
        arraySorted = true;   // start a new iteration;
                              //  maybe the array is sorted

        for ( int j = 0; j < array.length - i - 1; j++ )
        {
           if ( array[j] > array[j + 1] )
           {
              // swap the adjacent elements
              // and set arraySorted to false
              swap(array, j+1, j);

              arraySorted = false; // note that we swapped
           }
        }
     }
  }

}
```

Appendix D – Animal Classes

```
abstract class Animal
{

    abstract void talk();
}

class Dog extends Animal
{

    void talk()
    {
        System.out.println("Woof!");
    }
}

class Cat extends Animal
{

    void talk()
    {
        System.out.println("Meow.");
    }
}

class Interrogator
{

    static void makeItTalk(Animal subject)
    {
        subject.talk();
    }
}
```