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: January 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) Explain how overriding and the flexible use of superclass references provides a framework for run-time polymorphism.

(6 marks)

(b) Provide a class-based example, including a basic tester, which will illustrate run-time polymorphism.

(12 marks)

(c) Explain the importance of using the `instanceof` operator when we attempt to cast a superclass reference to a subclass reference. Provide an appropriate example. **Note** that your example does **not** have to provide full-fledged classes and code – just enough to illustrate the correct usage.

(7 marks)

Question 2

(a) Compare and contrast the *selection sort* and *bubble sort* algorithms. Your answer should include brief descriptions of how each algorithm works and also mention relative strengths and weaknesses of each.

(10 marks)

(b) Given the Sorter class in Appendix A 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.

(4 marks)

(c) Show what changes you would make to the bubbleSort code to allow the algorithm to quit if it determines that no more passes are required.

(5 marks)

(d) For the following array (13 elements), list the indices that BinarySearch will look at when searching for the search-key 70. Your answer should include detail of how the indices are calculated.

[6 9 14 21 30 45 46 51 57 67 72 88 92]

(6 marks)

Question 3

(a) Consider the code below which shows an attempt to deal with non-numeric input from the user.
Explain the weaknesses in this code and provide a solution that will deal with the issue in a more robust fashion.

(7 marks)

Explain briefly how you could incorporate your own exception objects to handle negative integers.

(4 marks)

```
public class ExceptionHandlingAttempt1
{
   public static void main(String[]args)
   {

      String opt;
      int cYear;

      try
      {
         opt = JOptionPane.showInputDialog(null, "Enter the
                                           year");

         cYear =  Integer.parseInt(opt);

      }
      catch(NumberFormatException e)
      {
         opt = JOptionPane.showInputDialog(null,
               "Input must be an integer!\nEnter the year");
         cYear =  Integer.parseInt(opt);
      }

      JOptionPane.showMessageDialog(null, "You entered " +
                                           cYear);

   }
}
```

(b) Briefly explain why wrappers are required in Java.

(3 Marks)

(c) Explain, using an example, what auto-boxing is in Java. Your answer should make reference to auto-unboxing also.

(4 marks)

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

Question 4

(a) With reference to the SuperClass and SubClass classes and associated tester in Appendix B, answer the following:

(I) *Shadowing of instance fields is a common beginner mistake when implementing inheritance*. Given that statement, can you suggest what may be wrong with the code provided?

(2 marks)

(II) What is the expected output? What is the actual output?

(4 marks)

(III) Reproduce the diagram below in your answer book and show the values of each instance field for `ob1` as they would be in the tester at the end of main().

(4 marks)

x (subclass):
x (superclass):
y (subclass):

(IV) Specify the relevant changes required to the code, so that the addXtoY() method will work properly. Note: you must not make any instance fields public.

(5 marks)

(b) Explain what static methods and static variables are and provide a simple example to illustrate their usage,
e.g. a class which keeps track of how many objects of the class are created, has an associated accessor method and also uses this information to assign a unique i.d. to each object.

(10 marks)

Question 5

Appendix C contains a Book class, a BookStoreTester class and sample output for partial title matching.

(a) Provide a BookStore class. The class should initially contain:

- An ArrayList<Book> reference called **books**.

(2 marks)

- An **addBook** method which will accept a book object as a reference. It will add the book to the ArrayList.

(3 marks)

- A method called **listAll** which uses an enhanced for loop to iterate over the ArrayList printing each book's details to System.out

(4 marks)

(b) Add a **searchByTitle** method to the BookStore class which will provide the following:

- It should have the signature :
  ```
  public ArrayList<Book> searchByTitle(String searchStr)
  ```
- The method should allow partial matches – hint: the String class has a `contains()` method which takes a string argument and returns true/false depending on whether the current string contains the argument string.
- The method should return all books which match the *searchStr*. If there are no matches it should return an empty list.

(12 marks)

(c) If the Book class did not override the `toString()` method, what effect would this have on the output from the tester?

(4 marks)

Appendix A – 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
  *
  *    @param array to sort
  */
  public static void bubbleSort( int[] array )
  {

     for ( int i = 0; i < array.length - 1; i++ )
     {
        arraySorted = true;   // start a new iteration;

        for ( int j = 0; j < array.length - i - 1; j++ )
        {
           if ( array[j] > array[j + 1] )
           {

              swap(array, j+1, j);

           }
        }
     }
  }

}
```

Appendix B – SuperClass, SubClass and associated tester

```
public class SuperClass
{
   private int x;

   public SuperClass(){
      x = 0;
   }

   public SuperClass(int val){
      x = val;
   }
}

public class SubClass extends SuperClass
{
   private int y;
   private int x;

   public SubClass()
   {
      y = 0;
   }

   public SubClass( int val1, int val2)
   {
      super(val1);
      y = val2;
   }

   public void addXtoY()
   {
      y = y+ x;
   }

   public int getX()
   {
      return x;
   }

   public int getY()
   {
      return y;
   }
}

public class SuperClassSubClassTester
{
   public static void main(String args[])
   {
      //create object with x=4 and y =9
      SubClass ob1 = new SubClass(4,9);

      ob1.addXtoY();

      System.out.println("X: " + ob1.getX() + ", Y: " +
                         ob1.getY());

   }
}
```

Appendix C : Part 1 – Book class

```
public class Book
{
   private String title;
   private String author;
   private double price;

   public Book(String title, String author, double price)
   {
      this.title = title;
      this.author = author;
      this.price = price;
   }

   public String getAuthor()
   {
      return author;
   }

   public String getTitle()
   {
      return title;
   }

   public double getPrice()
   {
      return price;
   }

   public String toString()
   {

      return ("title: " + title + "\t"
            + "author: " + author + "\t"
            + "price: " + price + "\n");
   }
}
```

Appendix C : Part 2 – BookStoreTester class

```
public class BookStoreTester
{
  public static void main(String args[])
  {
    BookStore theBookShop = new BookStore();

    theBookShop.addBook(new Book("Big Java", "Horstmann", 38.99));
    theBookShop.addBook(new Book("The Corrections", "Franzen", 12.99));
    theBookShop.addBook(new Book("Five Go Camping", "Enid Blyton", 7.90));
    theBookShop.addBook(new Book("Head First Java", "Kathy Sierra", 42.50));

    ArrayList<Book> searchResult;

    searchResult = theBookShop.searchByTitle("Java");

    System.out.println(searchResult);

  }
}
```

Appendix C : Part 3 – Sample output from BookStoreTester