Letterkenny Institute of Technology

Course code: OOPR CP603

YEAR 2 COMPUTING

(Common paper for all streams)

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

Date: Autumn 2013 **Examiners:** Mr. D. Hegarty
Ms. O. McMahon

Time allowed: 3 hours

INSTRUCTIONS
Answer any **FOUR** questions. All questions carry equal marks.

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

Appendix A 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)

Question 2

Appendix B consists of two parts: A Sorter class with selectionSort and bubbleSort methods; A Searcher class with a binarySearch method.

(a) The bubbleSort algorithm could potentially quit sorting if it realised that there were no swaps on the previous pass. Amend the method to provide this functionality (hint: use a boolean flag)

(6 marks)

(b) State what changes are required to the code to make the bubbleSort code sort in descending order.

(3 marks)

(c) For a list with 7 elements in it, determine how many comparisons would be required by the selectionSort algorithm.

(3 marks)

(d) For the Searcher class, provide a sequentialSearch method which will search for a given key in a **sorted** array (of type int). The method should return the index of the key or -1 if the key is not found.
Note: The method should quit searching when appropriate.

(6 marks)

(e) For the following array (13 elements), determine the indices that binarySearch will look at when searching for the search-key 70.

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

Your answer should identify what occurs on each pass in terms of the variables **`start, mid, end`**.

(7 marks)

Question 3

Distinguish between the Comparable interface and the Comparator interface.
Your answer should provide appropriate examples of class code which implements each interface and examples of their usage using the Collections class.

(25 marks)

Question 4

Distinguish between composition and inheritance in java. Provide an example, including test code, which demonstrates both, i.e. a superclass plus a subclass which have composition incorporated into them.

(17 marks)

Distinguish between method overloading and method overriding. Provide appropriate examples to illustrate your answer.

(8 marks)

Question 5

Given the BankAccount class and skeleton code for the CheckingAccount class in Appendices C and D, 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) Provide a method in the BankAccount class called `transfer` that will allow for transfer of money from one BankAccount to another (5 marks)

```
public class AccountTester
{
   public static void main(String[] args)
   {
      CheckingAccount harrysChecking
            = new CheckingAccount(120);

      harrysChecking.withdraw(70);
      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 6

Given the Person class below, answer the subsequent questions:

```
public class Person
{
   private String name;
   int age;

   public Person()
   {
   }
   public Person(String inName, int inAge)
   {
      name = inName;
      age = inAge;
   }

   public void printDetails()
   {
      System.out.print("Name: " + name + " Age: " + age);
   }
}
```

(a) Provide the code for a Student class which is a subclass of Person. Your class should contain :-

- An extra `int` instance field that stores the students ID number.
- A constructor that accepts three parameters – the student's name, the student's age, and the student's ID number.
- A version of `printDetails` that overrides the version in the superclass and prints out all the student's details.

(8 marks)

(b) Provide a tester class that :-

- Contains an ArrayList of references to Person objects
- Adds two Person objects and one Student object to the ArrayList
- Uses the ArrayList to demonstrate run-time polymorphism. Your answer should also explain why the code is polymorphic.

(10 marks)

(c) Explain why you should not use the relational operator == when comparing objects. What facility does Java provide instead, and how is it used?

(7 marks)

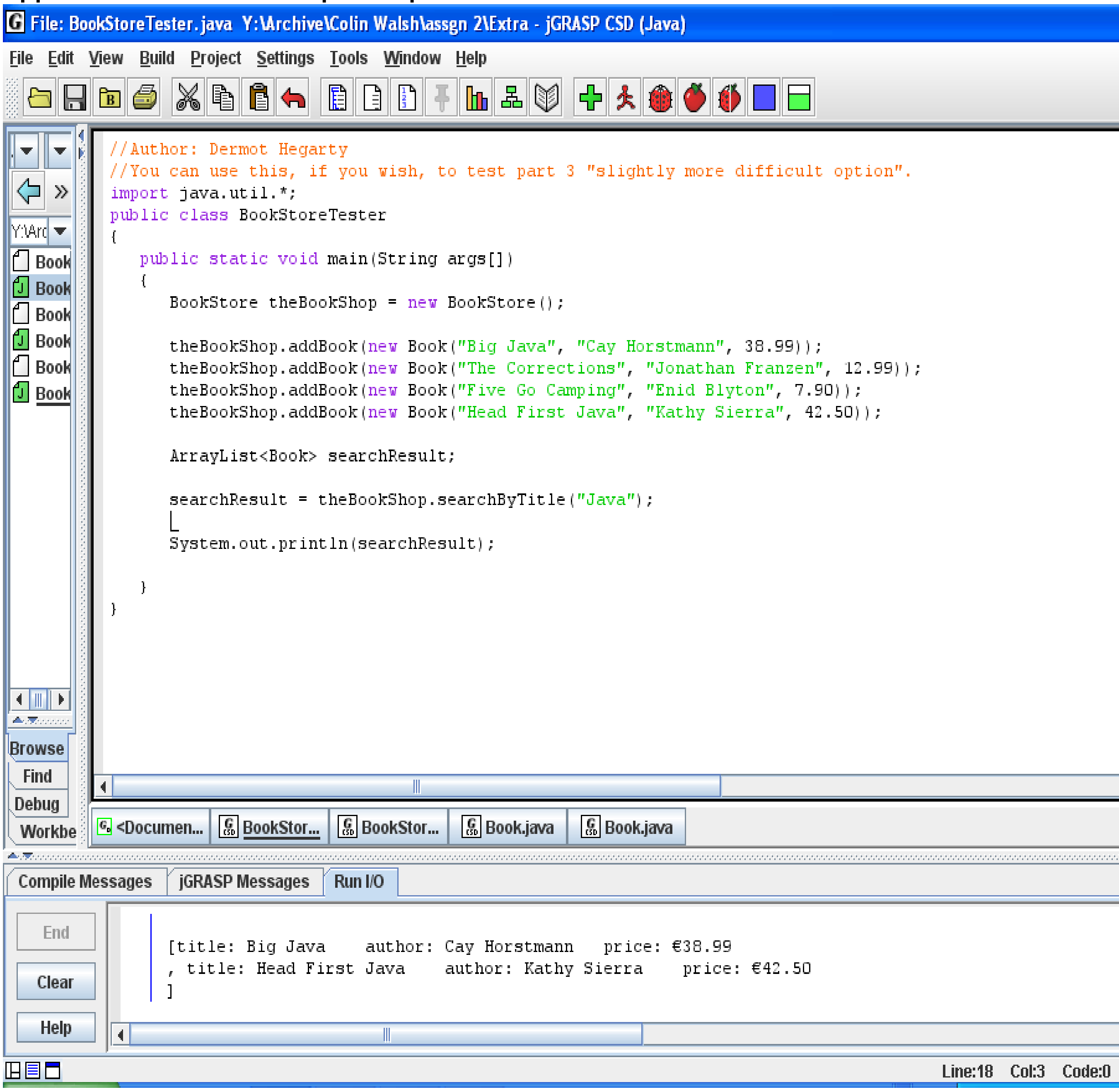
Appendix A : Part 1 – Book class

```
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 A : 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 A : Part 3 - Sample output from BookStoreTester


File: BookStoreTester.java Y:\Archive\Colin Walsh\assgn 2\Extra - jGRASP CSD (Java)
File Edit View Build Project Settings Tools Window Help
Browse
Find
Debug
Workbe
<Documen... BookStor... BookStor... Book.java Book.java
Compile Messages jGRASP Messages Run I/O
End
Clear
Help
Line:18 Col:3 Code:0

```
//Author: Dermot Hegarty
//You can use this, if you wish, to test part 3 "slightly more difficult option".
import java.util.*;
public class BookStoreTester
{
   public static void main(String args[])
   {
      BookStore theBookShop = new BookStore();

      theBookShop.addBook(new Book("Big Java", "Cay Horstmann", 38.99));
      theBookShop.addBook(new Book("The Corrections", "Jonathan 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);

   }
}
```

```
[title: Big Java    author: Cay Horstmann    price: €38.99
, title: Head First Java    author: Kathy Sierra    price: €42.50
]
```

Appendix B: Part 1 - 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);
     }
  }

  /**  Performs a Bubble Sort on an integer array,
  *     Note: this version does not stop once the array is sorted
  *    @param array to sort
  */
  public static void bubbleSort( int [] array )
  {
     for ( int i = 0; i < array.length – 1; i++ )
     {

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


//Continued...
```

```
  /**  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;
  }

}
```

Appendix B: part 2 - Searcher class

```
public class Searcher
{

  //This method will return the index of the searchItem in the array
  //If it doesn't find the searchItem, it will return -1 to indicate
  //this
  public static int binarySearch(int[] list, int searchItem)
  {
    int start=0;
    int end = list.length - 1;

    int mid = 0;

    boolean found = false;

    //Loop until found or end of list.
    while(start <= end && !found)
    {
        mid = (start + end) /2;

        if(list[mid] == searchItem)
        {
          found = true;
        }
        else
        {
            if(list[mid] > searchItem)
            {
               end = mid -1;
            }
            else
            {
               start = mid + 1;
            }
        }
    }

    if(found)
    {
       return mid;
    }
    else
    {
       return(-1);
    }
  }
}
```

Appendix C - 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 D – CheckingAccount class

```
/**
   A checking account that charges transaction fees.
*/
public class CheckingAccount extends BankAccount
{
   private int transactionCount;
   private double balance;

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

   /**
      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;
   }
}
```