

Letterkenny Institute of Technology

Course code: OOPR CP603

YEAR 2 COMPUTING

(Common paper for all streams)

Subject: Object Oriented Programming

Stage: 2

Date: Autumn 2016

Examiners: Mr. D. Hegarty
Dr. A. Belatreche

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

A basic Person class is given below. Answer the subsequent questions based on this class.

Note 1: You may find it useful to refer to appendix A for this question.

Note 2: You can assume that all code snippets are part of a valid tester class containing a `main()` method.

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

    public Person(String name, int age)
    {
        this.name = name;
        this.age = age;
    }
}
```

a) Override the `toString()` method.

(2 marks)

b) Amend the Person class so the code snippet below will work properly

```
ArrayList<Person> people = new ArrayList<Person>();

//Assume Person objects have been added to the list

if (people.contains(new Person("Adam Ant", 48)))
{
    //Do something
}
else
{
    //Do something else
}
```

(8 marks)

c) Assuming that I want to use the `sort` method of the `Collections` class to sort the `people` list alphabetically by name -

`Collections.sort(people);` - provide the required changes to the Person class.

(6 marks)

d) If I also wish to provide the additional capability of sorting by age, provide a mechanism that will allow this additional sorting capability. Show how you would invoke the `Collections.sort()` method for this.

(9 marks)

Question 2

a) What is wrong with the following interface?

```
public interface SomethingIsWrong {
    void aMethod(int aValue) {
        System.out.println("Hi Mom");
    }
}
```

(3 marks)

b) Explain what is required to fix the interface in part a).

(2 marks)

c) Distinguish between an abstract class and an interface.

(5 marks)

d) Explain the consequence of a subclass not overriding a superclass's abstract method.

(3 marks)

e) Using the information in Appendix B, create a Student class which is capable of doing part-time jobs.

The Student class should have the following instance fields:
name (String), id (int), course(String), spendingMoney (double).

In addition it should have the following methods:
an overloaded constructor; a method with the signature
`public int buyNoodles (double pricePerPacket)`.
This method should return the how many packets of noodles the student can buy.

Provide a snippet of test-code showing how a student object could be used.

Note: You should assume that the Job class is fully coded, i.e. you don't have to write this class.

(12 marks)

Question 3

Appendix C 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)

(5 marks)

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

(2 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.

(5 marks)

- (e) For the following array (13 elements), determine the indices that binarySearch will look at when searching for the search-key 33.

[2, 5, 7, 13, 15, 16, 18, 20, 22, 24, 30, 34, 39]

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

(7 marks)

- (f) How many passes are required in a binarySearch for an array/list containing 40 elements? You should explain your answer.

(3 marks)

Question 4

Given the BankAccount class and skeleton code for the CheckingAccount class in Appendix 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(125);

        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 5

The following class is given:

```
public class Swapper
{
    static void swap(ArrayList<Integer> list, int index1, int index2)
    {
        int temp = list.get(index1);
        list.set(index1, list.get(index2));
        list.set(index2, temp);
    }
}
```

- a) Provide valid test code which calls the `swap()` method and prints out the subsequent list.
(5 marks)

- b) Add a try-catch block to the method which will handle the obvious potential for out-of-bounds errors.
Note: your catch block should simply report the error, using the exception object's built-in facilities.
Also, you can just use the general Exception class for your error handling, if you wish.
(5 marks)

- c) Provide a basic example to illustrate the usage of the *enum type* in Java.
(8 marks)

- d) Show the modifications required to the BankAccount class in Appendix D so that the BankAccount objects can store an automatically generated account number.
Your answer should include the following:
 1. The account numbers should start at 1000.
 2. Use of a static variable and an accessor method to return an object's account number.

(7 marks)

Question 6

Appendix E 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.
(10 marks)

(c) Explain briefly (you do not have to provide the actual code) the modifications required to have a "search by genre" facility.

(6 marks)

Appendix A - Information on the Comparable and Comparator interfaces (Question 1)

Interface Comparable<T>

Type Parameters:

T - the type of objects that this object may be compared to

Method Summary

Methods	
Modifier and Type	Method and Description
int	<code>compareTo(T o)</code> Compares this object with the specified object for order.

This is how the Comparable interface would be written:

```
public interface Comparable<T>
{
    public int compareTo(T o);
}
```

Interface Comparator<T>

Type Parameters:

T - the type of objects that may be compared by this comparator

Method Summary

Methods	
Modifier and Type	Method and Description
int	<code>compare(T o1, T o2)</code> Compares its two arguments for order.
boolean	<code>equals(Object obj)</code> Indicates whether some other object is "equal to" this comparator.

Appendix B - PartTimeAble interface and Job class (Question 2)

Interface PartTimeAble

```
public interface PartTimeAble
```

The interface is designed for short-term jobs. Employees, Managers, and Students can all earn additional money via this interface. The specifics of this are left to each implementing class

Method Summary

Methods

Modifier and Type	Method and Description
void	<code>doJob(Job j)</code> what this method will entail differs from class to class

Method Detail

doJob

```
void doJob(Job j)
```

what this method will entail differs from class to class

Class Job

```
java.lang.Object
  Job
```

```
public class Job
extends java.lang.Object
```

Encapsulates information about a job. Note: can be used as standalone or in conjunction with the PartTimeAble interface

Field Summary

Fields

Modifier and Type	Field and Description
private java.lang.String	<code>jobDescription</code> A short description of the job, eg "temp office work"
private double	<code>rate</code> rate is hourly rate
private double	<code>time</code> time spent doing the job

Constructor Summary

Constructors

Constructor and Description
<code>Job(java.lang.String jobDescription, double rate, double time)</code>

Method Summary

Methods

Modifier and Type	Method and Description
java.lang.String	<code>getJobDescription()</code>
double	<code>getPrice()</code> Price is calculated as a product of rate and time
double	<code>getRate()</code>
double	<code>getTime()</code>
java.lang.String	<code>toString()</code>

Appendix C: Part 1 - Sorter class (Question 3)

```

/* 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 C: 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 D - BankAccount class (Question 4 and 5)

```

/**
 * 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;
    }

    /**
     * Withdraws money from the bank account.
     * @param amount the amount to withdraw
     */
    public void withdraw(double amount)
    {
        balance = balance - amount;
    }
}

```

Appendix D - CheckingAccount class (Question 5)

```

/**
 * 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;
    }
}

```

Appendix E : Part 1 - Book class (Question 6)

```

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 E : 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 E : 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

```
//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);
    }
}
```

Book
Book
Book
Book
Book

Browse
Find
Debug
Workbe

<Documen... BookStor... BookStor... Book.java Book.java

Compile Messages jGRASP Messages Run I/O

End
Clear
Help

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

Line:18 Col:3 Code:0