# Description: _LITfinalLOGO

# SUMMER EXAMINATIONS 2015

**Friday, 8th May 2015, 14.30 p.m. – 16.30 p.m.**

**KSDEV\_8\_Y2**

**Course:** Bachelor of Science (Hons) in Software Development

**Year:** One

**Subject:** Object Oriented & GUI Programming

**Time Allowed:** 2 Hours

**Instructions: 1.** You **MUST** answer **Q.1**

Answer **ANY OTHER TWO** questions.

**2.** Marks for **Q.1** are **50 marks**.

All other questions are **25 marks**.

**3.** Start each question on a new page.

**4.** Write the question number at the top of each page.

**5.** Circle the numbers of the questions you answer at the front of your answer book.

**Additional Attachments Exam Materials to accompany this paper:**

1. **Code for Q.4**

**Internal Examiners: External Examiners:**

Sharon Byrne Mr Brian Gillespie

**Q.1**  Answer any FIVE of the following **(10 marks per part)**

1. Ben Shneiderman's "Eight Golden Rules of Interface Design" are a guide to good interaction design. Write a brief note on each of the following, including examples to illustrate your points.
   1. Rule 1: Strive for consistency
   2. Rule 3: Offer informative feedback
   3. Rule 5: Offer simple error handling
   4. Rule 7: Support internal locus of control
2. Describe the role of the Windows Form Designer in Visual Studio.
3. With reference to the .NET framework, write a brief note on each of the following
4. Assemblies
5. Garbage Collection
6. Describe in detail the role of the following code from a C# program

**BinaryFormatter bformatter = new BinaryFormatter();**

**FileStream sfile = new FileStream("Students.bin", FileMode.Open, FileAccess.Read);**

**ArrayList slist = bformatter.Deserialize(sfile) as ArrayList;**

**sfile.Close();**

1. Describe in detail the role of the following two lines of code from the definition of the class Timer

**public delegate void ElapsedEventHandler(object source, ElapsedEventArgs e);**

**public event ElapsedEventHandler Elapsed;**

1. With respect to control layout in WPF briefly describe each of the following layout containers
2. Canvas
3. DockPanel
4. Grid
5. StackPanel
6. WrapPanel
7. Describe in detail the role of the following code fragment from a C# .NET program

**public class Student : ICloneable**

**{ private string name;**

**private int mark;**

**public Student(string name, int mark)**

**{ this.name = name;**

**this.mark = mark;**

**}**

**public object Clone()**

**{ return MemberwiseClone();**

**}**

**}**

**(Total 50 Marks)**

**Q. 2 Class Hierarchy**

A waged employee’s weekly pay is calculated by multiplying an hourly rate by the number of hours worked. A salaried employee is paid his annual salary divided by 52 each week.

Write suitable definitions in C# for the classes **Employee, Waged** and **Salaried** in the class hierarchy shown below, given the following conditions:

* The **class Employee** should be an abstract base class and the **Waged** and **Salaried** classes should be concrete classes **(3 marks)**
* The **class** **Employee** should have
  + a string field **name** and an integer field **id** **(1 mark)**
  + a constructor that is passed a **name** and an **id** **(2 marks)**
  + get properties for the two fields **(2 marks)**
  + an abstract method **WeeklyPay** which returns a double value

**(1 mark)**

* The **class** **Waged** should have
  + a double field **hourlyRate** and an integer field **hours** **(1 mark)**
  + a constructor that is passed a **name**, **id**, **hourlyRate** and **hours**

**(3 marks)**

* + get and set properties for the fields **(3 marks)**
  + a defined method **WeeklyPay** **(1 mark)**
* The **class** **Salaried** should have
  + a field **annualSalary** of type double **(1 mark)**
  + a constructor that is passed a **name**, **id** and **salary** **(3 marks)**
  + get and set properties for the field **(3 marks)**
  + a defined method **WeeklyPay** **(1 mark)**

##### class Employee

##### class Salaried

##### class Waged

**(Total 25 Marks)**

**Q.3 Equality**

1. Distinguish between the terms ‘reference equality’ and ‘value equality’ for objects of a class. Illustrate your answer with some sample code

**(12 marks)**

1. Some of the recommendations for implementing value equality for a class are
   1. Override the **virtual Object.Equals(obj Object)** method. This should just call the type-specific **Equals** method

**(4 marks)**

* 1. Implement the **IEquatable<T>** interface by providing a type-specific **Equals** method

**(6 marks)**

* 1. Override the **Object.GetHashCode( )** so that two objects with the same fields produce the same hash code

**(3 marks)**

For the **class GridPosition** partially defined below, redefine the class implementing the above recommendations.

**public class GridPosition**

**{**

**private int row;**

**private int col;**

**public GridPosition() {this.row = 0;this.col = 0;}**

**public GridPosition(int row, int col)**

**{this.row = row;this.col = col;}**

**public int Row**

**{**

**get{ return row; }**

**}**

**public int Col**

**{**

**get{ return col; }**

**}**

**}**

**(Total 25 Marks)**

**Q. 4** **Structured Exception Handling**

**a)** Describe the roles of the keywords **try**, **throw**, **catch** and **finally** that are used in the C# language to support the implementation of structured exception handling.

**(8 marks)**

**Attachment A** contains code for a **class Program** with methods **Main** and **Scale**. The method **Scale** may be used to scale a portion of the values in the array **list**. The method **Scale** may throw exceptions as specified in the code, if the method is passed invalid parameters. The method **Main** may also cause two additional different types of exceptions to be thrown.

1. Name the two additional types of exceptions that may be caused to be

thrown by the code in Main and indicate at what lines in the code these exceptions may occur.

**(4 marks)**

1. Re-write the method Main so that it implements structured exception

handling.

**(15 marks)**

**(Total 25 Marks)**

**Attachment A**

**class Program**

**{ static int[] list = { 22, 33, 66, 88, 44, 55, 22, 77, 55, 22 };**

**static void Main(string[] args)**

**{ Console.Write("Enter start index of portion of the list to scale : ");**

**int startIndex = int.Parse(Console.ReadLine());**

**Console.Write("Enter end index of portion of the list to scale : ");**

**int endIndex = int.Parse(Console.ReadLine());**

**Console.Write("Enter a scale factor : ");**

**int scaleFactor = int.Parse(Console.ReadLine());**

**Scale(startIndex, endIndex, scaleFactor);**

**Console.Write("Enter new value to insert in the list : ");**

**int newValue = int.Parse(Console.ReadLine());**

**Console.Write("Enter index where to insert the value : ");**

**int insertionIndex = int.Parse(Console.ReadLine());**

**list[insertionIndex] = newValue;**

**}**

**static public void Scale(int startIndex, int endIndex, int scaleFactor)**

**{ if (startIndex < 0)**

**throw ( new ArgumentOutOfRangeException("startIndex", startIndex, "Negative start index not allowed"));**

**if (endIndex > 9)**

**throw ( new ArgumentOutOfRangeException("endIndex", endIndex, "End index out of range"));**

**if (endIndex < startIndex)**

**throw ( new Exception("End index less than start index"));**

**if (scaleFactor <= 0)**

**throw ( new ArgumentOutOfRangeException("scaleFactor", scaleFactor, "Only positive scale factors allowed"));**

**for (int i = startIndex; i <= endIndex; i++)**

**{ list[i] \*= scaleFactor;**

**}**

**}**

**}**