|  |
| --- |
| LAB MANUAL |
| Object Oriented Programming |
| COURSE CODE: CSC-210 |

|  |
| --- |
| Department of Software Engineering  **Bahria University Karachi Campus**  13 National Stadium Road |

## Preface

This lab manual is designed for those readers who wish to start learning to program in an object-oriented programming language. It has been designed primarily as a first programming text. It is also suitable for those who already have some experience with another programming language, and who now wish to move on to an object-oriented one. Indeed, much of the material is based on courses delivered by the author to students with a wide range of both non-programming and programming backgrounds. The language we use to teach object-oriented programming is Java/C++/C#.

This laboratory guide is intended to facilitate understanding of the widely used Object Oriented Techniques such as polymorphism, abstraction, inheritance and encapsulation.

The language used for implementations is C++/C#/Java. In implementing the assignments a good programming style is very important, therefore some guide is given in Appendix A.

This guide is intended to be used by the students of Computer & Software Engineering Department of Bahria University Karachi Campus.

*SE Department, Bahria University Karachi Engr. Saniya Sarim*

Table of Contents

[Preface ii](#_Toc449965673)

1. [**INTRODUCTION TO JAVA** 5](#_Toc449965674)

[**Introduction** 6](#_Toc449965675)

[**Time Boxing** 8](#_Toc449965676)

[**Objectives** 8](#_Toc449965677)

[**Lab Tasks/Practical Work** 8](#_Toc449965678)

1. [**IN-DEPTH UNDERSTANDING OF CLASSES AND OBJECTS** 9](#_Toc449965679)

[**Introduction** 10](#_Toc449965680)

[**Objectives** 12](#_Toc449965681)

[**Lab Tasks/Practical Work** 12](#_Toc449965682)

1. [**IN-DEPTH UNDERSTANDING OF ACCESS MODIFIERS** 13](#_Toc449965683)

[**Introduction** 14](#_Toc449965684)

[**Time Boxing** 19](#_Toc449965685)

[**Objectives** 19](#_Toc449965686)

[**Lab Tasks/Practical Work** 20](#_Toc449965687)

1. [**UNDERSTANDING THE USES OF CONSTRUCTOR AND ITS TYPES** 21](#_Toc449965688)

[**Introduction** 22](#_Toc449965689)

[**Time Boxing** 29](#_Toc449965690)

[**Objectives** 29](#_Toc449965691)

[**Lab Tasks/Practical Work** 29](#_Toc449965692)

1. [**UNDERSTANDING THE CONCEPT OF OVERLOADING & EXCEPTION HANDLING** 30](#_Toc449965693)

[**Understanding the concept of Overloading & Exception Handling** 31](#_Toc449965694)

[**Introduction** 31](#_Toc449965695)

[Overloading: 31](#_Toc449965696)

[Exception Handling: 34](#_Toc449965697)

[**Time Boxing** 36](#_Toc449965698)

[**Objectives** 36](#_Toc449965699)

[**Lab Tasks/Practical Work** 37](#_Toc449965700)

1. [**UNDERSTANDING THE IMPLEMENTATION OF STATIC CLASSES & MEMBERS** 38](#_Toc449965701)

[**Understanding the implementation of Static Classes & Members** 39](#_Toc449965702)

[**Introduction** 39](#_Toc449965703)

[**Time Boxing** 40](#_Toc449965704)

[**Objectives** 40](#_Toc449965705)

[**Lab Tasks/Practical Work** 40](#_Toc449965706)

1. [**In-Depth understanding the concept of Inheritance** 41](#_Toc449965707)

[**In-Depth understanding the concept of Inheritance** 42](#_Toc449965708)

[**Introduction** 42](#_Toc449965709)

[**Time Boxing** 52](#_Toc449965710)

[**Objectives** 52](#_Toc449965711)

[**Lab Tasks/Practical Work** 52](#_Toc449965712)

1. [**Implementation & understanding of Multiple Inheritance** 53](#_Toc449965713)

[**Introduction** 54](#_Toc449965714)

[**Time Boxing** 55](#_Toc449965715)

[**Objectives** 56](#_Toc449965716)

[**Lab Tasks/Practical Work** 56](#_Toc449965717)

1. [**Understanding the concept of Composition, Aggregation and Association** 57](#_Toc449965718)

[**Understanding the concept of Composition, Aggregation and Association** 58](#_Toc449965719)

[**Introduction** 58](#_Toc449965720)

[**Time Boxing** 61](#_Toc449965721)

[**Objectives** 61](#_Toc449965722)

[**Lab Tasks/Practical Work** 62](#_Toc449965723)

1. [**Understanding the concept of Polymorphism** 63](#_Toc449965724)

[**Introduction** 64](#_Toc449965725)

[**Time Boxing** 65](#_Toc449965726)

[**Objectives** 65](#_Toc449965727)

[**Lab Tasks/Practical Work** 65](#_Toc449965728)

1. [**Understanding the concept of Abstract Methods and Classes** 66](#_Toc449965729)

[**Understanding the concept of Abstract Methods and Classes** 67](#_Toc449965730)

[**Introduction** 67](#_Toc449965731)

[**Time Boxing** 70](#_Toc449965732)

[**Objectives** 70](#_Toc449965733)

[**Lab Tasks/Practical Work** 70](#_Toc449965734)

1. [**IMPLEMENTATION OF INTERFACES** 71](#_Toc449965735)

[**Introduction** 72](#_Toc449965736)

[**Time Boxing** 73](#_Toc449965737)

[**Objectives** 73](#_Toc449965738)

[**Lab Tasks/Practical Work** 73](#_Toc449965739)

1. [**INTRODUCTION TO GUI BASED PROGRAMMING** 74](#_Toc449965740)

[**Introduction** 75](#_Toc449965741)

[**Time Boxing** 78](#_Toc449965742)

[**Objectives** 78](#_Toc449965743)

[**Lab Tasks/Practical Work** 78](#_Toc449965744)

1. [**GUI BASED CALCULATOR** 79](#_Toc449965745)

[**Introduction** 80](#_Toc449965746)

[**Time Boxing** 80](#_Toc449965747)

[**Objectives** 80](#_Toc449965748)

[**Lab Tasks/Practical Work** 80](#_Toc449965749)

1. [**STATIC & DYNAMIC BINDING** 81](#_Toc449965750)

[**Introduction** 82](#_Toc449965751)

[**Time Boxing** 84](#_Toc449965752)

[**Objectives** 84](#_Toc449965753)

[**Lab Tasks/Practical Work** 84](#_Toc449965754)

1. [References 85](#_Toc449965756)

**Lab Manual for Object Oriented Programming**

**Lab No. 1**

# **INTRODUCTION TO JAVA**

Objectives

To understand basic concept of JAVA using Netbeans.

**LAB # 01**

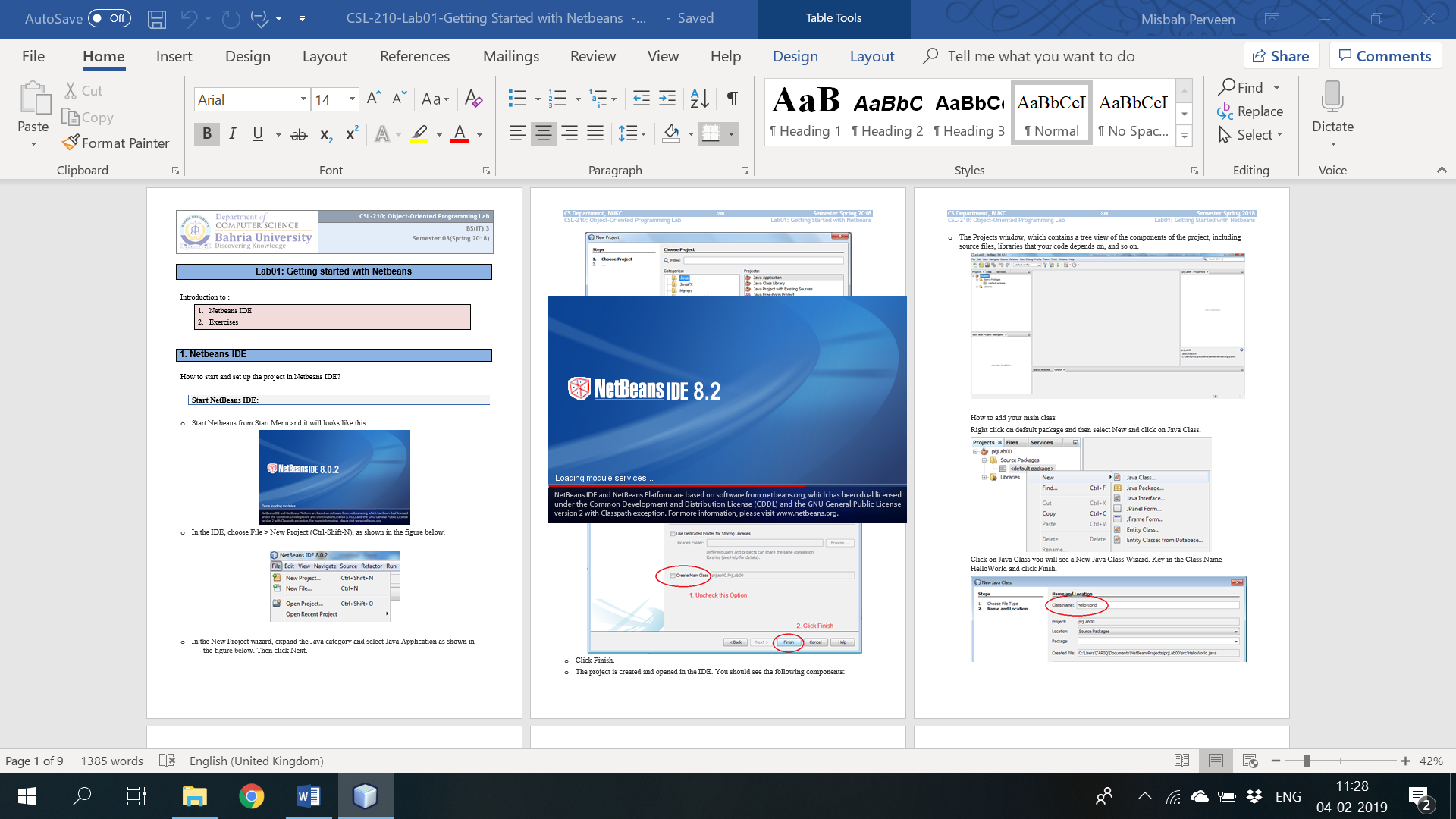
**Introduction to JAVA**

## **Introduction**

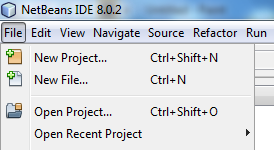
* 1. **Introduction to NetBeans**

To start and set up the project in Netbeans IDE:

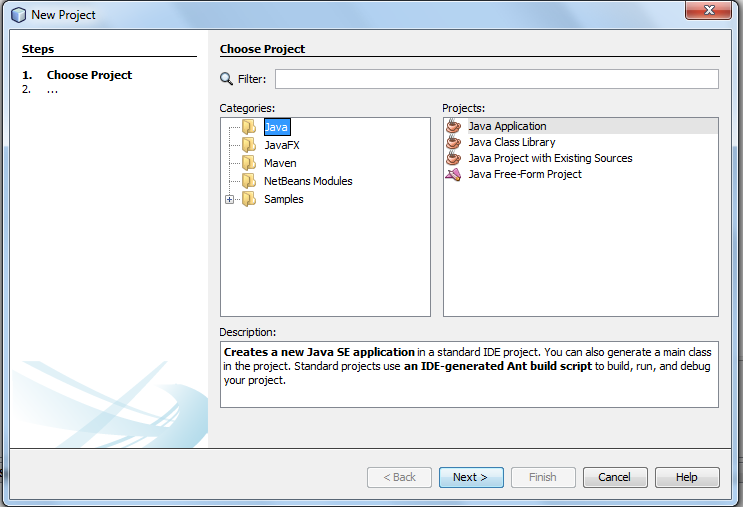
* Start Netbeans from Start Menu and it will looks like this



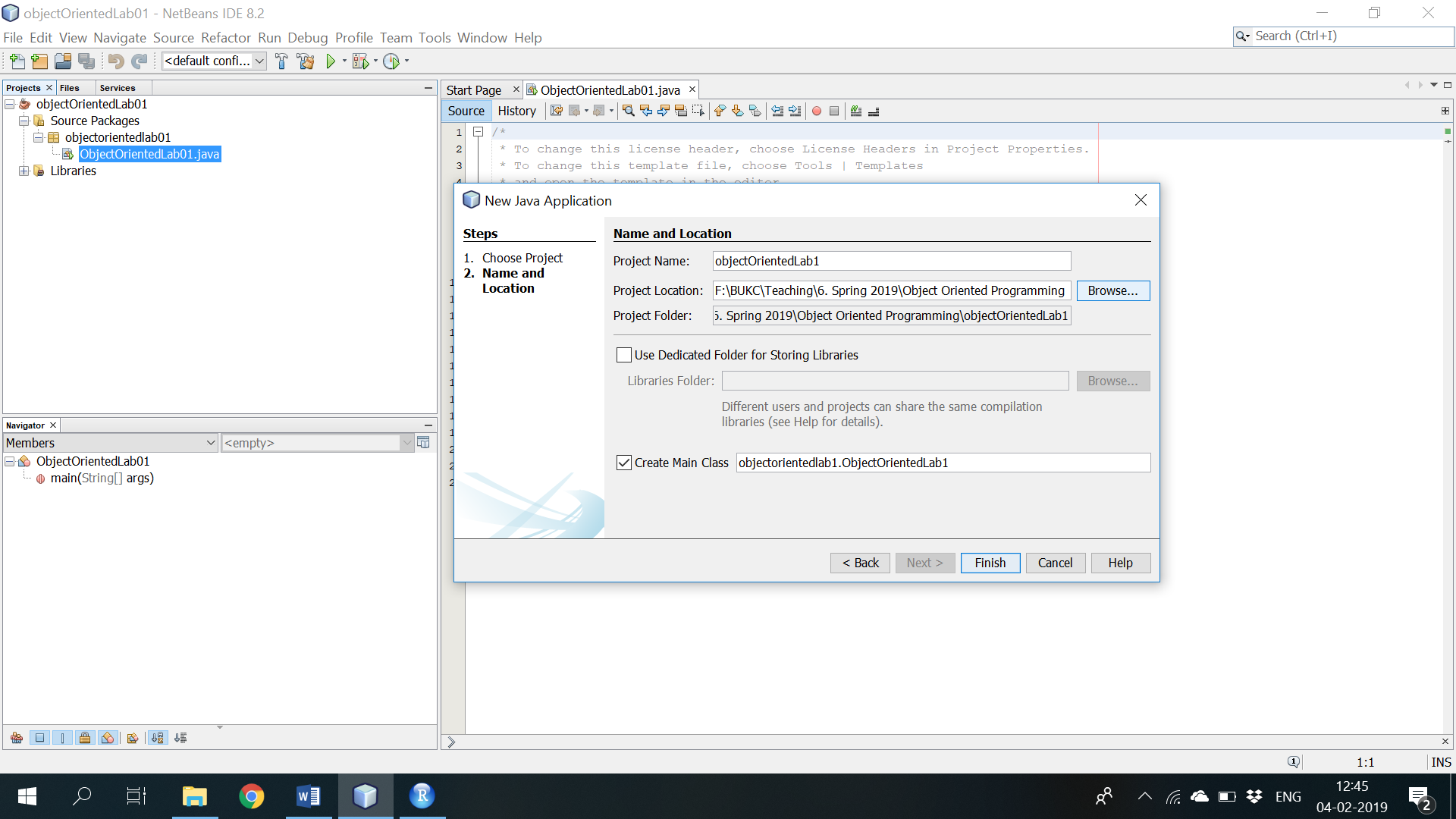
* In the IDE, choose File > New Project (Ctrl-Shift-N), as shown in the figure below.



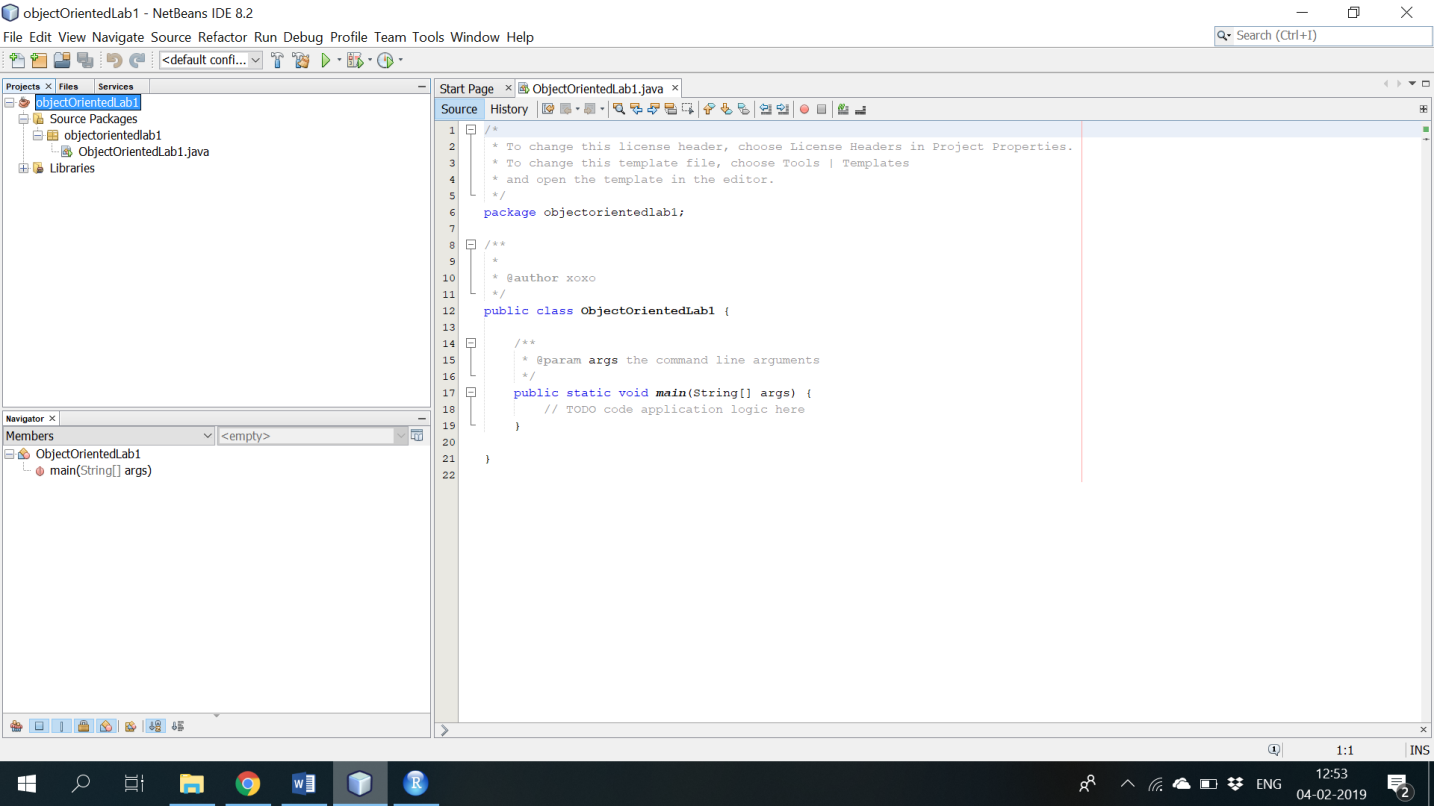
* In the New Project wizard, expand the Java category and select Java Application as shown in the figure below. Then click Next.



* In the Name and Location page of the wizard, do the following (as shown in the figure below):
* In the Project Name field, type objectOrientedLab1.
* Leave the Use Dedicated Folder for Storing Libraries checkbox unselected.
* Unselect the "Create Main Class".

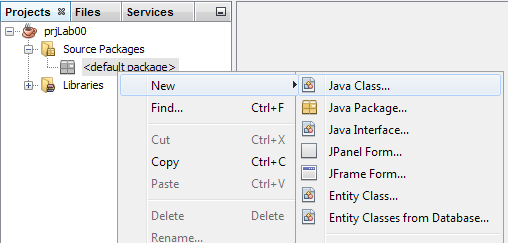


* Click Finish.
* The project is created and opened in the IDE. You should see the following components:
* The Projects window, which contains a tree view of the components of the project, including source files, libraries that your code depends on, and so on.

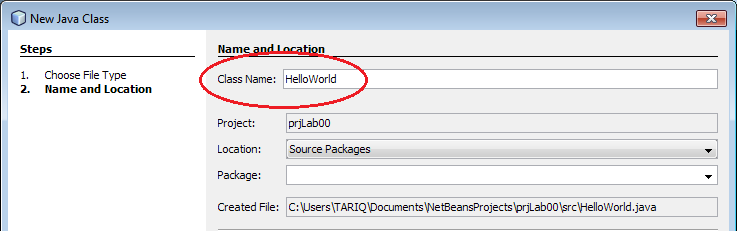


How to add your main class

Right click on default package and then select New and click on Java Class.



Click on Java Class you will see a New Java Class Wizard. Key in the Class Name HelloWorld and click Finsh.



* 1. **Adding Code to the Generated Source File:**

Because you have left the Create Main Class checkbox unselected in the New Project wizard, the IDE has not created. You can add the "Hello World!" message to the skeleton code by replacing the line:

**// TODO code application logic here**

with the line:

**System.out.println("Hello World!");**

Save the change by choosing File > Save.

The file should look something like the following:

**/\***

**\* To change this template, choose Tools | Templates**

**\* and open the template in the editor.**

**\*/**

**/\*\***

**\***

**\* @author PC**

**\*/**

**public class HelloWorld {**

**/\*\***

**\* @param args the command line arguments**

**\*/**

**public static void main(String[] args) {**

**System.out.println("Hello World!");**

**}**

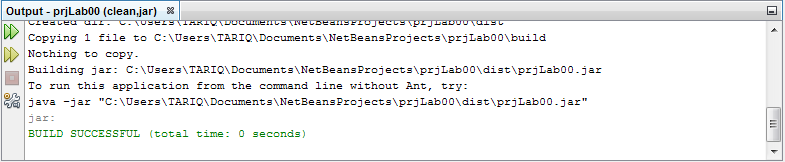
**}**

* 1. **Compiling the Source File:**

To compile your source file, choose Build > Build Main Project (F11) from the IDE's main menu.

You can view the output of the build process by choosing Window > Output > Output.

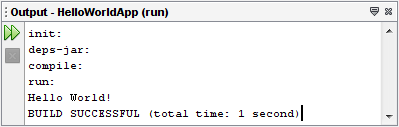
The Output window opens and displays output similar to what you see in the following figure.



* If the build output concludes with the statement BUILD SUCCESSFUL, congratulations! You have successfully compiled your program!
* If the build output concludes with the statement BUILD FAILED, you probably have a syntax error in your code. Errors are reported in the Output window as hyper-linked text. Click such a hyper-link to navigate to the source of an error. You can then fix the error and once again choose Build > Build Main Project.

Now that you have built the project, you can run your program.

* 1. **Running the Program:**
* From the IDE's menu bar, choose Run > Run Main Project (F6) OR 
* The next figure shows what you should now see.



Congratulations! Your program works!

* 1. **Comments**

**Comments** are English sentences inserted in a program to explain its purpose. We use comments to explain the purpose of a class, a method or a variable. There are two types of comments:

1. **Block comment**

/\* … \*/

* Comments are ignored by the compiler.
* Any message enclosed inside **/\* … \*/** is treated as a comment.
* This type of comment can span more than one line.

1. **Line comment**

// …

* Any message following double forward slash, **//**, up to the end of the line is also considered a comment.
* **Notice that** this type of comment *cannot* span more than one line.
* It is often used to comment variables.
  1. **Correcting Syntax Errors**

Some things to remember:

* Java is *case sensitive*, so, for example, the identifiers public, Public, and PUBLIC are all considered different. For reserved words such as public and void and previously defined identifiers such as String and System, you have to get the case right.
* When the compiler lists lots of errors, fix the first one (or few) and then recompile—often the later errors aren't really errors in the program, they just indicate that the compiler is confused from earlier errors.
* Always remember to close opened brackets, braces, and quotes.
* It is always important to remember to end every statement in Java with a *semicolon* ‘**;**’.
* Read the error messages carefully, and note what line numbers they refer to. Often the messages are helpful, but even when they aren't, the line numbers usually are.
* When the program compiles cleanly, run it.

## **Code in JAVA**

|  |  |  |
| --- | --- | --- |
| **S. No.** | **Purpose** | **Sample Code** |
| 1 | To print line (Output) | System.out.println (“Hello World!”); |
| 2 | To take inpute (Input) | import java.util.Scanner;  class Input {  public static void main(String[] args) {  Scanner input = new Scanner(System.in);  // Getting float input  System.out.print("Enter float: ");  float myFloat = input.nextFloat();  System.out.println("Float entered = " + myFloat);    // Getting double input  System.out.print("Enter double: ");  double myDouble = input.nextDouble();  System.out.println("Double entered = " + myDouble);  // Getting String input  System.out.print("Enter text: ");  String myString = input.next();  System.out.println("Text entered = " + myString);  }  } |
| 3 | If else statement (Conditional Statement) | int nbr= 5;  if ( nbr < 0 ){  System.out.println (nbr+“ is negative”);  }  else if ( nbr > 0 ) {  System.out.println (nbr+“ is positive”);  }  else {  System.out.println (nbr+“ is zero”);  } |
| 4 | Switch Statement (Conditional Statement) | Char = ‘A’;  int Left = 3, Right = 6;  switch (ch)  {  case 'A': System.out.println(Left + Right ); break;  case 'B': System.out.println(Left - Right ); break;  case 'C': System.out.println(Left \* Right ); break;  case 'D': System.out.println(Left / Right ); break;  default: System.out.println("Illegal operation" );  } |
| 5 | For Loop (Iterative Construct) | for (int j = 0; j < 3; j++)  {  System.out.println(j);  } |
| 6 | While Loop (Iterative Construct) | int num1 = 1,o num2=3;  while (num1 < num2) {  num1++;  System.out.print(num1);  } |
| 7 | Foreach Loop (Iterative Construct) | int num1 = 5, num2=3; |
| 8 | Do While Loop (Iterative Construct) | int num1 = 10, num2=3;  do{  num2++;  System.out.println(num1);  } while (num1 > num2) ; |
| 9 | 1D Array | int [ ] smallPrimes = {2,3,5,7,11,13};  int [ ] a = new int [100];  a[2]=43; |
| 10 | 2D Array | int[][] a = {  {1, 2, 3},  {4, 5, 6},  {7, 8, 9},  };  System.out.println("Length of row 1: " + a[0].length);  System.out.println("Length of row 2: " + a[1].length);  System.out.println("Length of row 3: " + a[2].length); |

## **Time Boxing**

|  |  |  |
| --- | --- | --- |
| Activity Name | Activity Time | Total Time |
| Login Systems + Setting up Netbeans | 3 mints + 5 mints | 8 mints |
| Walk through Theory & Tasks | 60 mints | 60 mints |
| Implement Tasks | 80 mints | 80 mints |
| Evaluation Time | 30 mints | 30 mints |
|  | Total Duration | 178 mints |

## **Objectives**

After completing this lab the student should be able to:

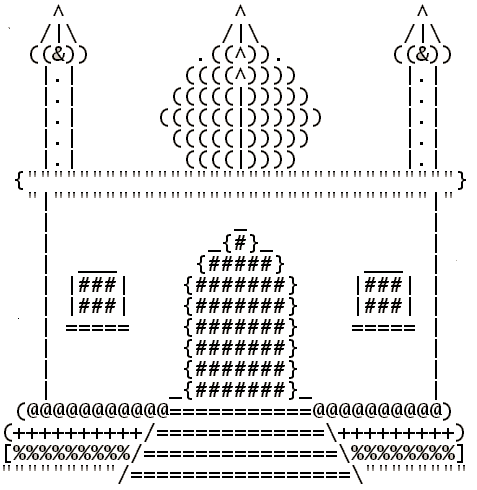
* Clearly understand the purpose and benefits of JAVA.
* Using Netbeans IDE

## **Lab Tasks/Practical Work**

**You are expected to:**

* Write brief information about yourself at the beginning of each program,
* Write comments explaining each program you write in this course, and
* Comment every variable whose purpose is not obvious from its name.

Write a program that prints a mosque, similar to the following:



Write a JAVA program, which receives the input of two integer numbers, operation (+,-,\*,/,%, power, square-root and factorial) and compute arthematic operations. Generate a menu for operations and ask user after every operation if they want to do another. (Hint use switch case)

Make a program in JAVA in which take no. of items, price of items and name of items as input from the user and give the discount according to the following conditions:

If from Bahria University give discount of 30%.

Else if the total amount is greater then 50,000 and less than 100,000 give discount of 20%.

Else if the total amount is greater then 100,000 give discount of 30%.

Write a JAVA program which will implement the following formulae using methids.

Automobile Tire Pressure: P = 0.37m(T + 460)/V

P = pressure in psi.

V = volume in cubic feet

m = mass of air in pounds

T = temperature in Fahrenheit

Pulley formulas

1. calculate the speed of one pulley if there are 2 pulleys connected with a belt:

RPM2 = diameter1/diameter2 \* RPM1

* + 1. calculate the amount of weight that can be lifted with a multiple pulley system:

weight lifted = force exerted \* number of up ropes

The body mass index (BMI), is a heuristic proxy for human body fat based on an individual's weight and height. BMI does not actually measure the percentage of body fat. We will be building a BMI calculator method. Body mass index (BMI) is computed using the the formula,



Where mass is the subject's weight in pounds (lb) and height is the height in inches (in). The value 703 is a factor to convert BMI to a value that matches the original BMI calculations done in metric units (i.e. kilograms-meters).

**Lab Manual for Object Oriented Programming**

**Lab No. 2**

# **IN- DEPTH UNDERSTANDING OF CLASSES AND OBJECTS**

Objectives

To understand basic concept of Classes and Objects

**LAB # 02**

**In-Depth Understanding of Classes and Objects**

## **Introduction**

### Why do we model?

In Object Oriented Programming we are trying to model either real world entities or processes and represent them in software. There are compelling reasons why we model:

* A model is a simplification of reality. We model because we cannot comprehend the complexity of a system in its entirety.
* We model to visualize, specify, construct, and document the structure and behavior of a system's architecture.
* A model is a complete description of a system from a particular perspective.

### Principles of Modeling

The model that we create is dependent on the problem that we are trying to solve and the entities in the scope of the problem.

* The choice of what models to create has a profound influence on how the problem is attacked and how a solution is shaped.
* Every model maybe expressed at different levels of precision.
* The best models are connected to reality.
* No single model is sufficient. Every non-trivial system is best approached through a small set of nearly independent models.

### Basic Principles of Object Orientation

#### Abstraction:

the most important or essential aspects of something while ignoring the less important details. Abstraction is dependent on perspective - what is important in one context may not be in another. We model our problem domain using abstractions.

#### Encapsulation:

physical localization of features into a single blackbox abstraction that hides their implementation behind a public interface. "Information hiding" is a corollary concept that indicates data fields are hidden from the user but the functionalities as implemented through functions are exposed.

#### Inheritance:

organization of abstractions according to some order (e.g. complexity, responsibility, etc.).

#### Polymorphism:

substitute variables or objects of one type with variables or objects of another type. Polymorphism gives us the ability to switch components without loss of functionality.

### Basic Concepts of Object Orientation

* Object
* Class
* Attribute
* Operation (Function)

### What is an Object?

An Object represents an entity either physical (box), conceptual (chemical process), or software (list).

An Object is a concept, an abstraction, a thing with sharp boundaries and meaning for an application. It has

* Identity - a name
* State - determined by the values of its attributes
* Behavior - determined by how the object acts or reacts to requests (messages) from other objects

An Object is represented as a rectangle with a underlined name in UML.

### What is a Class?

A Class is a description of a group of objects with common properties (attributes), behavior (operations), relationships, and semantics

A class is an abstraction. An object is an instance of a class.

### Example of a Class

* Class: Course
* Properties: Name, Location, Days Offered, Credit Hours, Professor
* Behavior: Add Student, Delete Student, Get Course Roster, Determine If Full

A class is represented by a compartmentalized rectangle in UML. It has three sections - Name, Attributes, and Operations. You can show as many or as few of the Attributes and Operations in the diagram. Most of the times for the sake of clarity the Attribute and Operation lists are suppressed.

You start from real world objects - abstract out what you do not care and go through the process of classification of what you care. A Class is the result of this classification. Classes are then used as templates within a software system to create software objects.

### What is an Attribute?

An Attribute is a named property of a class. It has a type. It describes the range of values that that property may hold.

### What is an Operation (Function)?

An Operation is a service that can be requested from any object of the Class to affect behavior. An Operation can either be a command or a question. A question should never change the state of the object only a command can. The outcome of the Operation depends on the current state of the object.

## **Time Boxing**

|  |  |  |
| --- | --- | --- |
| Activity Name | Activity Time | Total Time |
| Login Systems + Setting up Visual Studio Environment | 3 mints + 5 mints | 8 mints |
| Walk through Theory & Tasks | 60 mints | 60 mints |
| Implement Tasks | 80 mints | 80 mints |
| Evaluation Time | 30 mints | 30 mints |
|  | Total Duration | 178 mints |

## **Objectives**

After completing this lab the student should be able to:

* Clearly understand the purpose and benefits of Object Oriented Programming.

## **Lab Tasks/Practical Work**

1: Create a class student which contains the basic data about the student that takes the basic student data and displays it by using display method. An option of update is being provided to the user if he/she want to update the data, the required data being updated.

2: Create a class “computer” which contains specifications of computer, the program shall ask the user does he/she wants to open the system, if the user press “yes” then the system starts shows the initial loading and then displays the basic configuration of a system (by calling the method of display () ), update option is being provided by the user, values of the specified items are being updated once user decides to update that item.

3: Create a class of Employee which contains basic information about an employee, employee name, father’s name and salary etc are being displayed by the display method and the salary of employees or the designation of the employees: are being set/updated as per need.

4: Create a class of Automobile which contains specifications of a car, check whether the car is in - ON/start state if not them asks the user if he/she want to start the car, If the car is already in start state then first display the current status of the car which includes the horse power, color, made, engine type etc. providing an option to the user if he/she wants to update any of the mentioned part from the car, if user selects “YES” then it is updated according to the need of the user, else the program will be ended.

5.Implement a class Car, that has the following characteristics:

a) Brandname

b) PriceNew, which represents the price of the car when it was new

c) Color and

d) Odometer,which is milo meter shows number of milage travelled by car

The class should have:

A. A method getPriceAfterUse() which should return the price of the car after being used according to the following formula: car price after being used=priceNew\*(1-(odometer/600,00))

B. A method updateMilage (double travelled distance) that changes the current state of the car by increasing its milage, and

1. A method outputDetails() that will output to the screen all the information of the car, i.e. brandname,priceNew,price used,color and odometer.

**Lab Manual for Object Oriented Programming**

**Lab No.3**

# **IN-DEPTH UNDERSTANDING OF ACCESS MODIFIERS**

Objectives

To understand the use of access modifier and concept of basic inheritance and encapsulation.

**LAB # 03**

**In-Depth understanding of Access Modifiers**

## **Introduction**

Access modifiers are keywords used to specify the declared accessibility of a member or a type.

### Why to use access modifiers?

 Access modifiers are an integral part of object-oriented programming. They support the concept of encapsulation, which promotes the idea of hiding functionality. Access modifiers allow you to define who does or doesn't have access to certain features.

**In JAVA there are 5 different types of Access Modifiers.**

|  |  |
| --- | --- |
| ifier | Description |
| public | There are no restrictions on accessing public members. |
| private | Access is limited to within the class definition. This is the default access modifier type if none is formally specified |
| protected | Access is limited to within the class definition, within same package and any class that inherits from the class |
| PACKAGE (default) | package members are accessible in classes in the same package and the class itself |
|  |  |

### Public

The public keyword is an access modifier for types and type members. Public access is the most permissive access level.

There are no restrictions on accessing public members.

#### Accessibility:

* Can be accessed by objects of the class
* Can be accessed by derived classes

#### Example:

In the following example num1 is direct access.

**MAIN CLASS:**

package lab3;

public class Lab3 {

public static void main(String[] args) {

// TODO code application logic here

Class1 ob1 = new Class1();

//Direct access to public members

ob1.num1 = 100;

System.out.println("Number one value in main " + ob1.num1);

System.out.println();

}

}

**CLASS 1:**

package lab3;

public class Class1 {

public int num1;

}

### Private

### Private access is the least permissive access level.

Private members are accessible only within the body of the class or the struct in which they are declared.

#### Accessibility:

* Cannot be accessed by object
* Cannot be accessed by derived classes

#### Example:

In the following example num2 is not accessible outside the class.

**MAIN CLASS:**

package lab3;

public class Lab3 {

public static void main(String[] args) {

// TODO code application logic here

Class1 ob1 = new Class1();

//Direct access to public members

ob1.num1 = 100;

//Access to private member is not permitted

ob1.num2 = 20;

System.out.println("Number one value in main " + ob1.num1);

System.out.println();

}

}

**CLASS1:**

package lab3;

public class Class1 {

public int num1;

private int num2;

}

The above program will give compilation error, as access to private is not permissible.

### Protected

A protected member is accessible from within the class in which it is declared, and from within any class derived from the class that declared this member.

A protected member of a base class is accessible in a derived class only if the access takes place through the derived class type.

#### Accessibility:

* Cannot be accessed by object
* By derived classes
* Classes within same package

**MAIN CLASS:**

package lab3;

import newpackage.Class2;

public class Lab3 {

public static void main(String[] args) {

// TODO code application logic here

Class1 ob1 = new Class1();

//Direct access to public members

ob1.num1 = 100;

//Access to private member is not permitted

ob1.num2 = 20;

System.out.println("Number one value in main " + ob1.num1);

System.out.println();

Class2 ob2 = new Class2();

ob2.num1 = 70;

ob2.num2 = 10;

System.out.println("Number one value in main " + ob2.num1);

System.out.println();

}

}

**CLASS 1:**

package lab3;

public class Class1 {

public int num1;

protected int num2;

}

**CLASS 2:**

package newpackage;

public class Class2 {

public int num1;

protected int num2;

}

In the above program we try to access protected member in main it is not available.

### this

If the attribute in method header has the same name as the class attribute then use this keyword with class attribute.

#### Accessibility:

The various usages of 'THIS' keyword in Java are as follows:

* It can be used to refer instance variable of current class
* It can be used to invoke or initiate current class constructor
* It can be passed as an argument in the method call
* It can be passed as argument in the constructor call
* It can be used to return the current class instance

**Example 1:**

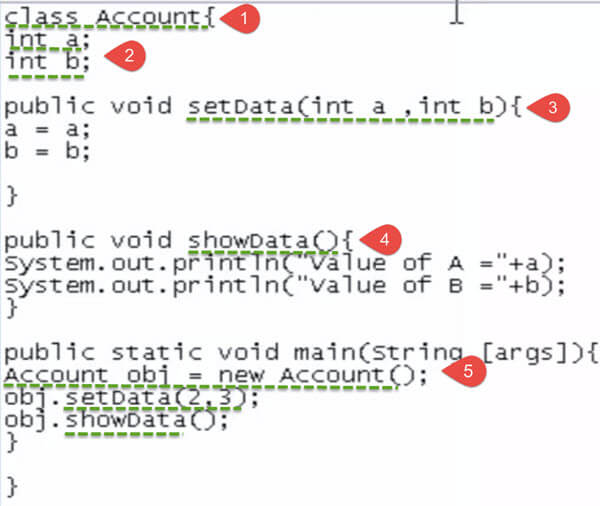
This example shows the use of keyword *this* with class attribute.

public void setName(String name) {

this.name = name;

}

**EXAMPLE 2:**



### Final

The **final keyword** in java is used to restrict the user. The java final keyword can be used in many context. Final can be:

1. variable
2. method
3. class

The final keyword can be applied with the variables, a final variable that have no value it is called blank final variable or uninitialized final variable. It can be initialized in the constructor only. The blank final variable can be static also which will be initialized in the static block only. We will have detailed learning of these. Let's first learn the basics of final keyword.

If you make any variable as final, you cannot change the value of final variable(It will be constant). There is a final variable speedlimit, we are going to change the value of this variable, but It can't be changed because final variable once assigned a value can never be changed.

**EXAMPLE:**

**class** Bike9{

**final** **int** speedlimit=90;//final variable

**void** run(){

  speedlimit=400;  //**Compile time error**

 }

**public** **static** **void** main(String args[]){

 Bike9 obj=**new**  Bike9();

 obj.run();

 }

}//end of class

## **Time Boxing**

|  |  |  |
| --- | --- | --- |
| Activity Name | Activity Time | Total Time |
| Login Systems + Setting up Visual Studio Environment | 3 mints + 5 mints | 8 mints |
| Walk through Theory & Tasks | 60 mints | 60 mints |
| Implement Tasks | 80 mints | 80 mints |
| Evaluation Time | 30 mints | 30 mints |
|  | Total Duration | 178 mints |

## **Objectives**

After completing this lab the student should be able to:

* Clearly understand the purpose and benefits of class.
* Understanding and implementation of object.

## **Lab Tasks/Practical Work**

1. Write a program using the concepts of a default constructor. Consider a computer system whose name, type, processor specification, ram, hard disk drives, mother board, optical drive etc, in a constructor, desired values are entered by the user in a get method (that takes information from the user) and the displays the inputted information via display method. The user shall be asked to change any of the provided information if he/she agrees to change the information then new values shall be asked from the user.
2. Implement a class *Car,* that has the following characteristics and should be implemented using proper access modifier:
3. *brandName*,
4. *priceNew,* which represents the price of the car when it was new,
5. *color*, and
6. *odometer*, which is milo meter shows number of milage travelled by car

The class should have:

1. A method *getPriceAfterUse()* which should return the price of the car after being used according to the following formula:

car price after being used = *priceNew* ()

1. A method *updateMilage(double traveledDistance)* that changes the current state of the car by increasing its milage, and
2. A method *outputDetails()* that will output to the screen all the information of the car, i.e., brand name, price new, price used, color, and odometer.
3. Design then implement a class to represent a **Flight**. A Flight has a *flight number*, a *source*, a *destination* and a *number of available seats. This* should be implemented using proper access modifier. The class should have:
4. A **constructor** to initialize the 4 instance variables. You have to shorten the name of the source and the destination to 3 characters only if it is longer than 3 characters by a call to the method in the ‘j’ part.
5. An **overloaded constructor** to initialize the *flight number* and the *number of available seats* instance variables only.

(**NOTE:** Initialize the *source* and the *destination* instance variables to empty string, i.e." ")

1. An **overloaded constructor** to initialize the *flight number* instance variable only.

(**NOTE:** Initialize the *source* and the *destination* instance variables to empty string; and the *number of available seats* to zero)

1. A **method** **public void reserve(int numberOfSeats)** to reserve seats on the flight. (**NOTE:** You have to check that there is enough number of seats to reserve)
2. A **method** **public void cancel(int numberOfSeats)** to cancel one or more reservations
3. A **toString** method to easily return the flight information as follows:

**Flight No: 1234**

**From: KAR**

**To: LAH**

**Available Seats: 18**

1. An **equals** method to compare 2 flights.

(**NOTE:** 2 Flights considered being equal if they have the same flight number)

1. The following method:

**private String shortAndCapital (String name) {**

**if (name.length() <= 3) {**

**return name.toUpperCase();**

**} else {**

**return name.substring(0,3).toUpperCase();**

**}**

**}**

**Write a test class for the *Flight* class you wrote. You should try to use all the methods you wrote.**

1. MyJava Coffee Outlet runs a catalog business. It sells only one type of coffee beans. The company sells the coffee in 2-lb bags only and the price of a single 2-lb bag is $5.50. when a customer places an order, the company ships the order in boxes. The boxes come in 3 sizes with 3 different costs:

|  |  |  |  |
| --- | --- | --- | --- |
|  | Large box | Medium box | Small box |
| capacity | 20 bags | 10 bags | 5 bags |
| cost | $1.80 | $1.00 | $0.60 |

The order is shipped using the least number boxes. For example, the order of 52 bags will be shipped in 2 boxes: 2 large boxes, 1medium and 1 small.

Develop an application that computes the total cost of an order.

Sample out put:

|  |
| --- |
| Number of Bags Ordered: 52  The Cost of Order: $ 286.00  Boxes Used:  2 Large - $3.60  1 Medium - $1.00  1 Small - $0.60  Your total cost is: $ 291.20 |

**Lab Manual for Object Oriented Programming**

**Lab No. 4**

# **UNDERSTANDING THE IMPLEMENTION OF STATIC CLASSES AND MEMBERS**

Objectives

To understand concept, usage and implementation of Static members

**LAB # 04**

## **Understanding the implementation of Static Classes & Members**

## **Introduction**

the usual way to communicate with a class, is to create a new instance of the class, and then work on the resulting object. In most cases, this is what classes are all about - the ability to instantiate multiple copies of the same class and then use them differently in some way. However, in some cases, you might like to have a class which you may use without instantiating it, or at least a class where you can use members of it without creating an object for it. For instance, you may have a class with a variable that always remains the same, no matter where and how it's used. This is called a static member, static because it remains the same.   
  
A class can be static, and it can have static members, both functions and fields. A static class can't be instantiated, so in other words, it will work more as a grouping of related members than an actual class. You may choose to create a non-static class instead, but let it have certain static members. A non-static class can still be instantiated and used like a regular class, but you can't use a static member on an object of the class. A static class may only contain static members.

### static

The static keyword in Java is used for memory management mainly. We can apply java static keyword with variables, methods, blocks and nested class. The static keyword belongs to the class than an instance of the class. The static can be:

* Variable (also known as a class variable)
* Method (also known as a class method)
* Block
* Nested class

If you declare any variable as static, it is known as a static variable.

* The static variable can be used to refer to the common property of all objects (which is not unique for each object), for example, the company name of employees, college name of students, etc.
* The static variable gets memory only once in the class area at the time of class loading.

**STUDENT CLASS:**

/Java Program to demonstrate the use of a static method.

**class** Student{

**int** rollno;

     String name;

**static** String college = "ITS";

     //static method to change the value of static variable

**static** **void** change(){

     college = "BBDIT";

     }

     //constructor to initialize the variable

     Student(**int** r, String n){

     rollno = r;

     name = n;

     }

     //method to display values

**void** display(){System.out.println(rollno+" "+name+" "+college);}

}

**MAIN CLASS:**

//Test class to create and display the values of object

**public** **class** TestStaticMethod{

**public** **static** **void** main(String args[]){

    Student.change();//calling change method

    //creating objects

    Student s1 = **new** Student(111,"Ali");

    Student s2 = **new** Student(222,"Osama");

    Student s3 = **new** Student(333,"Hamza");

    //calling display method

    s1.display();

    s2.display();

    s3.display();

    }

}

### Restrictions for the static method

There are two main restrictions for the static method. They are:

1. The static method can not use non static data member or call non-static method directly.
2. this and super cannot be used in static context.

First, here is an example of a static class:

public static class Rectangle

{

public static int CalculateArea(int width, int height)

{

return width \* height;

}

}

As you can see, we use the static keyword to mark the class as static, and then we use it again to mark the method, CalculateArea, as static as well. If we didn't do that, the compiler would complain, since we can't have a non-static member of a static class.   
  
To use this method, we call it directly on the class, like this:

System.Out.println("The area is: " + Rectangle.CalculateArea(5, 4));

We could add other helpful methods to the Rectangle class, but perhaps you are wondering why we are passing on width and height to the actual method, instead of storing it inside the class and then pulling them from there when needed? Because it's static! We could store them, but only one set of dimensions, because there is only one version of a static class. This is very important to understand.   
  
Instead, we can make the class non-static, and then have the CalculateArea as a utility function on this class:

public class Rectangle

{

private int width, height;

public Rectangle(int width, int height)

{

this.width = width;

this.height = height;

}

public void OutputArea()

{

System.out.println("Area output: " + Rectangle.CalculateArea(this.width, this.height));

}

public static int CalculateArea(int width, int height)

{

return width \* height;

}

}

As you can see, we have made the class non-static. We have also added a constructor, which takes a width and a height and assigns it to the instance. Then we have added an OutputArea method, which uses the static method to calculate the area. This is a fine example of mixing static members with non-static members, in a non-static class.   
  
A common usage of static classes, although frowned upon by some people, are utility/helper classes, where you collect a bunch of useful methods, which might not belong together, but doesn't really seem to fit elsewhere either.

## **Time Boxing**

|  |  |  |
| --- | --- | --- |
| Activity Name | Activity Time | Total Time |
| Login Systems + Setting up Visual Studio Environment | 3 mints + 5 mints | 8 mints |
| Walk through Theory & Tasks | 60 mints | 60 mints |
| Implement Tasks | 80 mints | 80 mints |
| Evaluation Time | 30 mints | 30 mints |
|  | Total Duration | 178 mints |

## **Objectives**

After completing this lab the student should be able to:

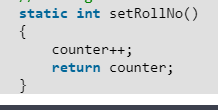
* Clearly understand the purpose and benefits of static classes & members.

## **Lab Tasks/Practical Work**

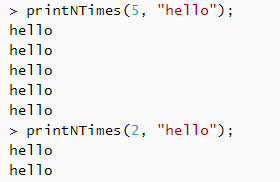
1. Write a program to calculate area of rectangle by using static method.Use parameterized constructor to assign width and height to the instance. Use Output area method which uses the static method to calculate the area.
2. Write a program to display Name, Enrollment Number, University Name and ,Semester of students that are from same university and semester using static fields and methods.(Hint: first set the university name and semester as follows:



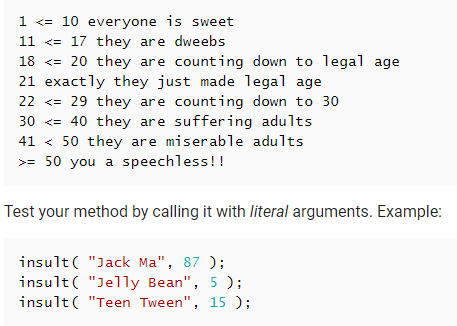
Then use static variable counter to get unique roll numbers as follows:



3.Write a static method called printNTimes that takes an integer n and a string (in that order) as its parameters and prints the string n times. For example



4.Write a static method called insult that has two paramaters, a String which represents a person’s name and an integer which represents the persons age. This  
method should create and **return** a String which is a personal insult based on the value of the argument age that was passed. Use the following age cuttoffs (or variations of your choosing) for creating your insults:



5.Write a static method called greetMe that greets you. The method should issue a prompt asking for your name, display a polite (or not so polite) greeting message and then prompt you to enter your age.

**Lab Manual for Object Oriented Programming**

**Lab No. 5**

# **UNDERSTANDING THE CONCEPT OF OVERLOADING**

Objectives

To understand basic concept of Method Overloading and Operator overloading

**LAB # 05**

## **Understanding the concept of Overloading**

## **Introduction**

## Overloading:

C++ allows you to specify more than one definition for a **function** name or an **operator** in the same scope, which is called **method overloading** and **operator overloading** respectively.

An overloaded declaration is a declaration that had been declared with the same name as a previously declared declaration in the same scope, except that both declarations have different arguments and obviously different definition (implementation).

When you call an overloaded **method** or **operator**, the compiler determines the most appropriate definition to use by comparing the argument types you used to call the function or operator with the parameter types specified in the definitions. The process of selecting the most appropriate overloaded function or operator is called **overload resolution**.

### Method overloading in C++:

You can have multiple definitions for the same method name in the same scope. The definition of the function must differ from each other by the types and/or the number of arguments in the argument list. You can not overload function declarations that differ only by return type.

Following is the example where same function **print()** is being used to print different data types:

#include <iostream>

using namespace std;

class printData

{

public:

void print(int i) {

cout << "Printing int: " << i << endl;

}

void print(double f) {

cout << "Printing float: " << f << endl;

}

void print(char\* c) {

cout << "Printing character: " << c << endl;

}

};

int main(void)

{

printData pd;

// Call print to print integer

pd.print(5);

// Call print to print float

pd.print(500.263);

// Call print to print character

pd.print("Hello C++");

return 0;

}

When the above code is compiled and executed, it produces the following result:

Printing int: 5

Printing float: 500.263

Printing character: Hello C++

### Operators overloading in C++:

You can redefine or overload most of the built-in operators available in C++. Thus a programmer can use operators with user-defined types as well.

Overloaded operators are functions with special names the keyword operator followed by the symbol for the operator being defined. Like any other function, an overloaded operator has a return type and a parameter list.

Box operator+(const Box&);

declares the addition operator that can be used to **add** two Box objects and returns final Box object. Most overloaded operators may be defined as ordinary non-member functions or as class member functions. In case we define above function as non-member function of a class then we would have to pass two arguments for each operand as follows:

Box operator+(const Box&, const Box&);

Following is the example to show the concept of operator over loading using a member function. Here an object is passed as an argument whose properties will be accessed using this object, the object which will call this operator can be accessed using **this** operator as explained below:

#include <iostream>

using namespace std;

class Box

{

public:

double getVolume(void)

{

return length \* breadth \* height;

}

void setLength( double len )

{

length = len;

}

void setBreadth( double bre )

{

breadth = bre;

}

void setHeight( double hei )

{

height = hei;

}

// Overload + operator to add two Box objects.

Box operator+(const Box& b)

{

Box box;

box.length = this->length + b.length;

box.breadth = this->breadth + b.breadth;

box.height = this->height + b.height;

return box;

}

private:

double length; // Length of a box

double breadth; // Breadth of a box

double height; // Height of a box

};

// Main function for the program

int main( )

{

Box Box1; // Declare Box1 of type Box

Box Box2; // Declare Box2 of type Box

Box Box3; // Declare Box3 of type Box

double volume = 0.0; // Store the volume of a box here

// box 1 specification

Box1.setLength(6.0);

Box1.setBreadth(7.0);

Box1.setHeight(5.0);

// box 2 specification

Box2.setLength(12.0);

Box2.setBreadth(13.0);

Box2.setHeight(10.0);

// volume of box 1

volume = Box1.getVolume();

cout << "Volume of Box1 : " << volume <<endl;

// volume of box 2

volume = Box2.getVolume();

cout << "Volume of Box2 : " << volume <<endl;

// Add two object as follows:

Box3 = Box1 + Box2;

// volume of box 3

volume = Box3.getVolume();

cout << "Volume of Box3 : " << volume <<endl;

return 0;

}

When the above code is compiled and executed, it produces the following result:

Volume of Box1 : 210

Volume of Box2 : 1560

Volume of Box3 : 5400

### Overloadable/Non-overloadableOperators:

Following is the list of operators which can be overloaded:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| + | - | \* | / | % | ^ |
| & | | | ~ | ! | , | = |
| < | > | <= | >= | ++ | -- |
| << | >> | == | != | && | || |
| += | -= | /= | %= | ^= | &= |
| |= | \*= | <<= | >>= | [] | () |
| -> | ->\* | new | new [] | delete | delete [] |

Following is the list of operators, which can not be overloaded:

|  |  |  |  |
| --- | --- | --- | --- |
| :: | .\* | . | ?: |

## **Time Boxing**

|  |  |  |
| --- | --- | --- |
| Activity Name | Activity Time | Total Time |
| Login Systems + Setting up Visual Studio Environment | 3 mints + 5 mints | 8 mints |
| Walk through Theory & Tasks | 60 mints | 60 mints |
| Implement Tasks | 80 mints | 80 mints |
| Evaluation Time | 30 mints | 30 mints |
|  | Total Duration | 178 mints |

## **Objectives**

After completing this lab the student should be able to:

* Clearly understand the concept of Method overloading.
* Clearly understand the concept of Operator overloading.

## **Lab Tasks/Practical Work**

1. Write a program which contains a class ‘Calculator’ contains multiple sum method by using method overloading concept.

2.Create a class to print the area of a square and a rectangle. The class has two methods with the same name but different number of parameters. The method for printing area of rectangle has two parameters which are length and breadth respectively while the other method for printing area of square has one parameter which is side of square.

1. Create a class 'Student' with three data members which are name, age and address. The constructor of the class assigns default values name as "unknown", age as '0' and address as "not available". It has two members with the same name 'setInfo'. First method has two parameters for name and age and assigns the same whereas the second method takes has three parameters which are assigned to name, age and address respectively. Print the name, age and address of 4 students.
2. Implement the Circle class to overload the + operator so that you can add two Circle objects. Adding two Circle object should give another Circle whose radius the sum of the radii of the two Circle objects.
3. Implement the Rectangle class to overload the + operator so that you can add two Rectangle objects. Adding two Rectangle objects should give another Rectangle object whose length is the sum of the lengths of the two Rectangle objects and whose breadth is the sum of the breadths of the two Rectangle objects.

5.Write a class Time which represents time. the class should have three fields for hours, minutes and seconds. It should have constructor to initialize the hours, minutes and seconds.  
A method printTime() to print the current time.  
Overload the following operators:  
plus operator (+) (add two time objects based on 24 hour clock)  
and < (compare two time objects)

**Lab Manual for Object Oriented Programming**

**Lab No. 4**

# **UNDERSTANDING THE USES OF CONSTRUCTOR AND ITS TYPES**

Objectives

To understand basic concept of Object Oriented Programming.

**LAB # 04**

**Understanding the uses of constructor and its types**

## **Introduction**

Constructors are specialized methods. They create instances of classes with runtime support. They are standardized initialization methods. We invoke a constructor with the new-keyword.

A special method of the class that will be automatically invoked when an instance of the class is created is called a constructor. The main use of constructors is to initialize private fields of the class while creating an instance for the class. When you have not created a constructor in the class, the compiler will automatically create a default constructor in the class. The default constructor initializes all numeric fields in the class to zero and all string and object fields to null.   
  
Some of the key points regarding the Constructor are:

* A class can have any number of constructors.
* A constructor doesn't have any return type, not even void.
* A static constructor can not be a parametrized constructor.
* Within a class you can create only one static constructor.

###### Constructors can be divided into 5 types:

#### Default Constructor

#### Parametrized Constructor

#### Copy Constructor

#### Static Constructor

#### Private Constructor

Now let us see  each constructor type with example as below

### Default Constructor A constructor without any parameters is called a default constructor; in other words this type of constructor does not take parameters. The drawback of a default constructor is that every instance of the class will be initialized to the same values and it is not possible to initialize each instance of the class to different values. The default constructor initializes:

1. All numeric fields in the class to zero.
2. All string and object fields to null.

#### Example

using System;  
namespace DefaultConstractor  
 {  
    class addition  
    {  
        int a, b;

        public addition()   //default contructor

        {

            a = 100;

            b = 175;

        }

        public static void Main()

        {

            addition obj = new addition(); //an object is created , constructor is called

            Console.WriteLine(obj.a);

            Console.WriteLine(obj.b);

            Console.Read();

        }

      }

    }

Now run the application, the output will be as in the following:



### Parameterized Constructor

 A constructor with at least one parameter is called a parametrized constructor. The advantage of a parametrized constructor is that you can initialize each instance of the class to different values.

using System;

namespace Constructor

{  
    class paraconstrctor  
    {  
      public  int a, b;  
      public paraconstrctor(int x, int y)  // decalaring Paremetrized Constructor with ing x,y parameter

        {

            a = x;

            b = y;

        }

   }

    class MainClass

    {

        static void Main()

        {

            paraconstrctor v = new paraconstrctor(100, 175);   // Creating object of Parameterized Constructor and ing values

            Console.WriteLine("-----------parameterized constructor example by vithal wadje---------------");

            Console.WriteLine("\t");

            Console.WriteLine("value of a=" + v.a );

            Console.WriteLine("value of b=" + v.b);

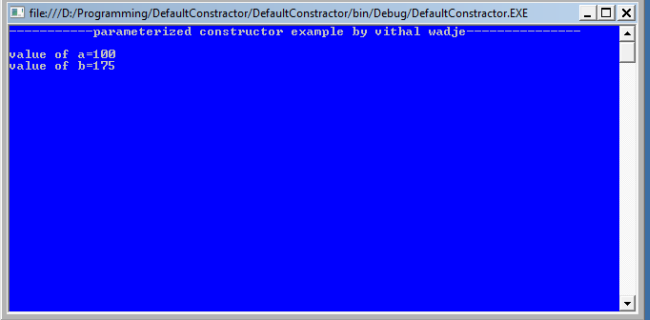
            Console.Read();

        }

    }

}

Now run the application, the output will be as in the following:



Copy Constructor  
The constructor which creates an object by copying variables from another object is called a copy constructor. The purpose of a copy constructor is to initialize a new instance to the values of an existing instance

Syntax:  
public employee(employee emp)  
{  
name=emp.name;  
age=emp.age;  
}  
  
The copy constructor is invoked by instantiating an object of type employee and ing it the object to be copied.  
  
**Example**

employee emp1=new  employee (emp2);

Now, emp1 is a copy of emp2.   
So let us see its practical implementation.

using System;

namespace copyConstractor

{

    class employee

    {

        private string name;

        private int age;

        public employee(employee emp)   // declaring Copy constructor.

        {

            name = emp.name;

            age = emp.age;

        }

        public employee(string name, int age)  // Instance constructor.

        {

            this.name = name;

            this.age = age;

        }

        public string Details     // Get deatils of employee

        {

            get

            {

                return  " The age of " + name +" is "+ age.ToString();

            }

        }

    }

    class empdetail

    {

        static void Main()

        {

            employee emp1 = new employee("Vithal", 23);  // Create a new employee object.

            employee emp2 = new employee(emp1);         **// here is emp1 details is copied to emp2.**

            Console.WriteLine(emp2.Details);

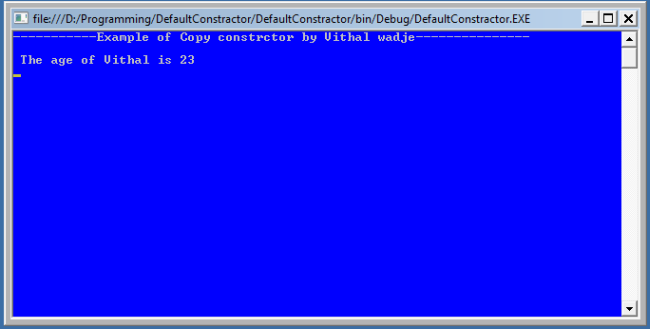
            Console.ReadLine();

        }

    }

}

Now run the program, the output will be as follows:

  
  
Static Constructor  
When a constructor is created as static, it will be invoked only once for all of instances of the class and it is invoked during the creation of the first instance of the class or the first reference to a static member in the class. A static constructor is used to initialize static fields of the class and to write the code that needs to be executed only once.  
Some key points of a static constructor is:

1. A static constructor does not take access modifiers or have parameters.
2. A static constructor is called automatically to initialize the class before the first instance is created or any static members are referenced.
3. A static constructor cannot be called directly.
4. The user has no control on when the static constructor is executed in the program.
5. A typical use of static constructors is when the class is using a log file and the constructor is used to write entries to this file.

**Syntax**  
class employee  
 {// Static constructor  
  static employee(){}  
 }

Now let us see it with practically

using System;

namespace staticConstractor

{

public class employee

{

    static employee() // Static constructor declaration{Console.WriteLine("The static constructor ");

}

public static void Salary()

 {

    Console.WriteLine();

    Console.WriteLine("The Salary method");

 }

}

class details

{

    static void Main()

    {

        Console.WriteLine("----------Static constrctor example by vithal wadje------------------");

        Console.WriteLine();

        employee.Salary();

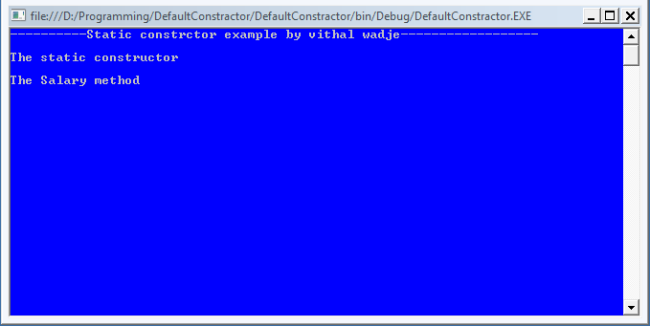
        Console.ReadLine();

    }

  }

}

Now run the program the output will be look as in the following:



### Private Constructor

When a constructor is created with a private specifier, it is not possible for other classes to derive from this class, neither is it possible to create an instance of this class. They are usually used in classes that contain static members only. Some key points of a private constructor are:

1. One use of a private constructor is when we have only static members.
2. It provides an implementation of a singleton class pattern
3. Once we provide a constructor that is either private or public or any, the compiler will not add the parameter-less public constructor to the class.

Now let us see it practically.

using System;

namespace defaultConstractor

{

    public class Counter

    {

        private Counter()   //private constrctor declaration

        {

        }

        public static int currentview;

        public static int visitedCount()

        {

            return ++ currentview;

        }

    }

    class viewCountedetails

    {

        static void Main()

        {

            // Counter aCounter = new Counter();   // Error

            Console.WriteLine("-------Private constructor example by vithal wadje----------");

            Console.WriteLine();

            Counter.currentview = 500;

            Counter.visitedCount();

            Console.WriteLine("Now the view count is: {0}", Counter.currentview);

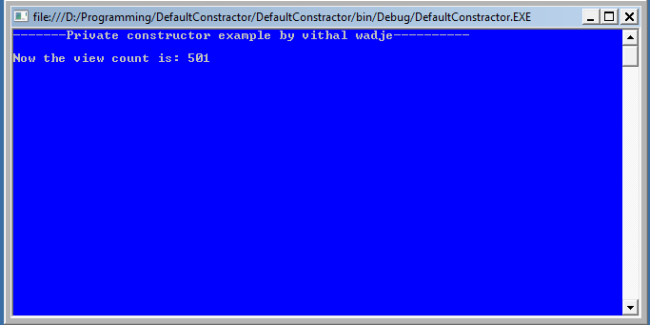
            Console.ReadLine();

        }

    }

}

Now run the application; the output is:



If you uncomment the preceding statement that is commented in the above program then it will generate an error because the constructor is inaccessible (private).

## **Time Boxing**

|  |  |  |
| --- | --- | --- |
| Activity Name | Activity Time | Total Time |
| Login Systems + Setting up Visual Studio Environment | 3 mints + 5 mints | 8 mints |
| Walk through Theory & Tasks | 60 mints | 60 mints |
| Implement Tasks | 80 mints | 80 mints |
| Evaluation Time | 30 mints | 30 mints |
|  | Total Duration | 178 mints |

## **Objectives**

After completing this lab the student should be able to:

* Clearly understand the purpose of access modifiers.
* Understand concepts of inheritance and encapsulation.

## **Lab Tasks/Practical Work**

1. Write a program using the concepts of a default constructor. Consider a computer system whose name, type, processor specification, ram, hard disk drives, mother board, optical drive etc, in a constructor, desired values are entered by the user in a get method (that takes information from the user) and the displays the inputted information via display method. The user shall be asked to change any of the provided information if he/she agrees to change the information then new values shall be asked from the user.
2. Create a program that determines the current time and date. The program must incorporate several Methods out of which three Methods should be constructors, the first one shall be a default constructor, the second and third one shall be an overloaded constructors, from which one methods deals with YEAR, MONTH AND DAY, whereas the second method deals with YEAR, MONTH,DAY,HOUR,MINUTES AND SECONDS. The other methods may include the set methods and get methods which sets and gets the described values.

**Lab Manual for Object Oriented Programming**

**Lab No. 5**

# **UNDERSTANDING THE CONCEPT OF OVERLOADING & EXCEPTION HANDLING**

Objectives

To understand basic concept of Method Overloading, Operator overloading and usage Exception handling.

**LAB # 05**

## **Understanding the concept of Overloading & Exception Handling**

## **Introduction**

## Overloading:

C++ allows you to specify more than one definition for a **function** name or an **operator** in the same scope, which is called **method overloading** and **operator overloading** respectively.

An overloaded declaration is a declaration that had been declared with the same name as a previously declared declaration in the same scope, except that both declarations have different arguments and obviously different definition (implementation).

When you call an overloaded **method** or **operator**, the compiler determines the most appropriate definition to use by comparing the argument types you used to call the function or operator with the parameter types specified in the definitions. The process of selecting the most appropriate overloaded function or operator is called **overload resolution**.

### Method overloading in C++:

You can have multiple definitions for the same method name in the same scope. The definition of the function must differ from each other by the types and/or the number of arguments in the argument list. You can not overload function declarations that differ only by return type.

Following is the example where same function **print()** is being used to print different data types:

#include <iostream>

using namespace std;

class printData

{

public:

void print(int i) {

cout << "Printing int: " << i << endl;

}

void print(double f) {

cout << "Printing float: " << f << endl;

}

void print(char\* c) {

cout << "Printing character: " << c << endl;

}

};

int main(void)

{

printData pd;

// Call print to print integer

pd.print(5);

// Call print to print float

pd.print(500.263);

// Call print to print character

pd.print("Hello C++");

return 0;

}

When the above code is compiled and executed, it produces the following result:

Printing int: 5

Printing float: 500.263

Printing character: Hello C++

### Operators overloading in C++:

You can redefine or overload most of the built-in operators available in C++. Thus a programmer can use operators with user-defined types as well.

Overloaded operators are functions with special names the keyword operator followed by the symbol for the operator being defined. Like any other function, an overloaded operator has a return type and a parameter list.

Box operator+(const Box&);

declares the addition operator that can be used to **add** two Box objects and returns final Box object. Most overloaded operators may be defined as ordinary non-member functions or as class member functions. In case we define above function as non-member function of a class then we would have to pass two arguments for each operand as follows:

Box operator+(const Box&, const Box&);

Following is the example to show the concept of operator over loading using a member function. Here an object is passed as an argument whose properties will be accessed using this object, the object which will call this operator can be accessed using **this** operator as explained below:

#include <iostream>

using namespace std;

class Box

{

public:

double getVolume(void)

{

return length \* breadth \* height;

}

void setLength( double len )

{

length = len;

}

void setBreadth( double bre )

{

breadth = bre;

}

void setHeight( double hei )

{

height = hei;

}

// Overload + operator to add two Box objects.

Box operator+(const Box& b)

{

Box box;

box.length = this->length + b.length;

box.breadth = this->breadth + b.breadth;

box.height = this->height + b.height;

return box;

}

private:

double length; // Length of a box

double breadth; // Breadth of a box

double height; // Height of a box

};

// Main function for the program

int main( )

{

Box Box1; // Declare Box1 of type Box

Box Box2; // Declare Box2 of type Box

Box Box3; // Declare Box3 of type Box

double volume = 0.0; // Store the volume of a box here

// box 1 specification

Box1.setLength(6.0);

Box1.setBreadth(7.0);

Box1.setHeight(5.0);

// box 2 specification

Box2.setLength(12.0);

Box2.setBreadth(13.0);

Box2.setHeight(10.0);

// volume of box 1

volume = Box1.getVolume();

cout << "Volume of Box1 : " << volume <<endl;

// volume of box 2

volume = Box2.getVolume();

cout << "Volume of Box2 : " << volume <<endl;

// Add two object as follows:

Box3 = Box1 + Box2;

// volume of box 3

volume = Box3.getVolume();

cout << "Volume of Box3 : " << volume <<endl;

return 0;

}

When the above code is compiled and executed, it produces the following result:

Volume of Box1 : 210

Volume of Box2 : 1560

Volume of Box3 : 5400

### Overloadable/Non-overloadableOperators:

Following is the list of operators which can be overloaded:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| + | - | \* | / | % | ^ |
| & | | | ~ | ! | , | = |
| < | > | <= | >= | ++ | -- |
| << | >> | == | != | && | || |
| += | -= | /= | %= | ^= | &= |
| |= | \*= | <<= | >>= | [] | () |
| -> | ->\* | new | new [] | delete | delete [] |

Following is the list of operators, which can not be overloaded:

|  |  |  |  |
| --- | --- | --- | --- |
| :: | .\* | . | ?: |

## Exception Handling:

An exception is a problem that arises during the execution of a program. A C++ exception is a response to an exceptional circumstance that arises while a program is running, such as an attempt to divide by zero.

Exceptions provide a way to transfer control from one part of a program to another. C++ exception handling is built upon three keywords: **try, catch a**nd **throw**.

#### throw:

A program throws an exception when a problem shows up. This is done using a **throw** keyword.

#### catch:

A program catches an exception with an exception handler at the place in a program where you want to handle the problem. The **catch** keyword indicates the catching of an exception.

#### try:

A **try** block identifies a block of code for which particular exceptions will be activated. It's followed by one or more catch blocks.

Assuming a block will raise an exception, a method catches an exception using a combination of the **try** and **catch** keywords. A try/catch block is placed around the code that might generate an exception. Code within a try/catch block is referred to as protected code, and the syntax for using try/catch looks like the following:

try

{

// protected code

}catch( ExceptionName e1 )

{

// catch block

}catch( ExceptionName e2 )

{

// catch block

}catch( ExceptionName eN )

{

// catch block

}

You can list down multiple **catch** statements to catch different type of exceptions in case your **try** block raises more than one exception in different situations.

### Throwing Exceptions:

Exceptions can be thrown anywhere within a code block using **throw**statements. The operand of the throw statements determines a type for the exception and can be any expression and the type of the result of the expression determines the type of exception thrown.

Following is an example of throwing an exception when dividing by zero condition occurs:

double division(int a, int b)

{

if( b == 0 )

{

throw "Division by zero condition!";

}

return (a/b);

}

### Catching Exceptions:

The **catch** block following the **try** block catches any exception. You can specify what type of exception you want to catch and this is determined by the exception declaration that appears in parentheses following the keyword catch.

try

{

// protected code

}catch( ExceptionName e )

{

// code to handle ExceptionName exception

}

Above code will catch an exception of **ExceptionName** type. If you want to specify that a catch block should handle any type of exception that is thrown in a try block, you must put an ellipsis, ..., between the parentheses enclosing the exception declaration as follows:

try

{

// protected code

}catch(...)

{

// code to handle any exception

}

The following is an example, which throws a division by zero exception and we catch it in catch block.

#include <iostream>

using namespace std;

double division(int a, int b)

{

if( b == 0 )

{

throw "Division by zero condition!";

}

return (a/b);

}

int main ()

{

int x = 50;

int y = 0;

double z = 0;

try {

z = division(x, y);

cout << z << endl;

}catch (const char\* msg) {

cerr << msg << endl;

}

return 0;

}

Because we are raising an exception of type **const char\***, so while catching this exception, we have to use const char\* in catch block. If we compile and run above code, this would produce the following result:

Division by zero condition!

## **Time Boxing**

|  |  |  |
| --- | --- | --- |
| Activity Name | Activity Time | Total Time |
| Login Systems + Setting up Visual Studio Environment | 3 mints + 5 mints | 8 mints |
| Walk through Theory & Tasks | 60 mints | 60 mints |
| Implement Tasks | 80 mints | 80 mints |
| Evaluation Time | 30 mints | 30 mints |
|  | Total Duration | 178 mints |

## **Objectives**

After completing this lab the student should be able to:

* Clearly understand the concept of Method overloading.
* Clearly understand the concept of Operator overloading.
* Clearly understand the purpose and implementation of Exception handling.

## **Lab Tasks/Practical Work**

Write a program which contains a class ‘Calculator’ contains multiple sum method by using method overloading concept also apply exception handling.

Write a program which change ‘+’ operator functionality by using operator overloading concept and also apply exception handling.

**Lab Manual for Object Oriented Programming**

**Lab No. 6**

# **UNDERSTANDING THE IMPLEMENTATION OF STATIC CLASSES & MEMBERS**

Objectives

To understand concept, usage and implementation of Static members

**Lab Manual for Object Oriented Programming**

**Lab No. 7**

# **In-Depth understanding the concept of Inheritance**

Objectives

To understand the concept of inheritance and its types.

**LAB # 07**

## **In-Depth understanding the concept of Inheritance**

## **Introduction**

**Inheritance** is one of the feature of Object-Oriented Programming ([OOPs](http://beginnersbook.com/2013/04/oops-concepts/)). Inheritance allows a class to use the properties and methods of another class. In other words, the derived class inherits the states and behaviors from the base class. The derived class is also called subclass and the base class is also known as super-class. The derived class can add its own additional variables and methods. These additional variable and methods differentiates the derived class from the base class.

Inheritance is a [compile-time](http://beginnersbook.com/2013/04/runtime-compile-time-polymorphism/) mechanism. A super-class can have any number of subclasses. But a subclass can have only one superclass. This is because Java does not support multiple inheritance.

The superclass and subclass have **“is-a”** relationship between them. Let’s have a look at the example below.

### Inheritance Example

Let’s consider a superclass Vehicle. Different vehicles have different features and properties however there few of them are common to all. Speed, color, fuel used, size are few which are common to all. Hence we can create a class ‘Vehicle’ with states and actions that are common to all vehicles. The subclass of this superclass can be any type of vehicle. Example: Class Car  A has all the features of a vehicle. But it has its own attributes which makes it different from other subclasses. By using inheritance we need not rewrite the code that we’ve already used with the Vehicle. The subclass can also be extended. We can make a class ‘Sports Car’ which extends ‘Car’. It inherits the features of both ‘Vehicle’ and ‘Car’.

The keyword used for inheritance is extends. Syntax:

public class ChildClass extends BaseClass {

// derived class methods extend and possibly override

}

### Here is the complete example:

// A class to display the attributes of the vehicle

class Vehicle {

String color;

int speed;

int size;

void attributes() {

System.out.println("Color : " + color);

System.out.println("Speed : " + speed);

System.out.println("Size : " + size);

}

}

// A subclass which extends for vehicle

class Car extends Vehicle {

int CC;

int gears;

void attributescar() {

// The subclass refers to the members of the superclass

System.out.println("Color of Car : " + color);

System.out.println("Speed of Car : " + speed);

System.out.println("Size of Car : " + size);

System.out.println("CC of Car : " + CC);

System.out.println("No of gears of Car : " + gears);

}

}

public class Test {

public static void main(String args[]) {

Car b1 = new Car();

b1.color = "Blue";

b1.speed = 200 ;

b1.size = 22;

b1.CC = 1000;

b1.gears = 5;

b1.attributescar();

}

}

### The output is

Color of Car : Blue

Speed of Car : 200

Size of Car : 22

CC of Car : 1000

No of gears of Car : 5

Note:  
The derived class inherits all the members and methods that are declared as public or protected. If declared as private it can not be inherited by the derived classes. The private members can be accessed only in its own class. The private members can be accessed through assessor methods as shown in the example below. The derived class cannot inherit a member of the base class if the derived class declares another member with the same name.

// A class to display the attributes of the vehicle

class Vehicle {

String color;

private int speed;

private int size;

public int getSize() {

return size;

}

public int getSpeed() {

return speed;

}

public void setSize(int i) {

size = i;

}

public void setSpeed(int i) {

speed = i;

}

}

// A subclass which extends for vehicle

class Car extends Vehicle {

int CC;

int gears;

int color;

void attributescar() {

// Error due to access violation

// System.out.println("Speed of Car : " + speed);

// Error due to access violation

//System.out.println("Size of Car : " + size);

}

}

public class Test {

public static void main(String args[]) {

Car b1 = new Car();

// the subclass can inherit 'color' member of the superclass

b1.color = 500;

b1.setSpeed(200) ;

b1.setSize(22);

b1.CC = 1000;

b1.gears = 5;

// The subclass refers to the members of the superclass

System.out.println("Color of Car : " + b1.color);

System.out.println("Speed of Car : " + b1.getSpeed());

System.out.println("Size of Car : " + b1.getSize());

System.out.println("CC of Car : " + b1.CC);

System.out.println("No of gears of Car : " + b1.gears);

}

}

The output is:

Color of Car : 500

Speed of Car : 200

Size of Car : 22

CC of Car : 1000

No of gears of Car : 5

### Types of inheritance

* [Multilevel Inheritance](http://beginnersbook.com/2013/12/multilevel-inheritance-in-java-with-example/)
* [Multiple Inheritance](http://beginnersbook.com/2013/05/java-multiple-inheritance/)
* [Hybrid Inheritance](http://beginnersbook.com/2013/10/hybrid-inheritance-java-program/)
* [Hierarchical Inheritance](http://beginnersbook.com/2013/10/hierarchical-inheritance-java-program/)

#### Constructors and Inheritance

The [constructor](http://beginnersbook.com/2013/03/constructors-in-java/) in the superclass is responsible for building the object of the superclass and the constructor of the subclass builds the object of subclass. When the subclass constructor is called during object creation, it by default invokes the default constructor of super-class. Hence, in inheritance the objects are constructed top-down. The superclass constructor can be called explicitly using the keyword super, but it should be first statement in a constructor. The keyword super always refers to the superclass immediately above of the calling class in the hierarchy. The use of multiple super keywords to access an ancestor class other than the direct parent is illegal.

class Shape {

private int length;

private int breadth;

public int getBreadth() {

return breadth;

}

public int getLength() {

return length;

}

public void setBreadth(int i) {

breadth = i;

}

public void setLength(int i) {

length = i;

}

// default Constructor

Shape() {

length = 0;

breadth = 0;

System.out.println("Inside default constructor of Shape ");

}

// Parameterized Constructor

Shape(int len, int bdth) {

length = len;

breadth = bdth;

System.out.println("Inside constructor of Shape ");

System.out.println("length : " + length);

System.out.println("breadth : " + breadth);

}

}

// A subclass which extends for shape

class Rectangle extends Shape {

private String type;

// default Constructor

Rectangle() {

super();

type = null;

System.out.println("Inside default constructor of rectangle ");

}

// Parameterized Constructor

Rectangle(String ty, int len, int bdth) {

super (len, bdth);

System.out.println("Inside constructor of rectangle ");

System.out.println("length : " + len);

System.out.println("breadth : " + bdth);

System.out.println("type : " + type);

}

public String getType() {

return type;

}

public void setType(String string) {

type = string;

}

}

// A subclass which extends for rectangle

class ColoredRectangle extends Rectangle {

private String color;

/\* default Constructor\*/

ColoredRectangle() {

super();

color = null;

System.out.println("Inside default constructor of coloredRectangle");

}

// Parameterized Constructor

ColoredRectangle(String c, String ty, int len, int bdth) {

super (ty, len, bdth);

System.out.println("Inside constructor of coloredRectangle ");

System.out.println("length : " + len);

System.out.println("breadth : " + bdth);

System.out.println("type : " + ty);

}

public String getColor() {

return color;

}

public void setColor(String string) {

color = string;

}

}

public class Test {

public static void main(String args[]) {

ColoredRectangle CR = new ColoredRectangle();

ColoredRectangle CR2 = new ColoredRectangle("Red","Big", 5, 2 );

}

}

#### **The output is:**

Inside default constructor of Shape

Inside default constructor of rectangle

Inside default constructor of coloredRectangle

Inside constructor of Shape

length : 5

breadth : 2

Inside constructor of rectangle

length : 5

breadth : 2

type : null

Inside constructor of coloredRectangle

length : 5

breadth : 2

type : Big

#### Inheritance and Method Overriding

By using super we can access the [overridden method](http://beginnersbook.com/2014/01/method-overriding-in-java-with-example/) in the super class.

class Shape {

private int length;

private int breadth;

// default Constructor

Shape() {

length = 0;

breadth = 0;

}

// Parameterized Constructor

Shape(int len, int bdth) {

length = len;

breadth = bdth;

}

void showattributes() {

System.out.println("length : " + length);

System.out.println("breadth : " + breadth);

}

}

// A subclass which extends for shape

class Rectangle extends Shape {

private String type;

/\* default Constructor

\*/

Rectangle() {

type = null;

}

// Parameterized Constructor

Rectangle(String ty, int len, int bdth) {

super(len,bdth);

type = ty;

}

void showattributes() {

// showattributes() of class Shape is called

super.showattributes();

System.out.println("type : " + type);

}

}

public class Test {

public static void main(String args[]) {

Rectangle rect = new Rectangle("Blue",5,7);

// showattributes() in rectangle is called

rect.showattributes();

}

}

The output is :

length : 5

breadth : 7

type : Blue

#### Inheritance & Abstract Classes

The superclasses are more general than their subclasses. Usually, the superclasses are made abstract so that the objects of its prototype cannot be made. So the objects of only the subclasses can be used. To make a class abstract, the keyword abstract is used in the class definition.

Abstract methods are methods which do not have method statements. The subclasse provides the method statements. The methods provided by the superclass needs to be overridden by the subclass. The class that has at least one abstract method should be made abstract. The [**abstract class**](http://beginnersbook.com/2013/05/java-abstract-class-method/) can not be instantiated because it does not define a complete implementation.

public abstract class {

….

}

#### **Using Final with methods**:

We can prevent a method from being overridden by using the keyword final at the start of its declaration. Final methods can not be overridden.

public abstract void methodname();

class Shape {

final void showattributes() {

System.out.println("Inside class shape ");

}

}

// A subclass which extends for shape

class Rectangle extends Shape {

void showattributes() { // Cannot override the final method

System.out.println("Inside class rectangle");

}

}

The method showattributes() cannot be overridden in the class rectangle because it is declared as final in class shape. It shows an error when we try to override it.

#### **Using Final with class**:

We can also prevent inheritance by making a class final. When a class is declared as final, its methods also become final. An abstract class cannot be declared as final because an abstract class is incomplete and its subclasses need to provide the implementation.

final class shape {

void showattributes() {

System.out.println("Inside class shape ");

}

}

/\* A subclass which extends for shape

\*/

class rectangle extends shape {

// The type rectangle cannot subclass the final class shape

void showattributes() {

System.out.println("Inside class rectangle");

}

}

The class shape cannot be inherited because it is declared as final. It will show an error when we try to inherit it.

###### **Reference:** Java 2: the complete reference: fifth Edition http://java.sun.com

## **Time Boxing**

|  |  |  |
| --- | --- | --- |
| Activity Name | Activity Time | Total Time |
| Login Systems + Setting up Visual Studio Environment | 3 mints + 5 mints | 8 mints |
| Walk through Theory & Tasks | 60 mints | 60 mints |
| Implement Tasks | 80 mints | 80 mints |
| Evaluation Time | 30 mints | 30 mints |
|  | Total Duration | 178 mints |

## **Objectives**

After completing this lab the student should be able to:

* Clearly understand the purpose and benefits of Inheritance & its types.

## **Lab Tasks/Practical Work**

1) Write a program that inherits a class named Alien and Pirates from a parent class Human. The human class has its own features like, Human can sleep, walk, talk etc. the Alien and Pirates class inheriting these functionalities as well as they have their characteristics, thus explaining the concepts of inheritance .

2) Write a program containing faculty class and student class, incorporating their own methods. Create a teaching assistant class inheriting from both faculty and student class.

**Lab Manual for Object Oriented Programming**

**Lab No. 8**

# **Implementation & understanding of Multiple Inheritance**

Objectives

To understand the concept of multiple inheritance.

**LAB # 08**

**Implementation & understanding of Multiple Inheritance**

## **Introduction**

Multiple Inheritance is a feature of C++ where a class can inherit from more than one classes.

The constructors of inherited classes are called in the same order in which they are inherited. For example, in the following program, B’s constructor is called before A’s constructor.

|  |
| --- |
| #include<iostream>  using namespace std;    class A  {  public:    A()  { cout << "A's constructor called" << endl; }  };    class B  {  public:    B()  { cout << "B's constructor called" << endl; }  };    class C: public B, public A  // Note the order  {  public:    C()  { cout << "C's constructor called" << endl; }  };    int main()  {      C c;      return 0;  } |

Output:

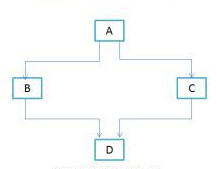
B's constructor called

A's constructor called

C's constructor called

C++ , Common lisp and few other languages supports multiple inheritance while java doesn’t support it. It is just to **remove ambiguity**, because **multiple inheritance** can cause ambiguity in few scenarios. One of the most common scenario is **Diamond problem.**

What is diamond problem?  
Consider the below diagram which shows multiple inheritance as Class D extends both Class B & C. Now lets assume we have a method in class A andclass B & C overrides that method in their own way. **Wait!! here the problem comes**– Because D is extending both B & C so if D wants to use the same method which method would be called (the overridden method of B or the overridden method of C). Ambiguity. That’s the main reason why Java doesn’t support multiple inheritance.



##### How to achieve multiple inheritance in Java using interfaces?

interface X

{

public void myMethod();

}

interface Y

{

public void myMethod();

}

class Demo implements X, Y

{

public void myMethod()

{

System.out.println(" Multiple inheritance example using interfaces");

}

}

As you can see that the class implemented two interfaces. A class can implement any number of interfaces. In this case there is no ambiguity even though both the interfaces are having same method. Why? Because methods in an interface are always [abstract](http://beginnersbook.com/2013/05/java-abstract-class-method/) by default, which doesn’t let them to give their implementation (or method definition ) in interface itself.

## **Time Boxing**

|  |  |  |
| --- | --- | --- |
| Activity Name | Activity Time | Total Time |
| Login Systems + Setting up Visual Studio Environment | 3 mints + 5 mints | 8 mints |
| Walk through Theory & Tasks | 60 mints | 60 mints |
| Implement Tasks | 80 mints | 80 mints |
| Evaluation Time | 30 mints | 30 mints |
|  | Total Duration | 178 mints |

## **Objectives**

After completing this lab the student should be able to:

* Clearly understand the concept of multiple inherritance.

## **Lab Tasks/Practical Work**

1) Write a program in which CAR class and BOAT class are inherited from Land Vehicle as well as water Vehicle whereas Amphibious vehicle class inherited from both Land Vehicle and Water Vehicle (as it can incorporate the characteristics of both), which are further inherited from Vehicle Class.

2) Write a program containing Fish class that has its own features and a Mermaid Class which possess its own features , create a Mermaid class that incorporates the features of both fish class and the woman class as well as it possess its own features.

**Lab Manual for Object Oriented Programming**

**Lab No. 9**

# **Understanding the concept of Composition, Aggregation and Association**

Objectives

To understand concept of composition, aggregation and association.

**LAB # 09**

## **Understanding the concept of Composition, Aggregation and Association**

## **Introduction**

### Extracting real world relationships from a requirement

The whole point of OOP is that your code replicates real world objects, thus making your code readable and maintainable. When we say real world, the real world has relationships. Let’s consider the simple requirement listed below:

1. Manager is an employee of XYZ limited corporation.
2. Manager uses a swipe card to enter XYZ premises.
3. Manager has workers who work under him.
4. Manager has the responsibility of ensuring that the project is successful.
5. Manager's salary will be judged based on project success.

If you flesh out the above five point requirement, we can easily visualize four relationships:-

1. Inheritance
2. Aggregation
3. Association
4. Composition

Let’s understand them one by one.

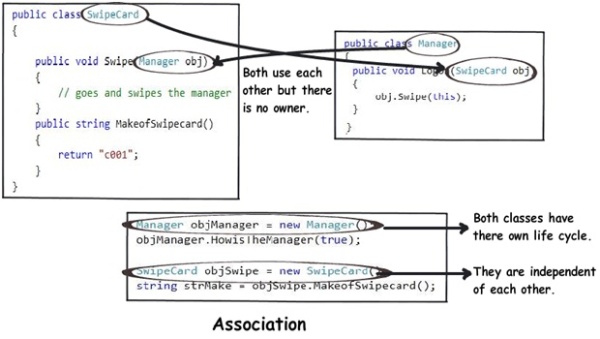
### Requirement 1: The IS A relationship

If you look at the first requirement (Manager is an employee of XYZ limited corporation), it’s a parent child relationship or inheritance relationship. The sentence above specifies that Manager is a type of employee, in other words we will have two classes: parent class Employee, and a child class Manager which will inherit from the Employee class.

**Note**: The scope of this article is only limited to aggregation, association, and composition. We will not discuss inheritance in this article as it is pretty straightforward and I am sure you can get 1000s of articles on the net which will help you in understanding it.

### Requirement 2: The Using relationship: Association

Requirement 2 is an interesting requirement (Manager uses a swipe card to enter XYZ premises). In this requirement, the manager object and the swipe card object use each other but they have their own object life time. In other words, they can exist without each other. The most important point in this relationship is that there is no single owner.



The above diagram shows how the SwipeCard class uses the Manager class and the Manager class uses theSwipeCard class. You can also see how we can create objects of the Manager class and SwipeCard class independently and they can have their own object life time.

This relationship is called the “Association” relationship.

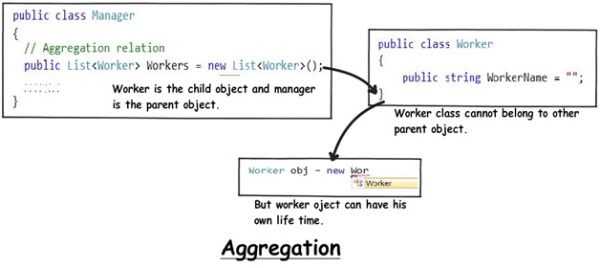
### Requirement 3: The Using relationship with Parent: Aggregation

The third requirement from our list (Manager has workers who work under him) denotes the same type of relationship like association but with a difference that one of them is an owner. So as per the requirement, theManager object will own Worker objects.

The child Worker objects can not belong to any other object. For instance, a Worker object cannot belong to aSwipeCard object.

But… the Worker object can have its own life time which is completely disconnected from the Manager object. Looking from a different perspective, it means that if the Manager object is deleted, the Worker object does not die.

This relationship is termed as an “Aggregation” relationship.



### Requirements 4 and 5: The Death relationship: Composition

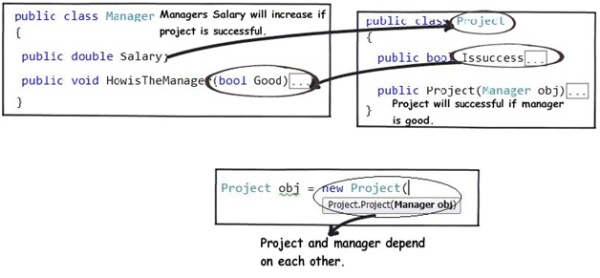
The last two requirements are actually logically one. If you read closely, the requirements are as follows:

1. Manager has the responsibility of ensuring that the project is successful.
2. Manager's salary will be judged based on project success.

Below is the conclusion from analyzing the above requirements:

1. Manager and the project objects are dependent on each other.
2. The lifetimes of both the objects are the same. In other words, the project will not be successful if the manager is not good, and the manager will not get good increments if the project has issues.

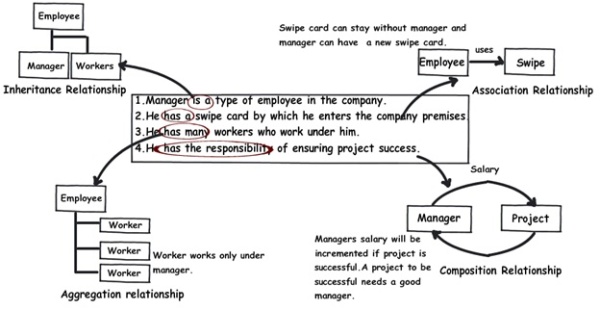
Below is how the class formation will look like. You can also see that when I go to create the project object, it needs the manager object.



This relationship is termed as the composition relationship. In this relationship, both objects are heavily dependent on each other. In other words, if one goes for garbage collection the other also has to be garbage collected, or putting from a different perspective, the lifetime of the objects are the same. That’s why I have put in the heading “Death” relationship.

### Putting things together

Below is a visual representation of how the relationships have emerged from the requirements.



### The source code

You can download the sample [source code](http://www.codeproject.com/KB/architecture/330447/Organization.zip)  
<http://www.codeproject.com/Articles/330447/Understanding-Association-Aggregation-and-Composit>

### Summarizing

To avoid confusion henceforth for these three terms, I have put forward a table below which will help us compare them from three angles: owner, lifetime, and child object.

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Association** | **Aggregation** | **Composition** |
| **Owner** | No owner | Single owner | Single owner |
| **Life time** | Have their own lifetime | Have their own lifetime | Owner's life time |
| **Child object** | Child objects all are independent | Child objects belong to a single parent | Child objects belong to a single parent |

## **Time Boxing**

|  |  |  |
| --- | --- | --- |
| Activity Name | Activity Time | Total Time |
| Login Systems + Setting up Visual Studio Environment | 3 mints + 5 mints | 8 mints |
| Walk through Theory & Tasks | 60 mints | 60 mints |
| Implement Tasks | 80 mints | 80 mints |
| Evaluation Time | 30 mints | 30 mints |
|  | Total Duration | 178 mints |

## **Objectives**

After completing this lab the student should be able to:

* Clearly understand the purpose and benefits of composition, aggregation and association.

## **Lab Tasks/Practical Work**

1. Write a program in which create two parent classes faculty and student and inherit them with BCE students and teaching assistant’s class respectively.
2. Write a program and create the objects of classes in class car to explain the concept of composition. Create several classes as engine, doors, capacity and wheel having their individual methods attributes. The object of these classes are created in a car class and they are set as public. The object of this car class is created in Main method and this with the help of this object we can call other classes as well and can use their functionalities.

**Lab Manual for Object Oriented Programming**

**Lab No. 10**

# **Understanding the concept of Polymorphism**

Objectives

To understand concept of Polymorphism.

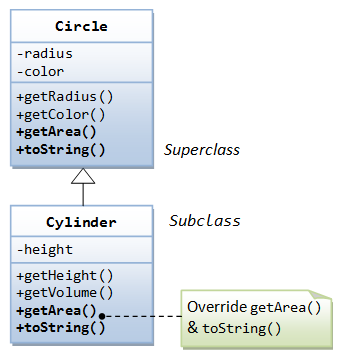
**LAB # 10**

**Understanding the concept of Polymorphism**

## **Introduction**

The word "polymorphism" means "many forms". It comes from Greek word "poly" (means many) and "morphos" (means form). For examples, in chemistry, carbon exhibits polymorphism because it can be found in more than one form: graphite and diamond. Each of the form has it own distinct properties.

##### Substitutability



A subclass possesses all the attributes and operations of its superclass (because a subclass inherited all attributes and operations from its superclass). This means that a subclass object can do whatever its superclass can do. As a result, we can substitute a subclass instance when a superclass instance is expected, and everything shall work fine. This is called substitutability.

In our earlier example of Circle and Cylinder: Cylinder is a subclass of Circle. We can say that Cylinder "is-a" Circle (actually, it "is-more-than-a" Circle). Subclass-superclass exhibits a so called "is-a" relationship.

Via substitutability, we can create an instance of Cylinder, and assign it to a Circle (its superclass) reference, as follows:

// Substitute a subclass instance to its superclass reference

Circle c1 = new Cylinder(5.0);

You can invoke all the methods defined in the Circle class for the reference c1, (which is actually holding a Cylinder object), e.g. c1.getRadius()and c1.getColor(). This is because a subclass instance possesses all the properties of its superclass.

However, you cannot invoke methods defined in the Cylinder class for the reference c1, e.g. c1.getHeight() and c1.getVolume(). This is because c1 is a reference to the Circle class, which does not know about methods defined in the subclass Cylinder.

c1 is a reference to the Circle class, but holds an object of its subclass Cylinder. The reference c1, however, retains its internal identity. In our example, the subclass Cylinder overrides methods getArea() and toString(). c1.getArea() or c1.toString() invokes the overridden version defined in the subclass Cylinder, instead of the version defined in Circle. This is because c1 is in fact holding a Cylinder object internally.

###### Summary

1. A subclass instance can be assigned (substituted) to a superclass' reference.
2. Once substituted, we can invoke methods defined in the superclass; we cannot invoke methods defined in the subclass.
3. However, if the subclass overrides inherited methods from the superclass, the subclass (overridden) versions will be invoked.

Below is a real-life example of polymorphism. For the example program and more details on this OOP concept refer polymorphism in java and runtime & compile time polymorphism.

* Polymorphism means to process objects differently based on their data type.
* In other words it means, one method with multiple implementation, for a certain class of action. And which implementation to be used is decided at runtime depending upon the situation (i.e., data type of the object)
* This can be implemented by designing a generic interface, which provides generic methods for a certain class of action and there can be multiple classes, which provides the implementation of these generic methods.

Lets us look at same example of a car. A car have a gear transmission system. It has four front gears and one backward gear. When the engine is accelerated then depending upon which gear is engaged different amount power and movement is delivered to the car.

Polymorphism could be static and dynamic both. Overloading is static polymorphism while, overriding is dynamic polymorphism.

Overloading in simple words means two methods having same method name but takes different input parameters. This called static because, which method to be invoked will be decided at the time of compilation

Overriding means a derived class is implementing a method of its super class.

## **Time Boxing**

|  |  |  |
| --- | --- | --- |
| Activity Name | Activity Time | Total Time |
| Login Systems + Setting up Visual Studio Environment | 3 mints + 5 mints | 8 mints |
| Walk through Theory & Tasks | 60 mints | 60 mints |
| Implement Tasks | 80 mints | 80 mints |
| Evaluation Time | 30 mints | 30 mints |
|  | Total Duration | 178 mints |

## **Objectives**

After completing this lab the student should be able to:

* Clearly understand the concept and purpose of Polymorphism

## **Lab Tasks/Practical Work**

1. Write a program that contains parent class employee that possess virtual methods and 4 derived classes and calculate the salaries of each by calling parent object using c++.

**Lab Manual for Object Oriented Programming**

**Lab No. 11**

# **Understanding the concept of Abstract Methods and Classes**

Objectives

To understand the concept of Abstract methods and classes.

**LAB # 11**

## **Understanding the concept of Abstract Methods and Classes**

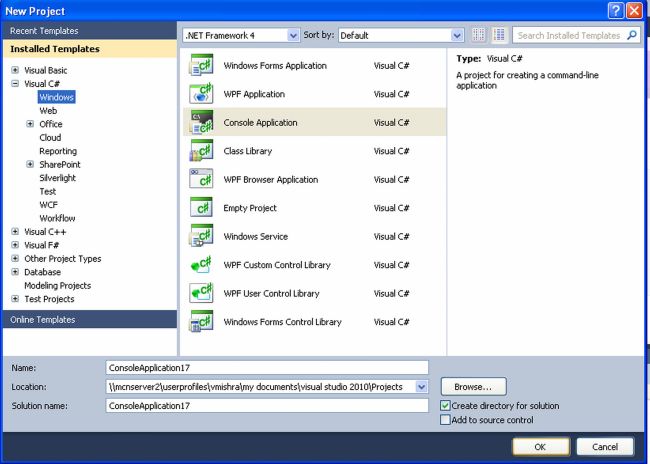
## **Introduction**

Abstract classes and abstract methods. With the help of abstraction we can achieve dynamic polymorphism i.e. the same method name in different classes but the same signature.

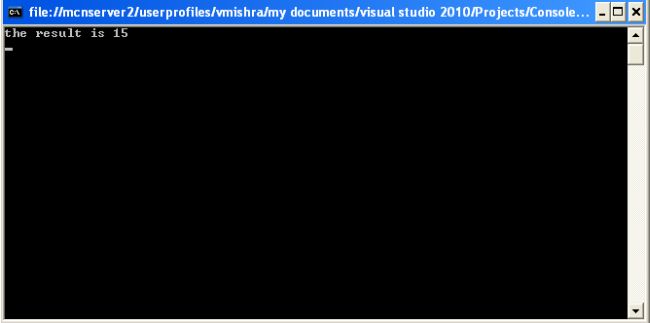
### Abstract class:

If a class is defined as abstract then we can't create an instance of that class. By the creation of the derived class object where an abstract class is inherit from, we can call the method of the abstract class.

Let's take an example: First of all, select a console application as follows:

  
  
and our code window looks like:  
  
using System;  
using System.Collections.Generic;  
using System.Linq;  
using System.Text;  
namespace ConsoleApplication13  
{  
abstract class mcn  
{  
   public int add(int a, int b)  
    {  
      return (a + b);  
    }  
}  
class mcn1 : mcn  
{  
  public int mul(int a, int b)  
  {  
   return a \* b;  
  }  
}  
class test  
{  
  static void Main(string[] args)  
{  
  mcn1 ob = new mcn1();  
  int result = ob.add(5, 10);  
  Console.WriteLine("the result is {0}", result);  
}  
}  
}

In the above program we can call the method of the abstract class mcn with the help of an object of the mcn1 class which inherits from the class mcn. When we run the above program the output is the addition of 5 & 10 (i.e. 15) which is shown as:



### Abstract Method

An Abstract method is a method without a body. The implementation of an abstract method is done by a derived class. When the derived class inherits the abstract method from the abstract class, it must override the abstract method. This requirement is enforced at compile time and is also called dynamic polymorphism.

The syntax of using the abstract method is as follows:

**<access-modifier>abstract<return-type>method name (parameter)**

The abstract method is declared by adding the abstract modifier the method.

using System;  
using System.Collections.Generic;  
using System.Linq;  
using System.Text;  
namespace ConsoleApplication14  
{  
  abstract class test1  
  {  
    public int add(int i, int j)  
    {  
      return i+j;   
    }  
    public abstract int mul(int i, int j);  
    }  
    class test2:test1  
    {  
     public override int mul(int i, int j)  
     {  
      return i\*j;  
     }  
  }  
  class test3:test1   
   {  
    public override int mul(int i, int j)  
    {  
      return i-j;   
    }  
   }  
  class test4:test2   
  {  
    public override int mul(int i, int j)  
   {  
    return i+j;  
   }  
  }  
  class myclass  
 {  
  public static void main (string[] args)  
  {  
    test2 ob= new test4();  
    int a = ob.mul(2,4);  
    test1 ob1= new test2();  
    int b = ob1.mul(4,2);  
    test1 ob2= new test3();  
    int c = ob2.mul(4,2);  
    Console.Write("{0},{1},{2}",a,b,c);  
    Console.ReadLine ();  
  }  
}  
}

In the above program, one method i.e. mul can perform various functions depending on the value passed as parameters by creating an object of various classes which inherit other classes. Hence we can achieve dynamic polymorphism with the help of an abstract method.

## **Time Boxing**

|  |  |  |
| --- | --- | --- |
| Activity Name | Activity Time | Total Time |
| Login Systems + Setting up Visual Studio Environment | 3 mints + 5 mints | 8 mints |
| Walk through Theory & Tasks | 60 mints | 60 mints |
| Implement Tasks | 80 mints | 80 mints |
| Evaluation Time | 30 mints | 30 mints |
|  | Total Duration | 178 mints |

## **Objectives**

After completing this lab the student should be able to:

* Clearly understand the purpose and benefits of abstract classes and methods.

## **Lab Tasks/Practical Work**

1. Write a program for exam department which provide abstract class and method of Exam type which contains general methods related to exams and can be used by different department for conducting exams.

**Lab Manual for Object Oriented Programming**

**Lab No. 12**

# **IMPLEMENTATION OF INTERFACES**

Objectives

To understand concept, purpose and implementation of interfaces.

**LAB # 12**

**Implementation of Interfaces**

## **Introduction**

As you've already learned, objects define their interaction with the outside world through the methods that they expose. Methods form the object's *interface* with the outside world; the buttons on the front of your television set, for example, are the interface between you and the electrical wiring on the other side of its plastic casing. You press the "power" button to turn the television on and off.

In its most common form, an interface is a group of related methods with empty bodies. A bicycle's behavior, if specified as an interface, might appear as follows:

interface Bicycle {

// wheel revolutions per minute

void changeCadence(int newValue);

void changeGear(int newValue);

void speedUp(int increment);

void applyBrakes(int decrement);

}

To implement this interface, the name of your class would change (to a particular brand of bicycle, for example, such as ACMEBicycle), and you'd use the implements keyword in the class declaration:

class ACMEBicycle **implements** Bicycle {

int cadence = 0;

int speed = 0;

int gear = 1;

// The compiler will now require that methods

// changeCadence, changeGear, speedUp, and applyBrakes

// all be implemented. Compilation will fail if those

// methods are missing from this class.

void changeCadence(int newValue) {

cadence = newValue;

}

void changeGear(int newValue) {

gear = newValue;

}

void speedUp(int increment) {

speed = speed + increment;

}

void applyBrakes(int decrement) {

speed = speed - decrement;

}

void printStates() {

System.out.println("cadence:" +

cadence + " speed:" +

speed + " gear:" + gear);

}

}

Implementing an interface allows a class to become more formal about the behavior it promises to provide. Interfaces form a contract between the class and the outside world, and this contract is enforced at build time by the compiler. If your class claims to implement an interface, all methods defined by that interface must appear in its source code before the class will successfully compile.

## **Time Boxing**

|  |  |  |
| --- | --- | --- |
| Activity Name | Activity Time | Total Time |
| Login Systems + Setting up Visual Studio Environment | 3 mints + 5 mints | 8 mints |
| Walk through Theory & Tasks | 60 mints | 60 mints |
| Implement Tasks | 80 mints | 80 mints |
| Evaluation Time | 30 mints | 30 mints |
|  | Total Duration | 178 mints |

## **Objectives**

After completing this lab the student should be able to:

* Clearly understand the purpose and benefits of Interfaces.

## **Lab Tasks/Practical Work**

1. Write a program which implements a interface of Banking System by having all standard functionalities and will be implemented by branches.

**Lab Manual for Object Oriented Programming**

**Lab No. 13**

# **INTRODUCTION TO GUI BASED PROGRAMMING**

Objectives

To understand the implementation of Windows Form Application toolbox.

**LAB # 12**

**Introduction to GUI Based Programming**

## **Introduction**

Windows Forms offers controls and components that perform a number of functions. The following table lists the Windows Forms controls and components according to general function. In addition, where multiple controls exist that serve the same function, the recommended control is listed with a note regarding the control it superseded. In a separate subsequent table, the superseded controls are listed with their recommended replacements.

|  |
| --- |
| Note**Note** |
| The following tables do not list every control or component you can use in Windows Forms; for a more comprehensive list, see [Controls to Use on Windows Forms](https://msdn.microsoft.com/en-us/library/3xdhey7w(v=vs.100).aspx) |

### [Recommended Controls and Components by Function](javascript:void(0))

|  |  |  |
| --- | --- | --- |
| Function | Control | Description |
| Data display | [DataGridView](https://msdn.microsoft.com/en-us/library/system.windows.forms.datagridview(v=vs.100).aspx)control | The [DataGridView](https://msdn.microsoft.com/en-us/library/system.windows.forms.datagridview(v=vs.100).aspx) control provides a customizable table for displaying data. The [DataGridView](https://msdn.microsoft.com/en-us/library/system.windows.forms.datagridview(v=vs.100).aspx) class enables customization of cells, rows, columns, and borders.  Note**Note**  The [DataGridView](https://msdn.microsoft.com/en-us/library/system.windows.forms.datagridview(v=vs.100).aspx) control provides numerous basic and advanced features that are missing in the [DataGrid](https://msdn.microsoft.com/en-us/library/system.windows.forms.datagrid(v=vs.100).aspx)control. For more information, see [Differences Between the Windows Forms DataGridView and DataGrid Controls](https://msdn.microsoft.com/en-us/library/ms171628(v=vs.100).aspx) |
| Data binding and navigation | [BindingSource](https://msdn.microsoft.com/en-us/library/system.windows.forms.bindingsource(v=vs.100).aspx) component | Simplifies binding controls on a form to data by providing currency management, change notification, and other services. |
|  | [BindingNavigator](https://msdn.microsoft.com/en-us/library/system.windows.forms.bindingnavigator(v=vs.100).aspx) control | Provides a toolbar-type interface to navigate and manipulate data on a form. |
| Text editing | [TextBox](https://msdn.microsoft.com/en-us/library/system.windows.forms.textbox(v=vs.100).aspx)control | Displays text entered at design time that can be edited by users at run time, or changed programmatically. |
|  | [RichTextBox](https://msdn.microsoft.com/en-us/library/system.windows.forms.richtextbox(v=vs.100).aspx)control | Enables text to be displayed with formatting in plain text or rich-text format (RTF). |
|  | [MaskedTextBox](https://msdn.microsoft.com/en-us/library/system.windows.forms.maskedtextbox(v=vs.100).aspx) control | Constrains the format of user input |
| Information display (read-only) | [Label](https://msdn.microsoft.com/en-us/library/system.windows.forms.label(v=vs.100).aspx) control | Displays text that users cannot directly edit. |
|  | [LinkLabel](https://msdn.microsoft.com/en-us/library/system.windows.forms.linklabel(v=vs.100).aspx)control | Displays text as a Web-style link and triggers an event when the user clicks the special text. Usually the text is a link to another window or a Web site. |
|  | [StatusStrip](https://msdn.microsoft.com/en-us/library/system.windows.forms.statusstrip(v=vs.100).aspx)control | Displays information about the application's current state using a framed area, usually at the bottom of a parent form. |
|  | [ProgressBar](https://msdn.microsoft.com/en-us/library/system.windows.forms.progressbar(v=vs.100).aspx)control | Displays the current progress of an operation to the user. |
| Web page display | [WebBrowser](https://msdn.microsoft.com/en-us/library/system.windows.forms.webbrowser(v=vs.100).aspx)control | Enables the user to navigate Web pages inside your form. |
| Selection from a list | [CheckedListBox](https://msdn.microsoft.com/en-us/library/system.windows.forms.checkedlistbox(v=vs.100).aspx) control | Displays a scrollable list of items, each accompanied by a check box. |
|  | [ComboBox](https://msdn.microsoft.com/en-us/library/system.windows.forms.combobox(v=vs.100).aspx)control | Displays a drop-down list of items. |
|  | [DomainUpDown](https://msdn.microsoft.com/en-us/library/system.windows.forms.domainupdown(v=vs.100).aspx) control | Displays a list of text items that users can scroll through with up and down buttons. |
|  | [ListBox](https://msdn.microsoft.com/en-us/library/system.windows.forms.listbox(v=vs.100).aspx)control | Displays a list of text and graphical items (icons). |
|  | [ListView](https://msdn.microsoft.com/en-us/library/system.windows.forms.listview(v=vs.100).aspx)control | Displays items in one of four different views. Views include text only, text with small icons, text with large icons, and a details view. |
|  | [NumericUpDown](https://msdn.microsoft.com/en-us/library/system.windows.forms.numericupdown(v=vs.100).aspx) control | Displays a list of numerals that users can scroll through with up and down buttons. |
|  | [TreeView](https://msdn.microsoft.com/en-us/library/system.windows.forms.treeview(v=vs.100).aspx)control | Displays a hierarchical collection of node objects that can consist of text with optional check boxes or icons. |
| Graphics display | [PictureBox](https://msdn.microsoft.com/en-us/library/system.windows.forms.picturebox(v=vs.100).aspx)control | Displays graphical files, such as bitmaps and icons, in a frame. |
| Graphics storage | [ImageList](https://msdn.microsoft.com/en-us/library/system.windows.forms.imagelist(v=vs.100).aspx)control | Serves as a repository for images. [ImageList](https://msdn.microsoft.com/en-us/library/system.windows.forms.imagelist(v=vs.100).aspx) controls and the images they contain can be reused from one application to the next. |
| Value setting | [CheckBox](https://msdn.microsoft.com/en-us/library/system.windows.forms.checkbox(v=vs.100).aspx)control | Displays a check box and a label for text. Generally used to set options. |
|  | [CheckedListBox](https://msdn.microsoft.com/en-us/library/system.windows.forms.checkedlistbox(v=vs.100).aspx) control | Displays a scrollable list of items, each accompanied by a check box. |
|  | [RadioButton](https://msdn.microsoft.com/en-us/library/system.windows.forms.radiobutton(v=vs.100).aspx)control | Displays a button that can be turned on or off. |
|  | [TrackBar](https://msdn.microsoft.com/en-us/library/system.windows.forms.trackbar(v=vs.100).aspx)control | Allows users to set values on a scale by moving a "thumb" along a scale. |
| Date setting | [DateTimePicker](https://msdn.microsoft.com/en-us/library/system.windows.forms.datetimepicker(v=vs.100).aspx) control | Displays a graphical calendar to allow users to select a date or a time. |
|  | [MonthCalendar](https://msdn.microsoft.com/en-us/library/system.windows.forms.monthcalendar(v=vs.100).aspx) control | Displays a graphical calendar to allow users to select a range of dates. |
| Dialog boxes | [ColorDialog](https://msdn.microsoft.com/en-us/library/system.windows.forms.colordialog(v=vs.100).aspx)control | Displays the color picker dialog box that allows users to set the color of an interface element. |
|  | [FontDialog](https://msdn.microsoft.com/en-us/library/system.windows.forms.fontdialog(v=vs.100).aspx)control | Displays a dialog box that allows users to set a font and its attributes. |
|  | [OpenFileDialog](https://msdn.microsoft.com/en-us/library/system.windows.forms.openfiledialog(v=vs.100).aspx) control | Displays a dialog box that allows users to navigate to and select a file. |
|  | [PrintDialog](https://msdn.microsoft.com/en-us/library/system.windows.forms.printdialog(v=vs.100).aspx)control | Displays a dialog box that allows users to select a printer and set its attributes. |
|  | [PrintPreviewDialog](https://msdn.microsoft.com/en-us/library/system.windows.forms.printpreviewdialog(v=vs.100).aspx) control | Displays a dialog box that displays how a control [PrintDocument](https://msdn.microsoft.com/en-us/library/system.drawing.printing.printdocument(v=vs.100).aspx) component will appear when printed. |
|  | [FolderBrowserDialog](https://msdn.microsoft.com/en-us/library/system.windows.forms.folderbrowserdialog(v=vs.100).aspx)control | Displays a dialog that allows users to browse, create, and eventually select a folder |
|  | [SaveFileDialog](https://msdn.microsoft.com/en-us/library/system.windows.forms.savefiledialog(v=vs.100).aspx) control | Displays a dialog box that allows users to save a file. |
| Menu controls | [MenuStrip](https://msdn.microsoft.com/en-us/library/system.windows.forms.menustrip(v=vs.100).aspx)control | Creates custom menus.  Note**Note**  The [MenuStrip](https://msdn.microsoft.com/en-us/library/system.windows.forms.menustrip(v=vs.100).aspx) is designed to replace the [MainMenu](https://msdn.microsoft.com/en-us/library/system.windows.forms.mainmenu(v=vs.100).aspx) control. |
|  | [ContextMenuStrip](https://msdn.microsoft.com/en-us/library/system.windows.forms.contextmenustrip(v=vs.100).aspx) control | Creates custom context menus.  Note**Note**  The [ContextMenuStrip](https://msdn.microsoft.com/en-us/library/system.windows.forms.contextmenustrip(v=vs.100).aspx) is designed to replace the [ContextMenu](https://msdn.microsoft.com/en-us/library/system.windows.forms.contextmenu(v=vs.100).aspx) control. |
| Commands | [Button](https://msdn.microsoft.com/en-us/library/system.windows.forms.button(v=vs.100).aspx)control | Starts, stops, or interrupts a process. |
|  | [LinkLabel](https://msdn.microsoft.com/en-us/library/system.windows.forms.linklabel(v=vs.100).aspx)control | Displays text as a Web-style link and triggers an event when the user clicks the special text. Usually the text is a link to another window or a Web site. |
|  | [NotifyIcon](https://msdn.microsoft.com/en-us/library/system.windows.forms.notifyicon(v=vs.100).aspx)control | Displays an icon in the status notification area of the taskbar that represents an application running in the background. |
|  | [ToolStrip](https://msdn.microsoft.com/en-us/library/system.windows.forms.toolstrip(v=vs.100).aspx)control | Creates toolbars that can have a Microsoft Windows XP, Microsoft Office, Microsoft Internet Explorer, or custom look and feel, with or without themes, and with support for overflow and run-time item reordering.  Note**Note**  The [ToolStrip](https://msdn.microsoft.com/en-us/library/system.windows.forms.toolstrip(v=vs.100).aspx) control is designed to replace the [ToolBar](https://msdn.microsoft.com/en-us/library/system.windows.forms.toolbar(v=vs.100).aspx) control. |
| User Help | [HelpProvider](https://msdn.microsoft.com/en-us/library/system.windows.forms.helpprovider(v=vs.100).aspx)component | Provides pop-up or online Help for controls. |
|  | [ToolTip](https://msdn.microsoft.com/en-us/library/system.windows.forms.tooltip(v=vs.100).aspx)component | Provides a pop-up window that displays a brief description of a control's purpose when the user rests the pointer on the control. |
| Grouping other controls | [Panel](https://msdn.microsoft.com/en-us/library/system.windows.forms.panel(v=vs.100).aspx) control | Groups a set of controls on an unlabeled, scrollable frame. |
|  | [GroupBox](https://msdn.microsoft.com/en-us/library/system.windows.forms.groupbox(v=vs.100).aspx)control | Groups a set of controls (such as radio buttons) on a labeled, nonscrollable frame. |
|  | [TabControl](https://msdn.microsoft.com/en-us/library/system.windows.forms.tabcontrol(v=vs.100).aspx)control | Provides a tabbed page for organizing and accessing grouped objects efficiently. |
|  | [SplitContainer](https://msdn.microsoft.com/en-us/library/system.windows.forms.splitcontainer(v=vs.100).aspx) control | Provides two panels separated by a movable bar.  Note**Note**  The [SplitContainer](https://msdn.microsoft.com/en-us/library/system.windows.forms.splitcontainer(v=vs.100).aspx) control is designed to replace the [Splitter](https://msdn.microsoft.com/en-us/library/system.windows.forms.splitter(v=vs.100).aspx) control. |
|  | [TableLayoutPanel](https://msdn.microsoft.com/en-us/library/system.windows.forms.tablelayoutpanel(v=vs.100).aspx) control | Represents a panel that dynamically lays out its contents in a grid composed of rows and columns. |
|  | [FlowLayoutPanel](https://msdn.microsoft.com/en-us/library/system.windows.forms.flowlayoutpanel(v=vs.100).aspx) control | Represents a panel that dynamically lays out its contents horizontally or vertically. |
| Audio | [SoundPlayer](https://msdn.microsoft.com/en-us/library/system.media.soundplayer(v=vs.100).aspx)control | Plays sound files in the .wav format. Sounds can be loaded or played asynchronously. |

### [Superseded Controls and Components by Function](javascript:void(0))

|  |  |  |
| --- | --- | --- |
| Function | Superseded control | Recommended replacement |
| Data display | [DataGrid](https://msdn.microsoft.com/en-us/library/system.windows.forms.datagrid(v=vs.100).aspx) | [DataGridView](https://msdn.microsoft.com/en-us/library/system.windows.forms.datagridview(v=vs.100).aspx) |
| Information Display (Read-only controls) | [StatusBar](https://msdn.microsoft.com/en-us/library/system.windows.forms.statusbar(v=vs.100).aspx) | [StatusStrip](https://msdn.microsoft.com/en-us/library/system.windows.forms.statusstrip(v=vs.100).aspx) |
| Menu controls | [ContextMenu](https://msdn.microsoft.com/en-us/library/system.windows.forms.contextmenu(v=vs.100).aspx) | [ContextMenuStrip](https://msdn.microsoft.com/en-us/library/system.windows.forms.contextmenustrip(v=vs.100).aspx) |
|  | [MainMenu](https://msdn.microsoft.com/en-us/library/system.windows.forms.mainmenu(v=vs.100).aspx) | [MenuStrip](https://msdn.microsoft.com/en-us/library/system.windows.forms.menustrip(v=vs.100).aspx) |
| Commands | [ToolBar](https://msdn.microsoft.com/en-us/library/system.windows.forms.toolbar(v=vs.100).aspx) | [ToolStrip](https://msdn.microsoft.com/en-us/library/system.windows.forms.toolstrip(v=vs.100).aspx) |
|  | [StatusBar](https://msdn.microsoft.com/en-us/library/system.windows.forms.statusbar(v=vs.100).aspx) | [StatusStrip](https://msdn.microsoft.com/en-us/library/system.windows.forms.statusstrip(v=vs.100).aspx) |
| Form layout | [Splitter](https://msdn.microsoft.com/en-us/library/system.windows.forms.splitter(v=vs.100).aspx) | [SplitContainer](https://msdn.microsoft.com/en-us/library/system.windows.forms.splitcontainer(v=vs.100).aspx) |

## **Time Boxing**

|  |  |  |
| --- | --- | --- |
| Activity Name | Activity Time | Total Time |
| Login Systems + Setting up Visual Studio Environment | 3 mints + 5 mints | 8 mints |
| Walk through Theory & Tasks | 60 mints | 60 mints |
| Implement Tasks | 80 mints | 80 mints |
| Evaluation Time | 30 mints | 30 mints |
|  | Total Duration | 178 mints |

## **Objectives**

After completing this lab the student should be able to:

* Clearly understand the toolbox and use of GUI Based Applications.

## **Lab Tasks/Practical Work**

1. Write a program to try some basic toolbox elements.

**Lab Manual for Object Oriented Programming**

**Lab No. 14**

# **GUI BASED CALCULATOR**

Objectives

Integration and use of Object Oriented Programming concept in daily life application development.

**LAB # 14**

**Implementation of Interfaces**

## **Introduction**

Information is provided in previous labs.

## **Time Boxing**

|  |  |  |
| --- | --- | --- |
| Activity Name | Activity Time | Total Time |
| Login Systems + Setting up Visual Studio Environment | 3 mints + 5 mints | 8 mints |
| Walk through Theory & Tasks | 60 mints | 60 mints |
| Implement Tasks | 80 mints | 80 mints |
| Evaluation Time | 30 mints | 30 mints |
|  | Total Duration | 178 mints |

## **Objectives**

After completing this lab the student should be able to:

* Integration and use of Object Oriented Programming concept in daily life application development.

## **Lab Tasks/Practical Work**

1. Create an calculator based application which follows OOP guidelines.  
   Use following

Encapsulation, Abstraction, Inheritance, Polymorphism, static members to store results.

**Lab Manual for Object Oriented Programming**

**Lab No. 15**

# **STATIC & DYNAMIC BINDING**

Objectives

To understand concept static & dynamic binding.

**LAB # 15**

**Static & Dynamic Binding**

## **Introduction**

### What is reference and object?

class Human{

....

}

class Boy extends Human{

public static void main( String args[]) {

/\*This statement simply creates an object of class

\*Boy and assigns a reference of Boy to it\*/

Boy obj1 = new Boy();

/\* Since Boy extends Human class. The object creation

\* can be done in this way. Parent class reference

\* can point to a child class object\*/

Human obj2 = new Boy();

}

}

Static and Dynamic Binding in Java

Association of method definition to the method call is known as binding. There are two types of binding: Static binding and dynamic binding. Lets discuss them one by one.

### Static Binding or Early Binding

The binding which can be resolved at compile time by compiler is known as static or early binding. All the static, private and final methods have always been bonded at [**compile-time**](http://beginnersbook.com/2013/04/runtime-compile-time-polymorphism/) . **Why binding of Static, final and private methods is always a static binding?**You would understand it better after reading dynamic binding. Still let me explain this – Compiler knows that all such methods cannot be overridden and will always be accessed by object of local class. Hence compiler doesn’t have any difficulty to determine object of class (local class for sure). That’s the reason binding for such methods is static.

#### Static binding example

class Human{

....

}

class Boy extends Human{

public void walk(){

System.out.println("Boy walks");

}

public static void main( String args[]) {

Boy obj1 = new Boy();

obj1.walk();

}

}

Here we have created an object of Boy class and calling the method walk()of the same class. Since nothing is ambiguous here, compiler would be able to resolve this binding during compile-time, such kind of binding is known as static binding.

### Dynamic Binding or Late Binding

When compiler is not able to resolve the call/binding at compile time, such binding is known as Dynamic or late Binding. Overriding is a perfect example of dynamic binding as in overriding both parent and child classes have same method. Thus while calling the overridden method, the compiler gets confused between parent and child class method(since both the methods have same name).

#### Dynamic binding example

Here Human is a super class and Boy is a child class since Boy extends Human. Both these classes have a same method void walk(). Since we have assigned the parent class reference to the child class object, during call ofwalk() method the compiler would not be able to decide which walk()method to call. It will be confused between the walk() method of Human class and walk() method of Boy class. Such kind of bindings are known as dynamic binding as compiler figures out the object type in runtime.

package beginnersbook.com;

class Human{

public void walk()

{

System.out.println("Human walks");

}

}

class Boy extends Human{

public void walk(){

System.out.println("Boy walks");

}

public static void main( String args[]) {

//Reference is of parent class

Human myobj = new Boy();

myobj.walk();

}

}

#### Output:

Boy walks

## **Time Boxing**

|  |  |  |
| --- | --- | --- |
| Activity Name | Activity Time | Total Time |
| Login Systems + Setting up Visual Studio Environment | 3 mints + 5 mints | 8 mints |
| Walk through Theory & Tasks | 60 mints | 60 mints |
| Implement Tasks | 80 mints | 80 mints |
| Evaluation Time | 30 mints | 30 mints |
|  | Total Duration | 178 mints |

## **Objectives**

After completing this lab the student should be able to:

* Clearly understand the concept of static & late binding

## **Lab Tasks/Practical Work**

1. Write a program which contains a human class inherited by student, teacher, staff classes and apply static and dynamic binding respectively.

# References

Anon., 2016. *Tutorials Point Object Oriented Programming.* [Online]   
Available at: http://www.tutorialspoint.com/cplusplus/cpp\_object\_oriented.htm

Mehra, P., 2016. *C Sharp Corner.* [Online]   
Available at: http://www.c-sharpcorner.com/uploadfile/puranindia/what-are-access-modifiers-in-C-Sharp/  
[Accessed 25 1 2016].

Naimipour, R. N. a. K., n.d. *Foundations of Algorithms Using C++ Pseudocode.* Third ed. s.l.:JONES AND BARTLETT PUBLISHERS.

Point, T., n.d. *Tutorials Point Data Structures and Algorithms.* [Online]   
Available at: http://www.tutorialspoint.com/data\_structures\_algorithms/index.htm

Weiss, M. A., n.d. *Data Structures and Algorithms Analysis in C++.* Fourth ed. s.l.:s.n.