**Object-Oriented Application Development**

**Practical 2**

**Part A**

1. Run the following programs and note the different **parameter passing methods**:

|  |  |
| --- | --- |
| a. | using System;  class ParameterPassing1  {  public static void Main()  {  char letter = 'a';  // pass by reference with keyword ref  // letter must have a value  ParameterPassing1.MethodA(ref letter);  // letter is changed  Console.WriteLine("Letter in Main () is {0}", letter);  }  public static void MethodA(ref char aLetter)  {  // aLetter refers to letter of Main  aLetter = 'z';  }  } |

|  |  |
| --- | --- |
| b. | using System;  class ParameterPassing2  {  public static void Main()  {  char letter;  // pass by reference with keyword out  // letter need not have a value  ParameterPassing2.MethodB(out letter);  // letter has a value  Console.WriteLine("Letter in Main () is {0}", letter);  }  public static void MethodB(out char aLetter)  {  // aLetter refers to letter of Main  aLetter = 'z';  }  } |

|  |  |
| --- | --- |
| c. | using System;  class ParameterPassing3  {  public static void Main()  {  char letter = 'a';  // pass by value  ParameterPassing3.MethodC(letter);  // letter is not changed  Console.WriteLine("Letter in Main () is {0}", letter);  }  public static void MethodC(char aLetter)  {  // aLetter gets a copy of the value from letter in Main  aLetter = 'z';  }  } |

1. Run the following program and see how the concept of **overloaded methods** or **method overloading** is applied in the WriteLine() method.

|  |
| --- |
| using System;  class Program  {  public static void Main()  {  // WriteLine() with string argument  // – method signature WriteLine(string)  Console.WriteLine("First a string");  // WriteLine()with int argument  // – method signature WriteLine(int)  Console.WriteLine(10);  // WriteLine()with double argument  // – method signature WriteLine(double)  Console.WriteLine(32.68);  }  } |

1. The following program consists of two classes. Create two files, one for each class.

Run the program. Note the **instance variable, property, and constructor** in the class definition and how they are used in the Main() method.

|  |
| --- |
| public class Person  {  // instance variable  private string name;  // === start of property  public string Name  {  // get member of property  get  {  return name;  }  // set member of property  set  {  name = value;  }  }  // === end of property  // === start of constructor (with one parameter)  public Person(string theName)  {  // use Name property to initialise instance variable name  Name = theName;  }  // === end of constructor  } |

|  |
| --- |
| using System;  public class PersonTest  {  public static void Main()  {  // object variable  // – reference type (to refer to Person object)  Person p;  // creation of Person object  // and assignment to object variable  p = new Person("Steve");  // use of property Name with get member  Console.WriteLine("Person name is " + p.Name);  // use of property Name with set member  p.Name = "Steve Jobs";  // use of property Name with get member  Console.WriteLine("Person name is now " + p.Name);  }  } |

1. Use the Person class given in Question 3 and each of the following PersonTestX classes containing the Main() method.

*Note:* Select **Compile** for the **Build Action** property of the Person class and one of the PersonTestX classes. For the rest of the PersonTestX classes, make sure **None** is selected for the **Build Action** property.

For each program, determine if there are any errors in the Main() method. If there are errors, determine what the error is. Correct the errors. *Note:* DO NOT make any modifications in the Person class. Only modify the statements in the Main() method.

(a)

|  |
| --- |
| using System;  public class PersonTest1  {  public static void Main()  {  Person p1 = new Person("Steve Jobs");  Console.WriteLine("Person name is " + p1.Name);  }  } |

Valid

Person name is Steve Jobs

(b)

|  |
| --- |
| using System;  public class PersonTest2  {  public static void Main()  {  Person p2 = new Person("Steve Jobs");  Console.WriteLine("Person name is " + p2.name);  }  } |

Invalid because name is

(c)

|  |
| --- |
| using System;  public class PersonTest3  {  public static void Main()  {  Person p3;  p3 = new Person("Steve", "Jobs");  Console.WriteLine("Person name is " + p3.Name);  }  } |

Invalid because the constructor can only read one statement

(d)

|  |
| --- |
| using System;  public class PersonTest4  {  public static void Main()  {  Person p4, p5;  p4 = new Person("Steve Jobs");  p5 = new Person("Bill Gates");  Console.WriteLine("First Person name is " + p4.Name);  Console.WriteLine("Second Person name is " + p5.Name);  }  } |

Valid

(e)

|  |
| --- |
| using System;  public class PersonTest5  {  public static void Main()  {  Person p6, p7;  p6 = new Person("Steve");  p7 = p6;  Console.WriteLine("First Person name is " + p6.Name);  Console.WriteLine("Second Person name is " + p7.Name);  }  } |

Valid

(f)

|  |
| --- |
| using System;  public class PersonTest6  {  public static void Main()  {  Person p8;  string name;  p8 = new Person(name);  Console.Write("Enter person name: ");  name = Console.ReadLine();  //here is the correct sequence  Console.WriteLine("Person name is " + p8.Name);  }  } |

Invalid because of logic error, sequence is wrong.

(g)

|  |
| --- |
| using System;  public class PersonTest7  {  public static void Main()  {  Person p9 = new Person(empty parameter);  Console.WriteLine("First Person name is " + p9.Name);  }  } |

Invalid

1. The following program consists of two classes. For the CourseTest class, complete the code according to the comments given.

|  |
| --- |
| public class Course  {  // instance variable  private string title;  // property with get member only  public string Title  {  get  {  return title;  }  //add a set  }  // constructor without parameters  public Course()  {  title = "Untitled";  }  } |

|  |
| --- |
| using System;  public class CourseTest  {  public static void Main()  {  // declare an object variable named myCourse of type Course  // create a Course object and assign it to variable myCourse  // display the title for the Course object  // add a statement to change the title to c# programming  // display the new title for the course title  }  } |

* Course myCourse;
* myCourse = newCourse( )
* Console.WriteLine(“The title of the course is ” + myCourse.Title);
* set

{

Title = value

}

* myCourse.Title = (“c# programming”);

Console.WriteLine(“The new title is ” + myCourse.Title);

1. Modify the Course class in Question 5 so that the title of a Course object can be changed. [*Hint:* add a *set* member for the Title property].

Add statements in the Main() method in the CourseTest class to change the course title to “C# Programming” and display the title again.

1. Consider the Course and CourseTest classes below.

The code in the two classes contains some errors. Correct the errors and run the program.

|  |
| --- |
| public class Course  {  // instance variable must have private access modifier  private string title;  public string Title  {  get  {  return Title; title  }  set  {  value = Title; //title = value  }  }  // constructor must have one parameter which represents the title  public Course(string aTitle)  {  aTitle = Title; Title = aTitle  }  } |

|  |
| --- |
| using System;  public class CourseTest  {  public static void Main()  {  Course myCourse;  string courseTitle;  //Here is the code line  myCourse = new Course(courseTitle);  Console.Write("Enter course title: ");  courseTitle = Console.ReadLine();  ~~myCourse.title = courseTitle;~~  Console.WriteLine("The course title is " + myCourse.Title);  }  } |

**Part B**

1. Create an Employee class which includes the following:

* Data members or instance variables representing the name (*string*), date of hire (*string*), and salary (*decimal*) of an employee.
* A constructor with 3 parameters that represent the name, date of hire, and salary.
* A property for the salary with get and set members.
* Properties for the name and date of hire with get members only; do not include set members.

1. Create a class named EmployeeTest1 to test the Employee class you created in Question 1. Make sure you test the constructor and the set and get members of the properties. The values for the data members can be hard-coded in the Main( ) method.
2. public class Employee
3. {
4. private string name;
5. public string Name
6. { get { return name; } }
7. private string date;
8. public string Date
9. { get { return date; } }
10. private decimal salary;
11. public decimal Salary
12. { get { return salary; }
13. set { salary = value; }
14. }
15. public Employee(string aName, string aDate, decimal aSalary)
16. {
17. name = aName;
18. date = aDate;
19. salary = aSalary;
20. }
21. }
22. Create another class named EmployeeTest2 to test the Employee class you created in Question 1. The values for the data members must be entered by the user.
23. public class ET2
24. {
25. static void Main()
26. {
27. string iName, iDate, input;
28. decimal iSalary;
29. Console.WriteLine("Please enter your name");
31. iName = Console.ReadLine();
32. Console.WriteLine("“Please enter your date of hire”");
33. iDate = Console.ReadLine();
34. Console.WriteLine("“Please enter your salary”");
35. input = Console.ReadLine();
36. iSalary = Convert.ToDecimal(input);
37. Employee et = new Employee(iName, iDate, iSalary);
38. Console.WriteLine("Name: {0} Date: {1} Salary: {2}", et.Name, et.Date, et.Salary);
39. }
40. }
41. Create a Date class with integer data members or instance variables representing the day, month, and year of a date. Include a constructor with 3 parameters that represent the day, month, and year. Include properties with get members only.
42. public class Date
43. {
44. private int day;
45. public int Day
46. { get { return day; } }
47. private int month;
48. public int Month
49. { get { return month; } }
50. private int year;
51. public int Year
52. { get { return year; }
54. }
55. public Date(int aDay,int aMonth, int aYear )
56. {
57. day = aDay;
58. month = aMonth;
59. year = aYear;
60. }
61. }
62. Create a class named DateTest to test the Date class you created in Question 4. Make sure you test the constructor and the get members of the properties. The values for the data members may be hard-coded in the Main( ) method.
63. Modify the Date class in Question 4 as follows:

* Add set members for the properties. The set members for day and month must ensure that the values are valid i.e. day values must be between 1 to 31 and month values must be between 1 to 12. Use if statements in the set member of the properties. If not valid, set the day and month to 0.
* Modify the constructor to use the set members.

Modify the DateTest class in Question 5 to test modified constructor and set members of the properties. Hard-code the values for day, month, and year and test for valid and invalid values.

public class Date

{

private int day;

public int Day

{

get { return day; }

set

{

if (day >= 31)

{ day = 0; }

else day = value;

}

}

private int month;

public int Month

{

get { return month; }

set

{

if (12 >= month)

{

month = value;

}

else month = 0;

}

}

private int year;

public int Year

{ get { return year; }

set { year = value; }

}

public Date(int aDay,int aMonth, int aYear )

{

day = aDay;

month = aMonth;

year = aYear;

}

}

public class DT

{

static void Main()

{

int t1 = 32, t2 = 12, t3 = 1222;

Date test = new Date(t1,t2,t3);

Console.Write(t1);

}

}