



## COMPSCI 280 S1 2010 Enterprise Software Development

Revision



## Exam

- ▶ Total = 100 marks
- ▶ ALL MCQ
  - ▶ Section 1: Ulrich (40marks)
    - ▶ 21 questions
  - ▶ Section 2: Angela (60 marks)
    - ▶ 38 questions

2



## Section 2

- ▶ Data types, operator, Control flow etc (7 questions)
- ▶ Arrays (2 questions)
- ▶ functions, pass by ref, by value etc (5 questions)
- ▶ Delegates (4 questions)
- ▶ class, struct, inheritance, indexer, operator overloading (10 questions)
- ▶ Excel (2 questions)
- ▶ Basic ado .net (3 questions)
- ▶ A database application (5 questions)

3



## Data types, operator, Control flow etc

- ▶ The Convert Class & Parse Method
- ▶ Formatting
- ▶ Comparison:
  - ▶ For value types, == returns true if the values of its operands are equal
  - ▶ For reference types, == returns true if its two operands refer to the same object
- ▶ Math.Round
  - ▶ x.5 -> Nearest even integer
- ▶ If (same as java)
- ▶ Switch
  - ▶ Value must be an integer or a string
  - ▶ Execution of the statement body begins at the selected statement and proceeds until the break statement transfers control out of the case body
  - ▶ Implicit fall through from one case to another if a case statement has no code.

4



## Data types, operator, Control flow

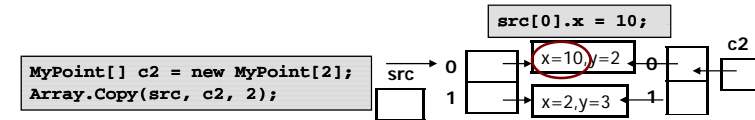
- ▶ Loops
  - ▶ Pre-test loops
    - ▶ while
    - ▶ for
    - ▶ foreach (cover in Arrays)
  - ▶ Post-test loop
    - ▶ do...while
- ▶ break
- ▶ continue
- ▶ goto
- ▶ using

5



## Arrays

- ▶ Arrays
- ▶ Arrays of Objects



- ▶ Multidimensional arrays

- ▶ Navigating elements
  - ▶ Nested loops are used to navigate the array elements

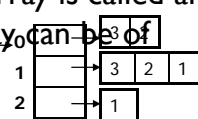
```
for (int i = 0; i < nums2.GetLength(0); i++){
  for (int j = 0; j < nums2.GetLength(1); j++){
    Console.WriteLine(nums2[i, j] + " ");
  }
  Console.WriteLine();
}
```

6



## Jagged Arrays

- ▶ An array of which each element is itself an array is called an array of arrays. The elements of a jagged array can be of different size
- ▶ Length & UpperBound
  - ▶ The jagged array is ONE dimension only.
- ▶ Navigating elements
  - ▶ Note: b2[i].GetLength(0) returns the number of the array element at the array index



```
int[][] b2 = new int[][] { new int[] {3,2}, new int[] {3,2,1}, new int[] {1}};
```

7



## Objects

- ▶ Constructors
- ▶ Fields & Properties
- ▶ Methods
- ▶ Pass by value
- ▶ Pass by ref
- ▶ Structs Vs Classes

8



## Passing mechanism

- ▶ A method may modify the programming element underlying the argument in the calling code. It depends on the following two factors:
  - ▶ The argument is being passed by value or by reference .
  - ▶ The argument data type is a value type or a reference type.
  - ▶ The default is to pass arguments by value.
- ▶ Passing parameters by reference
  - ▶ Change the value of the parameters and have that change persist.
  - ▶ Use the ref or out keyword
    - ▶ Note: ref requires that the variable be **initialized** before being passed.
    - ▶ Note: out arguments need not be initialized prior to being passed but calling method is required to **assign a value before** the method returns.
- ▶ You should choose the passing mechanism carefully for each argument

9

COMPSCI 280

Handout 08



## Passing Value-Type Parameters

- ▶ Case 1: value Type + Pass by value
  - ▶ Passing a copy of the variable to the method
  - ▶ The method cannot modify the variable itself

```
int x1 = 1;
UpdateIntByValue(x1);
Console.WriteLine(x1);
```

1 (unchanged)

```
public static void UpdateIntByValue(int v){
    v *= 2;
}
```

- ▶ Case 2: value Type + Passy by reference (ref/out)
  - ▶ Reference is passed into the method
  - ▶ Both the method definition and the calling method must explicitly use the *ref/out* keyword.
  - ▶ The method can modify the variable itself

```
int x2=1, x3=1;
UpdateIntByRef(ref x2);
UpdateIntByOut(out x3);
```

3 (changed)

x2 must be initialised.

```
public static void UpdateIntByRef(ref int v) {
    v *= 3;
}
```

4 changed

```
public static void UpdateIntByOut(out int v) {
    v = 4;
}
```

required to assign a value before the method returns

10

COMPSCI 280

ut 08



## Passing Reference-Type

- ▶ Case 3: Reference Type + Pass by value
  - ▶ The method cannot change the variable but can change members of the instance to which it points
  - ▶ A) The method can change the **members**

```
int[] y1 = {1, 2, 3};
UpdateRefTypeByValue(y1);
Console.WriteLine(y1[0]);
```

=10

```
public static void UpdateRefTypeByValue(int[] x){
    x[0] = 10;
}
```

- ▶ B) The method cannot change the variable
  - ▶ Example: you cannot assign a new array to the variable

```
int[] y2 = {1, 2, 3};
ReplaceRefTypeByValue(y2);
Console.WriteLine(y2[0]);
```

=1

```
public static void ReplaceRefTypeByValue(int[] x) {
    x = new int[] { 2, 3 };
}
```

Cannot change the entire array

11

COMPSCI 280

Handout 08



## Passing Reference Type

- ▶ Case 4: Reference Type + Pass by Reference
  - ▶ The method can change both the variable and members of the instance to which it points
  - ▶ A) The method can change the members

```
int[] y3 = {1, 2, 3};
UpdateRefTypeByRef(ref y3);
Console.WriteLine(y3[0]);
```

=10

```
public static void UpdateRefTypeByRef(ref int[] x){
    x[0] = 10;
}
```

change the member

- ▶ B) The method can also change the variable
  - ▶ Example: you can assign a new array to the variable

```
int[] y4 = {1, 2, 3};
ReplaceRefTypeByRef(ref y4);
Console.WriteLine(y4[0]);
```

=2

```
public static void ReplaceRefTypeByRef(ref int[] x){
    x = new int[] {2,3};
}
```

change the entire array

12

COMPSCI 2

Handout 08



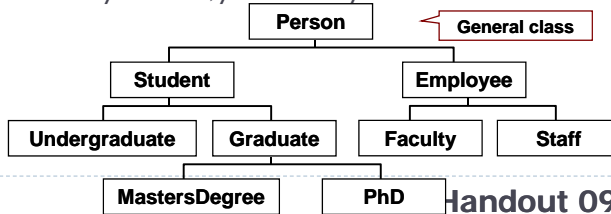
# Inheritance

## Rules:

- Derived classes inherit, and can extend, the properties, methods, and events of the base class.
- Derived classes can also override inherited methods with new implementations.
- All classes created are inheritable by default.

## Characteristics:

- Inheritance allows you to write and debug a class once, and then reuse that code over and over as the basis of new classes.
- Inheritance also allows you to use inheritance-based polymorphism, the ability to define classes that can be used interchangeably by client code at run time, but with functionally different, yet identically named methods or properties.



# new methods

- If the method in the derived class is preceded with the new keyword, the method is defined as being independent of the method in the base class.
  - Using the new keyword tells the compiler that your definition hides the definition contained in the base class.

```

class A {
  ...
  public void method1(){
    Console.WriteLine("A:Method1");
  }
}

class B : A {
  public void method1(){
    Console.WriteLine("B:Method1");
  }
}

class B : A {
  public new void method1(){
    Console.WriteLine("B:Method1");
  }
}
  
```

compiler will issue a warning

A:Method1  
B:Method1



# Virtual methods

- If an inherited property or method needs to behave differently in the derived class it can be overridden; that is, you can define a new implementation of the method in the derived class.
  - The property or method in the base class is marked with the virtual keyword, and
  - Use the Override keyword to shadow the member by redefining it in the derived class.
- Rules:
  - The base method, which is to be overridden, must be declared as virtual, abstract, or override.
  - Both the overridden and the overriding method or property must have the same access-level modifiers
  - The new implementation of a member can call the original implementation in the parent class by specifying by the **base keyword**

```

class Class1 {
  ...
  public virtual void method4(){
  }
}

class Class2 : Class1 {
  ...
  public override void method4() {}
}

Class1 c1 = new Class1();
Class2 c2 = new Class2();
c1.method4();
c2.method4();
  
```

COMPSCI280



# New Vs Virtual

- The virtual keyword is needed to specify that a method should override a method of its base class.
- Attempting to override a non-virtual method will result in a compile-time error unless the "new" keyword is added to the declaration, indicating the method is intentionally hiding the base class's method
- The modifier new hides the method just like override, however there're two major differences in between
  - The original method doesn't have to be virtual
  - If you use new to hide a method and refer to the Child class using a reference of the type of Base class to point to the Child object, then call the method, you'll get the original method not the new one.

```

class A {
  public void method1(){
    Console.WriteLine("A:Method1");
  }
  public virtual void method2() {
  ...
}

class B : A {
  public new void method1(){
    Console.WriteLine("B:Method1");
  }
  public override void method2() {
  ...
}
  
```

B:Method1  
B:Method2

Declare as A (super class)

A:Method1  
B:Method2

COMPSCI280



## Operator Overloading

- ▶ C# allows you to overload operators for use on your own classes.
  - ▶ You use method-like syntax with a return type and parameters,
  - ▶ But the name of the method is the keyword operator together with the operator symbol
  - ▶ Rules
    - ▶ All operators must be public
    - ▶ All operators must be static
    - ▶ A binary operator has two explicit arguments and a unary operator has one explicit argument
      - For binary operators, at least one of the operands must be of the **same type** as its class
      - For unary operators, the operand must be of the same type as the class
    - ▶ Some operators naturally come in **pairs**. You must define them both. Example: ==, !=
    - ▶ Operator parameters must **NOT** use the ref or out modifier

17

COMPSCI280

Handout10



## Value Types

- ▶ Implementing the Equality Operator (==) on Value Types
  - ▶ No default implementation of the equality operator (==) for value types
  - ▶ You should overload the **equality** operator (==) any time equality is meaningful
  - ▶ You should consider overriding the **Equals** method to gain increased performance

```
public override int GetHashCode(){
    return _h.GetHashCode();
}
public override bool Equals(object o){
    if (!(o is Hour)) {
        return false;
    }
    return this == (Hour)o;
}
```

Hour – structure type

no classes will be derived from Hour. Don't need to use GetType. Use the is operator

```
public static bool operator ==(Hour a, Hour b) {
    return a.h == b.h;
}
public static bool operator !=(Hour a, Hour b){
    return a.h != b.h;
}
```

18

COMPSCI280

Handout10



## Reference Types

- ▶ Implementing the Equality Operator (==) on Reference Types
  - ▶ default implementation of the equality operator (==) for reference type => check identity
  - ▶ You should consider overriding the **Equals** method on a reference type so that it compares the contents of two objects for equality

```
public override bool Equals(Object obj){
    if (obj == null || GetType() != obj.GetType())
        return false;
    MyPoint p = (MyPoint)obj;
    return (x == p.x) && (y == p.y);
}
public override int GetHashCode(){
    return x ^ y;
}
```

Check the object obj references an instance of the same type as this object

uses the GetType Method to determine whether the run-time types of the two objects are identical

MyPoint– class type

19

COMPSCI280

Handout10



## Indexers

- ▶ Indexers permit instances of a class or struct to be indexed in the same way as arrays.
  - ▶ Indexers are similar to properties except that their accessors take parameters.
    - ▶ A get accessor returns a value. A set accessor assigns a value.
  - ▶ The **this** keyword is used to define the indexers.
  - ▶ The **value** keyword is used to define the value being assigned by the set indexer.

```
class MyPointArray {
    private MyPoint[] ptArray = new MyPoint[10];
    public MyPoint this[int i] {
        get {
            return ptArray[i];
        }
        set {
            ptArray[i] = value;
        }
    }
}
```

```
MyPointArray ptArr = new MyPointArray();
ptArr[0] = new MyPoint();
ptArr[1] = new MyPoint(10, 20);
Console.WriteLine(ptArr[1].x);
```

20

COMPSCI280

Handout10



# Creating Delegates

- ▶ The type of a delegate is defined by the name of the delegate
 

```
public delegate void Print(string s);
```
- ▶ Each delegate is limited to referencing methods of a particular kind only. The type is indicated by the delegate declaration – the input parameters and return type
- ▶ Example:
  - ▶ Two methods with the same signature as the delegate declared above
 

```
public static void printToLower (string s) {
    Console.WriteLine("static:" + s.ToLower());
}
public void printToUpper (string s) {
    Console.WriteLine("instance:" + s.ToUpper());
}
```

Static method
  - ▶ Creating a New Delegate Object
    - ▶ Use the new operator
    - ▶ The argument is the method call, but without the arguments to the method
    - ▶ Delegate objects are immutable. (can't change)
    - ▶ A delegate can reference static or instance method

```
Print v1 = new Print(printToLower);
Print v2 = new Print(new Program().printToUpper );
```



# Delegates

- ▶ Using Arrays
- ▶ Named Method
- ▶ Anonymous Method
- ▶ Multicast delegates

```
Print v1 = printToLower;
v1("This is another test");
Print v2 = new Program(). printToUpper;
v2("This is another test");
```

```
Print a1 = delegate(string j){
    Console.WriteLine("Anonymous:" + j);
};
a1("Well done");
```

```
public delegate void Print(string s);
```

```
Print mp2 = null;
mp2 += printToLower;
mp2 += new Program().printToUpper;
mp2("Hello World");
```

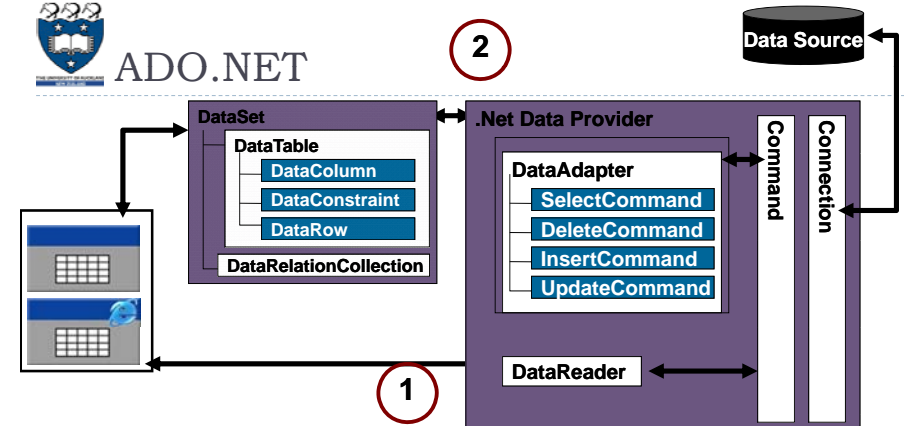


# Excel

- ▶ Create new workbook
- ▶ Refer to worksheet sheet1 / new worksheet
- ▶ Work on range
  - ▶ Set value
  - ▶ Set values by an array
  - ▶ Formula
  - ▶ AutoFill



# ADO.NET



- ▶ Connect to data source
  - ▶ Command
    - ▶ MUST open a connection
    - ▶ Execute a query
  - ▶ DataAdapter
    - ▶ Fill method
    - ▶ Update method
- ◆ Topics:
  - Connections
  - Data Commands
  - Data Adapters/Table Adapters
  - DataSets
  - Data Tables, DataRows, DataColumnns
  - DataViews
  - Data Binding
  - DataRelations

## Connection, Command, Parameter & DataReader

### Connection

- Property
  - ConnectionString **Lab23**
- Read the ConnectionString
  - Application configuration File, or
  - UDL file, or
  - Path: Application.StartupPath

### Command

- Constructor:
  - (sqlString, conn)
- Methods:
  - ExecuteScalar
  - ExecuteNonQuery
  - ExecuteReader
- Note:
  - Must open the connection first
  - Close the connection when finish

### Parameter

- SQL: "Select \* from Customers where CustomerID=?"
- Constructor:
  - (NameString, OleDbType, ExpressionString)
- Property: Value

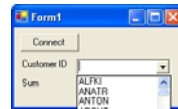
25

### OleDbType:

- VarWChar
- Integer
- Single...

### DataReader

- Creating by
  - cmd.ExecuteReader
- Methods
  - Read() **Lab24**
  - GetXXX
    - GetString(0)
    - GetInt32(1)
  - Item("ColumnName")

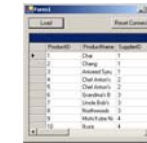


## TableAdapter & DataSet

### TableAdapter

- Manages Data exchange between a DataSet and a data source
  - Fill
  - Update
- Can Contain Multiple queries
- Built-in Connection
- Creating
  - Design Time: Data Source window and drag the TableAdapter onto the form designer area, or
  - new NVDDataSetTableAdapters.CustomersTableAdapter()

**Lab25**



### DataSet

- Local in-memory cache of relational Data (Disconnected data)
- Contains
  - DataTables
    - DataColumns
    - DataRows
  - DataRelations
  - Constraints
- Typed/ Untyped dataset
  - New DataSet
  - New NotyWindDataSet
- Binding DataSet to controls
  - Simple Binding
  - Complex Binding
- Filling dataset
  - Fill method of DataAdapter/TableAdapter
- Updating DataSet
  - Update method of TableAdapter

26

## DataTable, DataColumn & DataRow

### DataTable

- Properties
  - TableName
  - Columns Collection
  - Rows Collection
- Methods
  - Clear
  - Select(filterstring) As DataRow()
  - Copy
  - Clone
  - NewRow
- Events
  - ColumnChanging/ed
  - RowChanging/ed
  - RowDeleting/ed

### DataColumn

- Constructor
  - (ColumnName, Type, Expression)
- Data Type:
  - Type.GetType("System.String")
  - Type.GetType("System.Int32")

27

### DataColumn

- Expression-based column – read only
  - Calculation
  - Functions:
    - "Len(columnName)"
    - "Convert(expr, 'System.DateTime)'"

### DataRow

- Represents Data in a DataTable
- Iterate all rows using for each ...next
- Properties
  - Item
  - RowError
  - HasError
  - RowState
    - Modified ,Added, Deleted, unchanged, detached
  - RowVersion
    - Original, current, Proposed

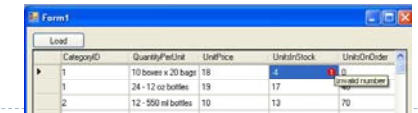
## DataRow

### DataRow

- Methods
  - HasVersion
  - SetColumnError(columnName, Error)
  - IsNull(DataColumn)
- Data Manipulation
  - Edit
    - BeginEdit
    - EndEdit
    - CancelEdit
  - Add
    - DS.DataTable.NewRow
    - ...
    - DS.DataTable.Rows.Add(drNewRow)
  - Delete
    - Dr.Delete()

### Validation

- ColumnChanging **Lab26**
  - Check args.Column.ColumnName
  - Check arg.ProposedValue
  - If invalid
    - Change the arg.ProposedValue
    - Restore its current version
    - SetColumnError



28

