Inheritance

Relationships

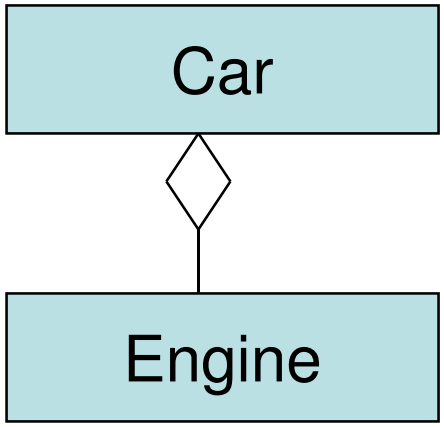
Two types of relationship are:

- Containment/Delegation
  - “HAS-A” relationship
- Inheritance
  - “IS-A” relationship

HCL

Containment / Delegation

- When the relationship is of type “HAS-A”, one object contains another object.
- For example:



Car HAS-A Engine

HCL

```
 public class Engine{
 private string engine_type;
 private int  engine_cc;
//other attributes….
 public EngineType{
      get{ return engine_type}
      set{ engine_type=value;}
  }
 public string GetStatus() {return "good";}
// get and set properties for other attributes
}
public class car{
 private Engine engine =new Engine();
 string model;
//other attributes …
// get and set properties for attributes
}
```

HCL

Delegation model

- Delegation is adding members to the containing class that make use of the contained object's functionality.
- For example: In case of car-engine, we can expose the functionality of contained engine class by adding a member in the containing class car.

HCL

```
public class Engine{
 private string engine_type;
 private int  engine_cc;
//other attributes….
 public EngineType{
     get{ return engine_type}
     set{ engine_type = value;}     }
 public string GetStatus() {return
  “good”;}
// get and set properties for other
  attributes
}
```

HCL

```
public class car{
 private Engine engine = new Engine();
 string model;
//other attributes …
// get and set properties for attributes

 public string GetEngineStatus(){
  return engine.GetStatus(); }

  public Engine EngineDetails {
      get{ return engine;}
      set{ engine = value;}
}
}
```

Exposing functionality

Adding custom property

HCL

Defining inheritance

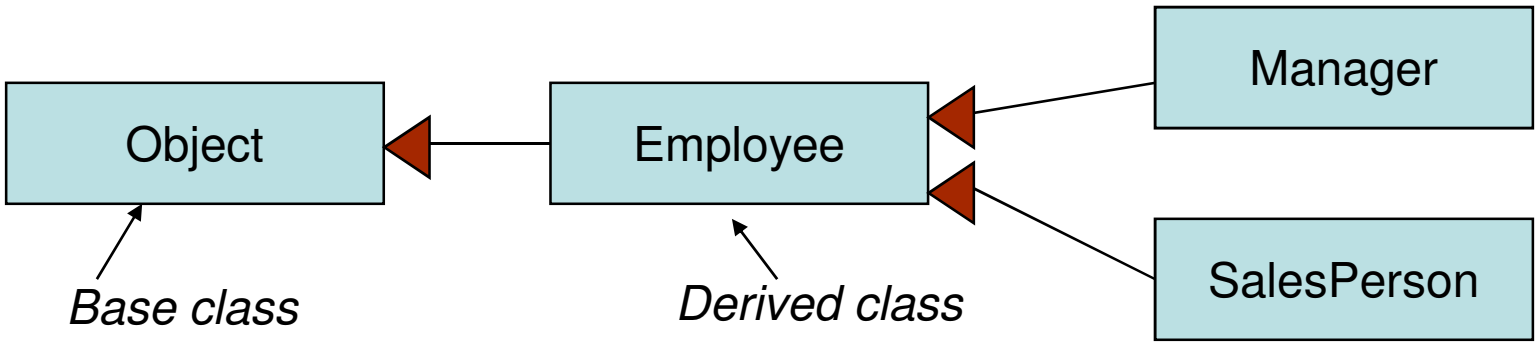
- Inheritance is an object oriented concept that allows hierarchical classifications. Using this concept, a general class can be created that defines traits common to a set of related items. This class can then be inherited by more specific classes with each adding those things that are unique to it.

- The class that is inherited is called super class or base class and the class that is inheriting is called a subclass or derived class

HCL

Understanding Inheritance

- When the relationship is of type “IS-A”, there is a dependency between types.
- The new classes may heave the functionality of other classes.
- For example, SalesPerson,Manager “IS-A” type of Employee
- By default all the C# classes automatically inherit from System.Object class.



HCL

Example- Inheritance

```
using System;
public class Employee{
public Employee()  {
Console.WriteLine("Employee Constructor.");
}
public void print()  {
Console.WriteLine("employee print");
  }

~Employee(){
Console.WriteLine("Employee destroyed");
}
}
```

HCL

```
public class Manager : Employee{
 public Manager() {
Console.WriteLine("Manager Constructor.");
}

~Manager(){
Console.WriteLine("Manager  destroyed");
}

public static void Main()    {
    Manager  child = new Manager();
    child.print();
   }

}
```

Using colon(:) one class inherits from another class

Public Employee methods automatically inherited in the Manager class

HCL

Order of constructor and destructor invocation

- Result of execution of the employee code-

```
Employee Constructor

Manager Constructor

employee print

Manager  destroyed

Employee destroyed
```

HCL

Calling base class constructor

- Base class default constructor is automatically invoked when the derived class instance is created.
- If base class does not have a default constructor then the compiler tags an error.
- In cases like this, the derived class constructor must explicitly invoke any of the base class parameterized constructor.
- Syntax: ***DerivedClass*:base(*parameter-list*)**

HCL

```
using System;

public class Employee
{
int id;
string name;
    public Employee(int id, string nm)
    {
        this.id=id;
        name=nm;
         Console.WriteLine("Employee Constructor.");
    }

    public void print()
    {
         Console.WriteLine("ID:"+ id+" Name :"+name);
    }
}
```

HCL

```
public class Manager : Employee
{
    public Manager(int id, string nm):base(id,nm)
    {

      Console.WriteLine("Manager Constructor.");
    }

    public static void Main()
    {
        Manager  child = new Manager(1,"ABC");

        child.print();
    }
}
```

HCL

Protected members

- The protected members of the base class can be accessed only by the base class members as well as the derived class members.

```
public class Employee{
…
protected void print() {
Console.WriteLine("ID:"+ id+" Name :"+name);
}}
public class Manager : Employee{…
print();}
```

Some other class outside the hierarchy

```
Manager child = new Manager (1,"ABC");
child.print();→ error
```

HCL

Preventing Inheritance

- If we want to ensure that no other class inherits from a class , then we need to seal the class using "sealed" keyword.
-
```
public class Employee{
//attributes and methods or properties
}
 public sealed class Manager : Employee{
  //attributes and methods or properties
  //specific to Manager
}
 public class SalesPerson : Manager{
  //attributes and methods or properties
  //specific to SalesPerson
}
```

Sealed class cannot be inherited by other class.

- `System.String` class is a sealed class.

HCL

Defining the same method in the derived class

```
public class Teacher{
 private string name;
public Teacher(string name){this.name=name;}
public void display(){
   System.Console.WriteLine("Teacher's Name:{0}",name);
}
protected string getName(){return name;}
}
public class HOD: Teacher{
   public HOD(string nm):base(nm) {}
   public void display(){
   System.Console.WriteLine("HOD's Name:{0}",getName());
}

public static void Main(){
HOD h= new HOD("a");
h.display();}.
}
```

Calls the display in the HOD class

Compiler warning generated

```
warning CS0108: 'HOD.display()'
hides inherited member
'Teacher.display()'.
Use the new keyword if hiding
was intended
```

HCL

Hiding method with new keyword

- As suggested by the compiler, on adding new keyword to the method warning disappears.

```
public new void display(){
  System.Console.WriteLine("HOD's
  Name:{0}",getName());
}
```

HCL

Casting references

- A derived class object can be automatically converted into base class object→ implicit conversion.
- Vice versa requires explicit casting
- Examples

```
object o= new Teacher("c");
Teacher t= new HOD("b");
HOD h1=(HOD) t;
```

- C# also allows custom casting→ class defining its own casting.

HCL

Custom casting

- Overloading the cast operator allows the custom casting.
- The keyword implicit or explicit indicate whether casting have to be implicit or explicit.
- The explicit must be specified if there are any values in the class attributes for which the cast will fail or if there is any risk of an exception being thrown.
- Syntax:

```
public static [implicit , explicit]
operator return-type (parameter)
```

HCL

```
using System;
class Money{
uint rupees;
uint paise;
public Money(uint rupees,uint paise){
this.rupees= rupees;
this.paise= paise;          }

public static implicit operator
double (Money m){
return m.rupees +(m.paise/100.0);
}
public void display(){
Console.WriteLine("Rs.
{0}.{1}",rupees,paise);}
```

HCL

```
public static explicit operator Money
(double d){
uint r=(uint)d;
uint p=(uint)((d-r)*100);
return new Money(r,p);          }

public static void Main(){
Money m=new Money(25,50);
double m1=m;
Console.WriteLine("double value "+m1);
double d= 30.75;
m=(Money)d;
m.display();
}}
```

HCL

Arriving at polymorphism

```
public class Teacher{
 private string name;
public Teacher(string name){this.name=name;}
public void display(){
      System.Console.WriteLine("Teacher's
Name:{0}",name);
}
protected string getName(){return name;}
}
public class HOD: Teacher{
      public HOD(string nm):base(nm) {}
      public new void display(){
      System.Console.WriteLine("HOD's
Name:{0}",getName());
}
public static void Main(){
Teacher h= new HOD("a");
h.display();}}
```

The same Teacher-HOD class that we created for hiding method example. Only change is this.

But this calls the display of the Teacher class!

Polymorphism

- By default the method call is resolved at the compile time.
- Compiler just calls the method based on the reference type and not on the actual object type the reference is pointing to.
- This is the reason why the Teacher's display() method is called and not HOD's.
- To defer the actual call mapping until the run-time is polymorphism.
- **In practice, polymorphism refers to an object's ability to use a single method name to invoke one of different methods at run time – depending on where it is in the inheritance hierarchy.**

HCL

Polymorphism: virtual and override keywords

If a base class wishes to define a method that may be redefined by a subclass, it must specify the method as virtual.

When a subclass wishes to redefine a virtual method, it has to use override keyword.

Example

```
public class Teacher{
 private string name;
 public Teacher(){}
 public Teacher(string name){ this.name=name; }
public virtual void display() {
   System.Console.WriteLine("Teacher's
   Name:{0}",name);
}
protected string getName(){return name;}}
```

HCL

```
public class HOD: Teacher{
   public HOD(){}
   public HOD(string nm):base(nm) {}
   public override void display(){
   System.Console.WriteLine("HOD's
   Name:{0}",getName());
}
public static void Main(){
Teacher h= new HOD("a");
h.display();
}
}
```

Displays→
HOD's Name:a

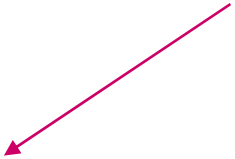
HCL

“sealed” methods

sealed keyword can be applied to methods to make sure that these methods are not overridden by any class.

```
public class Teacher{
public sealed void display();
 //. . . . .
 }

public class HOD:Teacher{
public override void display();{ }
}
```

Error!!! Cannot override a sealed method



HCL

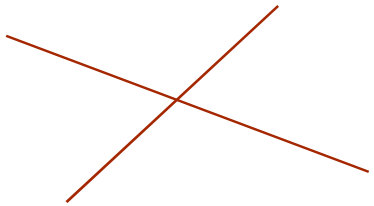
Abstract Classes

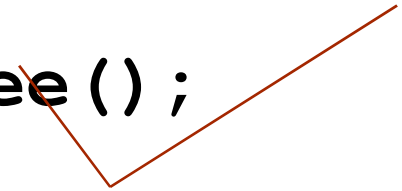
- Abstract classes are those classes which cannot be instantiated (instances cannot be created).
- They are used as base class containing common fields and functionality for the subclasses.
- An abstract class is created by using “abstract” keyword
- Abstract class can contain
  - concrete methods (methods with definition)
  - abstract methods (methods declaration only, no definition-body)

HCL

Example

```
public abstract class Employee{

public abstract void bonus();

public void display(){

…

}}

Employee e= new Employee();


Employee e= new HOD();
```





“Type of” object

- “**`is`**” keyword is used to check whether the base class reference is what derived type.

- “**`as`**” keyword is used to obtain a reference to the more derived type. If the types are incompatible then the reference is set to “null”.

HCL

```
public class Employee{
 protected string empname;
  public void display()
  {
  System.Console.WriteLine("Name:{0}",empname);
  }
}

 public class Manager : Employee{
 public   Manager(string name){
 empname=name;}
}

 public class SalesPerson : Employee{
  public SalesPerson(string name){
  empname=name;}
  public string GetName(){ return empname; }
}
```

HCL

```
public class Test
{
    public static void Main(){
        Employee e=new Manager("Raj");
        e=new SalesPerson("Rajesh");
        if(e is Manager){
        System.Console.WriteLine("Manager");
        }
        else {
        System.Console.WriteLine("SalesPerson");
    //System.Console.WriteLine("Name:{0}",e.GetName());
        SalesPerson sp=e as SalesPerson;
        if(sp!=null)
        System.Console.WriteLine("Name:
        {0}",sp.GetName());
        }
        }}
```

Error – reference of super class type pointing to object of sub class cannot access method of sub class.

OK

Using "as" reference pointing to sub class subobject (SalesPerson) now

HCL

System.Object

- In .NET every type is derived from a common type which is **`System.Object`**.
- Members
  - **`virtual Boolean Equals(Object obj)`**
    - returns true if the references being compared refer to the exact same item in memory.
    - Usually overriden
    - If overriden **`GetHashCode()`** must also be overridden

HCL

- **`virtual Int32 GetHashCode(Object obj)`**
  - returns an integer that identifies a specific object in memory
  - Usually overriden for the class whose instances are to be stored in certain types of collection
- **`virtual String ToString()`**
  - returns a string containing fully qualified name of the class of the object on which it is called
  - When the object is printed using **`Console.WriteLine`** method this method is automatically called.
  - Usually overriden to represent the internal state of the object

HCL

- **`Type GetType()`**
  - returns **`System.Type`** object that fully describes the details of the current object.
- **`virtual void Finalize()`**
  - protected method
  - Called when object is removed from the heap
  - Cannot be overriden in C#
  - The destructor implicitly calls the this method on the object's base class.
- **`Object MemberwiseClone()`**
  - protected method
  - Returns a new object that is member-wise copy of the current object
  - Does a shallow copy

HCL

Overriding some Object methods

```
using System;
class Rect{
private int width,height;

public Rect(int x,int y){
width=x;
height=y;
}

public override string ToString(){
return "width="+width+" height="+height;
}
public override int GetHashCode(){
return ToString().GetHashCode();
}
```

HCL

```
public override bool Equals(object o){
if(o !=null && o is Rect){
Rect b=(Rect)o;
if(b.width==this.width &&
b.height==this.height) return true;
}
return false;}

static void Main(){
Rect b1= new Rect(10,20);
Rect b2= new Rect(10,20);
if(b1.Equals(b2))
Console.WriteLine("b1("+ b1 + ") and b2(" + b2
+ ") are of same size ");
else
Console.WriteLine("b1("+ b1 + ") and b2(" + b2
+ ") are not of same size ");
}}
```

b.cs

Result:

```
b1(width=10
height=20) and
b2(width=10
height=20) are of
same size
```

HCL

Static methods of Object class

- `static bool Equals(object a,object b)`
- `static bool ReferenceEquals(object a, object b)`

- For the previous slides example→
- `object.Equals(b1,b2))` returns `true`
- `object.ReferenceEquals(b1,b2)` returns `false.`

HCL

Partial Types

- C# 2005 introduces a new concept that allows us to create classes (or types) across multiple .cs files.
- This is achieved using partial modifier.

t2.cs

```
namespace PT{
public partial class
Test{
public void f1(){
System.Console.WriteLin
e("f1() called");
}
}}
```

t1.cs

```
namespace PT{
public partial class
Test{
public void f2(){
System.Console.WriteLi
ne("f2() called");
}
static void Main(){
Test t= new Test();
t.f1();
t.f2();}}}
```

Compile command

```
csc t1.cs t2.cs
```

HCL