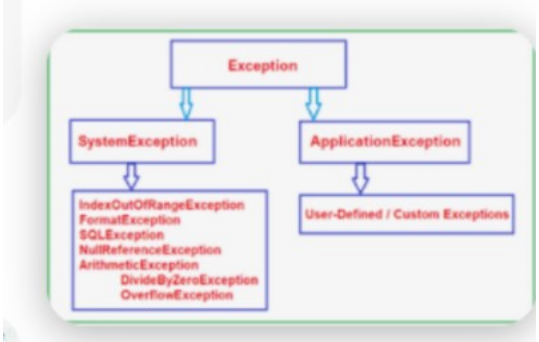
Exception Handling

Saturday, February 1, 2025 11:00 AM

Types of Errors we have in C#

1. Compilation Errors -- > Syntax errors ("" , Data type/ Keywords , ; ,spelling mistakes , trying to create an obj of interface or abstract class)
2. Runtime Errors (Logical errors , invalid input , some of the required resources ,) it's very useful to handle

In bank : transferring some amount from source account to destination account
Exception Handling : try / catch , throw
Here compile time are the errors only not the exceptions
In BCL
IndexOutOfRangeException
FormatException
NullReferenceException
FileNotFoundException
DivideByZeroException
IOException



The CLR creates the exception class object that is associated with that logical mistake (exception) and terminates the program by throwing that exception object by using throw keyword .

Abstract class is just like a concrete class only , having variables , methods

Interface : -- It's like a contract -- It is used to achieve pure abstraction(100%) (all methods are by default abstract methods means there is no implementation) , to achieve multiple inheritance (multiple classes can inherit multiple interfaces)
Multiple implementation (different classes can implement the same interface in different ways).

Can we define an attributes ..

Bank
Current account - > deposit , withdraw

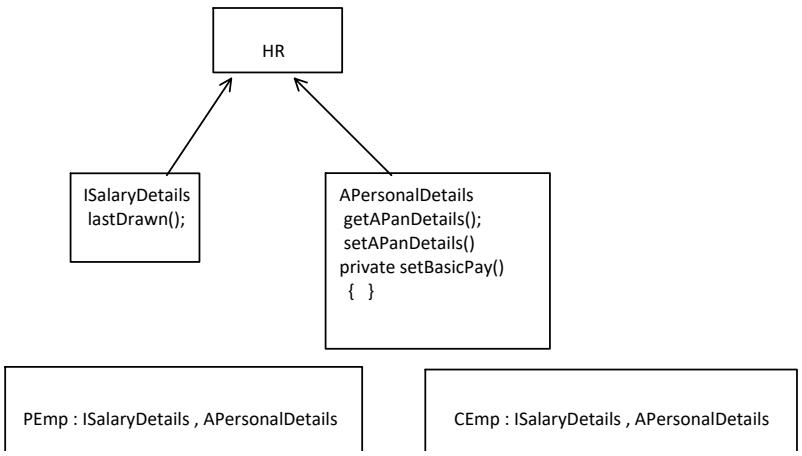
Saving account - > deposit , withdraw

HR is an interface -- > lastDrawn() , abstractclass -- abstract --> PanDetails() and non abstract methods-- Basicpay()

PermanentEmployee class
ContractualEmployee class

Swiggy --- Interface --- API --- price() , products() -- 100%

Haldiram : API PizzaHut



CREATING
RETREIVING
UPDATING
DELETING

SEARCHING & SORTING

Important in terms of DATA STRUCTURE:

Search --> on the basis of index no's
Sorting --- > ascending 2,4,6,8,10

Has to be optimized -- performance

Linear // Binary Search
Bubble sort , selection sort , insertion , merge sort and quick sort

Small list -- Linear search

Linear	2	3	1	10	50	60	70		

-- >
o(n) -- worst efficiency
o(1) -- Best efficiency

Binary Search -- needs elements in a sorted manner -- use case (large list)

Best case -- O(1) --

Worst case -- O(log n)

Bubble Sort --> compare adjacent element

Time complexity -- O(n2)

Selection sort -- first we need to find the smallest element

Time Complexity -- O(n2)

Insertion Sort --- nearly sorted

1 2 3