# School Management System

Create a scenario that models a **School System** using ***interfaces*** and ***abstraction***.

Guidelines:

* Create an interface defining the basic functionalities of a **school member** – should contains **Name** and **Age** properties(only getters) and **Display()** method.
* Then create an abstract class **implementing** common functionality for **school member**s. Add a ***constructor*** that receives ***name*** and ***age*** as parameters.
* Create two classes – **Student** and **Teacher** that represent a specific school member and implement abstraction already defined. Add **GradeLevel** property(only getter) for *Student* and **Subject** property for *Teacher*.
* For School System testing create a class **School** that contains a collection of school **members**(List<?>) with possibility to add member by **AddMember(***{member}***)** and displays information about each member in the collection by **DisplaySchoolMembers()** method.

At the end code bellow should work properly:

class Program

{

static void Main(string[] args)

{

// Creating school members: student and teacher

Student student = new Student("Alice", 15, 10);

Teacher teacher = new Teacher("Mr. Smith", 40, "Mathematics");

// Creating a school and adding members to it

School mySchool = new School();

mySchool.AddMember(student);

mySchool.AddMember(teacher);

// Displaying school members

mySchool.DisplaySchoolMembers();

Console.ReadKey();

}

}

# Library Management System

Create a scenario that models a **Library System** using ***interfaces*** and ***abstraction***.

Guidelines:

* Create an interface defining the basic functionalities of a **library item** – should contains **Title**, **Author**,and **Pages** properties(only getters) and **Display()** method.
* Then create an abstract class **implementing** common functionality for **library item**s. Add a ***constructor*** that receives ***title****,* ***author***, and ***pages*** as parameters.
* Create two classes – **Book** and **Magazine** that represent a specific library item and implement abstraction already defined.
* For Library System testing create a class **Library** that contains a collection of library **items**(List<?>) with possibility to add item by **AddItem(***{item}***)** and displays information about each item in the collection by **DisplayLibraryItems()** method.

At the end code bellow should work properly:

class Program

{

static void Main(string[] args)

{

// Creating library items: book and magazine

Book myBook = new Book("The Hobbit", "J.R.R. Tolkien", 300);

Magazine myMagazine = new Magazine("National Geographic", "Various", 100);

// Creating a library and adding items to it

Library myLibrary = new Library();

myLibrary.AddItem(myBook);

myLibrary.AddItem(myMagazine);

// Displaying library items

myLibrary.DisplayLibraryItems();

Console.ReadKey();

}

}

# Music Library System

Create a scenario that models a **Music Library** using ***interfaces*** and ***abstraction***.

Guidelines:

* Create an interface defining the basic functionalities of a **music item** – should contains **Title**, **Artist**, and **Year** properties(only getters) and **Play()** method.
* Then create an abstract class **implementing** common functionality for **music item**s. Add a ***constructor*** that receives ***title****,* ***artist****,* and ***years*** as parameters.
* Create two classes – **Song** and **Soundtrack** that represent a specific music item and implement abstraction already defined. Add **HasCover** property(only getter) for class *Song*.
* For Music System testing create a class **Album** that contains a collection of music **items**(List<?>) with possibility to add item by **AddMusicItem(***{item}***)** and plays all music items in the collection by **Play()** method.

At the end code bellow should work properly:

class Program

{

static void Main(string[] args)

{

// Creating music items: song and soundtrack

Song song = new Song("Live Is Life", "Opus", 1991, true);

Soundtrack soundtrack = new Soundtrack("Titanic", "James Horner", 1997);

// Creating an album and adding music items to it

Album myAlbum = new Album();

myAlbum.AddMusicItem(song);

myAlbum.AddMusicItem (soundtrack);

// Play items in the album

myAlbum.Play();

Console.ReadKey();

}

}

# Content Management System

Create a scenario that models a **Content System** using ***interfaces*** and ***abstraction***.

Guidelines:

* Create an interface defining the basic functionalities of a **content item** – should contains **Title** and **Display()** method.
* Then create an abstract class **implementing** common functionality for **content item**s. Add a ***constructor*** that receives ***title*** as parameter.
* Create two classes – **Article** and **Video** that represent a specific content item and implement abstraction already defined. Add **Author** property(only getter) for class *Article* and **DurationInMinutes** for class *Video*.
* For Content System testing create a class **ContentManager** that contains a collection of content **items**(List<?>) with possibility to add item by **AddContent(***{item}***)** and displays information about each item in the collection by **DisplayContent()** method.

At the end code bellow should work properly:

class Program

{

static void Main(string[] args)

{

// Creating content items: article and video

Article article = new Article("Understanding Interfaces", "John Doe");

Video video = new Video("Introduction to C#", 15);

// Creating a content manager and adding content items to it

ContentManager contentManager = new ContentManager();

contentManager.AddContent(article);

contentManager.AddContent(video);

// Displaying content items

contentManager.DisplayContent();

Console.ReadKey();

}

}

# Game Development Framework

Create a scenario that models a **Game Framework** using ***interfaces*** and ***abstraction***.

Guidelines:

* Create an interface defining the basic functionalities of a **game entity** – should contains property **Name** and **Update()** method.
* Then create an abstract class **implementing** common functionality for **game entity**. Add a ***constructor*** that receives ***name*** as parameter.
* Create two classes – **Player** and **Enemy** that represent a specific game entity and implement abstraction already defined. For class *Player* add **Health** property(getter and private setter) and **TakeDamage(**{int damage}**)** – reduced player health with given *damage*. For class *Enemy* add **AttackPower** property(only getter) and **Attack(**{player}**)** method that caused damage to the player(call player.TakeDamage(attackPower)).
* For Game Framework testing create a class **GameWorld** that contains a collection of game **entities**(List<?>) with possibility to add entity by **AddEntity(***{entity}***)** and updates each entity in the collection by **UpdateAllEntities()** method.

At the end code bellow should work properly:

class Program

{

static void Main(string[] args)

{

// Creating game entities: player and enemy

Player player = new Player("Player1");

Enemy enemy = new Enemy("Enemy1", 10);

// Creating a game world and adding entities to it

GameWorld gameWorld = new GameWorld();

gameWorld.AddEntity(player);

gameWorld.AddEntity(enemy);

// Updating all entities in the game world

gameWorld.UpdateAllEntities();

Console.ReadKey();

}

}