NAMA : SOFYAN ALI

TANGGAL TEST : 22-MARET-2024

JAWABAN EXAM :

1. **A. Logical and Review Code**

**1).** return application?.protected?.shieldLastRun;

**2).**

public (string Path, string Name) GetApplicationInfo()

{

return ("C:/apps/", "Shield.exe");

}

**3).**

class Laptop

{

public string Os{ get; } // cannot be modified

public Laptop(string os)

{

OS = os;

}

}

var laptop = new Laptop("macOs");

Console.WriteLine(Laptop.Os)

**4).**

while (true)

{

// populate list with 1000 integers

for (int i = 0; i < 1000; i++)

{

myList.Add(new Product(Guid.NewGuid().ToString(), i));

}

// do something with the list object

Console.WriteLine(myList.Count);

// clear the list and set its reference to null

myList.Clear();

}

**5).**

class EventSubscriber : IDisposable   
{   
 private readonly EventPublisher \_publisher;   
 private bool \_disposed = false;   
 public EventSubscriber(EventPublisher publisher)   
 {   
 \_publisher = publisher;   
 \_publisher.MyEvent += OnMyEvent;   
 }   
  
 private void OnMyEvent(object sender, EventArgs e)   
 {   
 Console.WriteLine("MyEvent raised");   
 }   
 protected virtual void Dispose(bool disposing)   
 {   
 if (\_disposed)   
 return;   
  
 if (disposing)   
 {   
 \_publisher.MyEvent -= OnMyEvent;   
 }   
  
 \_disposed = true;   
 }   
 public void Dispose()   
 {   
 Dispose(true);   
 GC.SuppressFinalize(this);   
 }   
}

Now modify the while loop in the Main method to dispose of the subscriber object:

while (true)   
{   
 var subscriber = new EventSubscriber(publisher);   
 // do something with the publisher and subscriber objects   
 subscriber.Dispose();   
}

6).

class TreeNode   
{   
 private readonly List<TreeNode> \_children = new List<TreeNode>();   
 public IReadOnlyList<TreeNode> Children => \_children;   
 public void AddChild(TreeNode child)   
 {   
 \_children.Add(child);   
 }   
 public void RemoveChildAt(int index)   
 {   
 \_children.RemoveAt(index);   
 }   
}

Now modify the while loop within the Main method to remove the old subtrees after you’re done using them:

while (true)   
{   
 // create a new subtree of 10000 nodes   
 var newNode = new TreeNode();   
 for (int i = 0; i < 10000; i++)   
 {   
 var childNode = new TreeNode();   
 newNode.AddChild(childNode);   
 }   
 rootNode.AddChild(newNode);   
 // remove the old subtrees to free up memory   
 if (rootNode.Children.Count > 10)   
 {   
 rootNode.RemoveChildAt(0);   
 }   
}

**7).**

Add a CachedItem class that contains the cached object and its expiration time.

 private sealed class CachedItem   
{   
 public object Value { get; set; }   
 public DateTime Expiration { get; set; }   
}

 Inside the Cache class, instead of using a Dictionary<int, object> to store the cached items, we need to use a Dictionary<int, CachedItem> type to store the cached items with additional information such as expiration time.

 private static readonly Dictionary<int, CachedItem> \_cache = new Dictionary<int, CachedItem>();

 Modify the Add method to store the cached item as a CachedItem with a specified lifespan.

 public static void Add(int key, object value, TimeSpan lifespan)   
{   
 \_cache.Add(key, new CachedItem { Value = value, Expiration = DateTime.Now + lifespan });   
}

 Modify the Get method to check the expiration time of the cached item and remove it from the cache if it has expired.

 public static object? Get(int key)   
{   
 if (!\_cache.ContainsKey(key))   
 {   
 return null;   
 }   
  
 CachedItem item = \_cache[key];   
  
 if (item.Expiration < DateTime.Now)   
 {   
 \_cache.Remove(key);   
 return null;   
 }   
 return item.Value;   
}

 Inside the Program class, modify the call to the Cache.Add method to pass the expiration argument:

Cache.Add(i, new object(), TimeSpan.FromMinutes(10));