Practical no.6: Working with URL routing and dependency injections.

Q1) **How does the ConstructorInjection class demonstrate dependency injection, and what would be the impact of changing the implementation of the IText interface?**

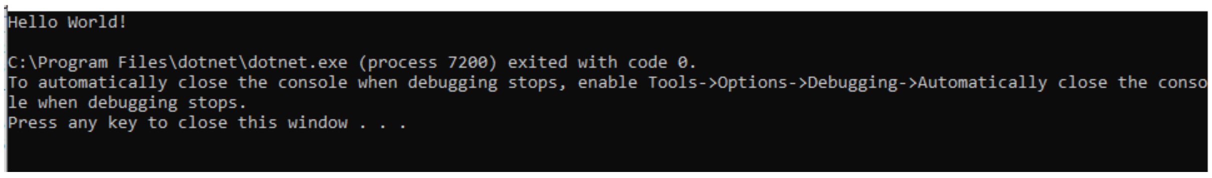
Steps:

Step 1) Open visual studio 2019 code -> create new project ->choose->**Console App (.Net Core)**

code:

```
using System;

namespace PropertyInjection
{
    public interface IText
    {
        void Print();
    }

    class Format : IText
    {
        public void Print()
        {
            Console.WriteLine("Hello World!");
        }
    }

    public class ConstructorInjection
    {
        private IText _text;

        public ConstructorInjection(IText text)
        {
            _text = text;
        }

        public void Print()
        {
            _text.Print();
        }
    }

    class Program
    {
        static void Main(string[] args)
        {
            ConstructorInjection cs = new ConstructorInjection(new Format());
            cs.Print();
            Console.ReadKey();
        }
    }
}
```

Output:



```
Hello World!

C:\Program Files\dotnet\dotnet.exe (process 7200) exited with code 0.
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the conso
le when debugging stops.
Press any key to close this window . . .
```

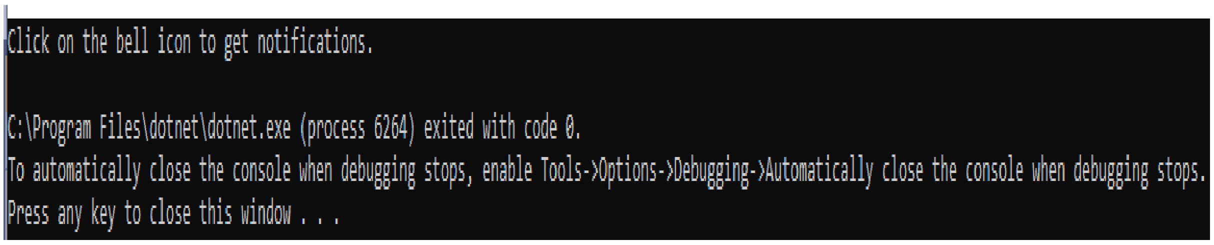
Q2) Explain how dependency injection is demonstrated in the provided code, particularly in the simple class's notify method.

code:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Text;
using System.Threading.Tasks;
```

```
public interface INotificationAction
{
    void ActOnNotification(string message);
}

class simple
{
    INotificationAction task = null;

    public void notify(INotificationAction at, string messages)
    {
        this.task = at;
        task.ActOnNotification(messages);
    }
}

class EventLogWriter : INotificationAction
{
    public void ActOnNotification(string message)
    {
        Console.WriteLine("Click on the bell icon to get notifications.");
    }
}
class Program
{
    static void Main(String[] args)
    {
        EventLogWriter elw = new EventLogWriter();
        simple at = new simple();
        at.notify(elw, "to logg");
        Console.ReadKey();
    }
}
```

Output:



```
Click on the bell icon to get notifications.

C:\Program Files\dotnet\dotnet.exe (process 6264) exited with code 0.
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the console when debugging stops.
Press any key to close this window . . .
```

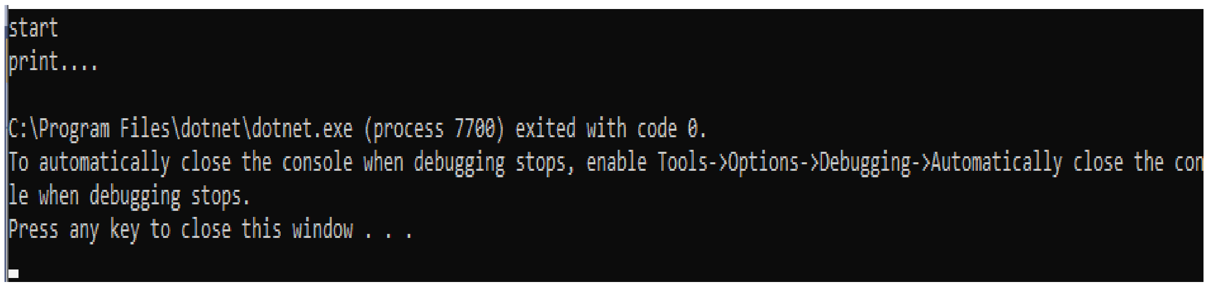
Q3) How does the client class utilize dependency injection through the run method, and what is the benefit of this approach?

code:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Text;
using System.Threading.Tasks;
public interface Iset
```

```
{
    void print();
}

public class service : Iset
{
    public void print()
    {
        Console.WriteLine("print....");
    }
}
public class client
{
    private Iset _Iset;
    public void run(Iset serv)
    {
        this._Iset = serv;
        Console.WriteLine("start");
        this._Iset.print();
    }
}

class method
{
    public static void Main()
    {
        client cn = new client();
        cn.run(new service());
        Console.ReadKey();
    }
}
```

Output:



```
start
print....

C:\Program Files\dotnet\dotnet.exe (process 7700) exited with code 0.
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the con
le when debugging stops.
Press any key to close this window . . .
```

Q 4. **Create an ASP.NET Core MVC application that demonstrates custom URL Routing by creating your own URL pattern. The pattern should accept a student name in the URL and display the student's name as plain text in the browser.**

1.Program.cs

using System;

```
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore;
using Microsoft.AspNetCore.Hosting;
using Microsoft.Extensions.Configuration;
using Microsoft.Extensions.Logging;

namespace WebApplication4
{
  public class Program
  {
    public static void Main(string[] args)
    {
      CreateWebHostBuilder(args).Build().Run();
    }

    public static IWebHostBuilder CreateWebHostBuilder(string[] args) =>
      WebHost.CreateDefaultBuilder(args)
        .UseStartup<Startup>();
  }
}
```

2. Index.cshtml

```
@{
  ViewData["Title"] = "Student Index";
}

<h2>Student Index</h2>
```

3. Startup.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Builder;
using Microsoft.AspNetCore.Hosting;
using Microsoft.AspNetCore.Http;
using Microsoft.AspNetCore.HttpsPolicy;
using Microsoft.AspNetCore.Mvc;
using Microsoft.Extensions.Configuration;
using Microsoft.Extensions.DependencyInjection;

namespace WebApplication4
{
   public class Startup
   {
      public Startup(IConfiguration configuration)
      {
         Configuration = configuration;
      }

      public IConfiguration Configuration { get; }

      // This method gets called by the runtime. Use this method to add services to the container.
      public void ConfigureServices(IServiceCollection services)
      {
         services.Configure<CookiePolicyOptions>(options =>
         {
            // This lambda determines whether user consent for non-essential cookies is needed for a
given request.
```

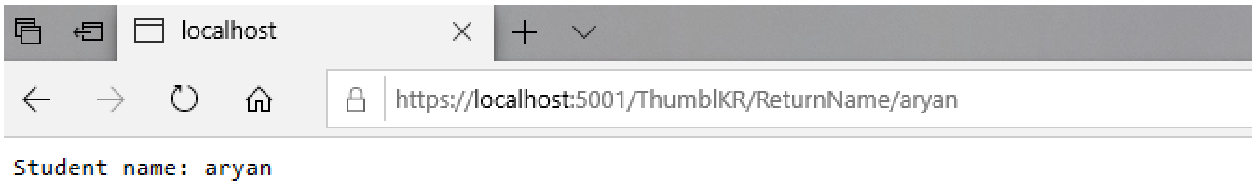
```
            options.CheckConsentNeeded = context => true;
            options.MinimumSameSitePolicy = SameSiteMode.None;
        });


        services.AddMvc().SetCompatibilityVersion(CompatibilityVersion.Version_2_1);
    }

    // This method gets called by the runtime. Use this method to configure the HTTP request
pipeline.
    public void Configure(IApplicationBuilder app, IHostingEnvironment env)
    {
        if (env.IsDevelopment())
        {
            app.UseDeveloperExceptionPage();
        }
        else
        {
            app.UseExceptionHandler("/Home/Error");
            app.UseHsts();
        }

        app.UseHttpsRedirection();
        app.UseStaticFiles();
        app.UseCookiePolicy();

        app.UseMvc(routes =>
        { //changes here
            routes.MapRoute(
                name: "MYAPI",
                template: "ThumblKR/ReturnName/{studentName}",
                defaults : new { Controller = "Student", action = "ReturnName" });
            });
        }
    }
}
```

4. StudentController.cs

```
using Microsoft.AspNetCore.Mvc;

namespace WebApplication4.Controllers
{
    public class StudentController : Controller
    {
        public IActionResult Index()
        {
            return View();
        } //changes here

        public IActionResult ReturnName(string studentName)
        {



            return Content("Student name: " + studentName);
        }
    }
}
```

Note : In browser type : https://localhost:5001/ThumblKR/ReturnName/ (your name)

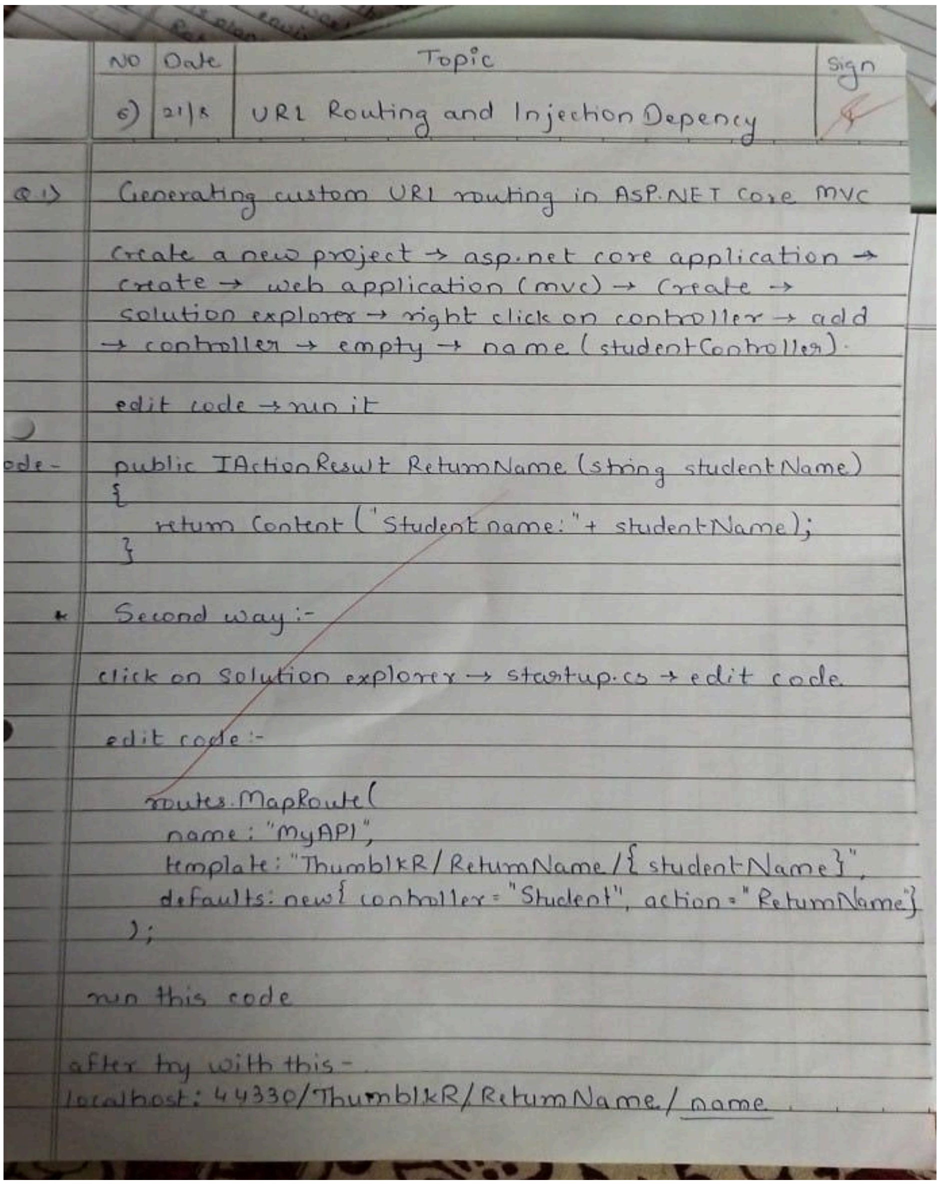
Output :


localhost

https://localhost:5001/ThumblKR/ReturnName/aryan

```
Student name: aryan
```

You can refer steps from here:


NO	Date	Topic	Sign
6)	21/8	URL Routing and Injection Depency	

Q.1) Generating custom URL routing in ASP.NET Core MVC

Create a new project → asp.net core application → create → web application (mvc) → Create → solution explorer → right click on controller → add → controller → empty → name (studentController).

edit code → run it

code-

```
public IActionResult ReturnName (string studentName)
{
   return Content ("Student name:"+ studentName);
}
```

* Second way :-

click on Solution explorer → startup.cs → edit code.

edit code :-

```
routes.MapRoute(
   name: "MyAPI",
   template: "ThumblKR/ReturnName/{studentName}",
   defaults: new{ controller="Student", action="ReturnName"}
);
```

run this code

after try with this -
localhost: 44330/ThumblKR/ReturnName/name.