Exercise 1, Method Overloading

Create a `Player` class which has three overloaded `Shout` methods. The first `Shout` method should take a string as argument, the second an integer and the third an `Enemy` object (`Enemy` being a simple class containing an `int Damage` property).

The first `Shout` method should print the string provided, the second should print *"[int] is my lucky number!"* and the third should print *"The enemy can do [Damage] damage to me."*. Test the overloaded methods of your `Player` in the `Main` method of your program.

Exercise 2, Extension methods

Create an extension method for the `List` class that returns a random item from the list. The extension method should work on lists of any type.

Create another extension method for the `List` class that shuffles the list.

Exercise 3, Creating and Using Generic Stacks

Using the generic `Stack` class (in `System.Collections.Generic`), create an empty stack of integers and an empty stack of strings. Create a static method that takes a generic `Stack` as an argument and a list of generic values (using the `params` keyword). The method should push all the generic values to the generic stack. Next, use this method to add some values to each stack and print them to the console.

Exercise 4, Interfaces and Generics

Declare a generic interface `IExplodable<T>` that contains a void `Explode(T radius)` method. Create a class `Bomb` that implements this interface. Provide an implementation for the `Explode` method such that a program that uses an instance of the `Bomb` class can explode the instance by providing a double to the `Explode` method, writing *"Boom!"* and the size of the explosion to the console.

Exercise 5, Operator Overloading

Create a `Time` class with two integer properties, `Hours` and `Minutes`. Overload the + operator so that it adds two `Time` objects, returning a new `Time` object. Create another overload method that adds an integer value (in minutes) to a `Time` object and returns a new `Time` object. Override the `ToString` method so that printing a `Time` object to the console displays *"Hours: X, Minutes: Y"*.

Exercise 6, Ternary Operator

Rewrite the code below to use the ternary operator (`?:`) (you should be able to condense the if-else logic into one line).

```
    int score = 42;
    string message;

    if (score > 1337)
    {
        message = "This is a new highscore!";
    }
    else
    {
        message = "You need more points to beat the highscore!";
    }
```

Verify that rewritten code works by testing it in the console.

Exercise 7, Delegates

Create a `Notifier` delegate that takes a string as an argument and returns `void`. Implement two methods, `SayHello` and `SayGoodbye` that prints *"Hello [name]"* and *"Goodbye [name]"*, respectively, to the console. The methods should be compatible with the `Notifier` delegate.

Use the `Notifier` delegate to first print *"Hello [name]"* to the console when the application is run. Next, modify the code so that *"Goodbye [name]"* is printed instead. Lastly, modify and add to the code so that *"Hello [name]"* and *"Goodbye [name]"* are both printed to the console.

Exercise 8, Lambda-expressions

Create a class `Car` with at least the properties `Color`, `EngineSize` and `FuelEconomy` (in liters pr 100km). Override `ToString()` so that it prints the properties of the car. Create a `List` of three or more cars with varying properties.

Create a predicate method that takes a `Car` as argument and checks if the car is a particular color. Use the predicate method in combination with the `FindAll()` method of the `List` instance to find all cars of that color in your list and then print them to the terminal.

Replace your predicate method with lambda expressions to find all cars that match the following:

- a particular color
- an engine size bigger than some value
- a fuel economy lower than some value
- a condition combining 2 or more properties

You should create a lambda expression for each condition above and test your lambda expressions one by one by printing the results to the terminal.