6. **[8 marks]**
   Which design pattern is implemented by the code within the `Example.cs` file?

   It is an implementation of the strategy pattern which says that if you have a class that does something specific, in our case travelling, in many different ways, you extract all of these algorithms in separate classess called strategies, in our case the different modes of travel, and rename the original class to context, in our case Transportation, that has a field for storing one of the strategies, working with all strategies (only) through the same generic interface, and delegating the work to a linked strategy instead of executing it on its own.

7. **[18 marks]**
   Create a single dimension array containing some string objects. Sort this vector by:

   - length (i.e., shortest to longest),
   - reverse length (i.e., longest to shortest),
   - first character, and
   - strings that contain the character `"` first, everything else second.

   You should use the functional programming features of C# to answer this question using LINQ.

8. **[25 marks]**

The fictitious company *ShiverOrSweat* develops temperature sensors for buildings. They have to develop a class which can be used for reading the temperature. You have just been hired by this company, and you are now to design part of this class, `Sensor`.

A method `Read` has already been written which returns a double indicating the temperature reading in celsius.

```
public double Read()
{
        //...
}
```

If the connection to the physical sensor is "out of order" `Read` returns `-300`, although this happens quite rarely.

- Implement a method `ReadTemperature` which calls the `Read` method. You should throw an exception when the `Read` method returns a value of `-300`.

  ```
  public double ReadTemperature()
  {
        // ...
  }
  ```

- For this example, why is it preferable to throw an exception as against returning the value `-300` which the calling method could check?
  Because else, what's the purpose of this method, the caller could just call the Read() method directly? Therefore perhaps the that ReadTemperature() is used to check if the connection between Read and the physical sensor is broken whilst in Read(), throwing an exception would mean that we don't always get back a number value and we want to keep the two concerns, reading input and checing its validity, separate.

- Why would it be preferable to return an `Option` rather than throwing an exception?

So that it always returns the same type of value regardless of whether the connection is out of order as the methods calling it may not be aware of the possibility that it throws an exception. Also, this way we a method calling it need not necessarily be bothered with the meaning of the values it returns so long as it returns the same type every time it's called. Or to enable fluent expressions. (builder style)

- Write a method `Monitor` which monitors the temperature. The method has two arguments, `low` and `high`.

  ```
  public void Monitor(double low, double high)
  {
        // ...
  }
  ```

  The method `Monitor` should read the temperature every second, and write out a warning (to the standard output) if the temperature is below `low`, or above `high`. If there is no response from the sensor for one minute the sensor the `Monitor` method should write a special warning to standard output.