Home task

INSURANCE COMPANY

2017

if...

Relax, we'll help you.

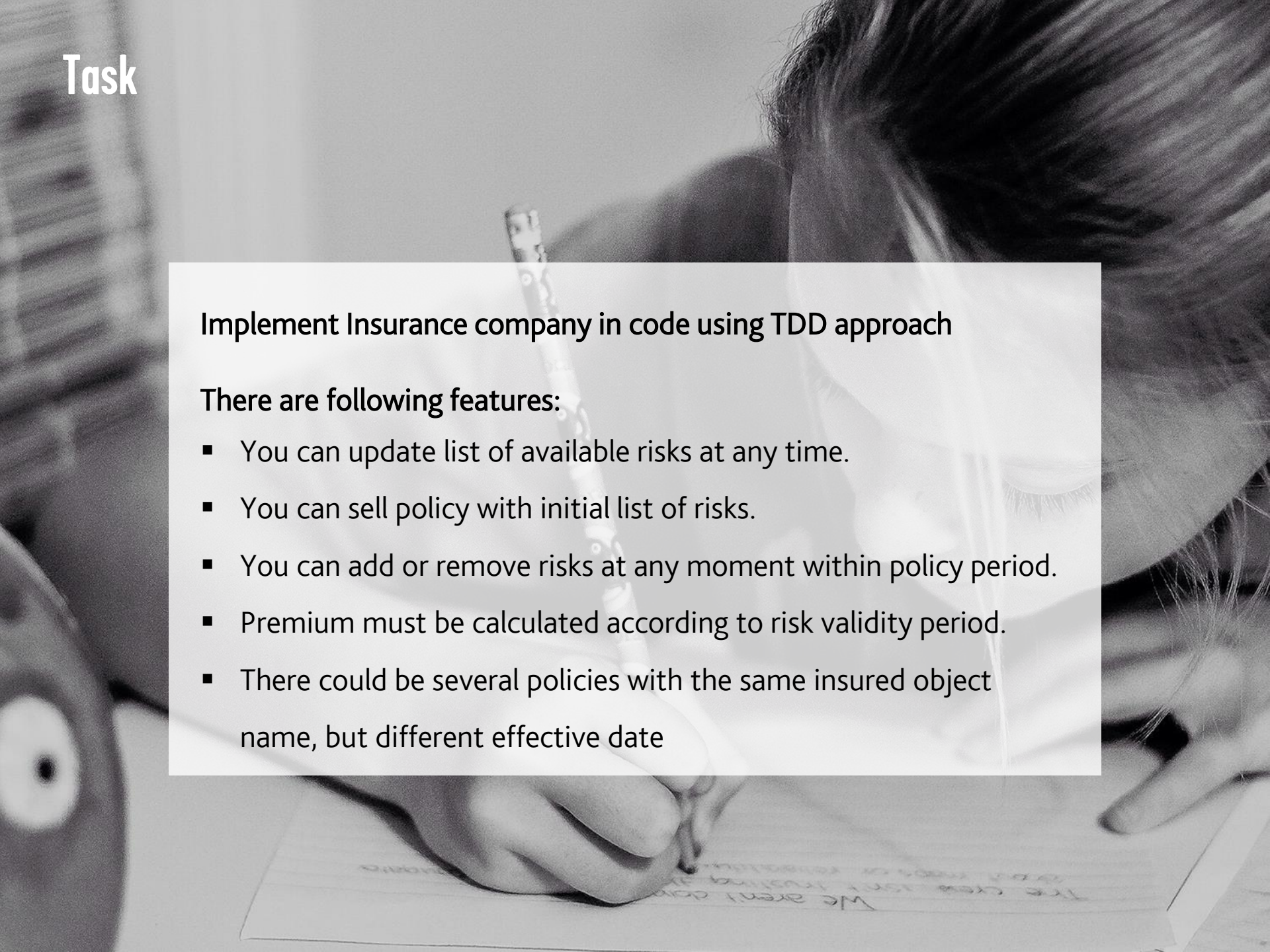
Insurance company

Think about it as a real insurance company

Implement rules which seems logical to you

Relax, we'll help you.

Task



Implement Insurance company in code using TDD approach

There are following features:

- You can update list of available risks at any time.
- You can sell policy with initial list of risks.
- You can add or remove risks at any moment within policy period.
- Premium must be calculated according to risk validity period.
- There could be several policies with the same insured object name, but different effective date

We are giving the interface of Insurance company

```
public interface IInsuranceCompany
{
    /// <summary>
    /// Name of Insurance company
    /// </summary>
    string Name { get; }

    /// <summary>
    /// List of the risks that can be insured. List can be updated at any time
    /// </summary>
    IList<Risk> AvailableRisks { get; set; }

    /// <summary>
    /// Sell the policy.
    /// </summary>
    /// <param name="nameOfInsuredObject">Name of the insured object. Must be unique in the given period.</param>
    /// <param name="validFrom">Date and time when policy starts. Can not be in the past</param>
    /// <param name="validMonths">Policy period in months</param>
    /// <param name="selectedRisks">List of risks that must be included in the policy</param>
    /// <returns>Information about policy</returns>
    IPolicy SellPolicy(string nameOfInsuredObject, DateTime validFrom, short validMonths, IList<Risk> selectedRisks);

    /// <summary>
    /// Add risk to the policy of insured object.
    /// </summary>
    /// <param name="nameOfInsuredObject">Name of insured object</param>
    /// <param name="risk">Risk that must be added</param>
    /// <param name="validFrom">Date when risk becomes active. Can not be in the past</param>
    /// <param name="effectiveDate">Point of date and time, when the policy effective</param>
    void AddRisk(string nameOfInsuredObject, Risk risk, DateTime validFrom, DateTime effectiveDate);

    /// <summary>
    /// Remove risk from the policy of insured object.
    /// </summary>
    /// <param name="nameOfInsuredObject">Name of insured object</param>
    /// <param name="risk">Risk that must be removed</param>
    /// <param name="validTill">Date when risk become inactive. Must be equal to or greater than date when risk become active</param>
    /// <param name="effectiveDate">Point of date and time, when the policy effective</param>
    void RemoveRisk(string nameOfInsuredObject, Risk risk, DateTime validTill, DateTime effectiveDate);

    /// <summary>
    /// Gets policy with a risks at the given point of time.
    /// </summary>
    /// <param name="nameOfInsuredObject">Name of insured object</param>
    /// <param name="effectiveDate">Point of date and time, when the policy effective</param>
    /// <returns></returns>
    IPolicy GetPolicy(string nameOfInsuredObject, DateTime effectiveDate);
}
```

Relax, we'll help you.

We are giving the interface of Insurance company

```
public struct Risk
{
    /// <summary>
    /// Unique name of the risk
    /// </summary>
    public string Name { get; set; }

    /// <summary>
    /// Risk yearly price
    /// </summary>
    public decimal YearlyPrice { get; set; }
}

public interface IPolicy {
    /// <summary>
    /// Name of insured object
    /// </summary>
    string NameOfInsuredObject { get; set; }

    /// <summary>
    /// Date when policy becomes active
    /// </summary>
    DateTime ValidFrom { get; set; }

    /// <summary>
    /// Date when policy becomes inactive
    /// </summary>
    DateTime ValidTill { get; set; }

    /// <summary>
    /// Total price of the policy. Calculate by summing up all insured risks.
    /// Take into account that risk price is given for 1 full year. Policy/risk period can be shorter.
    /// </summary>
    decimal Premium { get; set; }

    /// <summary>
    /// Initially included risks of risks at specific moment of time.
    /// </summary>
    IList<Risk> InsuredRisks { get; set; }
}
```

Relax, we'll help you.

Take into account

- Use Visual Studio 2013 express or any paid and/or newer version if available
- Use TDD approach
- Use C# language
- Think about OOP design patterns and S.O.L.I.D. principles
- In case of error, throw different type of exceptions for each situation
- Comments and code must be in English language
- No need for UI
- As a result we expect the solution with source code
- In case of any questions, please contact us
- Consider given code as a Insurance Company SDK

Relax, we'll help you.


Good Luck!