В програмите, които един софтуерен разработчик създава, често се налага да се наблюдава настъпването на различни събития или промяната на някакви стойности.

Като прост пример може да си представим един сензор, който измерва и излъчва различни стойности през интервали от време. Кой обаче наблюдава тези стойности? На теория може **неограничен** брой "наблюдатели" да се "закачат" към сензора с цел получаване на всяка нова измерена стойност.

Могат да съществуват обаче различни начини както за наблюдаване, така и за изпращане на стойности. В тази задача ще направим няколко такива, с помощта на предоставените хедъри и условията към тях, описани долу.

**Данните**

**Message**

Класът с данни се казва Message (съобщение) и съдържа конструктор с параметър от тип int, който запазва стойността му в публично поле от същия тип с името data. Стойността на това поле, веднъж инициализирана, не трябва да може да бъде променяна.

**Класове, които излъчват данни (Publishers)**

**SimplePublisher**

* методът subscribe позволява закачането на още един **конкретен** наблюдател към инстанцията
* методът unsubscribe позволява разкачането на **конкретен** наблюдател от инстанцията
* методът signal изпраща параметъра от тип Message на всички наблюдатели, закачени към инстанцията, чрез извикване на техния метод signal

**BacklogPublisher**

* методът subscribe позволява закачането на още един **конкретен** наблюдател към инстанцията, както и веднага му изпраща абсолютно всички предишни ("пропуснати") съобщения чрез signal (в реда, в който са били получени)
* методът unsubscribe позволява разкачането на **конкретен** наблюдател от инстанцията
* методът signal изпраща параметъра от тип Message на всички наблюдатели, закачени към инстанцията, чрез извикване на техния метод signal

**Класове, които наблюдават данни (Subscribers)**

Всеки един от тях трябва да съдържа публично поле с име id от тип std::string с цел идентификация (два наблюдетеля с едно и също id да се смятат за равни). Стойността на полето, веднъж инициализирана, не трябва да може да бъде променяна.

Получаването на съобщения става чрез signal, а четенето им след това - чрез read. Разликата между трите класа се проявява при четенето и обработването на получените данни.

При никакви получени данни и трите класа връщат 0 при четене.

**Averager**

* При четене връща средната стойност на всички данни, получени досега.

**MovingAverager**

* Съдържа публично поле с име windowSize от тип size\_t. Стойността на това поле, веднъж инициализирана, не трябва да може да бъде променяна.
* При четене връща средната стойност само на последните windowSize на брой съобщения.

**PeriodicSampler**

* Приема в конструктор period от тип size\_t
* При четене връща стойността на последното получено неигнорирано съобщение. За игнорирано съобщение смятаме всяко n-то съобщение след първото получено, за което n % period != 0

Пример:

Averager\* avg = new Averager("id1");

MovingAverager\* movAvg = new MovingAverager("id2", 5);

PeriodicSampler\* perSam = new PeriodicSampler("id3", 3);

​

SimplePublisher pub;

pub.subscribe(avg);

pub.subscribe(movAvg);

pub.subscribe(perSam);

​

pub.signal(1);

pub.signal(2);

pub.signal(3);

pub.signal(4);

pub.signal(5);

pub.signal(6);

pub.signal(7);

pub.signal(8);

pub.signal(9);

​

avg.read(); // трябва да връща 5 (ср. аритм. на 1...9)

movAvg.read(); // трябва да връща 7 (ср. аритм. на 5...9)

perSam.read(); // трябва да връща 7 (стойността на шестото съобщение след първото; игнорира стойности 8 и 9)

**Клас, който управлява наблюдатели**

Клас Repository служи като хранилище за различни наблюдатели, независимо от това от къде те получават своите данни.

Поддържа операциите добавяне (създаване) на нов наблюдател и достъп до наблюдател чрез неговото id.