Assignment 1: Pipe-and-Filter

Runa Capital Team

Peter Rezikov

Nikita Mingazov

Maxim Skorikov

Artur Tazhitdinov

Alexandra Kolesnikova

Mohamad Kassab

Bulat Gabbasov

Agenda

Business goals & Engineering Objectives

Quality Attribute Scenarios

Constraints

Functional Requirements

Views

Q & A

About the project

Customer builds instrumentation systems

Software should support variety of platforms

Software should be easily reconfigured

Process data stream “as quickly as possible”

Business Goals

Business Goal	Engineering Objective
Reduce costs of development of typical instrumentation system	Ease creation of different types of processors
	Provide framework for combination of processors into single unit
Expand on variety of markets including real-time systems	Maximize speed of processing data
	Provide ability to reconfigure system for a variety of applications and platforms

Quality Attribute Scenarios

Engineering Objective	Quality Attribute	Quality Attribute Scenario	Priority
Ease creation of different types of processors	Modifiability	A developer should be able to develop and test a new processor in 40 man hours.	H
Provide framework of combination of processors into single unit	Reusability	A developer should be able to construct a new filter network from prepared processors without creating new framework constructions in 24 man hours.	H
Maximize speed of processing data	Performance	A system should able to process data asynchronously and concurrently.	H
Provide ability to reconfigure system for a variety applications and platforms	Modifiability	A developer should be able to reconfigure existing system for new platform and test it in 8 man hours.	H

Technical Constraints

Java

Pipe-and-filter pattern

Filter Framework as basic class

Business Constraints

System should gather multiple “processors” together

This couples should form an application

This software should support multiple platforms

Functional Requirements: System A

Read data stream in binary format

Convert temperature measurements from Fahrenheit to Celsius

Convert altitude from feet to meters

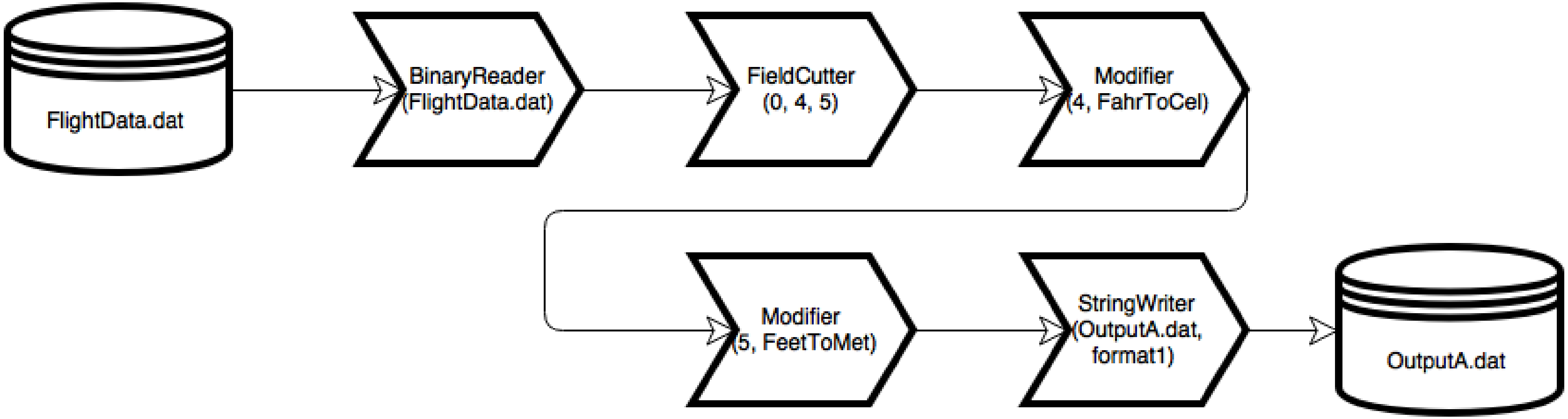
Write the output to a text file

Binary format description

$$\underbrace{\texttt{0000 0000}}_{\text{ID}}\ \underbrace{\texttt{0000 0143 91ec ca9e}}_{\text{Measurement}}\ \underbrace{\texttt{0000 0001}}_{\text{ID}}\ \underbrace{\texttt{400e fdb7}}_{\text{Measurement}}$$

Frame 1	ID: 000	Time Data	ID: 001	Data	...	ID: 005	Data
Frame 2	ID: 000	Time Data	ID: 001	Data	...	ID: 005	Data
...	...	...	...	...	...	...	...
Frame N	ID: 000	Time Data	ID: 001	Data	...	ID: 005	Data

System A dynamic views alternatives



FlightData.dat

BinaryReader
(FlightData.dat)

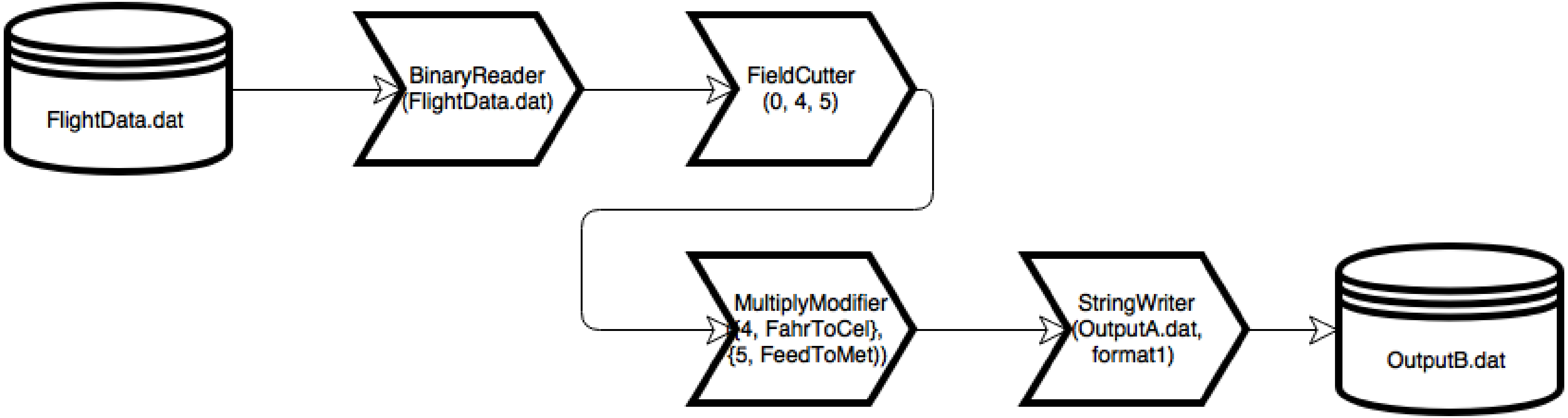
FieldCutter
(0, 4, 5)

Modifier
(4, FahrToCel)

Modifier
(5, FeetToMet)

StringWriter
(OutputA.dat,
format1)

OutputA.dat



FlightData.dat

BinaryReader
(FlightData.dat)

FieldCutter
(0, 4, 5)

MultiplyModifier
({4, FahrToCel},
{5, FeedToMet})

StringWriter
(OutputA.dat,
format1)

OutputB.dat

Functional Requirements: System B

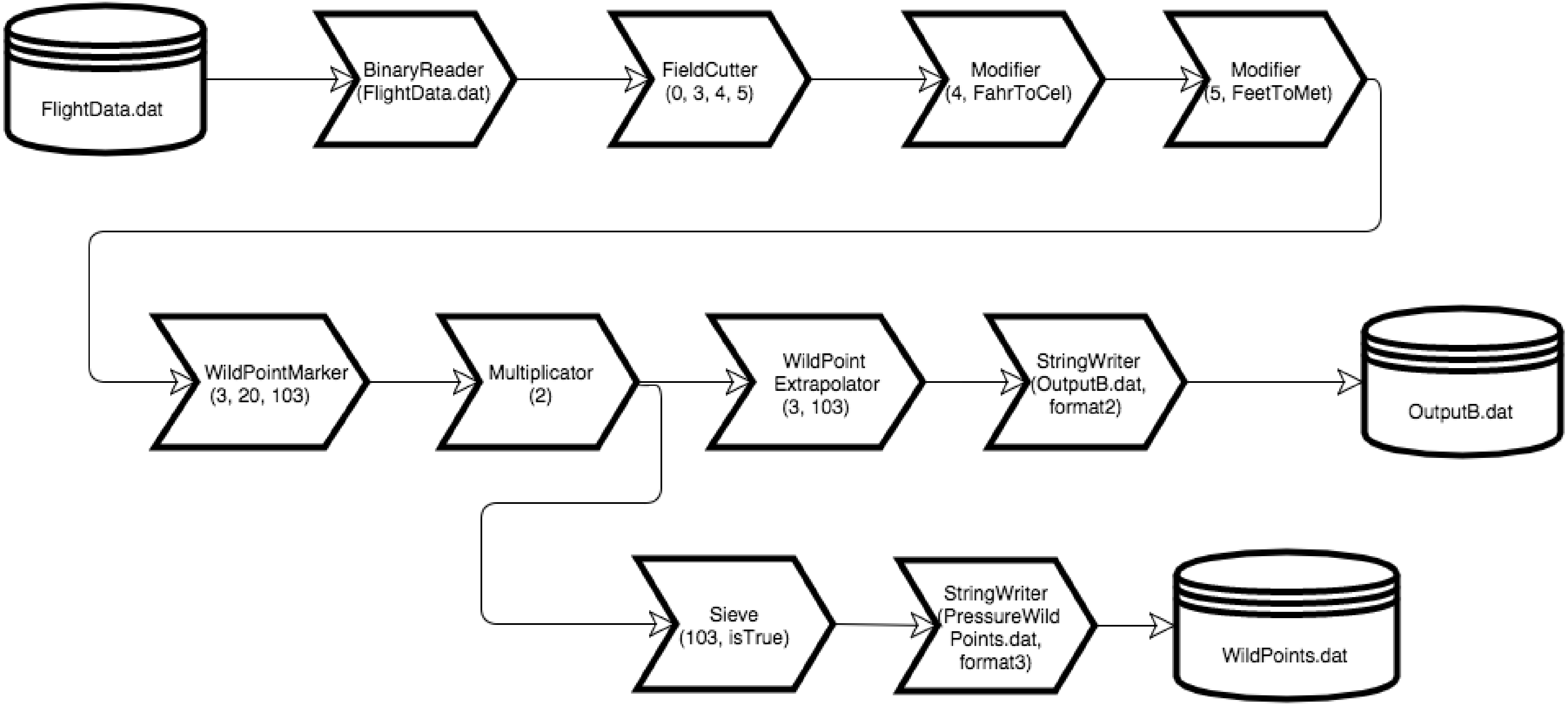
The same to the system A

Recognize Wild Points - negative measurements or measurements that deviates from previous on more than 10

Filter Wild Points into separate file

Replace Wild Points with extrapolated values

System B dynamic view



FlightData.dat
BinaryReader (FlightData.dat)
FieldCutter (0, 3, 4, 5)
Modifier (4, FahrToCel)
Modifier (5, FeetToMet)
WildPointMarker (3, 20, 103)
Multiplicator (2)
WildPoint Extrapolator (3, 103)
StringWriter (OutputB.dat, format2)
OutputB.dat
Sieve (103, isTrue)
StringWriter (PressureWild Points.dat, format3)
WildPoints.dat

Functional Requirements: System C

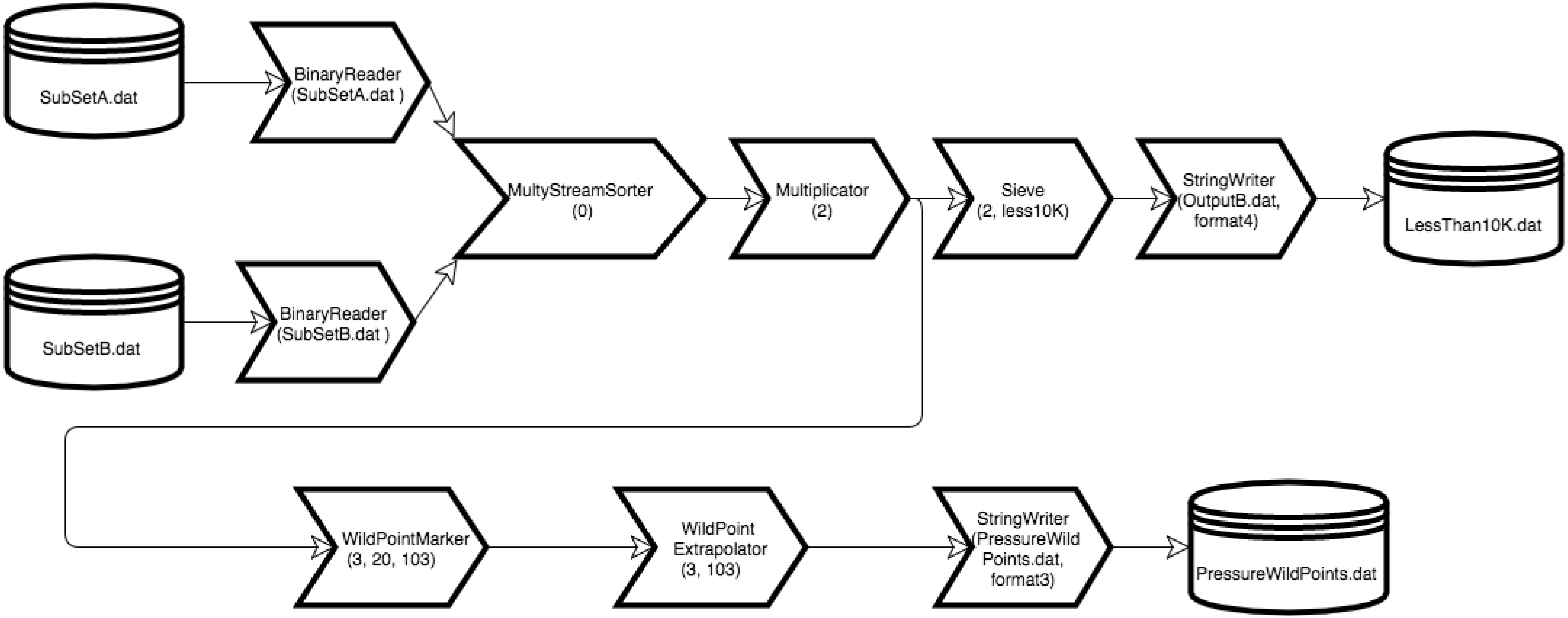
Read two input files sorted by timestamp

Merge them into single output stream

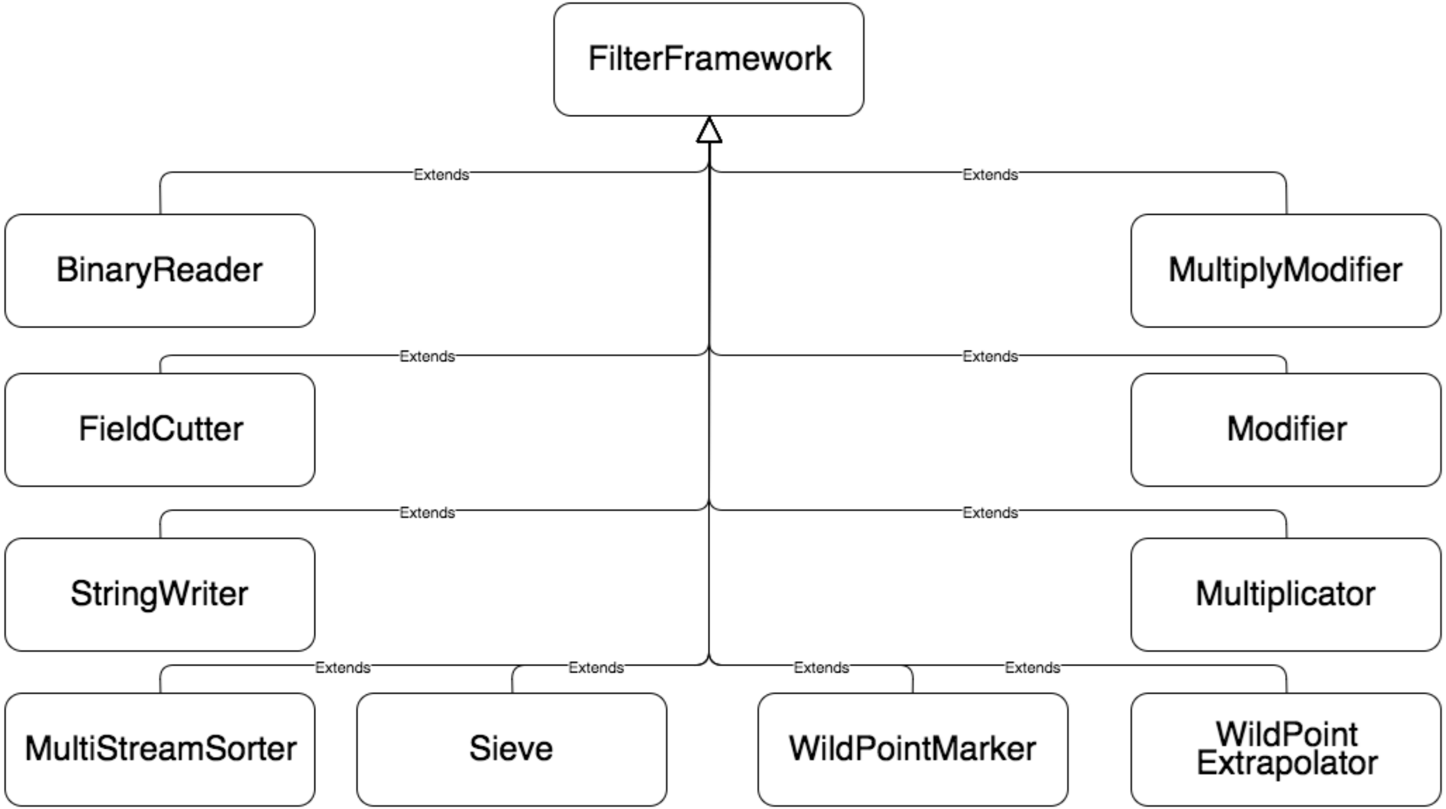
Produce two outputs:

- Data collected only below 10,000 feet
- Data with replaced pressure wildpoints

System C dynamic view


SubSetA.dat
BinaryReader (SubSetA.dat)
SubSetB.dat
BinaryReader (SubSetB.dat)
MultyStreamSorter (0)
Multiplicator (2)
Sieve (2, less10K)
StringWriter (OutputB.dat, format4)
LessThan10K.dat
WildPointMarker (3, 20, 103)
WildPoint Extrapolator (3, 103)
StringWriter (PressureWild Points.dat, format3)
PressureWildPoints.dat

Static view


FilterFramework
Extends
Extends
BinaryReader
MultiplyModifier
Extends
Extends
FieldCutter
Modifier
Extends
Extends
StringWriter
Multiplicator
Extends
Extends
Extends
Extends
MultiStreamSorter
Sieve
WildPointMarker
WildPoint Extrapolator

Q & A

Thank you for your attention!

Filters 1

BinaryReader
Purpose: Read binary stream from file
Config params: filePath:String
Input: None
Output: binary

MultiplyModifier (alternative to Modifier)
Purpose: Modify multiply field by given algorithms
Config params: Array(filedId, changer: function(8 bytes) -> 8 bytes)
Input: binary
Output: binary with each fieldId modified by reated changer

FieldCutter
Purpose: Filter not useful fields due to performance reasons
Config params: neccessaryIds<Array[int]>
Input: binary
Output: binary only with measurement with ids from neccessaryIds array

StringWriter:
Purpose: Write string data to the file
Config params: filePath:String, header:String, frameTemplate:String
frameTeblate example: %0{YYYY:DD:HH:MM:SS} %4{TTT.ttttt} %5{AAAAAA.aaaaa}
Input: binary
Output:
header and then
for each frame text in the given frameTemplate format

Modifier
Purpose: Modify one field by given algorithm
Config params: filedId, changer: function(8 bytes) -> 8 bytes
Input: binary
Output: binary with fieldId modified by changer

Filters 2

Multipicator:
Purpose: Get data from one stream and dublicate it to number of new streams
Config params: streamsNumber:Int
Input: binary
Output: streamsNumber binary streams

MultiStreamSorter:
Purpose: Gives sorted (by field value) stream from many streams (sorted by field value)
Config params: fieldId:Int
Input: multiply binary streams
Output: binary stream sorted by fieldId value
(It is ok in case of time, but for floats and double should be modified)

Sieve:
Purpose: Filter out frames by predicate applied to the field.
Config params: fieldId:Int, predicate: function(8 bytes) -> bool
Input: binary
Output: binary stream for which predicate for data from fieldId is true

Filters 3

WildPointMarker:
  Purpose: Find and mark wild points
  Config params: deviationFieldId:Int, maxDeviation:Int# future isDeviate function(8 bytes) -> Bool
  Input: binary
  Output: binary with marked wild points in fieldId, which deviates by maxDeviation and save status to deviationFieldId (0 or 1)

WildPointExtrapolator:
  Purpose: Extropolate wild points
  Config params: fieldId:Int
  Input: binary
  Output: binary with fieldId extrapolated by neighbor right frames