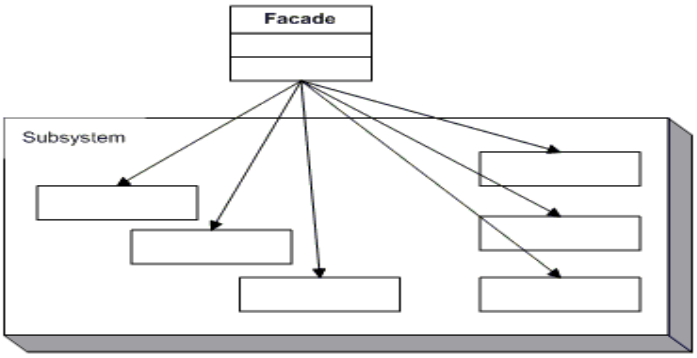
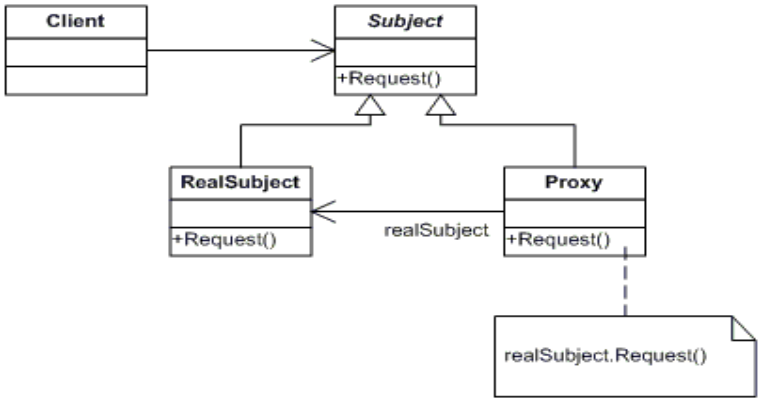
Software design patterns: 2010 – 2011

(exercise 4 – Proxy, Facade, Flyweight)

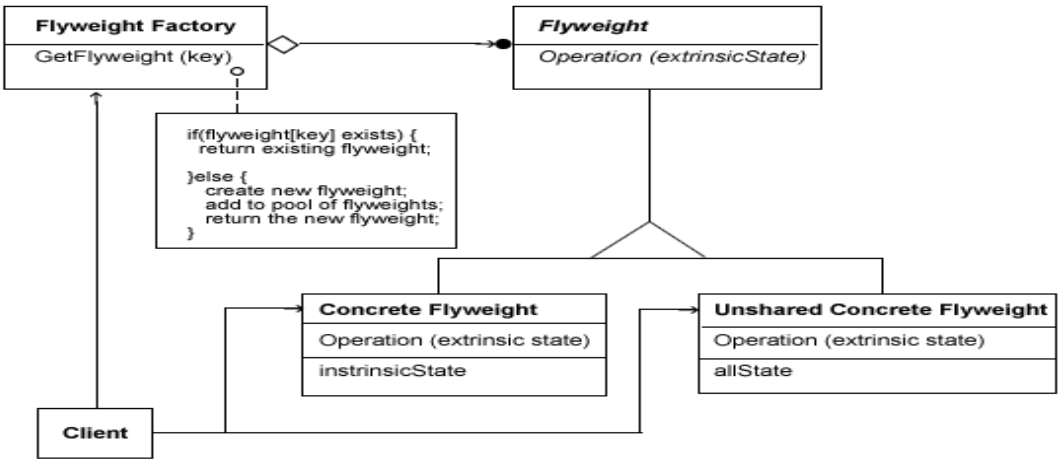
Facade: Предоставя унифициран интерфейс към набор от интерфейси в дадена подсистема.



Proxy: Предоставя заместител или празна имплементация на друг обект, за да контролира достъпа до него



Flyweight: Използва поделяне с т. нар. външни и вътрешни състояния, за ефективна поддръжка на големи количества малки обекти



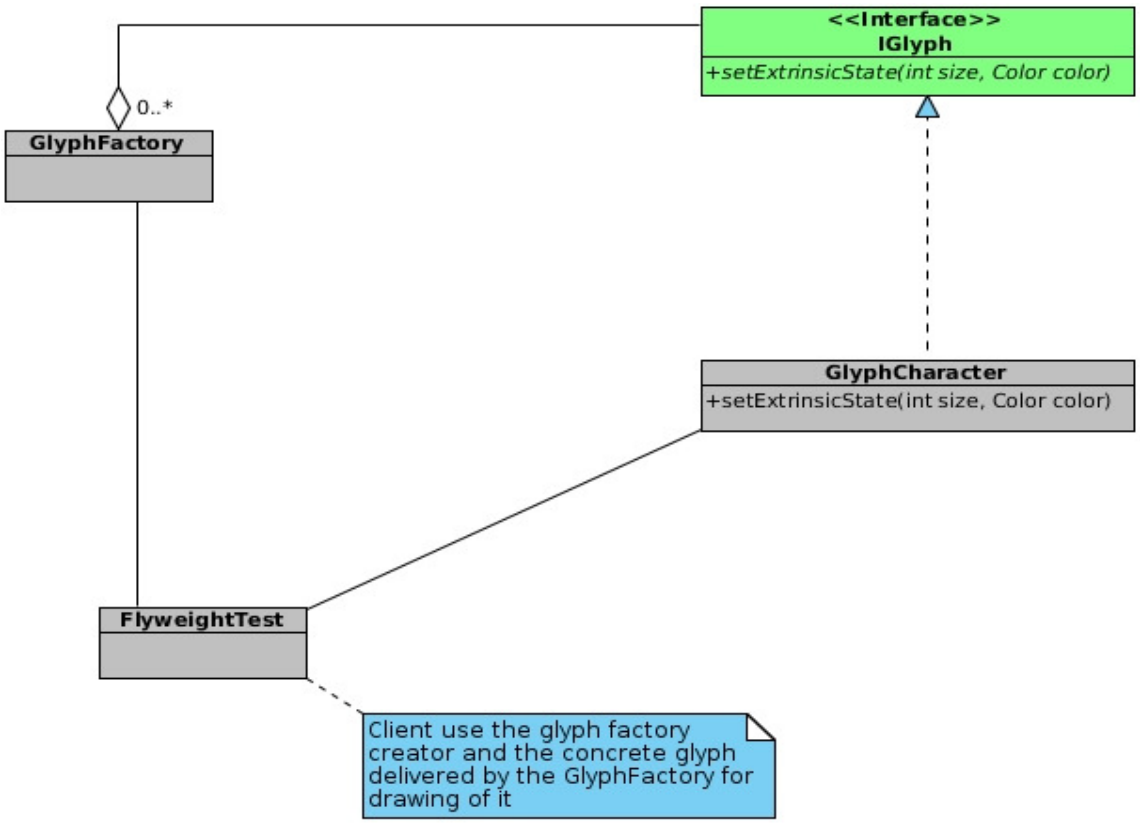
Задача 1:

Нека да стартираме тестовия клас : **FlyweightTest**
, чиято задача е да принтира на конзолата букви от английската азбука с определен цвят и размер. За целта той при всяка итерация – ново изискване да се принтира буква – създава инстанция на **GlyphCharacter** и изиква медота draw(), който се грижи за самото принтване. Резултатът от подобен вид реализирана програма е създаването на безброй много инстанции на **GlyphCharacter** класа, имащ грижата да принтира вътрешното си състояние.

Но ако помислим на нас фактически ни трябват най-много 27 инстанции на този клас: по една инстанция за всяка буква и разбира се, за да можем да принтираме буквите с различна големина и цвят, също така трябва да имаме възможност да променяма стойността на тези параметри за конкретна инстанция, представляваща конкретна буква.

За това, позовавяйки се на Flyweight шаблона, отразен в следната диаграма за нашия случай:

Нека рефакторираме кода по следния начин:



1.1. Създаддем нов интерфейс: **IGlyph**, който де декларира метода, необходим за инициализиране на т.н. extrinsic state (променящият се във вече стартирана програма) на конкретния **GlyphCharacter** : т.с. Размера и цвета на буквата , докато internal state (този, който не се променя) е самата буква – char

1.2. Нека добавим и **GlyphFactory**, чиято цел ще е да предоставя инстанции на **GlyphCharacter** като създава нова и я запазва в собственото си състояние само тогава, когато за пръв път се изисква обект **GlyphCharacter** с все още неизпозлвана буква, в противен случай , връща вече запазената инстанция на **GlyphCharacter** (вия изберете структурата, в която ще се пазят обектите, представляваща класа **GlyphCharacter**)

1.3. Нека сега използвайки **GlyphFactory,** променим тестовия сорс код на **FlyweightTest** и тестваме наново програмата

Задача 2:

Целта на програмата в папка: **properyBundleModule** е да **чете** стойността на зададено име на пропърти и **записва** в пропърти файл двойка: име на пропърти <–> стойност. Тези файлове често се използват в различните системи – примерно за превод на различни езици на едно интернет приложение: за всеки език има отделен файл като всеки файл името на пропъртито е едно и също, а стойността му: превода на значението за съответния език.

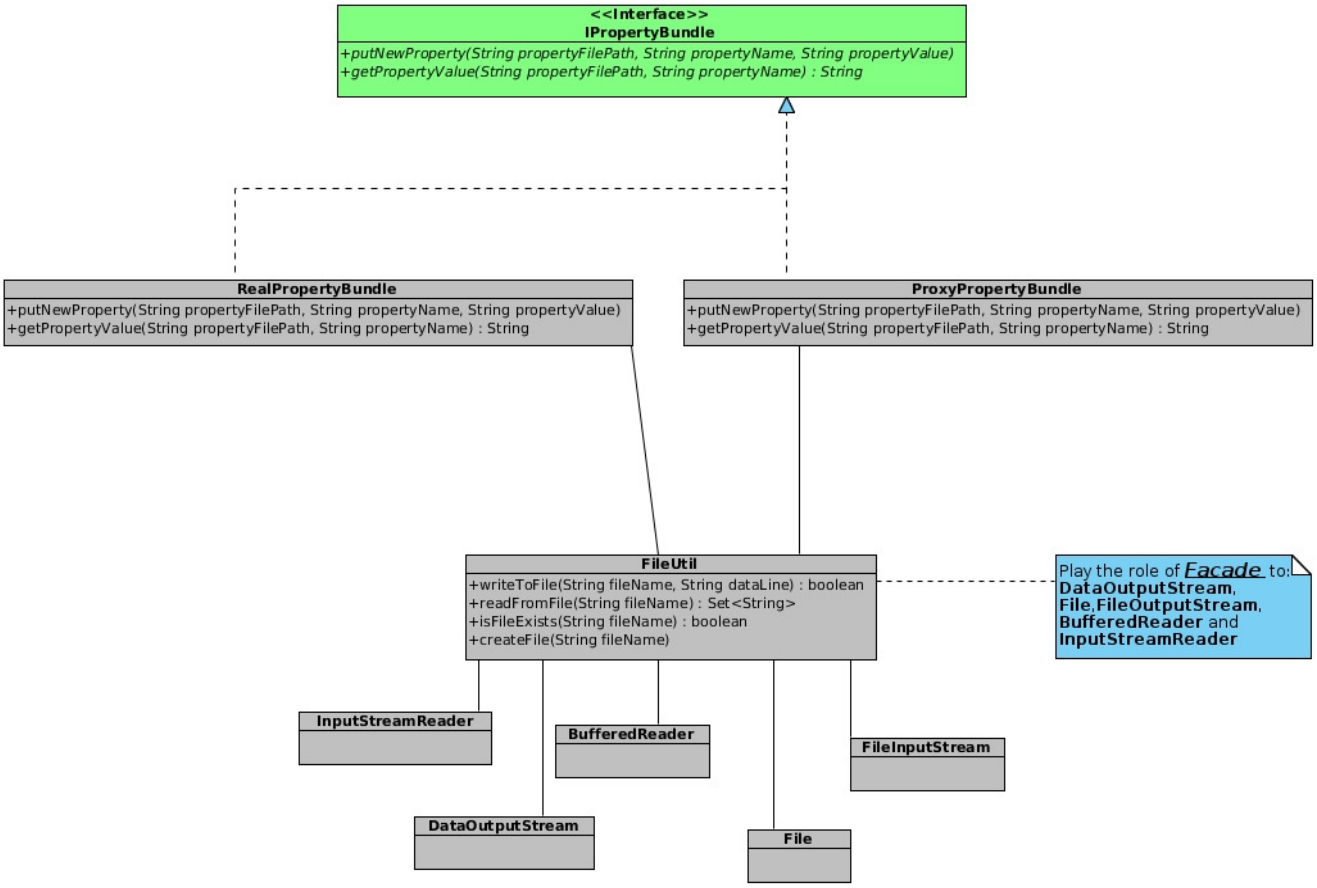
Ако стартираме програмата, чрез класа: **ProxyFacadePropertyBundleTest** ще забележим, че приграмата работи според изискванията.
Но нека се вгледаме в сорс кода на използвания клас: **PropertyBundle –** в него:

- имаме създаване на файл
- четене от файл
- записване във файл
- добавяне на нова двойка propertyName <-> propertyValue
- взимане на стойността за дадено пропърти

Силна обвързаност (strong coupling), слаба кохезия , трудно презипозлване на код , затруднена разширяемост са минусите в тази програма.
Нека подобрим този модул, използвайки следните два шаблона:

-> Facade design pattern
-> Proxy design pattern

и си помагаме със следната предоставена диаграма, изпълним следните стъпки:



2.1.Създадем класа **FileUtil**, който ще е нашата фасада (Fasade design pattern) към стандартните джаварскки класове за работа с файлове. В този клас имплементираме следните медоти, отговарящи на спецификацията:

2.1.1. **public static boolean** writeToFile(String fileName, String dataLine)
записва във файла с име fileName, добавяйки – не презаписвайки файла, стринга: dataLine (може да преизползвате отговарящия на опианисето код от **PropertyBundle класа**)

2.1.2. **public static** Set<String> readFromFile(String fileName)
чете всички редове от файл с име: fileName и ги връща като Set от стрингове – един ред от файле е един обект във връщащия резултат (множество) (може да преизползвате отговарящия на опианисето код от **PropertyBundle класа**)

2.1.3. **boolean** isFileExists(String fileName)
проверява дали файла с име fileName съществува

2.1.4. **public static void** createFile(String fileName)
създава нов файл с име: fileName

2.2.Сега ще имплементираме защитно пълномощтно : един от трите варианта на Proxy design шаблона като :

2.2.1. създадем интерфейса: **IPropertyBundle** , с добавените в него два метода, показани на диаграмата:

public void putNewProperty(String propertyFilePath, String propertyName, String propertyValue);

public String getPropertyValue(String propertyFilePath, String propertyName);

2.2.2. Създадем класа **RealPropertyBundle** имплементиращ създадения интерфейс като метода:

getPropertyValue(...) - взима всички редове от посочения файл (FileUtil.*readFromFile*(propertyFilePath)) и взима стойността на исканото проперти (може да преизползвате отговарящия на опианисето код от **PropertyBundle класа**),

докато за другия метод: **putNewProperty(...) -** конструира правилно реда за записване в следния вид: propertyName=propertyValue'\n' и извиква FileUtil.*writeToFile*(String fileName, String dataLine) метода за записване на новия ред

2.2.3 Създадем класа ProxyPropertyBundle , имплементиращ също интерфейса **IproxyBundle** и имащ инстанция към **RealPropertyBundle** като :

- в метода: **getPropertyValue(...)** проверява дали propertyFilePath или propertyName са коректно подадени : не са null pointer и не са празни стрингове. В такъв случай изписваме на конзолата подходящо съобщение и излизаме от метода

Ако проверките минат успешно подаваме параметрите на вътрешната инстанция на **RealPropertyBundle** калса да върне стойността на пропъртито

- в метода **putNewProperty(...)** проверява дали propertyFilePath или propertyName или propertyValue са коректно подадени : не са null pointer и не са празни стрингове. В такъв случай изписваме на конзолата подходящо съобщение и излизаме от метода.

Ако пурвите три проверки минат успешно проверяваме дали файла съществува, ако не съществува създава нов такъв (ипозлваме FileUtil.*isFileExists*(propertyFilePath) и FileUtil.*createFile*(propertyFilePath))

И разбира се: извикваме метода putNewProperty(...) на вътрешната инстанция на **RealPropertyBundle**, който да запише новата двойка параметри