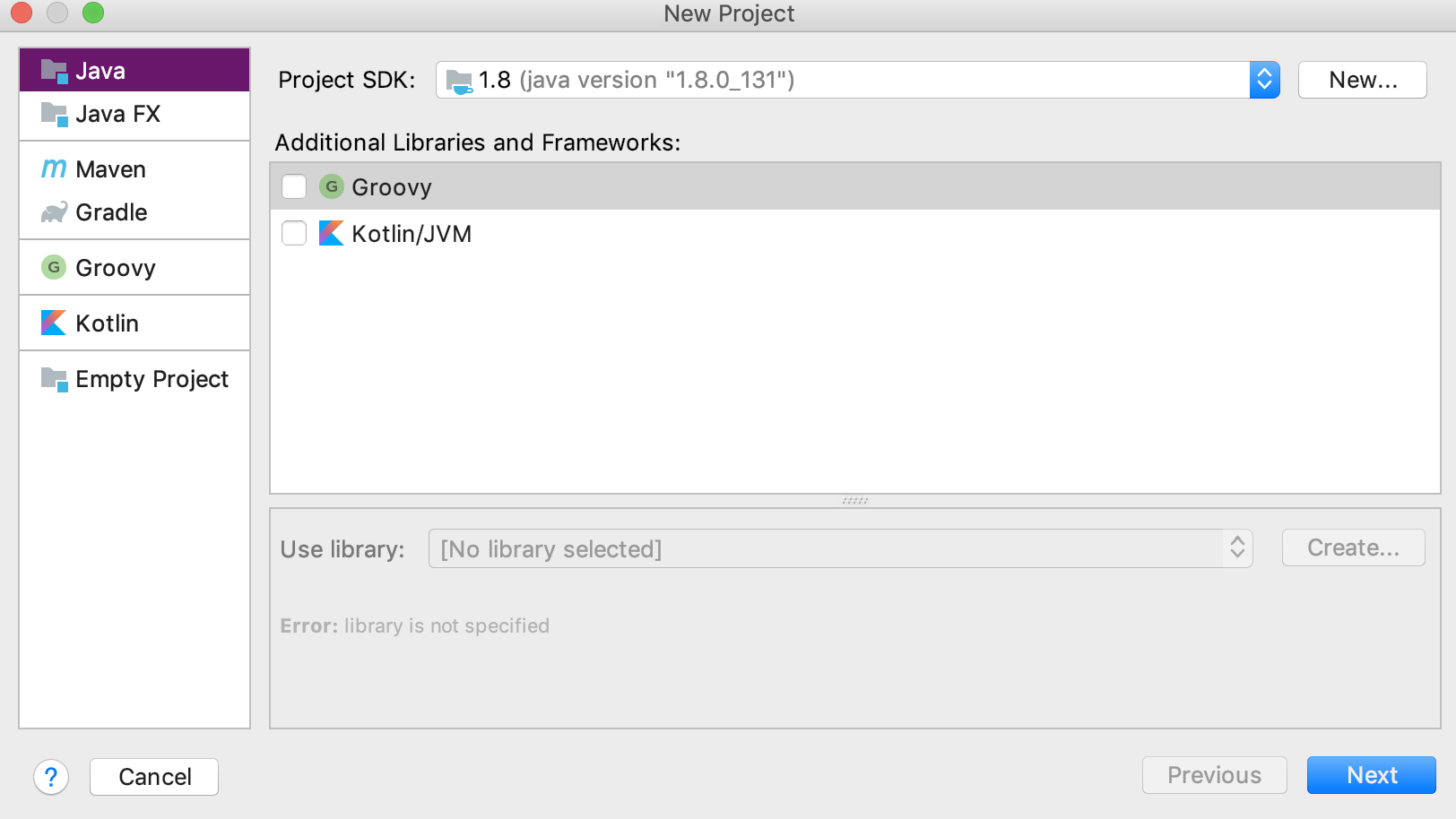
TASK – 1

1.

Se va realiza un proiect nou precum mai jos.

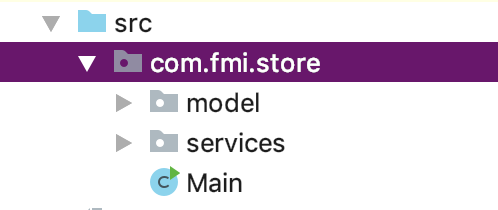


De tipul “Command line app”

Cu numele “store” si base package com.fmi.store

2.

În cadrul lui se vor crea două pachete noi model și service



3.

În model se vor crea

O clasă Category

Având câmpurile private:

String **name**;  
String **description**;  
Category **parent**;

o clasă Price

Având câmpurile private:

Double **price**;  
Date **startFrom**;  
Date **endFrom**;

o clasă Product

Având câmpurile private:

Category **category**;  
Price **price**;  
String **name**;  
String **smallDescription**;

O clasa User

Având câmpurile private:

Integer **id**;

String **username**;  
String **password**;

Ptr fiecare din clasele de mai sus se vor implementa getter-i și setter-i.

4

.

Se va crea o clasă FoodProduct ce moștenește Product

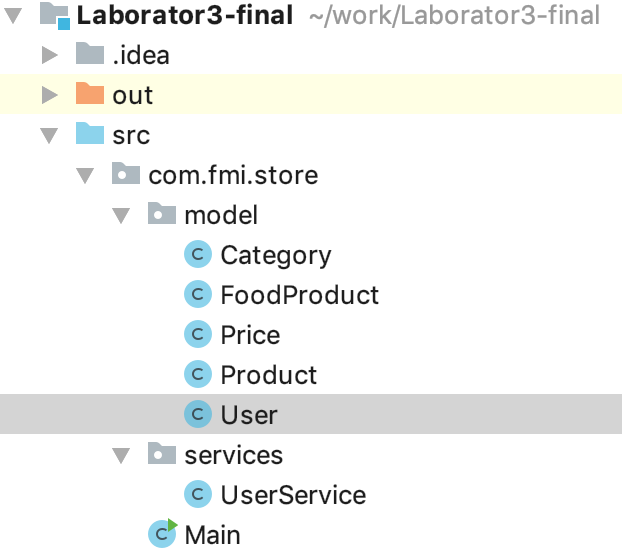
În cadrul ei se va suprascrie metoda getName ca fiind

**return "FoodProduct - name: "** + **super**.getName();

5.

În pachetul services se va crea o clasă de tip singleton UserService. Oricare tip de singleton este ok

Proiectul final ar trebui să arate precum mai jos



6.

Pentru oricare din clasele din model se va implementa (overwritten) o metodă *clone* din clasa părinte Object.

Implementarea ar trebui să arate precum

@Override  
**protected** Object clone() **throws** CloneNotSupportedException {  
 **return super**.clone();  
}

7.

Faceți același lucru pentru câteva din clasele pachetului model.

De asemenea clasele ar trebui să implementeze interfața: Cloneable.

8.

Testati din clasa Main funcționalități precum

Obiectul x.clone

Pentru a evidenția mai bine conținutul obiectului puteți suprascrie metoda toString de o manieră cât mai clară.

TASK -2

Testați în clasa main un obiect Product.

ex.

Product product = **new** Product();  
product.setName(**"produsul unu"**);

Se va instantia un alt obiect FoodProduct

FoodProduct foodProduct = **new** FoodProduct();  
foodProduct.setName(**"produsul doi"**);

Se vor apela metodele getName ale celor două obiecte și compara rezultatele

Se va instanția un obiect de tip Product cu un obiect FoodProduct

ex.

*// in general nu folositi nume de variabila de acest fel, este urat si contraindicat*Product foodProduct\_2 = **new** FoodProduct();  
foodProduct\_2.setName(**"foodProduct\_2"**);

TASK – 3

Se vor adăuga funcționalități pentru clasa UserService precum mai jos

Codul va fi inserat in clasa UserService, tot odată se va renunța la constructorul gol existent in singleton și se va păstra cel de mai jos.

*// vom reveni asupra colectiilor la momentul potrivit***private static** User[] *listOfUsers* = **new** User[4];  
  
  
**private** UserService(){  
 *listOfUsers*[0] = **new** User(1, **"Ioana"**, **"password"**);  
 *listOfUsers*[1] = **new** User(2, **"Adelin"**, **"password"**);  
 *listOfUsers*[2] = **new** User(3, **"Marcu"**, **"password"**);  
 *listOfUsers*[3] = **new** User(4, **"Doru"**, **"password"**);  
  
}

Nu uitați să vă adăugați in clasa User un constructor cu parametrii.

*// metoda getOne overloading***public** User getOne(Integer id) {

//TODO implement - to find user by id

}  
  
**public** User getOne(String username) {  
 // strings are using equals for comparing

//TODO implement - to find user by username

}

Task – 4

Similar se va crea o clasă ProductService în pachetul services