Documentation

Our application has all functional requirements

F1:
Zemědělec nebo Farmář, Zpracovatel, Sklad, Prodejce, Distribuce, Zákazník, Potravina. These classes are located in package *cz.cvut.fel.omo.api* . We have milk, orange and wheat farmers (parties package), Bakery, Storage, Shop, Distributor, Customer and Product.

F2:
Abstract class Operation and extended classes : Creation, PutIntoStorage, TakeFromStorage and Transaction. Located in *cz.cvut.fel.omo.api.operations* . Every operation interact with only one product, and has Party who did this Operation. Also Operation class implements Block interface, so Every Operation is a one block in blockchain. When Transaction Operation is created, money “transfer” is held in Transaction constructor body. OperationFactory (in our system Party) creates Operations.

F3:
BlockChain interface in *cz.cvut.fel.omo.api* and its implementation class BlockChainImpl in *z.cvut.fel.omo.api.impl.* BlockChainImpl class has *List<Operation> chain*. Every Operation is one block, block has method *getMyHash()* that calculate its hash dynamically depends on context and field *String prevBlockHash*. It is impossible to violate and change data in block, because BlockChain secure chain itself every day (method *secure()* in BlocChainImpl), if violation detected BlocChainImpl has *List<Operation> reservChain* with correct data to restore the block.

F4:
Product is implemented by Composite design Pattern *cz.cvut.fel.omo.api.product*. Product has its ProductType (enum in *cz.cvut.fel.omo.api*) It keeps All Product it was created from as *Reportable[] components.* Reportable means each has method *report()* which returns list of all operations provided with this Product and with its components.

F5:
We have Channel interface and ChannelImpl *cz.cvut.fel.omo.api* and *cz.cvut.fel.omo.api.channels.* Channel has participants (Parties) and method *sendRequest* that sends request to all participants. Channel-Party behavior implemented with Observer pattern.

F6:
DoubleSpendingRegulator class is responsible for detecting DoubleSpending problem *cz.cvut.fel.omo.transactions*. It keeps map of Ids of a Party and Products he sold and detects repeatings.

F7:
look F3 (*secure()* method)

F8:
Producing of Product implemented in classes ProductionProcess and “State” classes (*cz.cvut.fel.omo.production*). State design pattern. At first Preparation state go(Party prepare all components to create required amount of Products), after that Waiting state(wates for other Parties), the Producing state(all production logic) and End state.

F9:
Request class in *cz.cvut.fel.omo.transactions.* Request is created by Party and sended to particular channel (*buyProducts(ProductType type, int amount)* method in PartyImpl). Channel is implemented by Observer pattern so Parties can register&unregister.

F10.
Parties report, Food chain report, Security report ,Transaction report all is implemented in EcoSystem class *cz.cvut.fel.omo.api.* If you launch system with *launch()* method in EcoSystem after simulation work you can see any report you want. You will see instructions which program provide.