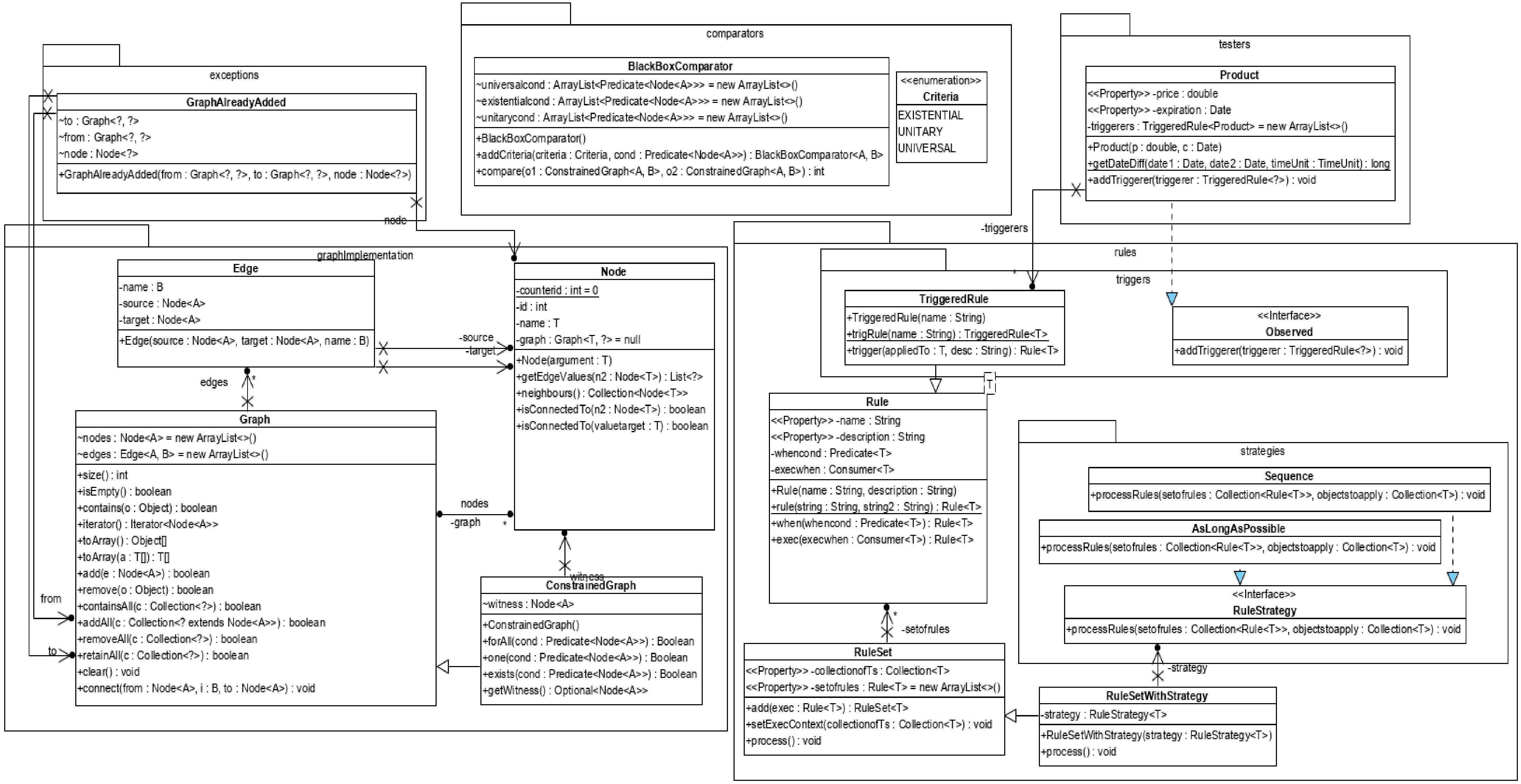
exceptions
GraphAlreadyAdded
~to : Graph<?, ?>
~from : Graph<?, ?>
~node : Node<?>
+GraphAlreadyAdded(from : Graph<?, ?>, to : Graph<?, ?>, node : Node<?>)
node
comparators
BlackBoxComparator
~universalcond : ArrayList<Predicate<Node<A>>> = new ArrayList<>()
~existentialcond : ArrayList<Predicate<Node<A>>> = new ArrayList<>()
~unitarycond : ArrayList<Predicate<Node<A>>> = new ArrayList<>()
+BlackBoxComparator()
+addCriteria(criteria : Criteria, cond : Predicate<Node<A>>) : BlackBoxComparator<A, B>
+compare(o1 : ConstrainedGraph<A, B>, o2 : ConstrainedGraph<A, B>) : int
<<enumeration>>
Criteria
EXISTENTIAL
UNITARY
UNIVERSAL
testers
Product
<<Property>> -price : double
<<Property>> -expiration : Date
-triggerers : TriggeredRule<Product> = new ArrayList<>()
+Product(p : double, c : Date)
+getDateDiff(date1 : Date, date2 : Date, timeUnit : TimeUnit) : long
+addTriggerer(triggerer : TriggeredRule<?>) : void
-triggerers
*
graphImplementation
Edge
-name : B
-source : Node<A>
-target : Node<A>
+Edge(source : Node<A>, target : Node<A>, name : B)
-source
-target
Node
-counterid : int = 0
-id : int
-name : T
-graph : Graph<T, ?> = null
+Node(argument : T)
+getEdgeValues(n2 : Node<T>) : List<?>
+neighbours() : Collection<Node<T>>
+isConnectedTo(n2 : Node<T>) : boolean
+isConnectedTo(valuetarget : T) : boolean
edges
*
Graph
~nodes : Node<A> = new ArrayList<>()
~edges : Edge<A, B> = new ArrayList<>()
+size() : int
+isEmpty() : boolean
+contains(o : Object) : boolean
+iterator() : Iterator<Node<A>>
+toArray() : Object[]
+toArray(a : T[]) : T[]
+add(e : Node<A>) : boolean
+remove(o : Object) : boolean
+containsAll(c : Collection<?>) : boolean
+addAll(c : Collection<? extends Node<A>>) : boolean
+removeAll(c : Collection<?>) : boolean
+retainAll(c : Collection<?>) : boolean
+clear() : void
+connect(from : Node<A>, i : B, to : Node<A>) : void
from
to
nodes
-graph
*
witness
ConstrainedGraph
~witness : Node<A>
+ConstrainedGraph()
+forAll(cond : Predicate<Node<A>>) : Boolean
+one(cond : Predicate<Node<A>>) : Boolean
+exists(cond : Predicate<Node<A>>) : Boolean
+getWitness() : Optional<Node<A>>
rules
triggers
TriggeredRule
+TriggeredRule(name : String)
+trigRule(name : String) : TriggeredRule<T>
+trigger(appliedTo : T, desc : String) : Rule<T>
<<Interface>>
Observed
+addTriggerer(triggerer : TriggeredRule<?>) : void
T
Rule
<<Property>> -name : String
<<Property>> -description : String
-whencond : Predicate<T>
-execwhen : Consumer<T>
+Rule(name : String, description : String)
+rule(string : String, string2 : String) : Rule<T>
+when(whencond : Predicate<T>) : Rule<T>
+exec(execwhen : Consumer<T>) : Rule<T>
*
-setofrules
RuleSet
<<Property>> -collectionofTs : Collection<T>
<<Property>> -setofrules : Rule<T> = new ArrayList<>()
+add(exec : Rule<T>) : RuleSet<T>
+setExecContext(collectionofTs : Collection<T>) : void
+process() : void
strategies
Sequence
+processRules(setofrules : Collection<Rule<T>>, objectstoapply : Collection<T>) : void
AsLongAsPossible
+processRules(setofrules : Collection<Rule<T>>, objectstoapply : Collection<T>) : void
<<Interface>>
RuleStrategy
+processRules(setofrules : Collection<Rule<T>>, objectstoapply : Collection<T>) : void
-strategy
RuleSetWithStrategy
-strategy : RuleStrategy<T>
+RuleSetWithStrategy(strategy : RuleStrategy<T>)
+process() : void

Implementation of Graph

3 classes where made to implement the functionality, Node and Graph, required by the main, and Edge. Nodes store the value and the graph they belong to in order to be able to call connection methods without explicitly calling the graph. Edges are stored in Graph and the connections functions of the latter are done for near nodes. An exception was done if you try to add a graph to a node that already belongs to a previous one.

BBC and ConstrainedGraph

ConstrainedGraph was extended from Graph and added methods to apply predicates. The BBC was implemented through the interface Comparator and differentiated between the existential, unitary and universal criteria.

Rule, RuleSet and RuleSetStrategy

Rules are done using predicates and consumers, the ruleset is just a collection of rules applied to a collection of generic types and the strategy passes that implementation to the strategy.

For strategies we made an interface that specified the processRules method. We made the AsLongAsPossible and the sequence strategy was only the one previously used in the stock RuleSet.