ASSIGNMENT 5 REPORT

Álvaro Castillo García

Pablo Ernesto Soëtard

Andrés Martín Manos

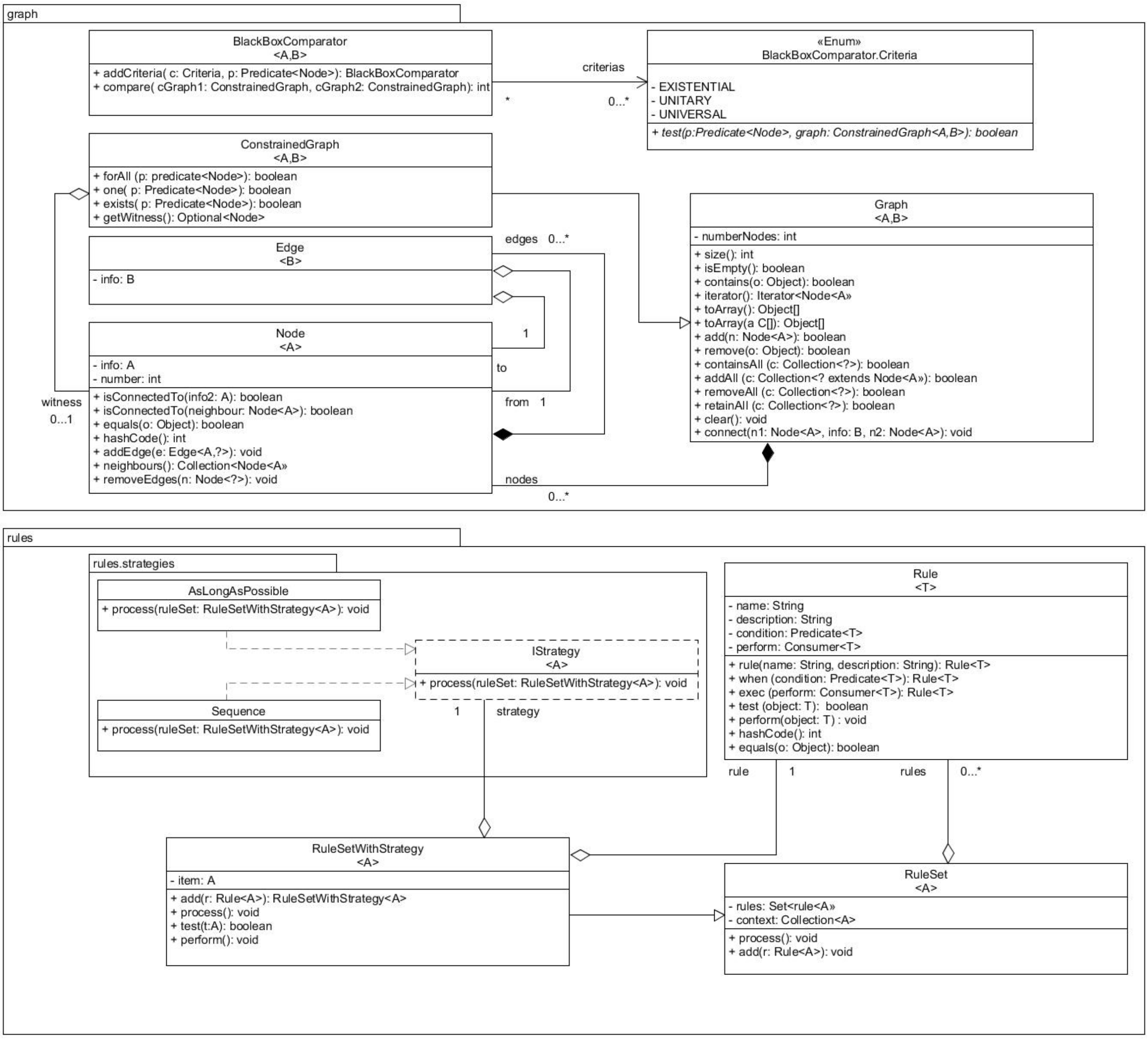
Universidad Autónoma de Madrid

Escuela Politécnica Superior

Ingeniería Informática bilingüe, grupo 2292

Software Analysis and Design

CLASS DIAGRAM


graph
BlackBoxComparator
<A,B>
+ addCriteria(c: Criteria, p: Predicate<Node>): BlackBoxComparator
+ compare(cGraph1: ConstrainedGraph, cGraph2: ConstrainedGraph): int
criterias
*
0...*
«Enum»
BlackBoxComparator.Criteria
- EXISTENTIAL
- UNITARY
- UNIVERSAL
+ test(p:Predicate<Node>, graph: ConstrainedGraph<A,B>): boolean
ConstrainedGraph
<A,B>
+ forAll (p: predicate<Node>): boolean
+ one(p: Predicate<Node>): boolean
+ exists(p: Predicate<Node>): boolean
+ getWitness(): Optional<Node>
Graph
<A,B>
- numberNodes: int
+ size(): int
+ isEmpty(): boolean
+ contains(o: Object): boolean
+ iterator(): Iterator<Node<A»
+ toArray(): Object[]
+ toArray(a C[]): Object[]
+ add(n: 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(n1: Node<A>, info: B, n2: Node<A>): void
edges 0...*
Edge
<B>
- info: B
1
Node
<A>
- info: A
- number: int
+ isConnectedTo(info2: A): boolean
+ isConnectedTo(neighbour: Node<A>): boolean
+ equals(o: Object): boolean
+ hashCode(): int
+ addEdge(e: Edge<A,?>): void
+ neighbours(): Collection<Node<A»
+ removeEdges(n: Node<?>): void
to
from 1
witness
0...1
nodes
0...*
rules
rules.strategies
AsLongAsPossible
+ process(ruleSet: RuleSetWithStrategy<A>): void
IStrategy
<A>
+ process(ruleSet: RuleSetWithStrategy<A>): void
Sequence
+ process(ruleSet: RuleSetWithStrategy<A>): void
1
strategy
Rule
<T>
- name: String
- description: String
- condition: Predicate<T>
- perform: Consumer<T>
+ rule(name: String, description: String): Rule<T>
+ when (condition: Predicate<T>): Rule<T>
+ exec (perform: Consumer<T>): Rule<T>
+ test (object: T): boolean
+ perform(object: T) : void
+ hashCode(): int
+ equals(o: Object): boolean
rule
1
rules
0...*
RuleSetWithStrategy
<A>
- item: A
+ add(r: Rule<A>): RuleSetWithStrategy<A>
+ process(): void
+ test(t:A): boolean
+ perform(): void
RuleSet
<A>
- rules: Set<rule<A»
- context: Collection<A>
+ process(): void
+ add(r: Rule<A>): void

In the above image, the class diagram of this assignment is shown. While developing the code we have faced many problems, for example, when we were implementing the BlackBoxComparator, we did not know how to implement the Criteria enum with its functions. After reading the theory, we realized that what we needed was internal classes and anonymous functions.

Other big problem that we had was the missunderstanding of the strategy “AsLongAsPossible”, but again, after reviewing the lambda expressions theory slides, we came across a function that does just what we wanted, the “anyMatch”, that allowed us to successfully implement that part of the assignment.

OPTIONAL PART

The pattern that we have used is the Observer. Product.java is the observable object, that will call notifyObservers() when the price of a product is changed (setPrice() is called). On the other hand, TriggeredRule is the observer object. It implements a method update(), that will recieve the object passed by notify() and will perform some actions.