# ANEXO

## Questão 1.4

```
public interface Dao<T, K> {
    T get(K key);
    Collection<T> getAll();
    void save(T instance) throws DaoException;
    void update(T instance) throws DaoException;
    T delete(K key);
    int count();
}
```

## Questão 1.6

```
public class A {
    private B b;
    private C c;

    public A(B b, C c) {
        this.b = b;
        this.c = c;
        b.addObservers(c);
    }

    public void execute() {
        char op;
        do {
            c.printMenu();
            op = c.readInput();
            switch (op) {
                case 'I':
                    b.increment(); break;
                case 'D':
                    b.decrement();break;
                case 'Q':
                    c.outputMessage("ADEUS");break;
                default:
                    c.error("incorrect option");
            }

        } while (op != 'Q');
    }

}

public class B extends Subject {

    private int value, max;

    public B(int max) {
        value=0;
        this.max=max;
    }

    public void increment() {
        value=(value+1)%max;
        notifyObservers(this);
    }

    public void decrement() {
```

```
        value=(max+(value-1))%max;
        notifyObservers(this);
    }

    @Override
    public String toString() {
        return value + "";
    }

}


public class C implements Observer {
    private static Scanner sc= new Scanner(System.in);
    public C(B b) {
        update(b);
    }

 public void printMenu() {
        System.out.println("menu");
        System.out.println("I - Incrementa");
        System.out.println("D - Decrementa");
        System.out.println("Q - Quit");
        System.out.println("Introduz a opção >");
    }

    public char readInput() {
        return sc.next().charAt(0);
    }

    @Override
    public void update(Object obj) {
        B b = (B) obj;
        System.out.println("----------------------");
        System.out.printf("---------(%s)----------\n\n", b);
    }

    public void error(String str) {
        System.out.println("ERROR " + str);
    }

    public void outputMessage(String str) {
        System.out.println(str);
    }
}
```

## Questão 2.3

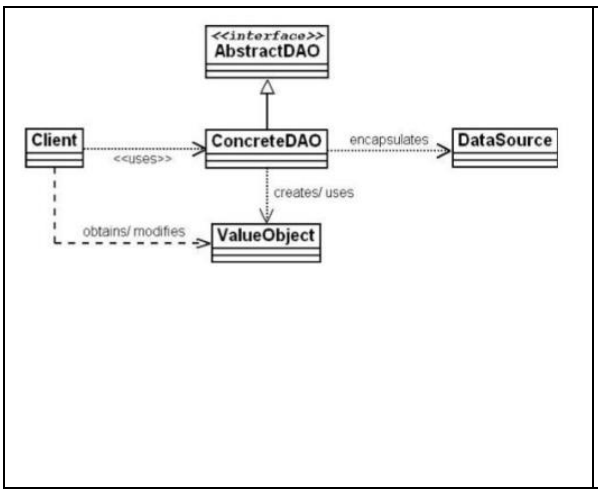
```
public class MapBST<K extends Comparable<K>, V> implements Map<K,V> {

    private BSTNode root;

    public MapBST() {
        this.root = null;
    }

    //other methods
    private class BSTNode {
        private K key;
        private V value;

        private BSTNode parent;
        private BSTNode left;
        private BSTNode right;
```

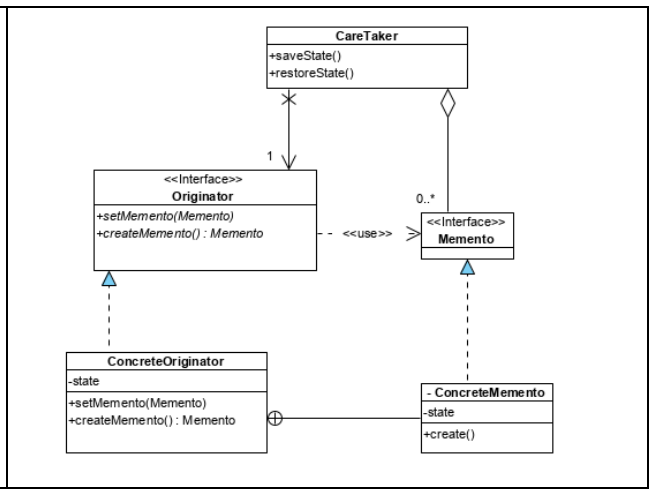
```
        public BSTNode(K key, V value, BSTNode parent, BSTNode left, BSTNode right) {
            this.key = key;
            this.value = value;
            this.parent = parent;
            this.left = left;
            this.right = right;
        }
    }
```

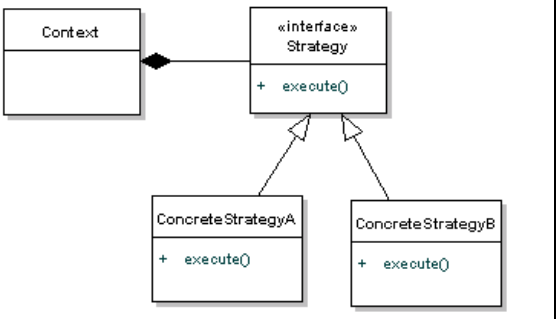
## Questão 2.4

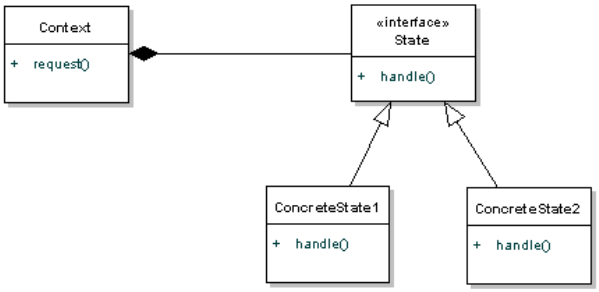
```
public class GraphAdjacencyList<V,E> implements Graph<V,E> {
    private List<MyVertex> vertices;
    public GraphAdjacencyList() { vertices = new ArrayList<>(); }

    public Edge<E,V> insertEdge<Vertex<V> v1, Vertex<V> v2, E element) {
        //Nota: ignoradas as validações
        MyVertex u = (MyVertex)v1;
        MyVertex v = (MyVertex)v2;
        MyEdge edge = new MyEdge(); edge.element = element;
        u.incident.add(edge);
        v.incident.add(edge);
        return edge;
    }
    //...
    private class MyVertex implements Vertex<V> {
        V element;
        List<MyEdge> incident;
        //...
    }
    private class MyEdge implements Edge<E,V> {
        E element;
    }
}
```

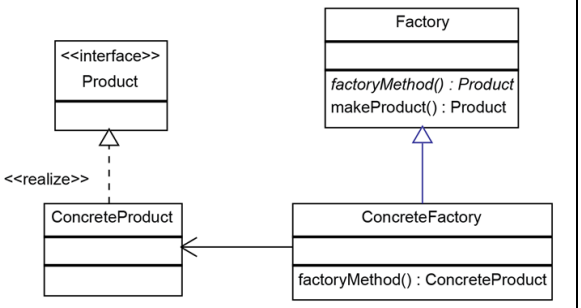
Patterns

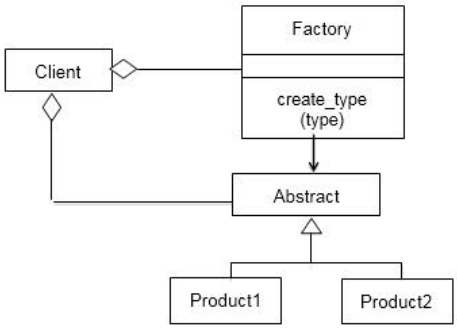


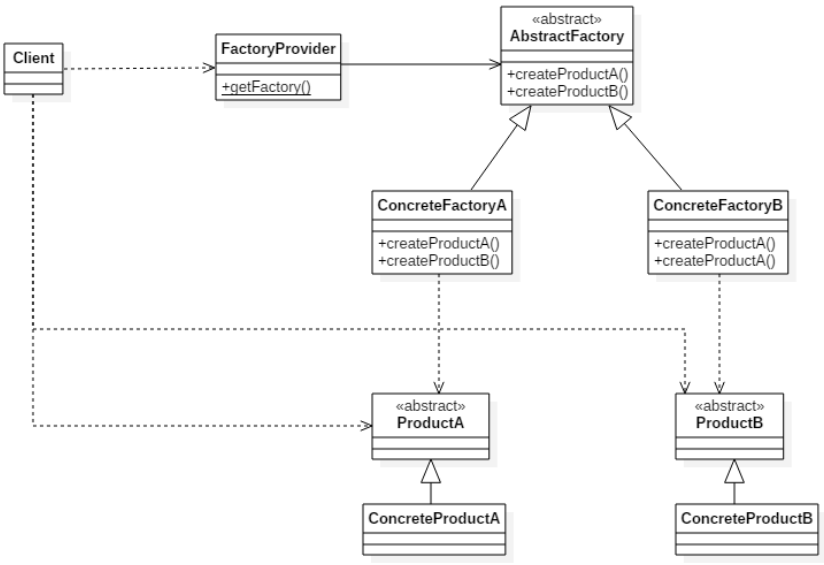



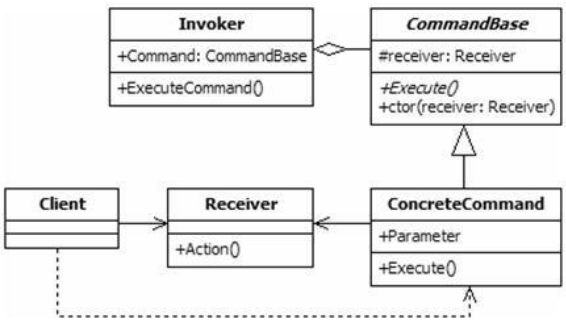

Context
«interface»
Strategy
+ execute()
ConcreteStrategyA
+ execute()
ConcreteStrategyB
+ execute()

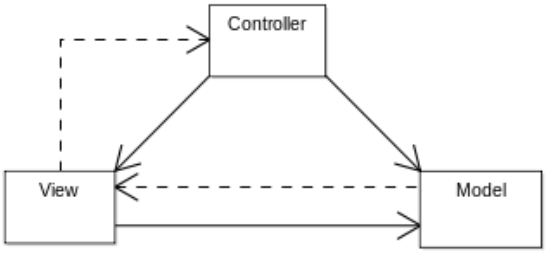

Context
+ request()
«interface»
State
+ handle()
ConcreteState1
+ handle()
ConcreteState2
+ handle()


Factory
<<interface>>
Product
factoryMethod() : Product
makeProduct() : Product
<<realize>>
ConcreteProduct
ConcreteFactory
factoryMethod() : ConcreteProduct


Client
Factory
create_type
(type)
Abstract
Product1
Product2


Client
FactoryProvider
+getFactory()
«abstract»
AbstractFactory
+createProductA()
+createProductB()
ConcreteFactoryA
+createProductA()
+createProductB()
ConcreteFactoryB
+createProductA()
+createProductA()
«abstract»
ProductA
«abstract»
ProductB
ConcreteProductA
ConcreteProductB


Invoker
+Command: CommandBase
+ExecuteCommand()
CommandBase
#receiver: Receiver
+Execute()
+ctor(receiver: Receiver)
Client
Receiver
+Action()
ConcreteCommand
+Parameter
+Execute()


Controller
View
Model