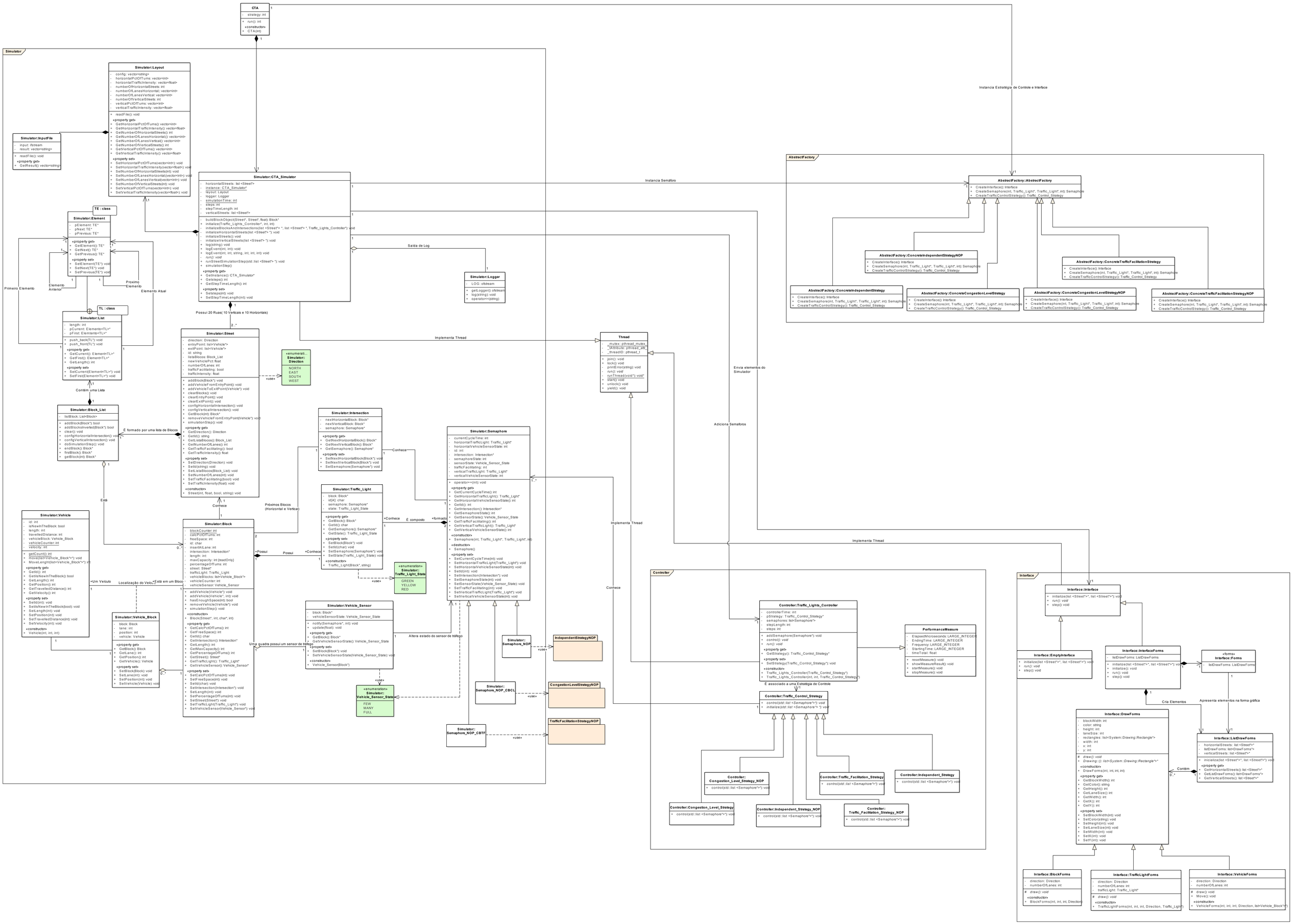
CTA
- strategy: int
+ run(): int
«constructor»
+ CTA(int)
Simulator
Simulator::Layout
- config: vector<string>
- horizontalPctOfTurns: vector<int>
- horizontalTrafficIntensity: vector<float>
- numberOfHorizontalStreets: int
- numberOfLanesHorizontal: vector<int>
- numberOfLanesVertical: vector<int>
- numberOfVerticalStreets: int
- verticalPctOfTurns: vector<int>
- verticalTrafficIntensity: vector<float>
+ readFile(): void
«property get»
+ GetHorizontalPctOfTurns(): vector<int>
+ GetHorizontalTrafficIntensity(): vector<float>
+ GetNumberOfHorizontalStreets(): int
+ GetNumberOfLanesHorizontal(): vector<int>
+ GetNumberOfLanesVertical(): vector<int>
+ GetNumberOfVerticalStreets(): int
+ GetVerticalPctOfTurns(): vector<int>
+ GetVerticalTrafficIntensity(): vector<float>
«property set»
+ SetHorizontalPctOfTurns(vector<int>): void
+ SetHorizontalTrafficIntensity(vector<float>): void
+ SetNumberOfHorizontalStreets(int): void
+ SetNumberOfLanesHorizontal(vector<int>): void
+ SetNumberOfLanesVertical(vector<int>): void
+ SetNumberOfVerticalStreets(int): void
+ SetVerticalPctOfTurns(vector<int>): void
+ SetVerticalTrafficIntensity(vector<float>): void
Simulator::InputFile
- input: ifstream
- result: vector<string>
+ readFile(): void
«property get»
+ GetResult(): vector<string>
Simulator::CTA_Simulator
- horizontalStreets: list <Street>
- instance: CTA_Simulator*
- layout: Layout
- logger: Logger
- simulationTime: int
- steps: int
- stepTimeLength: int
- verticalStreets: list <Street>
- buildBlockObject(Street*, Street*, float): Block*
+ initialize(Traffic_Lights_Controller*, int, int)
- initializeBlocksAndIntersections(list <Street*> *, list <Street*> *, Traffic_Lights_Controller*): void
- initializeHorizontalStreets(list <Street*> *): void
- initializeStreets(): void
- initializeVerticalStreets(list <Street*> *): void
+ log(string): void
+ logEvent(int, int): void
+ logEvent(int, int, string, int, int, int): void
- run(): void
+ runStreetSimulationStep(std::list <Street*> *): void
+ simulationStep()
«property get»
+ GetInstance(): CTA_Simulator*
+ GetSteps(): int
+ GetStepTimeLength(): int
«property set»
+ SetSteps(int): void
+ SetStepTimeLength(int): void
Simulator::Logger
- LOG: ofstream
+ getLogger(): ofstream
+ log(string): void
+ operator<<(string)
Simulator::Element
- pElement: TE*
- pNext: TE*
- pPrevious: TE*
«property get»
+ GetElement(): TE*
+ GetNext(): TE*
+ GetPrevious(): TE*
«property set»
+ SetElement(TE*): void
+ SetNext(TE*): void
+ SetPrevious(TE*): void
TE : class
TL : class
Primeiro Elemento
Elemento Anterior
Proximo Elemento
Elemento Atual
Simulator::List
- length: int
- pCurrent: Elemento<TL>*
- pFirst: Elemtento<TL>*
+ push_back(TL*): void
+ push_front(TL*): void
«property get»
+ GetCurrent(): Element<TL>*
+ GetFirst(): Element<TL>*
+ GetLength(): int
«property set»
+ SetCurrent(Element<TL>*): void
+ SetFirst(Element<TL>*): void
Contém uma Lista
Simulator::Block_List
- listBlock: ListBlock>
+ addBlock(Block*): bool
+ addBlocksInverted(Block*): bool
+ clear(): void
+ configHorizontalIntersection(): void
+ configVerticalIntersection(): void
+ doSimulationStep(): void
+ endBlock(): Block*
+ firstBlock(): Block*
+ getBlock(int): Block*
É formado por uma lista de Blocos
Está
Possui 20 Ruas(10 Verticais e 10 Horizontais)
2..*
Simulator::Street
- direction: Direction
- entryPoint: list<Vehicle*>
- exitPoint: list<Vehicle*>
- id: string
- listaBlocos: Block_List
- newVehiclePct: float
- numberOfLanes: int
- trafficFacilitating: bool
- trafficIntensity: float
+ addBlock(Block*): void
+ addVehicleFromEntryPoint(): void
+ addVehicleToExitPoint(Vehicle*): void
+ clearBlocks(): void
+ clearEntryPoint(): void
+ clearExitPoint(): void
+ configHorizontalIntersection(): void
+ configVerticalIntersection(): void
+ GetBlock(int): Block*
+ removeVehicleFromEntryPoint(Vehicle*): void
+ simulationStep(): void
«property get»
+ GetDirection(): Direction
+ GetId(): string
+ GetListaBlocos(): Block_List
+ GetNumberOfLanes(): int
+ GetTrafficFacilitating(): bool
+ GetTrafficIntensity(): float
«property set»
+ SetDirection(Direction): void
+ SetId(string): void
+ SetListaBlocos(Block_List): void
+ SetNumberOfLanes(int): void
+ SetTrafficFacilitating(bool): void
+ SetTrafficIntensity(float): void
«constructor»
+ Street(int, float, bool, string): void
«use»
«enumerati...
Simulator::
Direction
NORTH
EAST
SOUTH
WEST
Conhece
Próximos Blocos
(Horizontal e Vertical)
Simulator::Intersection
- nextHorizontalBlock: Block*
- nextVerticalBlock: Block*
- semaphore: Semaphore*
«property get»
+ GetNextHorizontalBlock(): Block*
+ GetNextVerticalBlock(): Block*
+ GetSemaphore(): Semaphore*
«property set»
+ SetNextHorizontalBlock(Block*): void
+ SetNextVerticalBlock(Block*): void
+ SetSemaphore(Semaphore*): void
Conhece
Simulator::Semaphore
- currentCycleTime: int
- horizontalTrafficLight: Traffic_Light*
- horizontalVehicleSensorState: int
- id: int
- intersection: Intersection*
- semaphoreState: int
- sensorState: Vehicle_Sensor_State
- trafficFacilitating: int
- verticalTrafficLight: Traffic_Light*
- verticalVehicleSensorState: int
+ operator++(int): void
«property get»
+ GetCurrentCycleTime(): int
+ GetHorizontalTrafficLight(): Traffic_Light*
+ GetHorizontalVehicleSensorState(): int
+ GetId(): int
+ GetIntersection(): Intersection*
+ GetSemaphoreState(): int
+ GetSensorState(): Vehicle_Sensor_State
+ GetTrafficFacilitating(): int
+ GetVerticalTrafficLight(): Traffic_Light*
+ GetVerticalVehicleSensorState(): int
«constructor»
+ Semaphore(int, Traffic_Light*, Traffic_Light*, int)
«destructor»
+ Semaphore()
«property set»
+ SetCurrentCycleTime(int): void
+ SetHorizontalTrafficLight(Traffic_Light*): void
+ SetHorizontalVehicleSensorState(int): void
+ SetId(int): void
+ SetIntersection(Intersection*): void
+ SetSemaphoreState(int): void
+ SetSensorState(Vehicle_Sensor_State): void
+ SetTrafficFacilitating(int): void
+ SetVerticalTrafficLight(Traffic_Light*): void
+ SetVerticalVehicleSensorState(int): void
1..*
Simulator::Traffic_Light
- block: Block*
- id[4]: char
- semaphore: Semaphore*
- state: Traffic_Light_State
«property get»
+ GetBlock(): Block*
+ GetId(): char
+ GetSemaphore(): Semaphore*
+ GetState(): Traffic_Light_State
«property set»
+ SetBlock(Block*): void
+ SetId(char): void
+ SetSemaphore(Semaphore*): void
+ SetState(Traffic_Light_State): void
«constructor»
+ Traffic_Light(Block*, string)
+Conhece
É composto
+formado
«enumeration»
Simulator::
Traffic_Light_State
GREEN
YELLOW
RED
«use»
Simulator::Vehicle
- id: int
- isNewInTheBlock: bool
- length: int
- travelledDistance: int
- vehicleBlock: Vehicle_Block
- vehicleCounter: int
- velocity: int
+ getCount(): int
+ moveList(list<Vehicle_Block*>*): void
+ MoveLenght(list<Vehicle_Block*>*): int
«property get»
+ GetId(): int
+ GetIsNewInTheBlock(): bool
+ GetLength(): int
+ GetPosition(): int
+ GetTravelledDistance(): int
+ GetVelocity(): int
«property set»
+ SetId(int): void
+ SetIsNewInTheBlock(bool): void
+ SetLength(int): void
+ SetPosition(int): void
+ SetTravelledDistance(int): void
+ SetVelocity(int): void
«constructor»
+ Vehicle(int, int, int)
+Um Veiculo
Localização do Veic...
+Está em um Bloco
Simulator::Block
- blockCounter: int
- calcPctOfTurns: int
- freeSpace: int
- id: char
- insertAtLane: int
- intersection: Intersection*
- length: int
- maxCapacity: int [readOnly]
- percentageOfTurns: int
- street: Street*
- trafficLight: Traffic_Light
- vehicleBlocks: listVehicle_Block*>
- vehicleCounter: int
- vehicleSensor: Vehicle_Sensor
+ addVehicle(Vehicle*): void
+ addVehicle(Vehicle*, int): void
+ hasEnoughSpace(int): bool
+ removeVehicle(Vehicle*): void
+ simulationStep(): void
«constructor»
+ Block(Street*, int, char*, int)
«property get»
+ GetCalcPctOfTurns():int
+ GetFreeSpace(): int
+ GetId(): char
+ GetIntersection(): Intersection*
+ GetLength(): int
+ GetMaxCapacity(): int
+ GetPercentageOfTurns(): int
+ GetStreet(): Street*
+ GetTrafficLight(): Traffic_Light*
+ GetVehicleSensor(): Vehicle_Sensor*
«property set»
+ SetCalcPctOfTurns(int): void
+ SetFreeSpace(int): void
+ SetId(char): void
+ SetIntersection(Intersection*): void
+ SetLength(int): void
+ SetPercentageOfTurns(int): void
+ SetStreet(Street*): void
+ SetTrafficLight(Traffic_Light*): void
+ SetVehicleSensor(Vehicle_Sensor*): void
+Possui
Possui
+Conhece
Simulator::Vehicle_Block
- block: Block
- lane: int
- position: int
- vehicle: Vehicle
«property get»
+ GetBlock(): Block
+ GetLane(): int
+ GetPosition(): int
+ GetVehicle(): Vehicle
«property set»
+ SetBlock(Block): void
+ SetLane(int): void
+ SetPosition(int): void
+ SetVehicle(Vehicle): void
0..*
Uma quadra possui um sensor de tráfego
Simulator::Vehicle_Sensor
- block: Block*
- vehicleSensorState: Vehicle_Sensor_State
+ notify(Semaphore*, int): void
+ update(float): void
«property get»
+ GetBlock(): Block*
+ GetVehicleSensorState(): Vehicle_Sensor_State
«property set»
+ SetBlock(Block*): void
+ SetVehicleSensorState(Vehicle_Sensor_State): void
«constructor»
+ Vehicle_Sensor(Block*)
Altera estado do sensor de tráfego
«use»
«enumeration»
Simulator::
Vehicle_Sensor_State
FEW
MANY
FULL
Simulator::
Semaphore_NOP
Simulator::
Semaphore_NOP_CBCL
Simulator::
Semaphore_NOP_CBTF
IndependentStrategyNOP
CongestionLevelStrategyNOP
TrafficFacilitationStrategyNOP
Implementa Thread
Thread
- _mutex: pthread_mutex_
- _IAttribute: pthread_attr_
- _threadID: pthread_t
+ join(): void
+ lock(): void
- printError(string): void
- run(): void
- runThread(void*): void*
+ start(): void
+ unlock(): void
+ yield(): void
Envia elementos do
Simulador
Adiciona Semaforos
Implementa Thread
Conhece
Instancia Semáforo
Instancia Estratégia de Controle e Interface
Saída de Log
AbstractFactory
AbstractFactory::AbstractFactory
+ CreateInterface(): Interface
+ CreateSemaphore(int, Traffic_Light*, Traffic_Light*, int): Semaphore
+ CreateTrafficControlStrategy(): Traffic_Control_Strategy
AbstractFactory::ConcreteIndependentStrategyNOP
+ CreateInterface(): Interface
+ CreateSemaphore(int, Traffic_Light*, Traffic_Light*, int): Semaphore
+ CreateTrafficControlStrategy(): Traffic_Control_Strategy
AbstractFactory::ConcreteTrafficFacilitationStrategy
+ CreateInterface(): Interface
+ CreateSemaphore(int, Traffic_Light*, Traffic_Light*, int): Semaphore
+ CreateTrafficControlStrategy(): Traffic_Control_Strategy
AbstractFactory::ConcreteIndependentStrategy
+ CreateInterface(): Interface
+ CreateSemaphore(int, Traffic_Light*, Traffic_Light*, int): Semaphore
+ CreateTrafficControlStrategy(): Traffic_Control_Strategy
AbstractFactory::ConcreteCongestionLevelStrategy
+ CreateInterface(): Interface
+ CreateSemaphore(int, Traffic_Light*, Traffic_Light*, int): Semaphore
+ CreateTrafficControlStrategy(): Traffic_Control_Strategy
AbstractFactory::ConcreteCongestionLevelStrategyNOP
+ CreateInterface(): Interface
+ CreateSemaphore(int, Traffic_Light*, Traffic_Light*, int): Semaphore
+ CreateTrafficControlStrategy(): Traffic_Control_Strategy
AbstractFactory::ConcreteTrafficFacilitationStrategyNOP
+ CreateInterface(): Interface
+ CreateSemaphore(int, Traffic_Light*, Traffic_Light*, int): Semaphore
+ CreateTrafficControlStrategy(): Traffic_Control_Strategy
Controller
Controller::Traffic_Lights_Controller
- controllerTime: int
- pStrategy: Traffic_Control_Strategy*
- semaphores: listSemaphore*>
- stepLength: int
- steps: int
+ addSemaphore(Semaphore*): void
+ control(): void
- run(): void
«property get»
+ GetStrategy(): Traffic_Control_Strategy*
«property set»
+ SetStrategy(Traffic_Control_Strategy*): void
«constructor»
+ Traffic_Lights_Controller(Traffic_Control_Strategy*)
+ Traffic_Lights_Controller(int, int, Traffic_Control_Strategy*)
PerformanceMeasure
- ElapsedMicroseconds: LARGE_INTEGER
- EndingTime: LARGE_INTEGER
- Frequency: LARGE_INTEGER
- StartingTime: LARGE_INTEGER
- timeTotal: float
+ resetMeasure(): void
+ showMeasureResult(): void
+ startMeasure(): void
+ stopMeasure(): void
É associado a uma Estratégia de Controle
Controller::Traffic_Control_Strategy
+ control(std::list <Semaphore*>*): void
+ initialize(std::list <Semaphore*> *): void
Controller::
Congestion_Level_Strategy_NOP
+ control(std::list <Semaphore*>*): void
Controller::Congestion_Level_Strategy
+ control(std::list <Semaphore*>*): void
Controller::Traffic_Facilitation_Strategy
+ control(std::list <Semaphore*>*): void
Controller::Independent_Strategy
+ control(std::list <Semaphore*>*): void
Controller::Independent_Strategy_NOP
+ control(std::list <Semaphore*>*): void
Controller::
Traffic_Facilitation_Strategy_NOP
+ control(std::list <Semaphore*>*): void
Interface
Interface::Interface
+ initialize(list <Street*>*, list <Street*>*): void
+ run(): void
+ step(): void
Interface::EmpyInterface
+ initialize(list <Street*>*, list <Street*>*): void
+ run(): void
+ step(): void
Interface::InterfaceForms
- listDrawForms: ListDrawForms
+ initialize(list <Street*>*, list <Street*>*): void
+ initialize(): void
+ run(): void
+ step(): void
«forms»
Interface::Forms
- listDrawForms: ListDrawForms
Cria Elementos
Apresenta elementos na forma gráfica
Interface::DrawForms
- blockWidth: int
- color: string
- height: int
- laneSize: int
- rectangles: list<System::Drawing::Rectangle*>
- width: int
- x: int
- y: int
draw(): void
+ Drawing::(): list<System::Drawing::Rectangle*>*
«constructor»
+ DrawForms(int, int, int, int)
«property get»
+ GetBlockWidth(): int
+ GetColor(): string
+ GetHeight(): int
+ GetLaneSize(): int
+ GetWidth(): int
+ GetX(): int
+ GetY(): int
«property set»
+ SetBlockWidth(int): void
+ SetColor(string): void
+ SetHeight(int): void
+ SetLaneSize(int): void
+ SetWidth(int): void
+ SetX(int): void
+ SetY(int): void
Interface::ListDrawForms
- horizontalStreets: list <Street*>
- listDrawForms: listDrawForms*>
- verticalStreets: list <Street*>
+ inicializa(list <Street*>*, list <Street*>*): void
«property get»
+ GetHorizontalStreets(): list <Street*>
+ GetListDrawForms(): listDrawForms*>
+ GetVerticalStreets(): list <Street*>
Contém
0..*
Interface::BlockForms
- direction: Direction
- numberOfLanes: int
draw(): void
«constructor»
+ BlockForms(int, int, int, Direction)
Interface::TrafficLightForms
- direction: Direction
- numberOfLanes: int
- trafficLight: Traffic_Light*
draw(): void
«constructor»
+ TrafficLightForms(int, int, int, Direction, Traffic_Light*)
Interface::VehicleForms
- direction: Direction
- numberOfLanes: int
draw(): void
+ Move(): void
«constructor»
+ VehicleForms(int, int, int, Direction, list<Vehicle_Block*>*)