Simulator hereda los atributos y métodos de AbstractSimulator

Por lo tanto, BeepSimulator hereda los atributos y métodos de Abstract Simulator.

BeepSimulator hereda los atributos y métodos de Simulator

```python
class AbstractSimulator(object):
    def __init__(self):
        self.events = None # Insert an abstract event
    def insert(self, e):
        self.events.insert(e) #AbstractEvent
    def cancel(self, e):
        raise NotImplementedError("Method not implemented")

class Simulator(AbstractSimulator):
    def __init__(self):
        super().__init__()
        self.time = None
    def now(self):
        return self.time
    def doAllEvents(self):
        while self.events.size() > 0:
            e = self.events.removeFirst()
            self.time = e.time
            e.execute(self)

class BeepSimulator(Simulator):
    def __init__(self):
        super().__init__()
    def start(self):
        self.events = ListQueue()
        self.insert(Beep(4.0))
        self.insert(Beep(1.0))
        self.insert(Beep(1.5))
        self.insert(Beep(2.0))

        self.doAllEvents()
```

(*)

BeepSimulator usa el atributo events y el método insert() y

(*) Ejecuta todos los eventos de la cola (ListQueue()) ordenados por tiempo

Señala que heredará los atributos de Simulator

cola vacía

decorator

```python
from abc import ABC, abstractmethod

class AbstractEvent(ABC):
    @abstractmethod
    def execute(simulator): #Abstract Simulator
        pass

class OrderedSet(ABC):
    @abstractmethod
    def insert(x):
        pass
    @abstractmethod
    def removeFirst():
        pass
    @abstractmethod
    def size():
        pass
    @abstractmethod
    def remove(x):
        pass
```

clase abstracta

métodos abstractos

Implica que todo evento debe tener sí o sí un método execute

Sobre escrita

ListQueue hereda a OrderSet

↳ Obliga a que tenga los 4 métodos!

```python
class ListQueue(OrderedSet):
    def __init__(self):
        self.elements = list()
    def insert(self, x):
        i = 0
        while i < len(self.elements) and self.elements[i] < x:
            i += 1
        self.elements.insert(i, x)
    def removeFirst(self):
        if len(self.elements) == 0:
            return None
        x = self.elements.pop(0)
        return x
    def remove(self, x):
        for i in range(len(self.elements)):
            if self.elements[i] == x:
                return self.elements.pop(i)
        return None
    def size(self):
        return len(self.elements)
```

herencia a Event

Beep hereda a Event

```python
class Event(AbstractSimulator, ABC):
    def __init__(self):
        self.time = None
    def __lt__(self, y):
        if isinstance(y, Event):
            return self.time < y.time
        else:
            raise ValueError('This is no an event')
    def __eq__(self, other):
        return self.__dict__ == other.__dict__
```

```python
class Beep(Event):
    def __init__(self, time):
        super().__init__()
        self.time = time
    def execute(self, simulator):
        print('this is time ', simulator.time)
```