CHAPTER 1

Finding the North with Compass

In this chapter, we will work on an alternative way to represent directions and move computation in the 2D plan.

1.1 Existing situation

Computing new position based on a direction.

In the Robot implementation proposed in Chapter **??**, we computed the new position of a robot as follows:

```
computeNewPosition: anInteger
  "Returns a point representing the location of the next move."

  ^ direction = #east
    ifTrue: [ self x + anInteger @ self y ]
    ifFalse: [ direction = #west
      ifTrue: [ self x - anInteger @self y ]
      ifFalse: [ direction = #north
          ifTrue: [ self x @ (self y + anInteger)]
          ifFalse: [ self x @ (self y - anInteger) ].
        ]
      ]
```

This is not that nice.

Opposite direction

Similarly, we computed the opposite direction as follows:

```
computeOppositeDirection: aDirection
  "Returns the opposite direction.
  Note that this implementation should be rewritten taking into
    account Compass' way of representing direction and their
    computation'"

  ^ aDirection = #east
    ifTrue: [ #west ]
    ifFalse: [ aDirection = #west
      ifTrue: [ #east ]
      ifFalse: [ aDirection = #north
          ifTrue: [ #south]
          ifFalse: [ #north].
        ]
      ]
```

1.2 Representing directions

We propose that you define a little hierarchy with the class `CpDirection` as a root and as subclasses the four main directions and based on it compute the opposite and a new position in an adjacent position.

Note that by design we avoided directly referring to subclasses but use the root as a factory of instances of its subclasses.

Make sure that the following tests pass and define new ones for each scenario.

```
testSouthReturnOneRowDownPosition

  | newPos |
  newPos := CpDirection south * ( 3 @ 2).
  self assert:  newPos x equals: 3.
  self assert:  newPos y equals: 3.
```

```
testWestReturnLeftPosition

  | newPos |
  newPos := CpDirection west * ( 3 @ 2).
  self assert:  newPos x equals: 2.
  self assert:  newPos y equals: 2.
```

New position at a given distance

While the message `*` was given the next adjacent position, define tests and introduce the message `in: aDistance from: aPosition`.

```
testEastInDistanceReturnRightPosition

  | newPos |
  newPos := CpDirection east in: 3 from: (3 @ 2).
  self assert:  newPos x equals: 6.
  self assert:  newPos y equals: 2.
```

1.3 Introducing NorthWest, SouthEast, and friends

Now that you have got your four positions and all your tests green. Introduce the missing directions: NorthWest, NorthEast, SouthEast, and SouthWest. And enjoy this design.