Maze

A maze consists of rooms and doors:

- An door is either
  - a door into a room
  - an escape to a particular place

- A room has two doors, left and right
Door Data Definition

interface IDoor {
}

class Into implements IDoor {
   Room next;
   Into(Room next) {
      this.next = next;
   }
}

class Escape implements IDoor {
   String name;
   Escape(String name) {
      this.name = name;
   }
}
class Room {
    IDoor left;
    IDoor right;
    Room(IDoor left, IDoor right) {
        this.left = left;
        this.right = right;
    }
}

Copy
class Examples {
    IDoor meadow = new Escape("meadow");
    IDoor street = new Escape("street");
    Room ms = new Room(meadow, street);
    Room planets = new Room(new Escape("mars"),
                            new Escape("venus"));
    Room maze = new Room(new Into(ms),
                        new Into(planets));
}
Finding Paths

Implement the IDoor method canEscape that takes a string and returns a boolean indicating whether an escape with the given name is available.

Replace the canEscape method with a escapePath method that takes a string and returns either a path of “left” and “right” leading to the escape, or a failure value:

Path escapePath(String dest)
Paths

A path result is either

• failure

• immediate success

• left followed by a (successful) path

• right followed by a (successful) path

We’ll need a Path interface with an isOk method
Paths

interface IPath {
    boolean isOk();
}

class Fail implements IPath {
    Fail() {
    }
    public boolean isOk() { return false; }
}

class Success implements IPath {
    Success() {
    }
    public boolean isOk() { return true; }
}

class Right implements IPath {
    IPath rest;
    Right(IPath rest) { this.rest = rest; }
    public boolean isOk() { return true; }
}

class Left implements IPath {
    IPath rest;
    Left(IPath rest) { this.rest = rest; }
    public boolean isOk() { return true; }
}
Door Variations and Person Attributes

Eventually, we want locked doors, short doors, magic doors, and other kinds of doors.

Finding an escape will depend on having keys, being a certain height, etc.

Instead of adding more and more arguments to `escapePath`, let’s introduce a `Person` to carry attributes.

Replace the destination-string argument of `escapePath` with a `Person` argument, where a `Person` has a destination and height.
Short Doors

Add a new kind of door, a short door, where a person must be less that the door’s height to pass

Adding a short door requires only the declaration of a `Short` class — no other code changes!
Locked Doors

Add a new kind of door, a locked door, where a person must have a key to pass

Besides adding Locked, we change Person to add the notion of keys to the person

In contrast to adding new variants, adding new operations requires changing the class
Racket versus Java

Racket:

- New variant ⇒ change old functions
- New function ⇒ no changes to old code

Java:

- New variant ⇒ no changes to old code
- New method ⇒ change old classes

This is the essential difference between functional programming and object-oriented programming.