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.
Door Classes

Person

String dest
double height

boolean isDest(String)
boolean isShorter(double)

IDoor

IPath escapePath(Person)

Escape

String name

IPath escapePath(Person)

Into

Room next

IPath escapePath(Person)
Short Doors

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

```
IDoor
IPath escapePath(Person)
```

```
Escape
String name
IPath escapePath(Person)
```

```
Into
Room next
IPath escapePath(Person)
```

```
Short
Room next
double height
IPath escapePath(Person)
```

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

A **Person** now needs keys...
Locked Doors

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

<table>
<thead>
<tr>
<th>Person</th>
</tr>
</thead>
<tbody>
<tr>
<td>String dest</td>
</tr>
<tr>
<td>double height</td>
</tr>
<tr>
<td>String key;</td>
</tr>
<tr>
<td>boolean isDest(String)</td>
</tr>
<tr>
<td>boolean isShorter(double)</td>
</tr>
<tr>
<td>boolean hasKey(String)</td>
</tr>
</tbody>
</table>

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