Stag

Stag

Stag stands for Simple Text Adventure Game

The Stag assignment gives more details about what that means

Stag development

This is my own personal step-by-step solution to the Stag assignment

It is mostly intended for people who are unsure about classes and objects and pointers, or unsure about how to design or develop a larger project

Unit testing is crucial for confidence, but I won't do any here - the code isn't extensive or complex

The problem is that it is ***open ended*** and the danger is rushing off in the wrong direction









Places

It seems to me best to start with places

Each place should be an object, and I'm going to define a class `Place` to describe them

I am going to develop the `Place` class in painfully tiny steps to start with

Place names

I've decided to take a hint from the assignment, and use place names exclusively, avoiding the complications of directions like "north", "south" etc.

So, I want my adventure game to look like something like this:

```
You are in the house.
There are exits to the garden and the road.

> go garden

You are in the garden.
The only way out is back into the house.
```

Names and descriptions

So, it seems as if a place is going to have a name and a description, something like this:

```
class Place {
    String name, description;
}
```

Code for creating the house might be:

```
house = new Place();
house.name = "house";
house.description = "You are in the house.\n" +
  "There are exits to the garden and the road."
```

Main program

Let's write a `Stag` main program class

The only purpose is to experiment with the `Place` class while I develop it

So I'm not going to worry about its design, and I might move code in and out of it a lot as I go along

To start with, it needs code to set up a game map (with one place), and a field `here` to record the current place

Stag class


```
class Stag {
  Place here;

  void setup() {
    Place house = new Place();
    house.name = "house";
    house.description =
      "You are in the house.  " +
      "There are exits to the garden and the road.";
    here = house;
  }

  void run() {
    setup();
    System.out.println(here.description);
  }
...
```

Working program

At this point I have a working program, so I'm going to take a snapshot

Here are the classes:

Stag.java
Place.java

It doesn't look like much, but I've (a) decided places are objects of a `Place` class (b) decided a place has a name which appears in commands (c) decided a place has a description which appears when you go there and (d) created a main program to experiment with

Methods

What methods does the `Place` class need at the moment? What code needs to be moved into it?

Well, a place should have total responsibility for its own name and description

One way of sorting this out is to make the fields private:

```
class Place {
    private String name, description;
}
```

Now the program doesn't work, and I need to refactor it until it works again

Initialisation

One thing that fails is the initialisation of the fields

I need a constructor that takes in the two field values

```
class Place {
    private String name, description;

    Place(String name0, String description0) {
        name = name0;
        description = description0;
    }
}
```

An idiom

A common idiom, which you see all over the place, is to use the same name for the args as for the fields, and then use `this` to tell them apart, so I am going to change my mind:

```
class Place {
    private String name, description;

    Place(String name, String description) {
        this.name = name;
        this.description = description;
    }
}
```

Arrival

One other thing that doesn't work is printing out the description

Let's make that the responsibility of the place and define an `arrive` method:

```
void arrive() {
  System.out.println(description);
}
```

Worry

I'm worried about the fact that a place is doing printing

Should a place have responsibility for printing?

Ideally, I would say not

For example, if I switched to a graphical user interface later (one with text in, not pictures) I would have to change `Place`

For now, I am just going to make a 'todo' note about it

Repair

Now I can fix up the `Stag` class:

```
class Stag {
  Place here;

  void setup() {
    Place house = new Place("house",
      "You are in the house.\n" +
      "There are exits to the garden and the road."
    );
    here = house;
  }

  void run() {
    setup();
    here.arrive();
  }

...
```

Working program

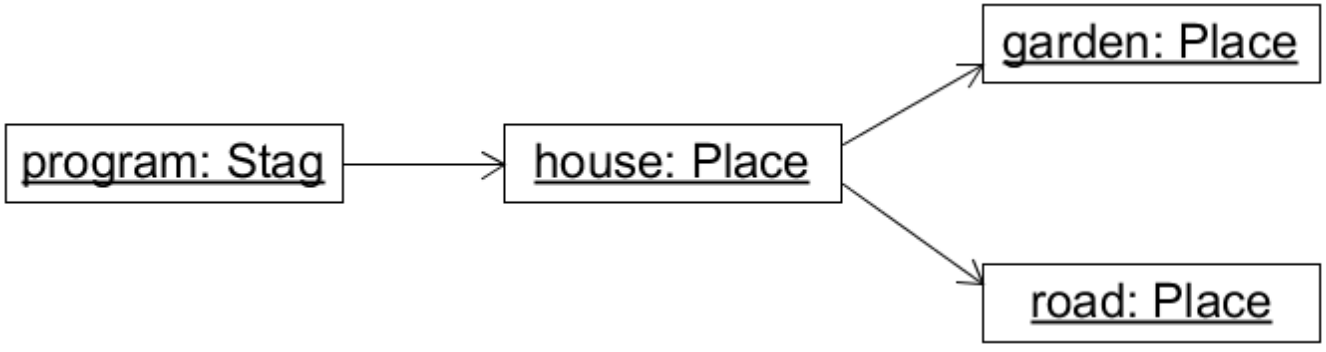
I have a working program again, with these files:

- Stag.java
- Place.java
- todo.txt

All I've done since the last working program is to move some code into the right place

Exits

A `Place` can have *exits* which somehow say where you can go from one place to another



Ideally, I want those exits to be ***pointers*** to other `Place` objects

String exits

This is a possibility:

```
class Place {
    String exit1, exit2;
```

For the house object, fields `exit1`, `exit2` would have the values `"garden"` and `"road"`

The problem with this is that each string will need to be used somehow to find a place object, and it is not clear 'who' has that responsibility or how it would work

Place exits

This is another possibility:

```
class Place {
    Place exit1, exit2;
```

It may look odd for a `Place` to contain `Place`s

But `exit1` is a ***reference*** to a `Place` object, i.e. a ***pointer***

Pointers are implicit in Java – objects are ***always*** accessed via pointers

Limited exits

After experimenting, I came to the conclusion that this ***is*** the right way for `Place`s to point to each other:

```
class Place {
    Place exit1, exit2;
```

But there are other problems: I would have to define enough variables to cope with the maximum number of exits, and then what would happen when I have less than the maximum?

An array of exits

How about this:

```
class Place {
    Place[] exits;
```

Now there is no limit – the array can be exactly the right size for the actual number of exits

This might be awkward when the number of exits changes, but how often will that be? Almost never

I'll accept this – I can always change my mind later

Stag update

Let's get a full program together to see how it feels

```
house.exits = new Place[] {garden, road};
```

```
void run() {
  setup();
  here.arrive();
  here = here.exits[0];
  here.arrive();
}
```

This simulates a `'go garden'` command

Thinking about here

The main program only has a field `here`, not a list or set or map of places (after setup, anyway)

Can I carry on this way?

If there is only ever access to the current place, could there be places which are unreachable?

It seems that if there are, then they would be unreachable in the game anyway

So let's try to continue with this idea

Local place names

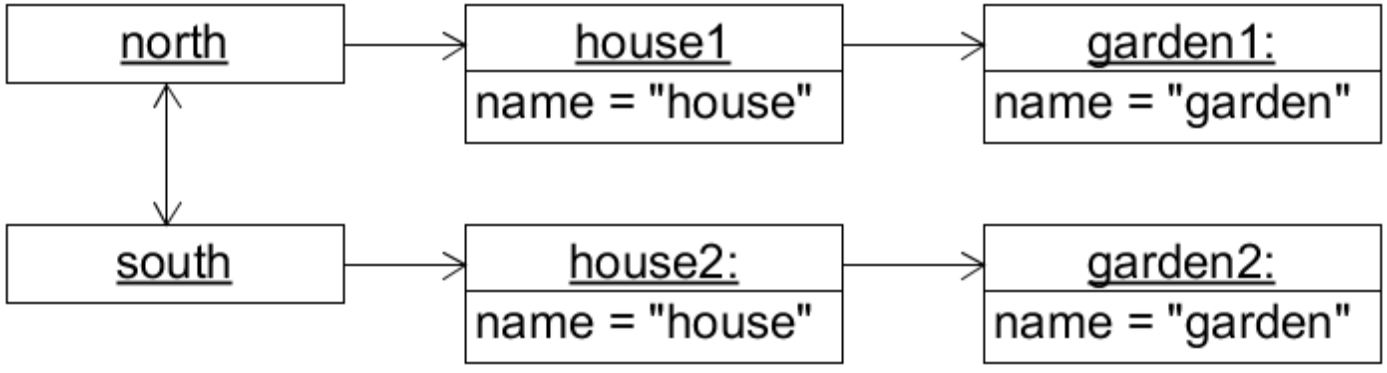
Are place names local? If so, I should be able to do this:

```
You are at the north end of the road.
> go house
You are in the red house.
> go garden
You are in the garden of the red house.
```

```
You are at the south end of the road.
> go house
You are in the blue house.
> go garden
You are in the garden of the blue house.
```

Object diagram

Here's an object diagram to match:



Provided that there is never a place which points to both houses, or to both gardens, this should work

Care is needed in setting up the initial data, that's all

It looks OK, so I'll make a todo note to keep it like this

Working program

I have a working program again, with these classes:

Stag.java
Place.java
todo.txt

I've added only a small amount of code, but made some quite big decisions

Places are responsible for the data which allows the player to move around and, after construction, that data is handled in a purely local way

Going places

To work out where the code should go which handles exits, let's use the same technique as before and make the exits private:

```
private Place[] exits;
```

I then need to work out:

- how to initialise the exits
- how to get from one place to another

Initialisation

Since the exits are fixed (for now anyway) it would seem right to pass them in the `Place` constructor

But a bit of experimentation shows up a problem

If the house is the first object to be constructed, the garden and road objects ***are not available yet*** to be passed in

When there is a cycle of objects, there is ***no*** object you can construct first, and cycles are ***normal*** in this case!

Two phases

There isn't really any perfect solution

The simplest approach is to have a phase 1 where all the place objects are constructed, without exits

And then in phase 2, use a separate `exits` method to initialise all the exits

This is not perfect: the `exits` method is not restricted to be called only once, but the method itself could enforce that - make a note

The exits method

Let's add the exits method in `Place`

```
void exits(Place[] exits) {
    this.exits = exits;
}
```

and a call to it in `Stag`

```
house.exits(new Place[] {garden, road});
```

Passing arrays

Is it possible to do better than this?

```
(in Place) void exits(Place[] exits) {
(in Stag)  house.exits(new Place[] {...});
```

Yes:

```
(in Place) void exits(Place... exits) {
(in Stag)  house.exits(garden, road);
```

The three dots `...` are a Java feature (varargs)

The argument variable `exits` is still an array, but the entries can be passed as separate arguments

Moving

There is another bit of code in `Stag` that doesn't work:

```
here = here.exits[0];
```

If a place is completely responsible for its own exits, then it seems as if the `Place` class should be responsible for moving from place to place

Move method

Let's start like this:

```
? move(?) {
    ?
}
```

Here, each ? means "haven't decided yet"

What goes in, what does the method do, and what comes out?

Passing a string

When the main program receives a command `'go garden'`, it has only the name `"garden"` as a string to work with, and no way to convert it into a place, so it is best to pass the string to `move`:

```
? move(String name) {
    ?
}
```

Changing here

The `move` method can do any printing which might be needed, but in the end, the `here` in the main program object needs to be updated

One possibility is that the `Place` object has access to the `Stag` object so that it can do the updating

Is that sensible? No, it's not - it is upside down

Or, `here` could be made a static variable of the `Place` class, making it a global which any object can get at from anywhere

Is that sensible? No, it's not

Returning here

The best bet is for the `move` method not to update `here` at all, but to return it from the call:

```
Place move(String name) {
    ?
}
```

What if the name isn't found among the exits?

Then `move` could return `null`, but then `Stag` would have to test for it

A possibly better idea is for the place to return itself

Searching

How does `move` find the right exit? Search?

```
Place move(String name) {
    for (Place place : exits) {
        if (place.name.equals(name)) {
            place.arrive();
            return place;
        }
    }
}
```

This doesn't work yet, because what happens if there is no exit with the right name?

Failed searches

If `move` doesn't find the exit, it complains

```
Place move(String name) {
    for (Place place : exits) {
        if (place.name.equals(name)) {
            place.arrive();
            return place;
        }
    }
    System.out.println("You can't go there.");
    return this;
}
```

A feeling

This loop for searching doesn't feel right

```
for (Place place : exits)
```

It feels as if there should be a better data structure which does the search automatically, maybe more efficiently, though efficiency is unlikely to matter

But let's leave it as it is for now and make a note

Also, places are printing again - there is already a note to review that sometime

Updating Stag

As usual, the `Stag` class needs to be updated to catch up with the changes made to `Place`

```
void run() {
    setup();
    here.arrive();
    here = here.move("garden");
}
```

This simulates a single `'go'` command

Working program

I've reached a new working program

The `Place` class is starting to feel 'real', so I've added some comments

- Stag.java
- Place.java
- todo.txt

Commands

Now that there is full mobility, it is worth starting to read in commands from the user, to allow playing

The player will only be able to move around at the moment, but that will be a good test of the `Place` class

The first task is to find out how to read in lines of text from the user

Google searches

The first technique I found on Google used `BufferedInputStream` and looked too complicated

The second technique I found used `Scanner` and looked better, but was still a bit messy with unwanted exceptions

The third technique I found used a new class `Console` added in Java 6, and it seemed just right (though I later found it doesn't work in Cygwin)

Getting started

Let's begin a `read` method for reading in a command and finding its verb and noun

For now anyway, let's put it in the `Stag` class

```
import java.io.*;

void read() {
    Console console = System.console();
    String line = console.readLine("> ");
}
```

It feels as if the first line should be executed once only, but that would complicate the code, and it is cheap

Splitting

I need to split (or 'scan' or 'parse') the command into its verb and noun

A Google search reveals the `split` method on strings:

```
void read() {
    Console console = System.console();
    String line = console.readLine("> ");
    String[] words = line.split(" ");
}
```

Catching

Let's put the verb and noun into fields:

```
String verb, noun;

void read() {
    Console console = System.console();
    String line = console.readLine("> ");
    String[] words = line.split(" ");
    verb = words[0];
    noun = words[1];
}
```

There is no validation - make a note

Loop

Now I can sort out a very simple play loop:

```
void run() {
  setup();
  here.arrive();
  while (true) {
    read();
    if (verb.equals("go")) here = here.move(noun);
    else System.out.println("Do what?");
  }
}
```

Setup

Now I can connect up all the places, so I can wander around without trouble:

```
house.exits(garden, road);
garden.exits(house);
road.exits(house);
```

Working program

Now I have a playable working program:

- Stag.java
- Place.java
- todo.txt

If I got this far and ran out of time for the assignment, I would have submitted this, and would have felt quite proud of it

There are only 80 lines of code, but they have quite good 'value for money'

Things

The next issue to sort out is *things* that can be carried around - what is the life-cycle of a thing?

First, you see a description of the item when you arrive at a new place:

```
You are in the garden.
There is a spade lying on the ground.
```

Then you use `'take'` and `'drop'` on it

Finally, you take it to the right place to solve a puzzle with it

Thing strings

Let's develop a `Thing` class

All a thing seems to need for the moment is a name and a description

This is similar to a place, but there probably won't be enough common code to be worth making a parent class which both `Place` and `Thing` inherit from

```
class Thing {
    String name, description;
}
```

Thing methods

It is difficult to imagine what code might sit inside the `Thing` class besides printing the strings, and even that probably depends on context, e.g. whether taking or dropping

So, instead of making the fields `private`, let's make them `final` instead, which means read-only

```
class Thing {
    final String name, description;
}
```

Constructing things

With `final` fields, the only code needed inside the `Thing` class is a constructor:

```
class Thing {
    final String name, description;

    Thing(String name, String description) {
        this.name = name;
        this.description = description;
    }
}
```

Storing things

The next problem is to store things in a place, ready to be found and picked up

The things stored in one place are going to come and go during the game, so a flexible data structure is needed

Let's try a list:

```
private List<Thing> things;

things = new ArrayList<Thing>();
```

Things coming and going

Methods are needed to put a thing in a place and get it out

```
void put(Thing x) {
    things.add(x);
}

Thing get(String name) {
    for (Thing thing : things) {
        if (thing.name.equals(name)) {
            things.remove(thing);
            return thing;
        }
    }
    return null;
}
```

Thoughts

There is a loop to search for a thing, like the one to search for an exit - is there a better data structure?

Are names of things local? No, because things can be carried around so that they meet up with each other - make a note

Can I get away with simple one-word names for things? Probably not, but there is a hint in the assignment that phrases can be used (like 'red key') - make a note

Data structure

It is time to look for a better data structure for storing things (and exits) to

- get rid of the lookup loops
- make sure there is only one thing with a given name
- later perhaps, allow a thing to be ***replaced***

Sets?

Perhaps a set of things would be right

To get that to work, the `Thing` class might have to have an `equals` method added to compare things by name

Some experimentation with sets shows that there would be no convenient way to use a ***name*** (string) to lookup a ***thing*** (object)

Maps?

The standard data structure which uses a 'key' to lookup a 'value' is a `Map`

For things, a `Map<String,Thing>` would be the right type

No `equals` method is needed for this - things don't need to be globally unique by name, just unique within each map

That means maps ***are*** suitable, ***and*** could be used for exits as well

Using maps

Code for putting and getting things is now:

```
Map<String,Thing> things;

void put(Thing x) {
    things.put(x.name, x);
}

Thing get(String name) {
    return things.get(name);
}
```

This looks a whole lot better, and it is worth switching over to this for exits as well

Place updates 1

The `Place` class needs a fairly radical overhaul:

The `Map` class is imported from `java.util`, and the exits and things are now maps

```
/* A place in a STAG adventure game. */

import java.util.*;

class Place {
  private String name, description;
  private Map<String,Place> exits;
  private Map<String,Thing> things;
```

Place updates 2

The constructor needs to create empty maps:

```
Place(String name, String description) {
    this.name = name;
    this.description = description;
    exits = new HashMap<String,Place>();
    things = new HashMap<String,Thing>();
}
```

Place updates 3

The `exits` method needs to insert into the map

```
// Add the exits, after construction
void exits(Place... exits) {
    for (Place p: exits) this.exits.put(p.name, p);
}
```

Place updates 4

The `move` method now looks like this

```
// Go to a named place, print details,
// and return the new place (or the old one).
Place move(String name) {
  Place place = exits.get(name);
  if (place != null) {
    place.arrive();
    return place;
  }
  System.out.println("You can't go there.");
  return this;
}
```

Place updates 5

The `put` and `get` methods are now:

```
// Drop a thing into this place.
void put(Thing x) {
  things.put(x.name, x);
}

// Take a thing from this place
Thing get(String name) {
  Thing x = things.get(name);
  things.remove(name);
  return x;
}
```

Place updates 6

When arriving in a place, the things that are there need to be announced:

```
// Print descriptions when you arrive at a place
void arrive() {
  System.out.println(description);
  for (Thing x: things.values()) {
    System.out.println(x.description);
  }
}
```

Initial things

The `put` method in a place can be used to set up the initial things, as well as to implement the `'drop'` command, so in the `Stag` class `setup` method:

```
Thing spade = new Thing("spade", "...");
garden.put(spade);
```

Carrying things

Where should I put things that are being carried by the player? In a map variable in the `Stag` class:

```
import java.util.*;
```

```
Map<String,Thing> luggage;
```

```
luggage = new HashMap<String,Thing>();
```

Take and drop

The command loop can be expanded now:

```
while (true) {
  read();
  if (verb.equals("go")) here = here.move(noun);
  else if (verb.equals("take")) {
    luggage.put(noun, here.get(noun));
  }
  else if (verb.equals("drop")) {
    here.put(luggage.get(noun));
  }
  else System.out.println("Do what?");
}
```

Playing

The game can now be played

When the player takes an object, or drops an object, there is no direct feedback

However, the fact that an object in a place is announced on arrival means it is possible to tell that everything is working correctly

Working program

The playable working program at this point is:

- Stag.java
- Place.java
- Thing.java
- todo.txt

There are 120 lines of code now, and both places and things are pretty well sorted out

Confirmation

When the user drops some thing, there should be a message printed out to confirm it

From past experience, this should be in the `put` method

But, when `put` is used for initialisation, it should be silent - so which is best?

- have two methods
- have one method with an extra boolean argument
- have a global flag to switch printing on/off
- do the printing outside of the `Place` class

Thinking

I was never happy with having printing inside the `Place` class - it shouldn't be the responsibility of a place to do printing

In other words, ***user interface*** issues do not belong inside the `Place` class

So I am thinking about moving ***all*** printing code out of the `Place` class

Arrival

The `arrive` method is currently:

```
void arrive() {
  System.out.println(description);
  for (Thing x: things.values()) {
    System.out.println(x.description);
  }
}
```

Maybe it ***is*** the responsibility of the `Place` to do this, because otherwise ***external*** code would have to loop through the things, so `Place` would have to export all the things

Compromise

How about a compromise?

```
String arrive() {
  String s = description + "\n";
  for (Thing x: things.values()) {
    s = s + x.description + "\n";
  }
  return s;
}
```

Now the place is responsible for looping through the things and ***organising*** the text to be printed

But the caller is responsible for ***printing*** the text with `System.out.print()`, or displaying it in a window

Moving

The `move` method is currently:

```
Place move(String name) {
  Place place = exits.get(name);
  if (place != null) {
    System.out.print(place.arrive());
    return place;
  }
  System.out.println("You can't go there.");
  return this;
}
```

There's no reason for much of this to stay inside `Place`, and the method doesn't change `here` anyway, so the name is misleading too

Finding

The `move` method can be replaced by:

```
Place find(String name) {
  return exits.get(name);
}
```

The decision making and printing can all be moved out into the `Stag` class

Going

The immediate changes to `Stag` are:

```
void run() {
  setup();
  System.out.print(here.arrive());
  while (true) {
    read();
    if (verb.equals("go")) {
      Place there = here.find(noun);
      if (there != null) {
        System.out.print(there.arrive());
        here = there;
      }
      else System.out.println("You can't go there.");
    }
...
```

Taking and dropping

Feedback for take can now be added:

```
else if (verb.equals("take")) {
  Thing x = here.get(noun);
  if (x != null) {
    System.out.println("You pick up the " + x.name);
    luggage.put(noun, x);
  }
  else System.out.println("What " + noun + "?");
}
```

And very similarly for drop

Working program

The playable working program at this point is:

- Stag.java
- Place.java
- Thing.java
- todo.txt

This stage consisted just of moving code out of `Place`

The code for executing commands is expanding a lot and needs sorting out - add a note

Puzzles

Sorting out puzzles is perhaps the hardest part of the design

Here's one example puzzle to think about

A box needs to be opened with a key to reveal a diamond

It seems reasonable to think of the box as the puzzle

So do puzzles have a separate class, or a class which extends `Thing`, or are they just `Things`?

A separate puzzle class

Suppose there is a separate thing-like class for puzzles

Then a place might have to contain a collection of puzzles as well as a collection of things, and there might need to be code which tries a noun twice to see if it is a thing or a puzzle

That doesn't sound DRY to me

Puzzles as things

What if puzzles are things?

Maybe the `Puzzle` class extends the `Thing` class

Since puzzles are (probably) not carried, maybe it is better if `Puzzle` and `Thing` have a common parent class `Entity`, say, so that they are compatible

This sounds better

Box action

The box action `'open box'` needs to:

- check that the verb `'open'` has been used
- check that the player is carrying the key
- reveal the diamond
- change the state of the box to 'opened'

Verbs

To ***open*** a box, ***crack*** an egg, ***slide*** a panel, ***push*** a button, ***pull*** a lever and so on might all be the same kind of puzzle

The player needs to use the right verb, and the verb can be stored in the object:

```
class Puzzle {
  String verb;
```

Later, maybe a puzzle will respond to multiple verbs

Needs

To open the box, the player needs the key

A puzzle can hold the thing that the player needs to be carrying to solve the puzzle

```
class Puzzle {
  Thing need;
```

It will turn out to be more convenient to use a string

Later, maybe multiple things will be needed

Responses

When the player tries to open the box, there needs to be a failure or success response:

```
> open box
You can't open it without a key.    OR
You open it to reveal a diamond.
```

The responses can be stored in the puzzle:

```
class Puzzle {
  String failure, success;
```

Revealing

When the box is opened, the diamond is revealed

The box can have the diamond attached to it, ready to be revealed

```
class Puzzle {
  Thing content;
}
```

Later maybe there will be multiple things to reveal

Changing state

What does it mean to change the box to 'opened'?

The box now has a different description, and behaves differently - only the name is the same

The easiest way to change the state radically like that, without writing special code for each puzzle, is to ***replace*** the box object by a different box object

Now the change of state is in the data, and all that's needed in the code is a way to replace things

Replacements

Suppose that the action for solving the box puzzle has two things in it to be revealed

One is the diamond, and the other is the new box

When the replacement box is dropped into the current place, assuming it has the name `"box"`, the same as the original, it will ***automatically*** replace the original box, because of the `Map<String,Thing>` which things are stored in

Multiple contents

That suggests that the box has two things in it to be revealed, and it is worth generalising straight away to a list of things:

```
class Puzzle {
  List<Thing> contents;
```

Working program

The playable working program at this point is:

- Stag.java
- Place.java
- Thing.java
- Puzzle.java
- todo.txt

The code is quite messy at this point, and needs tidying

There are oddities such as `'open garden'` giving `'What garden?'`


march
walk
run
crawl
tiptoe
drag
push
jump
leap
hop
skip
hit
stretch
lift
put down
dive
lean
crouch
sit
squat
bend
pick up
hold
carry
slap / hit
punch
kick
catch
throw
pull

Actions

The next thing I want to do is to smooth out the code which carries out actions

The code for `'go'` ought to be in the `Place` class

The code for `'take'` and `'drop'` ought to be in the `Thing` class

The code for `'open'` and other `'solve'` verbs ought to be in the `Puzzle` class

Nouns

At the moment, when the verb is `'go'`, the exits are searched

And when the verb is `'take'`, `'drop'` or `'open'` the links in a place are searched

And when the verb is `'drop'`, the luggage is searched

This isn't DRY

Links

Suppose a place has a set of links to entities

And an entity is a place or a thing lying around or a piece of luggage being carried or a puzzle

Then there is ***one*** piece of code which looks up a noun to find an entity

And this lookup is independent of the verb

And the entity can be asked whether or not it wants to respond to the verb

Rewrite

This is a fairly heavy refactoring exercise, giving `Place`, `Thing`, `Puzzle` a parent class `Entity` with:

```
Place act(Place here, String verb, PrintStream out) {
    out.println("You can't " + verb + " the " + name);
    return here;
}
```

Entities are again responsible for printing, but they are passed a print stream, for user interface independence

Things being carried are stored in the current place instead of a bag, so that `act` can make ***any*** world changes via `here`

Working program

The playable working program at this point is:

- Stag.java
- Entity.java
- Place.java
- Thing.java
- Puzzle.java
- todo.txt

There's more to do, as always, but I'll stop there