# 05 Polymorphism

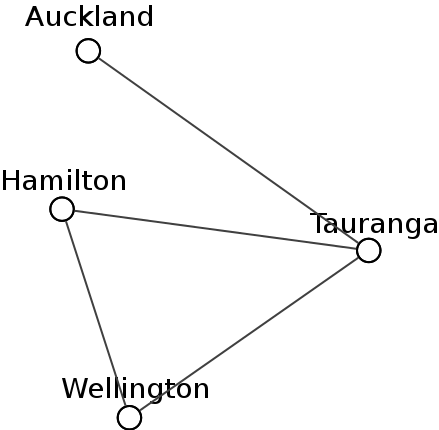
## Part 1

For this part we will use the ParticleSystemExample from 08 Examples.zip. Extract and run the program.

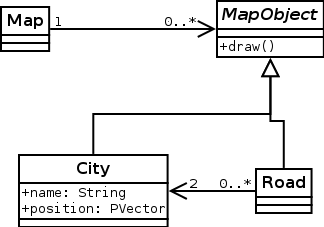
1. This example has five classes. If we consider the tab with setup() and draw() to be a class called “main”, then there are six classes. Drawing the structure of this program as a UML diagram will help you understand it. Your UML diagram should show all of the (i) class names, (ii) fields/properties, (iii) methods, (iv) public/protected/private modifiers, (v) association relationships with multiplicities and (vi) inheritance relationships.  
     
   Note that arraylists are typically not fields but implementations of association relationships, so you don’t need to include them in the list of fields.
2. The program currently has only a single class of particles called Particle. All particles have a default colour of white. To make the animation more interesting, assign a random colour to each particle inside the constructor of the Particle class.
3. The particles are fairly simple, i.e. they are simply circles with a fixed size. For the ExplosionEmitter, we would like the particles to get larger as time goes by, to give the illusion that they are “flying toward” the viewer.  
     
   Create a new class called GrowingParticle that extends Particle to achieve this. The simplest way to achieve this is to override the move() method inside GrowingParticle so that the value of the property size is adjusted appropriately whenever the particle moves.
4. Demonstrate the GrowingParticle class by making a random half of the particles emitted by the explosion emitter into objects of type GrowingParticle. That is, inside the ExplosionEmitter class, you can replace  
     
   Particle particle = new Particle(position.x,  
    position.y, velocity.x, velocity.y);  
     
   with  
     
   Particle particle;  
   if (random(1)<0.5)  
   particle = new Particle(position.x, position.y,  
    velocity.x, velocity.y);  
   else   
   particle=new GrowingParticle(position.x, position.y,  
    velocity.x, velocity.y);  
     
   Explain how this is polymorphism.
5. The program currently creates only two particle emitters inside the setup() method. Modify the program so that *no* emitters are initially created. Instead, whenever the user clicks the mouse, a particle emitter is created at the mouse click location and added to the emitters list. The emitter should be an object of type HoseParticleEmitter or an ExplosionEmitter with 50% probability.  
     
   Note that you should *not* modify the draw() method at all.  
     
   Explain how polymorphism and arraylists enable you to generate many different emitters of different types without the need to modify the draw() method.

## Part 2

A map is made up of places (e.g. cities) and routes connecting the places (e.g. roads). To make life simple, lets assume the roads are all straight. Here is an example of a map:



Write a program to draw a map in an object oriented way. The class design for the program is as follows:

‘

Note the following in case it’s not obvious:

* the MapObject class is abstract
* City and Road inherit from MapObject
* each Road object must be connected to exactly two City objects – this means that you should put references to the City objects inside the Road class
* the Map class is a simple class that contains an arraylist of MapObjects.

You should not duplicate any coordinate data in the program. In order to determine where the roads start and end, you can use the references to the City objects contained in the Road class, e.g currentRoad.firstCity.position.x.

The setup()/draw() methods should simply create and draw a Map object.

To demonstrate your program, have it draw a map such as the one shown, with three extra cities added.