Program:

```
#include <iostream>
using namespace std;

//Region
//Forward declaration of Region class to use it can be used in
//Animal class
class Region;
//Base class for all organisms
class Organism
{
  protected:
    string name; //Name of organism
  public:
   Organism(const string& n) : name(n) { }
   //These are pure virtual functions, meaning all derived classes
   //must define them.
   virtual void live() = 0;
   virtual void eat() = 0;
   virtual void sleep() = 0;
   virtual void move() = 0;
   virtual void type() = 0;
//Get name of Organism
 string getName() const
{
   return name;
}
virtual ~Organism() { } //virtual destructor so derived classes
                        are deleted correctly
;
```

```
//Derived class Plant is a type of organism
class Plant : public Organism
{ public:
    Plant(const string& n) : Organism(n) { }
 //implementing the virtual functions
void live()
{ cout << *name* << " is photosynthesizing.\n";
}
void eat()
{ cout << name << " absorbs sunlight and nutrients.\n";
}
void sleep()
{ cout << name << " is sleeping.\n";
}
void move()
{ cout << name << " cannot move.\n";
void type()
 { cout << name << " is a Plant.\n";
  }
};
//Animal also inherits from organism class
class Animal : public Organism
{ protected:
    Region* region; //Pointer to the region the animal belongs to
```

```
public:
  Animal(const string& n, Region* r) : Organism(n), region(r){}
void move()
{ cout<< name << " is moving\n";
}
void sleep()
  { cout<< name << " is sleeping\n";
  }
};
//Harbivore is a type of Animal
//class Harbivore : public Animal
{ public:
    Harbivore(const string& n, Region* r) : Animal(n,r) {}
  void eat()
    { cout<< name<< " is eating plants\n";
    }
  void live()
  { cout << name << " is alive \n"; }
  void type()
  { cout<< name << " is a Harbivore\n";
  }
};
//class Carnivore is another type of Animal that hunts
//others animals
```

```
class Carnivore: public Animal
{ private:
    Animal* prey; //This is carnivores Target (Association)
  public:
    Carnivore (const string& n, Region* r): Animal (n, R), prey
(nullptr) {}
  void setprey(Animal* p)
  { prey = p;
  }
  void live()
  { cout << name << " is hunting";
    if (prey)
    { cout << name << " is targeting " << prey -> getName() << endl;
    }
  void eat()
  { if (prey)
    { cout << Name << " eats " << prey -> getName() << endl;
    }
    else
    { cout << name << "finds no prey \n";
    }
}
```

```
void type()
 { cout<< name<< " is a Carnivore \n";
 }
};
// A region is a collection of Organisms
class Region
{ private:
    string name;
    Organism* organism[10]; //stores up to 10 organisms
    int count; //number of organisms added
  public:
    Region(const string& n): name(n), count(0){}
   ~Region()
   { //delete all organisms when Region is destroyed
    for (int i=0; i<count; i++)
    { delete organisms[i];
    }
    void addOrganisms (Organism* org)
    { if (count<10)
       { organisms[count++]=org;
       }
       else
       { cout<< "Region is full can't add more organisms
         \n";
       }
   }
   void display()
   { cout<< "\n--Region: "<< name<< "--\n\n";
```

```
for (int i=0; i<count; i++)
 {
   organisms[i]->live(); // calls appropriate function using polymorphism
 }
Organism* getDominantSpecies()
{
   Organism* dominant = nullptr;
   size_t maxlen = 0;
   for (int i=0; i<count; i++)
   {if (organisms[i]->getName().length()>maxlength)
    {  dominant = organism[i];
       maxlen = organism[i]->getName().length();
    }
   }.
  return dominant;
};
//world class that holds multiple Regions (composition)
Class World
 { private:
     Region* regions[5];
     int count;
   Public:
     World() : count(0){ }
   ~World()
    { for(int i=0; i< count; i++)
       { delete regions[i];
       }
    }
```

```
Void addRegion(Region* r)
{
    if (count< 5)
    {
      regions[count++]=r;
    }
    else
    {
      cout<<"World is full, can't add more regions.\n";
    }
}
 void display()
 {
    for(int i= 0;i < count; i++)
    {
      regions[i]->display();
    }
 };
int main()
{ World world;
  //create a region
  Region * forest = new Region ("Rain Forest");
  //Create some organisms
  Plant* p1 =new Plant ("Fern");
  Herbivore* h1 = new Herbivore("Deer",forest);
  Carnivore* c1 = new Carnivore ("Tiger",forest);
  //set prey for the carnivore
  c1->setPrey(h1);
  // Add organisms to the region
    forest->addOrganism(p1);
    forest->addOrganism(h1);
    forest->addOrganism(c1);
```

```
//Add region to the world
world.addRegion(forest);
    world.display();

//show the dominant species
    cout<<"\n Dominant species in forest: ";
    Organism* dom= forest->getDominantSpecies();
    if(dom)
    { dom->type();
    }
    return 0;
}
```