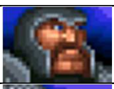
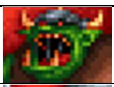
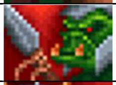
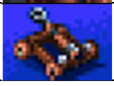
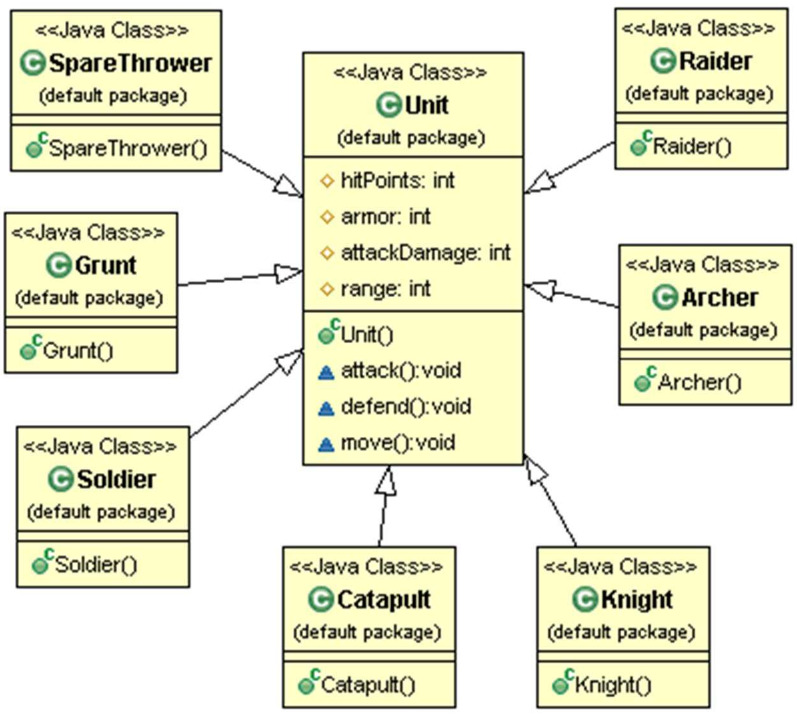
Module: Technologies and Paradigms of Programming
Topic: Design Patterns

The Warcraft game: Humans vs Orcs© is a strategy game in real time where two species fight between them: humans and orcs. Every Specie has the same number of military units, but they are specialised for every specie. We can see a summary in the following table:

Type	Specie			
	Humans		Orcs	
Infantry		Soldier		Grunt
Archer		Archer		Spear Thrower
Horseman		Knight		Raiders
SiegeMachine		Catapult		Catapult

These characters have to be represented in memory according to the next hierarchy of classes:


<<Java Class>> Unit (default package)
hitPoints: int
armor: int
attackDamage: int
range: int
Unit()
attack():void
defend():void
move():void
<<Java Class>> SpareThrower (default package)
SpareThrower()
<<Java Class>> Raider (default package)
Raider()
<<Java Class>> Grunt (default package)
Grunt()
<<Java Class>> Archer (default package)
Archer()
<<Java Class>> Soldier (default package)
Soldier()
<<Java Class>> Catapult (default package)
Catapult()
<<Java Class>> Knight (default package)
Knight()

Module: Technologies and Paradigms of Programming
Topic: Design Patterns

From the above hierarchy, it has been modelled an artificial intelligence (A.I.) to manage the Orcs. A fragment of that class AI**Orcs** is on the right, where we can observe a method which creates a combat unit made of several units of each type.

```java
// Artificial Intelligence for Orcs
public class AIOrcs {

/**
  * Generation of a group of attack for A.I.
  * @return An attack group with Orcs
*/
public Unit[] createGroupOfAttack()
{
       // Array of Orcs Units
       Unit[] groupOfAttack = new Unit[10];

       // 4 x Infantry
       for(int x = 0; x < 4; x++)
              groupOfAttack[x] = new Grunt();
       // 3 x archer
       for(int x = 4; x < 7; x++)
              groupOfAttack[x] = new SpareThrower();
       // 2 x horseman
       groupOfAttack[7] = new Raider();
       groupOfAttack[8] = new Raider();
       // 1 x siege machine
       groupOfAttack[9] = new Catapult();

       return groupOfAttack;
}
```

Programmers could see that this A.I. works very well, so they have decided to use it for both Orcs and Humans.

Therefore, the class **AICommon** has been renamed. Then, a constructor **AICommon(String Specie)** will create orcs units ("orcs") or human units ("humans") depending on the string Specie. We want to apply the most suitable design pattern to avoid creating a conditional sentence for every **new** when a specific type of unit is created.

Answer to the following questions:

1. From the design patterns which have been studied along the different lectures, which would be the most appropriate in order to facilitate to the class **AICommon** to choose the specie, and then create a specific number of units to be managed in every object? **Justify the answer.**

   The pattern should be a Creational type, because we have to create objects. Therefore, the most appropriate pattern is the **Abstract Factory**, because we have to create different families of objects (Unit objects in the problem). Defining a family by specie: human or orc.
   This pattern allows to disengage "the need of creating a new object" from "the application of the operator new over a specific class". So, the object which create the units would be a different object from the one which needs and uses them after.
   Moreover, the Abstract Factory is the creational pattern to assure that after establishing the specie (the family), the factory of objects will generate exclusively products of its family.

2. After applying this pattern, could it be possible to generate units for both species in the same attack group? **Justify the answer.**

   No, this situation is not possible. The specific factory will generate human units or orc units. Therefore, each specific factory will generate only units of a concrete specie.

3. Which structure of classes has to be added, how will it integrate with the classes which already exist? Draw the diagram. Indicate the function for every new class. **(2.5 points)**

We have to create a FactoryAbstract interface with operations which return units for each type: **`Unit createInfantry(); Unit createArcher(); Unit createHorseman(); Unit createSiegeMachine();`**[1]

*The interface would implement the HumanFactory and OrcFactory classes. A specific factory for every specie. Every specific Factory will implement the former operations, creating via the operator **new** units for its specie. For example, in the HumanFactory:*

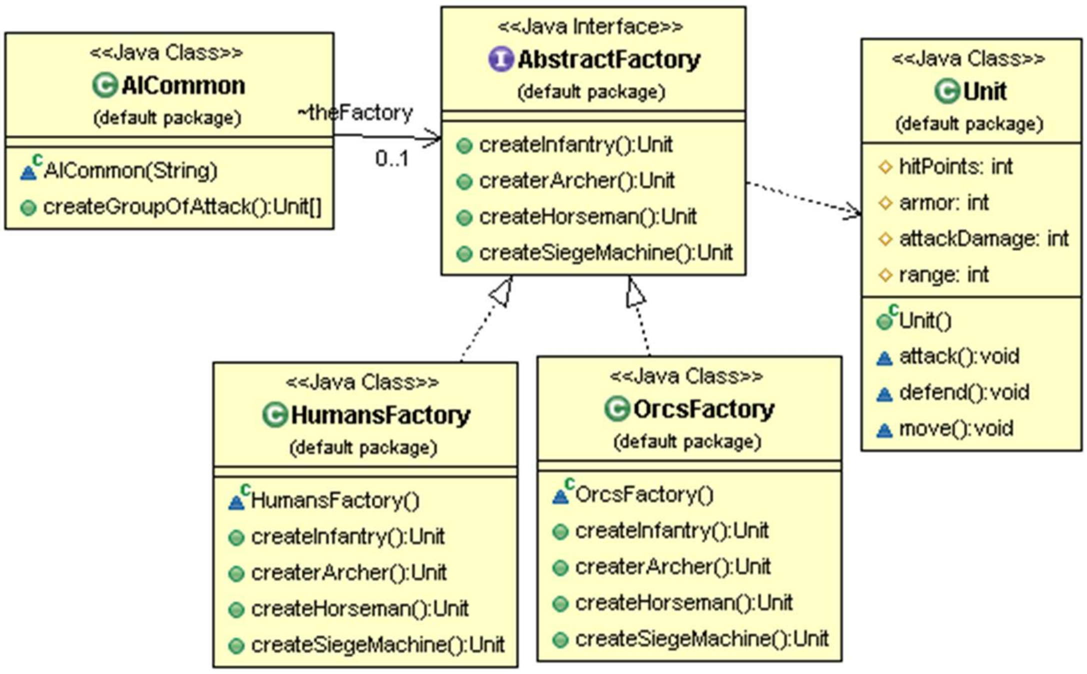
```
Unit createInfantry(){

   return new Soldier();

}
```

Finally, the class AICommon will use **`FactoryAbstract`** *by composition to generate its units.*

So, the class which needs the units and uses them (AICommon) is different from the class which creates the objects in the end (FactoryOrc or FactoryHuman).



4. Implement the class AICommon with its new constructor and its method createGroupOfAttack(). **(2.5 points)**

```
// Artifitial Intelligence common
public class IACommon {

      FactoryAbstract theFactory;

      IACommon(String specie){
```

[1] According to the general architecture, an abstract interface could be defined for every type of product: IInfantry, IArcher, IHorseman, ISiegeMachine, and every type of a specific unit would implement its interface. Therefore, for example createHorseman() would return an object of type IHorseman. However, in our case it is not necessary because all units have a base class in common: Unit.

```
            // We create the specific factory according to the specie
        if(specie.equals("orcs"))
                theFactory = new FactoryOrcs();
            else
                theFactory = new FactoryHumans();
        }

        /**
         * Generation of an attack group of the A.I.
         * @return An attack group orc or human depending on the factory
         */
        public Unit[] createGroupOfAttack()
        {
            // Array of Orcs Units
            Unit[] groupOfAttack = new Unit[10];

            // 4 x infantry
            for(int x = 0; x < 4; x++)
                groupOfAttack[x] = theFactory.createInfantry();
            // 3 x archer
            for(int x = 4; x < 7; x++)
                groupOfAttack[x] = theFactory.createArcher();
            // 2 x horseman
            groupOfAttack[7] = theFactory.createHorseman();
            groupOfAttack[8] = theFactory.createHorseman();
            // 1 x Siege Machine
            groupOfAttack[9] = theFactory.createSiegeMachine();

            return groupOfAttack;
        }
}
```

5. Would it be very difficult to add a third type of specie to the game? For example:

   Specie No-Dead (Skeleton/LancerZombie/HorsemanNoHead/CannonOfPlague)

   Which changes should be done?

   It would not be very difficult. It is not necessary to modify any factory (classes) already created. Basically, it means to create new classes.
   1º/ We have to define these new units as subclasses of Unit.
   *2º/ We will create a new specific factory **DeadFactory**, and its operation* **`createInfantry()`** *will create objects from the class **Skeleton***, and in the same way with the rest of types of units.

6. Would it be very difficult to add a new type of units for each specie? For example:

Type	**Specie**					
	Humanos		**Orcs**		No-Dead	
Magician		Conjurer		Wizard		Nigromante

   Which modification should be done? (1 point)

   It would be a bit more difficult than the former section, because it has to be modified all specific factories that we have defined before.
   1º/ Create new units as a subclass of Unit.
   2º/ In the abstract factory, it has to be created a new method: **`Unit createMagician();`**
   3º/ Implement this operation in a specific factory.
   For example, for the **`HumanFactory`**:

```
Unit createMagician(){
  return new Conjurer();
}
```