## OOP Sample Exam 2 – 2. Army of Creatures

**Armies** are fighting with their ancient mythological creatures.

* Each **creature** has properties: **attack**, **defense**, **health points**, **damage** and **list of specialties**.
* Each **specialty** can apply special rules (changing a creature’s properties) during the battle.
* The battle consists of turns. Each turn can be one of 3 possible actions:
  + **Adding** new creatures to one of the armies
  + **Attacking** – one creature from one army attacks another creature from another army
  + **Skipping** turn. In this action the creature does not attack, but can receive some bonuses to its properties

You are given an object-oriented implementation of the creatures, specialties and battles between the armies.

You task is to **extend the existing code** following all the requirements described in this document.

Examine the existing code for hints and to better understand how it works. Pay special attention to where the specialty methods are called (**ApplyWhenAttacking**, **ApplyWhenDefending**, **ApplyAfterDefending**, **ChangeDamageWhenAttacking**, **ApplyOnSkip**).

To give you an overview of the code: There are **3 folders** in the given project:

* Some of the code is already implemented in the ArmyOfCreatures.Logic namespace (**/Logic/ folder**). **You are not allowed to change, add or remove code in this folder**. The important classes in this folder are:
  + The **BattleManager** class which is responsible for simulating a battle between two armies of creatures. It contains the logic of the **Add**, **Attack** and **Skip** actions during a battle.
  + The **CreaturesInBattle** contains the current properties for creatures in a battle. It has 3 important methods: **DealDamage**, **Skip** and **StartNewTurn**.
  + The abstract class **Creature** and a few creature implementations
  + The abstract class **Specialty** and a few specialty implementations
  + The **CreaturesFactory** class is responsible for creating creatures from a given type name as a string.
* There is an ArmyOfCreatures.Console namespace (**/Console/ folder**) which is responsible for reading and writing to the console and command parser for the 4 commands controlling the battles between the armies.
  + **You are not allowed to change, add or remove code from this folder, too.**
    - The only methods **you are allowed to change** in the ArmyOfCreatures.Console namespace are:
      * The static method **GetCreaturesFactory** in the **ArmyOfCreaturesMain** class.
      * The static method **GetBattleManager** in the **ArmyOfCreaturesMain** class.
  + You should not concern yourself with handling input and output data – the engine does it for you.
* There is an empty namespace ArmyOfCreatures.Extended (/Extended/ folder) in which you should put all your code.
  + In the folder named Creatures in the Extended folder put your implementations of the Creature class
  + In the folder Specialties put your implementations of the Specialty class
  + All other code files should be put directly in the folder Extended
  + You can safely delete the DeleteMe.cs file

### Commands

There are 4 commands that the application supports:

* add command – adds *Count* number of *CreatureType* creatures to one of the two armies (with number *ArmyId*)
  + Syntax: add *Count CreatureType(ArmyId)*
  + Example: add 10 Archangel(2) – adds 10 archangels to the second army
  + Note: no two creatures with both the same creature type and army id will be added
* attack command – executes an attack between creature with type *AttackerType* from army with number *AttackerArmyId* and creature with type *DefenderType* from army with number *DefenderArmyId*
  + Syntax: attack *AttackerType*(*AttackerArmyId*) *DefenderType*(*DefenderArmyId*)
  + Example: attack Angel(2) Goblin(1) – the angels from the second army attack the goblins from the first army
* skip command – skips the turn of the given *CreatureType* from the given *ArmyId*
  + Syntax: skip *CreatureType(ArmyId)*
  + Example: skip Griffin(2) – skips the turn of the griffins from the second army
* exit command – Immediately exits the console application
  + Syntax: exit

### Tasks

* Add class **Goblin**. The **Goblin** is a creature with attack **4**, defense **2**, health points **5** and damage **1.5** and has **no specialties**.
  + *Hint: Examine other successors of the* ***Creature*** *class*
* Add class **AncientBehemoth**. The **AncientBehemoth** is a creature with attack **19**, defense **19**, health points **300** and damage **40**, and has the following specialties:
  + **ReduceEnemyDefenseByPercentage** specialty with 80% defense reduction
  + **DoubleDefenseWhenDefending** specialty for 5 rounds
  + *Hint: The class* ***AncientBehemoth*** *is similar to* ***Behemoth*** *creature class.*
* Add class **DoubleDamage**. The **DoubleDamage** is a specialty that **doubles the current damage** during battle.
  + The **DoubleDamage** class should have only one **constructor** that accepts one argument – the number of rounds for the specialty to have effect. After these rounds (attacks) the effect of this specialty stops.
    - The number of rounds in the constructor should be greater than 0
    - The number of rounds in the constructor should be less than or equal to 10
  + Override the default **ToString()** implementation to return the name of the specialty with the number of rounds remaining in parenthesis. Example: “**DoubleDamage(7)**”
  + *Hint: The class* ***Hate*** *(specialty) also changes the damage during battle.*
  + *Hint: The class* ***DoubleDefenseWhenDefending*** *also has fixed rounds of effectiveness.*
* Add class **WolfRaider**. The **WolfRaider** is a creature with attack **8**, defense **5**, health points **10**, damage **3.5** and:
  + **DoubleDamage** specialty for 7 rounds
* Add class **Griffin.** The **Griffin** is a creature with attack **8**, defense **8**, health points **25** anddamage **4.5**. It also has the following specialties:
  + **DoubleDefenseWhenDefending** for 5 rounds
  + **AddDefenseWhenSkip** with 3 defense points
  + **Hate** specialtyto **WolfRaider** creatures
    - *Hint: The* ***Angel****,* ***Archangel****,* ***Devil*** *and* ***ArchDevil*** *creatures also have the* ***Hate*** *specialty.*
* Add class **AddAttackWhenSkip. AddAttackWhenSkip** is a specialty that **adds attack points** to the permanent attack points of the creature and is applied **when a creature skips** its turn.
  + The class should have only one **constructor** which accepts an integer argument (between 1 and 10, inclusive) – the value to add to the permanent attack of the creature during a skip action in battle.
  + Override the default **ToString()** implementation to return the name of the specialty with the number of attack to add in parenthesis. Example: “**AddAttackWhenSkip(3)**”
  + *Hint: The class* ***AddAttackWhenSkip*** *is similar to* ***AddDefenseWhenSkip****.*
* Add class **DoubleAttackWhenAttacking**. The **DoubleAttackWhenAttacking** is a specialty. It **doubles the current attack** when a creature is attacking.
  + The class should have only one **constructor** that accepts one argument – the number of rounds for the specialty to have effect. After these rounds the effect of this specialty stops.
    - The number of rounds in the constructor should be greater than 0
  + Override the default **ToString()** implementation to return the name of the specialty with the number of rounds left in parenthesis. Example: “**DoubleAttackWhenAttacking(4)**”
* Add class **CyclopsKing**. The **CyclopsKing** is a creature with attack **17**, defense **13**, health points **70**, damage **18**, and the following specialties:
  + **AddAttackWhenSkip** with 3 attack points for each skip action.
  + **DoubleAttackWhenAttacking** for 4 rounds
  + **DoubleDamage** for 1 round
* **Implement support for working with 3 armies (instead of only 2 in the current implementation)**
  + The console application should be able to process commands where the ***ArmyId*** is equal to 3
    - **add 10 ArchDevil(3)** should be a valid command
    - **attack Angel(1) ArchDevil(3)** should also be a valid command
    - See the **second zero test**
  + Remember: You are NOT allowed to edit the **BattleManager** class or the command classes in the **Console** folder.

### Additional Requirements

* Name all classes exactly as explained above
  + For example the class containing **AddAttackWhenSkip** specialty should be called exactly **AddAttackWhenSkip**
* All reference arguments that are passed to externally visible methods **should be checked against** **null**.
  + Throw an **ArgumentNullException** when the argument is **null**.
* Implement all described data validations
  + For example: As described above, the **AddAttackWhenSkip** constructor should accept only positive values between 1 and 10, inclusive.
* Your code should compile without any warnings.
* Do not hide existing methods with the **new** keyword.
* Follow all the described rules and DO NOT change the existing code as described above.