The goal of this lab is practice the design and implementation of a complex object using the Builder pattern.

Building Dungeons

One of the tasks that game developers face is the creation of complex game levels using a relatively small number of simple classes that represent a large variety of elements in lots of games. Each game has its own type of room, its own set of monsters, and treasures that are specific to it. For instance, a game set in a medieval fantasy could have dark dungeon rooms full of orcs and goblins while a game set in the future could have rooms full of digital screens full of other humans and aliens. To ensure that their worlds are built to their specification, they will often construct levels using the Builder pattern where the Builder object is not only responsible for creating the final Level object, but will also ensure that the objects that are added to the level are from a predefined set.

What to do

**Package:** dungeon

Start by downloading [this dungeon.zip archive](https://northeastern.instructure.com/courses/63372/files/7682362?wrap=1)[download](https://northeastern.instructure.com/courses/63372/files/7682362/download?download_frd=1)that contains four classes that can be used to create a level in a game: Level, Room, Monster, Treasure. After adding these files to your project take a moment to familiarize yourself with each of them.

Implement a MedievalLevelBuilder class that can be used to construct a level in a game that is set in a medieval fantasy. This class will be used to not only build a level for a game, but it will also ensure that all of the details of the game are consistent. To do this your builder class should have the following:

* A constructor that takes the number of the level that is being created since most games have many levels. It should also take non-negative values for the target number of rooms, monsters, and treasures that the level is expected to have.
* An addRoom method that has a single parameter for the room's description and adds a room with that description to the level. The method throws an IllegalStateException if too many rooms are added to the level.
* Methods for adding four different types of "monsters" to the specified room as the first parameter (by index starting with 0):
  1. A **goblin** is a "mischievous and very unpleasant, vengeful, and greedy creature whose primary purpose is to cause trouble to humankind" and are the weakest type of monster in our level. They are quite numerous and often travel in packs. The addGoblins method should add the specified number of goblins to the specified room giving each 7 hit points.
  2. An **orc** is a "brutish, aggressive, malevolent being serving evil" but tends to be more of a loner than the goblins. The addOrc method should add a single orc to the specified room giving them 20 hit points.
  3. Even stronger than orcs are orges. An **ogre** is a "large, hideous man-like being that likes to eat humans for lunch". They have 50 hit points.
  4. Our dungeon can also contain humans that will be stored as a type of monster. The details of the human must be provided to the addHuman method.

Adding monsters to the level should raise an IllegalStateException if the target room has not been created or if the target number of monsters has already been reached and an IllegalArgumentException if the target room is invalid

* Methods for adding four different types of treasure to the specified room as the first parameter (by index starting with 0):
  1. The addPotion method should add "a healing potion" (value = 1) to the specified room.
  2. The addGold method should add "pieces of gold" of the specified value to the specified room.
  3. The addWeapon method should add the specified weapon to the specified room. All weapons have a value equal to 10.
  4. The addSpecial method can be used to add the most exclusive treasures to the specified room.

Similar to adding monsters, adding treasures should raise an IllegalStateException if the target room has not been created or if the number of treasures has already been reached and an IllegalArgumentException if the target room is invalid.

* Finally, your builder class should have a build method. This method should return the level only after all rooms, monsters, and treasures have been added. If it is called before completion, it should raise an IllegalStateException.