Java project step 1

Please read the project description and refer to it while reading this.

Our first step will be to create an XML parser for the dungeon files. Included in this .zip file is an exampleParserSource file that shows how to create a parser for different xml files using the SAXparser. While you will not have complete classes for the dungeon objects created by the parser, you will need skeleton to build skeleton classes. parserXML.pdf in the Overview directory shows classes that are built in my project, and a description of the xml read for the project is shown below. You should have methods defined that receive values of all of the properties of an creature, action, etc. Your output will be a printout indicating each method and constructor call of one of the classes shown in parserXML.pdf during the parsing of the xml files. The easiest way to do this is to simply put print statements in each of the functions and constructors in your skeleton classes.

We have included a sample parser in the StudentXMLHandler.java file. There are comments in the StudentXMLHandler.java file that describe how the parsing works. You should also read the README.txt file that describes the high-level description of the structure of the entire example.

To run the example, do *`javac Test.java`* followed by *`java Test <filename>`*, where *`<filename>`* is the file in *`xmlfiles`* to be parsed. There is only one file there.

Structure of the XML file.

The XML file has the following high-level structure.

It starts by declaring the start of the dungeon, i.e.

```
<?xml version="1.0" encoding="UTF-8"?>
<Dungeon name="testDrawing" width="n" topHeight="n" gameHeight="n" bottomHeight="n">
<room and passage definitions>
</Dungeon>
```

Where **name** is the name string for an object, **`width`**, **`topHeight`**, **`gameHeight`** and **`bottomHeight`** are the logical display size parameters mentioned in the description of the project in RogueProjectInstructions, and ***n*** is an integer value for the parameter. These will always be string versions of the integer value.

Next Rooms and passages are defined. Rooms and passages can happen in any order. Creatures (monsters and the player) and items are always defined as part of a room within a room definition.

A group of rooms is defined as:

```
<Rooms>
       One or more room declarations
</Room>
```

A room is declared as:

```
<Room room="n">
<visible>n</visible>
<posX>n</posX>
```

```
<posY>n</posY>
<width>n</width>
<height>n</height>
<creature and item definitions>
</Room>
```

n is again an integer value that is basically an id that may be useful in debugging. **posX** and **posY** are the coordinates within the game area of the upper left hand corner of the room. **width** is how wide the room is, and **height** is how tall it is. *creature and item definitions* are the definitions of creatures and items that at the start of the game are in the room.

There is no required order among **visible**, **posX**, **posY, width, height** and *<creature and item definitions>*.

Monster creatures are defined as:

```
<Monster name="cccc" room="n" serial="n">
<visible>n</visible>
<posX> n </posX>
<posY> n </posY>
<type>T</type>
<hp> n</hp>
<maxhit> n</maxhit>
<creature action definitions>
</Monster>
```

cccc is a string that gives the name of the monster, i.e., “Troll”, “Hobgoblin” or “Snake”. *n* is as above an integer. **room** is the number of the room that the monster is in, primarily for debugging purposes in the reference implementation of this, and the serial is just another identifying number for the monster, again primarily for debugging purposes.

visible should be 0 if invisible and 1 if visible. In the current version of the game, all objects are visible, and so this should always be 1. **posX** and **posY** are the coordinates of the monster at the start of the game relative to the upper left corner of the room the monster is in. hp is the number of hit points the monsters health starts at, and maxHit is the maximum number of hitpoints of damage a monster can deliver.

There is no required order among **visible**, **posX**, **posY, type, hp** and **maxHit**.

Player definitions.

Players are defined like Monsters, except the name is “Player” and the type is ‘@’.

Players can also have item definitions included in their definition, i.e. the line *<creature action definitions>* in the above should be line *<creature action and item definitions>*. The only items that will be defined in a Player are one sword and one armor.

Creature action definitions:

```
<CreatureAction name="cccc" type = "tttt">
<actionMessage>cccc</actionMessage>
<actionIntValue>n</actionIntValue>
<actionCharValue>c</actionCharValue>
</CreatureAction>
```

`name` specifies the name of the creature action, e.g., *YouWin*. Type is the type of the action, i.e., **`hit`**, **`death`**, **`move`**, specifying that it should be performed when the creature is hit, killed or moves. actionIntValue and actionCharValue specify parameters to the action, and their meaning is determined by the specific action type. *n* and *c* are an integer and character, respectively, and *cccc* and *tttt* are strings.

There is no required order among **`actionMessage`**, **`actionIntValue`** and **`actionCharValue.`**

Item definitions:

<*ItemType* **name="***cccc***" room="***n***" serial="** *n***">**
<visible>*n***</visible>**
<posX>*n***</posX>**
<posY>*n***</posY>**
<item action definitions>
</*ItemType***>**

ItemType is either Scroll, Armor or Sword, i.e., is the name of the kind of item this is. cccc and n are strings and integers, respectively. Room is the room the item is in at the start of the game and serial is a unique id for the item in the room. I included room and serial for debugging purposes.

`visible` should be 0 if invisible and 1 if visible. In the current version of the game, all objects are visible. **posX** and **posY** are the coordinates of the item relative to the coordinate of the upper left corner of the room. Item action definitions are defined below.

There is no required order among **`visible`**, **posX**, **posY** and *<item action definitions>*.

Item action definitions:

Item action definitions look like Creature action definitions.

What you should turn in:

In the video turnin location on Brightspace, turn in

A video of you downloading your code from the Brightspace directory, building and running it on the dungeon.xml and testdrawing.xml files. The video should be short – I do not need a long explanation of what your code does, just the building and running of your project. If I need explanations of things I will contact you.

In the code turnin location on Brightspace, turn in

A directory userid.

Under it should be a *src* directory and an *xmlfiles* directory *javac src/Test.java* should compile the main *Test* class and any related classes. *java src.Test <filename>* should parse the file *xmlfiles/filename.* The *Tes*class in the example shows how to do this. Note that your IDE may use a different path to get to the file, and you may need to adjust the path to your file after putting your code into the proper form for turning it in. Do a quick test on a file before submitting to make sure everything is working.

You may use packages if you want, just make sure *javac src/Test.java* does compile your program.