CS 249: Assignment 06

UML Diagram (10%)

You will submit a SINGLE diagram that contains:

- The class diagram for the Item class
- The class diagram for the Tome class
- The class diagram for the Creature class
- The class diagram for the Rat class
- The class diagram for the Skeleton class
- The diagram for the Drawable interface
- The diagram for the Loadable interface
- The relationship between them

If a method is overridden OR an implementation is provided, include it in the diagram. Otherwise, you can leave it out.

Important notes:

- Remember the difference between inheritance (solid line) and implementing (dashed line)!
- Remember that interfaces have a special syntax for their titles! (The extra <<interface>> part before the title!)
- Remember that abstract class names, interface names, and abstract methods should be italicized!
    - In UMLet: /this is italicized/
- You do NOT need to identify in the UML diagram that a method declares an exception (or mention exceptions at all).
- You do NOT need to diagram GameBoard, GameFileException, GameState, or Oblivion!

Programming Assignments (85%)

GameBoard.java

This assignment will utilize GameBoard from the previous assignment. You must include the correct import line in any code that relies on GameBoard:

import edu.*sitnet*.assign04.GameBoard;

…where *sitnet* is your SITNET ID in lowercase. **If GameBoard was NOT working in the previous assignment, it MUST pass the tests for GameBoard now!**

GameFileException.java

Create a java file with a public class GameFileException. **GameFileException INHERITS from Exception.** The purpose of this class is to indicate when there is a problem loading the game files that contain information about items and creatures. ***GameFileException will NOT need to have any instance data of its own.*** It will have the following public methods:

- **public GameFileException(String message)**
  - Call super(message) in this constructor to take advantage of Exception's constructor.
- **public GameFileException(String message, Exception e)**
  - Call super(message, e) in this constructor to take advantage of Exception's constructor.

Loadable.java

Create a java file with a public ***interface*** Loadable. It contains ONE public ***abstract*** method:

- ***public abstract void load(Scanner input) throws GameFileException;***

Drawable.java

Create a java file with a public ***interface*** Drawable. It contains ONE public ***abstract*** method:

- ***public abstract void draw(GameBoard map);***

Item.java

Create a java file with a public class Item. **Item IMPLEMENTS the interface Loadable**. An Item holds a String ID (which defaults to an empty String) and an int value (which defaults to zero). This class will have the following public methods:

- **public Item()**
  - If you have not set the ID to "" and the value to zero, do so now.
- **public Item(String ID, int value)**
  - Store ID and value in the class.
- **public String getID()**
  - Return the ID.
- **public int getValue()**
  - Return the value.
- **public void setID(String ID)**
  - Store the ID.
- **public void setValue(int value)**
  - Store the value.
- **public String toString()**
  - Return the ID + " with value " + value
  - Example: if ID = "SPOON" and value = 104, return "SPOON with value 104"

- **public void load(Scanner input) throws GameFileException**
    - In a try block:
        - Scanner input ALREADY has been created. Someone is giving you a Scanner object to read from.
        - Read in the ID using input.next().
        - Read in the value using input.nextInt().
    - Catch Exception e
        - Set ID back to “”
        - Set value back to 0.
        - Throw a new GameFileException with message “Error loading Item” and **include the original Exception e.**

Tome.java

Create a java file with a public class Tome. **Tome INHERITS from Item**. Tome stores the skill (as a String that defaults to “”) that reading the Tome will improve (e.g., if the skill is “sorcery”, reading the Tome will improve your sorcery skills). ***The ONLY instance data this class should define itself is the skill; it should NOT have its own ID and value data (instead relying on its lineage with Item).*** This class will have the following public methods:

- **public Tome()**
    - If the skill has not be set to “”, do so now.
- **public Tome(String ID, int value, String skill)**
    - Call the super constructor to set ID and value.
    - Store the skill in this class.
- **public String getSkill()**
    - Return the skill.
- **public void setSkill(String skill)**
    - Store the skill.
- **public void read()**
    - Using System.out.println(), print “Skill “ + skill + “ increased!”
    - Example: if the skill is “cooking”, then print “Skill cooking increased!”
- **public String toString()**
    - Return the result from calling the superclass’s toString + “, enhances “ + skill.
    - Example: if the ID = “MAPS_101”, the value = 93, and the skill = “geography”, then return “MAPS_101 with value 93, enhances geography”
- **public void load(Scanner input) throws GameFileException**
    - In a try block:
        - Scanner input ALREADY has been created. Someone is giving you a Scanner object to read from.
        - Call super.load(input).
        - Read in the skill using input.next().

    - Catch Exception e
        - Set ID back to "" (remember you have access to setID()).
        - Set value back to 0 (remember you have access to setValue()).
        - Set skill back to "".
        - Throw a new GameFileException with message "Error loading Tome" and **include the original Exception e.**

Creature.java

Create a java file with a public ***abstract*** class Creature. **Creature IMPLEMENTS both Loadable and Drawable**. HOWEVER, since Creature is an abstract class, it only provides code for load(), leaving draw() to the child classes. Creature stores a current row and column for the position of the Creature (both are ints and both default to zero). This class will have the following **protected** methods:

- **protected Creature()**
    - If the row and column have not been set to zero, do so now.
- **protected Creature(int row, int col)**
    - Store the row and col in this class.

It will also contain the following public methods:

- **public int getRow()**
    - Returns the row.
- **public int getCol()**
    - Returns the column.
- **public void setRow(int row)**
    - Stores the row.
- **public void setCol(int col)**
    - Stores the column.
- **public void load(Scanner input) throws GameFileException**
    - In a try block:
        - Scanner input ALREADY has been created. Someone is giving you a Scanner object to read from.
        - Read in the row using input.nextInt().
        - Read in the col using input.nextInt().
    - Catch Exception e
        - Set row to zero.
        - Set col to zero.
        - Throw a new GameFileException with message "Error loading Creature" and **include the original Exception e**.

Rat.java

Create a java file with a public class Rat. **Rat INHERITS from Creature**. Because Creature did not define code for draw() (and Rat is a concrete class), Rat will have to do so now. **Rat does NOT store any instance data of its own! Instead, it relies on the data in Creature**. Also, this class does NOT have to provided code for load() (Creature has already taken care of that). This class will have the following public methods.

- **public Rat()**
  - Basically does nothing.
- **public Rat(int row, int col)**
  - Calls the superclass's constructor to set row and col.
- **public String toString()**
  - Returns "Rat at " + getRow() + "," + getCol().
  - Example: if the Rat is located at row = 6 and col = 7, then return "Rat at 6,7"
- **public void draw(GameBoard map)**
  - Call setPos on map to draw an 'R' at the row and col position of the Rat.

Skeleton.java

Create a java file with a public class Skeleton. **Skeleton INHERITS from Creature**. Because Creature did not define code for draw() (and Skeleton is a concrete class), Skeleton will have to do so now. **Skeleton does NOT store any instance data of its own! Instead, it relies on the data in Creature**. Also, this class does NOT have to provided code for load() (Creature has already taken care of that). This class will have the following public methods.

- **public Skeleton()**
  - Basically does nothing.
- **public Skeleton(int row, int col)**
  - Calls the superclass's constructor to set row and col.
- **public String toString()**
  - Returns "Skeleton at " + getRow() + "," + getCol().
  - Example: if the Skeleton is located at row = 6 and col = 7, then return "Skeleton at 6,7"
- **public void draw(GameBoard map)**
  - Call setPos on map to draw an 'S' at the row and col position of the Skeleton.

GameState.java

Create a java file with a public class GameState. **GameState implements Loadable**. This class contains a list of Loadable items as well as a GameBoard for the map (12 rows, 30 columns, default fill character a period '.'). This class will have the following public methods:

- **public Loadable createLoadable(String typeName) throws GameFileException**
  - This method is essentially a **factory method**.
  - If typeName is "Skeleton", "Rat", "Item", or "Tome", return a new object with the appropriate class.
  - Otherwise, throw a new GameFileException with the message "Unknown type: " + typeName
- **public void load(Scanner input) throws GameFileException**
  - Clear your map and list of Loadable objects
  - Use TOKEN-BASED methods to read from the Scanner.
  - Read in the number of lines in the file as an int.
  - For each line:
    - Get typeName using input.next()
    - Call createLoadable to get the Loadable object m
    - Call m.load(input)
    - Add m to your list of Loadable objects
    - If m is also Drawable, draw m on your GameBoard map
- **public String toString()**
  - The contents of the String returned should be as follows:
    - "MAP:\n"
    - baseMap.getBoardString() + "\n" (where baseMap is your GameBoard)
    - "CREATURES:\n"
    - Add the Strings for ONLY the Creatures in your list of Loadable objects (one on each line)
      - Format should be: "* " + c + "\n" (where c is each Creature)
    - "INVENTORY:\n"
    - Add the Strings for ONLY the Items in your list of Loadable objects (one on each line)
      - Format should be: "* " + item + "\n" (where item is each Item)

- **public void save(String filename) throws GameFileException**
  - In a try block:
    - Create a PrintWriter to open the file identified by *filename* for writing.
    - Use writer.print(this.toString()).
      - **NOTE: Using print(), not println().**
    - **Either use the try-with-resources syntax OR manually close the PrintWriter object!**

- Catch Exception e
  - Throw a new GameFileException with message "Failed to save file!" and **include the original exception e**.

Oblivion.java

This program has been provided for you.

First, the program creates a GameState object.

Then, program asks for a level filename. Once provided, it does the following:

- Creates a Scanner object to read from the file.
- Calls load() on the GameState object.
- If an error occurs, an exception is thrown, but the program should not crash. However, it stops reading the file, so you may not have all of the things (this is normal, so long as your output matches; see below).
- Prints out the contents of the GameState object.

The text files to load are included in a folder data under the assign06 project. This folder is also where output files will be written to.

For debugging purposes, you can uncomment the e.printStackTrace() at line 31 to see the full problem, BUT be sure to comment it again before running the tests!

Output of Oblivion: Level_1.txt (user input in blue, STDERR in red):

```
Enter level filename:
data/Level_1.txt
MAP:
.............................
.............................
.............................
..S..........................
.............................
.............................
............R................
.............................
.............................
.............................
.............................
.............................

CREATURES:
* Skeleton at 3,2
* Rat at 6,12
INVENTORY:
```

Output of Oblivion: Level_2.txt (user input in blue, STDERR in red):

```
Enter level filename:
data/Level_2.txt
MAP:
.............................
.............................
.............................
.............................
..S..........................
.............................
.............................
..............R..............
.............................
...R.........................
....S........................
.............................

CREATURES:
* Skeleton at 4,2
* Rat at 7,15
* Rat at 9,3
* Skeleton at 10,4
INVENTORY:
```

Output of Oblivion: Level_3.txt (user input in blue, STDERR in red):

```
Enter level filename:
data/Level_3.txt
MAP:
.............................
.............................
.............................
.............................
.............................
.............................
.............................
.............................
.............................
.............................
.............................
.............................

CREATURES:
INVENTORY:
* CUP with value 5
* TOME_OF_MAGIC with value 120, enhances sorcery
* FORK with value 7
```

Output of Oblivion: Level_4.txt (user input in blue, STDERR in red):

```
Enter level filename:
data/Level_4.txt
MAP:
.............................
.............................
.............................
.......S.....................
.............................
.............................
.............................
.........R...................
.............................
.............................
............R................
..S..........................

CREATURES:
* Rat at 10,12
* Skeleton at 3,7
* Rat at 7,9
* Skeleton at 11,2
INVENTORY:
* GROGNAK_COMIC with value 170, enhances courage
* SPOON with value 2
* TAO_OF_PROGRAMMING with value 25, enhances wisdom
```

Output of Oblivion: BadTome.txt (user input in blue, STDERR in red):

```
Enter level filename:
data/BadTome.txt
Game File Error: Error loading Tome
MAP:
.............................
.............................
.............................
.............................
.............................
.............................
.............................
.............................
.............................
.............................
............R................
.............................

CREATURES:
* Rat at 10,12
INVENTORY:
```

Output of Oblivion: BadCreature.txt (user input in blue, STDERR in red):

```
Enter level filename:
data/BadCreature.txt
Game File Error: Error loading Creature
MAP:
.............................
.............................
.............................
.......S.....................
.............................
.............................
.............................
.............................
.............................
.............................
............R................
.............................

CREATURES:
* Rat at 10,12
* Skeleton at 3,7
INVENTORY:
* GROGNAK_COMIC with value 170, enhances courage
```

Output of Oblivion: BadItem.txt (user input in blue, STDERR in red):

```
Enter level filename:
data/BadItem.txt
Game File Error: Error loading Item
MAP:
.............................
.............................
.............................
.......S.....................
.............................
.............................
.............................
.........R...................
.............................
.............................
............R................
.............................

CREATURES:
* Rat at 10,12
* Skeleton at 3,7
* Rat at 7,9
INVENTORY:
* GROGNAK_COMIC with value 170, enhances courage
```

Testing Screenshot (5%)

Submit a screenshot showing the results of running the test program(s).

Grading

Your OVERALL assignment grade is weighted as follows:

- 5% - Testing results screenshot
- 10% - UML diagram
- 85% - Programming assignments