You will be creating a Harry Potter Dice Game. More specifically, you will be creating a class that will make the programming of the game much easier.

The flow of the game is as follows:

1. Both players begin with 50 health points.

2. Both players take turns attacking and defending by rolling a die. Player 1 begins the attack and Player 2 begins the defense first.

3. Player 1 rolls a die to attack. The number on the die determines which Offensive Spell can be cast.

4. Player 2 rolls a die to defend. The number on the die determines which Defensive Spell can be cast.

5. Health of the players is adjusted to account for any damage.

6. It is now Player 2's turn and he/she rolls a die to select an Offensive Spell. Player 1 rolls a die to defend.

7. Health is adjusted to account for any damage.

8. This process repeats until a player's health is reduced to 0.

You will need to create a Player class, see below

| Player |
| --- |
| `String name`<br>`int health`<br>`String [] offense_spells = …(see starter code)` |
| `void setName(String name)` – sets Character name to n<br><br>`int roll()` -returns a ranomd number between 1-6<br><br>`String getName()` – returns Character's name<br><br>`void set_Health_OffensePlayer(int playerRoll, int opponentRoll)` – sets health based on outcome of play rolls. You will use the roll results as parameters<br><br>`void set_Health_DefensePlayer(int playerRoll, int opponentRoll)` – sets health based on outcome of play rolls. You will use the roll results as parameters<br><br>`String getOSpell(int playerRoll)` – returns the spell used by the offensive player<br><br>`String getDSpell(int playerRoll)` – returns the spell used by the defensive player<br><br>`int getHealth()` -- returns the character's health |

## Offensive Spells

| SPELL | DESCRIPTION | ROLL | DAMAGE & EFFECTS |
| --- | --- | --- | --- |
| **Confundo** | Confuses opponent | 1 | 8 |
| **Locomotor Mortis** | Locks opponents legs | 2 | 9 |
| **Stupefy** | Stuns opponent | 3 | 10 |
| **Expelliarmus** | Disarms opponent | 4 | 9 + opponent is not allowed to defend |
| **Incendio** | Fire attack | 5 | 12 |
| **Petrificus Totalus** | Immobilizes opponent | 6 | 12 |

## Defensive Spells

| SPELL | ROLL | EFFECT |
| --- | --- | --- |
| **Finite Incantum** | 3 | Defends any spell that roll 3 or under |
| **Episkey** | 4 | Reduces damage by 5 |
| **Protego** | 5 | The spell's effects are blocked and deflected back |
| **Expecto Patronum** | 6 | Defends any spell |

Examples

| |
|---|
| Offensive Player rolls 1<br>Defensive Player rolls 3<br>No change to health |
| Offensive Player rolls 4<br>Defensive Player rolls 6<br>Defense health decrease by 4 |
| Offensive Player rolls 2<br>Defensive Player rolls 4<br>Defense health decrease by 9 |

You will also need to make a main method where the game is played using these methods (see example)

NAME: _____ TOTAL: / 40

| CATEGORY | CRITERIA | < LEVEL 1<br>0 – 49% | LEVEL 1<br>50 – 59% | LEVEL 2<br>60 – 69% | LEVEL 3<br>70 – 79% | LEVEL 4<br>80 – 100% | MARK |
|---|---|---|---|---|---|---|---|
| **Knowledge and Understanding** | Demonstrates an understanding of how to create a Character & Dice class and how to use the methods included | • Demonstrates little or no understand-ing of how to create a Character & Dice class<br>**0-4.9** | • Demonstrates limited understand-ing of how to create a Character & Dice class<br>**5.0-5.9** | • Demonstrates some understand-ing of how to create a Character & Dice class<br>**6.0-6.9** | • Demonstrates considerable understand-ing of how to create a Character & Dice class<br>**7.0-7.9** | • Demonstrates thorough understand-ing of how to create a Character & Dice class<br>**8.0-10** | **/10** |
| **Thinking** | The program meets all the specifications required<br><br>Validates program to ensure the program produces correct results | • Program meets little or none of the required specifications<br>• Validates program with little or no success<br>**0-4.9** | • Program meets a limited number of the required specifications<br>• Validates program with limited success<br>**5.0-5.9** | • Program meets some of the required specifications<br>• Validates program with some success<br>**6.0-6.9** | • Program meets most of the required specifications<br>• Validates program with considerable success<br>**7.0-7.9** | • Program meets all of the required specifications<br>• Validates program with great success<br>**8.0-10** | **/10** |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| **Communication** | Provides internal documentation that clearly explains the methods and the program logic | • Documents program logic with little or no success<br>**0-4.9** | • Documents program logic with limited success<br>**5.0-5.9** | • Documents program logic with some success<br>**6.0-6.9** | • Documents program logic with considerable success<br>**7.0-7.9** | • Documents program logic with great success<br>**8.0-10** | **/10** |
| **Application** | Effectively applies programming skills and knowledge of Java to create a program using a Character & Dice class and the String class | • Applies programming knowledge and skills with little or no success<br>**0-4.9** | • Applies programming knowledge and skills with limited success<br>**5.0-5.9** | • Applies programming knowledge and skills with some success<br>**6.0-6.9** | • Applies programming knowledge and skills with considerable success<br>**7.0-7.9** | • Applies programming knowledge and skills with great success<br>**8.0-10** | **/10** |

**CURRICULUM EXPECTATIONS THAT ARE COVERED IN THIS ASSIGNMENT:**

A3.1 Demonstrate the ability to use existing subprograms within computer programs.
A3.2 Write subprograms that use parameter passing and appropriate variable scope to perform tasks within programs.
A4.1 Demonstrate the ability to identify and correct syntax, logic and run-time errors in computer programs.
A4.2 Use workplace and professional conventions correctly to write programs and internal documentation.
A4.5 Demonstrate the ability to validate a program using test cases.
B2.1 Design programs from a program template or skeleton.
B2.3 Apply the principle of modularity to design reusable code *(e.g., subprograms, classes)* in computer programs.
B2.4 Represent the structure and components of a program using industry-standard programming tools.