**Creating an interactive text-based game using Java**

**Interaction summary**:

1. Game is introduced.
2. User is prompted to input their name or username.
3. Indicates what monster the user can get in the battle with.
4. Indicates the player’s and monster’s health, weak attack and strong attack
5. The user is given four choices: This required the user to input a number to match the action they want to take.
6. Weak attack
7. Strong attack
8. Next monster
9. Exit game
10. Attacks: Attacks the monster with lower or higher power depending on which one is selected. The player also gets attacked back in retaliation by the monster. Shows how much damage was given and how much damage was received. Health and attacks are indicated after each attack has been done. Strong attack gives a maximum of 20 in addition to the weak attack and can only be executed after weak attack has been executed twice.
11. Next monster: Skips the current monster and goes to the next one to start the battle.
12. Exit game: Exists the game and program.
13. The player can win or lose depending on how much attack they are giving in comparison to receiving. When health hits zero for either player or monster, the game is over. The winner is indicated at the end based on who defeated who.

**Technical summary:**

* **Main**:
* Starts the game using the constructor from Game class.
* **Game**:
* Scanner field is created to take in user input for their name and their action choice.
* Random field is created to generate random number for the players and monsters to be selected. They are selected from arraylist created for them. It is also used to generate a number for weak and strong attacks caused by both the fighters.
* Constructor is used to call initialise and start methods.
* Method called initialise is used to introduce the game and to prompt the user to input their name. It then calls the method called addPlayer which creates objects for players of different health and attack numbers using the user’s name. It also calls the method called addMonster which creates objects for monsters.
* Method called start is used to start the game cycle/loop. It creates randomly generated fighter by using createFighter method and prints it using printFighters. It also prints the actions the user can take by using the printAction method. While the game is still being played, the user can keep inputting their choice of what type of attack they want to do and whether they want to fight this monster or the next one. They can also exit the game. At the end of the loop, it prints the player who won the game with more than 0 health using the defeatedFighter method.
* Method called defeatedFighter is created to indicate the winner and the loser of the game. It is based on whether the fighter is alive or not. This works with boolean field being used from the Fighter class.
* Method called createFighters is used to create the player and fighter based on random number generation. It uses the fighter from the arraylist based on the random index being inputted in the get method. It also indicates which two fighters can start a battle.
* Method called printFighters is used to print the values from the toString method overridden in the Fighter class. It prints out the details for both the player and the monster.
* Method called printAction is used to print the options the user has regarding the game play. It includes choices like weak attack, strong attack, next monster and exit game.
* Method called addPlayer creates objects of the Player class where each object has a different health and different attacks but the same name. This is so that the player is given a variety at random when they start a game. All of these are added to the arraylist called players.
* Method called addMonster creates objects of the Monster class where each object indicates the different monsters created. Each of them has a different name, health and attacks. All of these are added to the arraylist called monsters.
* **Fighter**:
* Fighter is the superclass to Player and Monster. It contains the standard common behaviour expected from both fighters. It includes the name, health, indication of the fighter being alive or dead, weak attack, strong attack and the counter for weak attack.
* In the constructor, fields like name, health and the initial weak attack are initialised each time the object is created. The weak attack is based on random number generated in addition to 5 indicating 5 as the minimum number. The weak attack is then used to work out the strong attack by adding a random number upto 20. Also, the fighter’s health is checked to see if it is alive or not. If the health is above 0, then the fighter is alive.
* Getters are used to encapsulate fields that belong to the class. Values of the fields can be used this way.
* Method called damage is created which works out the value of how much damage a fighter would take if they are attacked. It is based on the calculation of attack being subtracted from the health. It also ensures that the health does not become negative as any health less than 0 is indicated as 0 health.
* Method called doWeakAttack is created which works out how much damage is caused by a weak attack. It also includes a counter for the weak attack. This counter is set to allow the player to use a strong attack once the weak attack has been used twice.
* Method called doStrongAttack is created which works out how much damage is caused by a strong attack. It checks to see if the weak attack has been executed twice and then allows the strong attack to be executed. It also sets the counter to 0 as each time a strong attack is used, it needs regeneration of two weak attacks to be used again.
* Method called toString is overridden to display details about the fighter.
* **Player**:
* Player is the subclass of Fighter as it inherits from it.
* **Monster**:
* Monster is the subclass of Fighter as it inherits from it.