# AGE OF GLADIATORS


[redacted]

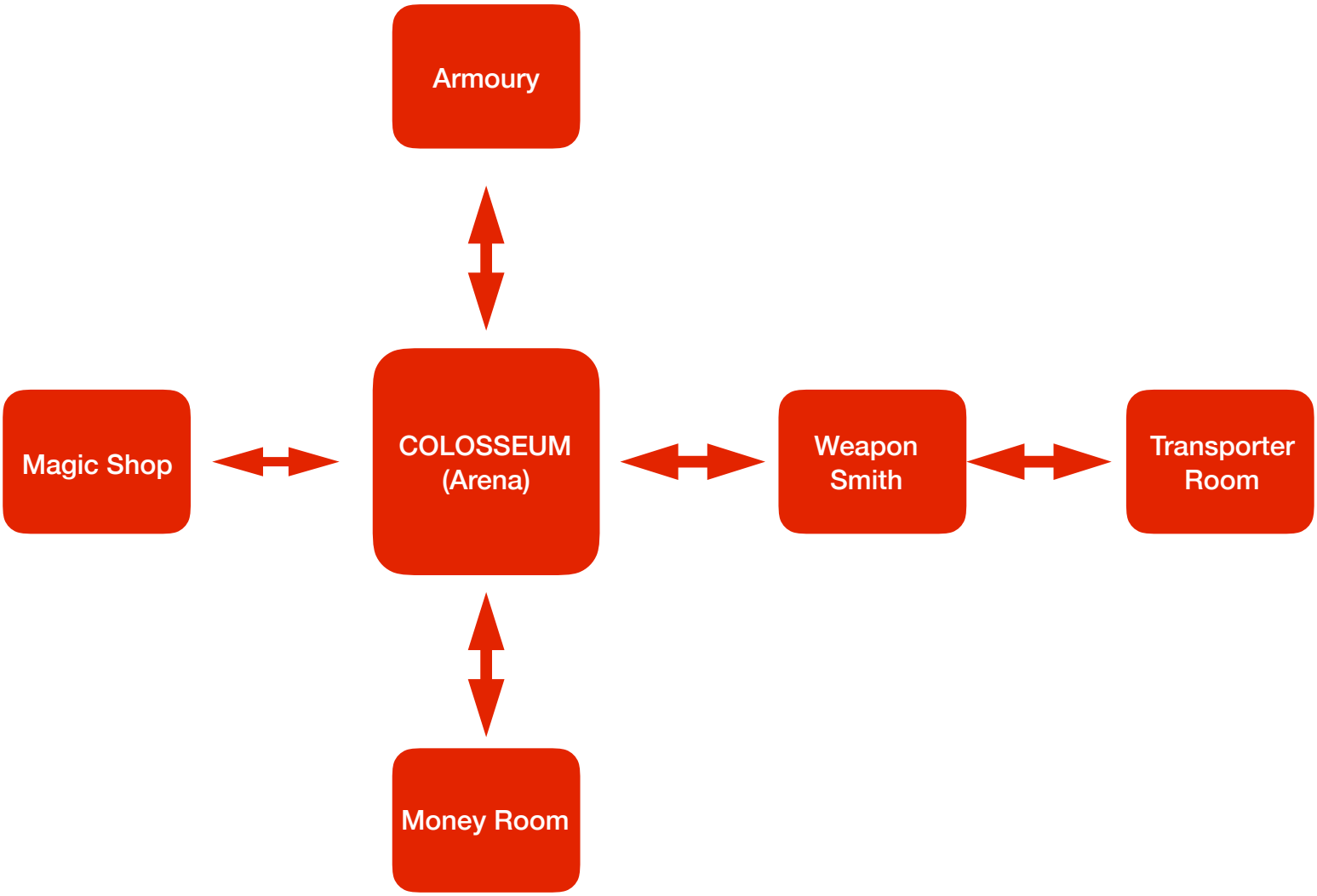
[redacted]

By - Pranav Bheemsetty

Student Number - [redacted]

# Table of Contents

# Game Summary




Age of Gladiators is a simple text-based - game.

The player needs to win battles against his opponent in the Colosseum.

In order to finish the game and emerge victorious , the player must collect two spells , five armours , five weapons and win a total of 34 battles .

# Functionality

The game currently has 6 rooms - Colosseum , Money Room, Magic Shop, Armoury, Weapon Smith & the Transporter Room. All the rooms have their very own exits . If the player tries to go in another direction , it is prompted that there is no room.

There are four characters in total - Gold Smith , Weapon Smith , Armour Smith & the Magician. You can speak to these characters only if you are in only in certain specific rooms.

## Colosseum Functionality

The game starts with the player currently in the Colosseum. The player is given an initial energy of 10 (stored in the playerEnergy variable). The opponent is given a random energy (stored in the opponentEnergy) of range 1 to 25 .The battle is started when the player types the command word ‘battle’. If the energy of the player is greater than the energy of the opponent , the outcome is a win else its a loss. The battle wins are stores in the ‘battlesWon’ and 'battlesToRedeem’ variable. The player can also check how many battles he has won , if he has lost a track of it .

The player can also type the command ‘progress’ to check what else he needs to do in order to win the game.

If the player feels lost or wants to check his next move , he can type in the command ‘map’ to see a pictorial representation of all the rooms along with the directions.

## Money Room Functionality

With reference to the exits shown , the player can go to the Money Room and redeem the battles won . When he enters money room , he is first greeted by the Gold Smith. In order to gain more understanding about the functionality of the room , the player is asked to type in the command work ‘speak’ . The Gold Smith then gives him the necessary set of instructions that can be performed in the specific room .

When the ‘redeem’ command is typed the player gets a total of 5 gold coins (stored in the ‘balance’ variable) for reach battle won . The ‘battlesToRedeem' variable is then decremented by 1 .If the player wants to have a larger balance , he can go back to the Colosseum and use the ‘battle’ command again to win and increase his balance.

## Armour Room , Weapon Smith and Magic Shop Functionality

After redeeming the battles for Gold Coins , the player can then either go to the Magic Shop , Weapon Smith or the Armour rooms to collect items for battle .In-order to collect a weapon or an armour the player needs to have a balance of  10 gold coins , and for a spell the player needs to have a balance of 15 gold coins.

Before , typing in any command , the player once again needs to type in the word ‘speak’ to the respective characters present in the room .

In-order to collect a weapon the player needs to type in the pick-weapon command. If he wants to collect an armour he needs to type in the pick-armour command. All the weapons and armours are picked in a linear order from the characters .

If the player has a sufficient balance and he collects an armour or a weapon , the ‘balance’ is decremented by 10 , the ‘weapon/armourWeight’ is incremented by 1, the ‘progress’ variable is incremented by 2.94 %, the ‘playerEnergy' is incremented by 2 and the ‘opponentEnergy’ is incremented by 1.

In-order to collect a weapon , the player needs to type in the pick-spell command. A spell is picked from a total of 5 random spells

If the player has a sufficient balance and he collects an spell, the ‘balance’ is decremented by 15 , the ‘spellWeight’ is incremented by 1, the ‘progress’ variable is incremented by 2.94 %, the ‘playerEnergy' is incremented by 3 and the ‘opponentEnergy’ is incremented by 1

The player can exit and enter any number of times to collect items from these rooms. The player can have a maximum spellWeight of 2 and weapon/armourWeight of 5 each.

## Transporter Room Functionality

The player can access transporter room only from the weapon smith . The main functionality of this room is to magically transport you to some random room or even the Colosseum in the game. In-order to start the the player needs to type in the word ‘generate’ .

| Command Word Usage | Room |
| --- | --- |
| help, go, quit, map , back, balance | All rooms |
| generate | Transporter |
| speak | All rooms except Colosseum and Transporter |
| available-weapons | Weapon Smith |
| redeem | Money Room |
| battle, battle-again,battles-won | Colosseum |
| show-stats | Magic Shop, Weapon Smith & Armour |
| pick-weapon | Weapon Smith |
| pick-armour | Armour |
| pick-spell | Magic Shop |

Note - All the base and challenge tasks have been implemented

# SOURCE CODE Class - Game

```
/**
 * This class is the main class of the "Age of Gladiators" application
 * "Age of Gladiators" is a very simple, text based adventure game .Users can walk
around
 * rooms , pick weapons , armours , spells and increase their chances of winning
 * battles in the colosseum.
 *
 * To play the game, create an instance of this class and call the "play" method.
 *
 *This main class creates and initialises all the others : it creates all the rooms,
 *creates the parser and starts the game. It also evaluates and executes the
commands theat the
 *parser returns.
 *
 */

import java.util.Random;
import java.util.*;

public class Game
{
    private Parser parser;
    private Room currentRoom, nextRoom;
    private int
balance,r,opponentEnergy,weaponWeight,spellWeight,armourWeight,battlesWon,bat
tlesToRedeem,roomCounter,minEnergy,maxEnergy;
    private float progress;
    private boolean speak = false;
    String direction;
    private Room colosseum , armoury , weaponSmith , moneyRoom , magicShop ,
transporter;
    private String commandWord;
    private Character playerTalk = new Character();
    private int playerEnergy = 10;
    private Item link = new Item();
    private Item warItems = new Item();
    private ArrayList<Room>roomRecord = new ArrayList<>();

    /**
     * Create the game and initialise its internal map.
     */
    public Game()
    {
        createRooms();
        parser = new Parser();

    }
```

```
    /**
     * Create all the rooms and link their exits together.
     */

    private void createRooms()
    {

        // create the rooms
        // initialise room exits

        colosseum = new Room("in the Colesseum");
        armoury = new Room("in the Armoury Shop");
        weaponSmith = new Room("with the Weaponsmith");
        moneyRoom = new Room("in the Money Room");
        magicShop = new Room("in the Magic Shop");
        transporter = new Room("in the Transporter Room");

        // Map - The colosseum is in the center.
        // Go to the 'map' method to see a pictorial view of all tthe rooms and their

        // Colosseum arena exits
        colosseum.setExit("|West|", magicShop);
        colosseum.setExit("|South|", moneyRoom);
        colosseum.setExit("|East|", weaponSmith);
        colosseum.setExit("|North|",armoury);

        // Armoury room exits
        armoury.setExit("|South-to-C|", colosseum);  // The letter 'C' in the exit "|South-
to-C|"stands for colosseum
        armoury.setExit("|Arena-South|",moneyRoom);  // The above exit means that
the moneyRoom is in the north direction of the colosseum. The same goes for
similar exits which have the word 'Arena'
        armoury.setExit("|Arena-West|",magicShop);
        armoury.setExit("|Arena-East|",weaponSmith);

        // Weaponsmith room exits
        weaponSmith.setExit("|West-to-C|", colosseum); // The letter 'C' in the exit "|
West-to-C|"stands for colosseum
        armoury.setExit("|Arena-South|",moneyRoom);
        weaponSmith.setExit("|Arena-North|", armoury);
        weaponSmith.setExit("|Arena-South|",moneyRoom);
        weaponSmith.setExit("|Arena-West|",magicShop);
        weaponSmith.setExit("|East-to-T|",transporter); // The letter 'T' in the exit "|
East-to-T|"stands for Transporter

        // Transporter exits
        transporter.setExit("|West-to-W|",weaponSmith); // The letter 'C' in the exit "|
West-to-W|"stands for weapon Smith
```

```
        // Money room exits
        moneyRoom.setExit("|North-to-C|", colosseum);
        moneyRoom.setExit("|Arena-North|", armoury);
        moneyRoom.setExit("|Arena-West|",magicShop);
        moneyRoom.setExit("|Arena-East|",weaponSmith);

        // Magic shop exits
        magicShop.setExit("|East-to-C|",colosseum);
        magicShop.setExit("|Arena-North|", armoury);
        magicShop.setExit("|Arena-South|",moneyRoom);
        magicShop.setExit("|Arena-East|",weaponSmith);

    }

    /**
     *  Main play routine.  Loops until end of play.
     */
    public void play()
    {
        printWelcome();

        // Enter the main command loop.  Here we repeatedly read commands and
        // execute them until the game is over.

        boolean finished = false;
        while (! finished) {
            Command command = parser.getCommand();
            finished = processCommand(command);

        }
        System.out.println("Thank you for playing.  Good bye.");
    }

    /**
     * Print out the opening message for the player.
     */
    private void printWelcome()
    {
        System.out.println();
        System.out.println("Welcome to the age of Gladiators");
        System.out.println("You need to win battles in order to emerge victorious");
        System.out.println("You will have an initial energy of 10 . Your opponent will
have a energy range between 1 to 25 (Randomly decided)");
        System.out.println("Increse your energy by collecting weapons , aromour and
spells");
        System.out.println("Type 'help' if you need help.");
        System.out.println("NOTE - COMMANDS ARE CASE-SENSITIVE . TYPE
CAREFULLY!. Type the word 'go' before you decide to exit . eg 'go |North|'." );
        System.out.println("Type 'progress' to check your progress in the game ");
```

```
    currentRoom = colosseum;  // start game outside
    System.out.println(currentRoom.getLongDescription());

    System.out.println();
    map();

    colosseum();


}

/**
 * Given a command, process (that is: execute) the command.
 * @param command The command to be processed.
 * @return true If the command ends the game, false otherwise.
 */
private boolean processCommand(Command command)
{
    boolean wantToQuit = false;

    if(command.isUnknown()) {
        System.out.println("I don't know what you mean...");
        return false;
    }

    commandWord = command.getCommandWord();
    if (commandWord.equals("help")) {
        printHelp();
    }
    else if (commandWord.equals("go")) {
        goRoom(command);
    }

    else if (commandWord.equals("quit")) {

        wantToQuit = quit(command);
    }

    else if(commandWord.equals("map")){
        map();

    }

    else if(commandWord.equals("back")){
        back();

    }
    else if (commandWord.equals("balance")){
        System.out.println("Your current balance is :"+ balance);
```

```
        }
        else if (currentRoom == transporter && commandWord.equals("generate")){
            transporter();
        }

        else if (currentRoom == moneyRoom && commandWord.equals("speak") ){
            playerTalk.speakToGoldSmith();

        }

        else if (currentRoom == armoury && commandWord.equals("speak")){
            playerTalk.speakToArmourSmith();

        }

        else if (currentRoom == weaponSmith && commandWord.equals("speak")){
            playerTalk.speakToWeaponSmith();
        }

        else if (currentRoom == weaponSmith && commandWord.equals("available-
weapons")) {
            warItems.weaponInventory();
        }

        else if (currentRoom == magicShop && commandWord.equals("speak")){
            playerTalk.speakToMagician();

        }

        else if(commandWord.equals("progress")){
            progressStats();

        }

        else if (currentRoom == moneyRoom && commandWord.equals("redeem")){
            if(battlesToRedeem > 0){

                battlesToRedeem -= 1;
                balance += 5;
                System.out.println("Your current balance is :" + balance);
                System.out.println("You can redeem a total of " + battlesToRedeem + "
battles.");
            }

            else {

                System.out.println("Go to the Colosseum and battle again . You can't
redeem anymore :(");
```

```
            }

        }

        else if(currentRoom == colosseum && (commandWord.equals("battle") ||
commandWord.equals("battle-again"))){
            energyLevels();

            if(spellWeight == 2 && armourWeight == 5 && weaponWeight ==5 &&
battlesWon == 34){

                System.out.println("You have won the tournament. Congratulations ! You
should feel proud of your self.");
                System.out.println("Type 'quit' to close the game");
            }

        }

        else if((currentRoom == colosseum || currentRoom == moneyRoom ) &&
commandWord.equals("battles-won")){
            System.out.println("You have won a total of " + battlesToRedeem + "
battles.");


        }

        else if ((currentRoom == armoury || currentRoom == weaponSmith ||
currentRoom == magicShop)  && commandWord.equals("show-stats")){
            energyStats();
        }

        else if (currentRoom == weaponSmith && commandWord.equals("pick-
weapon")){

            pickWeapon(command);

        }

        else if (currentRoom == armoury && commandWord.equals("pick-armour")){
            pickArmour(command);

        }
        else if (currentRoom == magicShop && commandWord.equals("pick-spell")){
            pickSpell(command);

        }

        // else command not recognised.
        return wantToQuit;
```

```
    }


    /**
     * Shows a pictorial representian of the directions to room .
     */
    public void map(){
        System.out.println("                    ARMOURY"
);
        System.out.println("                      ↑
North           ");
        System.out.println("               ------------------
↑            ");
        System.out.println("   MAGIC SHOP ← |COLOSSEUM (Arena)| → WEAPON
SMITH → TRANSPORTER ROOM                  West ← COMPASS → East          ");
        System.out.println("               ------------------
↓            ");
        System.out.println("                      ↓
South               ");
        System.out.println("                  MONEY ROOM
");

    }


    /**
     * Shows statistics of items presesnt in other rooms . (Such as weapons , armour
and spell)
     */
    public void energyStats(){
        link.increaseInEnergyLevels();

    }

    /**
     * Shows your current progress in the game.
     */

    public void progressStats(){
        System.out.println("You need to win a total of 10 battles and have a spell
weight of 2 and weapon/armour weight of 5 each");
        System.out.println();
        System.out.println("Your current progress in the game " + progress +"
percent");
        System.out.println();
        System.out.println("Your current weapon weight is :" + weaponWeight);
        System.out.println("Your current armour weight is :" + armourWeight);
        System.out.println("Your current spell weight is  :" + spellWeight);
```

```
        System.out.println("You have won a total of " + battlesWon +" battles");

    }


    /**
     * Print out some help information.
     * Here we print some stupid, cryptic message and a list of the
     * command words.
     */
    private void printHelp()
    {

        System.out.println();
        System.out.println("Type 'map' to see the directions you can go");
        System.out.println("The exits for each room are provided when you first enter
the room");
        System.out.println();
        System.out.println("Your command words are:");
        parser.showCommands();
        System.out.println("Note :- Some of the commands will work only in specific
rooms.");
    }

    /**
     * Takes you back to the previousRoom
     */

    public void back(){
        if(roomCounter <= 0){

            System.out.println("You can't go further back.");

        }
        else{
            currentRoom = roomRecord.get(--roomCounter);
            System.out.println(currentRoom.getLongDescription());
        }

    }

    /**
     * Try to go in to one direction. If there is an exit, enter the new
     * room, otherwise print an error message.
     */

    private void goRoom(Command command)
    {
        if(!command.hasSecondWord()) {
```

```
            // if there is no second word, we don't know where to go...
            System.out.println("Go where?");
            return;
        }

        direction = command.getSecondWord();

        // Try to leave current room.
        nextRoom = currentRoom.getExit(direction);

        if (nextRoom == null) {
            System.out.println("There is no room!");
        }
        else {
            roomCounter ++;
            roomRecord.add(currentRoom);
            currentRoom = nextRoom;

            System.out.println(currentRoom.getLongDescription());
            if(currentRoom == transporter){
                System.out.println("Welcome to the Transporter room. You will get
transported to a random location based on the random number generated.");
                System.out.println("Type 'generate' to get started !");
            }

            else if(currentRoom == moneyRoom){
                System.out.println("Welcome to the Money Room. If you want to exit the
room , use the above available commands.");
                System.out.println("Type 'speak' to talk to the goldsmith");

            }

            else if (currentRoom == armoury){
                System.out.println("Welcome to the Armoury Shop.If you want to exit the
room , use the above available commands.");
                System.out.println("Type 'speak' to talk to the armourysmith");
            }

            else if(currentRoom == magicShop){
                System.out.println("Welcome to the Magic Shop.If you want to exit the
room , use the above available commands.");
                System.out.println("Type 'speak' to talk to the magician");

            }

            else if (currentRoom == weaponSmith){
                System.out.println("Welcome to the Weapon Shop.If you want to exit the
room , use the above available commands.");
                System.out.println("Type 'speak' to talk to the weaponsmith");
```

```
            }
            else if(currentRoom == colosseum){

                colosseum();

            }
        }
    }

    /**
     * Its picks a weapon , if you have the necessary balance .
     * With every weapon you pick , the method deducts the balance by 10 gold
coins , increases the player energy by 2 ,opponent energy by 1,
     * progress by 2.94 % and weapon weight by 1
     *
     * If  the weapon weight is equal to 5 , you cannot pick any further.
    **/

    public void pickWeapon(Command command){
        if(balance >= 10){
            balance -=10;
            playerEnergy += 2;
            opponentEnergy += 1;
            progress += 2.94 ;

            if(weaponWeight == 5){
                System.out.println("You have reached your maximum capacity. You
cannot carry anymore weapons.");
            }

            else {
                System.out.println("The current items in your inventory are:");
                warItems.addWeapon();
                warItems.myInventory();
                weaponWeight += 1;
                System.out.println();
                System.out.println("Your current weapon weight is " + weaponWeight);

            }

            System.out.println();

            if(warItems.getList1().isEmpty()){
                System.out.println("The weapon smith doesn't have any weapons left.
Please exit the room");
            }
            else{
                System.out.println("The weapons available with the weaponSmith are :");
```

```
            warItems.weaponInventory();
        }

        System.out.println();
        System.out.println("Your current balance is " + balance + " gold coins");
    }

    else{
        System.out.println("You either don't have the balance to redeem a weapon
or the weaponsmith does'nt have any left ! Go to the Colosseum again and win
battles or try another room");
    }

}


/**
 * Its picks a armour , if you have the necessary balance .
 * With every armour you pick , the method deducts the balance by 10 gold
coins , increases the player energy by 2 ,opponent energy by 1,
 * progress by 2.94 % and armour weight by 1
 *
 * If  the armour weight is equal to 5 , you cannot pick any further.
**/

public void pickArmour(Command command){
    if(balance >= 10){
        balance -=10;
        playerEnergy += 2;
        opponentEnergy += 1;
        progress += 2.94 ;

        if(armourWeight == 5){
            System.out.println("You have reached your maximum capacity. You
cannot carry anymore armour.");
        }

        else {
            System.out.println("The current items in your inventory are:");
            warItems.addArmour();
            warItems.myInventory();
            armourWeight += 1;
            System.out.println();
            System.out.println("Your current armour weight is " + armourWeight);
        }

        System.out.println();
        if(warItems.getList2().isEmpty()){
```

```
            System.out.println("The armour smith doesn't have any armour left.
Please exit the room");
        }
        else{
            System.out.println("The armour available with the armourSmith are :");
            warItems.armourInventory();
        }

        System.out.println();
        System.out.println("Your current balance is " + balance + " gold coins");
    }

    else{
        System.out.println("You either don't have the balance to redeem a armour or
the weaponsmith does'nt have any left ! Go to the Colosseum again and win battles
or try another room");
    }

}

/**
 * Its picks a spell at random , if you have the necessary balance .
 * With every spell you pick , the method deducts the balance by 15 gold coins ,
increases the player energy by 3 ,opponent energy by 1,
 * progress by 2.94 % and spell weight by 1
 *
 * If the spell weight is equal to 2 , you cannot pick any further.
**/


public void pickSpell(Command command){
    if(balance >= 15){
        if(spellWeight == 2){
            System.out.println("You have reached your maximum capacity. You
cannot carry anymore spells.");
        }

        else {

            balance -=15;
            playerEnergy += 3;
            opponentEnergy += 1;
            progress += 2.94 ;

            System.out.println("The current items in your inventory are:");
            warItems.addSpell();
            warItems.myInventory();
            spellWeight += 1;
            System.out.println();
```

```
            System.out.println("Your current spell weight is " + spellWeight);
        }

        System.out.println();
        if(warItems.getList3().isEmpty()){
            System.out.println("The magician doesn't have any spells left. Please exit
the room");
        }
        else{
            System.out.println("The spells available with the magician are :");
            warItems.spellInventory();
        }

        System.out.println();
        System.out.println("Your current balance is " + balance + " gold coins");
    }

    else{
        System.out.println("You either don't have the balance to redeem a spell or
the weaponsmith doesnt have any left or you have reached your maximum spell
weight. ! \n Go to the Colosseum again and win battles or try another room");
    }

}

/**
 * Sends you to a random room when the current room is the transporter
 *
 */
public void transporter(){
    int min = 1;
    int max = 5;

    Random rand = new Random();

    r = rand.nextInt(max-min) + min;
    System.out.println("Room number generated : " + r);

    if(r == 1){
        currentRoom = armoury;
        playerTalk.speakToArmourSmith();

    }
    else if (r == 2){
        currentRoom = moneyRoom;
        playerTalk.speakToGoldSmith();
    }
    else if(r == 3){
        currentRoom = magicShop;
```

```
            playerTalk.speakToMagician();
        }
        else if (r == 4){
            currentRoom = weaponSmith;
            playerTalk.speakToWeaponSmith();
        }
        else {
            currentRoom = colosseum;

        }
        System.out.println(currentRoom.getLongDescription());

    }

    /**
     *
     * Instructions to be shown when the current room is the colosseum
     */
    public void colosseum(){
        System.out.println();
        System.out.println("Welcome to the main battle arena - THE COLOSSEUM. If
you want to exit the room , use the above available commands." );
        System.out.println("Type 'map' to show your current location");
        System.out.println("You will be fighting with top gladiators from all over the
Roman Empire. The outcome of the battle is decided by energy of the player");
        System.out.println("If your energy levels are higher than your opponent, you
win , else you loose");
        System.out.println("For every battle you win, you can redeem 4 gold coins ");
        System.out.println("Type 'battle' to start the battle");
    }


    /**
     *
     * Logic for the battle in the colosseum
     * The opponent's energy is generated by random based on the range .
     * If the player's energy is less than the opponent's energy , you then loose , else
you win
     *
     */
    public void energyLevels(){
        minEnergy = 1;
        maxEnergy = 25;

        Random rand = new Random();

        opponentEnergy = rand.nextInt(maxEnergy-minEnergy)+minEnergy;
        System.out.println("Opponent Energy: " + opponentEnergy);
        System.out.println("Your energy:" + playerEnergy);
```

```
        if(playerEnergy > opponentEnergy){
            battlesToRedeem += 1;
            battlesWon +=1;
            progress += 2.94 ;

            System.out.println("You won !");
            System.out.println("Exit the arena if you want to redeem items. Type 'map'
to see where you want to go");

        }
        else {
            System.out.println("You lost !");

        }

        System.out.println("Type 'battle again' to try once more" );
        System.out.println("Type 'battles-won' to check your wins");
        System.out.println("Type 'progress' to check your progress in the game ");

    }

    /**
     * "Quit" was entered. Check the rest of the command to see
     * whether we really quit the game.
     * @return true, if this command quits the game, false otherwise.
     */
    private boolean quit(Command command)
    {
        if(command.hasSecondWord()) {
            System.out.println("Quit what?");
            return false;
        }
        else {
            return true;  // signal that we want to quit
        }
    }
}
```

# SOURCE CODE Class - Item

```
/**
 *  This class is part of the "Age of Gladiators" application.
 * "Age of Gladiators" is a very simple, text based adventure game.
 *
 * The Item class hold holds information of items present in different rooms (i.e
weapons , armour , spells , player inventory ).
 *
 */
import java.util.*;
import java.util.Random;

public class Item
{
    private ArrayList<String>weapons;   // stores types of weapons
    private ArrayList<String>armour;    // stores types of armour
    private ArrayList<String>myInventory; // stores items that are added to the player
inventory
    private ArrayList<String>magicSpells; // stores types of magic spells
    private Random randomGenerator;
    private int r;

    public Item(){
        weapons = new ArrayList<>();
        weapons.add("Bow & Arrow - weapon1");
        weapons.add("Sword - weapon2");
        weapons.add("Dagger- weapon3");
        weapons.add("Spear- weapon4");
        weapons.add("Trident- weapon5");

        armour = new ArrayList<>();
        armour.add("Sword Belt");
        armour.add("Arm Guard");
        armour.add("Leg Guard");
        armour.add("Breast Plate");
        armour.add("Steel Helmet");

        myInventory = new ArrayList<>();

        magicSpells = new ArrayList<>();
        magicSpells.add("Vitality Spell");
        magicSpells.add("Strength Spell");
        magicSpells.add("Agility Spell");
        magicSpells.add("Defence Spell");
        magicSpells.add("Stamina Spell");
        magicSpells.add("Attack Spell");
```

```
    }

    /**
     * Displays statistics of various weapons, armours and spells
     */
    public void increaseInEnergyLevels(){
        System.out.println();
        System.out.println("CAN REDEEM ONLY FROM WEAPON SMITH");
        System.out.println("1) Bow & Arrow - Need 10 gold coins | INCREASES
ENERGY BY 2 \n2) Sword - Need 10 gold coins |INCREASES ENERGY BY 2 \n3)
Dagger - Need 10 gold coins | INCREASES ENERGY BY 2 \n4) Spear - Need 10
gold coins | INCREASES ENERGY BY 2 \n5) Trident -  Need 10 gold coins |
INCREASES ENERGY BY 2 ");
        System.out.println("---------------------------------");
        System.out.println("CAN REDEEM ONLY FROM ARMOUR SMITH");
        System.out.println("6) Sword Belt - Need 10 gold coins | INCREASES ENERGY
BY 2 \n7) Arm Guard - Need 10 gold coins | INCREASES ENERGY BY 2 \n8) Leg
Guard - Need 10 gold coins |INCREASES ENERGY BY 2 \n9) Breast Plate - Need
10 gold coins | INCREASES ENERGY BY 2 \n10) Steel Helmet - Need 10 gold coins
| INCREASES ENERGY BY 2 ");
        System.out.println("---------------------------------");
        System.out.println("CAN REDEEM ONLY FROM MAGIC SHOP");
        System.out.println("11) Vitality Spell - Need 15 gold coins | INCREASES
ENERGY BY 3 \n12) Strength Spell - Need 15 gold coins | INCREASES ENERGY BY
3 \n13) Defence Spell- Need 15 gold coins |INCREASES ENERGY BY 3 \n14)
Stamina Spell - Need 15 gold coins | INCREASES ENERGY BY 3 \n15) Attack Spell
- Need 15 gold coins | INCREASES ENERGY BY 3 ");
        System.out.println();
    }

    /**
     * Returns the list of weapons
     */
    public ArrayList<String> getList1()
    {
        return weapons;
    }

    /**
     * Returns the list of armours
     */
    public ArrayList<String> getList2(){
        return armour ;
    }

    /**
     * Returns the list of magic spells
     */
```

```
public ArrayList<String> getList3(){
    return magicSpells ;
}

/**
 * Prints out all the items in the weapons inventory
 */
public void weaponInventory(){
    for (String weaponsInventory: weapons){
        System.out.println(weaponsInventory);
    }
}

/**
 * Prints out all the items in the armour inventory
 */
public void armourInventory(){

    for ( String armourInventory: armour){
        System.out.println(armourInventory);
    }

}

/**
 * Prints out all the items in the spellInventory()
 */
public void spellInventory(){
    for(String spellsInventory : magicSpells){
        System.out.println(spellsInventory);
    }

}

/**
 *
 * Add the first item from the weapon inventory to the player inventory.
 * Once added that item is removed from the weapon inventory
 */
public void addWeapon(){
    myInventory.add(weapons.get(0));
    weapons.remove(0);

}

/**
 *
 * Add the first item from the armour inventory to the player inventory.
 * Once added that item is removed from the armour inventory
```

```
     */

    public void addArmour(){
        myInventory.add(armour.get(0));
        armour.remove(0);

    }

    /**
     *
     * Add a randrom item from the spell inventory to the player inventory.
     * Once added that item is removed from the spell inventory
     */

    public void addSpell(){

        itemGenerator();

        if(magicSpells.contains(magicSpells.get(r))){

            String spell = magicSpells.get(r);
            myInventory.add(spell);
            magicSpells.remove(spell);
        }

        else{
            System.out.println("Try the 'pick-spell' command again");

        }

    }

    /**
     * Generates a random number , which is used to choose an item from the spell
inventory
     */

    public void itemGenerator(){

        int min = 0;
        int max = 4;
        Random rand = new Random();

        int r = rand.nextInt(max-min)+min;
        System.out.println("Item generated : " + r);

    }

    /**
```

```
     * Prints out all the items in the player inventory
     */

    public void myInventory(){
        for (String playerInventory: myInventory){
            System.out.println(playerInventory);
        }

    }

}
```

# SOURCE CODE  Class - Character

```
/**  This class is part of the "Age of Gladiators" application.
 * "Age of Gladiators" is a very simple, text based adventure game.
 *
 *
 * The Character class hold the following information :-
 *
 * There are total of 4 characters in the game Gold Smith , Weapon Smith , Money
Smith and Magician
 * When the player in a certain room and types the 'speak' command ,the character
present in the room , gives out the instrutions on how
 * to proceeed further
 */
public class Character
{
    Character goldSmith;
    Character weaponSmith;
    Character armourSmith;
    Character magician;

    public void Character(){
        goldSmith = new Character();
        weaponSmith = new Character();
        armourSmith = new Character();
        magician = new Character();

    }
```

```
    /**
     * When the player speaks to the gold smith , the below output is shown
     */
    public void speakToGoldSmith(){
        System.out.println();
        System.out.println(" GOLD-SMITH : Collect gold coins for battles won and
exchange them to upgrade your armour ,weapons and magic potions");
        System.out.println(" GOLD-SMITH : For each battle you win, you can redeem 5
gold coins");
        System.out.println(" GOLD-SMITH : Check your balance and redeem your
coins");
        System.out.println(" GOLD-SMITH : Type 'battles-won' to check how much
you can redeem ");
        System.out.println(" GOLD-SMITH : Type 'balance' to check the gold coins you
have ");
        System.out.println(" GOLD-SMITH : Type 'redeem' to redeem gold coins ");

    }

    /**
     * When the player speaks to the weapon smith , the below output is shown
     */

    public void speakToWeaponSmith(){
        System.out.println();
        System.out.println("WEAPON-SMITH: Redeem weapons for the gold coins you
have");
        System.out.println("WEAPON-SMITH: Each weapon will cost you a total of 10
gold coins");
        System.out.println("WEAPON-SMITH: Type 'balance' to check how many
weapons you can redeem");
        System.out.println("WEAPON-SMITH: Type 'show-stats' to see statistics of
items available");
        System.out.println("WEAPON-SMITH: Type 'pick-weapon' to add a weapon to
your inventory");
        System.out.println("WEAPON-SMITH: Type 'help' if you have any doubts");
        System.out.println();

    }

    /**
     * When the player speaks to the armour smith , the below output is shown
     */
    public void speakToArmourSmith(){
        System.out.println();
        System.out.println("ARMOUR-SMITH: Redeem an armour for the gold coins
you have");
        System.out.println("ARMOUR-SMITH: Each armour will cost you a total of 10
gold coins");
```

```
        System.out.println("ARMOUR-SMITH: Type 'balance' to check how many
armours you can redeem");
        System.out.println("ARMOUR-SMITH: Type 'show-stats' to see statistics of
items available");
        System.out.println("ARMOUR-SMITH: Type 'pick-armour' to add a armour to
your inventory");
        System.out.println("ARMOUR-SMITH: Type 'help' if you have any doubts");
        System.out.println();

    }

    /**
     * When the player speaks to the magician , the below output is shown
     */

    public void speakToMagician(){
     System.out.println();
        System.out.println("MAGICIAN: Redeem weapons for the gold coins you
have");
        System.out.println("MAGICIAN: Each weapon will cost you a total of 15 gold
coins");
        System.out.println("MAGICIAN: Type 'balance' to check how many spells you
can redeem");
        System.out.println("MAGICIAN: Type 'show-stats' to see statistics of items
available");
        System.out.println("MAGICIAN: Type 'pick-spell' to add a spell to your
inventory");
        System.out.println("MAGICIAN: Type 'help' if you have any doubts");
        System.out.println();

    }
}
```

# SOURCE CODE  Class - Room

```
import java.util.Set;
import java.util.HashMap;

/**
 * Class Room - a room in an adventure game.
 *
 * This class is part of the "Age of Gladiators" application.
```

```
 * "Age of Gladiators" is a very simple, text based adventure game.
 *
 * A "Room" represents one location in the scenery of the game.  It is
 * connected to other rooms via exits.  For each existing exit, the room
 * stores a reference to the neighboring room.
 *
 *
 */

public class Room
{
    private String description;
    private HashMap<String, Room> exits;        // stores exits of this room.

    /**
     * Create a room described "description". Initially, it has
     * no exits. "description" is something like "a kitchen" or
     * "an open court yard".
     * @param description The room's description.
     */
    public Room(String description)
    {
        this.description = description;
        exits = new HashMap<>();
    }

    /**
     * Define an exit from this room.
     * @parshowAllam direction The direction of the exit.
     * @param neighbor  The room to which the exit leads.
     */
    public void setExit(String direction, Room neighbor)
    {
        exits.put(direction, neighbor);
    }



    /**
     * @return The short description of the room
     * (the one that was defined in the constructor).
     */
    public String getShortDescription()
    {
        return description;
    }

    /**
     * Return a description of the room in the form:
```

```
     *     You are in the kitchen.
     *     Exits: north west
     * @return A long description of this room
     */
    public String getLongDescription()
    {
        return "You are " + description + ".\n" + getExitString();
    }

    /**
     * Return a string describing the room's exits, for example
     * "Exits: north west".
     * @return Details of the room's exits.
     */
    private String getExitString()
    {
        String returnString = "Exits:";
        Set<String> keys = exits.keySet();
        for(String exit : keys) {
            returnString += " " + exit;
        }
        return returnString;
    }

    /**
     * Return the room that is reached if we go from this room in direction
     * "direction". If there is no room in that direction, return null.
     * @param direction The exit's direction.
     * @return The room in the given direction.
     */
    public Room getExit(String direction)
    {
        return exits.get(direction);
    }
}
```

# SOURCE CODE Class - Command

```
/**
 * This class is part of the "Age of Gladiators" application.
 * "Age of Gladiators" is a very simple, text based adventure game.
 *
```

```
 * This class holds information about a command that was issued by the user.
 * A command currently consists of three strings: a command word , a second
 * word and a third word
 *
 * The way this is used is: Commands are already checked for being valid
 * command words. If the user entered an invalid command (a word that is not
 * known) then the command word is <null>.
 *
 */

public class Command
{
    private String commandWord;
    private String secondWord;
    private String thirdWord;

    /**
     * Create a command object. First and second word must be supplied, but
     * either one (or both) can be null.
     * @param firstWord The first word of the command. Null if the command
     *                  was not recognised.
     * @param secondWord The second word of the command.
     * @param thirdWord  The third word of the command
     */
    public Command(String firstWord, String secondWord, String thirdWord)
    {
        commandWord = firstWord;
        this.secondWord = secondWord;
        this.thirdWord = thirdWord;
    }

    /**
     * Return the command word (the first word) of this command. If the
     * command was not understood, the result is null.
     * @return The command word.
     */
    public String getCommandWord()
    {
        return commandWord;
    }

    /**
     * @return The second word of this command. Returns null if there was no
     * second word.
     */
    public String getSecondWord()
    {
        return secondWord;
    }
```

```
    /**
     * @return The third word of this command. Returns null if there was no
     * third word.
     */

    public String getThirdWord(){
        return thirdWord;
    }

    /**
     * @return true if this command was not understood.
     */
    public boolean isUnknown()
    {
        return (commandWord == null);
    }

    /**
     * @return true if the command has a second word.
     */
    public boolean hasSecondWord()
    {
        return (secondWord != null);
    }
}
```

# SOURCE CODE  Class - CommandWords

```
/**
 * This class is part of the "Age of Gladiators" application.
 * "Age of Gladiators" is a very simple, text based adventure game.
 *
 * This class holds an enumeration of all command words known to the game.
 * It is used to recognise commands as they are typed in.
 *
 *
 */

public class CommandWords
{
    // a constant array that holds all valid command words
    private static final String[] validCommands = {
```

```
            "go","back", "quit", "help" , "balance" , "generate" , "show", "redeem" ,
"map", "battle" , "battles-won","show-stats","speak","available-weapons", "pick-
weapon", "pick-armour" , "pick-spell","progress"
        };

    /**
     * Constructor - initialise the command words.
     */
    public CommandWords()
    {
        // nothing to do at the moment...
    }

    /**
     * Check whether a given String is a valid command word.
     * @return true if it is, false if it isn't.
     */
    public boolean isCommand(String aString)
    {
        for(int i = 0; i < validCommands.length; i++) {
            if(validCommands[i].equalsIgnoreCase(aString))
                return true;
        }
        // if we get here, the string was not found in the commands
        return false;
    }

    /**
     * Print all valid commands to System.out.
     */
    public void showAll()
    {
        for(String command: validCommands) {
            System.out.print(command + "  ");
        }
        System.out.println();
    }
}
```

# SOURCE CODE  Class - Parser

```
import java.util.Scanner;

/**
```

```
 * This class is part of the "Age of Gladiators" application.
 * "Age of Gladiators" is a very simple, text based adventure game.
 *
 * This parser reads user input and tries to interpret it as an "Adventure"
 * command. Every time it is called it reads a line from the terminal and
 * tries to interpret the line as a three word command. It returns the command
 * as an object of class Command.
 *
 * The parser has a set of known command words. It checks user input against
 * the known commands, and if the input is not one of the known commands, it
 * returns a command object that is marked as an unknown command.
 *
 */
public class Parser
{
    private CommandWords commands;  // holds all valid command words
    private Scanner reader;         // source of command input

    /**
     * Create a parser to read from the terminal window.
     */
    public Parser()
    {
        commands = new CommandWords();
        reader = new Scanner(System.in);
    }

    /**
     * @return The next command from the user.
     */
    public Command getCommand()
    {
        String inputLine;   // will hold the full input line
        String word1 = null;
        String word2 = null;
        String word3 = null;

        System.out.print("> ");     // print prompt

        inputLine = reader.nextLine();

        // Find up to three words on the line.
        Scanner tokenizer = new Scanner(inputLine);
        if(tokenizer.hasNext()) {
            word1 = tokenizer.next();      // get first word
            if(tokenizer.hasNext()) {
                word2 = tokenizer.next();      // get second word
                // note: we just ignore the rest of the input line.
                if(tokenizer.hasNext()){
```

```
                word3 = tokenizer.next();    // get third word
            }
        }
    }

    // Now check whether this word is known. If so, create a command
    // with it. If not, create a "null" command (for unknown command).
    if(commands.isCommand(word1)) {
        return new Command(word1, word2 , word3);
    }
    else {
        return new Command(null, word2 , word3);
    }
  }

  /**
   * Print out a list of valid command words.
   */
  public void showCommands()
  {
      commands.showAll();
  }
}
```