Write a Java program to implement a text based RPG (Role Playing Game).

This program will exercise the use of inheritance and polymorphism. You will need an abstract class called `Actor`. The code for the abstract class `Actor` will be provided for you. You will have two classes—`Player` and `Monster`—that inherit from `Actor`.

The program will implement a class `Map` as a 9x9 matrix of `char`. The `Player` should begin at the center of the map and be represented as **"P"**. You should randomly place a `Monster`, represented as **"M"** on the `Map`. A blank can be represented as **"-"**. Following is an example output:

```
- - - - - - - - -
- - - - - - - - -
- - - - - - - - -
- - - - - - - - -
- - - - P - - - -
- - - - - - - - -
- - M - - - - - -
- - - - - - - - -
- - - - - - - - -
up: w | down: s | left: a | right: d | quit: q
move:
```

The `Player` should be able to move around the `Map` using 'wsad' for moving up, down, left, or right respectively. Typing 'q' will quit the program. If the player attempts to move off of the `Map`, the player should not move.

Once the `Player` moves into a location that contains a `Monster` initiate a battle loop that generates the attacks for each `Player`. Following is an example output:

```
- - - - - - - - -
- - - - - - - - -
- - - - - - - - -
- - - - - - - - -
- - M P - - - - -
- - - - - - - - -
- - - - - - - - -
- - - - - - - - -
- - - - - - - - -
up: w | down: s | left: a | right: d | quit: q
move: a
Link encounters Ganon!
Link HP: 50 | Ganon HP: 50
Link attacks with 38
Ganon HP: 12
Ganon attacks with 20
Link HP: 30
Link attacks with 37
Ganon HP: -25

Combat is over!
Ganon is defeated!
```

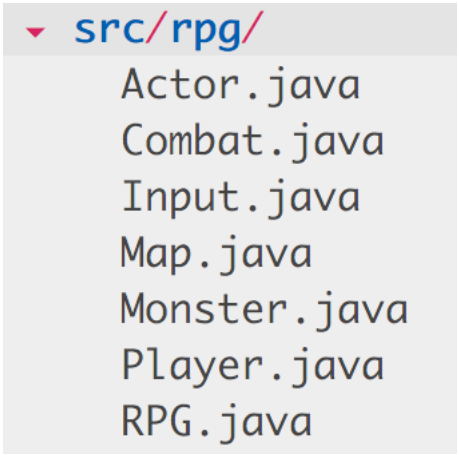
`Actor` will contain an abstract method called `generateRandomAttack()` that must be overridden in the `Player` and `Monster` classes. Override `generateRandomAttack()` so that `Player` will generate a different amount of damage than `Monster`.

```
abstract class Actor {
	//...
	abstract int generateRandomAttack();
}

class Player extends Actor {
	//...
	@Override
	public int generateRandomAttack()
	{ // ... }
}

class Monster extends Actor {
	//...
	@Override
	public int generateRandomAttack()
	{ // ... }
}
```

Since both `Player` and `Monster` inherit from `Actor`, methods can be written with formal parameters of type `Actor` which will be able to receive either a `Monster` or `Player` object. Following is an example:

```
private void checkDefeat(Actor actor) {
    if (actor.getHP() < 1) {
        System.out.println("\nCombat is over!");
        System.out.println(actor.getName() + " is defeated!\n");
        System.exit(0);
    }
}
```

You may implement this program with any number of classes that you wish. However you are required to have classes `Actor`, `Player`, `Monster`, and `Map`. My solution is implemented with the following classes:



Combat.java
- contains combat logic

Input.java
- contains input logic

RPG.java
- contains `main()` driver function
- game loop

Following are function headers and psuedocode for a possible solution:

```
public class Combat {

    public Combat()
    {}

    public void initCombat(Player player, Monster monster) {}
        // 1. Prints each actor's name and HP
        // 2. Has an infinite loop 'while(true)' that calls attackSequence()
              for each actor

    public void attackSequence(Actor attacker, Actor defender) {}
        // 1. Calls getDamage(attacker)
        // 2. Sets defender's HP to new value
        // 3. Calls combatResults()
        // 4. Prints defender's new HP
        // 5. Calls checkDefeat(defender)

    private void combatResults(Actor actor, int damage) {}
        // 1. Prints actor's name and attack damage

    private int getDamage(Actor actor) {}
        // 1. Calls actor.generateRandomAttack()

    private void checkDefeat(Actor actor) {}
        // Code given on page two of this document
}

public class Input {

    // Contains private Scanner member variable

    public Input() {}
        // 1. Initializes Scanner object

    public char readInput() {}
        // 1. Reads only the first char from keyboard input
        // 2. Loop to receive valid input—'while(!validIinput(c))'
        // 3. Returns valid char input

    private boolean validInput(char input) {}
        // 1. Return boolean if input is/isn't valid
}
```

```
public class Map {
    private final int SIZE;
    private final int PLAYER_LOCATION;
    private char[][] map;
    private final Player player;
    private final Monster monster;
    private final Combat combat;

    public Map() {}
        // 1. Initialize SIZE and PLAYER_LOCATION
        // 2. Call initializeMap()
        // 3. Call generateRandomLocation() for monsterX and monsterY
        // 4. Ensure monster is not generated in the same location as player
        // 5. Call Player, Monster, and Combat constructors
        // 6. Initialize 9x9 map of char

    private void initializeMap() {}
        // 1. Initialize each cell of map to '-'

    public void drawMap() {}
        // 1. Call initializeMap()
        // 2. Call assignActorLocation() for player and monster objects
        // 3. Print map to screen with a space between each element

    public void assignActorLocation(Actor actor) {}
        // 1. Get actor x and y location
        // 2. Assign actor.getSymbol() to location on map

    private int generateRandomLocation() {}
        // 1. Return random number within range of map edges


    public void movePlayer(char move) {}
        // 1. Get player current location
        // 2. Use switch statement to determine player new location or quit
        // 3. Call validCoordinates() to determine if player is at map edge
        // 4. If playerEncountersMonster() call combat.initCombat()

    private boolean playerEncountersMonster() {}
        // 1. Return true if player location matches monster location

    private boolean validCoordinates(int newX, int newY) {}
        // 1. Return true if player location within range of map
}

public class RPG {

    public static void main(String[] args) {

        Map map = new Map();
        Input input = new Input();
        map.drawMap();

        while (true) { / ...}
    }

}
```