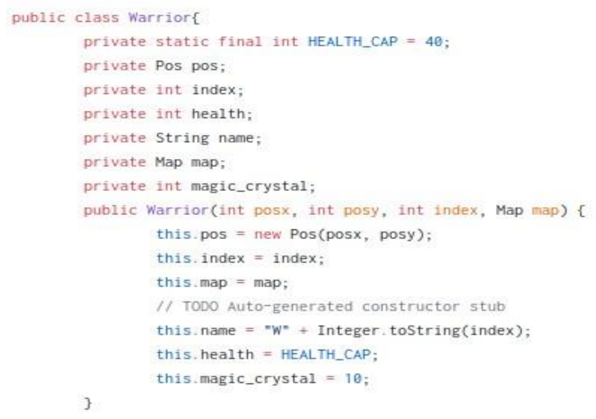
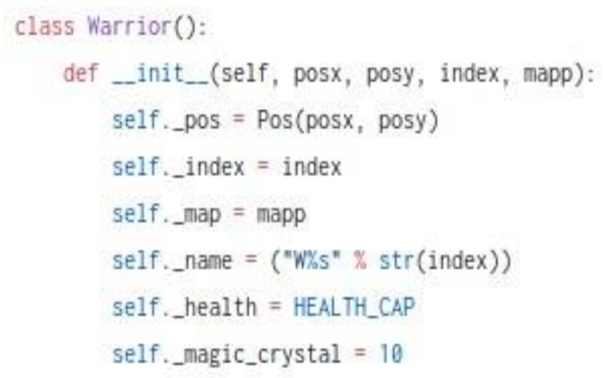
Q1)



```
public class Warrior{
        private static final int HEALTH_CAP = 40;
        private Pos pos;
        private int index;
        private int health;
        private String name;
        private Map map;
        private int magic_crystal;
        public Warrior(int posx, int posy, int index, Map map) {
                this.pos = new Pos(posx, posy);
                this.index = index;
                this.map = map;
                // TODO Auto-generated constructor stub
                this.name = "W" + Integer.toString(index);
                this.health = HEALTH_CAP;
                this.magic_crystal = 10;
        }
```

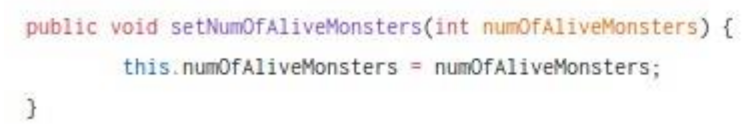


```
class Warrior():
    def __init__(self, posx, posy, index, mapp):
        self._pos = Pos(posx, posy)
        self._index = index
        self._map = mapp
        self._name = ("W%s" % str(index))
        self._health = HEALTH_CAP
        self._magic_crystal = 10
```

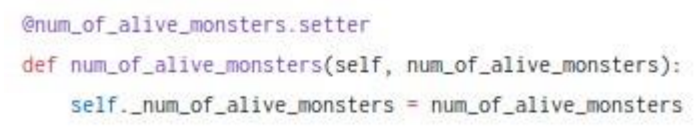
*We know that Java does not support dynamic typing as we see on the left side. Therefore Java code is extremely wordy. On the other hand, python supports the dynamic typing which is completely readable as we see on the right side code. Another advantage would be the ability to do duck-typing. For instance, on the left code, the instance “map” **have to** be an instance of Map class. However, on the right, “mapp” can basically be any object.

Q2)

Scenario 1:



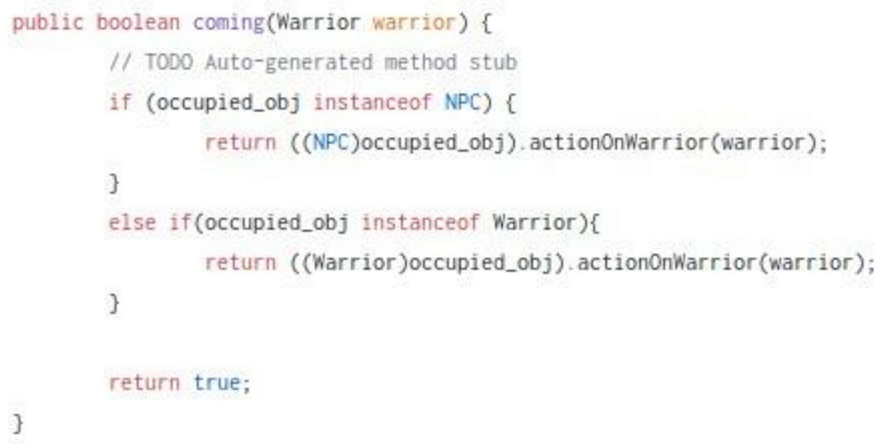
```
public void setNumOfAliveMonsters(int numOfAliveMonsters) {
        this.numOfAliveMonsters = numOfAliveMonsters;
}
```



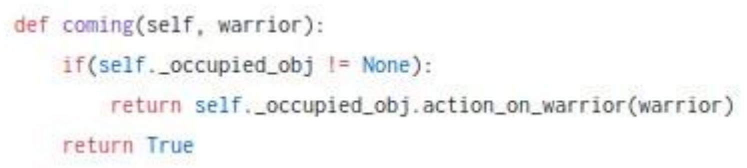
```
@num_of_alive_monsters.setter
def num_of_alive_monsters(self, num_of_alive_monsters):
    self._num_of_alive_monsters = num_of_alive_monsters
```

*The getters and setter in Java require you to write a function and use it every time you need to excess it. But in python, when we define setters in the way shown in “num_of_alive_monsters” function, we can just invoke the object’s field and give it any value we want instead of calling a function. For ex. In java myMap.setNumOfAliveMonsters = 5; but in python my_map.num_of_alive_monsters = 5

Scenario 2:



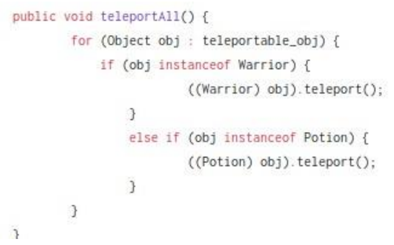
```java
public boolean coming(Warrior warrior) {
        // TODO Auto-generated method stub
        if (occupied_obj instanceof NPC) {
                return ((NPC)occupied_obj).actionOnWarrior(warrior);
        }
        else if(occupied_obj instanceof Warrior){
                return ((Warrior)occupied_obj).actionOnWarrior(warrior);
        }

        return true;
}
```



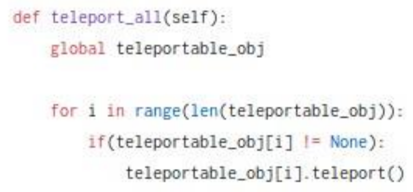
```python
def coming(self, warrior):
    if(self._occupied_obj != None):
        return self._occupied_obj.action_on_warrior(warrior)
    return True
```

*In Java, there are things such as type-checking, type-casting etc. which make the programmer to write more codes and therefore opens door for doing errors for programmer. However, in the Python, as long as right objects are passed into function, the programmer does not worry anything like type-checking. Thus, doing so saves time and makes programmer to do less errors in writing a code.

Q3)



```java
public void teleportAll() {
        for (Object obj : teleportable_obj) {
            if (obj instanceof Warrior) {
                    ((Warrior) obj).teleport();
                }
                else if (obj instanceof Potion) {
                    ((Potion) obj).teleport();
                }
        }
}
```



```python
def teleport_all(self):
    global teleportable_obj

    for i in range(len(teleportable_obj)):
        if(teleportable_obj[i] != None):
            teleportable_obj[i].teleport()
```

*As we know Java does not duck-typing and as we see on above left Java code, it is very wordy, whereas on the right side Python code is very readable and less code. It will be clear when we write tones of lines of codes, the number of object types which we must check will increase in Java, however, such things is not a problem in Python code.