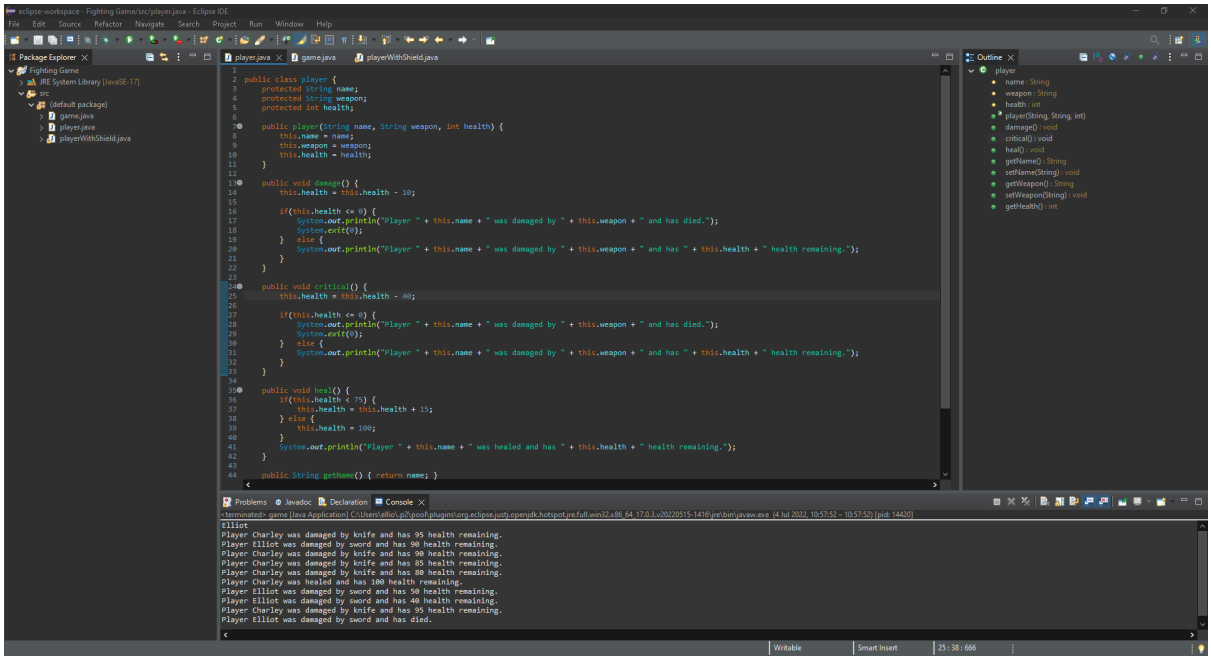
Practice: Understanding Object-Oriented Programming | SDTL3 A2.1



I am using the eclipse IDE from IBM.

I have decided to create a fighting game where players can be damaged, healed and when all health is lost the game ends. Certain players can have shields to help them take less damage.

```java

public class game {
    public static void main(String[] args) {
        player player1 = new player("Elliot", "sword", 100);
        playerWithShield player2 = new playerWithShield("Charley", "knife", 100);

        System.out.println(player1.getName());

        player2.damage();

        player1.damage();
        player2.damage();

        player2.damage();
        player2.damage();
        player2.heal();
        player1.critical();

        player1.damage();
        player2.damage();
        player1.critical();
        player1.damage();
        player1.damage();

        player2.damage();
        player1.heal();
    }
}
```

```
Problems  Javadoc  Declaration  Console
<terminated> game [Java Application] C:\Users\ellio\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.
Elliot
Player Charley was damaged by knife and has 95 health remaining.
Player Elliot was damaged by sword and has 90 health remaining.
Player Charley was damaged by knife and has 90 health remaining.
Player Charley was damaged by knife and has 85 health remaining.
Player Charley was damaged by knife and has 80 health remaining.
Player Charley was healed and has 100 health remaining.
Player Elliot was damaged by sword and has 50 health remaining.
Player Elliot was damaged by sword and has 40 health remaining.
Player Charley was damaged by knife and has 95 health remaining.
Player Elliot was damaged by sword and has died.
```

```java
public class player {
    protected String name;
    protected String weapon;
    protected int health;

    public player(String name, String weapon, int health) {
        this.name = name;
        this.weapon = weapon;
        this.health = health;
    }

    public void damage() {
        this.health = this.health - 10;

        if(this.health <= 0) {
            System.out.println("Player " + this.name + " was damaged by " + this.weapon + " and has
died.");
            System.exit(0);
        }   else {
            System.out.println("Player " + this.name + " was damaged by " + this.weapon + " and has " +
this.health + " health remaining.");
        }
    }

    public void critical() {
        this.health = this.health - 40;

        if(this.health <= 0) {
            System.out.println("Player " + this.name + " was damaged by " + this.weapon + " and has
died.");
            System.exit(0);
        }   else {
            System.out.println("Player " + this.name + " was damaged by " + this.weapon + " and has " +
this.health + " health remaining.");
        }
    }

    public void heal() {
        if(this.health < 75) {
            this.health = this.health + 15;
        } else {
            this.health = 100;
        }
        System.out.println("Player " + this.name + " was healed and has " + this.health + " health
remaining.");
    }

    public String getName() { return name; }
    public void setName(String name) { this.name = name; }

    public String getWeapon() { return weapon; }
    public void setWeapon(String weapon) { this.weapon = weapon; }

    public int getHealth() { return health; }
}
```

This is the class for all player types. It contains a constructor method which sets the base vales for the player, methods to deal damage: either simple damage or critical damage and getter/setter methods to retrieve data from the player class.

```java
public class playerWithShield extends player {
    private boolean shield;

    public playerWithShield(String name, String weapon, int health) {
        super(name, weapon, health);
        this.shield = true;
    }


    public void damage() {
        this.health = this.health - 5;

        if(this.health <= 0) {
            System.out.println("Player " + this.name + " was damaged by " + this.weapon + " and has
died.");
            System.exit(0);
        }   else {
            System.out.println("Player " + this.name + " was damaged by " + this.weapon + " and has " +
this.health + " health remaining.");
        }
    }

    @Override
    public void critical() {
        this.health = this.health - 30;

        if(this.health <= 0) {
            System.out.println("Player " + this.name + " was damaged by " + this.weapon + " and has
died.");
            System.exit(0);
        }   else {
            System.out.println("Player " + this.name + " was damaged by " + this.weapon + " and has " +
this.health + " health remaining.");
        }
    }
}
```

I then created another class for a player with a shield equipped. This class inherited the base functionality of the player class, but the damage and critical methods have been overridden with a lower damage value.

I have added a super method to the constructor so the values in the parent class can be changed.

Recently in work I have been working with C# to create a login audit API. I have used many object-oriented programming concepts such as using interfaces to abstract my code, using getter/setter classes to store data in a structured way. I have then used the entity framework in .Net to pull data from a database and return the results as a JSON result with multiple API endpoints.

I have recently learned about how to use OOP and the MVC pattern to create components that use data and update data sources and then render the data within a view. I have used this in some of my own projects/challenges I have worked on.

I have learned that OOP is a way of ensuring that you don't have to constantly repeat lines of code or functions to work on a set of data. It seems to be a better approach to the old way of writing multiple functions that change variables which can end up being messy or 'spaghetti code'.

I am looking forward to practicing OOP in JavaScript as I see it as a much better way of working with a collection of data and maintaining the state of an application.

I want to learn more about interfaces in C# and how they can abstract the functionality of my programs and make it easier for other developers to work on my program.