examples for each OOP principle Encapsulation, Abstraction, Inheritance, Polymorphism, and Composition

Encapsulation:

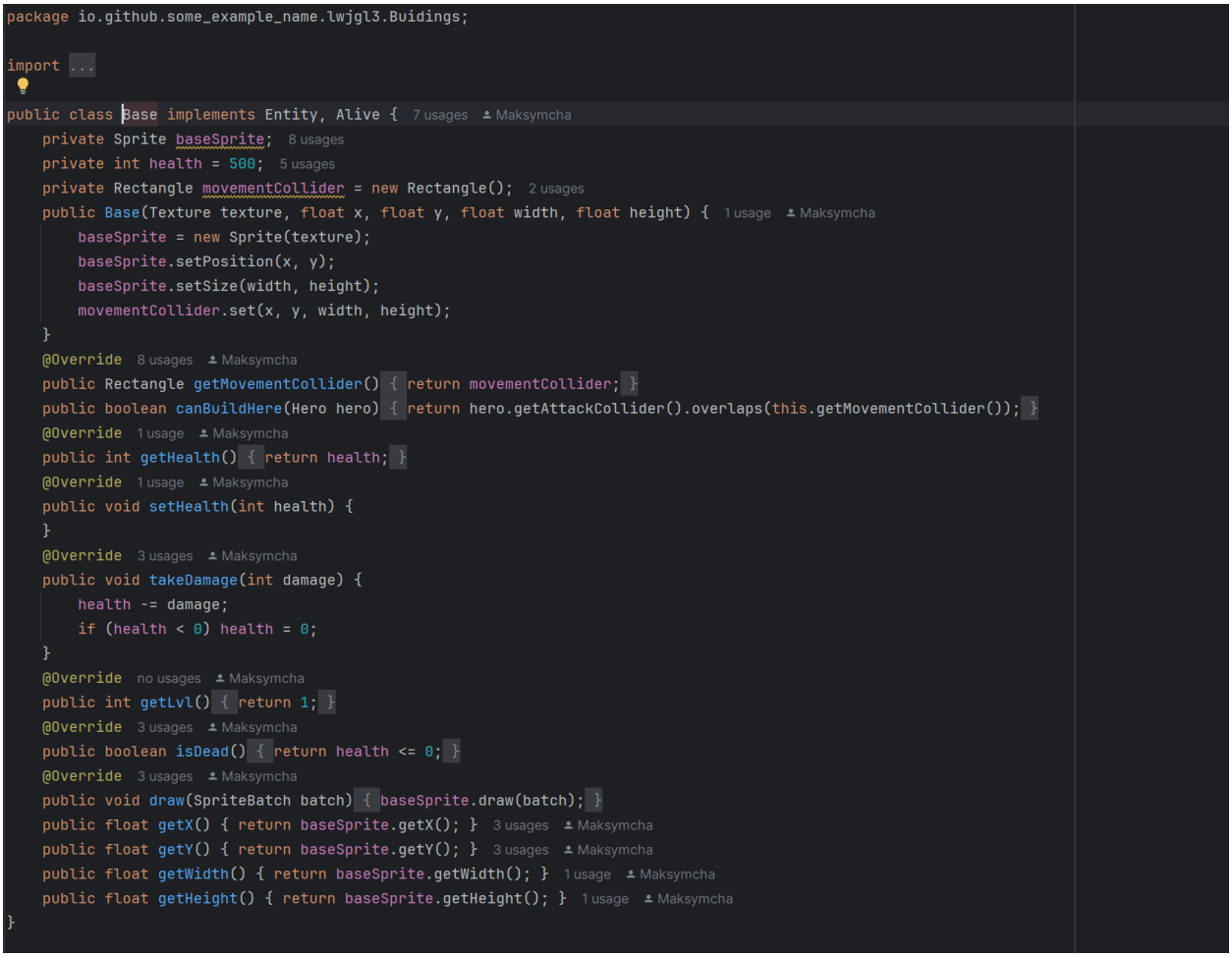
1. Barrier Class

- **Purpose**: Represents a defensive structure in a game with a health attribute that must be protected.
- The Barrier class hides its health attribute by making it private. Public methods (getHealth and setHealth) provide controlled access, enforcing rules like ensuring health never drops below zero or exceeds a maximum limit.

```
package io.github.some_example_name.lwjgl3.Buidings;
import com.badlogic.gdx.graphics.Texture;
import io.github.some_example_name.lwjgl3.Alive;

public class Barrier extends Building implements Alive {  7 usages  Maksymcha *
    private int maxHealth;  3 usages
    private int health;  5 usages
    public Barrier(int level, int cost, String type, Texture[] textures) {  1 usa
        super(level, cost, type, textures);
        maxHealth = 100;
        health = maxHealth;
    }
    @Override  no usages  Maksymcha
    public int getLvl() { return 0; }
    @Override  1 usage  Maksymcha
    public int getHealth() { return health; }
    @Override  1 usage  Maksymcha *
    public void setHealth(int health) { this.health = Math.max(0, Math.min(heal
    @Override  3 usages  Maksymcha
    public void takeDamage(int damage) { setHealth(health - damage); }
    @Override  3 usages  Maksymcha
    public boolean isDead() { return health <= 0; }
}
```

- **Purpose**: Represents a central structure (e.g., a player's headquarters) with health and a collision area.

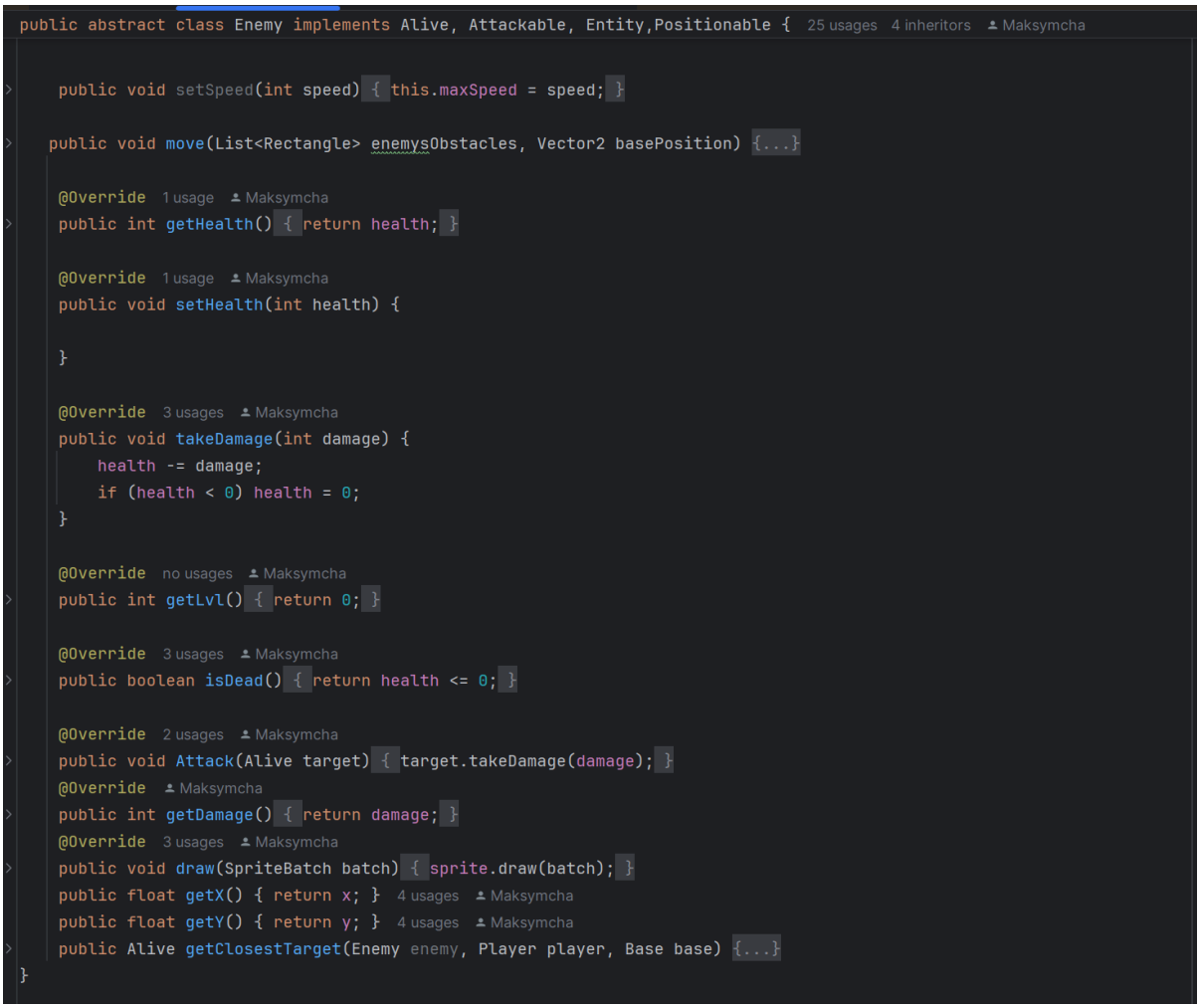
The Base class encapsulates health and movementCollider as private fields. External classes can query the health or collider but cannot modify them directly. This protects the base from unintended changes, such as resizing its collider arbitrarily, which could break collision detection logic.



```java
package io.github.some_example_name.lwjgl3.Buidings;

import ...

public class Base implements Entity, Alive {  7 usages  Maksymcha
    private Sprite baseSprite;  8 usages
    private int health = 500;  5 usages
    private Rectangle movementCollider = new Rectangle();  2 usages
    public Base(Texture texture, float x, float y, float width, float height) {  1 usage  Maksymcha
        baseSprite = new Sprite(texture);
        baseSprite.setPosition(x, y);
        baseSprite.setSize(width, height);
        movementCollider.set(x, y, width, height);
    }
    @Override  8 usages  Maksymcha
    public Rectangle getMovementCollider() { return movementCollider; }
    public boolean canBuildHere(Hero hero) { return hero.getAttackCollider().overlaps(this.getMovementCollider()); }
    @Override  1 usage  Maksymcha
    public int getHealth() { return health; }
    @Override  1 usage  Maksymcha
    public void setHealth(int health) {
    }
    @Override  3 usages  Maksymcha
    public void takeDamage(int damage) {
        health -= damage;
        if (health < 0) health = 0;
    }
    @Override  no usages  Maksymcha
    public int getLvl() { return 1; }
    @Override  3 usages  Maksymcha
    public boolean isDead() { return health <= 0; }
    @Override  3 usages  Maksymcha
    public void draw(SpriteBatch batch) { baseSprite.draw(batch); }
    public float getX() { return baseSprite.getX(); }  3 usages  Maksymcha
    public float getY() { return baseSprite.getY(); }  3 usages  Maksymcha
    public float getWidth() { return baseSprite.getWidth(); }  1 usage  Maksymcha
    public float getHeight() { return baseSprite.getHeight(); }  1 usage  Maksymcha
}
```

3.Enemy Class

- **Purpose**: Models an enemy unit with health and damage attributes.
  The Enemy class uses protected fields for health and damage, allowing subclasses (like BossEnemy) to access them directly while hiding them from unrelated classes. Public methods provide controlled access, ensuring that health or damage.

- 

4.Turret Class

- **Purpose**: Represents a defensive turret with damage and range attributes.
- The Turret class encapsulates damage and range as private fields, exposing them via getters. This read-only access ensures that a turret's attack properties remain consistent unless modified through specific methods, such as upgrading the turret in-game.

```java
package io.github.some_example_name.lwjgl3.Buidings;

import ...

public class Turret extends Building {
    private int damage;
    private float range;
    private float attackSpeed;
    private float lastShotTime = 0;

    public Turret(int level, int cost, String type, Texture[] textures, float attackSpeed, float range, int damage) {
        super(level, cost, type, textures);
        this.attackSpeed = attackSpeed;
        this.range = range;
        this.damage = damage;
    }

    public void shoot(Enemy enemy) {}

    public float getLastShotTime() { return lastShotTime; }

    public void setLastShotTime(float lastShotTime) { this.lastShotTime = lastShotTime; }

    public float getAttackSpeed() { return attackSpeed; }

    public float getRange() { return range; }

    @Override
    public void upgradeAttributes() {

    }
}
```

Abstraction:

1.Building Abstract Class

- **Purpose**: Provides a blueprint for all building types (e.g., turrets, barriers).
  The Building abstract class defines shared properties (like a sprite) and methods (like draw) that all buildings must implement. This abstraction lets the game handle all buildings consistently while allowing each type to customize its behavior.

```java
public abstract class Building {  14 usages  3 inheritors  Maksymcha
    protected int level;  2 usages
    protected int cost;  1 usage
    protected String type;  1 usage
    protected Texture[] textures;  1 usage
    protected Sprite currentSprite;  9 usages
    protected float x, y;  4 usages
    protected Rectangle movementCollider;  2 usages

    public Building(int level, int cost, String type, Texture[] textures) {
        this.level = level;
        this.cost = cost;
        this.type = type;
        this.textures = textures;
        this.currentSprite = new Sprite(textures[level - 1]);
        this.x = 0;
        this.y = 0;
        this.movementCollider = new Rectangle(x, y, currentSprite.getWidth(),
    }

    public int getLevel() { return level; }
    public float getX() { return x; }
    public float getY() { return y; }
    public void setX(float x) {  1 usage  Maksymcha
        this.x = x;
        currentSprite.setX(x);
    }
    public void setY(float y) {  1 usage  Maksymcha
        this.y = y;
        currentSprite.setY(y);
    }
    public float getWidth() { return currentSprite.getWidth(); }
```

2.Alive Interface

- **Purpose**: Defines entities that have health and can take damage.
  The Alive interface abstracts health management, allowing entities like players, enemies, or structures to share a one number of functions. Each implementer can define how damage or death is handled, promoting flexibility.

```java
package io.github.some_example_name.lwjgl3;

import com.badlogic.gdx.math.Rectangle;

public interface Alive {  17 usages  8 implementations  Maksymcha
    float getX();  3 usages  4 implementations  Maksymcha
    float getY();  3 usages  4 implementations  Maksymcha
    int getLvl();  no usages  4 implementations  Maksymcha
    boolean isDead();  3 usages  4 implementations  Maksymcha
    void takeDamage(int damage);  3 usages  4 implementations  Maksymcha
    int getHealth();  1 usage  4 implementations  Maksymcha
    void setHealth(int health);  1 usage  4 implementations  Maksymcha
    Rectangle getMovementCollider();  8 usages  4 implementations  Maksymcha
}
```

Composition:

1. Player Class

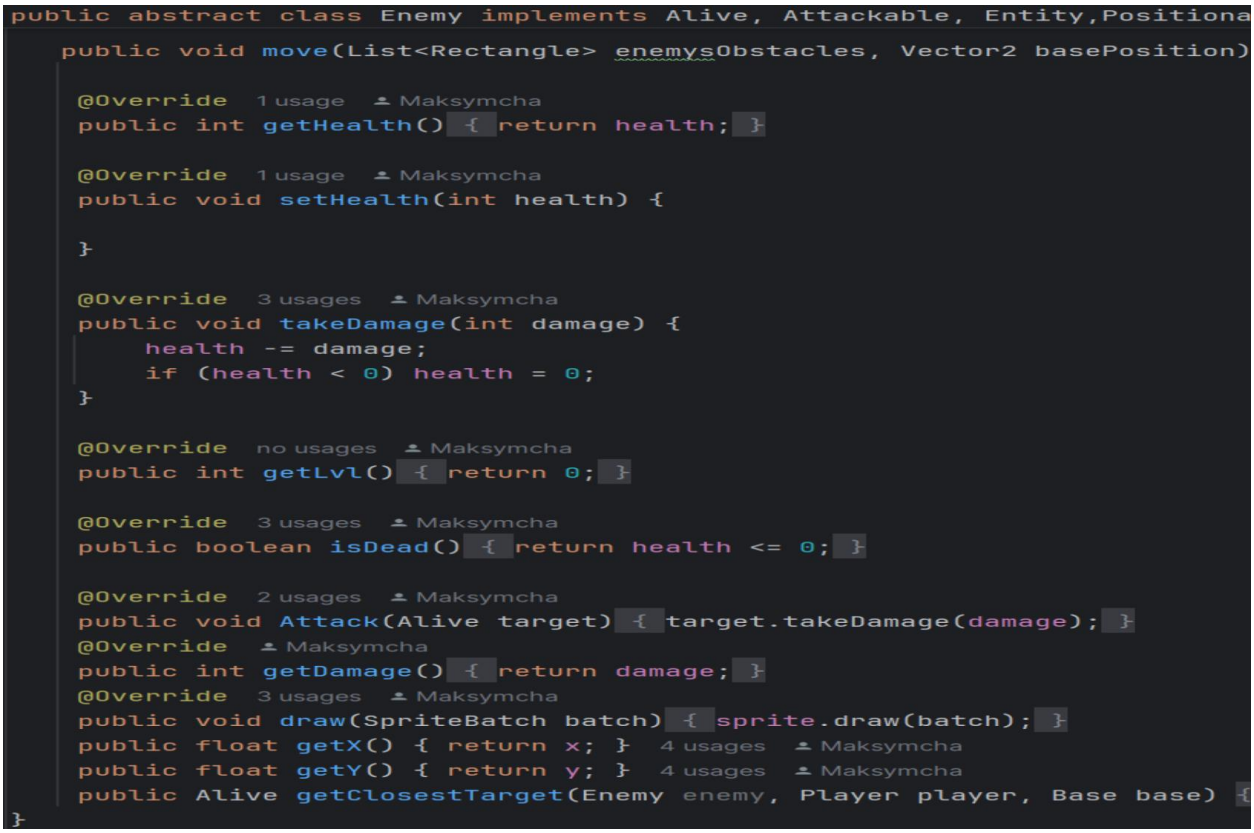
- **Purpose**: Represents the player, managing a hero and buildings.
- The Player class composes a Hero and a list of Building objects, delegating tasks like movement to the hero and rendering of buildings to the buildings.

```java
public class Player {  3 usages  Maksymcha
    private Hero hero;  3 usages
    private ArrayList<Building> buildings = new ArrayList<>();  3 usages
    private int money = 500;  3 usages

    public Player(Hero hero) { this.hero = hero; }

    public void buildBarrier(Base base, Texture[] barrierTextures, float barr
        if (money >= 100 && base.canBuildHere(hero)) {
            float barrierX = base.getX() + base.getWidth() / 2 - barrierWidth
            float barrierY = base.getY() + base.getHeight() / 2 - barrierHeig
            Rectangle intendedCollider = new Rectangle(barrierX, barrierY, ba
            boolean occupied = false;
            for (Building building : buildings) {
                if (building.getMovementCollider().overlaps(intendedCollider)
                    occupied = true;
                    break;
                }
            }
            if (!occupied) {
                    Barrier barrier = new Barrier( level: 1,  cost: 100,  type: "Ba
                    barrier.setX(barrierX);
                    barrier.setY(barrierY);
                    buildings.add(barrier);
                    money -= 100;

            }
        }
    }

    public Hero getHero() { return hero; }  5 usages  Maksymcha
```

Polymorphism:

1. Enemy's Attack Method

- **Purpose**: Allows enemies to attack any Alive entity.
- The attack method in Enemy uses the Alive interface, so it can target any Alive object (e.g., Hero, Base). Each target's takeDamage method have its specific logic.



```
public abstract class Enemy implements Alive, Attackable, Entity,Positiona
    public void move(List<Rectangle> enemysObstacles, Vector2 basePosition)

    @Override  1 usage  Maksymcha
    public int getHealth() { return health; }

    @Override  1 usage  Maksymcha
    public void setHealth(int health) {

    }

    @Override  3 usages  Maksymcha
    public void takeDamage(int damage) {
        health -= damage;
        if (health < 0) health = 0;
    }

    @Override  no usages  Maksymcha
    public int getLvl() { return 0; }

    @Override  3 usages  Maksymcha
    public boolean isDead() { return health <= 0; }

    @Override  2 usages  Maksymcha
    public void Attack(Alive target) { target.takeDamage(damage); }
    @Override  Maksymcha
    public int getDamage() { return damage; }
    @Override  3 usages  Maksymcha
    public void draw(SpriteBatch batch) { sprite.draw(batch); }
    public float getX() { return x; }  4 usages  Maksymcha
    public float getY() { return y; }  4 usages  Maksymcha
    public Alive getClosestTarget(Enemy enemy, Player player, Base base) {
}
```

ChatGpt:

he wrote me the method move for enemy and hero, as well as the method drawColliders in enemy base hero to be clearer from what the enemy dies or the hero takes damage, was written by him, as well as the method distanceBetween