

# M03 - Weapons by Interface

YiChen\_2020\_10\_25

## Weapons by Interface

Use Interfaces to create the following actions that a vehicle may take:

- **Health:**
  - object has a health value of 100%
  - health value can be decremented
  - repairs can return health value to 100%

```
Vehicle.java VehicleType.java Weapon.java AirVehicle.java FireSonicWaveWeapon.java

/**
 * set Speed
 * @param speed Speed
 */
public void setSpeed(double speed) { this.speed = speed; }

/**
 * getHealthReduced
 * @return HealthReduced
 */
@Override
public double getHealthReduced() { return healthReduced; }

/**
 * set HealthReduced
 * @param healthReduced HealthReduced
 */
public void setHealthReduced(double healthReduced) { this.healthReduced = healthReduced; }

}

public class Vehicle {
    private String name;
    private VehicleType type;
    private double health = 100.00;
    private Pedal pedal;
    private double speed;
    private SteeringWheel steeringWheel;

    @Override
    public void fire(WeaponType weaponType) {
        switch (weaponType) {
            case FIREBULLET:
                FireBulletWeapon fireBullet = new FireBulletWeapon(new Size(1, 0.4, 0.4), this.getPosition(), this.getSteeringWheel(), this.getDirection(), speed, healthReduced: 10);
                this.fireFireBullet(fireBullet);
                break;
        }
    }
}
```

- **Fire bullet:**
  - bullet goes in same direction as vehicle
  - bullet travels fast and is small (you pick speed, size)
  - bullet must collide exactly with other vehicles, to reduce their health by 10%

```
/**
 * parent weapons class
 */
public class Vehicle {
    private String name;
    private VehicleType type;
    private double health = 100.00;
    private Pedal pedal;
    private double speed;
    private SteeringWheel steeringWheel;
    private Direction direction;
    private Position position;
    private RearDefenseInNewWeapon rearDefenseMine;
    private ArrayList<FireBulletWeapon> fireBulletList = new ArrayList();
    private ArrayList<FireSonicWaveWeapon> fireSonicWaveList = new ArrayList();
    private ArrayList<InTheAirChimneyBlastWeapon> inTheAirChimneyBlastList = new ArrayList();

    /**
     * fire(WeaponType weaponType) {
     * h(weaponType) {
     * case FIREBULLET:
     *     FireBulletWeapon fireBullet = new FireBulletWeapon(new Size(1, 0.4, 0.4), this.getPosition(), this.getSteeringWheel(), this.getDirection(), speed, healthReduced: 10);
     *     this.fireFireBullet(fireBullet);
     *     break;
     * }
     */
}
```

- **Fire sonic wave:**
  - sonic goes in same direction as vehicle
  - sonic wave travels slow, and is large (you pick speed, size)
  - sonic wave can collide with other vehicles, reduce their health by 5%

```

Weapon.java | FireBulletWeapon.java | Main.java | LandVehicle.java | FireSonicWaveWeapon.java | MyVehicle.java
/**
 * set Direction
 * @param direction Direction
 */
public void setDirection(Direction direction) { this.direction = direction; }

/**
 * get Speed
 * @return Speed
 */
@Override
public double getSpeed() { return speed; }

case FIRESONICWAVE:
    FireSonicWaveWeapon fireSonicWave = new FireSonicWaveWeapon(new Size(2, 1), this.getPosition(), this.getSteeringWheel(), this.getDirection(), speed: 2, healthReduced: 5);
    this.fireFireSonicWave(fireSonicWave);
    break;

```

- **Rear defense mine**

- mine is stationary, at the place vehicle left it.
- mine is large (you pick size)
- mine can collide with other vehicles, reduce their health by 25%

```

va | FireBulletWeapon.java | Main.java | LandVehicle.java | FireSonicWaveWeapon.java | RearDefenseMineWeapon.java | MyVehicle.java
@Override
public double getSpeed() { return 0; }

/**
 * set Position
 * @param position position
 */
public void setPosition(Position position) { this.position = position; }

/**
 * get HealthReduced
 * @return healthReduced
 */
@Override
public double getHealthReduced() { return healthReduced; }

/**
 * set HealthReduced
 * @param healthReduced healthReduced
 */
public void setHealthReduced(double healthReduced) { this.healthReduced = healthReduced; }

/**
 * initialize weapon
 */
private void initWeapon() {
    RearDefenseMineWeapon rearDefenseMine = new RearDefenseMineWeapon(new Size(2, 2), this.getPosition(), healthReduced: 25);
    this.setRearDefenseMine(rearDefenseMine);
}

```

- **In-the-Air chimney blast**

- blows a area directly above with a straight up blast
- blast starts in this vehicles location, going up with a radius that you choose.
- blast can collide with other vehicles, reduce their health by 5%

```

break;
case INTHEAIRCHIMNEYBLAST:
    InTheAirChimneyBlastWeapon inTheAirChimneyBlast = new InTheAirChimneyBlastWeapon(this.getPosition(), radius: 2, speed: 2, healthReduced: 5);
    this.fireInTheAirChimneyBlast(inTheAirChimneyBlast);
    break;
}

```

```

4  * class InTheAirChimneyBlastWeapon
5  * blows a area directly above with a straight up blast
6  * blast starts in this vehicles location, going up wit a radius that you choose.
7  * blast can collide with other vehicles, reduce their health by 5%
8  */
9  public class InTheAirChimneyBlastWeapon extends Weapon implements WeaponInterface {
10     private Position position;
11     private double radius;
12     private double speed;
13     private double healthReduced;
14
15     /**
16      * InTheAirChimneyBlastWeapon
17      * @param position position
18      * @param radius radius
19      * @param speed speed
20      * @param healthReduced healthReduced
21      */
22     public InTheAirChimneyBlastWeapon(Position position, double radius, double speed, double healthReduced) {
23         this.position = position;
24         this.radius = radius;
25         this.speed = speed;
26         this.healthReduced = healthReduced;
27     }

```

- Weapons:
  - Connect one vehicle to the keyboard and/or mouse for firing weapons (or get weapons to fire at random times)
  - process a weapons strike on a vehicle ... decrements health, changes directions, whatever you want.

Write a Java program that creates 3 land vehicles and 3 air vehicles, and a loop performing:

```

1  package com.weapons;
2
3  /**
4   * class Weapon collect all weapons
5   */
6  public class Weapon {
7      private String name;
8      private Vehicle vehicle;
9
10     /**
11      * getName
12      * @return name
13      */
14     public String getName() { return name; }
15
16     /**
17      * setName
18      * @param name name
19      */
20     public void setName(String name) { this.name = name; }
21
22     /**
23      * getVehicle
24      * @return weapons
25      */
26     public Vehicle getVehicle() { return this.vehicle; }
27
28     /**
29      * set Vehicle
30      * @param vehicle weapons
31      */
32     public void setVehicle(Vehicle vehicle) { this.vehicle = vehicle; }
33 }

```

- Give the vehicle random settings for speed, direction, and all other settings (wings up, ...whatever)

```

on.java x FireBulletWeapon.java x Main.java x LandVehicle.java x FireSonicWaveWeapon.java x RearDefen
/**
 * LandVehicle
 * @param type LandVehicle
 * @param name model
 */

public LandVehicle(VehicleType type, String name) {
    super(type, name);
    this.setSpeed(levelSpeed);
    Random random = new Random();
    double randomX = random.nextInt( bound: 21) - 10;
    double randomY = random.nextInt( bound: 21) - 10;
    this.setPosition(new Position(randomX, randomY, 0));
    this.setSteeringWheel(SteeringWheel.CENTER);
    this.setDirection(Direction.STRAIGHT);
    this.initWeapon();
}

/**
 * initialize weapon
 */
private void initWeapon() {
    RearDefenseMineWeapon rearDefenseMine = new RearDefenseMineWeapon(new Size( 2,

```

- The vehicle is updated every second, and it changes position depending on the steering wheel, or propeller, or whatever.

```

x=3.07, y=0.0, z=10.93, name=Bombardier, steeringwheel=LEFT, speed=93.0%, direction=UP
x=9.0, y=-9.08, z=9.08, name=Boeing, steeringwheel=CENTER, speed=92.0%, direction=DOWN
x=2.0, y=-9.5, z=0.0, name=MyVehicle, steeringwheel=CENTER, speed=50.0%, direction=STRAIGHT
x=-3.0, y=-5.39, z=0.0, name=Toyota, steeringwheel=CENTER, speed=61.0%, direction=STRAIGHT
x=1.62, y=-2.0, z=0.0, name=Ford, steeringwheel=LEFT, speed=37.5%, direction=STRAIGHT
x=-1.6, y=9.0, z=0.0, name=Volkswagen, steeringwheel=LEFT, speed=60.0%, direction=STRAIGHT
x=3.0, y=9.03, z=10.0, name=Airbus, steeringwheel=CENTER, speed=104.0%, direction=STRAIGHT
LANDVEHICLE MyVehicle's health: 75.0

/**
 * start a thread to move weapons every 5s
 * @param vehicle
 */
public void scheduleMove(Vehicle vehicle) {
    final long timeInterval = 5000;
    Runnable runnable = new Runnable() {
        public void run() {
            while (true) {
                switch (vehicle.getType()) {
                    case LANDVEHICLE:
                        if (vehicle.getHealth() > 0) {
                            moveLandVehicle(vehicle);
                        }
                        break;
                    case AIRVEHICLE:
                        if (vehicle.getHealth() > 0) {
                            moveAirVehicle(vehicle);
                        }
                        break;
                }
                try {
                    Thread.sleep(timeInterval);
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
            }
        }
    };
    Thread thread = new Thread(runnable);
    thread.start();
}

```

- Every 10 seconds the controls are randomly nudged in one direction or another, to simulate someone at the controls. (display the change in text output)...*this includes random new firing of weapons above.*

```

C:\Users\YI Chen\JDKs\openjdk-15-1\bin\java.exe -javaagent:C:\Program Files\JetBrains\IntelliJ IDEA
x=3.07, y=0.0, z=10.93, name=Bombardier, steeringwheel=LEFT, speed=93.0%, direction=UP
x=9.0, y=-9.08, z=9.08, name=Boeing, steeringwheel=LEFT, speed=92.0%, direction=DOWN
x=2.0, y=-9.5, z=0.0, name=MyVehicle, steeringwheel=LEFT, speed=50.0%, direction=STRAIGHT
x=-3.0, y=-5.39, z=0.0, name=Toyota, steeringwheel=LEFT, speed=61.0%, direction=STRAIGHT
x=1.62, y=-2.0, z=0.0, name=Ford, steeringwheel=LEFT, speed=37.5%, direction=STRAIGHT
x=-1.6, y=9.0, z=0.0, name=Volkswagen, steeringwheel=LEFT, speed=60.0%, direction=STRAIGHT
x=3.0, y=9.03, z=10.0, name=Airbus, steeringwheel=LEFT, speed=104.0%, direction=STRAIGHT
LANDVEHICLE MyVehicle's health: 75.0

```

```

public void scheduleMove(Vehicle vehicle) {
    final long timeInterval = 5000;
    Runnable runnable = new Runnable() {
        public void run() {
            while (true) {
                switch (vehicle.getType()) {
                    case LANDVEHICLE:
                        if (vehicle.getHealth() > 0) {
                            moveLandVehicle(vehicle);
                        }
                        break;
                    case AIRVEHICLE:
                        if (vehicle.getHealth() > 0) {
                            moveAirVehicle(vehicle);
                        }
                        break;
                }
                try {
                    Thread.sleep(timeInterval);
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
            }
        }
    };
}

```

```

/**
 * start thread to fire a weapon every 2s
 * @param vehicle vehicle
 */
public void scheduleFire(Vehicle vehicle) {
    final long timeInterval = 2000;
    Runnable runnable = new Runnable() {
        public void run() {
            while (true) {
                int randomWeaponType = new Random().nextInt(WeaponType.values().length);
                WeaponType weaponType = WeaponType.values()[randomWeaponType];
                vehicle.fire(weaponType);
                try {
                    Thread.sleep(timeInterval);
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
            }
        }
    };
    Thread thread = new Thread(runnable);
    thread.start();
}

```

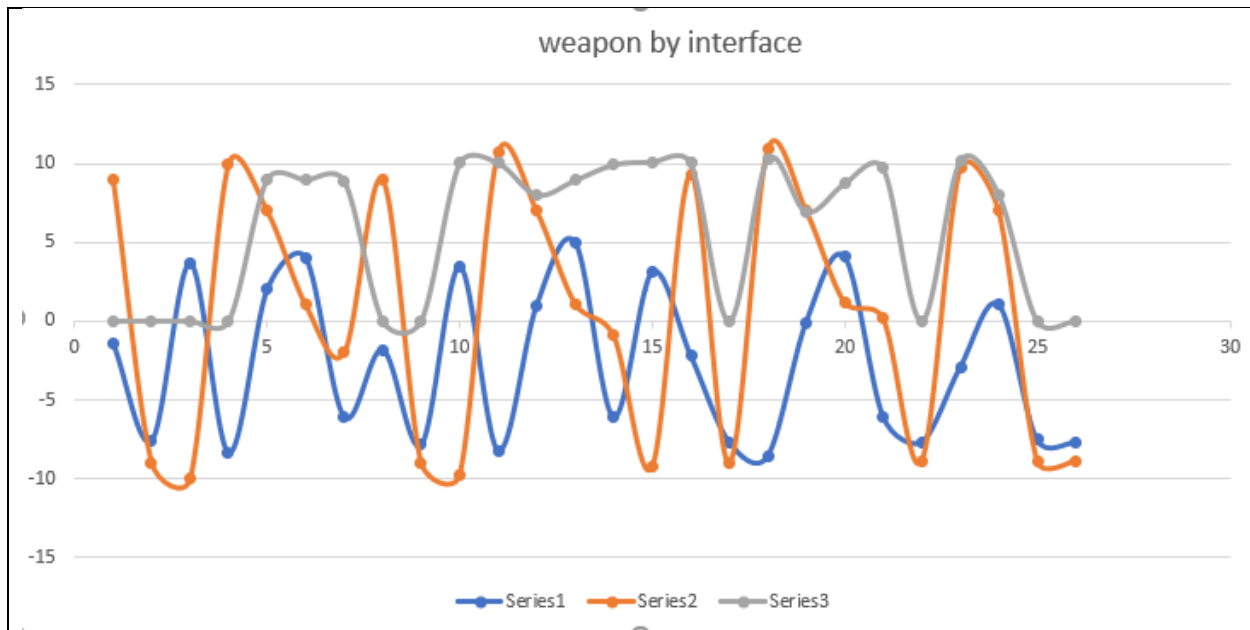
- The vehicle rebounds from outer boundaries so it stays within your viewing area.

```

private void countPositionsForWeapons(SteeringWheel steeringWheel, Direction direction, Position position, double speed) {
    if (direction.equals(Direction.STRAIGHT)) {
        if (steeringWheel.equals(SteeringWheel.LEFT)) {
            double x = position.getX() + (speed / 100) * -1;
            if (x < -10 || x > 10) {
                x = 0;
            }
            position.setX(x);
        } else if (steeringWheel.equals(SteeringWheel.CENTER)) {
            double y = position.getY() + (speed / 100) * 1;
            if (y > 100) {
                y = 0;
            }
            position.setY(y);
        } else if (steeringWheel.equals(SteeringWheel.RIGHT)) {
            double x = position.getX() + (speed / 100) * 1;
            if (x < -10 || x > 10) {
                x = 0;
            }
            position.setX(x);
        }
    }
}

```

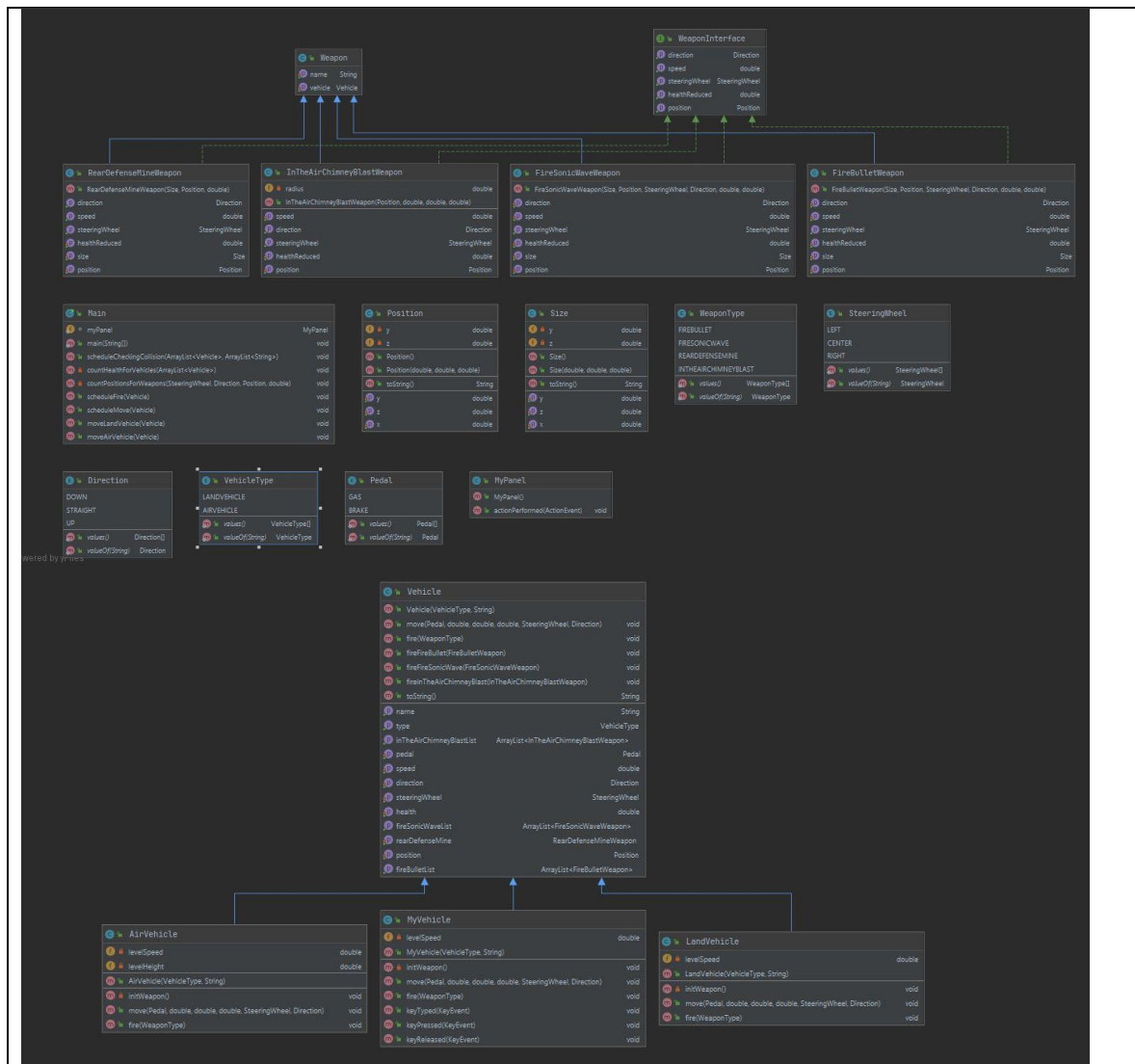
- Text output shows the location each second, in x,y,z format, capable of being loaded into a spreadsheet.



- The physics of the movement need not be perfect, but left and right would change the x, y coordinates, and height would change the z coordinate (or y coordinate?) in the proper general direction.

```
x=3.08, y=-9.23, z=10.11, name=Volkswagen, steeringwheel=CENTER, speed=34.33%, direction=STRAIGHT
x=-2.18, y=9.33, z=10.04, name=Ford, steeringwheel=CENTER, speed=33.9%, direction=STRAIGHT
x=-7.67, y=-9.0, z=0.0, name=MyVehicle, steeringwheel=RIGHT, speed=18.49%, direction=STRAIGHT
x=-8.55, y=10.92, z=10.25, name=Toyota, steeringwheel=LEFT, speed=59.61%, direction=STRAIGHT
x=-0.07, y=7.0, z=6.94, name=Airbus, steeringwheel=LEFT, speed=107.91%, direction=DOWN
x=4.06, y=1.19, z=8.74, name=Boeing, steeringwheel=LEFT, speed=98.82%, direction=STRAIGHT
x=-6.08, y=0.17, z=9.71, name=Bombardier, steeringwheel=CENTER, speed=109.08%, direction=STRAIGHT
```

Install [ObjectAid](#) in your eclipse to produce a class diagram of your homework, showing the classes used, and their relationship.



Copy the table below into your package-level JavaDoc documentation, and PDF. For every concept, describe how you implemented, or could implement it.

S SRP<sup>#</sup> Single responsibility principle  
 O OCP<sup>#</sup> Open/closed principle  
 L LSP<sup>#</sup> Liskov substitution principle

Enum direction class have responsibility over a single part of that program's move functionality

The implementation of the Vehicle class is relatively simple. It just has a constructor, a public method to add position, and a method that brows a direction, so on.

illustrate this point was in LSP

I ISP<sup>#</sup> Interface segregation principle  
 D DIP<sup>#</sup> Dependency inversion principle

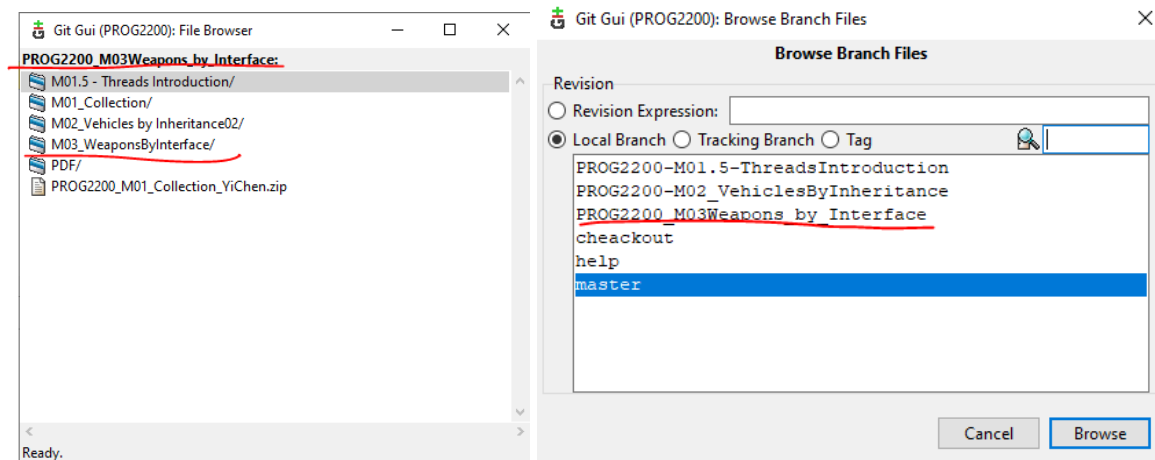
the interface-segregation principle (ISP) states that no client should be forced to depend on methods it does not use.<sup>#</sup>

That directly depends on one of the implementation classes is the Main class, which instantiates a moveAndVehicle object and moveAiVehicle.

The most effective way I have seen to

Submit Artifacts for marking:

- You will be asked to explain your code in class during a code review. Be prepared to demonstrate your code, and answer questions.
- your code into the git server, using a branch labeled PROG2200-Mxx (where xx is the module number)



- Submit a simple PDF with code and running output (no TOC, paragraphs, ...)
- Submit a movie (MP4) of you explaining your code and running your code.