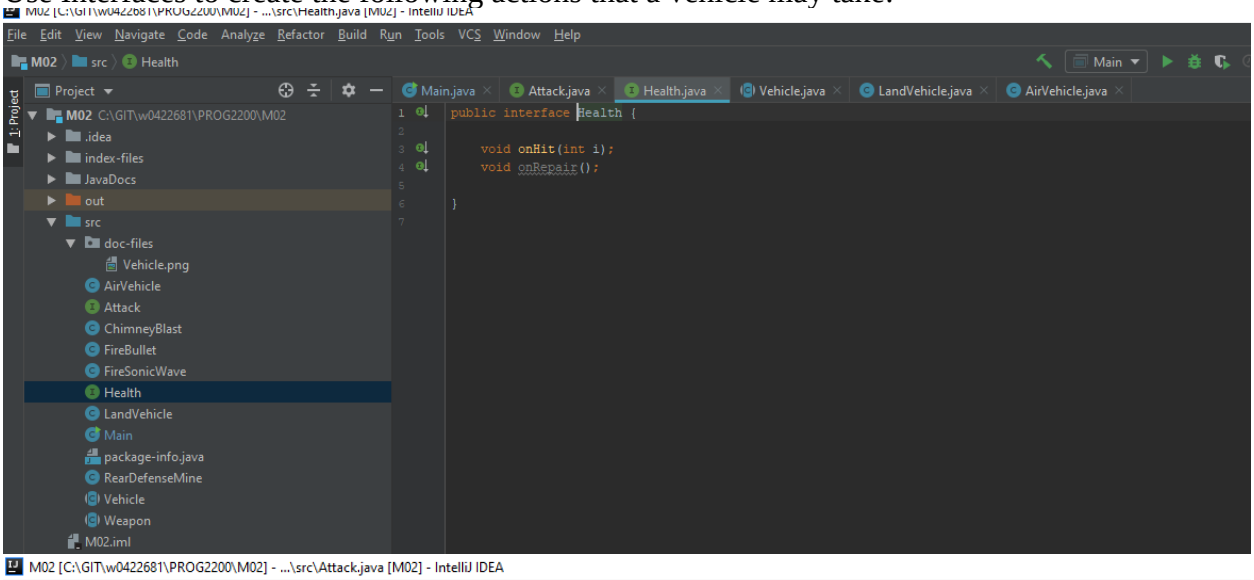
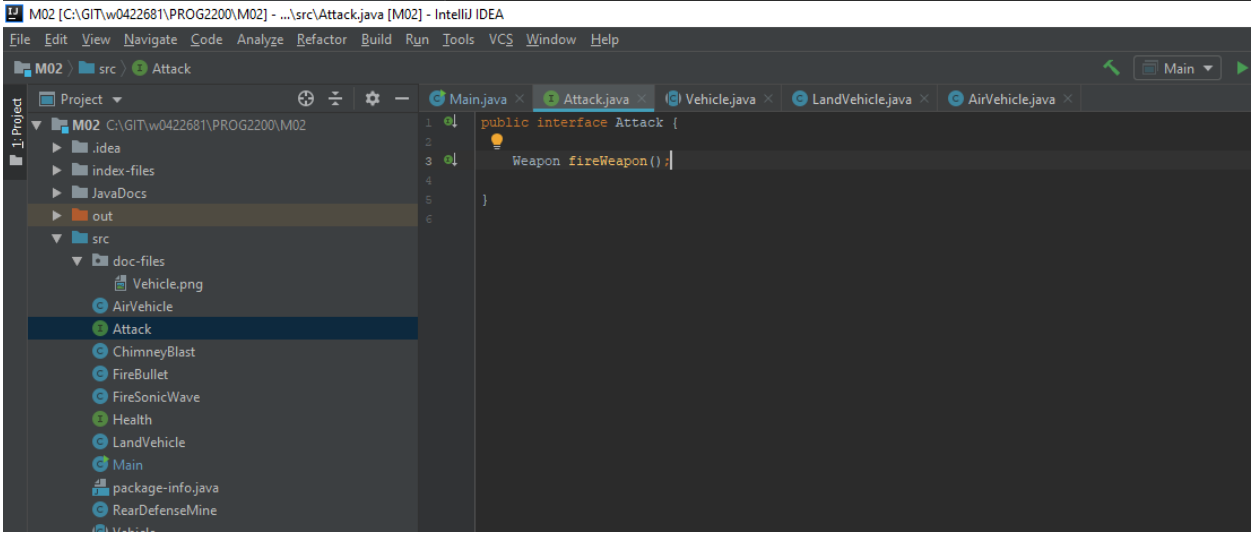
M03 - Weapons by Interface

Weapons by Interface

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



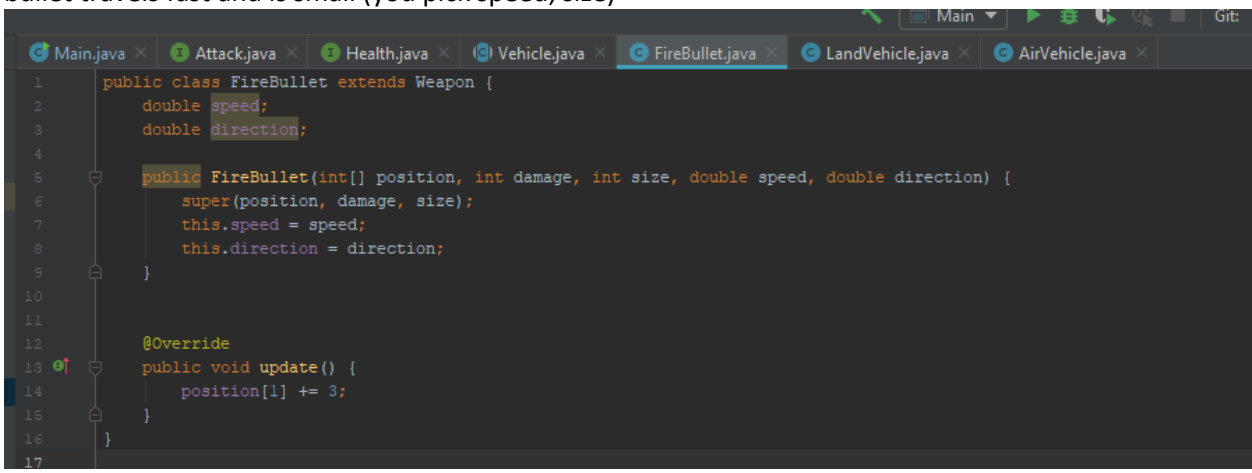


- **Health**:
  - object has a health value of 100%
  - health value can be decremented

```
public abstract class Vehicle implements Health, Attack {
    int[] position;
    double health;
    double speed; // Speed of Vehicles.
    double direction; // Direction of travel on virtual plane
    double accelerationValue = 5;
    double maxDirection = 15;
    double minDirection = -15;
    int maxHeight = 500;
    int maxThresholdX = 100;
    int maxThresholdY = 1000;
    int minThresholdY = 0;
    Enum steeringDirection;
```

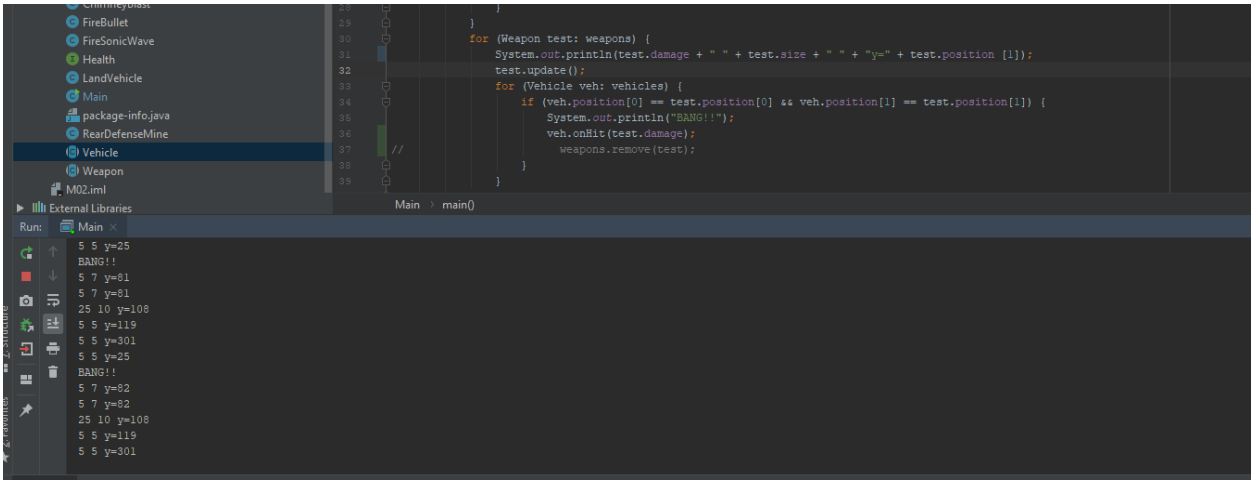
- repairs can return health value to 100%

- **Fire bullet**:
  - bullet goes in same direction as vehicle
  - bullet travels fast and is small (you pick speed, size)

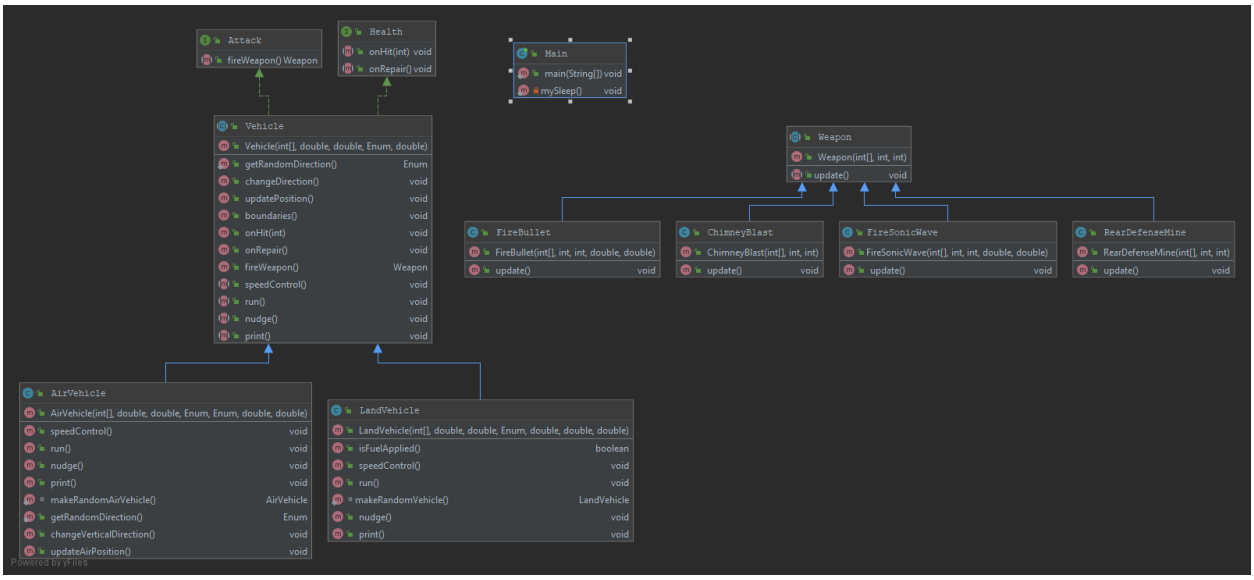


  - bullet must collide exactly with other vehicles, to reduce their health by 10%
- **Fire sonic wave:**
  - sonic goes in same direction as vehicle
  - sonic wave travels slow, and is large (you pick speed, size)
  - sonic wave can collide with other vehicles, reduce their health by 5%
- **Rear defense 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%
- **In-the-Air chimney blast**
  - blows a area directly above with a straight up blast
  - blast starts in this vehicles location, going up wit a radius that you choose.
  - blast can collide with other vehicles, reduce their health by 5%

```java
public Weapon fireWeapon() {
    Random rand = new Random();
    int weapon = rand.nextInt( bound: 4);
    switch(weapon) {
        case 0:
            double speedFaster = this.speed + 20;
            int[] bulletPosition = this.position.clone();
            int bulletDamage = 10;
            int bulletSize = 1;
            return new FireBullet(bulletPosition, bulletDamage, bulletSize, speedFaster, direction);
        case 1:
            double speedSlower = this.speed - 30;
            int[] wavePosition = this.position.clone();
            int waveDamage = 5;
            int waveSize = 7;
            return new FireSonicWave(wavePosition, waveDamage, waveSize, speedSlower, direction);
        case 2:
            int mineDamage = 25;
            int mineSize = 10;
            return new RearDefenseMine(position.clone(), mineDamage, mineSize);
        case 3:
            int blastDamage = 5;
            int blastSize = 5;
            return new ChimneyBlast(position.clone(), blastDamage, blastSize);
        default:
            return null;
    }
}
```



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



	acronym	Concept	My Application of this concept
S	SRP	Single responsibility principle	Every method does one thing and one thing only
O	OCP	Open/closed principle	We use abstract interfaces where thins can be changed on the fly for specific requirements
L	LSP	Liskov substitution principle	My ArrayLists have been made to house all objects that inherit from their respective base class
I	ISP	Interface segregation principle	I use an abstract method for speedControl() since air and land handle this differently.
D	DIP	Dependency inversion principle	As an example my Vehicle would be a low level, and my land and air would be high level.