Test 8
Set 1

1.

```
// Define the Animal interface
interface Animal {
    void makeSound();
}

// Define the Dog class that implements Animal
class Dog implements Animal {
    @Override
    public void makeSound() {
        System.out.println("Woof");
    }
}

// Define the Cat class that implements Animal
class Cat implements Animal {
    @Override
    public void makeSound() {
        System.out.println("Meow");
    }
}

// Define the Cow class that implements Animal
class Cow implements Animal {
    @Override
    public void makeSound() {
        System.out.println("Moo");
    }
}
```

```
// Define the Zoo class with the main method
public class Zoo {
   public static void main(String[] args) {
      // Create instances of Dog, Cat, and Cow
      Animal dog = new Dog();
      Animal cat = new Cat();
      Animal cow = new Cow();

      // Call makeSound() on each instance
      dog.makeSound(); // Outputs: Woof
      cat.makeSound(); // Outputs: Meow
      cow.makeSound(); // Outputs: Moo
   }
}
```

Issue:

- The Cow class implements the Animal interface, but its makeSound method is incorrectly implemented. It currently outputs "Woof", which is not the sound a cow makes.

Fixing the Issue:

- To fix the issue, you need to update the `makeSound` method in the `Cow` class so that it outputs the correct sound, "Moo".

```
Microsoft Windows [Version 10.0.22631.3958]
(c) Microsoft Corporation. All rights reserved.

C:\Users\nk930\OneDrive\文件\java>javac Zoo.java

C:\Users\nk930\OneDrive\文件\java>java Zoo
Woof
Meow
Moo

C:\Users\nk930\OneDrive\文件\java>
```

2. // Define the abstract class Vehicle

```
abstract class Vehicle {
    abstract void startEngine();

    public void display() {
        System.out.println("Vehicle is ready.");
    }
}

// Define the Car class extending Vehicle
class Car extends Vehicle {
    @Override
    void startEngine() {
        System.out.println("Car engine started.");
    }
}

// Define the Boat class extending Vehicle
class Boat extends Vehicle {
    @Override
    void startEngine() {
        System.out.println("Boat engine started.");
    }

    public void anchor() {
        System.out.println("Boat is anchored.");
    }
}

// Define the Main class with the main method
public class Main {
```

```
    public static void main(String[] args) {
        Vehicle myCar = new Car();
        Vehicle myBoat = new Boat();

        myCar.startEngine(); // Outputs: Car engine started.
        myBoat.startEngine(); // Outputs: Boat engine started.

        // Casting myBoat to Boat to access the anchor() method
        if (myBoat instanceof Boat) {
            Boat boat = (Boat) myBoat;
            boat.anchor(); // Outputs: Boat is anchored.
        }
    }
}
```

Issue

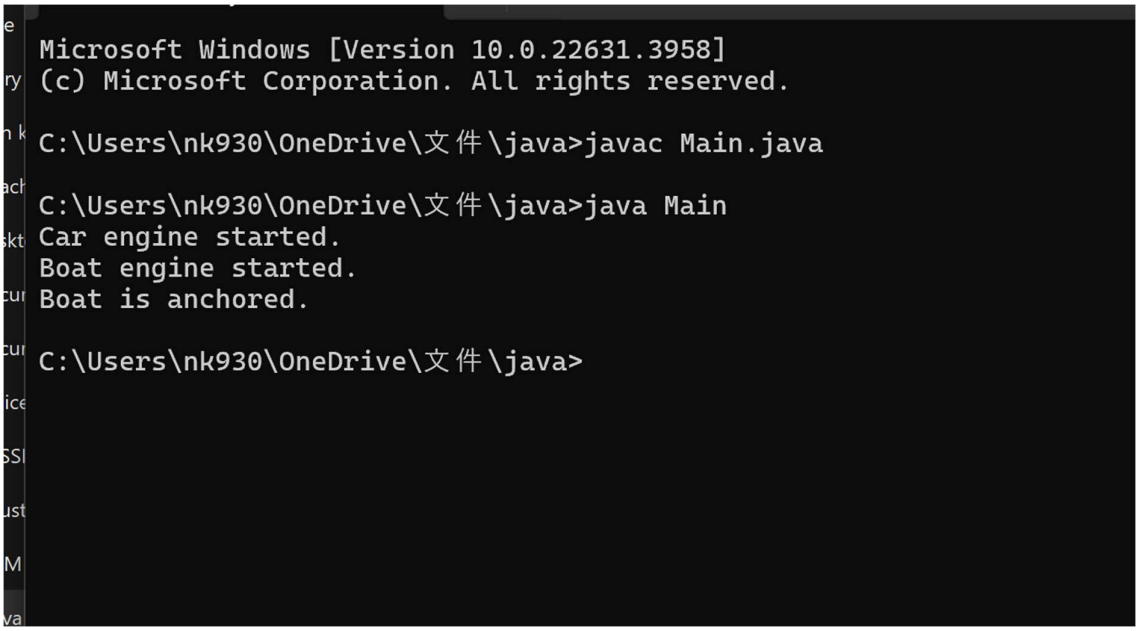
1. **Type of Reference**: In the main method, myBoat is declared as a Vehicle type:

   ```
   Vehicle myBoat = new Boat();
   ```

How can you ensure that methods specific to a subclass are accessible when needed?

Type Casting: Explicitly cast the Vehicle reference to the Boat type before calling the anchor() method. This tells the compiler to treat the reference as a Boat object, allowing access to Boat-specific methods.

Instance Check: Before casting, it's a good practice to check if the object is indeed an instance of the subclass using instanceof


Microsoft Windows [Version 10.0.22631.3958]
(c) Microsoft Corporation. All rights reserved.
C:\Users\nk930\OneDrive\文件\java>javac Main.java
C:\Users\nk930\OneDrive\文件\java>java Main
Car engine started.
Boat engine started.
Boat is anchored.
C:\Users\nk930\OneDrive\文件\java>