**Debugging Exercise**

**Exercise 1:**

public class Main {

public static void main(String[] args) {

System.out.println("Hello, World!")

}

}

**Answer:**

public class Main {

public static void main(String[] args) {

System.out.println("Hello, World!"); **// Add semicolon here, as in java each statement should end with a semicolon(;)**

}

}

**Exercise 2:**

public class Main {

public static void main(String[] args) {

int[] numbers = {1, 2, 3, 4, 5);

System.out.println(numbers[5]);

}

}

**Answer:**

public class Main {

public static void main(String[] args) {

int[] numbers = {1, 2, 3, 4, 5}; **// Corrected the closing bracket**

System.out.println(numbers[4]); **// Arrays are zero-indexed, so index 4 corresponds to the fifth element**

}

}

**Exercise 3:**

class Animal {

void eat() {

System.out.println("This animal eats food.");

}

}

class Dog extends Animal {

void eat() {

System.out.println("This dog eats meat.");

}

}

public class Main {

public static void main(String[] args) {

Animal myDog = new Dog();

myDog.eat();

}

}

**Answer:**

As there is no compile-time error in the given code, here is the output of the program:

**This dog eats meat.**

It shows the runtime polymorphic behavior where the eat method of the ‘Dog’ class, which overrides the same method in the Animal class, is invoked even though the reference is of type ‘Animal’. This behavior is possible due to the dynamic method dispatch feature in Java.