Problem 1: Basic Enum Declaration

Problem: Create an enum `Day` with values representing days of the week. Write a program that prints a custom message based on whether the current day is a weekday or weekend.

Solution:

```
java
Copy code
// Enum to represent days of the week
enum Day {
    MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY, SATURDAY, SUNDAY
}

public class Main {
    public static void main(String[] args) {
        Day today = Day.SATURDAY;

        switch (today) {
            case SATURDAY, SUNDAY:
                System.out.println("It's the weekend! Time to relax.");
                break;
            default:
                System.out.println("It's a weekday. Time to work!");
        }
    }
}
```

Output:

```
css
Copy code
It's the weekend! Time to relax.
```

Problem 2: Enum with Methods

Problem: Extend the `Day` enum from the previous problem by adding a method `isWeekend()` that returns true if the day is a weekend, and false otherwise.

Solution:

```
java
Copy code
// Enum with methods
enum Day {
    MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY, SATURDAY, SUNDAY;

    // Method to check if it's a weekend
    public boolean isWeekend() {
        return this == SATURDAY || this == SUNDAY;
    }
}

public class Main {
    public static void main(String[] args) {
        Day today = Day.TUESDAY;

        if (today.isWeekend()) {
            System.out.println(today + " is a weekend.");
        } else {
            System.out.println(today + " is a weekday.");
        }
    }
}
```

Output:

```
csharp
Copy code
TUESDAY is a weekday.
```

Problem 3: Enum with Constructor and Fields

Problem: Create an enum `Planet` with constants representing the planets of our solar system. Each constant should have a `mass` and `radius`. Add a method to calculate the gravitational force on a given planet using the formula:

gravity=G×massradius2\text{gravity} = \frac{G \times \text{mass}}{\text{radius}^2}gravity=radius2G×mass

Where GGG is a constant (`6.67430e-11`).

Solution:

```
java
Copy code
// Enum with fields and constructor
enum Planet {
    MERCURY(3.30e+23, 2.44e6),
    VENUS(4.87e+24, 6.05e6),
    EARTH(5.97e+24, 6.37e6),
    MARS(6.42e+23, 3.39e6);

    private final double mass;    // in kilograms
    private final double radius;  // in meters

    // Gravitational constant
    private static final double G = 6.67430e-11;

    // Constructor
    Planet(double mass, double radius) {
        this.mass = mass;
        this.radius = radius;
    }

    // Method to calculate surface gravity
    public double surfaceGravity() {
        return G * mass / (radius * radius);
    }
}

public class Main {
    public static void main(String[] args) {
        for (Planet planet : Planet.values()) {
            System.out.printf("Gravity on %s: %f m/s²%n", planet,
planet.surfaceGravity());
        }
    }
}
```

Output:

```
bash
Copy code
Gravity on MERCURY: 3.703026 m/s²
Gravity on VENUS: 8.872647 m/s²
Gravity on EARTH: 9.800644 m/s²
Gravity on MARS: 3.710636 m/s²
```

Problem 4: Enum with Abstract Methods

Problem: Create an enum `Operation` that can perform basic arithmetic operations (ADD, SUBTRACT, MULTIPLY, DIVIDE). Each enum constant should implement an abstract method `apply()` that performs the appropriate calculation.

Solution:

```
java
Copy code
// Enum with abstract methods
enum Operation {
    ADD {
        @Override
        public double apply(double x, double y) {
            return x + y;
        }
    },
    SUBTRACT {
        @Override
        public double apply(double x, double y) {
            return x - y;
        }
    },
    MULTIPLY {
        @Override
        public double apply(double x, double y) {
            return x * y;
        }
    },
    DIVIDE {
        @Override
        public double apply(double x, double y) {
            return x / y;
        }
    };

    public abstract double apply(double x, double y);
}

public class Main {
    public static void main(String[] args) {
        double a = 10, b = 5;

        for (Operation op : Operation.values()) {
            System.out.printf("%f %s %f = %f%n", a, op, b, op.apply(a, b));
        }
    }
}
```

Output:

```
sql
Copy code
10.000000 ADD 5.000000 = 15.000000
10.000000 SUBTRACT 5.000000 = 5.000000
10.000000 MULTIPLY 5.000000 = 50.000000
10.000000 DIVIDE 5.000000 = 2.000000
```

Problem 5: Enum Implementing an Interface

Problem: Create an interface `Movable` with a method `move()`. Then create an enum `Direction` that implements `Movable` for four directions: `NORTH`, `SOUTH`, `EAST`, and `WEST`. Each direction should print a custom movement message.

Solution:

```
java
Copy code
// Interface Movable
interface Movable {
    void move();  // Interface method
}

// Enum implementing the interface
enum Direction implements Movable {
    NORTH {
        @Override
        public void move() {
            System.out.println("Moving North!");
        }
    },
    SOUTH {
        @Override
        public void move() {
            System.out.println("Moving South!");
        }
    },
    EAST {
        @Override
        public void move() {
            System.out.println("Moving East!");
        }
    },
    WEST {
        @Override
        public void move() {
            System.out.println("Moving West!");
        }
    }
}

public class Main {
    public static void main(String[] args) {
        Direction dir = Direction.NORTH;
        dir.move();

        dir = Direction.WEST;
        dir.move();
    }
}
```

Output:

```
Copy code
Moving North!
Moving West!
```